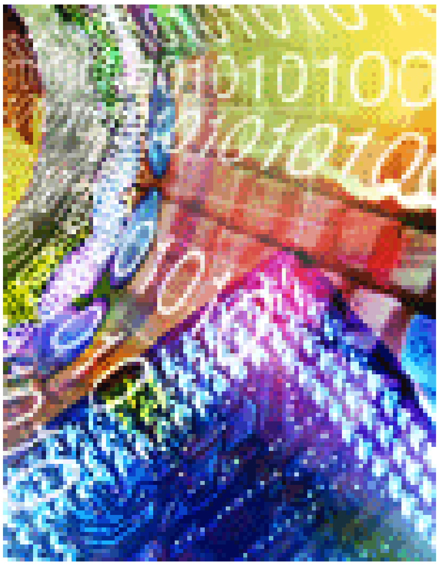
WILEY ENCYCLOPEDIA OF COMPUTER SCIENCE and ENGINEERING

VOLUME 1

Benjamin W. Wah, EDITOR

Wiley Encyclopedia of Computer Science and Engineering



FullTitle of Book: Wiley Encyclopedia Of Computer Science And Engineering
Editor(s): Wah
Publisher: Wiley-interscience
YearPublished: Nov., 2008
ISBN-10: 0471383937
ISBN-13: 978-0471383932
Size& Format: 2362 pages

- Applications
- Computer Vision
- Computing Milieux
- Data
- Foundation and Theory
- Hardware and Architecture
- Image Processing and Visualization
- Intelligent Systems
- IS
- Parallel and Distributed Systems
- Software

ASYNCHRONOUS TRANSFER MODE NETWORKS

Asynchronous transfer mode, or ATM, is a network transfer technique capable of supporting a wide variety of multimedia applications with diverse service and performance requirements. It supports traffic bandwidths ranging from a few kilobits per second (e.g., a text terminal) to several hundred megabits per second (e.g., high-definition video) and traffic types ranging from continuous, fixed-rate traffic (e.g., traditional telephony and file transfer) to highly bursty traffic (e.g., interactive data and video). Because of its support for such a wide range of traffic, ATM was designated by the telecommunication standardization sector of the International Telecommunications Union (ITU-T, formerly CCITT) as the multiplexing and switching technique for Broadband, or high-speed, ISDN (B-ISDN) (1).

ATM is a form of packet-switching technology. That is, ATM networks transmit their information in small, fixed-length packets called cells, each of which contains 48 octets (or bytes) of data and 5 octets of header information. The small, fixed cell size was chosen to facilitate the rapid processing of packets in hardware and to minimize the amount of time required to fill a single packet. This is particularly important for real-time applications such as voice and video that require short packetization delays.

ATM is also connection-oriented. In other words, a virtual circuit must be established before a call can take place, where a call is defined as the transfer of information between two or more endpoints. The establishment of a virtual circuit entails the initiation of a signaling process, during which a route is selected according to the call's quality of service requirements, connection identifiers at each switch on the route are established, and network resources such as bandwidth and buffer space may be reserved for the connection.

Another important characteristic of ATM is that its network functions are typically implemented in hardware. With the introduction of high-speed fiber optic transmission lines, the communication bottleneck has shifted from the communication links to the processing at switching nodes and at terminal equipment. Hardware implementation is necessary to overcome this bottleneck because it minimizes the cell-processing overhead, thereby allowing the network to match link rates on the order of gigabits per second.

Finally, as its name indicates, ATM is asynchronous. Time is slotted into cell-sized intervals, and slots are assigned to calls in an asynchronous, demand-based manner. Because slots are allocated to calls on demand, ATM can easily accommodate traffic whose bit rate fluctuates over time. Moreover, in ATM, no bandwidth is consumed unless information is actually transmitted. ATM also gains bandwidth efficiency by being able to multiplex bursty traffic sources statistically. Because bursty traffic does not require continuous allocation of the bandwidth at its peak rate, statistical multiplexing allows a large number of bursty sources to share the network's bandwidth.

Since its birth in the mid-1980s, ATM has been fortified by a number of robust standards and realized by a significant number of network equipment manufacturers. International standards-making bodies such as the ITU and independent consortia like the ATM Forum have developed a significant body of standards and implementation agreements for ATM (1,4). As networks and network services continue to evolve toward greater speeds and diversities, ATM will undoubtedly continue to proliferate.

ATM STANDARDS

The telecommunication standardization sector of the ITU, the international standards agency commissioned by the United Nations for the global standardization of telecommunications, has developed a number of standards for ATM networks. Other standards bodies and consortia (e.g., the ATM Forum, ANSI) have also contributed to the development of ATM standards. This section presents an overview of the standards, with particular emphasis on the protocol reference model used by ATM (2).

Protocol Reference Model

The B-ISDN protocol reference model, defined in ITU-T recommendation I.321, is shown in Fig. 1(1). The purpose of the protocol reference model is to clarify the functions that ATM networks perform by grouping them into a set of interrelated, function-specific layers and planes. The reference model consists of a user plane, a control plane, and a management plane. Within the user and control planes is a hierarchical set of layers. The user plane defines a set of functions for the transfer of user information between communication endpoints; the control plane defines control functions such as call establishment, call maintenance, and call release; and the management plane defines the operations necessary to control information flow between planes and layers and to maintain accurate and fault-tolerant network operation.

Within the user and control planes, there are three layers: the physical layer, the ATM layer, and the ATM adaptation layer (AAL). Figure 2 summarizes the functions of each layer (1). The physical layer performs primarily bit-level functions, the ATM layer is primarily responsible for the switching of ATM cells, and the ATM adaptation layer is responsible for the conversion of higher-layer protocol frames into ATM cells. The functions that the physical, ATM, and adaptation layers perform are described in more detail next.

Physical Layer

The physical layer is divided into two sublayers: the physical medium sublayer and the transmission convergence sublayer (1).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

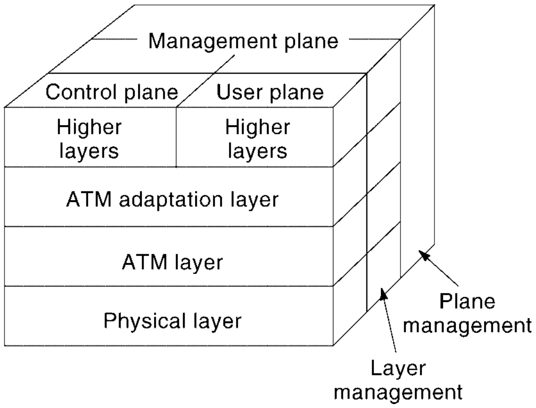


Figure 1. Protocol reference model for ATM.

Physical Medium Sublayer. The physical medium (PM) sublayer performs medium-dependent functions. For example, it provides bit transmission capabilities including bit alignment, line coding and electrical/optical conversion. The PM sublayer is also responsible for bit timing (i.e., the insertion and extraction of bit timing information). The PM sublayer currently supports two types of interface: optical and electrical.

Transmission Convergence Sublayer. Above the physical medium sublayer is the transmission convergence (TC) sublayer, which is primarily responsible for the framing of data transported over the physical medium. The ITU-T recommendation specifies two options for TC sublayer transmission frame structure: cell-based and synchronous digital hierarchy (SDH). In the cell-based case, cells are transported continuously without any regular frame structure. Under SDH, cells are carried in a special frame structure based on the North American SONET (synchronous optical network) protocol (3). Regardless of which transmission frame structure is used, the TC sublayer is responsible for the following four functions: cell rate decoupling, header error control, cell delineation, and transmission frame adaptation. Cell rate decoupling is the insertion of idle cells at the sending side to adapt the ATM cell stream's rate to the rate of the transmission path. Header error control is the insertion of an 8-bit CRC in the ATM cell header to protect the contents of the ATM cell header. Cell delineation is the detection of cell boundaries. Transmission frame adaptation is the encapsulation of departing cells into an appropriate framing structure (either cell-based or SDH-based).

ATM Layer

The ATM layer lies atop the physical layer and specifies the functions required for the switching and flow control of ATM cells (1).

There are two interfaces in an ATM network: the user-network interface (UNI) between the ATM endpoint and the ATM switch, and the network-network interface (NNI) between two ATM switches. Although a 48-octet cell payload is used at both interfaces, the 5-octet cell header differs slightly at these interfaces. Figure 3 shows the cell header structures used at the UNI and NNI (1). At the UNI, the header contains a 4-bit generic flow control (GFC) field, a 24-bit label field containing virtual path identifier (VPI) and virtual channel identifier (VCI) subfields (8 bits for the VPI and 16 bits for the VCI), a 2-bit payload type (PT) field, a 1-bit cell loss priority (CLP) field, and an 8-bit header error check (HEC) field. The cell header for an NNI cell is identical to that for the UNI cell, except that it lacks the GFC field; these four bits are used for an additional 4 VPI bits in the NNI cell header.

The VCI and VPI fields are identifier values for virtual channel (VC) and virtual path (VP), respectively. A virtual channel connects two ATM communication endpoints. A virtual path connects two ATM devices, which can be switches or endpoints, and several virtual channels may be multiplexed onto the same virtual path. The 2-bit PT field identifies whether the cell payload contains data or control information. The CLP bit is used by the user for explicit indication of cell loss priority. If the value of the CLP is 1, then the cell is subject to discarding in case of congestion. The HEC field is an 8-bit CRC that protects the contents of the cell header. The GFC field, which appears only at the UNI, is used to assist the customer premises network in controlling the traffic flow. At the time of writing, the exact procedures for use of this field have not been agreed upon.

Layer management	Higher layer functions	Higher layers	
	Convergence	CS	AAL
	Segmentation and reassembly	SAR	
	Generic flow control Cell header generation/extraction Cell VPI/VCI translation Cell multiplex and demultiplex	ATM	
	Cell rate decoupling Header error control (HEC) Cell delineation Transmission frame adaptation Transmission frame generation/recovery	TC	Physical layer
	Bit timing Physical medium	PM	

Figure 2. Functions of each layer in the protocol reference model.

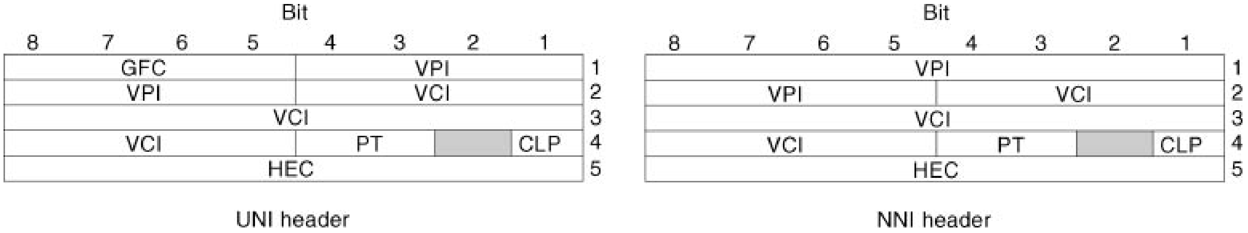


Figure 3. ATM cell header structure.

ATM Layer Functions

The primary function of the ATM layer is VPI/VCI translation. As ATM cells arrive at ATM switches, the VPI and VCI values contained in their headers are examined by the switch to determine which outport port should be used to forward the cell. In the process, the switch translates the cell's original VPI and VCI values into new outgoing VPI and VCI values, which are used in turn by the next ATM switch to send the cell toward its intended destination. The table used to perform this translation is initialized during the establishment of the call.

An ATM switch may either be a VP switch, in which case it translates only the VPI values contained in cell headers, or it may be a VP/VC switch, in which case it translates the incoming VPI/VCI value into an outgoing VPI/VCI pair. Because VPI and VCI values do not represent a unique end-to-end virtual connection, they can be reused at different switches through the network. This is important because the VPI and VCI fields are limited in length and would be quickly exhausted if they were used simply as destination addresses.

The ATM layer supports two types of virtual connections: switched virtual connections (SVC) and permanent, or semipermanent, virtual connections (PVC). Switched virtual connections are established and torn down dynamically by an ATM signaling procedure. That is, they exist only for the duration of a single call. Permanent virtual connections, on the other hand, are established by network administrators and continue to exist as long as the administrator leaves them up, even if they are not used to transmit data.

Other important functions of the ATM layer include cell multiplexing and demultiplexing, cell header creation and extraction, and generic flow control. Cell multiplexing is the merging of cells from several calls onto a single transmission path, cell header creation is the attachment of a 5-octet cell header to each 48-octet block of user payload, and generic flow control is used at the UNI to prevent short-term overload conditions from occurring within the network.

ATM Layer Service Categories

The ATM Forum and ITU-T have defined several distinct service categories at the ATM layer (1,4). The categories defined by the ATM Forum include constant bit rate (CBR), real-time variable bit rate (VBR-rt), non-real-time variable bit rate (VBR-nrt), available bit rate (ABR), and unspecified bit rate (UBR). ITU-T defines four service categories, namely, deterministic bit rate (DBR), statistical bit rate (SBR), available bit rate (ABR), and ATM block transfer (ABT). The first of the three ITU-T service categories correspond roughly to the ATM Forum's CBR, VBR, and ABR classifications, respectively. The fourth service category, ABT, is solely defined by ITU-T and is intended for bursty data applications. The UBR category defined by the ATM Forum is for calls that request no quality of service guarantees at all. Figure 4 lists the ATM service categories, their quality of service (QoS) parameters, and the traffic descriptors required by the service category during call establishment (1,4).

The constant bit rate (or deterministic bit rate) service category provides a very strict QoS guarantee. It is targeted at real-time applications, such as voice and raw video, which mandate severe restrictions on delay, delay variance (jitter), and cell loss rate. The only traffic descriptors required by the CBR service are the peak cell rate and the cell delay variation tolerance. A fixed amount of bandwidth, determined primarily by the call's peak cell rate, is reserved for each CBR connection.

The real-time variable bit rate (or statistical bit rate) service category is intended for real-time bursty applications (e.g., compressed video), which also require strict QoS guarantees. The primary difference between CBR and VBR-rt is in the traffic descriptors they use. The VBR-rt service requires the specification of the sustained (or average) cell rate and burst tolerance (i.e., burst length) in addition to the peak cell rate and the cell delay variation

ITU-T service categories	DBR	SBR	ABT	ABR	
ATM forum service categories	CBR	VBR-rt	VBR-nrt	ABR	UBR
Cell loss rate	Specified				Unspecified
Cell transfer delay	Specified			Unspecified	
Cell delay variation	Specified		Unspecified		
Traffic descriptors (contract)	PCR/CDVT	PCR/CDVT SCR/BT		PCR/CDVT MCR/ACR	PCR/CDVT

PCR = Peak Cell Rate; SCR = Sustained Cell Rate; CDVT = Cell Delay Variation Tolerance; BT = Burst Tolerance; MCR = Minimum Cell Rate; ACR = Allowed Cell Rate.

Figure 4. ATM layer service categories.

tolerance. The ATM Forum also defines a VBR-nrt service category, in which cell delay variance is not guaranteed.

The available bit rate service category is defined to exploit the network's unused bandwidth. It is intended for non-real-time data applications in which the source is amenable to enforced adjustment of its transmission rate. A minimum cell rate is reserved for the ABR connection and therefore guaranteed by the network. When the network has unused bandwidth, ABR sources are allowed to increase their cell rates up to an allowed cell rate (ACR), a value that is periodically updated by the ABR flow control mechanism (to be described in the section entitled "ATM Traffic Control"). The value of ACR always falls between the minimum and the peak cell rate for the connection and is determined by the network.

The ATM Forum defines another service category for non-real-time applications called the unspecified bit rate (UBR) service category. The UBR service is entirely best effort; the call is provided with no QoS guarantees. The ITU-T also defines an additional service category for non-real-time data applications. The ATM block transfer service category is intended for the transmission of short bursts, or blocks, of data. Before transmitting a block, the source requests a reservation of bandwidth from the network. If the ABT service is being used with the immediate transmission option (ABT/IT), the block of data is sent at the same time as the reservation request. If bandwidth is not available for transporting the block, then it is simply discarded, and the source must retransmit it. In the ABT service with delayed transmission (ABT/DT), the source waits for a confirmation from the network that enough bandwidth is available before transmitting the block of data. In both cases, the network temporarily reserves bandwidth according to the peak cell rate for each block. Immediately after transporting the block, the network releases the reserved bandwidth.

ATM Adaptation Layer

The ATM adaptation layer, which resides atop the ATM layer, is responsible for mapping the requirements of higher layer protocols onto the ATM network (1). It operates in ATM devices at the edge of the ATM network and is totally absent in ATM switches. The adaptation layer is divided into two sublayers: the convergence sublayer (CS), which performs error detection and handling, timing, and clock recovery; and the segmentation and reassembly (SAR) sublayer, which performs segmentation of convergence sublayer protocol data units (PDUs) into ATM cell-sized SAR sublayer service data units (SDUs) and vice versa.

In order to support different service requirements, the ITU-T has proposed four AAL-specific service classes. Figure 5 depicts the four service classes defined in recommendation I.362 (1). Note that even though these AAL service classes are similar in many ways to the ATM layer service categories defined in the previous section, they are not the same; each exists at a different layer of the protocol reference model, and each requires a different set of functions.

	Class A	Class B	Class C	Class D
Timing relation between source and destination	Required		Not required	
Bit rate	Constant	Variable		
Connection mode	Connection oriented		Connectionless	

Figure 5. Service classification for AAL.

AAL service class A corresponds to constant bit rate services with a timing relation required between source and destination. The connection mode is connection-oriented. The CBR audio and video belong to this class. Class B corresponds to variable bit rate (VBR) services. This class also requires timing between source and destination, and its mode is connection-oriented. The VBR audio and video are examples of class B services. Class C also corresponds to VBR connection-oriented services, but the timing between source and destination needs not be related. Class C includes connection-oriented data transfer such as X.25, signaling, and future high-speed data services. Class D corresponds to connectionless services. Connectionless data services such as those supported by LANs and MANs are examples of class D services.

Four AAL types (Types 1, 2, 3/4, and 5), each with a unique SAR sublayer and CS sublayer, are defined to support the four service classes. AAL Type 1 supports constant bit rate services (class A), and AAL Type 2 supports variable bit rate services with a timing relation between source and destination (class B). AAL Type 3/4 was originally specified as two different AAL types (Type 3 and Type 4), but because of their inherent similarities, they were eventually merged to support both class C and class D services. AAL Type 5 also supports class C and class D services.

AAL Type 5. Currently, the most widely used adaptation layer is AAL Type 5. AAL Type 5 supports connection-oriented and connectionless services in which there is no timing relation between source and destination (classes C and D). Its functionality was intentionally made simple in order to support high-speed data transfer. AAL Type 5 assumes that the layers above the ATM adaptation layer can perform error recovery, retransmission, and sequence numbering when required, and thus, it does not provide these functions. Therefore, only nonassured operation is provided; lost or corrupted AAL Type 5 packets will not be corrected by retransmission.

Figure 6 depicts the SAR-SDU format for AAL Type 5 (5,6). The SAR sublayer of AAL Type 5 performs segmentation of a CS-PDU into a size suitable for the SAR-SDU payload. Unlike other AAL types, Type 5 devotes the entire 48-octet payload of the ATM cell to the SAR-SDU; there is no overhead. An AAL specific flag (end-of-frame) in the

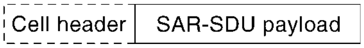


Figure 6. SAR-SDU format for AAL Type 5.

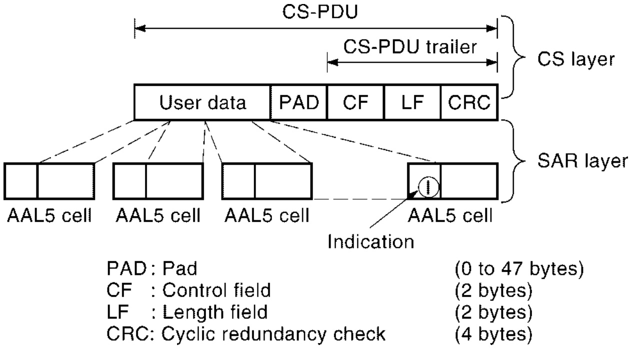

CS-PDU
CS-PDU trailer
CS layer
User data
PAD
CF
LF
CRC
SAR layer
I
AAL5 cell
AAL5 cell
AAL5 cell
AAL5 cell
Indication
PAD : Pad (0 to 47 bytes)
CF : Control field (2 bytes)
LF : Length field (2 bytes)
CRC: Cyclic redundancy check (4 bytes)

Figure 7. CS-PDU format, segmentation and reassembly of AAL Type 5.

ATM PT field of the cell header is set when the last cell of a CS-PDU is sent. The reassembly of CS-PDU frames at the destination is controlled by using this flag.

Figure 7 depicts the CS-PDU format for AAL Type 5 (5,6). It contains the user data payload, along with any necessary padding bits (PAD) and a CS-PDU trailer, which are added by the CS sublayer when it receives the user information from the higher layer. The CS-PDU is padded using 0 to 47 bytes of PAD field to make the length of the CS-PDU an integral multiple of 48 bytes (the size of the SAR-SDU payload). At the receiving end, a reassembled PDU is passed to the CS sublayer from the SAR sublayer, and CRC values are then calculated and compared. If there is no error, the PAD field is removed by using the value of length field (LF) in the CS-PDU trailer, and user data is passed to the higher layer. If an error is detected, the erroneous information is either delivered to the user or discarded according to the user's choice. The use of the CF field is for further study.

AAL Type 1. AAL Type 1 supports constant bit rate services with a fixed timing relation between source and destination users (class A). At the SAR sublayer, it defines a 48-octet service data unit (SDU), which contains 47 octets of user payload, 4 bits for a sequence number, and a 4-bit CRC value to detect errors in the sequence number field. AAL Type 1 performs the following services at the CS sublayer: forward error correction to ensure high quality of audio and video applications, clock recovery by monitoring the buffer filling, explicit time indication by inserting a time stamp in the CS-PDU, and handling of lost and misinserted cells that are recognized by the SAR. At the time of writing, the CS-PDU format has not been decided.

AAL Type 2. AAL Type 2 supports variable bit rate services with a timing relation between source and destination (class B). AAL Type 2 is nearly identical to AAL Type 1, except that it transfers service data units at a variable bit rate, not at a constant bit rate. Furthermore, AAL Type 2 accepts variable length CS-PDUs, and thus, there may exist some SAR-SDUs that are not completely filled with user data. The CS sublayer for AAL Type 2 performs the following functions: forward error correction for audio and video services, clock recovery by inserting a time stamp in the CS-PDU, and handling of lost and misinserted cells. At the time of writing, both the SAR-SDU and CS-PDU formats for AAL Type 2 are still under discussion.

AAL Type 3/4. AAL Type 3/4 mainly supports services that require no timing relation between the source and destination (classes C and D). At the SAR sublayer, it defines a 48-octet service data unit, with 44 octets of user payload; a 2-bit payload type field to indicate whether the SDU is at the beginning, middle, or end of a CS-PDU; a 4-bit cell sequence number; a 10-bit multiplexing identifier that allows several CS-PDUs to be multiplexed over a single VC; a 6-bit cell payload length indicator; and a 10-bit CRC code that covers the payload. The CS-PDU format allows for up to 65535 octets of user payload and contains a header and trailer to delineate the PDU.

The functions that AAL Type 3/4 performs include segmentation and reassembly of variable-length user data and error handling. It supports message mode (for framed data transfer) as well as streaming mode (for streamed data transfer). Because Type 3/4 is mainly intended for data services, it provides a retransmission mechanism if necessary.

ATM Signaling

ATM follows the principle of out-of-band signaling that was established for N-ISDN. In other words, signaling and data channels are separate. The main purposes of signaling are (1) to establish, maintain, and release ATM virtual connections and (2) to negotiate (or renegotiate) the traffic parameters of new (or existing) connections (7). The ATM signaling standards support the creation of point-to-point as well as multicast connections. Typically, certain VCI and VPI values are reserved by ATM networks for signaling messages. If additional signaling VCs are required, they may be established through the process of metasignaling.

ATM TRAFFIC CONTROL

The control of ATM traffic is complicated as a result of ATM's high-link speed and small cell size, the diverse service requirements of ATM applications, and the diverse characteristics of ATM traffic. Furthermore, the configuration and size of the ATM environment, either local or wide area, has a significant impact on the choice of traffic control mechanisms.

The factor that most complicates traffic control in ATM is its high-link speed. Typical ATM link speeds are 155.52 Mbit/s and 622.08 Mbit/s. At these high-link speeds, 53-byte ATM cells must be switched at rates greater than one cell per 2.726 μs or 0.682 μs, respectively. It is apparent that the cell processing required by traffic control must perform at speeds comparable to these cell-switching rates. Thus, traffic control should be simple and efficient, without excessive software processing.

Such high speeds render many traditional traffic control mechanisms inadequate for use in ATM because of their reactive nature. Traditional reactive traffic control mechanisms attempt to control network congestion by responding to it after it occurs and usually involves sending feedback to the source in the form of a choke packet. However, a large bandwidth-delay product (i.e., the amount of traffic that can be sent in a single propagation delay time) renders many reactive control schemes ineffective in high-speed networks. When a node receives feedback, it may have already transmitted a large amount of data. Consider a cross-continental 622 Mbit/s connection with a propagation delay of 20 ms (propagation-bandwidth product of 12.4 Mbit). If a node at one end of the connection experiences congestion and attempts to throttle the source at the other end by sending it a feedback packet, the source will already have transmitted over 12 Mb of information before feedback arrives. This example illustrates the ineffectiveness of traditional reactive traffic control mechanisms in high-speed networks and argues for novel mechanisms that take into account high propagation-bandwidth products.

Not only is traffic control complicated by high speeds, but it also is made more difficult by the diverse QoS requirements of ATM applications. For example, many applications have strict delay requirements and must be delivered within a specified amount of time. Other applications have strict loss requirements and must be delivered reliably without an inordinate amount of loss. Traffic controls must address the diverse requirements of such applications.

Another factor complicating traffic control in ATM networks is the diversity of ATM traffic characteristics. In ATM networks, continuous bit rate traffic is accompanied by bursty traffic. Bursty traffic generates cells at a peak rate for a very short period of time and then immediately becomes less active, generating fewer cells. To improve the efficiency of ATM network utilization, bursty calls should be allocated an amount of bandwidth that is less than their peak rate. This allows the network to multiplex more calls by taking advantage of the small probability that a large number of bursty calls will be simultaneously active. This type of multiplexing is referred to as statistical multiplexing. The problem then becomes one of determining how best to multiplex bursty calls statistically such that the number of cells dropped as a result of excessive burstiness is balanced with the number of bursty traffic streams allowed. Addressing the unique demands of bursty traffic is an important function of ATM traffic control.

For these reasons, many traffic control mechanisms developed for existing networks may not be applicable to ATM networks, and therefore novel forms of traffic control are required (8,9). One such class of novel mechanisms that work well in high-speed networks falls under the heading of preventive control mechanisms. Preventive control attempts to manage congestion by preventing it before it occurs. Preventive traffic control is targeted primarily at real-time traffic. Another class of traffic control mechanisms has been targeted toward non-real-time data traffic and relies on novel reactive feedback mechanisms.

Preventive Traffic Control

Preventive control for ATM has two major components: call admission control and usage parameter control (8). Admission control determines whether to accept or reject a new call at the time of call set-up. This decision is based on the traffic characteristics of the new call and the current network load. Usage parameter control enforces the traffic parameters of the call after it has been accepted into the network. This enforcement is necessary to ensure that the call's actual traffic flow conforms with that reported during call admission.

Before describing call admission and usage parameter control in more detail, it is important to first discuss the nature of multimedia traffic. Most ATM traffic belongs to one of two general classes of traffic: continuous traffic and bursty traffic. Sources of continuous traffic (e.g., constant bit rate video, voice without silence detection) are easily handled because their resource utilization is predictable and they can be deterministically multiplexed. However, bursty traffic (e.g., voice with silence detection, variable bit rate video) is characterized by its unpredictability, and this kind of traffic complicates preventive traffic control.

Burstiness is a parameter describing how densely or sparsely cell arrivals occur. There are a number of ways to express traffic burstiness, the most typical of which are the ratio of peak bit rate to average bit rate and the average burst length. Several other measures of burstiness have also been proposed (8). It is well known that burstiness plays a critical role in determining network performance, and thus, it is critical for traffic control mechanisms to reduce the negative impact of bursty traffic.

Call Admission Control. Call admission control is the process by which the network decides whether to accept or reject a new call. When a new call requests access to the network, it provides a set of traffic descriptors (e.g., peak rate, average rate, average burst length) and a set of quality of service requirements (e.g., acceptable cell loss rate, acceptable cell delay variance, acceptable delay). The network then determines, through signaling, if it has enough resources (e.g., bandwidth, buffer space) to support the new call's requirements. If it does, the call is immediately accepted and allowed to transmit data into the network. Otherwise it is rejected. Call admission control prevents network congestion by limiting the number of active connections in the network to a level where the network resources are adequate to maintain quality of service guarantees.

One of the most common ways for an ATM network to make a call admission decision is to use the call's traffic descriptors and quality of service requirements to predict the "equivalent bandwidth" required by the call. The equivalent bandwidth determines how many resources need to be reserved by the network to support the new call at its requested quality of service. For continuous, constant bit rate calls, determining the equivalent bandwidth is simple. It is merely equal to the peak bit rate of the call. For bursty connections, however, the process of determining the equivalent bandwidth should take into account such factors as a call's burstiness ratio (the ratio of peak bit rate to average bit rate), burst length, and burst interarrival time. The equivalent bandwidth for bursty connections must be chosen carefully to ameliorate congestion and cell loss while maximizing the number of connections that can be statistically multiplexed.

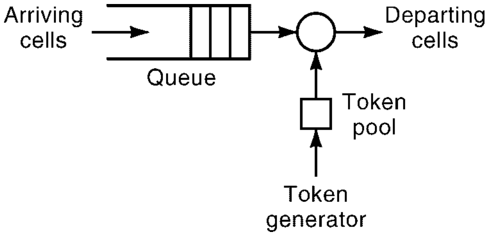


Figure 8. Leaky bucket mechanism.

Usage Parameter Control. Call admission control is responsible for admitting or rejecting new calls. However, call admission by itself is ineffective if the call does not transmit data according to the traffic parameters it provided. Users may intentionally or accidentally exceed the traffic parameters declared during call admission, thereby overloading the network. In order to prevent the network users from violating their traffic contracts and causing the network to enter a congested state, each call's traffic flow is monitored and, if necessary, restricted. This is the purpose of usage parameter control. (Usage parameter control is also commonly referred to as policing, bandwidth enforcement, or flow enforcement.)

To monitor a call's traffic efficiently, the usage parameter control function must be located as close as possible to the actual source of the traffic. An ideal usage parameter control mechanism should have the ability to detect parameter-violating cells, appear transparent to connections respecting their admission parameters, and rapidly respond to parameter violations. It should also be simple, fast, and cost effective to implement in hardware. To meet these requirements, several mechanisms have been proposed and implemented (8).

The leaky bucket mechanism (originally proposed in Ref. 10) is a typical usage parameter control mechanism used for ATM networks. It can simultaneously enforce the average bandwidth and the burst factor of a traffic source. One possible implementation of the leaky bucket mechanism is to control the traffic flow by means of tokens. A conceptual model for the leaky bucket mechanism is illustrated in Fig. 5.

In Fig. 8, an arriving cell first enters a queue. If the queue is full, cells are simply discarded. To enter the network, a cell must first obtain a token from the token pool; if there is no token, a cell must wait in the queue until a new token is generated. Tokens are generated at a fixed rate corresponding to the average bit rate declared during call admission. If the number of tokens in the token pool exceeds some predefined threshold value, token generation stops. This threshold value corresponds to the burstiness of the transmission declared at call admission time; for larger threshold values, a greater degree of burstiness is allowed. This method enforces the average input rate while allowing for a certain degree of burstiness.

One disadvantage of the leaky bucket mechanism is that the bandwidth enforcement introduced by the token pool is in effect even when the network load is light and there is no need for enforcement. Another disadvantage of the leaky bucket mechanism is that it may mistake nonviolating cells for violating cells. When traffic is bursty, a large number of cells may be generated in a short period of time, while conforming to the traffic parameters claimed at the time of call admission. In such situations, none of these cells should be considered violating cells. Yet in actual practice, leaky bucket may erroneously identify such cells as violations of admission parameters. A virtual leaky bucket mechanism (also referred to as a marking method) alleviates these disadvantages (11). In this mechanism, violating cells, rather than being discarded or buffered, are permitted to enter the network at a lower priority (CLP = 1). These violating cells are discarded only when they arrive at a congested node. If there are no congested nodes along the routes to their destinations, the violating cells are transmitted without being discarded. The virtual leaky bucket mechanism can easily be implemented using the leaky bucket method described earlier. When the queue length exceeds a threshold, cells are marked as "droppable" instead of being discarded. The virtual leaky bucket method not only allows the user to take advantage of a light network load but also allows a larger margin of error in determining the token pool parameters.

Reactive Traffic Control

Preventive control is appropriate for most types of ATM traffic. However, there are cases where reactive control is beneficial. For instance, reactive control is useful for service classes like ABR, which allow sources to use bandwidth not being used by calls in other service classes. Such a service would be impossible with preventive control because the amount of unused bandwidth in the network changes dynamically, and the sources can only be made aware of the amount through reactive feedback.

There are two major classes of reactive traffic control mechanisms: rate-based and credit-based (12,13). Most rate-based traffic control mechanisms establish a closed feedback loop in which the source periodically transmits special control cells, called resource management cells, to the destination (or destinations). The destination closes the feedback loop by returning the resource management cells to the source. As the feedback cells traverse the network, the intermediate switches examine their current congestion state and mark the feedback cells accordingly. When the source receives a returning feedback cell, it adjusts its rate, either by decreasing it in the case of network congestion or increasing it in the case of network underuse. An example of a rate-based ABR algorithm is the Enhanced Proportional Rate Control Algorithm (EPRCA), which was proposed, developed, and tested through the course of ATM Forum activities (12).

Credit-based mechanisms use link-by-link traffic control to eliminate loss and optimize use. Intermediate switches exchange resource management cells that contain "credits," which reflect the amount of buffer space available at the next downstream switch. A source cannot transmit a new data cell unless it has received at least one credit from its downstream neighbor. An example of a credit-based mechanism is the Quantum Flow Control (QFC) algorithm, developed by a consortium of reseachers and ATM equipment manufacturers (13).

HARDWARE SWITCH ARCHITECTURES FOR ATM NETWORKS

In ATM networks, information is segmented into fixed-length cells, and cells are asynchronously transmitted through the network. To match the transmission speed of the network links and to minimize the protocol processing overhead, ATM performs the switching of cells in hardware-switching fabrics, unlike traditional packet switching networks, where switching is largely performed in software.

A large number of designs has been proposed and implemented for ATM switches (14). Although many differences exist, ATM switch architectures can be broadly classified into two categories: asynchronous time division (ATD) and space-division architectures.

Asynchronous Time Division Switches

The ATD, or single path, architectures provide a single, multiplexed path through the ATM switch for all cells. Typically a bus or ring is used. Figure 9 shows the basic structure of the ATM switch proposed in (15). In Fig. 6, four input ports are connected to four output ports by a time-division multiplexing (TDM) bus. Each input port is allocated a fixed time slot on the TDM bus, and the bus is designated to operate at a speed equal to the sum of the incoming bit rates at all input ports. The TDM slot sizes are fixed and equal in length to the time it takes to transmit one ATM cell. Thus, during one TDM cycle, the four input ports can transfer four ATM cells to four output ports.

In ATD switches, the maximum throughput is determined by a single, multiplexed path. Switches with N input ports and N output ports must run at a rate N times faster than the transmission links. Therefore, the total throughput of ATD ATM switches is bounded by the current capabilities of device logic technology. Commercial examples of ATD switches are the Fore Systems ASX switch and Digital's VNswitch.

Space-Division Switches

To eliminate the single-path limitation and increase total throughput, space-division ATM switches implement multiple paths through switching fabrics. Most space-division switches are based on multistage interconnection networks, where small switching elements (usually 2 × 2 cross-point switches) are organized into stages and provide multiple paths through a switching fabric. Rather than being multiplexed onto a single path, ATM cells are space-switched through the fabric. Three typical types of space-division switches are described next.

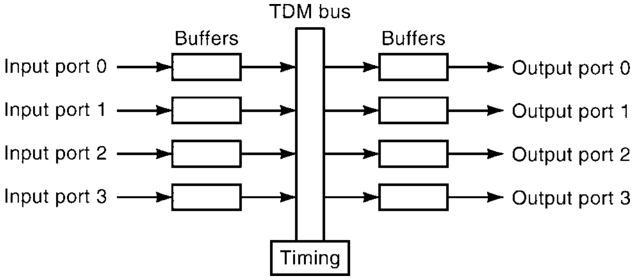


Figure 9. A 4 × 4 asynchronous time division switch.

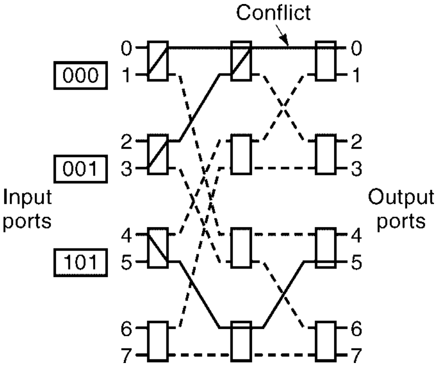


Figure 10. A 8 × 8 Banyan switch with binary switching elements.

Banyan Switches. Banyan switches are examples of space-division switches. An $N \times N$ Banyan switch is constructed by arranging a number of binary switching elements into several stages ($\log_2 N$ stages). Figure 10 depicts an 8 × 8 self-routing Banyan switch (14). The switch fabric is composed of twelve 2 × 2 switching elements assembled into three stages. From any of the eight input ports, it is possible to reach all the eight output ports. One desirable characteristic of the Banyan switch is that it is self-routing. Because each cross-point switch has only two output lines, only one bit is required to specify the correct output path. Very simply, if the desired output addresses of a ATM cell is stored in the cell header in binary code, routing decisions for the cell can be made at each cross-point switch by examining the appropriate bit of the destination address.

Although the Banyan switch is simple and possesses attractive features such as modularity, which makes it suitable for VLSI implementation, it also has some disadvantages. One of its disadvantages is that it is internally blocking. In other words, cells destined for different output ports may contend for a common link within the switch. This results in blocking all cells that wish to use that link, except for one. Hence, the Banyan switch is referred to as a blocking switch. In Fig. 10, three cells are shown arriving on input ports 1, 3, and 4 with destination port addresses of 0, 1, and 5, respectively. The cell destined for output port 0 and the cell destined for output port 1 end up contending for the link between the second and third stages. As a result, only one of them (the cell from input port 1 in this example) actually reaches its destination (output port 0), while the other is blocked.

Batcher–Banyan Switches. Another example of space-division switches is the Batcher–Banyan switch (14). (See Fig. 11.) It consists of two multistage interconnection networks: a Banyan self-routing network and a Batcher sorting network. In the Batcher–Banyan switch, the incoming cells first enter the sorting network, which takes the cells and sorts them into ascending order according to their output addresses. Cells then enter the Banyan network, which routes the cells to their correct output ports.

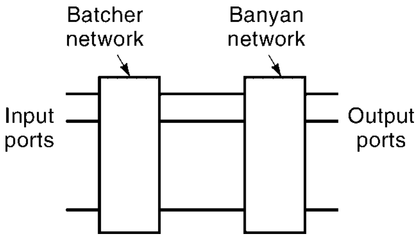


Figure 11. Batcher–Banyan switch.

As shown earlier, the Banyan switch is internally blocking. However, the Banyan switch possesses an interesting feature. Namely, internal blocking can be avoided if the cells arriving at the Banyan switch's input ports are sorted in ascending order by their destination addresses. The Batcher–Banyan switch takes advantage of this fact and uses the Batcher soring network to sort the cells, thereby making the Batcher–Banyan switch internally nonblocking. The Starlite switch, designed by Bellcore, is based on the Batcher–Banyan architecture (16).

Crossbar Switches. The crossbar switch interconnects N inputs and N outputs into a fully meshed topology; that is, there are N^2 cross points within the switch (14). (See Fig. 12.) Because it is always possible to establish a connection between any arbitrary input and output pair, internal blocking is impossible in a crossbar switch.

The architecture of the crossbar switch has some advantages. First, it uses a simple two-state cross-point switch (open and connected state), which is easy to implement. Second, the modularity of the switch design allows simple expansion. One can build a larger switch by simply adding more cross-point switches. Lastly, compared to Banyan-based switches, the crossbar switch design results in low transfer latency, because it has the smallest number of connecting points between input and output ports. One disadvantage to this design, however, is the fact that it uses the maximum number of cross points (cross-point switches) needed to implement an $N \times N$ switch.

The knockout switch by AT&T Bell Labs is a nonblocking switch based on the crossbar design (17,18). It has N inputs and N outputs and consists of a crossbar-based switch with a bus interface module at each output (Fig. 12).

Nonblocking Buffered Switches

Although some switches such as Batcher–Banyan and crossbar switches are internally nonblocking, two or more cells may still contend for the same output port in a nonblocking switch, resulting in the dropping of all but one cell. In order to prevent such loss, the buffering of cells by the switch is necessary. Figure 13 illustrates that buffers may be placed (1) in the inputs to the switch, (2) in the outputs to the switch, or (3) within the switching fabric itself, as a shared buffer (14). Some switches put buffers in both the input and output ports of a switch.

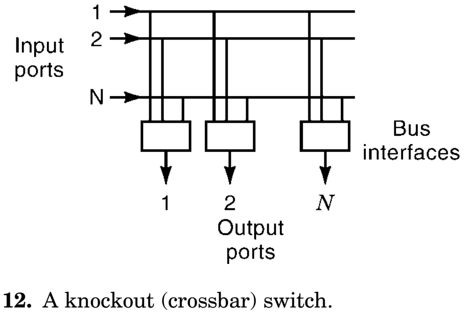


Figure 12. A knockout (crossbar) switch.

The first approach to eliminating output contention is to place buffers in the output ports of the switch (14). In the worst case, cells arriving simultaneously at all input ports can be destined for a single output port. To ensure that no cells are lost in this case, the cell transfer must be performed at N times the speed of the input links, and the switch must be able to write N cells into the output buffer during one cell transmission time. Examples of output buffered switches include the knockout switch by AT&T Bell Labs, the Siemens & Newbridge MainStreetXpress switches, the ATML's VIRATA switch, and Bay Networks' Lattis switch.

The second approach to buffering in ATM switches is to place the buffers in the input ports of the switch (14). Each input has a dedicated buffer, and cells that would otherwise be blocked at the output ports of the switch are stored in input buffers. Commercial examples of switches with input buffers as well as output buffers are IBM's 8285 Nways switches, and Cisco's Lightstream 2020 switches.

A third approach is to use a shared buffer within the switch fabric. In a shared buffer switch, there is no buffer at the input or output ports (14). Arriving cells are immediately injected into the switch. When output contention happens, the winning cell goes through the switch, while the losing cells are stored for later transmission in a shared buffer common to all of the input ports. Cells just arriving at the switch join buffered cells in competition for available outputs. Because more cells are available to select from, it is possible that fewer output ports will be idle when using the shared buffer scheme. Thus, the shared buffer switch can achieve high throughput. However, one drawback is that cells may be delivered out of sequence because cells that arrived more recently may win over buffered cells during contention (19). Another drawback is the increase in the number of input and output ports internal to the switch. The Starlite switch with trap by Bellcore is an example of the shared buffer switch architecture (16). Other examples of shared buffer switches include Cisco's Lightstream 1010 switches, IBM's Prizma switches, Hitachi's 5001 switches, and Lucent's ATM cell switches.

CONTINUING RESEARCH IN ATM NETWORKS

ATM is continuously evolving, and its attractive ability to support broadband integrated services with strict quality of service guarantees has motivated the integration of ATM and existing widely deployed networks. Recent additions to ATM research and technology include, but are not limited to, seamless integration with existing LANs [e.g., LAN emulation (20)], efficient support for traditional Internet IP networking [e.g., IP over ATM (21), IP switching (22)], and further development of flow and congestion control

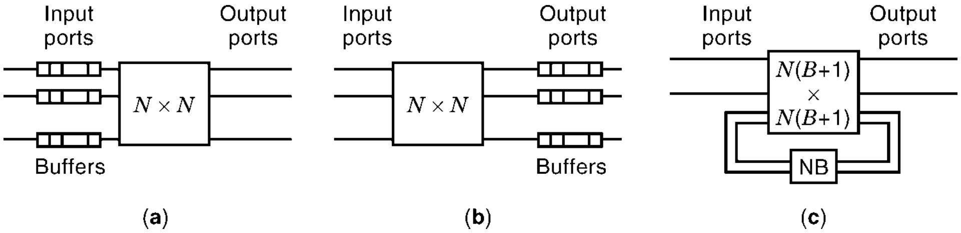


Figure 13. Nonblocking buffered switches.

algorithms to support existing data services [e.g., ABR flow control (12)]. Research on topics related to ATM networks is currently proceeding and will undoubtedly continue to proceed as the technology matures.

BIBLIOGRAPHY

1. *CCITT Recommendation I-Series.* Geneva: International Telephone and Telegraph Consultative Committee.
2. J. B. Kim, T. Suda and M. Yoshimura, International standardization of B-ISDN, *Comput. Networks ISDN Syst.*, **27**: 1994.
3. *CCITT Recommendation G-Series.* Geneva: International Telephone and Telegraph Consultative Committee.
4. *ATM Forum Technical Specifications [Online].* Available www: www.atmforum.com
5. *Report of ANSI T1S1.5/91-292*, Simple and Efficient Adaptation Layer (SEAL), August 1991.
6. *Report of ANSI T1S1.5/91-449*, AAL5—A New High Speed Data Transfer, November 1991.
7. *CCITT Recommendation Q-Series.* Geneva: International Telephone and Telegraph Consultative Committee.
8. J. Bae and T. Suda, Survey of traffic control schemes and protocols in ATM networks, *Proc. IEEE*, **79**: 1991.
9. B. J. Vickers et al., Congestion control and resource management in diverse ATM environments, *IECEJ J.*, **J76-B-I** (11): 1993.
10. J. S. Turner, New directions in communications (or which way to the information age?), *IEEE Commun. Mag.*, **25** (10): 1986.
11. G. Gallassi, G. Rigolio, and L. Fratta, ATM: Bandwidth assignment and bandwidth enforcement policies. *Proc. GLOBECOM'89.*
12. ATM Forum, ATM Forum Traffic management specification version 4.0, *af-tm-0056.000*, April 1996, Mountain View, CA: ATM Forum.
13. Quantum Flow Control version 2.0, *Flow Control Consortium, FCC-SPEC-95-1*, [Online], July 1995. http://www.qfc.org
14. Y. Oie et al., Survey of switching techniques in high-speed networks and their performance, *Int. J. Satellite Commun.*, **9**: 285–303, 1991.
15. M. De Prycker and M. De Somer, Performance of a service independent switching network with distributed control, *IEEE J. Select. Areas Commun.*, **5**: 1293–1301, 1987.
16. A. Huang and S. Knauer, Starlite: A wideband digital switch. *Proc. IEEE GLOBECOM'84*, 1984.
17. K. Y. Eng, A photonic knockout switch for high-speed packet networks, *IEEE J. Select. Areas Commun.*, **6**: 1107–1116, 1988.
18. Y. S. Yeh, M. G. Hluchyj, and A. S. Acampora, The knockout switch: A simple, modular architecture for high-performance packet switching, *IEEE J. Select. Areas Commun.*, **5**: 1274–1283, 1987.
19. J. Y. Hui and E. Arthurs, A broadband packet switch for integrated transport, *IEEE J. Select. Areas Commun.*, **5**: 1264–1273, 1987.
20. ATM Forum, LAN emulation over ATM version 1.0. *AF-LANE-0021*, 1995, Mountain View, CA: ATM Forum.
21. IETF, IP over ATM: A framework document, *RFC-1932*, 1996.
22. Ipsilon Corporation, IP switching: The intelligence of routing, *The Performance of Switching* [Online]. Available www.ipsiolon.com

Tatsuya Suda
University of California, Irvine
Irvine, California

A

AIRCRAFT COMPUTERS

AIRCRAFT ANALOG COMPUTERS

Early aircraft computers were used to take continuous streams of inputs to provide flight assistance. Examples of aircraft analog inputs are fuel gauge readings, throttle settings, and altitude indicators. Landau (1) defines an analog computer as a computer for processing data represented by a continuous physical variable, such as electric current. Analog computers monitor these inputs and implement a predetermined service when some set of inputs calls for a flight control adjustment. For example, when fuel levels are below a certain point, the analog computer would read a low fuel level in the aircraft's main fuel tanks and would initiate the pumping of fuel from reserve tanks or the balancing of fuel between wing fuel tanks. Some of the first applications of analog computers to aircraft applications were for automatic pilot applications, where these analog machines took flight control inputs to hold altitude and course. The analog computers use operational amplifiers to build the functionality of summers, adders, subtracters, and integrators on the electric signals.

Aircraft Digital Computers

As the technologies used to build digital computers evolved, digital computers became smaller, lighter, and less power-hungry, and produced less heat. This improvement made them increasingly acceptable for aircraft applications. Digital computers are synonymous with stored-program computers. A stored-program computer has the flexibility of being able to accomplish multiple different tasks simply by changing the stored program. Analog computers are hard-wired to perform one and only one function. Analog computers' data, as defined earlier, are continuous physical variables. Analog computers may be able to recognize and process numerous physical variables, but each variable has its unique characteristics that must be handled during processing by the analog computer. The range of output values for the analog computer is bounded as a given voltage range; if they exceed this range, they saturate. Digital computers are not constrained by physical variables. All the inputs and outputs of the digital computer are in a digital representation. The processing logic and algorithms performed by the computer work in a single representation of the cumulative data. It is not uncommon to see aircraft applications that have analog-to-digital and digital-to-analog signal converters. This method is more efficient than having the conversions done within the computers. Analog signals to the digital computer are converted to digital format, where they are quickly processed digitally and returned to the analog device through a digital-to-analog converter as an analog output for that device to act upon. These digital computers are smaller, more powerful, and easier to integrate into multiple areas of aircraft applications.

Landau (1) defines a digital computer as a computer for processing data represented by discrete, localized physical signals, such as the presence or absence of an electric current. These signals are represented as a series of bits with word lengths of 16, 32, and 64 bits. See ***micro-computers*** for further discussion.

Wakerly (2) shows number systems and codes used to process binary digits in digital computers. Some important number systems used in digital computers are binary, octal, and hexadecimal numbers. He also shows conversion between these and base-10 numbers, as well as simple mathematical operations such as addition, subtraction, division, and multiplication. The American Standard Code for Information Interchange (ASCII) of the American National Standard Institute (ANSI) is also presented, which is Standard No. X3.4-1968 for numerals, symbols, characters, and control codes used in automatic data processing machines, including computers. Figure 1 shows a typical aircraft central computer.

Microcomputers

The improvements in size, speed, and cost through computer technologies continually implement new computer consumer products. Many of these products were unavailable to the average consumer until recently. These same breakthroughs provide enormous functional improvements in aircraft computing. Landau (1) defines microcomputers as very small, relatively inexpensive computers whose central processing unit (CPU) is a microprocessor. A microprocessor (also called MPU or central processing unit) communicates with other devices in the system through wires (or fiber optics) called lines. Each device has a unique address, represented in binary format, which the MPU recognizes. The number of lines is also the address size in bits. Early MPU machines had 8-bit addresses. Machines of 1970 to 1980 typically had 16-bit addresses; modern MPU machines have 256 bits.

Common terminology for an MPU is *random access memory* (RAM), *read only memory* (ROM), *input-output*, clock, and *interrupts*. RAM is volatile storage. It holds both data and instructions for the MPU. ROM may hold both instructions and data. The key point of ROM is that it is nonvolatile. Typically, in an MPU, there is no operational difference between RAM and ROM other than its volatility. Input-output is how data are transferred to and from the microcomputer. Output may be from the MPU, ROM, or RAM. Input may be from the MPU or the RAM. The clock of an MPU synchronizes the execution of the MPU instructions. Interrupts are inputs to the MPU that cause it to (temporarily) suspend one activity in order to perform a more important activity.

An important family of MPUs that greatly improved the performance of aircraft computers is the Motorola M6800 family of microcomputers. This family offered a series of

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

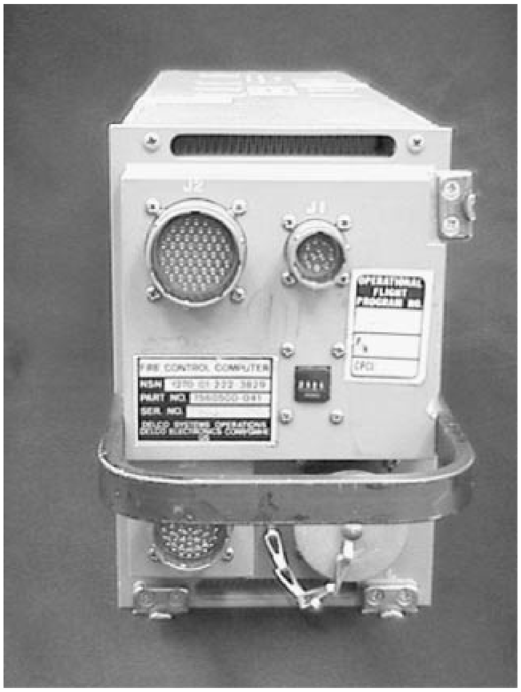


Figure 1. Typical aircraft central computer.

improvements in memory size, clock speeds, functionality, and overall computer performance.

Personal Computers

Landau (1) defines personal computers as electronic machines that can be owned and operated by individuals for home and business applications such as word processing, games, finance, and electronic communications. Hamacher et al. (3) explain that rapidly advancing very large-scale integrated circuit (VLSI) technology has resulted in dramatic reductions in the cost of computer hardware. The greatest impact has been in the area of small computing machines, where it has led to an expanding market for personal computers.

The idea of a personally owned computer is fairly new. The computational power available in handheld toys today was only available through large, costly computers in the late 1950s and early 1960s. Vendors such as Atari, Commodore, and Compaq made simple computer games household items. Performance improvements in memory, throughput, and processing power by companies such as IBM, Intel, and Apple made facilities such as spreadsheets for home budgets, automated tax programs, word processing, and three-dimensional virtual games common household items. The introduction of Microsoft's Disk Operating System (DOS) and Windows has also added to the acceptance of the personal computers through access to software applications. Improvements in computer technology offer continual improvements, often multiple times a year. The durability and portability of these computers is beginning to allow them to replace specialized aircraft computers that had strict weight, size, power, and functionality requirements.

AVIONICS

In the early years of aircraft flight, technological innovation was directed at improving flight performance through rapid design improvements in aircraft propulsion and airframes. Secondary development energies went to areas such as navigation, communication, munitions delivery, and target detection. The secondary functionality of aircraft evolved into the field of avionics. Avionics now provides greater overall performance and accounts for a greater share of aircraft lifecycle costs than either propulsion or airframe components.

Landau (1) defines avionics [avi(ation) + (electr)onics] as the branch of electronics dealing with the development and use of electronic equipment in aviation and astronautics. The field of avionics has evolved rapidly as electronics has improved all aspects of aircraft flight. New advances in these disciplines require avionics to control flight stability, which was traditionally the pilot's role.

Aircraft Antennas

An important aspect of avionics is receiving and transmitting electromagnetic signals. Antennas are devices for transmitting and receiving radio-frequency (RF) energy from other aircraft, space applications, or ground applications. Perry and Geppert (4) illustrate the aircraft electromagnetic spectrum, influenced by the placement and usage of numerous antennas on a commercial aircraft. Golden (5) illustrates simple antenna characteristics of dipole, horn, cavity-backed spiral, parabola, parabolic cylinder, and Cassegrain antennas.

Radiation pattern characteristics include elevation and azimuth. The typical antenna specifications are polarization, beam width, gain, bandwidth, and frequency limit.

Computers are becoming increasingly important for the new generation of antennas, which include phased-array antennas and smart-skin antennas. For phased-array antennas, computers are needed to configure the array elements to provide direction and range requirements between the radar pulses. Smart-skin antennas comprise the entire aircraft's exterior fuselage surface and wings. Computers are used to configure the portion of the aircraft surface needed for some sensor function. The computer also handles sensor function prioritization and deinterleaving of conflicting transmissions.

Aircraft Sensors

Sensors, the eyes and ears of an aircraft, are electronic devices for measuring external and internal environmental conditions. Sensors on aircraft include devices for sending and receiving RF energy. These types of sensors include radar, radio, and warning receivers. Another group of sensors are the infrared (IR) sensors, which include lasers and heat-sensitive sensors. Sensors are also used to measure direct analog inputs; altimeters and airspeed indicators are examples. Many of the sensors used on aircraft have their own built-in computers for serving their own functional requirements such as data preprocessing, filtering, and analysis. Sensors can also be part of a computer interface suite that provides key aircraft computers with the direct environmental inputs they need to function.

Aircraft Radar

Radar (radio detection and ranging) is a sensor that transmits RF energy to detect air and ground objects and determines parameters such as the range, velocity, and direction of these objects. The aircraft radar serves as its primary sensor. Several services are provided by modern aircraft radar, including tracking, mapping, scanning, and identification. Golden (5) states that radar is tasked either to detect the presence of a target or to determine its location. Depending on the function emphasized, a radar system might be classified as a search or tracking radar.

Stimson (6) describes the decibel (named after Alexander Graham Bell) as one of the most widely used terms in the design and description of radar systems. The decibel (dB) is a logarithmic unit originally devised to express power ratios, but also used to express a variety of other ratios. The power ratio in dB is expressed as 10 $\log_{10} P_2/P_1$, where P_2 and P_1 are the power levels being compared. Expressed in terms of voltage, the gain is $(V_2/V_1)^2$ dB provided the input voltage V_1 and output voltage V_2 are across equal resistances.

Stimson (6) also explains the concept of the pulse repetition frequency (PRF), which is the rate at which a radar system's pulses are transmitted: the number of pulses per second. The interpulse period T of a radar is given by $T = 1/\mathrm{PRF}$. For a PRF of 100 Hz, the interpulse period would be 0.01 s.

The Doppler Effect, as described by Stimson (6), is a shift in the frequency of a radiated wave, reflected or received by an object in motion. By sensing Doppler frequencies, radar not only can measure range rates, but can also separate target echoes from clutter, or can produce high-resolution ground maps. Computers are required by an aircraft radar to make numerous and timely calculations with the received radar data, and to configure the radar to meet the aircrew's needs.

Aircraft Data Fusion

Data fusion is a method for integrating data from multiple sources in order to give a comprehensive solution to a problem (multiple inputs, single output). For aircraft computers, data fusion specifically deals with integrating data from multiple sensors such as radar and infrared sensors. For example, in ground mapping, radar gives good surface parameters, whereas the infrared sensor provides the height and size of items in the surface area being investigated. The aircraft computer takes the best inputs from each sensor, provides a common reference frame to integrate these inputs, and returns a more comprehensive solution than either single sensor could have given.

Data fusion is becoming increasingly important as aircrafts' evolving functionality depends on off-board data (information) sources. New information such as weather, flight path re-routing, potential threats, target assignment, and enroute fuel availability are communicated to the aircraft from its command and control environment. The aircraft computer can now expand its own solution with these off-board sources.

Aircraft Navigation

Navigation is the science of determining present location, desired location, obstacles between these locations, and best courses to take to reach these locations. An interesting pioneer of aircraft navigation was James Harold Doolittle (1886–1993). Best known for his aircraft-carrier-based bomber raid on Tokyo in World War II, General Doolittle received his Master's and Doctor of Science degrees in aeronautics from Massachusetts Institute of Technology, where he developed instrumental *blind flying* in 1929. He made navigation history by taking off, flying a set course, and landing without seeing the ground. For a modern aircraft, with continuous changes in altitude, airspeed, and course, navigation is a challenge. Aircraft computers help meet this challenge by processing the multiple inputs and suggesting aircrew actions to maintain course, avoid collision and weather, conserve fuel, and suggest alternative flight solutions.

An important development in aircraft navigation is the Kalman filter. Welch and Bishop (7) state that in 1960, R.E. Kalman published his famous paper describing a recursive solution to the discrete-data linear filtering problem. Since that time, due in large part to advances in digital computing, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation. The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) implementation of the least-squares method. The filter is very powerful in several aspects: It supports estimation of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.

The global positioning system (GPS) is a satellite reference system that uses multiple satellite inputs to determine location. Many modern systems, including aircraft, are equipped with GPS receivers, which allow the system access to the network of GPS satellites and the GPS services. Depending on the quality and privileges of the GPS receiver, the system can have an instantaneous input of its current location, course, and speed within centimeters of accuracy. GPS receivers, another type of aircraft computer, can also be programmed to inform aircrews of services related to their flight plan.

Before the GPS receiver, the inertial navigation systems (INS) were the primary navigation system on aircraft. Fink and Christiansen (8) describe inertial navigation as the most widely used "self-contained" technology. In the case of an aircraft, the INS is contained within the aircraft, and is not dependent on outside inputs. Accelerometers constantly sense the vehicle's movements and convert them, by double integration, into distance traveled. To reduce errors caused by vehicle attitude, the accelerometers are mounted on a gyroscopically controlled stable platform.

Aircraft Communications

Communication technologies on aircraft are predominately radio communication. This technology allows aircrews to communicate with ground controllers and other aircraft. Aircraft computers help establish, secure, and amplify these important communication channels.

These communication technologies are becoming increasingly important as aircraft become interoperable. As the dependency of aircraft on interoperability increases, the requirements to provide better, more reliable, secure point-to-point aircraft communication also increases. The aircraft computer plays a significant role in meeting this challenge by formatting and regulating this increased flow of information.

Aircraft Displays

Displays are visual monitors in aircraft that present desired data to aircrews and passengers. Adam and Gibson (9) illustrate F-15E displays used in the Gulf War. These illustrations show heads-up displays (HUDs), vertical situation displays, radar warning receivers, and low-altitude navigation and targeting system (Lantirn) displays typical of modern fighter aircraft. Sweet (10) illustrates the displays of a Boeing 777, showing the digital bus interface to the flight-deck panels and an optical-fiber data distribution interface that meets industry standards.

Aircraft Instrumentation

Instrumentation of an aircraft means installing data collection and analysis equipment to collect information about the aircraft's performance. Instrumentation equipment includes various recorders for collecting real-time flight parameters such as position and airspeed. Instruments also capture flight control inputs, environmental parameters, and any anomalies encountered in flight test or in routine flight. One method of overcoming this limitation is to link flight instruments to ground recording systems, which are not limited in their data recording capacities. A key issue here is the bandwidth between the aircraft being tested and its ground (recording) station. This bandwidth is limited and places important limitations on what can be recorded. This type of data link is also limited to the range of the link, limiting the aircraft's range and altitude during this type of flight test. Aircraft computers are used both in processing the data as they are being collected on the aircraft and in analyzing the data after they have been collected.

Aircraft Embedded Information Systems

Embedded information system is the latest terminology for an embedded computer system. The software of the embedded computer system is now referred to as embedded information. The purpose of the aircraft embedded information system is to process flight inputs (such as sensor and flight control) into usable flight information for further flight system or aircrew use. The embedded information system is a good example of the merging of two camps of computer science applications. The first, and larger, camp is the management of information systems (MIS). The MIS dealt primarily with large volumes of information, with primary applications in business and banking. The timing requirements of processing these large information records are measured in minutes or hours. The second camp is the real-time embedded computer camp, which was concerned with processing a much smaller set of data, but in a very timely fashion. The real-time camp's timing requirement is in microseconds. These camps are now merging, because their requirements are converging. MIS increasingly needs real-time performance, while real-time systems are required to handle increased data processing workloads. The embedded information system addresses both needs.

Aircraft and the Year 2000

The year 2000 (Y2K) was a major concern for the aircraft computer industry. Many of the embedded computers on aircraft and aircraft support functions were vulnerable to Y2K faults because of their age. The basic problem with those computers was that a year was represented by its low-order two digits. Instead of the year having four digits, these computers saved processing power by using the last two digits of the calendar year. For example, 1999 is represented as 99, which is not a problem until you reach the year 2000, represented as 00. Even with this representation, problems are limited to those algorithms sensitive to calendar dates. An obvious problem is when an algorithm divides by the calendar date, which is division by 0. Division by 0 is an illegal computer operation, causing problems such as infinite loops, execution termination, and system failure. The most commonly mentioned issue is the subtraction of dates to determine time durations and to compare dates. The problem is not that the computer programs fail in a very obvious way (e.g., divide-by-zero check) but rather that the program computes an incorrect result without any warning or indication of error. Lefkon and Payne (11) discuss Y2K and how to make embedded computers Y2K-compliant.

Aircraft Application Program Interfaces

An application programming interface (API) is conventionally defined as an interface used by one program to make use of the services of another program. The human interface to a system is usually referred to as the user interface, or, less commonly, the human–computer interface. Application programs are software written to solve specific problems. For example, the embedded computer software that paints the artificial horizon on a heads-up display is an application program. A switch that turns the artificial horizon on or off is an API. Gal-Oz and Isaacs (12) discuss APIs and how to relieve bottlenecks of software debugging.

Aircraft Control

Landau (1) defines a control as an instrument or apparatus used to regulate a mechanism or a device used to adjust or control a system. There are two concepts with control. One is the act of control. The other is the type of device used to enact control. An example of an act of control is when a pilot initiates changes to throttle and stick settings to alter flight path. The devices of control, in this case, are the throttle and stick.

Control can be active or passive. Active control is force-sensitive. Passive control is displacement-sensitive.

Mechanical control is the use of mechanical devices, such as levers or cams, to regulate a system. The earliest form of mechanical flight control was wires or cables, used to activate ailerons and stabilizers through pilot stick and foot pedal movements. Today, hydraulic control, the use of fluids for activation, is typical. Aircraft control surfaces are connected to stick and foot pedals through hydraulic lines. Pistons in the control surfaces are pushed or pulled by associated similar pistons in the stick or foot pedal. The control surfaces move accordingly.

Electronic control is the use of electronic devices, such as motors or relays, to regulate a system. A motor is turned on by a switch, and it quickly changes control surfaces by pulling or pushing a lever on the surface. Automatic control is a system-initiated control, which is a system-initiated response to a known set of environmental conditions. Automatic control was used for early versions of automatic pilot systems, which tied flight control feedback systems to altitude and direction indicators. The pilot sets his desired course and altitude, which is maintained through the flight control's automatic feedback system.

To understand the need for computers in these control techniques, it is important to note the progression of the complexity of the techniques. The earliest techniques connected the pilot directly to his control surfaces. As the aircraft functionality increased, the pilot's workload also increased, requiring his (or his aircrew's) being free to perform other duties. Additionally, flight characteristics became more complex, requiring more frequent and instantaneous control adjustments. The use of computers helped offset and balance the increased workload in aircraft. The application of computers to flight control provides a means for processing and responding to multiple complex flight control requirements.

Aircraft Computer Hardware

For aircraft computers, hardware includes the processors, buses, and peripheral devices inputting to and outputting from the computers. Landau (1) defines hardware as apparatus used for controlling a spacecraft; the mechanical, magnetic, and electronic design, structure, and devices of a computer; and the electronic or mechanical equipment that uses cassettes, disks, and so on. The computers used on an aircraft are called *processors*. The processor takes inputs from peripheral devices and provides specific computational services for the aircraft.

There are many types and functions of processors on an aircraft. The most obvious processor is the central computer, also called the mission computer. The central computer provides direct control and display to the aircrew. The federated architecture (discussed in more detail later) is based on the central computer directing the scheduling and tasking of all the aircraft subsystems. Other noteworthy computers are the data processing and signal processing computers of the radar subsystem and the computer of the inertial navigation system. Processors are in almost every component of the aircraft. Through the use of an embedded processor, isolated components can perform independent functions as well as self-diagnostics.

Distributed processors offer improved aircraft performance and, in some cases, redundant processing capability. *Parallel* processors are two or more processors configured to increase processing power by sharing tasks. The workload of the shared processing activity is distributed among the pooled processors to decrease the time it takes to form solutions. Usually, one of the processors acts as the lead processor, or master, while the other processor(s) act as slave(s). The master processor schedules the tasking and integrates the final results, which is particularly useful on aircraft in that processors are distributed throughout the aircraft. Some of these computers can be configured to be parallel processors, offering improved performance and redundancy. Aircraft system redundancy is important because it allows distributed parallel processors to be reconfigured when there is a system failure. Reconfigurable computers are processors that can be reprogrammed to perform different functions and activities. Before computers, it was very difficult to modify systems to adapt to their changing requirements. A reconfigurable computer can be dynamically reprogrammed to handle a critical situation, and then it can be returned to its original configuration.

Aircraft Buses

Buses are links between computers (processors), sensors, and related subsystems for transferring data inputs and outputs. Fink and Christiansen (8) describe two primary buses as data buses and address buses. To complete the function of an MPU, a microprocessor must access memory and peripheral devices, which is accomplished by placing data on a bus, either an address bus or a data bus, depending on the function of the operation. The standard 16-bit microprocessor requires a 16-line parallel bus for each function. An alternative is to multiplex the address or data bus to reduce the number of pin connections. Common buses in aircraft are the Military Standard 1553 Bus (Mil-Std-1553) and the General-Purpose Interface Bus (GPIB), which is the IEEE Standard 488 Bus.

Aircraft Software

Landau (1) defines software as the programs, routines, and so on for a computer. The advent of software has provided great flexibility and adaptability to almost every aspect of life, which is especially true in all areas of aerospace sciences, where flight control, flight safety, in-flight entertainment, navigation, and communications are continuously being improved by software upgrades.

Operation Flight Programs. An operational flight program (OFP) is the software of an aircraft embedded computer system. An OFP is associated with an aircraft's primary flight processors, including the central computer, vertical and multiple display processors, data processors, signal processors, and warning receivers. Many OFPs in use today require dedicated software integrated support environments to upgrade and maintain them as the mission requirements of their parent aircraft are modified. The software integrated support environment [also called avionics integrated support environment (AISE), centralized software support activity (CSSA), and software integration laboratory (SIL)] not only allows an OFP to be updated and maintained, but also provides capabilities to perform unit testing, subsystem testing, and some of the integrated system testing.

Assembly Language. Assembly language is a machine (processor) language that represents inputs and outputs as digital data and that enables the machine to perform operations with those data. For a good understanding of the Motorola 6800 Assembler Language, refer to Bishop (13). According to Seidman and Flores (14), the *lowest-level* (closest to machine) language available to most computers is assembly language. When one writes a program in assembly code, alphanumeric characters are used instead of binary code. A special program called an assembler (provided with the machine) is designed to take the assembly statements and convert them to machine code. Assembly language is unique among programming languages in its one-to-one correspondence between the machine code statements produced by the assembler and the original assembly statements. In general, each line of assembly code assembles into one machine statement.

Higher-Order Languages. Higher-order languages (HOLs) are computer languages that facilitate human language structures to perform machine-level functions. Seidman and Flores (14) discuss the level of discourse of a programming language as its distance from the underlying properties of the machine on which it is implemented. A low-level language is close to the machine, and hence provides access to its facilities almost directly; a high-level language is far from the machine, and hence insulated from the machine's peculiarities. A language may provide both high-level and low-level constructs. Weakly typed languages are usually high-level, but often provide some way of calling low-level subroutines. Strongly typed languages are always high-level, and they provide means for defining entities that more closely match the real-world objects being modeled. Fortran is a low-level language that can be made to function as a high-level language by use of subroutines designed for the application. APL, Sobol, and SETL (a set-theoretic language) are high-level languages with fundamental data types that pervade their language. Pascal, Cobol, C, and PL/I are all relatively low-level languages, in which the correspondence between a program and the computations it causes to be executed is fairly obvious. Ada is an interesting example of a language with both low-level properties and high-level properties. Ada provides quite explicit mechanisms for specifying the layout of data structures in storage, for accessing particular machine locations, and even for communicating with machine interrupt routines, thus facilitating low-level requirements. Ada's strong typing qualities, however, also qualify it as a high-level language.

High-level languages have far more expressive power than low-level languages, and the modes of expression are well integrated into the language. One can write quite short programs that accomplish very complex operations. Gonzalez (15) developed an *Ada Programmer's Handbook* that presents the terminology of the HOL Ada and examples of its use. He also highlights some of the common programmer errors and examples of those errors. Sodhi (16) discusses the advantages of using Ada. Important discussions of software lifecycle engineering and maintenance are presented, and the concept of configuration management is presented.

The package concept is one of the most important developments to be found in modern programming languages, such as Ada, Modula-2, Turbo Pascal, C++, and Eiffel. The designers of the different languages have not agreed on what terms to use for this concept: Package, module, unit, and class are commonly used. It is generally agreed, however, that the package (as in Ada) is the essential programming tool to be used for going beyond the programming of very simple class exercises to what is generally called software engineering or building production systems. Packages and package-like mechanisms are important tools used in software engineering to produce production systems. Feldman (17) illustrates the use of Ada packages to solve problems.

Databases. Database are essential adjuncts to computer programming. Databases allow aircraft computer applications the ability to carry pertinent information (such as flight plans or navigation waypoints) into their missions, rather than generating them enroute. Databases also allow the aircrew to collect performance information about the aircraft's various subsystems, providing a capability to adjust the aircraft in flight and avoid system failures.

Elmasri and Navathe (18) define a database as a collection of related data. Data are described as known facts that can be recorded and have implicit meaning. A simple example consists of the names, telephone numbers, and addresses of an indexed address book. A database management system (DBMS) is a collection of programs that enable users to create and maintain a database. The DBMS is hence a general-purpose software system that facilitates the processes of defining, constructing, and manipulating databases for various applications.

Verification and Validation. A significant portion of the aircraft computer's lifecycle cost is system and software testing, performed in various combinations of unit-level, subsystem-level, integrated-system-level, developmental, and operational testing. These types of tests occur frequently throughout the life of an aircraft system because there are frequent upgrades and modifications to the aircraft and its various subsystems. It is possible to isolate acceptance testing to particular subsystems when minor changes are made, but this is the exception. Usually, any change made to a subsystem affects other multiple parts of the system. As aircraft become increasingly dependent on computers (which add complexity by the nature of their interdependences), and as their subsystems become increasingly integrated, the impact of change also increases drastically. Cook (19) shows that a promising technology to help understand the impact of aircraft computer change is the Advanced Avionics Verification and Validation (AAV&V) program developed by the Air Force Research Laboratory.

Sommerville (20) develops the concepts of program verification and validation. Verification involves checking that the program conforms to its specification. Validation involves checking that the program as implemented meets the expectations of the user.

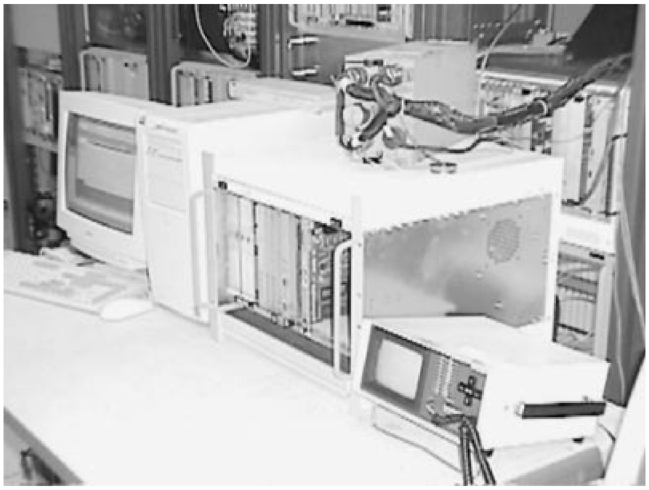


Figure 2. An aircraft avionics support bench.

Figure 2 shows an aircraft avionics support bench, which includes real components from the aircraft such as the FCC line replaceable unit (LRU) sitting on top of the pictured equipment. Additional equipment includes the buses, cooling, and power connection interfaces, along with monitoring and displays. On these types of benches, it is common to emulate system and subsystem responses with testing computers such as the single-board computers illustrated.

Figure 3 shows another verification and validation asset called the workstation-based support environment. This environment allows an integrated view of the aircraft's performance by providing simulations of the aircraft's controls and displays on computer workstations. The simulation is interfaced with stick and throttle controls, vertical situation displays, and touch-screen avionics switch panels.

Object-Oriented Technology. Object-oriented (OO) technology is one of the most popular computer topics of the 1990s. OO languages such as C++ and Ada 95 offer tremendous opportunities to capture complex representations of data and then save these representations in reusable objects. Instead of using several variables and interactions to describe some item or event, this same item or event is described as an object. The object contains its variables, control-flow representations, and data-flow representations. The object is a separable program unit, which can be reused, reengineered, and archived as a program unit. The power of this type of programming is that when large libraries of OO programming units are created, they can be called on to greatly reduce the workload of computer software programming. Gabel (21) says that OO technology lets an object (a software entity consisting of the data for an action and the associated action) be reused in different parts of the application, much as an engineered hardware product can use a standard type of resistor or microprocessor. Elmasri and Navathe (18) describe an OO database as an approach with the flexibility to handle complex requirements without being limited by the data types and query languages available in traditional database systems.

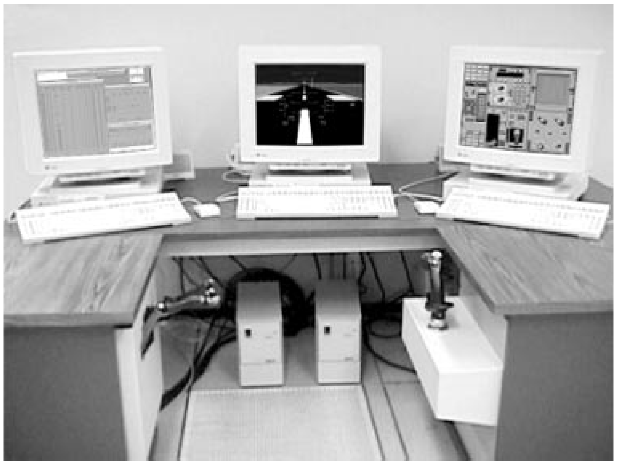


Figure 3. A workstation-based aircraft avionics support environment.

Open System Architecture. Open system architecture is a design methodology that keeps options for updating systems open by providing liberal interfacing standards. Ralston and Reilly (22) state that open architectures pertain primarily to personal computers. An open architecture is one that allows the installation of additional logic cards in the computer chassis beyond those used with the most primitive configuration of the system. The cards are inserted into slots in the computer's motherboard—the main logic board that holds its CPU and memory chips. A computer vendor that adopts such a design knows that, because the characteristics of the motherboard will be public knowledge, other vendors that wish to do so can design and market customized logic cards. Open system architectures are increasingly important in modern aircraft applications because of the constant need to upgrade these systems and use the latest technical innovations. It is extremely difficult to predict interconnection and growth requirements for next-generation aircraft, which is exactly what an open architecture attempts to avoid the need for.

Client-Server Systems. A client-server system is one in which one computer provides services to another computer on a network. Ralston and Reilly (22) describe the file-server approach as an example of client-server interaction. Clients executing on the local machine forward all file requests (e.g., open, close, read, write, and seek) to the remote file server. The server accepts a client's requests, performs its associated operation, and returns a response to the client. Indeed, if the client software is structured transparently, the client need not even be aware that files being accessed physically reside on machines located elsewhere on the network. Client-server systems are being applied on modern aircraft, where highly distributed resources and their aircrew and passenger services are networked to application computers.

Subsystems. The major subsystems of an aircraft are its airframe, power plant, avionics, landing gear, and controls. Landau (1) defines a subsystem as any system that is part of a larger system. Many of the subsystems on an aircraft have one or more processors associated with them. It is a complex task to isolate and test the assorted subsystems.

Another layer of testing below subsystem testing is *unit testing*. A *unit* of a subsystem performs a function for it. For example, in the radar subsystem, the units include its signal processor and its data processor. In order to test a system adequately, each of its lowest-level items (units) must be tested. As the units affect and depend on each other, another layer of testing addresses that layer of dependences. In the same fashion, subsystem testing is performed and integrated with associated subsystems. It is important to test not only at the unit and the subsystem level, but at the system and operational level. The system level is where the subsystems are brought together to offer the system functionality. *System integration* is the process of connecting subsystem components into greater levels of system functionality until the complete system is realized. The *operational level* of testing is where the subsystem is exercised in its actual use.

Line Replaceable Units. LRUs are subsystems or subsystem components that are self-contained in durable boxes containing interface connections for data, control, and power. Many LRUs also contain built-in test (BIT) capabilities that notify air and maintenance crews when a failure occurs. A powerful feature of LRUs is that functionality can be compartmentalized. When a failure is detected, the LRU can easily be pulled and replaced, restoring the aircraft to service within moments of detection.

Graceful Degradation. All systems must have plans to address partial or catastrophic failure. System failure in flight controls is often catastrophic, whereas system failure in avionics can be recovered from. For this reason, most flight-critical systems have built-in redundant capabilities (sometimes multiple layers of redundancy), which are automatically activated when the main system or subsystem fails. Degraded system behavior occurs when the main system fails and backup systems are activated. The critical nature of system failure requires immediate activation of backup systems and recognition by all related subsystems of the new state of operation. *Graceful degradation* is the capability of aircraft computers to continue operating after incurring system failure. Graceful degradation is less than optimal performance, and may activate several layers of decreasing performance before the system fails. The value of graceful degradation is that the aircrew has time to respond to the system failure before a catastrophic failure occurs.

AEROSPACE

Computer technologies have helped provide a continuum of improvements in aircraft performance that has allowed the airspace where aircraft operate to increase in range and altitude. Landau (1) defines aerospace as the Earth's atmosphere and the space outside it, considered as one continuous field. Because of its rapidly increasing domain of air and space travel, the U. S. Air Force is beginning to refer to itself as the U. S. Aerospace Force. Modern air-space vehicles are becoming increasingly dependent on information gleaned from ground stations, satellites, other air-space vehicles, and onboard sensors to perform their mission. These vehicles use signals across the electromagnetic spectrum. Antennas can be found in multiple locations on wings, the fuselage, tails, and draglines. If antennas are located too close together, their signals can interfere with each other, called crossed frequency transmission. This interference reduces the efficiency of each affected antenna. Placement of multiple antennas requires minimizing the effects of crossed frequency transmissions. Techniques for minimization include antenna placement, filtering, and timing, which presents another challenge for aircraft computers to sort and process these multiple signals. Perry and Geppert (4) show how the aircraft electromagnetic spectrum is becoming busy, and thus, dangerous for aerospace communications.

Legacy Systems

Legacy systems are fielded aircraft, or aircraft that are in active use. Probably the only nonlegacy aircraft are experimental or prototype versions. Legacy aircraft are often associated with aging issues, more commonly known as parts obsolescence. A growing problem in these systems is the obsolescence of entire components, including the many computers used on them. Aircraft, like many other systems, are designed with expected lifetimes of 10 to 15 years. Because of the high replacement costs, lifetimes are often doubled and tripled by rebuilding and updating the aircraft. To reduce costs, as many of the original aircraft components as possible are kept. Problems develop when these components are no longer produced or stockpiled. Sometimes, subsystems and their interfaces have to be completely redesigned and produced at great cost in order to keep an aircraft in service. System architectures and standard interfaces are constantly being modified to address these issues. Aircraft evolve during their lifetimes to a more open architecture. This open architecture, in turn, allows the aircraft components to be more easily replaced, thus making further evolution less expensive.

Unmanned Air Vehicles

Unmanned air vehicles (UAVs) are aircraft that are flown without aircrews. Their use is becoming increasingly popular for military applications. Many of the new capabilities of UAVs come from the improved computers. These computers allow the vehicles to have increased levels of autonomy and to perform missions that once required piloted aircraft. Some of these missions include reconnaissance and surveillance. These same types of missions are finding increasing commercial importance. UAVs offer tremendous advantages in lifecycle cost reductions because of their small size, ease of operation, and ability to be adapted to missions.

MAN–MACHINE SYSTEMS

An aircraft is an example of a man–machine system. Other examples are automobiles and boats. These machines have the common attribute of being driven by a human. Landau (1) defines man–machine systems as sets of manually performed and machine-performed functions, operated in conjunction to perform an operation. The aircraft computer is constantly changing the role of the human in the aircraft machine. The earliest aircraft required the constant attention of the pilot. Improved flight control devices allowed the pilot freedom for leisure or for other tasks. Modern aircraft computers have continued the trend of making the aircraft more the machine and less the man system.

Human Factors of Aircraft Computers

Human factors is the science of optimal conditions for human comfort and health in the human environment. The human factors of aircraft computers include the positioning of the controls and displays associated with the aircrew's workloads. They also provide monitoring and adjustment of the aircraft human environment, including temperature, oxygen level, and cabin pressure.

Man–Machine Interface

The *man–machine interface* is the place where man's interactions with the aircraft coordinate with the machine functionality of the aircraft. An example of a man–machine interface is the API, which is where a person provides inputs to and receives outputs from computers. These types of interfaces include keyboards (with standard ASCII character representation), mouse pads, dials, switches, and many varieties of monitors. A significant interface in aircraft comprises their associated controls and displays, which provide access to the flight controls, the sensor suite, the environmental conditions, and the aircraft diagnostics through the aircraft's central computer. Control sticks, buttons, switches, and displays are designed based on human standards and requirements such as seat height, lighting, accessibility, and ease of use.

Voice-Activated Systems. Voice-activated systems are interfaces to aircraft controls that recognize and respond to aircrew's verbal instructions. A voice-activated input provides multiple input possibilities beyond the limited capabilities of hands and feet. Voice-activated systems have specified sets of word commands and are trained to recognize a specific operator's voice.

Aircraft Computer Visual Verification

Visual verification is the process of physically verifying (through sight) the correct aircraft response to environmental stimuli. This visual verification is often a testing requirement. It is usually done through the acceptance test procedure (ATP) and visual inspections of displays through a checklist of system and subsystem inputs. Until recently, visual verification has been a requirement for pilots, who have desired the capability to see every possibility that their aircraft might encounter. This requirement is becoming increasingly difficult to implement because of the growing complexity and workload of the aircraft's computers and their associated controls and displays. In the late 1980s to early 1990s, it required about 2 weeks to visually verify the suite of an advanced fighter system's avionics. This verification can no longer be accomplished at all with current verification and validation techniques. Several months would be required to achieve some level of confidence that today's modern fighters are flight-safe.

Air Traffic Control

Air traffic control is the profession of monitoring and controlling aircraft traffic through an interconnected ground-based communication and radar system. Perry (23) describes the present capabilities and problems in air traffic control. He also discusses the future requirements for this very necessary public service. Air traffic controllers view sophisticated displays, which track multiple aircraft variables such as position, altitude, velocity, and heading. Air traffic control computers review these variables and give the controllers continuous knowledge of the status of each aircraft. These computers continuously update and display the aircraft in the ground-based radar range. When potential emergency situations, such as collision, develop, the computer highlights the involved aircraft on the displays, with plenty of lead time for the controller to correct each aircraft's position.

AIRCRAFT CONTROL AND COMPUTERS

D' Azzo and Houpis (24) give a good explanation of the complexity of what is needed for an aircraft control system. The feedback control system used to keep an airplane on a predetermined course or heading is necessary for the navigation of commercial airliners. Despite poor weather conditions and lack of visibility, the airplane must maintain a specified heading and altitude in order to reach its destination safely. In addition, in spite of rough air, the trip must be made as smooth and comfortable as possible for the passengers and crew. The problem is considerably complicated by the fact that the airplane has six degrees of freedom, which makes control more difficult than control of a ship, whose motion is limited to the surface of the water.

A flight controller is used to control aircraft motion. Two typical signals to the system are the correct flight path, which is set by the pilot, and the level position of the airplane. The ultimately controlled variable is the actual course and position of the airplane. The output of the control system, the controlled variable, is the aircraft heading.

In conventional aircraft, three primary control surfaces are used to control the physical three-dimensional attitude of the airplane: the elevators, the rudder, and the ailerons. A directional gyroscope (gyro) is used as the error-measuring device. Two gyros must be used to provide control of both heading and attitude of the airplane. The error that appears in the gyro as an angular displacement between the rotor and case is translated into a voltage by various methods, including the use of transducers such as potentiometers, synchros, transformers, or microsyns. Selection of the method used depends on the preference of the gyro manufacturer and the sensitivity required. Additional stabilization for the aircraft can be provided in the control system by rate feedback. In other words, in addition to the primary feedback, which is the position of the airplane, another signal proportional to the angular rate of rotation of the airplane around the vertical axis is fed back in order to achieve a stable response. A *rate gyro* is used to supply this signal. This additional stabilization may be absolutely necessary for some of the newer high-speed aircraft.

In reading through this example, it should be obvious that as the complexity of the control feedback system of the aircraft increases, a need for computer processing to evaluate the feedback and to adjust or recommend flight control adjustments exists. Additional feedback may come from global positioning, from ground-based navigation systems through radio inputs, and from other aircraft. The computer is able to integrate these inputs into the onboard flight control inputs and provide improved recommendations for stable flight.

REAL-TIME SYSTEMS

The computers on aircraft are required to perform their functions within short times. Flight control systems must make fine adjustments quickly in order to maintain stable flight. Sensor suites must detect and analyze potential threats before it is too late. Cabin pressure and oxygen must be regulated as altitude changes. All these activities, plus many others on aircraft, must happen in real time.

Nielsen (25) defines a real-time system as a controlled (by software or firmware) system that performs all of its process functions within specified time constraints. A real-time system usually includes a set of independent hardware devices that operate at widely differing speeds. These devices must be controlled so that the system as a whole is not dependent on the speed of the slowest device. Hatley and Pirbhai (26) describe timing as one of the most critical aspects of modern real-time systems. Often, the system's response must occur within milliseconds of a given input event, and every second it must respond to many such events in many different ways.

Flight-Critical Systems

Flight-critical systems are those activities of an aircraft that must be completed without error in order to maintain life and flight. The aircraft flight controls, engines, landing gear, and cabin environment are examples of flight-critical systems. Failures in any of these systems can have catastrophic results. Flight-critical systems are held to tight levels of performance expectations, and often have redundant backups in case of failure.

Federated Systems

Federated systems are loosely coupled distributed systems frequently used in aircraft system architectures to tie multiple processors in multiple subsystems together. The loose coupling allows the multiple subsystems to operate somewhat autonomously, but have the advantage of the shared resources of the other subsystems. A typical aircraft federated system might include its central computer, its INS, its radar system, and its air-vehicle management system. The INS provides the radar with the aircraft's present position, which is reported to the pilot through displays put forth by the central computer. The pilot adjusts his course through the air-vehicle management system, which is updated by the INS, and the cycle is repeated. These subsystems perform their individual functionality while providing services to each other.

Cyclic Executive

A *cyclic executive* on an aircraft computer provides a means to schedule and prioritize all the functions of the computer. The executive routine assigns the functions and operations to be performed by the computer. These assignments are given a specific amount of clock time to be performed. If the assignment does not complete its task in its allocated time, it is held in a wait state until its next clock period. From the beginning of the clock period to its end is one clock cycle. High-priority functions are assigned faster clock cycles, whereas low-priority functions are assigned slower cycles. For example, the high-priority executive function might be assigned a speed of 100 cycles per second, whereas some lower-priority function might have 5 cycles per second to complete its tasks. Sometimes, the latter might take several clock cycles to perform a task. An additional feature of cyclic executives is that they are equipped with interrupts, which allow higher-priority systems to break into the executive assignments for system-level assigned tasking.

There are several types of scheduling methodologies that provide performance improvements in cyclic executives. One of the more prominent is rate monotonic analysis (RMA), which determines the time requirement for each function and the spare time slots, and then makes time assignments.

THE NETWORK-CENTRIC AIRCRAFT

In the age of the World Wide Web (www), it is hard to imagine the concept of platform-centric systems, such as many of the aircraft that are in service today. These aircraft were built with the requirement to be self-sufficient, safe, and survivable. Dependency on off-board inputs was minimized as advanced avionics technologies allowed aircraft to assess and respond to their environment flight dynamics independently. These aircraft have been conceived, created, and maintained right up to this new information age. It takes significant effort to open the architectures of these aircraft, in order for their existing capabilities to be enhanced by outside information. Fortunately, the adaptability and flexibility of aircraft computers makes this process possible for many of these aircraft.

The modern aircraft (conceived, created, and maintained since the mid-1990s) is a network-centric aircraft. These aircraft take full advantage of the platform-centric systems with independent suites of avionics and aircraft computers. However, they have the additional ability to adapt to their environmental flight dynamics, which is possible because these systems have access to the most recent information about their environment. They can interactively communicate with other aircraft entering and leaving their environment, as well as take advantage of the information services available in that environment. The aircraft computers work very much the same as in the platform-centric aircraft, but with improved and broader information than was available before (27,28).

The network-centric aircraft can take full advantage of route changes caused by heavy air traffic, threats, or weather. It can send its systems self-diagnostics ahead to maintenance crews, who can have parts and resources available reducing the service re-cycling time of the aircraft. It can inform passengers and crew about their individual travel plans and the options available to them as they arrive at their destinations. It can help air traffic controllers and flight planners manage the dynamic workload of the many aircraft in service.

BIBLIOGRAPHY

1. S. Landou, *Webster Illustrated Contemporary Dictionary, Encyclopedic Edition*. Chicago: J. G. Ferguson, 1992.
2. J. F. Wakerly, *Digital Design Principles and Practices*. Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 1–48, 53–138.
3. V. C. Hamacher, Z. G. Vranesic, and S. G. Zaky, *Computer Organization*, 2nd ed. New York: McGraw-Hill, 1984.
4. T. Perry and L. Geppert, Do portable electronics endanger flight, *IEEE Spectrum*, **33**(9): 26–33, 1996.
5. A. Golden, *Radar Electronic Warfare*. Washington: AIAA Education Series, 1987.
6. G. W. Stimson, *Introduction to Airborne Radar*. El Segundo, CA: Hughes Aircraft, 1983, pp. 107, 151–231.
7. G. Welch and G. Bishop, An introduction to the Kalman filter, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, http://www.cs.unc.edu/~welch/media/pdf/kalman.pdf, 1997.
8. D. Fink and D. Christiansen, *Electronics Engineers' Handbook*, 3rd ed., New York: McGraw-Hill, 1989.
9. J. Adam and T. Gibson, Warfare in the information age, *IEEE Spectrum*, **28**(9): 26–42, 1991.
10. W. Sweet, The glass cockpit, *IEEE Spectrum*, **32**(9): 30–38, 1995.
11. D. Lefkon and B. Payne, Making embedded systems year 2000 compliant, *IEEE Spectrum*, **35**(6): 74–79, 1998.
12. S. Gal-Oz and M. Isaacs, Automate the bottleneck in embedded system design, *IEEE Spectrum*, **35**(8): 62–67, 1998.
13. R. Bishop, *Basic Microprocessors and the 6800*. Hasbrouck Heights, NJ: Hayden, 1979.
14. A. Seidman and I. Flores, *The Handbook of Computers and Computing*. New York: Van Norstrand Reinhold, 1984, pp. 327–502.
15. D. W. Gonzalez, *Ada Programmer's Handbook*. Redwood City, CA: Benjamin/Cummings, 1991.
16. J. Sodhi, *Managing Ada Projects*. Blue Ridge Summit, PA: TAB Books, 1990.
17. M. B. Feldman and E. B. Koffman, *Ada Problem Solving and Program Design*. Reading, MA: Addison-Wesley, 1992.
18. R. Elmasri and S. B. Navathe, *Fundamentals of Database Design*, 2nd ed. Redwood City, CA: Benjamin/Cummings, 1994.
19. R. Cook, The advanced avionics verification and validation II final report, Air Force Research Laboratory Technical Report ASC-99-2078, Wright-Patterson AFB.
20. I. Sommerville, *Software Engineering*, 3rd ed. Reading, MA: Addison-Wesley, 1989.
21. D. Gabel, Software engineering, *IEEE Spectrum*, **31**(1): 38–41, 1994.
22. A. Ralston and E. Reilly, *Encyclopedia of Computer Science*. New York: Van Nostrand Reinhold, 1993.
23. T. Perry, In search of the future of air traffic control, *IEEE Spectrum*, **34**(8): 18–35, 1997.
24. J. J. D' Azzo and C. H. Houpis, *Linear Control System Analysis and Design*, 2nd ed. New York: McGraw-Hill, 1981, pp. 143–146.
25. K. Nielsen, *Ada in Distributed Real-Time Systems*. New York: Intertext, 1990.
26. D. J. Hatley and I. A. Pirbhai, *Strategies for Real-Time System Specification*. New York: Dorset House, 1988.
27. D. S. Alberts, J. J. Garstka, and F. P. Stein, *Network Centric Warfare*. Washington D.C.: CCRP Publication Series, 2000.
28. D. S. Alberts and R. E. Hayes, *Power to the Edge*. Washington D.C.: CCRP Publication Series, 2003.

FURTHER READING

G. Buttazo, *Hard Real-Time Computing Systems*. Norwell, MA: Kluwer, 1997.

R. Comerford, PCs and workstations, *IEEE Spectrum*, **30**(1): 26–29, 1993.

D. Dooling, Aerospace and military, *IEEE Spectrum*, **35**(1): 90–94, 1998.

J. Juliussen and D. Dooling, Small computers, aerospace & military, *IEEE Spectrum*, **32**(1): 44–47, 76–79, 1995.

K. Kavi, *Real-Time Systems, Abstractions, Languages, and Design Methodologies*. Los Alamitos, CA: IEEE Computer Society Press, 1992.

P. Laplante, *Real-Time Systems Design and Analysis, an Engineer's Handbook*. Piscataway, NJ: IEEE Press, 1997.

M. S. Roden, *Analog and Digital Communication Systems*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1985.

H. Taub, *Digital Circuits and Microprocessors*. New York: McGraw-Hill, 1982.

C. Weitzman, *Distributed Micro/Minicomputer*. Englewood Cliffs, NJ: Prentice-Hall, 1980.

CHARLES P. SATTERTHWAITE
United States Air Force
Wright-Patterson AFB, Ohio.

COMPUTERIZED DICTIONARIES: INTEGRATING PORTABLE DEVICES, TRANSLATION SOFTWARE, AND WEB DICTIONARIES TO MAXIMIZE LEARNING

BACKGROUND STUDIES ON BILINGUAL AND ELECTRONIC DICTIONARIES

Many articles comparing various types of dictionaries may be found in the first fully annotated bibliographic review of studies in this broad field of **lexicography** (the making of dictionaries, whether print or electronic), entitled *Pedagogical Lexicography Today* by Dolezal and McCreary (1), under either the *learner dictionary* category or under *traditional dictionaries* designed for native readers. Articles on learner dictionaries are grouped by their central focus, namely by whether they are mainly dealing with *bilingual* (giving first language or L1 translations), *bilingualized* (including both L1 and L2 information), or only *monolingual* (providing only English-to-English or other L2 to/from L2 definitions) explanations of target language (TL) vocabulary. Laufer and Kimmel (2) described patterns of use, comparing a particular dictionary's degree of accessibility versus difficulty for learners, finding that "Each learner was classified by his favorite look-up pattern...on the basis of these, we argue that the bilingualised dictionary is very effective as it is compatible with all types of individual preferences." (p. 361) (for more information on computerized dictionary writing systems, see http://nlp.fi.muni.cz/dws06/).

Lexical computing is a field of most concern to language teachers, computational linguists, and lexicographers involved in making dictionary writing systems (DWS), software for writing and producing a dictionary. It might include an editor, a database, a web interface, and various management tools (for allocating work, etc.), operating with a dictionary grammar, which specifies the internal structure of the dictionary. Robert Lew (3), whose dissertation provides a massive database for further research in this field, considered the receptive use of bilingual, monolingual, and semi-bilingual dictionaries by Polish learners of English, asking the most basic question for language teachers and dictionary designers (lexicographers) to consider, namely the question of which dictionary is best for whom? Other studies have compared the use of various types of glosses, such as "(paper, electronic textual, electronic pictorial, electronic, and video) on reading comprehension, translation, the number of words looked up, time-on-task and satisfaction of dictionary users. Others investigated incidental vocabulary learning via computer glosses, as reported by Laufer and Levitzky-Aviad (4). Loucky (5–8) compared Japanese college students' accessing speeds for portable devices with using software or mobile phone dictionaries.

Akbulut (9–11) compared the supposed advantage that adding various types of multimedia glossing might bring to language learners. Two crucial findings are well summarized in Chun (12): "...previous studies have found that L2 vocabulary is remembered better when learners look up picture or video glosses in addition to translations of unfamiliar words, but that when given the choice, learners tend to prefer and use the simple translation of words... In summary, research during the last ten years (1995–2005) has found that bilingual dictionaries and multimedia glosses have a more direct impact on vocabulary acquisition than on overall reading comprehension. ..." (pp. 78–81).

A history of lexicography and dictionary development in Japan may be found in Nakao's (13)*The State of Bilingual Lexicography in Japan: Learners' English-Japanese/ Japanese-English Dictionaries*. Other researchers who have examined the individual preferences, needs, and skills of dictionary users (both monolingual and bilingual) include Baxter (14), Tomaszczyk (15), Hartmann (16), Piotrowski (17), Atkins and Knowles (18), and Nuccorini (19). Hulstijn and Atkins (20) suggested that use of electronic dictionaries be studied more systematically. Laufer and Hill (21) examined how users' CALL dictionary look-up behaviors affected their retention. Those who design dictionaries for language learners, whether traditional text or electronic types of dictionaries, can gain much insight from more individualized, long-term studies done in countries where they have a consumer base.

Tomaszczyk (15), who first questioned foreign language learners regarding their preferences and dictionary usage, stated that the vast majority of his close to 450 Polish respondents "would like their dictionaries to give much more extensive treatment to every type of information... would like to have an omnibus dictionary which would cover everything anyone has ever thought of including in dictionaries and encyclopedias" (p. 115). Today, Internet search engines seem to do just that, but are often far too broad, especially for limited English proficiency (LEPs) learners to use efficiently. One solution to this problem is to use the writer's Virtual Language Learning Encyclopedia site at *www.CALL4ALL.us*. Providing instant links to most web dictionaries found on its Dictionaries (D) page <at http://www.call4all.us///home/_all.php?fi=d>, this site enables anyone to find vocabulary information for 500 language pairs systematically, by giving simultaneous instant free access to over 2500 online dictionaries. Moreover, this online multilingual dictionary portal now integrates the many functions of *Wordchamp.com's* versatile *Webreader* on each of its pages, thereby providing automatic glossing from English into over 100 languages for any website, including 40 online newspapers in 10 major languages.

Paper Versus Electronic Dictionaries

Electronic dictionaries are undoubtedly greatly gaining in popularity, so much so that they will soon dominate the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

dictionary scene (22–26). Lew (3) noted these recent trends stating:

> It has been claimed that with the move from paper to online dictionaries, restrictions of space would disappear. That, however, is a simplification at best. While storage space may indeed become irrelevant, there are still severe restrictions as to how much information can be displayed at a time. In fact, even the best currently available display devices are still easily beaten by the old-fashioned printed paper in terms of visual resolution. So space-saving issues will still be with for at least as long as the visual modality is primarily used for information transfer from dictionary to user. . .on-screen presentation of entries has much to offer. . .to the researcher by way of convenience, including a potential to log responses automatically, thus obviating the need for the laborious paperwork and keyboarding at the data entry stage, as well as allowing "unobtrusive observation". (p. 157)

The equivalence of on-screen and paper formats should not be taken for granted, as Laufer (27) found significant and substantial differences in word recall scores between marginal paper glosses and on-screen pop-up window glosses.

DOING LEXICOGRAPHY IN AN ELECTRONIC AGE

Tono (28) predicted the advantages of online media using machine translation, saying "Electronic dictionaries have great potential for adjusting the user interface to users' skill level[s] so that learners with different needs and skills can access information in. . . different way[s]." (p. 216)

First of all, one must note that electronic dictionaries have developed based on a healthy integration of developments in **computerized corpus linguistics** and modern technology, used to enhance learning in many fields, particularly **computer-assisted language learning (or CALL)** or **computer-mediated communications (CMC).**

Laufer and Kimmel (2) provide a clear summary of this field, noting that

> If the consumer is to benefit from the lexicographer's product, the dictionary should be both useful and usable. We suggest a definition of dictionary usefulness as the extent to which a dictionary is helpful in providing the necessary information to its user. Dictionary usability, on the other hand, can be defined as the willingness on the part of the consumer to use the dictionary in question and his/her satisfaction from it. Studies of dictionary use by L2 learners . . . reveal that dictionary usefulness and dictionary usability do not necessarily go hand in hand. (pp. 361–362)

Laufer and Levitzky-Aviad's (4) study recommends working toward designing a bilingualized electronic dictionary (BED) more clear and useful for second language production. Whereas conventional bilingual L1-L2 dictionaries list translation options for L1 words without explaining differences between them or giving much information about how to use various functions, Laufer and Levitzky-Aviad (4) examined the usefulness of an electronic Hebrew-English-English (L1-L2-L2) minidictionary designed for production. Their results demonstrated the superiority of fully bilingualized L1-L2-L2 dictionaries and some unique advantages of the electronic format. Their literature review provides a good overview of this field:

> Surveys of dictionary use indicate that the majority of foreign language learners prefer bilingual L2-L1 dictionaries and use them mainly to find the meaning of unknown foreign (L2) words (Atkins 1985; Piotrowsky 1989). However, if learners writing in L2 need an L2 word designating a familiar L1 concept, they do not readily turn to an L1-L2 dictionary for help. The reason for this may lie in a serious limitation of most L1-L2 bilingual dictionaries. They rarely differentiate between the possible L2 translations of the L1 word, nor do they provide information regarding the use of each translation option. . . An electronic dictionary can fulfill the above requirements since it can combine the features of an L2-*L1bilingual dictionary*, an L1-L2 bilingual dictionary and an L2 monolingual dictionary. The advent of electronic dictionaries has already inspired research into their use and their usefulness as on-line helping tools and as contributors to incidental vocabulary learning. The built in log files can keep track of words looked up, type of dictionary information selected (definition, translation, example, etc.), the number of times each word was looked up, and the time spent on task completion. (pp. 1–2)

Although most electronic dictionaries do **auto-archiving** of any new words by means of their **history search function**, most online dictionaries do not have a means of tracking student use, except for programs like *Wordchamp.com* or *Rikai.com*, which give students a way to archive words they have double-clicked. These words may later be seen, printed, and reviewed. In fact, Wordchamp.com, by far the most sophisticated online electronic dictionary and vocabulary development program, allows users to make online flashcards with sentence examples and links to online texts where target words are found in context. It can also automatically generate about 10 types of online vocabulary quizzes and provides a free **course management system (CMS)** for monitoring students' work online. Wordchamp's Webreader provides the most versatile online glossing engine known, already for over 100 languages, with more being added regularly.

Teachers need to show learners how to best integrate the use of such portable and online dictionaries to make them maximally effective for their language development, in both receptive and productive aspects. Chun (12) noted that learners who could read online text with "access to both internally (instructor-created) glossed words as well as externally glossed words. . . recalled a significantly greater number of important ideas than when they read an online text and had access only to an external (portable electronic) dictionary" (p. 75).

Loucky (29) also examined how to best maximize L2 vocabulary development by using a depth of lexical processing (DLP) scale and vocabulary learning strategies (VLSs) taxonomy together with online CALL resources and systematic instruction in the use of such strategies. It used 40 of the 58 VLSs identified in Schmitt's earlier taxonomy. An electronic dictionary use survey (see Appendix) was designed to solicit information about how students used various computerized functions of electronic or online dictionaries at each major phase of lexical processing to help learners maximize processing in the following eight stages of vocabulary learning: (1) assessing degree of word knowledge, (2) accessing new word meanings, (3) archiving new information for study, (4) analyzing word parts and origins, (5) anchoring new words in short-term memory, (6) associating words in related groups for long-term retention, (7) activating words through productive written or oral use, and (8) reviewing/recycling and then retesting them. Portable devices or online programs that could monitor and guide learners in using these essential strategies should be further developed.

In Loucky's (7) findings, despite being one grade level higher in their proficiency, English majors were outperformed on all types of electronic dictionaries by Computer majors. The author concluded that familiarity with computerized equipment or computer literacy must have accounted for this, and therefore should be carefully considered when developing or using electronic dictionary programs of any sort for language or content learning. His study compared vocabulary learning rates of Japanese college freshmen and functions of 25 kinds of electronic dictionaries, charting advantages, disadvantages, and comments about the use of each (for details, see Loucky (7) Table 1 and Appendix 3; Loucky (8) Tables 1 and 2. For a comparative chart of six most popular EDs for English<->Japanese use, see www.wordtankcentral.com/compare.html).

Generally speaking, language learners prefer access to both first and second language information, and beginning to intermediate level learners are in need of both kinds of data, making monolingual dictionaries alone insufficient for their needs. As Laufer and Hadar (30) and others have shown the benefits of learners using fully bilingualized dictionaries, the important research question is to try to determine which kinds of electronic portable, software, or online dictionaries offer the best support for their needs. Grace (31) found that sentence-level translations should be included in dictionaries, as learners having these showed better short- and long-term retention of correct word meanings. This finding suggests a close relationship exists between processing new terms more deeply, verifying their meanings, and retaining them.

Loucky (32) has researched many electronic dictionaries and software programs, and more recently organized links to over 2500 web dictionaries, which are now all accessible from the site http://www.call4all.us///home/_all.php?fi=d. His aim was to find which kind of EDs could offer the most language learning benefits, considering such educational factors as: (1) better learning rates, (2) faster speed of access, (3) greater help in pronunciation and increased comprehensibility, (4) providing learner satisfaction with ease of use, or user-friendliness, and (5) complete enough meanings to be adequate for understanding various reading contexts.

As expected, among learners of a common major, more proficient students from four levels tested tended to use EDs of all types more often and at faster rates than less language-proficient students did. In brief, the author's studies and observations and those of others he has cited [e.g., Lew (3)] have repeatedly shown the clear benefits of using EDs for more rapid accessing of new target vocabulary. They also point out the need for further study of archiving, and other lexical processing steps to investigate the combined effect of how much computers can enhance overall lexical and language development when used more intelligently and systematically at each crucial stage of first or second language learning. Regular use of portable or online electronic dictionaries in a systematic way that uses these most essential phases of vocabulary acquisition certainly does seem to help stimulate vocabulary learning and retention, when combined with proper activation and recycling habits that maximize interactive use of the target language. A systematic taxonomy of vocabulary learning strategies (VLSs) incorporating a 10-phase set of specific recyclable strategies is given by Loucky (7,29) to help advance research and better maximize foreign language vocabulary development (available at http://www.call4all.us///home/_all.php?fi=../misc/forms).

A summary of Laufer and Levitzky-Aviad's (4) findings is useful for designers, sellers, and users of electronic dictionaries to keep in mind, as their study showed that: "the best dictionaries for L2 written production were the L1-L2-L2 dictionaries... Even though the scores received with the paper version of the L1-L2-L2 dictionary were just as good, the electronic dictionary was viewed more favorably than the paper alternative by more learners. Hence, in terms of usefulness together with user preference, the electronic version fared best" (p. 5). Such results should motivate CALL engineers and lexicographers to produce fully bilingualized electronic dictionaries (as well as print versions), specifically designed not merely to access receptive information to understand word meanings better, but also for L2 production, to practically enable students to actively use new terms appropriately as quickly as possible.

SURVEYING USE OF ELECTRONIC DICTIONARIES

To more thoroughly analyze and compare the types of dictionaries being used by Japanese college students in three college engineering classes, two kinds of surveys were designed by Loucky (29). The first was a general survey about purchase, use, and preferences regarding electronic dictionaries. The second survey (shown in the Appendix) asked questions about how various computerized functions were used at each major phase of lexical processing. The aim was to help learners maximize these eight essential phases of vocabulary learning: (1) assessing degree of word knowledge; (2) accessing new word meanings; (3) archiving new information for study; (4) analyzing word parts and origins; (5) anchoring new words in short-term memory; (6) associating words in related groups for long-term retention; (7) activating words through productive written or oral use; and (8) reviewing/recycling and re-testing them. After re-evaluating how well new words are learned by post-tests, any words not fully understood should be remet through planned re-encounters, retellings, and activities that encourage learners to repeat the vocabulary learning cycle again so that relearning and reactivation can take place.

Table 1. Comparative Chart of Some Translation Software[*]

Al Misbar Translation http://www.almisbar.com/salam_trans.html • Paid Subscription	1 Language Pair • English <-> Arabic
Amikai http://www.amikai.com/products/enterprise/ (under *Translation Demo*) • Free demo version (up to 100 characters) • Full version can be customized with dictionaries.	13 Language Pairs
Babel Fish http://babelfish.altavista.com/ • Can translate a web page or up to 150 words of text.	18 Language Pairs
Ectaco LingvoBit http://www.poltran.com/	1 Language Pair • English <-> Polish
Kielikone WebTranSmart https://websmart.kielikone.fi/eng/kirjaudu.asp • Registration Required • Per-word fee must be paid in advance for translations.	1 Language Pair • English <-> Finnish
ParsTranslator http://www.parstranslator.com/	1 Language Pair • English <-> Farsi
PROMT-Online http://translation2.paralink.com/	7 Language Pairs
Reverso http://www.reverso.net/text_translation.asp • Can translate text or web pages. • Special characters can be inserted onscreen.	5 Language Pairs
SDL Enterprise Translation Server http://www.sdl.com/enterprise-translation-server • Free demonstration (up to 200 words) • Can translate text or web pages. • Used by FreeTranslation.com	5 Language Pairs
SYSTRANBox http://www.systranbox.com/ • Can translate a web page or up to 150 words of text. • Used by AOL, Lycos, Terra, Google, Voila, Wanadoo, Free.fr, and others. • Check results with a human translator.	16 Language Pairs
SYSTRANet http://www.systranet.com/systran/net • More tools than SYSTRANsoft • More language pairs • Quality varies by language pair and subject matter. Check results with a human translator. • Must sign up for a password, but delivery of password is in seconds.	18 Language Pairs

SYSTRANSoft http://www.systransoft.com/ • Can translate a web page, a file (TXT, RTF, or HTML) or up to 150 words of text. • Quality varies by language pair and subject matter. Check results with a human translator.	15 Language Pairs
Tarjim (Registration Required) http://tarjim.ajeeb.com/	1 Language Pair • English > Arabic
Wordchamp.com http://wordchamp.com • Free to all.	Over 100 Language Pairs Instant Glossing; Auto-Archiving; Online Flashcard and Test Creation; Files can be shared internationally between distance learners, as well as internally within intact classes, using its currently free Course Management System (CMS).

*Free unless stated otherwise. Summarized from site by author.

The first survey described Japanese college students' preferences and reasons for purchasing EDs. The second showed self-reported use of PEDS and how their respective functions were seen to aid in different phases of L2 vocabulary learning. Students compared their use to that of print dictionaries. A majority of East Asian students surveyed expressed a preference for using mobile or online dictionaries rather than carry bulkier book dictionaries, although a few English students carry both. These ED preferences and patterns of use need more investigation, but probably hold true wherever the level of economic development is sufficient to support their purchase, as well as the use and availability of Internet access to online dictionary and Webreader glossing functions.

Kobayashi (33) compared the use of pocket electronic versus printed dictionaries to examine the effects of their use on LPSs used. The three major strategies she distinguished were consulting, inferring versus ignoring new terms. She found that "Pocket electronic dictionaries (PEDs) are rapidly becoming popular among L2 learners. Although many L2 learners depend on dictionaries, the prevalent view among L2 researchers and educators is that learners should use dictionaries sparsely. They encourage students to use another lexical processing strategy (LPS), contextual guessing, for better vocabulary learning and reading comprehension. [But] are dictionaries indeed so harmful?" (p. 2).

As some educators and researchers have been concerned about the pedagogical value of EDs because of their perceived limitations, such as insufficient information provided, the possibility of discouraging contextual guessing, and a supposed negative impact on word retention (34-38), these educators' and researchers' concerns require more investigation. So far, however, language learners' preference for them, and EDs' rapidly improving functions appear to be scuttling most of these previous claims. Although native readers have far larger working vocabularies to guess from context, most second language readers prefer and benefit greatly from having both monolingual and bilingual/mother tongue glosses available to them. Kobayashi (39) found that

1. More than two-thirds of the students owned a PED, and most of those who owned a PED exclusively used it regardless of purposes.
2. The PEDs owned by most students cost $100–$400, were of high quality, and did not have the disadvantages identified in other studies, such as brief definitions, the absence of examples, and inaccurate information.
3. Most students were satisfied with their PEDs, especially with their portability, and ease to look up a word, and ease to change from one dictionary to another.
4. The perceived disadvantages included the relative unavailability (or inaccessibility) of detailed usage information, examples, and grammatical information.
5. PEDs enabled students to use different types of dictionaries in different places.
6. Although both PED users and PD users depended on dictionaries, PED users used dictionaries more often. This was especially the case with smaller vocabulary size students.
7. PD users and PED users did not significantly differ in terms of their LPS use, except for the sheer frequency of dictionary consultation.
8. There was a possibility that PED users consulted dictionaries at the expense of contextual guessing.
9. Although students depended on dictionaries, whether PEDs or PDs, they also used guessing strategies frequently. They often used a dictionary to confirm guessed meaning. This was particularly the case with successful students.
10. Larger and smaller vocabulary size students differed in their use of LPSs such as basic dictionary use, extended dictionary use for meaning, extended dictionary use for usage, extended dictionary use for grammatical information, lookup strategies, note-taking strategies, guessing strategies using immediate context, guessing strategies using wider context, combined use of LPSs, and selective use of LPSs.
11. Higher and lower reading ability students differed in their use of LPSs such as basic dictionary use, extended dictionary use for meaning, extended dictionary use for usage, extended dictionary use for grammatical information, lookup strategies, self-initiation, note-taking strategies,

> guessing strategies using immediate context, guessing strategies using wider context, and selective use of LPSs (p.2).

SURVEYING AND MONITORING USE OF VOCABULARY LEARNING STRATEGIES

Vocabulary researchers such as Schmitt (40), Kudo (41), Orita (42), and Loucky (29) have examined more than 50 other effective vocabulary learning strategies, coming up with some useful taxonomies that makers of dictionaries should be aware of and seek to maximize in their design of electronic features and functions in particular. Language learners do appear to benefit greatly from specific strategy training in this essential area of language development (43).

Loucky (29) has presented useful surveys of CBDs or EDs presented in *CALICO Journal*. He also included many recommendations for how to properly integrate computerized lexicons, both portable and online, into CALL as effectively and enjoyably as possible. He explained a useful taxonomy of VLS for all designers and users of computerized dictionaries to help students maximize their learning of target language vocabulary. *CALL Journal* in December, 2005, highlighted the www.CALL4All.us website, showing how learners and teachers may use its extensive encyclopedia of preorganized online dictionaries and language learning links to produce more effective and enjoyable reading and vocabulary learning lessons. These tools include the use of online glossing engines and reading labs, word-surfing games, vocabulary profilers most useful for text analysis and simplification, readability analyzers, and so on.

State-of-the-Art Technical Features

Probably the company offering the largest variety of functions and types of computerized dictionaries for the most languages is Ectaco, whose U.K. site enables one to search for both type of software/platform and particular language pair combination sought. It can be accessed at http://www.ectaco.co.uk/how-find/. Their programs for handheld, portable devices may be found at http://www.ectaco.co.uk/Software-for-Pocket-PC/.

Electronic Dictionaries

Electronic dictionary and electronic translator handhelds are modern, lightweight, and fashionable gadgets with a great variety of features. An electronic translator or dictionary is becoming a definite must-have in many areas of business. More expensive devices are based on advanced speech recognition and text-to-speech technologies. Advanced models may include these useful functions: 1) a business organizer, 2) bidirectional, 3) voice recognition or synthesis, 4) extensive vocabularies (up to 1,000,000 words), 5) grammar references, and 6) phrase banks containing colloquial expressions and common phrases, irregular verbs, and more. Ectaco offers more than 70 titles for over 20 languages at: http://www.ectaco.co.uk/Electronic-Dictionaries/.

Translation Software

For example, Ectaco has devices featuring a wide range of software products, over 220 titles, translation tools, and learning aids for over 35 languages designed for all standard computer platforms, such as Windows, Pocket PC, and Palm OS. Many devices have tools for various language goals (e.g., text translators, accent removers, bidirectional talking dictionaries, localization tools, and language office tools), which include speaking and nonspeaking EOs, voice and travel language translators, handheld PDAs, and software bundles for Pocket PCs, Windows, Palm OS, and Cell phones.

Although some online dictionaries charge fees, a majority are now available for free use. Most of these are now organized at the author's www.CALL4ALL.us site, under Dictionaries Galore! http://www.call4all.us///home/_all.php?fi=d. Many examples of excellent translation software programs and portable, software, and online dictionaries can be seen and even ordered from these sites directly, or from those shown in Table 1.

1. http://www.ectaco.co.uk/how-find/ (Ectaco).
2. http://www.call4all.us///prod/_order.php?pp=2 (For language learning software, http://www. call4all.us///home/_all. php?fi=d links to most web dictionaries).
3. http://www.wor.com/shopping/ (World of Reading Language Learning Software).
4. http://speedanki.com/ (Speedanki.com offers Kanji Level Tests and flash cards to help one learn and review for national Japanese Proficiency Tests).
5. http://quinlanfaris.com/?cat=3 (Compares technical functions and differences between Seiko and Canon Wordtanks and the Seiko SR-E9000 PEDs).
6. http://flrc.mitre.org/Tools/reports/products_list.pl?LID=199# (Translation software and professional tools used for customized and specialized dictionary creations. Completeness of report is dependent on the completeness of the data entries and is expected to improve rapidly over time. Information is provided by each developer or vendor).
7. http://flrc.mitre.org/Tools/internetMT.pl * (These translation programs are intended for giving a general gist of meaning, not as a substitute for human translation. However, this site is the best quick view of machine translation options online, covering 13 online translation engines).

Computerized Dictionaries and Translation Software Programs Available

The most detailed and extensive table of translation software and computerized dictionary products may be found at the Foreign Language Resource Center's http://flrc.mitre.org/Tools/reports/products_list.pl?LID=202. Information provided by each developer or vendor at that site includes company, product names and version, and descriptions of languages and functions included. As about 75 companies are listed, only the list of companies providing these kinds of products will be listed here to make online searches possible. Computerized translation software companies include the following: ABLE Innovations, Alis Technologies; Acapela Group; Agfa Monotype Corporation; Al-Buraq; Arabeyes; Arabic OCR; arabsun.de; ARABVISTA; AramediA; Arava Institute for Environmental Studies; ATA Software Technology Limited; Alchemy Software Development; Abbyy Software House; Applications Technology; Ascender Corporation; Atril UK, Ltd.; Attensity Corporation; Basic Language Systems Corporation; Basis Technology; CACI, Inc.; Ciyasoft Corporation; CIMOS; Automatic Vocalization for Arabic; Automatic-Topic–Detection/ Abstract of Document; Compure, Computer & Language Technology; Ectaco; Galtech Soft, Ltd.; GlobalSight Corporation; International Systems Consultancy; IBM; Ice-LC Software; Idiom Technologies, Inc.; Jubilant Technologies, Inc.; Language Analysis Systems; Language Engineering Company; Language Weaver, Inc., LLC; Lingua; Linguist's Software; Lockheed-Martin; Marine Acoustics, Inc.–VoxTec; Paragon Software GmbH piXlogic; Postchi.com; Melingo, Ltd.; MetaTexis Software and Services; Microsoft Corporation; MultiCorpora R&D, Inc.; Nattiq Technologies; Nisus Software; NovoDynamics.com (Detects new application programming interface, API); Paragon Software; Sakhr Software Company; SDL International; SIL International Publishing Services; Smart Link Corporation; Tavultesoft Pty, Ltd.; Telelingua; THUNDERSTONE SOFTWARE; TM SYSTEMS; TRADOS Corporation; Transclick, Inc.; Translation Experts; translation.net; United Nations Educational, Scientific and Cultural Organization (UNESCO); United Nations; University of California, Davis; University of Maryland; U.S. Army Intel Center; Verity; WORDFAST; World Health Organization; WorldLanguage Resources; and Xerox–The Document Company.

Among the various types of advanced applications provided by innumerable types of software from these companies are multilingual translation; dictionaries; language learning applications and toolkits; speech recognition; information retrieval; multilingual word processing, spelling, and grammar; optical character recognition with easy insertion into Windows word processing; and web development and browsing.

Discussion and Pedagogical Implications

Common findings can now be summarized about electronic lexicons from a broad reading of research in the field by Kobayashi (33), Laufer and Hill (44), and Hill and Laufer (45), combined with the author's findings as follows:

1. PEDs facilitate L2 learning rather than hindering it. Regardless of whether they are using electronic or print dictionaries, successful students use effective lexical processing strategies. Moreover, PEDs facilitate dictionary use. Therefore, the use of PEDs should not be discouraged.
2. Rather than discouraging the use of PEDs, teachers could advise students to use a PED and a PD for different purposes.
3. Dictionary use and contextual guessing are not mutually exclusive. Successful learners use both dictionaries and contextual guessing more often than less successful learners. Dictionary use should not be frowned on for the reason that it hinders contextual guessing.
4. Many LPSs involving dictionary use and guessing are helpful for both vocabulary learning and reading. These strategies should be taught to students.
   a. Teachers should give students instruction in how to use a dictionary effectively, particularly how to look for a variety of information and what dictionaries are available.
   b. Guessing is also important for vocabulary learning and reading. Teachers should give students instruction in how to guess at word meaning using wider and immediate contexts.
   c. The ability to use a dictionary selectively is also important. Teachers should instruct students when to use a dictionary and when to turn to other LPSs.
5. Some strategies are more important for vocabulary learning than reading comprehension, and some strategies are more important for reading comprehension than for vocabulary learning. These strategies should be taught considering the desired skills and purposes of a reader or language learner (29,33).
6. Successful language learners tend to use a much wider variety of effective lexical and text processing strategies than do less proficient, unsuccessful learners, regardless of whether they use electronic or print dictionaries.
7. Teachers often observe that the more frequently EDs are used in a consistent manner with regular archiving and activation of new word information, and the more systematically new vocabulary is used and reviewed, that retention results are better.

Quality and amount of review techniques or media functions used by a learner largely determine both their *degree of retention* and *speed and percentage of retrieval* of new target terms and language forms. Reaction and retrieval times can be improved by giving *more recent and frequent encounters* with target terms, helping to reactivate them by building further memory traces. Along with recycling and review techniques to improve recognition and prediction skills, *reassessing of learning must be done regularly with frequent individual feedback to maximize motivation and acquisition*. CALL should capitalize on these language learning insights to design maximally efficient vocabulary learning programs for use both online and with portable devices.

When constructing or using online vocabulary learning programs, these same crucial vocabulary learning steps and strategies need to be encouraged by specific questions in text and functions used by the programs. There should also be a tracking or feedback mechanism to help teachers monitor learning, and to guide and prompt learners not to forget to do any of these essential phases of lexical processing.

GENERAL TRENDS AND FUTURE FRUITFUL RESEARCH AREAS

Major benefits of using portable devices include their mobility and instant archiving or storage in history memos for future useful quick review. Web dictionaries like those organized at the author's site, however, provide much more potential, as one can copy and paste between any of over 2000 online lexicons organized there for over 500 language pairs. www.CALL4ALL.us provides a "Virtual Rosetta Stone," not only of the full range of monolingual and multilingual web dictionaries, but also a vast language education links library for studying most of these languages as well.

Another main advantage of modern translation technology is that it is much more efficient. One saves a lot of time, as there is no more turning of book pages and searching for words endlessly. Words you are looking for are at your fingertips, just one click away. Each online dictionary has 400,000 entries, for example, in the case of Ectaco programs, and far more are freely available from web dictionaries organized at www.CALL4ALL.us's dictionaries page at http://www.call4all.us///home/_all.php?fi=d. Recommendations for integrating the use of web dictionaries with language learning programs online are given in Loucky (32). The 10 types of sites are organized to help teachers and students more efficiently combine the benefits of electronic and online dictionaries with CALL websites to produce more effective and enjoyable content and language learning lessons.

The general trends over the past 10 years have been for PEDs to become more prevalent because of their speedy access to language meanings, grammar data, collocations/ corpus examples, compact size, improved features, and convenience of use as well as economical pricing. Some feature as many as 32 lexicons or more, pronunciation support, Internet connectivity, review games, automatic history of searches for review, and so on. Translation software and CD-ROM dictionaries, being more expensive and limited to the location of one's PC, have not become as popular. Web and phone dictionaries appear to be the "tool of choice" of most students, as these functions are often provided at their schools or included in their cell phone services at little or no extra charge. Assistive reading pens made by *Quickionary* also offer promise to those who can afford them. They also seem to enhance learners' interest and motivation levels, and thus help to contribute to higher levels of vocabulary retention, although how to best do so online is a major question in need of further study. Some of the most promising online glossing programs being tested now can be recommended for further research in this area: 1) Wordchamp.com, 2) Rikai.com, 3) Wordsurfing.-com, and 4) Babelfish.com.

CONCLUSIONS AND RECOMMENDATIONS

To conclude, CALL and website e-learning developers need to remember that teachers need to be able to scale their language and vocabulary learning activities from those that require simpler and easier processing for lower level students, to activities that require deeper and more complex lexical processing for more advanced language learners using various kinds of EDs, both online and offline, whether stationary or mobile. It is also important for teachers to give more clear guidance about particular kinds of EDs, especially including good online programs for learning, to help maximize the use of their functions for education. We can only help maximize each program's effectiveness if students learn how to use their various functions as efficiently as possible to help them at each stage of processing new words outlined above. Further helpful guidelines and goals to examine when seeking to integrate new insights and innovations from CALL into the field of foreign language reading and vocabulary development are given by Sokmen (46). In her words, among the many areas in need of further systematic research in this field, "we need to take advantage of the possibilities inherent in computer-assisted learning, especially hypertext linking, and create software which is based on sound principles of vocabulary acquisition theory . . . programs which specialize on a useful corpus. . . provide. . .[for] expanded rehearsal, and engage the learner on deeper levels and in a variety of ways as they practice vocabulary. There is also the fairly unchartered world of the Internet as a source for meaningful activities for the classroom and for the independent learner" (p. 257).

In this way, using proven portable devices, multimedia translation software, and well-designed, interactive websites as much as possible, language learning can be made much more interesting and effective as these CALL resources are all used as tools for developing more balanced communication skills, which emphasize blending active production and interactive, content-based learning with authentic tasks and materials made much more accessible, comprehensible, and memorable with the help of modern technology. All in all, we can be quite optimistic about the future of EDs, as de Schryver (25) is. Listing 118 "lexicographers' dreams" in summarized tables, he masterfully "incorporates almost every speculation ever made about electronic dictionaries (EDs)" (p. 61) in Roby's terms (47).

Roby (47) further notes that not only technical hardware, but also human "fleshware" is the most crucial element when designing EDs, otherwise users may drown in a sea of data. One cannot drink efficiently from a fire hose. As he states, "Sophisticated software and huge hardware cannot guarantee the quality of an electronic dictionary. . . Good online dictionaries will be equipped with 'spigots' that allow users to draw manageable amounts of information. . . Information must be internalized for it to be considered knowledge." In the vast reaches of virtual e-learning cyberspace, one does indeed require a common gold standard compass, or better yet, a virtual Rosetta Stone for language learning, such as those helpful sites provided here.

As second language learners venture into "terra incognita" they do need clear maps and strategies to improve their navigation on various WebQuests for knowledge. Roby (47, p. 63) correctly asserts that "Dictionaries can be guides because they 'potentially intersect with every text of the language: in a sense all texts lead to the dictionary' (quoting Nathan). . . Learners can make forays into cyberspace with an electronic dictionary as a navigational [tool]. And in a real sense, one can expect to see portable, wireless dictionaries that will both allow physical mobility and afford Internet access." (In fact, most mobile phones and WiFi laptops already do).

Tailoring computerized dictionaries to effectively support learners' needs will require specific attention to their types, functions, and uses to best guide learners and teachers to most effective integration of these portable and online tools into language and science education. Research is showing us that all future EDs would do well to include preorganized categories of terms, searchable by topic and semantic field. Five examples of these already found online include: 1) UCREL's Semantic Analysis System located at http://www.comp.lancs.ac.uk/ucrel/usas/ with 21 major A–Z discourse fields; 2) Variation in English Words and Phrases (VIEW) at http://view.byu.edu/; 3) this writer's bilingualized Semantic Field Keyword Approach covering about 2000 intermediate to advanced terms in nine academic disciplines found at: http://www.call4all.us///misc/sfka.php; 4) ThinkMap's Visual Thesaurus at http://www.visualthesaurus.com/index.jsp?vt ; and 5) Wordnet found at http://wordnet.princeton.edu/. This writer's www.CALL4ALL.us site helps to integrate essential, common core vocabulary in many of these academic disciplines with most web dictionaries for 500 major world language pairs. For an overview, see its site map at (http://www.call4all.us///home/_all.php?fi=0) or see Loucky (32,48,49).

In the final analysis, probably what learners are guided to do with new terms will prove to be a more important learning factor than multimedia glossing and text concordancer options alone can provide. New technologies do indeed offer more powerful resources than ever before for independent or classroom study of languages. Word learning options will probably be best maximized when computing power is used to enhance learners' access to various types of EDs of high quality simultaneously in all fields, while likewise providing them with the means to auto-archive and organize new target vocabulary as they are shown how to actively use these new terms productively as soon as possible.

APPENDIX

Survey of Computerized Bilingual Dictionaries (27)

Name your Book Dictionary or Electronic/Computerized Bilingual Dictionary:

Model #: Cost:

NAME: ID/YEAR: Reading Level:

a. Grade:

b. Headwords:

c. %VLS Used:

d. DLP Level:

e. AVQ/IP:

Accessing & Archiving
Time: ___________minutes
(for 15 Laufer & Hadar terms)

1. **Assessing Vocabulary Size:**
   Check your manual to see how many words it has for
   a. English:
   b. Japanese—(or other L1):
   c. Kanji Study—
   d. How many words do you think you know in English?

2. **Accessing—Frequency of Use—How many times do you use it each day?**
   a. For English to Japanese what % of the time?
   b. For Japanese to English, what % of the time?
   c. To check unknown Kanji, what % of the time?

3. **Archiving—How do you record new words found?**
   a. In my textbook in the margins
   b. On paper or in a Vocabulary Notebook
   c. I don't record new words
   d. My CBD can record and save new words I've looked up. If so, tell how:
   e. Can it do Automatic Recording and Review (of last 1–20 words) (called a History Search)
   f. Can you save and store new words manually?
   g. Can you Save and Print Text Files or Notes on new words?

4. **Analyzing Special Functions or Features**—Does your CBD have any Special Functions or Features which help you to break up new words into parts to better understand their grammar, origins or meaning?
   If so, please try to explain how to use them and tell how often you do so. (Use Manual)
   Does it give special information about word parts, grammar, or the origin of words?
   Does it give any common phrases? _____Yes ______No ____Not Sure
   Does it give any sentence examples? ____Yes ____No ____Not Sure

5. **Anchoring New Words in Memory**—Does your Electronic Dictionary have any special Visual Images or Auditory Sounds or other special functions to help illustrate new word meanings, forms or use to help you better remember them? ___Yes _____No
   If so, tell what these special functions are and try to explain how they work to help you fix new words in your memory.

6. **Associating Functions—Does your Electronic Dictionary help you to organize your vocabulary learning in any way?**
   For example, can you put words into Study Groups?
   Do you organize your vocabulary learning or notebook in any special order or way to help you remember new words? Do you group any words together to better remember or learn them? If so, please tell how you do so.

If your computerized dictionary, translation website, or software helps you to do this in any way, please tell how:

7. **Activating Functions—Does your Electronic Dictionary give you any ways to USE new words right away?** ____Yes ____No If so, how?

Can you think of some ways ON YOUR OWN that you could USE new words you have looked up more actively or creatively? If so, tell how:

8. **Review: Do you review any new words after finding their meanings?**

____No ____Sometimes ____Yes, usually If so, tell how does your Electronic Dictionary help you to review or retest new words? Does your ED/CBD have any Vocabulary Practice Games that you can use for review and practice? If so describe. If it had, what level would you start to study at?

Does your CBD have any Special Functions or Features which help you study new words, such as challenge games, memos, word search history, and so on to help you learn, analyze, review or remember new words? ____Yes _____No _____Not Sure If so, please explain how to use them:

FURTHER READING

G. Cumming S. Cropp, and R. Sussex, On-line lexical resources for language learners: assessment of some approaches to word formation, *System*, **22** (3): 369–377, 1994.

J. H. Hulstijn When do foreign-language readers look up the meaning of unfamiliar words? The influence of task and learner variables, *Modern Lang. J.* **77** (2): 139–147, 1993.

BIBLIOGRAPHY

1. F. T. Dolezal and D. R. McCreary, Pedagogical Lexicography Today: A Critical Bibliography on Learners' Dictionaries with Special Emphasis on Language Learners and Dictionary Users. *Lexicographica, Series Maior 96*. Tubingen: Max Niemeyer Verlag, 1999.
2. B. Laufer and M. Kimmel, Bilingualized dictionaries: how learners really use them, *System*, **25**: 361–362, 1997.
3. R. Lew, *Which dictionary for whom? Receptive use of bilingual, monolingual and semi-bilingual dictionaries by Polish learners of English*. Poznan: Motivex, 2004.
4. B. Laufer and T. Levitzky-Aviad, Towards a bilingualized dictionary for second language production. AsiaLEX, Singapore, 2005, pp. 1–6.
5. J. P. Loucky, Assessing the potential of computerized bilingual dictionaries for enhancing English vocabulary learning, in P. N. D. Lewis, (ed.), *The Changing Face of CALL: A Japanese Perspective,*Lisse: Swets & Zeitlinger, 2002, pp. 123–137.
6. J. P. Loucky, Comparing translation software and OCR reading pens. In M. Swanson, D. McMurray, and K. Lane (eds.), *Pan-Asian Conference 3 at 27thInternational Conference of JALT, National Conference Proceedings CD*, Kitakyushu, Japan, 2002, pp. 745–755.
7. J. P. Loucky, Improving access to target vocabulary using computerized bilingual dictionaries, *ReCALL,* **14** (2): 293–312, 2003.
8. J. P. Loucky, Using computerized bilingual dictionaries to help maximize English vocabulary learning at Japanese colleges, *CALICO J.***21**, (1): 105–129, 2003.
9. Y. Akbulut, Exploration of the effects of multimedia annotations on L2 incidental vocabulary learning and reading comprehension of freshman ELT students. Paper presented at EuroCALL, Vienna, Austria, 2004.
10. Y. Akbulut, Factors affecting reading comprehension in a hypermedia environment. Paper presented at EuroCALL, Vienna, Austria, 2004.
11. Y. Akbulut, Foreign language reading through hypermedia: predictors of vocabulary learning and reading comprehension, *6th International Educational Technology Conference*, Famagusta, Northern Cyprus, April 19–21, 2006, pp. 43–50.
12. D. Chun, CALL technologies for L2 reading, in L. Ducate and N. Arnold (eds.), *Calling on CALL: From Theory and Research to New Directions in Foreign Language Teaching,* CALICO Monograph Series, Volume 5, 2006 pp. 69–98.
13. K. Nakao, The state of bilingual lexicography in Japan: learners' English-Japanese/Japanese-English dictionaries, *In. J. Linguist.*, **11** (1): pp. 35–50, 1998.
14. J. Baxter, The dictionary and vocabulary behaviour: a single word or a handful?, *TESOL Quarterly*, **14**: 325–336, 1980.
15. J. Tomaszczyk, On bilingual dictionaries: the case for bilingual dictionaries for foreign language learners, in R. R. K. Hartmann (ed.), *Lexicography: Principles and Practice*, New York: Academic Press, 1983, pp. 41–51.
16. R. R. K. Hartmann, What we (don't) know about the English language learner as a dictionary user: a critical select bibliography, in M. L. Tickoo (ed.), *Learners Dictionaries: State of the Art*, (Anthology Series 23). Singapore: SEAMO Regional Language Centre, 1989, pp. 213–221.
17. T. Piotrowski, Monolingual and bilingual dictionaries: fundamental differences, in M. L. Tickoo (ed.), *Learners' Dictionaries: State of the Art*, Singapore: SEAMO Regional Language Centre, 1989, pp. 72–83.
18. B. T. S. Atkins, and F. E. Knowles, Interim report on the Euralex/AILA research project into dictionary use, in T. Magay and J. Zigány, (eds.), *Budalex '88 proceedings: Papers from the Euralex Third International Congress*, Budapest: Akadémiai Kiado, 1990, pp. 381–392.
19. S. Nuccorini, Monitoring dictionary use, in H. Tommola, K. Varantola, T. Salmi-Tolonen, and J. Schopp (eds.), Euralex '92 Proceedings I-II (Part I), *Studia Translatologica, Series A*, **2**, 89–102, 1992, Tampere, Finland: University of Tampere.
20. J. H. Hulstijn and B. T. S. Atkins, Empirical research on dictionary use in foreign-language learning: survey and discussion, in B. T. S. Atkins, (ed.), *Using dictionaries. Studies of Dictionary Use by Language Learners and Translators, (Lexicographica Series Maior 88.)* Tübingen: Niemeyer, 1998, pp.7–19.
21. B. Laufer and M. Hill, What lexical information do L2 learners select in a CALL dictionary and how does it affect retention?, *Language Learn. Technol.*, **3**, (2): 58–76, 2002. Available: http://llt.msu.edu/.
22. S. Koren, Quality versus convenience: comparison of modern dictionaries from the researcher's, teacher's and learner's points of view, *TESL Electron. J.*, **2** (3): 1–16, 1997.

23. W. J. Meijs, Morphology and word-formation in a machine-readable dictionary: problems and possibilities, *Folia Linguistica*, **24** (1–2): 45–71, 1990.

24. H. Nesi, Electronic dictionaries in second language vocabulary comprehension and acquisition: the state of the art, in U. Heid, S. Event, E. Lehmann, and C. Rohrer (eds.), *Proceedings of the Ninth EURALEX International Congress*, EURALEX 2000, Stuttgart, Germany, Stuttgart: Institut for Maschinelle Sprachverarbeitung, Universität Stuttgart, 2000, pp. 839–841.

25. G-M. de Schryver, Lexicographers' dreams in the electronic-dictionary age, *Int. J. Lexicography*, **16** (2): 143–199, 2003.

26. P. Sharpe, Electronic dictionaries with particular reference to the design of an electronic bilingual dictionary for English-speaking learners of Japanese, *Int. J. Lexicography*, **8** (1): 39–54, 1995.

27. B. Laufer, Electronic dictionaries and incidental vocabulary acquisition: does technology make a difference?, in *Proceedings of the Ninth EURALEX International Congress*, EURALEX 2000, Stuttgart, Germany,U. Heid, S. Evert, E. Lehmann, and C. Rohrer (eds.), Stuttgart: Institut fur Maschinelle Sprachverarbeitung, Universität Stuttgart, 2000, pp. 849–853.

28. Y. Tono, On the effects of different types of electronic dictionary interfaces on L2 learners' reference behaviour in productive/receptive tasks, in U. Heid, S. Evert, E. Lehmann, and C. Rohrer (eds.), *EURALEX 2000 Proceedings*, Stuttgart, Germany, 2000, pp. 855–861.

29. J. P. Loucky, Maximizing vocabulary development by systematically using a depth of lexical processing taxonomy, CALL resources, and effective strategies, *CALICO J.*, **23**, (2): 363–399, 2006.

30. B. Laufer and L. Hadar, Assessing the effectiveness of monolingual, bilingual, and "bilingualized" dictionaries in the comprehension and production of new words, *Modern Lang. J.*, **81**: 189–196, 1997.

31. C. A. Grace, Retention of word meaning inferred from context and sentence level translations: implications for the design of beginning level CALL software, *Modern Lang. J.*, **82** (4): 533–544, 1998.

32. J. P. Loucky, Combining the benefits of electronic and online dictionaries with CALL Web sites to produce effective and enjoyable vocabulary and language learning lessons, *Comp. Assisted Lang. Learning*, **18**, (5): pp. 389–416, 2005.

33. C. Kobayashi, Pocket electronic versus printed dictionaries: the effects of their use on lexical processing strategies, On JALT 2004: Language Learning for Life Conference CD, K. Bradford-Watts, C. Ikeuchi, and M. Swanson (eds.). *JALT 2004 Conference Proceedings.*Tokyo: JALT, 2005, pp. 395–415.

34. A. Taylor and A. Chan, Pocket electronic dictionaries and their use, in W. Martin et al. (eds.), *Euralex 1994 Proceedings* Amsterdam: Vrije Universiteit, 1994, pp. 598–605.

35. G. M. Tang, Pocket electronic dictionaries for second language learning: help or hindrance?, *TESL Canada J.*, **15**: 39–57, 1997.

36. H. Nesi, A user's guide to electronic dictionaries for language learners, *Int. J. Lexicography*, **12** (1): 55–66, 1999.

37. H. Nesi and G. Leech, Moving towards perfection: the learners' (electronic) dictionary of the future, in H. Thomas and P. Kerstin (eds.), *The Perfect Learners' Dictionary?*, Tübingen: Max Niemeyer Verlag, 1999, pp. 295–306.

38. T. Koyama and O. Takeuchi, Comparing electronic and printed dictionaries: how the difference affected EFL learning, *JACET Bull.*, **38**: 33–46, 2004.

39. C. Kobayashi, Examining the effects of using pocket electronic versus printed dictionaries on lexical processing strategies. Handout at JALT National Convention, Nara, 2004.

40. N. Schmitt, *Vocabulary: Description, Acquisition and Pedagogy*, Cambridge:Cambridge University Press, 1997, pp. 200–203.

41. Y. Kudo, L2 vocabulary learning strategies. Available: http://www.nrc.hawaii.edu/networks/NW14/NW14.pd.

42. M. Orita, Vocabulary learning strategies of Japanese EFL learners: their actual use and perception of usefulness, in M. L. Kyuoki (ed.), *JACET Annual Review of English Learning and Teaching*, **8**: 27–41, 2003, Miyazaki, Japan: Miyazaki University.

43. I. Kojic-Sabo and P. Lightbown, Student approaches to vocabulary learning and their relationship to success, *Modern Lang. J.*, **83** (2): 176–192, 1999.

44. B. Laufer and M. Hill, What lexical information do L2 learners select in a call dictionary and how does it affect word retention?, *Lang. Learn. Technol.*, **3** (2): 58–76, 2000.

45. M. Hill and B. Laufer, Type of task, time-on-task and electronic dictionaries in incidental vocabulary acquisition, *Int. Rev. Applied Linguist.*, **41** (2): 87–106, 2003.

46. A. Sokmen, Current trends in teaching second language vocabulary, in N. Schmitt and M. McCarthy (eds.), *Vocabulary: Description, Acquisition and Pedagogy*, Cambridge: Cambridge University Press, 1997, pp. 237–257.

47. W. B. Roby, The internet, autonomy, and lexicography: a convergence?, *Melanges CRAPEL*, No. 28. Centre de Recherche et d'Applications Pédagogiques En Langues, Publications Scientifiques, 2006.

48. J. P. Loucky, Harvesting CALL websites for enjoyable and effective language learning, in *The Proceedings of JALT CALL 2005, Glocalization: Bringing people together*, Ritsumeikan University, BKC Campus, Shiga, Japan, June 3–5, 2005, pp. 18–22.

49. J. P. Loucky, Developing integrated online English courses for enjoyable reading and effective vocabulary learning, in *The Proceedings of JALT CALL 2005, Glocalization: Bringing People Together,* Ritsumeikan University, BKC Campus, Shiga, Japan, June 3–5, 2005, pp. 165–169.

John Paul Loucky
Seinan JoGakun University
Fukuokaken, Japan

ELECTRONIC WARFARE

INTRODUCTION

Over the last century, there has been a burgeoning use of the electromagnetic (EM) spectrum for military purposes, including those related to communications, navigation, and targeting.

This dependence is embedded in many modern warfare doctrines and technologies, such as:

- Revolution in military affairs;
- Network-centric warfare;
- Information warfare;
- Rapid decisive operations;
- Intelligence, surveillance, target acquisition, and reconnaissance;
- Precision guided weapons.

Given the importance of the EM environment to military operations, there is obvious reason for safeguarding its use by friendly forces, denying its use by enemy forces, and defeating enemy efforts to achieve the same objectives. Electronic warfare (EW) encompasses the broad and somewhat ill-defined mix of military tactics, techniques, procedures, technology, and organizational structures that address these concerns (1, 2). It is also related to some civilian technologies and applications, which include spectrum monitoring and radio astronomy.

Historical experience has repeatedly demonstrated the importance of EW as highlighted by an extensive body of declassified information that pertains to operations by both sides in World War II (WW2)(3–5), and by more recent accounts concerning the Korean, Vietnam, Six-Day and Yom Kippur Wars, and the campaigns in the Falklands, Lebanon, Kosovo, Chechnya, and Iraq (6–11).

EW continues to be widely recognized as a powerful force multiplier, and the development and application of EW concepts and technologies consequently remains a high priority (12,13). For the greatest effect, its use is regulated by planning structures that tailor it to situational requirements and procedures intended to deny the enemy as much knowledge as possible relating to its specific capabilities and deployment structures. For this reason, many aspects of EW are highly classified.

Formally, the roles of EW are subdivided into:

1. Electronic support (ES) - taking advantage of signals emitted by an opponent's systems;
2. Electronic attack (EA) - degrading the ability of an opponent to use his systems;
3. Electronic protection (EP) - safeguarding the effective operation of friendly force electronic systems against enemy EA and ES activities.

The following article presents a breakdown of EW in this order, with attention given to both technical system concepts and relevant operational doctrine.

ELECTRONIC SUPPORT

ES, which is also known as electronic support measures, concerns the sensing of communication, radar, and other electromagnetic signals of potential interest. ES sensors perform the following technical functions:

1. Signal detection - determining the presence of a signal;
2. Signal classification - associating the signal with a type of modulation or function;
3. Signal parameter and feature extraction - measuring various signal parameters; such as carrier frequency, power, transmission start and end times, and bandwidth;
4. Emitter identification - determining the type of system that the signal is associated with;
5. Signal intercept - recovering the message content from communication signals;
6. EW analysis - inferring the organization and structure of enemy networks, dispositions of forces and operational intent from communications traffic patterns and message content;
7. Geo-location - determining the positions of signal emitters.

Several points concerning ES deserve emphasis. First, its passive nature has the great advantage that valuable intelligence can be produced without an adversary being aware. Second, the mere suspicion of its use can cause an adversary to restrict its use of communication systems and active sensors, which thereby reduces their operational value. Finally, radar ES systems often can detect a radar transmitter at ranges considerably in excess of the useful range of the radar (14).

The organization and processing of information provided by ES sensors is a complex problem. Much value of ES sensor outputs can be lost if information does not reach the appropriate commanders and other potential users in a timely way. Complicating factors include the volume of information, the difficulty of interpreting it, and the need to protect sensitive information concerning ES capabilities. The last point is a very real concern. During WW2, the decryption of German communication signals coded with the Enigma cipher provided immensely valuable intelligence to the British. Accordingly, every effort was made to avoid arousing suspicions that the Enigma cipher was anything other than unbreakable. For example, reconnaissance aircraft would be dispatched to "find" an important convoy whose orders had in fact been revealed by the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

decryption of Enigma messages, which thereby gave the impression that the attack that followed was the direct result of routine aerial reconnaissance (5).

The diversity of the roles performed by ES systems has resulted in a significant degree of specialization in the design of the systems themselves and their organization and control.

Tactical ES

Tactical ES is the deployment of an ES capability in direct support of field operations. It typically resides within some form of dedicated EW unit that may be either part of the maneuver force's echelon or assigned to support it under an operational (OPCON) or tactical (TACON) command and control relationship. Examples of tactical ES are found in land, air, and sea operational environments, where objectives include:

1. The intercept, direction finding, and analysis of battlefield communications signals by ground-based assets to determine the composition and geographical distribution of enemy forces and the immediate intentions of its elements, from fighter to commander. When ES is performed by an EW unit native to the maneuver force, "intentions and warnings" tip-offs are reported directly to field unit commanders and their staff. The unit may also acquire and disseminate intelligence for consumption strictly within Signals Intelligence (SIGINT) channels (see below) and generate technical information for internal process refinement;
2. The detection and direction finding of battlefield surveillance radars by ground-based radar ES;
3. The detection and analysis by a radar warning receiver (RWR) of radar signals associated with enemy target acquisition, tracking, and fire control systems, to provide aircraft pilots with situational awareness and warnings of threats. This information is essential for the timely initiation of suitable countermeasures, which may include a combination of EA and evasive maneuvers;
4. A general surveillance capability by a warship's radar ES systems to track military, merchant, or clandestine ships and fishing vessels using the signals received from their navigation radars. These systems also support self-protection functions against radars associated with threat weapon systems. On larger platforms, there are usually more provisions for analyzing ES information, fusing it with other intelligence, and distributing it to other platforms, channels and organizations (including SIGINT).

The capability to geo-locate transmitters associated with communication, navigation, and radar systems is particularly important; even approximate indications of the direction of an enemy position or platform provided by direction finding (DF) are valuable from a situational-awareness perspective. Estimates of the positions of individual emitters can be determined by obtaining lines-of-bearing from spatially separated sites and solving for the positions where they intersect. Geo-location is particularly important for communication signals when the message content cannot be extracted because of encryption or other techniques. Appendix 1 provides an overview of various DF techniques that can be used for the geo-location of signal sources by ES systems.

An additional EW analysis (EWA) capability is often associated with units that deploy ES assets. EWA is a military intelligence function that specializes in drawing operational inferences from EW data. Its main purpose is to determine the enemy's "electronic order of battle," which is a comprehensive representation of its electronics systems, including their identification, geographical disposition, and where possible the association of this equipment with specific units within a command-control structure. An EWA cell may also be responsible for maintaining communication target lists and selecting information for dissemination to Intelligence organizations.

Tactical communications ES is a particularly challenging problem in urban environments. Multipath propagation effects can be expected to degrade the accuracy of radio-frequency direction-finding systems. Furthermore, opposition forces can be expected to make use of the civilian communications infrastructure, which results in a requirement to sift rapidly through a large amount of communications traffic to find the signals of interest.

Signals Intelligence

SIGINT is the strategic application of ES performed under the control of national intelligence organizations, such as the National Security Agency in the U.S., and the Government Communication Headquarters in the U.K. The term relates variously to the type of information produced, the systems used to produce it, and to the community that controls the ES systems and the analysis and distribution of their products. SIGINT "products" are disseminated via highly classified channels and, except in exceptional circumstances, are released only for use in the wider national or Military Intelligence communities after being "sanitized" of any distinguishing elements that could reveal the source. On the battlefield, there may be some overlap between SIGINT and tactical ES activities and platforms, with EW units sometimes tasked to serve both functions simultaneously.

SIGINT comprises communications intelligence (COMINT) and electronic intelligence (ELINT). COMINT is concerned with the message content of communication signals, information about communication traffic patterns, and the locations of the associated transmitters, with a strong emphasis on determining higher-level or "strategic" command and control structures. ELINT is the collection of technical or "parametric" information about the radar and other noncommunications equipment (15).

ELINT has several important uses. First, theoretical analysis of the signal parameters allows inferences to be drawn about the functions, capabilities, and limitations of the systems associated with the signals, and hence, more broadly, about enemy early warning or targeting capabilities. Second, ELINT data are used to construct emitter libraries or databases that are fundamental to EA and EP operations. For each known type of radar, information is collected on the signal parameters for the various operating modes, the estimated radar performance, its intended function(s), and the platforms the radar is known to be installed on. An ES system on a ship or tactical aircraft correlates the parameters of observed signals with the database entries to identify the radar systems that transmitted them, and, if an observed signal is associated with a threat, it provides the information needed to select and execute the most appropriate countermeasures.

SIGINT operations often involve the use of specialized equipment deployed on either dedicated or multiuse platforms, which include satellites, ships, and aircraft. During the Cold War, suitable types of aircraft were extensively modified to perform SIGINT. By operating at altitudes of 10 km or higher, useful ranges could be extended to hundreds of km for the intercept of microwave radar signals. Consequently, intelligence could be acquired from aircraft flying at the periphery of the Soviet defense perimeter. For a period, specialized high-altitude aircraft could even conduct operations over Soviet territory by flying above the effective ceiling of interceptor aircraft and ground based antiaircraft weapons. After improved Soviet antiaircraft defenses made overflights impractical, the West hurriedly deployed satellite-based systems (16).

In recent years, much interest has been aroused by the idea of integrating ES information derived at different levels (tactical, operational, and strategic) by EW and SIGINT units with similar objectives, but possibly different reporting mechanisms. For instance, modern strategies for Netcentric Warfare involve the accumulation of various kinds of data and intelligence at a central point where it can be fused to produce more complete assessments. However, many practical challenges exist in reconciling technical possibilities with doctrine. Complicating factors and risks involved with centralized analysis schemes include:

1. The quantity of data generated by advanced ES systems may tax the analysis systems that must sort through it;
2. Delays in the reporting chain, where key information may take longer to reach its ultimate destination after passing through a central accumulation point;
3. The expense and complexity of deploying communication systems with adequate bandwidth;
4. Standardization issues for technical interfaces, and the complexity of both designing and maintaining interfaces for systems that were originally designed for different purposes and may be based on widely differing technologies;
5. Complications that affect the handling and distribution of information resulting from classification issues and, in the case of multinational environments, the willingness of individual nations to declare and release their information to others;
6. The risks of commanders relying too heavily on the formation of a “complete intelligence picture” in lieu of trusting their judgment and intuition, which can lead to decision paralysis.

ES System Technologies and Implementation

ES systems are typically comprised of antenna, receiver, and processing sub-systems. Early ES systems were often improvisations based on civilian equipment. For example, receivers developed for radio amateurs had relatively good sensitivity and frequency coverage and were widely used by the Allies during WW2. The National HRO, which had excellent frequency resolution, was used to intercept communication signals in the medium and high frequency bands. The Hallicrafters S-27, which provided contiguous coverage in the lower portion of the very high frequency (VHF) band, was widely used to receive signals associated with German VHF radar, air-to-air communication, and bombing navigation systems. These receivers, although useful, had significant limitations. Their frequency coverage was limited, and their effectiveness was heavily dependent on the training and skill of the operators.

The continued evolution of the technologies used by communication and radar systems has contributed to the development of specialized ES receivers. A fundamental issue concerns the differences in the waveforms used by communication and radar signals.

Most communication systems transmit a continuous or near-continuous narrow bandwidth signal during the transmission of a message. A primary goal is to make efficient use of bandwidth to transmit information, which thereby allows the available radio frequency bands to be divided between many users. Communication signals have continued to evolve:

1. The bandwidth and channel spacing associated with conventional narrowband signals has decreased because of developments in more efficient modulation formats and accurate frequency references and synthesizers;
2. Digital modulation techniques are being increasingly used to transmit information in the form of binary data;
3. Time division multiplexing access techniques are being used by some systems, such as those based on the GSM cell phone standard, to provide a way of time sharing bandwidth between multiple users;
4. Classes of spread-spectrum techniques are being used in some military and civilian communication systems. Frequency hopping (FH) systems superpose periodic changes on the center frequency of a transmitted signal following a predetermined sequence. These changes typically occur at rates that are tens or hundreds of times per second. The portion of a transmission that corresponds to a dwell at a single frequency is often referred to as a hop. To minimize interference between FH communication systems, careful coordination is needed in the assignment of hop frequencies and/or the codes that define the hop sequences. Direct sequence spread spectrum (DSSS) uses a different approach. In the basic form, a pseudo-random number (PRN) sequence is used by the transmitter to spread the narrowband information content over a much larger bandwidth. The receiver uses the same PRN sequence to recover the information. Multiple systems

can share the same bandwidth without seriously interfering with each other if they are assigned different PRN sequences. Code division multiple access (CDMA) cell phone systems are a major application of DSSS techniques. Because the detection of spread-spectrum signals often requires special techniques (17), these signals are sometimes referred to as low probability of intercept signals.

5. Mobile communication systems and networks have proliferated and are widely used. These systems are based on the idea of dividing a geographical area into cells. Each cell has a base station that performs the functions of relaying messages between the short-range handset radios within the cell and a communication network interface to other carriers, such as the public telephone system network. Cellular telephone systems usually operate in the ultra high frequency band.

The classic pulsed-radar concept, however, involves the transmission of short duration pulses with relatively large time intervals between successive pulses. This method sidesteps the difficult problem of detecting the relatively weak signals reflected from the target during the simultaneous transmission of a high power signal. Requirements for range resolution often dictate the use of pulse widths on the order of a microsecond or less, which thereby results in relatively large bandwidths on the order of MHz. The waveforms used by advanced radars have increased in sophistication:

1. Coherent radars transmit signals whose waveforms are precisely defined;
2. Frequency or phase modulation may be used to increase range resolution;
3. The time intervals between successive pulses (pulse repetition interval) may be varied in a periodic or random sequence (pulse repetition interval stagger);
4. Multifunction radars select between different waveforms depending on the functionality that is required[1];

Application requirements for high angular resolution and compact antenna dimensions have motivated the extensive use of frequencies above 8 GHz.

The differences between radar and communication signals have motivated the development of specialized ES equipment:

1. Communication ES receivers feature extended frequency coverage to reduce the need to use different receivers, selective filters for separating signals that are closely spaced in frequency, comprehensive capabilities for demodulating the signal message content, and provisions for the measurement of signal parameters;
2. Radar ES receivers emphasize microwave frequency coverage and are optimized for the reception of pulse signals;
3. Specialized radar ES receivers have been developed for strategic and tactical applications. For example, electronic intelligence receivers are designed for the precision measurement of signal parameters, whereas radar warning receivers are designed to provide warnings of threat signals, be simple to use, and satisfy size and cost constraints;
4. Multichannel receivers have been developed to process multiple signals from antenna arrays with the accurate phase and amplitude matching needed for applications such as direction finding.

General trends in all systems include the use of precision frequency references and synthesizers to permit accurate and repeatable tuning, progressive reductions in size, and the use of form factors that permit the convenient installation of multiple receivers in standardized rack configurations.

Communication ES Signal Processing. The classic communication ES receiver implementation is basically a high-quality manually controlled superheterodyne receiver. Signal search was performed by the operator manually tuning the receiver through the frequency range known to be used by the adversary's radios and listening to the outputs of the available demodulator(s) for signals of interest. When such a signal was found, the operator would listen to the demodulated signal and record the observations. If available, a DF system would be tuned to the frequency and measurements obtained for the signal angle of arrival. This process required the attention of a skilled operator and had the additional weakness that short duration transmissions on new frequencies could be missed, particularly if the frequency ranges to be covered could not be divided up among multiple systems and operators. Another weakness concerned the size, weight, and power consumption of the equipment.

Modern purpose-designed communication EW receivers provide significant enhancements:

1. Computer controlled operation via standard digital interfaces;
2. Accurate high-speed tuning and reduced phase noise that results from the use of high-quality crystal oscillators as frequency references and sophisticated frequency synthesis techniques;
3. Provisions for phase coherent operation of multiple receivers to allow commonality of hardware between systems used for signal search and DF;
4. Built-in-test functionality;
5. Reduced size, weight, and power consumption.

Digital signal processing techniques are being adopted for advanced ES systems. Digital filter bank concepts based on the Fast Fourier Transform algorithm allow a single wideband receiver to process and detect the individual

[1] For example, the optimal waveforms for discriminating between a moving target on the ground and the surrounding terrain would be unsuitable for providing extreme range resolution.

signals present within a large instantaneous bandwidth. Also, if the system dwells on a fixed center frequency, digital downconverters can be used to extract the narrowband signals within the receiver bandwidth and software demodulators used to recover the message content from each signal.

Advanced wideband communication ES sensors based on digital filter bank techniques have some very desirable advantages:

1. A large frequency range can be scanned quickly; the tuning frequency step size can be orders of magnitude larger than the required frequency resolution. This method substantially reduces or eliminates the likelihood that a short duration transmission will be missed and can provide some capability for detecting at least some hops transmitted by a frequency hopping radio;
2. The use of Constant False Alarm Rate techniques allows the system detection processing parameters to be adjusted automatically to achieve the best possible sensitivity without incurring erroneous signal detections at a rate that exceeds a set value, even if the environmental noise is frequency dependent and time variant (18);
3. Algorithms can be implemented to determine the type of modulation used by a signal and the modulation parameters;
4. Raw signal data can be acquired and stored for off-line analysis;
5. Demodulators implemented in software can accommodate a wide range of modulation types;
6. DF functionality can be integrated into the system to provide a measurement of the angle of arrival for each signal that is detected;
7. Reports of signal detections and the measured signal parameters can be automatically stored in a database and transferred to EW analysis and intelligence systems for subsequent processing;
8. Remote controlled or autonomous operation of ES systems is feasible.

However, wideband signal processing techniques also incur disadvantages. Early implementations tended to be expensive and have significant performance limitations. A major problem concerns dynamic range, which is a measure of the ability of a system to process strong and weak signals simultaneously. This issue is of considerable importance for wideband communications ES systems because weak signals of interest and strong signals will often coexist in the same frequency range. The dynamic range of a practical system is dependent on the noise and spurious signals, which are generated in the system by various mechanisms. One of the most important of these mechanisms is third order intermodulation distortion. This occurs when two or more signals present within the system bandwidth interact because of nonlinearities in the system signal processing. The spurious signals that result remain within the system bandwidth and, depending on the size of the input signals and the nature of the system nonlinearities, can be large enough to be detected and interpreted as actual signals in subsequent processing. To avoid this undesirable result, the detection processing must be adjusted to reduce the effective system sensitivity. Thus, the presence of strong input signals tends to degrade the ability of the system to detect and process weak signals usefully. The problem is aggravated as the system bandwidth is increased because the number of strong signals within the system bandwidth can also be expected to increase. Fortunately, progressive advances in radio frequency components, analog-to-digital converters, and digital processor hardware have substantially resolved these issues, particularly when careful system design choices and tradeoffs are made. Nevertheless, a well-designed narrowband receiver may still offer advantages with respect to usable sensitivity and selectivity in a dense signal environment that includes strong signals.

In addition to its message content, a communication signal contains information that can be used to classify the type of signal, and, with some limitations, to identify individual emitters.

The measurement of the modulation type and parameters is an important topic for communications ES systems. Conventional communication systems use modulation techniques to embed information on a sinusoidal carrier signal. The choice of modulation type and implementation parameters is dependent on application requirements and various factors, such as the need for interoperability with other radio systems as well as technology and cost constraints. Advances in communication theory coupled with the availability of low-cost digital signal processing hardware have motivated the use of sophisticated digital modulation techniques to provide favorable trade-offs between bandwidth efficiency, sensitivity to propagation effects, and hardware implementation costs. At the same time, simple, classic modulation techniques, such as analog frequency modulation, remain in widespread use, in part to maintain interoperability with older systems.

Knowledge of the modulation type and parameters associated with a signal is of considerable practical value. Requirements for interoperability have led to the standardization of the modulation types used by military radios. For example, the tactical VHF radios used in ground operations typically support analog FM and digital FSK modulations in accordance with standards such as MIL-STD-188-242. If a signal has a modulation type and parameters associated with a communication system known to be used by an adversary, then it can be flagged as a potential signal of interest and prioritized to receive attention. Also, because emitters that are communicating with each other will generally use the same modulation type, this knowledge can be used to support or reject hypotheses that concern the membership of a given emitter in a network. Finally, knowledge of the modulation type and parameters facilitates the selection of an appropriate demodulation technique to recover the message content.

Because of the diversity of modulation standards and the effects of multipath propagation and nonideal radio system implementations, the modulation recognition problem is nontrivial. Algorithms for modulation recognition have been described in various papers, of which Refs. 19–22 are representative examples.

A related idea is based on the observation that the signal waveforms generated by practical radio transmitters will differ in subtle ways depending on implementation details and component tolerances, and that these differences can be sufficient to distinguish between transmitters that are very similar or even nominally identical. Various techniques have also been proposed to extract and measure appropriately selected features from a signal and use statistical tests to determine whether the feature measurements match those of previously observed signals (23, 24).

Radar ES Signal Processing. Various analog and digital approaches have been used in radar ES receivers to detect signals and measure their parameters. Descriptions and performance analyses of the more common ones have been published 25–27. The radar ES receivers used for current radar ES systems deployed for the self-protection of platforms such as aircraft and surface ships generate pulse descriptor words (PDWs) for each radar pulse that is received. Each PDW consists of digital data that represents the principal signal parameters, typically frequency, power, time of arrival, pulse duration, and if available, angle of arrival and modulation type (phase or frequency). Early implementations made extensive use of analog techniques to generate PDWs, but more recent implementations are making increasingly extensive use of digital techniques.

Pulse train deinterleaving is required because the pulses that are received from the various radars in the signal environment will be interleaved in time (i.e., in a sequence of received radar pulses there is no certainty that for a given pulse in the sequence, the previous or next pulses in the sequence will be from the same radar). Deinterleaving is typically performed in a two-stage process. First, clustering is performed as pulses are received to form clusters or groups of pulses having similar characteristics. A subset of the signal parameters contained in the PDWs, typically frequency, angle of arrival, and pulse duration, are used in this stage. The second stage involves analyzing the time relationships [Pulse Repetition Interval (PRI) deinterleaving] between the pulses collected in each cluster to identify patterns that are consistent with the hypothesis that they were transmitted by a single radar. In addition to the radar PRI behavior, the radar scan pattern can be inferred by examining the time history of the measured power of received pulses in a deinterleaved pulse train. For example, a radar that is performing a circular scan will illuminate the platform carrying the ES system with its main beam response at uniform intervals in time.

Emitter identification involves comparing the various parameters that have been measured for each of the resultant deinterleaved pulse trains with those in an EW library and identifying the best match.

In practice, many potential difficulties may occur. The PDWs generated by the receiver will contain errors that result from various sources. At least some clusters formed in the first stage will have broad ranges. For example, a large frequency range may be needed to accommodate a frequency agile radar. Consequently, some clusters may overlap. Accurate PRI deinterleaving can be very difficult to perform with limited signal data sets; many modern radars have complex PRI staggers (i.e., the time intervals between successive pulses transmitted by a radar vary randomly or follow patterns that repeat only over a long period). Deinterleaving errors can result in the pulse train transmitted by such a radar being fragmented into two or more partial pulse trains. Finally, EW databases can have errors, be incomplete, or as a result of ambiguities, may be unable to provide a unique identification.

More sophisticated approaches are being investigated for the extraction of features that can be used to provide additional information for the classification and identification of radar signals. For radars that use frequency or phase modulation to improve range resolution, knowledge of the type of modulation waveform and its parameters is useful for classification purposes. Also, the waveforms transmitted by radar systems often have distinctive features, which are sometimes referred to as unintentional modulation on pulse (UMOP). Various techniques have been proposed for the extraction and processing of waveform features for signal identification.

ELECTRONIC ATTACK

EA, which is also known as Electronic Countermeasures, involves actions intended to degrade the ability of an adversary to make use of the electromagnetic spectrum. It may be active or passive in nature.

EA Against Communication Signals

EA against communication signals can be carried out as deception operations or jamming.

Deception operations involve the transmission of signals to mislead the enemy intentionally. For example, after a ground formation has been redeployed for operations elsewhere, simulated radio traffic may be maintained to give the impression that the formation is still in its original location. Another technique involves the transmission of messages that contain misleading information in the expectation that the message content will be recovered and used by the adversary. Deception operations must be carefully designed and organized to be convincing; the information provided should be consistent with other information that the intended recipient believes to be true. Large-scale deception operations that involve carefully coordinated activities can influence an adversary's strategic planning with decisive effect. Several accounts of highly successful Allied deception operations in WW2 have been published (5, 28).

Jamming is intended to prevent an adversary from reliably receiving his communication signals by the transmission of signals that interfere with their reception. In the simplest form, a jammer consists of an antenna, power amplifier, and signal generator programmed to produce a signal with an appropriately chosen waveform. It is also possible to use a conventional transmitter or radio as an improvised jammer. Jamming systems are often deployed with an adjunct ES capability to ascertain the frequencies of signals worth jamming and to assess the effects of the jamming operation.

To be effective, jamming requires that the ratio of jammer and communication signal powers (J/S ratio) at the victim radio receiver be sufficient to degrade communication activity adequately. High-power transmitters may be used in combination with directional antennas and the judicious positioning of the jammer near the area where jamming coverage is desired.

Several distinct types of communication jamming techniques are as follows:

Narrowband Jamming. Individual communication signals can be attacked by transmitting an appropriately designed narrowband-jamming signal on the frequency used by the target signal. To determine whether the target signal is still being transmitted, the jamming may be periodically stopped and an ES capability used to check for the presence of the signal. This method of attack has several advantages. First, the jamming range is maximized because the full jamming power is focused on a single signal. Second, the likelihood of interference with own side communication is minimized because only a small part of the radio spectrum is affected. If the jamming signal can be switched rapidly between frequencies, then a single transmitter may be able to jam two or more narrowband signals on a time shared basis.

A follower jammer is a special case of narrowband jammer used to jam a FH signal. The practical implementation of the concept is challenging; each hop transmission must be detected, its frequency measured by the ES functionality integrated with the jammer and, before more than a fraction of the hop is transmitted, the jamming transmitter must be tuned to the hop frequency (29). One difficulty is that the jammer must discriminate reliably between the hops from the target transmitter and any other frequency hopping communication systems that may be operating in the environment. A more fundamental issue concerns the propagation delays associated with, first, the path from the transmitter to the jammer, and, second, the path from the jammer to the victim receiver. If the end result is that the overall delay, including the jammer response time, approaches the hop duration, then the effectiveness of the jamming will be degraded.[2]

Barrage Jamming. A wideband jamming signal is used to degrade communication activities over a relatively wide range of frequencies. A high-power jammer may be needed to provide a useful range. A partial-band jammer is a variation on the barrage jammer concept. The aim is to jam a bandwidth that is sufficiently large enough to include a sufficient proportion of the hops transmitted by a FH radio to make it unusable. The idea is that, by not attempting to jam the full bandwidth used by the frequency hopping radio, the jammer power within the hop bandwidth can be kept higher and can provide an increase in the effective range of the jammer.

Many issues must be considered with respect to communication jamming:

1. Jamming often interferes with own side communication;
2. The value of information that is obtained by ES may be considered to be of greater military value than the effect of disrupting communication;
3. An adversary can infer the presence of enemy forces with EW capabilities from the observation of jamming signals and, if given time, may find ways of countering its effects.

Consequently, aside from some specialized applications, the decision to carry out communication jamming is usually made at a relatively high level and is closely coordinated with operational plans.

The deployment of communications jammers on aircraft provides several advantages. The jammer is mobile and can be positioned quickly to affect the desired area while minimizing the effect on friendly forces. Also, the required transmitter power can be reduced because, for a given range, the propagation losses are normally much lower than they would be for the signals from a ground based jammer. Recently, serious interest has been expressed in the idea of using low-power communications jammers on small unmanned air vehicles (UAVs) to provide localized jamming coverage in the direct support of small-unit operations (30).

EA Against Radar Signals

EA against radar signals is often concerned with degrading the performance of surveillance, target acquisition, and target tracking radars to protect platforms such as aircraft and surface ships. The value of these platforms and the potential effectiveness of radar-guided weapons has led to much emphasis being placed on EA.

Active EA techniques are used to create false targets or otherwise degrade the operation of the victim radar:

1. A noise jammer transmits wideband noise in the frequency ranges used by radar systems of potential concern, which makes it difficult for the radar to detect the target and get a range measurement;
2. A range gate pull-off jammer attempts to create a false target that seems to move away from the jammer platform. The jammer first creates a false target at the jammer platform by transmitting a pulse timed to coincide with the arrival of each pulse transmitted by the victim radar. The timing of successive pulses is gradually shifted so that the jammer pulses received by the victim radar correspond to a target that is moving away from the jammer platform. The digital radio frequency memory (DRFM) improves the technique by storing and transmitting a replica of the radar-pulse waveform. This method makes it more difficult for the radar to discriminate against the jammer signal.

[2] This problem can be avoided if the hop frequency sequence can be predicted using observations of the hop frequencies and *a priori* knowledge of the algorithm used to generate the hop sequence.

Several practical problems are noted in the deployment of jammers. The operation of jammers used for the self-protection of platforms, such as aircraft, is usually restricted to the jamming of threat signals as required. This method minimizes several risks, which include the possibility of interference with other systems on the platform, and that the presence of the platform can be inferred by the detection and direction finding of signals transmitted by the jammer. In this situation, an integrated ES capability for performing the detection, characterization, and assessment of threat signals is required to provide information needed for the control of the jammer. One way of sidestepping this issue is to deploy jammers on specialized platforms, and if possible to perform the jamming outside the defended air space. Other solutions include the towing of jammers behind the platform to be protected, or deploying jammers on UAVs.

Passive EA techniques attempt to degrade the effectiveness of enemy radars without transmitting signals. A widely used idea is to create false targets by dropping chaff (typically metal coated plastic strips) from aircraft to confuse tracking radars associated with antiaircraft defense systems. Chaff can also be dispersed via rockets or shells fired from platforms such as ships as a countermeasure to radar-guided missiles. Another approach is to tow decoys behind an aircraft or ship. The use of passive EA to confuse the guidance systems of antiaircraft or antiship missiles is often combined with maneuvers designed to position the platform to minimize the likelihood that the missile-guidance system will reacquire its target or that the missile will fortuitously pass near its target. Another form of passive EA concerns the use of stealth techniques to reduce the reflected energy returned to a radar transmitter by a platform (i.e., reduce the apparent radar cross section of the platform). The effectiveness of this technique is increased if combined with active EA from other platforms.

Other forms of EA are also important. Radar systems can be destroyed by missiles designed to home in on the signals transmitted by the radar. Conventional military operations against deployed systems identified by EW sensors or other intelligence are also possible. Recently, the concept of using directed energy or electromagnetic pulse (EMP) to damage or disrupt the operation of electronic equipment has received attention.

ELECTRONIC PROTECTION

Electronic protection, also known as electronic-counter-counter measures, concerns techniques and technologies intended to preserve the ability of defense electronic systems to operate in hostile electromagnetic environments.

Active EP includes measures taken to enhance the ability of defense electronic equipment to operate without hindrance by enemy EW.

Protection against intercept and jamming of communication signals can be provided in various ways:

1. Equipment can be designed to operate over wide frequency ranges, which offers improved opportunities for a system to switch to quieter frequencies if interference or jamming is encountered;
2. Directional antennas can be employed to make the interception of a signal difficult for a receiver outside the main beam response of the transmitting antenna. Jamming resistance can be achieved if the direction that the jamming signal is coming from corresponds to a null in the receiving antenna directional response.
3. Careful choices of sites may be able to take advantage of terrain masking of areas potentially usable by jammers or ES systems;
4. Power management allows the transmitter power to be set at the minimum level required for reliable communication. Low-power operation is desirable for short-range communication because the range at which the signal can be detected and intercepted is reduced. High power levels can be used to provide reliable operation over longer ranges and/or to overcome jamming;
5. Low probability of intercept techniques can be used to render DF and intercept difficult. FH techniques are widely used by modern tactical radios;
6. Redundancy can be achieved by design and/or tactical procedures to limit the damage caused by the effects of enemy EA; for example, different types of communication systems can be networked and managed to ensure that the disruption of one system does not prevent the communication of important information.

Similar techniques are applicable to radar systems with several differences:

1. A radar system may be able to search over a restricted range of angles and still perform its mission requirements. An ES system outside the search area will not be illuminated by the mainbeam of the radar antenna and may have difficulty detecting the signals;
2. Radar antennas are generally designed to be highly directive to provide angle resolution. However, antenna designs that also achieve low sidelobe levels are desirable for several reasons. First, sensitive ES systems can usefully detect pulses that correspond to the antenna sidelobes if these are sufficiently large. Second, some jamming techniques make use of signals that are received through sidelobes in the radar antenna response and therefore confuse the radar into showing a target at an angle offset from the jammer;
3. Frequency agility involves changing the transmitter frequency pulse to pulse or between groups of pulses. It has some similarities to the use of FH by communication systems, although the primary ideas are to complicate the task of an ES system in interpreting whether the received pulses are from one or more radars, and to reduce the effectiveness of single frequency jammers.

4. LPI radars tend to use continuous wave signals with frequency or phase modulation to provide the desired range resolution. Technical considerations generally restrict the average transmitter power with the result that they are most suited to applications in which long range is not required. Against these signals, conventional radar ES systems are usually limited to very short detection ranges because of the low transmitter power and the effect of receiver optimizations for the processing of short duration pulse signals.[3]

Passive EP generally places considerable emphasis on training and operational procedures. Some of the most spectacular EW successes, such as the decryption of messages ciphered by the German Enigma machine in WW2, resulted, at least in part, from the failure of radio operators to follow correct procedures. The security of communication systems can be compromised in many possible ways. Examples include the transmission of unimportant or unnecessarily long messages; the repeated transmission of the same message with and without encryption; the failure to use code words and available EP capabilities, such as power management, FH, and encryption; and the failure to safeguard encryption equipment and keys. The likelihood of such lapses can be reduced substantially by the institution of suitable procedures followed by training under realistic conditions.

Emission Security policy includes defining procedures and techniques for minimizing the possibility of sensitive information being obtained from the intercept of RF signals that are generated unintentionally in the operation of computer or other electronic systems.

In field or operational environments, tactical EP strategy is set by Emission Control (EMCON) orders, which define specific rules for the management of electromagnetic emissions (12) during a military operation. These rules attempt to strike a balance between various requirements:

1. Maintaining command and control capabilities;
2. Limiting mutual interference between friendly systems;
3. Limiting the useful information that enemy ES can provide;
4. The execution of deception operations.

EMCON rules include the following:

1. Restrictions on transmit power times and use of radio black-out policy;
2. Guidelines, such as frequency allocations and approved system configurations;
3. Restrictions on the type of information that can be transmitted (and thus denied to the enemy);

ADDITIONAL TOPICS

EW and Navigation Systems

Before WW2, specialized direction-finding systems were developed for navigation purposes. From measurements of the angles to radio stations or beacons at known locations, position estimates could be computed. Although there were limitations on the achievable accuracy, this capability was extremely important, particularly at night and in bad weather. During WW2, more sophisticated systems were developed and deployed. Examples include Knickebein, X-Gerat, Y-Gerat, Decca Navigator, GEE, G-H, and Oboe.

Various efforts were made to jam the signals associated with these systems, particularly those used for bombing navigation.[4] Luftwaffe attempts to use the Knickebein, X-Great, and Y-Gerat navigation systems to guide bombers to targets in the U.K. were successfully countered by jamming, although a series of damaging raids was conducted using the X-Gerat system before effective jamming techniques were devised (5). German attempts to jam allied systems, such as GEE and Oboe, were generally less successful.

For example, by the time successful jamming was initiated against Oboe signals at 200 MHz, the Mark III version had moved to 3 GHz. At this frequency, the technical capabilities of the Germans were inadequate for the implementation of effective countermeasures.

In addition, both sides made efforts to interfere with enemy radio beacons, sometimes with the result that aircraft got lost or were even captured after landing in unfriendly territory.

After WW2, various navigation systems were developed and deployed. More recently, the global positioning system (GPS) has become very important, particularly in Western countries, because of the availability of worldwide coverage and the high accuracy that can be achieved. This availability has led to the widespread use of GPS for guiding precision weapons and defining target locations. The military importance of GPS has motivated the development and marketing of GPS jammers. At the same time, recognition of the potential impact of GPS jamming has resulted in serious efforts to develop and implement anti-jam features in military GPSs (31).

EW and IFF Systems

Identification friend foe (IFF) systems are used to provide a means of quickly and positively identifying friendly aircraft. When an unknown aircraft is observed, the IFF system transmits a specially coded signal and looks for the transmission of an appropriate signal in response from the IFF system in the unknown aircraft.

After early IFF systems were deployed in British bombers during WW2, the Germans discovered that the bombers could be tracked by transmitting signals to trigger their IFF systems and observing the IFF signals trans-

[3] An interesting idea is to use commercial FM radio stations as a transmitter in a bistatic radar system. The receivers are located some distance from the transmitter, and the signal processing is designed to measure the relative time shifts between the signal that propagates directly from the transmitter to the receiver and the signal that arrives via a reflection from the target.

[4] Investigations in the UK revealed that bombing attacks carried out at night were often ineffective without the use of electronic navigation aides (5).

mitted in response. Significant losses of aircraft resulted until it was realized that the IFF signals were being exploited, and the systems were removed from the aircraft (5). Since then, significant efforts have been made to reduce the vulnerability of modern IFF systems to EW.

Countermeasures Against IR Sensors

Passive infrared (IR) sensors have important military applications (32). Antiaircraft missiles using IR guidance systems have proven to be very effective in the absence of effective countermeasures, particularly for low-altitude air defense. Other important applications include ground-to-air and air-to-ground target acquisition, fire control, and night vision. In ground combat, the use of IR sensor technology has greatly increased the effectiveness of operations at night and under conditions of bad weather and haze. The usefulness of IR sensors has been enhanced progressively by technical advances in IR detectors and the processing of their outputs. IR sensors have been evolved to operate in both the long-wave infrared and mid-wave infrared bands. These dualband sensors can provide robust performance over a wide range of environmental conditions.

The importance of IR sensors has motivated the expenditure of considerable effort on the development of technology and techniques designed to reduce the effectiveness of IR sensors and their associated weapon systems. This work is very comprehensive and includes modeling and experimental measurements of the IR radiation emitted by platforms, such as ships and aircraft, and the behavior of threat IR sensors.

Flares have been widely used as decoys to distract the IR sensor-based missile guidance systems for the protection of aircraft. The use of flares is often combined with evasive action to ensure that the missile-guidance system continues to track the flare and that the missile's path toward the flare does not take it near the aircraft. Infrared counter measure (IRCM) systems generate an IR signature whose power is modulated in a way that is intended to confuse the tracking system associated with typical IR sensor-based guidance systems. Directional infrared counter measures systems extend the IRCM concept by directing the modulated IR energy toward the threat sensor. Another idea is to use a laser to blind the IR sensor.

IR deception techniques for aircraft have achieved significant successes against more basic IR sensors. However, the development of increasingly sophisticated IR sensors has necessitated continued work on the development of IR countermeasures.

Improvised IR deception measures have been used with some success to simulate ground targets.

The reduction of IR signatures associated with platforms, such as surface ships and aircraft, can significantly improve their survivability. Various measures have been used:

- Cooling visible exhaust duct metal surfaces with air or water;
- Shrouding visible exhaust duct metal surfaces;
- Cooling engine exhaust plumes by mixing them with cool ambient air;
- Cooling exposed surfaces heated by the sun with water;
- Coating exposed surfaces with low-emittance materials;
- Covering ground-based assets with IR camouflage netting.

FUTURE TRENDS IN EW TECHNOLOGY

The evolution of EW technology and concepts is driven by various factors, which include changing operational requirements and technology advances. Future systems will provide significant capability enhancements and other benefits:

1. The development and widespread deployment of capable cell phone networks and their adoption for military purposes means that ES, even at the tactical level, cannot be limited to explicitly military communication systems;
2. Requirements to shorten development cycles and reduce cost will favor increasing use of commercial-off-the-shelf technology and open standards. The implementation of digital signal processing algorithms in software running on general purpose processors and hardware based on field-programmable gate array technology provides a combination of flexibility and performance;
3. Specialized systems will tend to be replaced by multifunction systems. The concept of integrating ES and EA functionality with communication and radar systems will receive increasing attention (33);
4. Networking of EW assets and technical advances will tend to blur the distinction between tactical and strategic EW;
5. Simulators and other aids are being developed to provide realistic scenarios for EW training without requiring large-scale exercises and/or expensive equipment;
6. Models and simulations will be increasingly used to assess EW effectiveness with the aim of determining appropriate system design trade-offs and contributing to the development of EW doctrine;
7. Automated ES and EA systems will be added to the sensors carried by UAVs and platforms such as reconnaissance vehicles;
8. Smart antennas will improve the robustness of communication systems in a jamming environment;
9. The future development of aircraft and naval platforms will place increasing emphasis on signature management;
10. Decoys will be increasingly used for platform protection.

In practice, the application of technical advances will be moderated by various practical issues: There are always competing priorities for personnel and funding. Sophisticated

EW systems are often very expensive to develop and deploy and can be quickly rendered obsolescent by technology advances and changing application requirements. The development of sophisticated defense electronics systems presents formidable challenges. Many systems fall far short of initial expectations for various reasons, which range from faulty technology or trade-off analyses, the failure of anticipated technical advances to materialize, and changing application requirements. The problems involved with the introduction of advanced technology systems into service are considerable:

- Integration into platforms;
- Integration with other systems;
- Provisions made for maintenance;
- Development of suitable doctrine;
- Provisions for interoperability with allied forces;
- Training of users.

It is very easy to underestimate some of these issues. An otherwise capable system may be completely unsuitable for service use if the user interface is poorly thought out. A system may work well in the hands of skilled engineers who have an intimate understanding of its operation, but, in an operational environment, it may be virtually unusable by service personnel, even if they have substantial training and experience. Another common problem is that communications capacity required for the networking of battlefield sensors may not be available, or, if provided by communication satellites, may be prohibitively expensive.

APPENDIX 1 - GEO-LOCATION OF SIGNAL SOURCES FOR COMMUNICATIONS AND RADAR ES

GENERAL CONCEPTS

Several fundamental properties of electromagnetic waves can be used for the geo-location of signal sources:

- the signal propagates at a constant known velocity;
- the phase surfaces are perpendicular to the direction of propagation;
- the electric and magnetic field vectors are perpendicular to the direction of propagation.

In free space, the signal seems to spread out radially from a source and arrive at a receiver via a line-of-sight path. Various techniques have been developed to exploit these properties to obtain lines of position for the transmitters associated with radio, radar, and navigation systems. Using measurements from a sufficient number of sites, the locus of positions for an emitter can be uniquely solved. In practice, the problem can often be usefully simplified by the assumption that the source and sensor sites are located on a plane. If errors can be neglected, then the resulting lines of position (LOPs) pass through the position of the signal source, which thereby results in an unambiguous position estimate. The process of solving the location of a signal source from the LOPs is known as triangulation. In practice, various error sources will affect the estimated LOPs and, with multiple lines of position, the intersections of the LOPs will occur at multiple points or, in some cases, will fail to occur. Many sources of error are present in practice:

1. Environmental noise and interfering signals;
2. Thermal noise and spurious signals generated within the sensor;
3. Mutual coupling between the pairs of elements in an antenna array;
4. Gain and phase mismatches in cables and receivers used in systems that use multiple receivers to measure gain or phase differences;
5. Uncertainties in the positions of the sensors;
6. Propagation effects;
7. Geometric factors caused by the relative location of the emitter and sensors.

In a ground-based environment, propagation effects are very important. The received signal will usually arrive at the sensor via multiple paths (multipath propagation) caused by reflections from terrain features and man-made structures. Many of these error mechanisms will result in systematic bias errors that cannot be removed by averaging. However, various possibilities exist for minimizing the effects of error sources:

1. Careful positioning of sensor sites to minimize terrain masking of areas of interest and local reflections, and provide favorable source-sensor geometries;
2. The elevation of the sensor antenna on a suitable mast;
3. Increasing the number of sensors.

The statistical behavior of errors that develop in estimating the position of a source, and their sensitivity to measurement errors, has been analyzed extensively for various geo-location techniques (34–36). Algorithms for making the best use of available information from sensor arrays have been developed (37,38).

DIRECTION-FINDING TECHNIQUES

Direction finding (DF) is based on the idea of directly measuring the direction that the signal wave front is propagating. Extensive research has been applied to the development of DF techniques. Moreover, many design variables and implementation technologies are possible. Consequently, the design of practical DF systems reflects the trade-offs that are relevant to the specific application requirements. The most common ideas (37, 39, 40) are summarized in the following sections.

DF TECHNIQUES BASED ON AMPLITUDE MEASUREMENTS

The most basic form of DF is to perform an angular search using a directional antenna whose directional characteristics are known and find the angle at which the received power is either a maximum or a minimum. The choice depends on whether a well-defined maxima or null in the directional response exists. The antenna can be continuously rotated and a suitable electro-mechanical system used to display the angle that corresponds to the minimum (or maximum) received signal power. One limitation of this scheme concerns the difficulty of measuring the DF of a signal that is present for a short duration. Nevertheless, some DF systems for radar ES are based on the use of a rotating parabolic reflector antenna. The relative simplicity, coupled with the capability against weak signals provided by the high antenna gain, partly compensates for the other limitations.

Amplitude comparison DF is a more sophisticated idea. The desired angular coverage is divided into sectors, and each sector is associated with a directional antenna having a beam width comparable with the angular width of the sector and a receiver that is designed to measure the amplitude of an observed signal. The angle of arrival is determined in two stages. First, the pair of receivers associated with the largest signal power measurements is found. A coarse estimate of the angle of arrival is defined as the mid-angle between the angles that each of the antennas is pointed. Second, the angle of arrival estimate is refined by computing the ratio of the amplitudes, and using a look-up table, or a calculation based on a model of the directional gain of the antennas, to produce a fine-angle estimate. A trade-off occurs between the number of antennas and the achievable accuracy. This technique is often used in radar ES systems; it is relatively straightforward to implement, and, for microwave frequencies, the antennas are relatively compact. RWRs used in fighter aircraft often use four antennas to provide 360° angular coverage, whereas ES systems for naval craft often use six or eight antennas.

Many communications ES systems use amplitude-comparison DF techniques based on the Adcock pair antenna. This technique is based on the idea of taking the vector difference of the output signals from two closely spaced vertical monopole or dipole antenna elements. The result is a figure-8 gain pattern with the null occurring for signals that propagate across the baseline of the antenna elements. The separation of the antenna elements involves a compromise depending on the frequency range to be covered. Too close a spacing reduces the sensitivity whereas too large a spacing results in a distorted gain pattern. The Watson-Watt DF system, in its simplest form, consists of two Adcock pairs oriented at right angles. The angle of arrival of a received signal can be directly determined from the ratios of the signal powers measured from the two Adcock antenna pairs. With some additional processing, an unambiguous DF measurement can be obtained. At the cost of increased size and complexity, improved performance and frequency coverage can be obtained by using four Adcock pairs.

Interferometric DF Systems

The basic interferometric DF system consists of a pair of monopole or dipole antenna elements that are separated by less than half a signal wavelength and the means for measuring the phase difference between their output signals. Using the measured signal frequency, the known signal propagation velocity, and the antenna separation, the signal angle of arrival with respect to the antenna baseline can be computed. The angle of arrival measured for this arrangement is ambiguous; the signal can arrive from either side of the baseline. This limitation can be resolved by adding one or more antenna elements to form a two-dimensional array for each pair of antenna elements, an angle of arrival estimate relative to the baseline of the antenna pair is obtained. By solving for the result that is most consistent with these measurements, an unambiguous estimate for the angle of arrival is obtained. One implementation uses an array of 5 antennas positioned in a regular pentagon to form 10 antenna pairs, five of which correspond to the faces of the pentagon and the other five to the diagonals (41).

The interferometric DF technique is expensive in hardware. Each antenna in the array requires a dedicated channel from a multichannel receiver that has accurate phase-matching between the channels. Digital signal processing techniques facilitate the implementation of such systems, one point being that phase-matching errors can be corrected by measuring them with a suitable calibration signal, storing their values in a table, and using the stored calibration data to correct subsequent measurements. The correlative DF techniques used by some systems are another development of this concept. Well-designed interferometric DF systems have a relatively good reputation for accuracy, particularly when a large antenna array is used.

Single-Channel DF Systems

To minimize size, cost, weight, and power consumption, several DF system implementations have been developed that require only a single-channel receiver. The pseudo-doppler DF technique is distinguished by the use of a circular array of uniformly spaced antennas with a commutator switch that sequentially connects one antenna in the the array at a time to the receiver. The effect is analogous to moving a single antenna element on a circular track and contributes a sinusoidal phase modulation to the received signal. An estimate of the angle of arrival is obtained by measuring the relative phase shift of this modulation component. The Watson-Watt technique has also been applied successfully to single-channel DF systems.

Single-channel DF techniques are widely used for low-cost portable systems. However, a relatively long observation time is needed compared with the conventional Watson-Watt and interferometric techniques.

Other DF Techniques

Other DF techniques are possible and have some advantages. Circular antenna arrays using the Butler matrix network can provide unambiguous DF with a receiver having as few as two channels. A theoretical comparison of their performance with other techniques is given in Ref. 42. Super-resolution techniques, such as the multiple-signal classification (MUSIC) algorithm (43), have the ability to resolve multiple signal sources in angle, even when their signals overlap in frequency. However, the large antenna arrays and the cost of the associated receiver and processing hardware are difficult to justify for most applications.

Attempts have been made to use power measurements to provide an indication of range. This method presents some difficulties. The actual power radiated by a transmitter is dependent on various factors that include the antenna configuration, height, and the selected transmitter output power (if this functionality is available). Furthermore, in a ground environment, propagation losses depend on the nature of the terrain. The usefulness of power measurements increases if measurements are available from multiple sites.

TIME DIFFERENCE OF ARRIVAL AND FREQUENCY DIFFERENCE OF ARRIVAL GEO-LOCATION TECHNIQUES

The basic concept of geo-location using time difference of arrival (TDOA) measurements can be illustrated by considering a pair of spatially separated receivers and a signal source at an unknown location. Given the assumptions of line of sight propagation paths and fixed-signal propagation velocity, the signals observed at the receivers arrive with delays proportional to the distances from the signal source to the receivers. The difference in delays corresponds to the TDOA.

Given a TDOA measurement and knowledge of the signal-propagation velocity and the receiver locations, the locus of possible transmitter positions can be solved. If the problem is simplified to two dimensions by assuming the signal source and receivers lie on a plane, then the resulting line of position is a hyperbola. Given three or more receivers, the hyperbolic lines of position obtained for the different pairs of receivers will intersect at the signal source location if sources of error can be neglected.

Two basic approaches are used for measuring TDOAs. The first is applicable if a time domain feature of the signal waveform can be easily identified. For example, the time of arrival (TOA) of a pulse modulated signal can be measured by performing amplitude demodulation to obtain the pulse waveform and measuring the absolute time that corresponds to a suitable reference point on the leading edge of the pulse waveform, such as the point where the pulse reaches a fixed fraction of the peak power level. The TDOA can then be obtained by taking the difference between the corresponding TOAs observed at two locations. The second requires that the signals from the receiver sites be relayed to a single site where the relative time differences are measured using signal processing techniques, such as cross-correlation.

TDOA based geo-location techniques involve several complications. The requirement for the accurate measurement of very small relative time delays necessitates carefully designed and engineered systems. If the signals received at the separate sites must be relayed to a common site for processing, then the requirements for suitable data links may involve issues of cost and practicality. Nevertheless, TDOA geo-location techniques have some attractive advantages:

- specialized receiving antennas are not required;
- the orientation of the receiving antenna is not critical;
- several methods can be used to confirm that a signal received at different sites is from the same transmitter;
- the accuracy is relatively unaffected by multipath propagation that occurs in the immediate vicinity of the receiver sites.

The differential frequency shifts that result from relative motions of the transmitters and receivers complicates the signal processing needed for TDOA estimation. With suitable processing, these frequency differences can be estimated and used to define lines or surfaces on which the signal source lies. FDOA based techniques are primarily applicable to airborne or satellite platforms and can be combined with geo-location based techniques based on TDOA measurements.

MINIMIZATION OF ERROR SOURCES

The performance of practical geo-location systems can be improved in several ways.

DF Techniques

The performance of DF systems can vary widely, depending on the implementation and choice of deployment sites:

1. System design choices and trade-offs need to be considered carefully. Antenna arrays with large baselines tend to have performance advantages, but they are generally undesirable for tactical applications. Conversely, attempts to cover a large frequency range with a single antenna array involve significant challenges;
2. Gain and phase mismatches contributed by the receiver hardware and the cables between the antenna and receiver can be corrected by measuring the errors and subtracting them from future measurements. The errors can be measured by using a suitable signal source and radio frequency switch to apply a calibration signal at the point where the cables connect to the antenna. Measurements obtained at suitably chosen test frequencies can be used to construct a calibration table containing the amplitude and phase-correction factors required at each of the test frequencies;
3. Systematic errors contributed by the antenna can be corrected using a calibration table to provide correction values to be subtracted from the measurements. A one-dimensional calibration table can be constructed by carrying out controlled tests using signals transmitted from a fixed angle at frequencies spaced through the frequency range covered by the system and measuring the discrepancy between the actual and observed angles. Because the errors generally must be angle dependent, the use of a two-dimensional calibration table is desirable. This table can be constructed by the repeating the procedure for angles

distributed around the full 360° interval. Interpolation can be used to generate calibration values for intermediate frequencies and angles.

4. The choice of sites for the deployment of DF systems is critical. Ideally, the site should be free of features that contribute to multipath propagation, and line-of-sight propagation should be possible over the area of interest. In these respects, the elevation of the antenna is an important factor. Another consideration is that the sites should be selected to provide favorable sensor-target geometries for geo-location via triangulation.
5. Geo-location performance improves as the number of sites from which DF information is available increases.

TDOA and FDOA Techniques

The performance of TDOA and FDOA geo-location systems is sensitive to system-implementation choices, the nature of the signals of interest, and various aspects of the system deployment:

1. If the system operation is dependent on the relaying of signals received at the sensor sites to a common site for processing, the system must be able to perform this function without significantly degrading the quality of the signals.
2. Provisions must be made to account for the delays contributed by the relaying of the signals observed at the sensor sites to a common site; these delays must be removed or accounted for.
3. The performance of TDOA estimation processing depends on the signal-to-noise ratio and the presence of suitable information contained in the signal modulation. Narrowband signals may require higher signal-to-noise ratios and/or longer observation times to achieve the desired accuracy;
4. Frequency shifts that result from relative motions of the receivers and transmitter affect TDOA measurement processing. If, for scenarios of interest, they are sufficiently important, then provisions must be made in the TDOA estimation processing to remove them. If FDOA information is used for geo-location, then the most favorable results will be obtained when the sensors move rapidly, because this action increases the relative frequency shifts, and a given error in frequency measurement becomes less significant. Also, uncertainties contributed by the movement of the signal source are reduced.

BIBLIOGRAPHY

1. D. C. Schleher, *Electronic Warfare in the Information Age*, Norwood, MA: Artech House, 1999.
2. R. Poisel, *Introduction to Communications Electronic Warfare Systems*, Norwood, MA: Artech House, 2002.
3. A. Price, *Instruments of Darkness: The History of Electronic Warfare*, Encore Editions, 1978.
4. A. Price, *The History of US Electronic Warfare, Volume 1, The Years of Innovation–Beginnings to 1946*, Norwood, MA: Artech House, 1984.
5. R. V. Jones, *Most Secret War*, London, UK: Hamish Hamilton, 1978.
6. M. Arcangelis, *Electronic Warfare: From the Battle of Tsushima to the Falklands and Lebanon Conflicts*, Dorset, United Kingdom: Blandford Press, 1985.
7. A. Price, *The History of US Electronic Warfare, Volume II, The Renaissance Years, 1946 to 1964*, Norwood, MA: Artech House, 1989.
8. A. Price, *The History of US Electronic Warfare, Volume III, Rolling Thunder Through Allied Force, 1964 to 2000*, Norwood, MA: Artech House, 2000.
9. P. Mihelich, Jamming systems play secret role in iraq, Available: http://www.cnn.com/2007/TECH/08/13/cied.jamming.-tech/index.html.
10. R. J. Hanyok, Spartans in Darkness: American SIGINT and the Indochina War, 1945–1975, Vol. **7,** Center for Cryptologic History, National Security Agency, 2002.
11. D. Eshel, EW in the Yom Kippur War, *J. Electronic Defense*, **30**(10): 2007.
12. Joint Publication 3-13.1, Electronic Warfare, January 25, 2007, Available: http://www.fas.org/irp/doddir/dod/jp3-13-1.pdf.
13. M. Streetly (ed.), *Janes Radar and Electronic Warfare Systems 2005-2006*, Surrey, UK: Jane's Information Group, 2005.
14. P. W. East, ESM Range Advantage, *IEE Proc.*, **144**(4): 1985.
15. R. G. Wiley, *ELINT: The Interception and Analysis of Radar Signals*, Norwood, MA: Artech House, 2006.
16. R. A. McDonald and S. K. Moreno, Raising the periscope ... grab and poppy: america's early ELINT satellites, *Center for the Study of National Reconnaissance, National Reconnaissance Office,* Chantilly, VA, September 2005.
17. P. Hill, E. Adams, and V. Comley, Techniques for detecting and characterizing covert communications signals, *Proc. European Conference on Security and Detection*, April 1997.
18. R. Inkol, S. Wang, and S. Rajan, FFT filter bank-based CFAR detection schemes, *Proc. of Midwest Symposium on Circuits and Systems*, August 5–8, 2007.
19. Y. T. Chan and L. G. Gadbois, Identification of the modulation type of a signal, *Signal Processing*, **16**(2): 1989.
20. K. Nandi and E. E. Azzouz, Algorithms for automatic modulation recognition of communication signals, *IEEE Trans. Commun.*, **40**(4): 1998.
21. D. Boudreau, C. Dubuc, F. Patenaude, M. Dufour, J. Lodge, and R. Inkol, A fast automatic modulation recognition algorithm and its implementation in a spectrum monitoring application, *Proc. of MILCOM 2000*, 2000.
22. O. A. Dobre, A. Abdi, Y. Bar-Ness, and W. Su, Survey of automatic modulation classification techniques: classical approaches and new trends, *IET Communications*, **1**(2): 2007.
23. K. I. Talbot, P. R. Duley, and M. H. Hyatt, Specific emitter identification and verification, *Northrop Grumman Technol. Rev. J.*, 2003.
24. O. H. Tekbas, N. Serinken, and O. Ureten, An experimental performance evaluation of a novel transmitter identification system under varying environmental conditions, *Can. J. Elect. Comput. Eng.*, **29**(3): 2004.
25. J. B. Tsui, *Microwave Receivers with Electronic Warfare Applications*, New York: Wiley, 2005.
26. P. W. East, Microwave intercept receiver sensitivity estimation, *IEE Proc., Radar, Sonar Navigation*, **132**(4): 1997.

27. D. E. Maurer, R. Chamlou, and K. O. Genovese, Signal processing algorithms for electronic combat receiver applications, *John Hopkins APL Tech. Dig.*, **18**(1): 1997.
28. A. C. Brown, *Bodyguard of Lies*, New York: Harper and Row, 1975.
29. K. Burda, The performance of the follower jammer with a wideband-scanning receiver, *J. Elect. Eng.*, **55**(1-2): 2004.
30. G. Goodman, New challenges for ground EW –democratized jamming, *J. Electronic Defense*, **30**(10): 2007.
31. S. Rounds, Jamming protection of GPS receivers, *GPS World*, 2004.
32. R. D. Hudson, The military applications of remote sensing by infrared, *IEEE Proc.*, **63**(1): 1975.
33. G. C. Tavik et al, The advanced multifunction RF concept, *IEEE Trans. Microwave Theory Tech.*, **53**(3): 2005.
34. R. G. Stansfield, Statistical theory of DF fixing, *J. IEE*, 1947.
35. P. C. Chestnut, Emitter location accuracy using TDOA and differential Doppler, *IEEE Trans. Aerosp. Electron. Syst.*, 1982.
36. D. J. Torrieri, Statistical theory of passive location systems, *IEEE Trans. Aerosp. Electron. Syst.*, **20** 1984.
37. R. Poisel, *Electronic Warfare Target Location Methods*, Norwood, MA: Artech House, 2005.
38. D. Elsaesser, The discrete probability density method for target geolocation, *Proc. Canadian Conference on Electrical and Computer Engineering*, May 2006.
39. S. E. Lipsky, *Microwave Passive Direction Finding*, Raleigh, NC: SciTech Publishing, 2003.
40. S. Chandran (editor), *Advances in Direction-of-Arrival Estimation*, Norwood, MA: Artech House, 2006.
41. N. King, I. Pawson, M. Baker, R. Shaddock, and E. Stansfield, Direction Finding, U.S. Patent 4,639,733, January 27, 1987.
42. W. Read, An Evaluation of the Watson-Watt and Butler Matrix Approaches for Direction Finding, DREO Technical Report 1999-092, September 1999.
43. R. O. Schmidt, Multiple emitter location and signal parameter estimation, *IEEE Trans. Antennas Propag.*, **AP-34**, 1986.

ROBERT INKOL
Defence R & D Canada
Ottawa, Ontario, Canada

ENVIRONMENTAL SCIENCE COMPUTING

ENVIRONMENT AND SOCIETY

The environmental problems are becoming more and more important for the modern society, and their importance will certainly be increased in the near future. High pollution levels (high concentrations and/or high depositions of certain harmful chemical species) may cause damage to plants, animals, and humans. Moreover, some ecosystems can also be damaged (or even destroyed) when the pollution levels become very high. This explains why the pollution levels must be studied carefully in the efforts

- to predict the appearance of high pollution levels, which may cause different damages in our environment and/or
- to decide what must be done to keep the harmful concentrations and/or depositions under prescribed acceptable limits.

The control of the pollution levels in highly developed and densely populated regions in the world is an important task that has to be handled systematically. This statement is true for many regions in Europe and North America but also other parts of the world are under economic development currently and urgent solutions of certain environmental problems either are already necessary or will soon become necessary. The importance of this task has been increasing steadily in the beginning of the new millennium. The need to develop reliable and easily applicable control strategies for keeping harmful pollution levels under prescribed limits will become even more important in the next decades.

Climate changes are causing another challenging problem for the modern society. The quick changes have many different consequences. The impact of the climatic changes on the pollution levels is one of the consequences, and this consequence must be investigated carefully by studying the relationship between climatic changes and high pollution levels. It should also be mentioned here that there is a feedback: The pollution levels influence the climatic changes. It is necessary to couple environmental models with climatic models to study fully the interrelations between climatic changes and pollution levels. This task is very challenging.

Mathematical models are powerful tools when the trends in the development of the pollution levels and the measures which the society must take to ensure a sustainable development are studied. These models are often very complex and lead to huge computational tasks. Some tasks cannot be treated even if powerful modern computers are used.

IMPORTANT TASKS TREATED BY THE ENVIRONMENTAL MODELS

Advanced mathematical models for studying environmental phenomena can be used successfully to design control strategies for keeping the pollution under critical levels under the assumption that these models produce reliable results. The application of comprehensive environmental models in sensitivity tests is important in the efforts

- to understand better the physical and chemical processes involved in the environmental phenomena or to treat efficiently the tasks proposed by policy makers and
- to ensure that the control strategies for keeping the pollution under the prescribed acceptable limits are reliable.

Sensitivity tests can be applied to resolve these two tasks. It is important to study the sensitivity of concentrations and deposition of harmful pollutants caused by variations of:

- anthropogenic emissions,
- biogenic emissions,
- meteorological conditions,
- velocity rates of chemical reactions,
- boundary conditions,
- initial conditions, and
- numerical algorithms.

This list is certainly not complete and can be continued. It is even more important to emphasize the fact that this list tells us that the task of performing complete sensitivity analysis by applying large-scale environmental models is extremely large and very difficult. Finally, many terms, in which the parameters from the above list are involved, are nonlinear. The nonlinearity causes great difficulties, which can be resolved only by conducting many experiments with different scenarios and studying carefully the results to find typical trends and relationships.

The difficulties are increased because there are interconnections of the effects because of variation of different parameters. For example, the variation of both the anthropogenic emissions and the biogenic emissions may lead to some effects, which are not observed when only the anthropogenic emissions or only the biogenic emissions are varied. The effects caused by simultaneous variations of several key parameters can only be studied by increasing the number of scenarios used in the experiments. Thus, the tasks become larger and more difficult.

The necessity of validating the results is an additional source for difficulties. The problem of designing a completely reliable routine for validating the results of the sensitivity tests is still open. Two approaches can be used (and, in fact, are commonly used) in the attempts to validate the results from sensitivity analysis tests:

- comparisons with measurements and
- comparisons with results obtained by other models.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

The objections, which can be raised against the complete reliability of the comparisons with measurements for the validation of the model results, are many and serious. The most important objection is the well-known fact that two different quantities are compared when such a procedure is used. The measurement is a quantity, either concentration or deposition, which is obtained at a given geographical point (the location of the measurement station). The corresponding quantity, which is the quantity calculated by the model, is a representative mean value averaged in some surrounding (determined by the spatial discretization chosen) of the point in which the measurement station is located. This fact implies that even if both the measurement and the corresponding calculated result are exact (which will never happen in practice), they will in general be different. Another implication, which is even more important from a practical point of view, is the following: We should expect to improve the possibility (the potential possibility, at least) of getting better validation results by using comparisons with measurements when we increase the spatial resolution of the model, but the computational tasks become larger and much more difficult when the spatial resolution is increased. It may become necessary to replace some physical and chemical mechanisms used in the model with coarse resolution with more accurate mechanisms when the resolution is refined. Finally, the need for accurate input data for large-scale models defined on refined grids also causes great difficulties.

The objections, which can be made in the case where the results obtained by two or more models are compared, are also both many and serious. It is extremely difficult to determine in a reliable manner the precise reason for differences of results produced by different models. The answer of the following question is interesting when a long sequence of sensitivity tests is run: *What is the relationship between the parameter that is varied and the studied quantity (the concentration or the deposition of a certain harmful pollutant)?* If two models are run with the same sequence of sensitivity tests and if the relationship between the parameter that is varied and the model results is different for the two models, then the difference may, partially or totally, be caused by several reasons. The differences may, for example, be caused (or, at least, be influenced to some degree) by the use of different projections and/or resolutions in the different models, by the application of different sets of input data, by the fact that the numerical methods used are different, by the fact that the chemical schemes are not the same, and so on. It is not clear how these unwanted reasons for differences of the results from two or more models can be eliminated to study only the relationship between the parameter we are varying and the studied quantity.

It should be emphasized here that the objections against the two commonly used procedures for validating results from sensitivity tests indicate that it is necessary to be cautious. It should also be emphasized, however, that it is absolutely necessary to do such comparisons. Many sound conclusions can be drawn after such comparisons, but one should not fully rely on the results of the comparisons. One should continue the search for better and more reliable validation tests.

MATHEMATICAL DESCTIPTION OF ENVIRONMENTAL MODELS

The environmental models are normally described by systems of partial differential equations (PDEs). The number of equations is equal to the number of chemical species studied by the modes and the unknown functions are concentrations of these species. Five basic stages are described in the development of a large environmental model:

- one has to select the physical and chemical processes that are to be taken into account during the development of the model,
- the selected processes must be described mathematically,
- the resulting system of PDEs must be treated by numerical methods,
- the reliability of the obtained results must be evaluated and
- conclusions should be drawn.

It is important to take into account *all* relevant physical and chemical processes during the development of the models. If more physical and chemical processes are included in the model, then one should expect that more accurate and more reliable results can be calculated. However, two difficulties are related to the attempt to include as many as possible physical and chemical processes in the model:

- The complexity of the model is increased when more processes are included in it. The treatment of the model on the available computers might become very difficult, and even impossible, when too many processes are taken into account.
- Some physical and chemical processes are still not well understood, which means that such processes must either be neglected or some simplified mechanisms, based on uncertain assumptions and/or on experimental data, must be developed.

It is necessary to find a reasonable compromise related to the number of processes that are to be taken into account when a large environmental model is developed. This reasoning explains also why it is necessary to validate the model results.

The selected physical and chemical processes have to be described by mathematical terms. Some more- or less-standard rules exist that can be used for the mathematical description of different processes. Several examples, which are related to air pollution modes, are listed below:

- The transport caused by the wind (called advection) is described by using terms that contain first-order spatial derivatives of the unknown functions (the concentrations of the studied pollutants) multiplied by the wind velocities.
- The diffusion of the concentrations is expressed by second-order spatial derivatives multiplied by the diffusivity coefficients.

- The chemical reactions are represented by nonlinear mathematical terms.
- The change of the concentrations in time is given by first-order derivatives of the concentrations of the pollutants with respect to time.

It should be stressed here that the above rules are applicable not only to air pollution models but also to many other environmental models.

When all selected physical and chemical processes are expressed by some mathematical terms, then these terms have to be combined in a system of PDEs. For example, when long-range transport air pollution is studied, the system of PDEs, which represents the mathematical model, can be written as follows (it should be mentioned that similar systems are used in other environmental models):

$$\begin{aligned}
\frac{\partial c_i}{\partial t} &= -u\frac{\partial c_i}{\partial x} - v\frac{\partial c_i}{\partial y} && \text{horizontal advection}\\
&\quad -\frac{\partial}{\partial x}\left(K_x\frac{\partial c_i}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_y\frac{\partial c_i}{\partial y}\right) && \text{horizontal diffusion}\\
&\quad +Q_i(t,x,y,z,c_1,c_2,\ldots,c_q) && \text{chemical reactions}\\
&\quad +E_i(t,x,y,z) && \text{emissions}\\
&\quad +(k_{1i}+k_{2i})c_i && \text{dry and wet deposition}\\
&\quad -w\frac{\partial c_i}{\partial z} + \frac{\partial}{\partial z}\left(K_z\frac{\partial c_i}{\partial z}\right) && \text{vertical exchange}\\
i &= 1,2,\ldots,q && q-\text{number of chemical species} \qquad (1)
\end{aligned}$$

where

- $c_i = c_i(t,x,y,z)$ is the concentration of the chemical species i at point (x,y,z) of the space domain and at time t of the time interval,
- $u = u(t,x,y,z)$, $v = v(t,x,y,z)$ and $w = w(t,x,y,z)$ are wind velocities along the Ox, Oy and Oz directions respectively at point (x,y,z) and time t,
- $K_x = K_x(t,x,y,z)$, $K_y = K_y(t,x,y,z)$ and $K_z = K_z(t,x,y,z)$ are diffusivity coefficients at point (x,y,z) and time t (it is often assumed that K_x and K_y are nonnegative constants, whereas the calculation of K_z is normally rather complicated),
- $k_{1i} = k_{1i}(t,x,y,z)$ and $k_{2i} = k_{2i}(t,x,y,z)$ are deposition coefficients (dry and wet deposition, respectively) of chemical species i at point (x,y,z) and time t of the time interval (for some species these coefficients are nonnegative constants; the wet deposition coefficients k_{2i} are equal to zero when it is not raining).

Normally, it is not possible to solve exactly the systems of PDEs by which the large environmental models are described mathematically. Therefore, the continuous systems of the type in Equation (1) are to be *discretized*. Assume that the space domain, on which Equation (1) is defined, is a parallelepiped (this is as a rule the case when environmental models are to be handled) and that $x \in [a_1, b_1]$, $y \in [a_2, b_2]$, $z \in [a_3, b_3]$ and $t \in [a, b]$. Consider the grid-points (t_n, x_j, y_k, z_m) where $x_j = a_1 + j\Delta x$, $j = 0,1,2,\ldots,N_x$, $y_k = a_2 + k\Delta y$, $k = 0,1,2,\ldots,N_y$, $z_m = a_3 + m\Delta z$, $m = 0,1,2,\ldots,N_z$ and $t_n = a_3 + n\Delta t$, $n = 0,1,2,\ldots,N_t$. Assume also that the initial values $c_i(a,x,y,z)$ are given. Then, the task of finding the exact solution of the unknown functions c_i at all points (t,x,y,z) of the domain (infinite number of points) is reduced to the task of finding approximations of the values of the functions c_i at the points (t_n, x_j, y_k, z_m); the number of these points can be very large (up to many millions), but it is finite. The original task is relaxed in two ways. First, the number of points at which the problem is treated is reduced to the number of grid-points, and then it is required to find approximate solutions instead of the exact solution.

In the example given above, equidistant grids are introduced (i.e., Δx, Δy, Δz, and Δt are constants). Nonequidistant grids can also be used. The vertical grids are normally not equidistant.

It is assumed here that Cartesian coordinates have been chosen. Other coordinates, for example, spherical coordinates, can also be used.

The above two remarks illustrate the fact that the discretization can be performed in different ways. The important thing is that the main idea remains the same: One considers *approximate* values of the unknown functions at a *finite* number of grid-points, which are defined by the discretization chosen, instead of the exact solution of Equation (1) on the whole continuous space domain.

Numerical methods must be used to find approximate values of the solution at the grid-points. It is also appropriate to split the model, the system of PDEs of the type in Equation (1), into several submodels (subsystems), which are in some sense simpler. Another advantage when some splitting procedure is applied is that the different subsystems have different properties, and one can try to select the best numerical method for each subsystem.

It is clear that if some splitting procedure and appropriate numerical methods are already chosen, then any continuous system of the type in Equation (1), which represents an environmental model, is replaced by several *discrete* submodels that have to be treated on the available computers.

As mentioned above, the model described by Equation (1) is an air pollution model. However, it must be emphasized, once again, that many environmental models are also described by systems of partial differential equations and, thus, can be treated similarly.

NEED FOR EFFICIENT ORGANIZATION OF THE COMPUTATIONAL PROCESS

The discretization of the system of PDEs by which the environmental models are described mathematically leads to huge computational tasks. The following example illustrates clearly the size of these tasks. Assume that

- $N_x = N_y = 480$ (when a 4800 km × 4800 km-domain covering Europe is considered, then this choice of the discretization parameters leads to 10 km × 10 km-horizontal cells),

- $N_z = 10$ (i.e., 10 layers in the vertical direction are introduced) and
- $N_s = q = 56$ (the chemical scheme contains 56 species).

Then, the number of equations that are to be handled at each time-step is $(N_x + 1)(N_y + 1)(N_z + 1)N_s = 142,518,376$. A run over a time-period of one year with a time stepsize $\Delta t = 2.5$ seconds will result in $N_t = 213,120$ time-steps. When studies related to climatic changes are to be carried out, it is necessary to run the models over a time period of many years. When the sensitivity of the model to the variation of some parameters is studied, many scenarios (up to several hundreds) are to be run. This short analysis demonstrates the fact that the computational tasks that occurs when environmental studies are to be carried out by using large-scale models are enormous. Therefore, it is necessary:

- to select fast but sufficiently accurate numerical methods and/or splitting procedures,
- to exploit efficiently the cash memories of the available computer,
- to parallelize the code

in the efforts to make a large environmental model tractable on the available computers. It should be mentioned that it may be impossible to handle some very large environmental models on the computers currently available even when the above three conditions are satisfied.

SOME APPLICATIONS

Results from two important applications,

(1) the impact of biogenic emissions on pollution levels and
(2) the influence of the climatic changes on pollution levels in Europe,

will be given to illustrate the usefulness of models when some environmental phenomena are studied.

Bad Days

High ozone concentrations can cause damage to human health. Therefore, critical levels for ozone have been established in the European Union (EU) as well as in other parts of the world. Some of these critical levels are legislated in the EU Ozone Directive. Assume that $c_{\max}$ is the maximum of the 8-hour averages of the ozone concentrations in a given day at site **A**. If the condition $c_{\max} > 60$ ppb is satisfied, then the day under consideration is declared as a “bad day” for site **A**. “Bad days” have damaging effects on some groups of human beings (for example people who suffer from asthmatic diseases). Therefore, the number of such days should be reduced as much as possible. Two aims are stated in the Ozone Directive issued by the EU Parliament in year 2002:

- **Target aim**. The number of “bad days” in the European Union should not exceed 25 days after year 2010.
- **Long-term aim**. No “bad day” should occur in the European Union (the year after which the long-term aim has to be satisfied is not specified in the EU Ozone Directive).

Biogenic Emissions and “Bad Days”

The distribution of the “bad days” in different parts of Europe is shown in the two upper plots of Fig. 1: (a) in left-hand-side plot for year 1989 and (b) in the right-hand-side plot for year 2003. The numbers of the “bad days” in Europe is in general reduced considerably in the period from 1989 to 2003. The significant reduction of the human-made (anthropogenic) emissions in Europe is the reason for these reductions. The reductions of the human-made (anthropogenic) emissions in Europe are shown in Fig. 2. The reduction of the emissions of nitrogen oxides (NOx emissions) and volatile organic compounds (VOC emissions) is most important for the changes of the numbers of “bad days.”

The results presented in the two upper plots are obtained by using “normal biogenic emissions.” Increased biogenic emissions (“high biogenic emissions”) were also used. The changes of the “bad days” when high biogenic emissions are used instead of the normal biogenic emissions are shown on the two lower plots in Fig. 1 (for year 1989 in the left-hand-side plot, and for year 2003 in the right-hand-side plot). A great difference can be observed: The largest increases of the “bad days” in year 1989 are in Eastern and South-Eastern Europe, whereas the largest increases in year 2003 occur in Germany and some surrounding countries. It is not very clear what the reason for the different behavior is. Some possible explanations are given below.

Consider the NOx and VOC emissions, which contribute to the formation and destruction of ozone (and, thus, these emissions play an important role when the numbers of “bad days” are to be studied). Consider also the two major countries, Germany and Ukraine, where the differences are largest. Comparing the changes of the emissions for years 1989 and 2003, it is observed (Table 1) that whereas the reductions of the VOC emissions are nearly the same (60.3% for Germany and 53.4% for Ukraine), the reduction of the NOx emissions of nitrogen oxides in Germany (46.3%) is nearly four times larger than the corresponding reduction in Ukraine (12.1%). This disparity might be one reason for the different patterns in the lower plots of Fig. 1.

Some combinations of NOx and VOC emissions imply dominance of the formation of ozone, whereas other combinations increase the destruction rate. Assume that the biogenic emissions are kept fixed (the normal biogenic emissions). The results indicate that the combinations of these two types of emissions in Eastern Europe increase the formation rate of ozone (in the transition from year 1989 to year 2003), whereas the opposite situation is observed in Central Europe. This result could also be observed by comparing the upper two plots in Fig. 1: The numbers of “bad days” in the region around Moscow is increased in 2003 despite the fact that the both the NOx and VOC

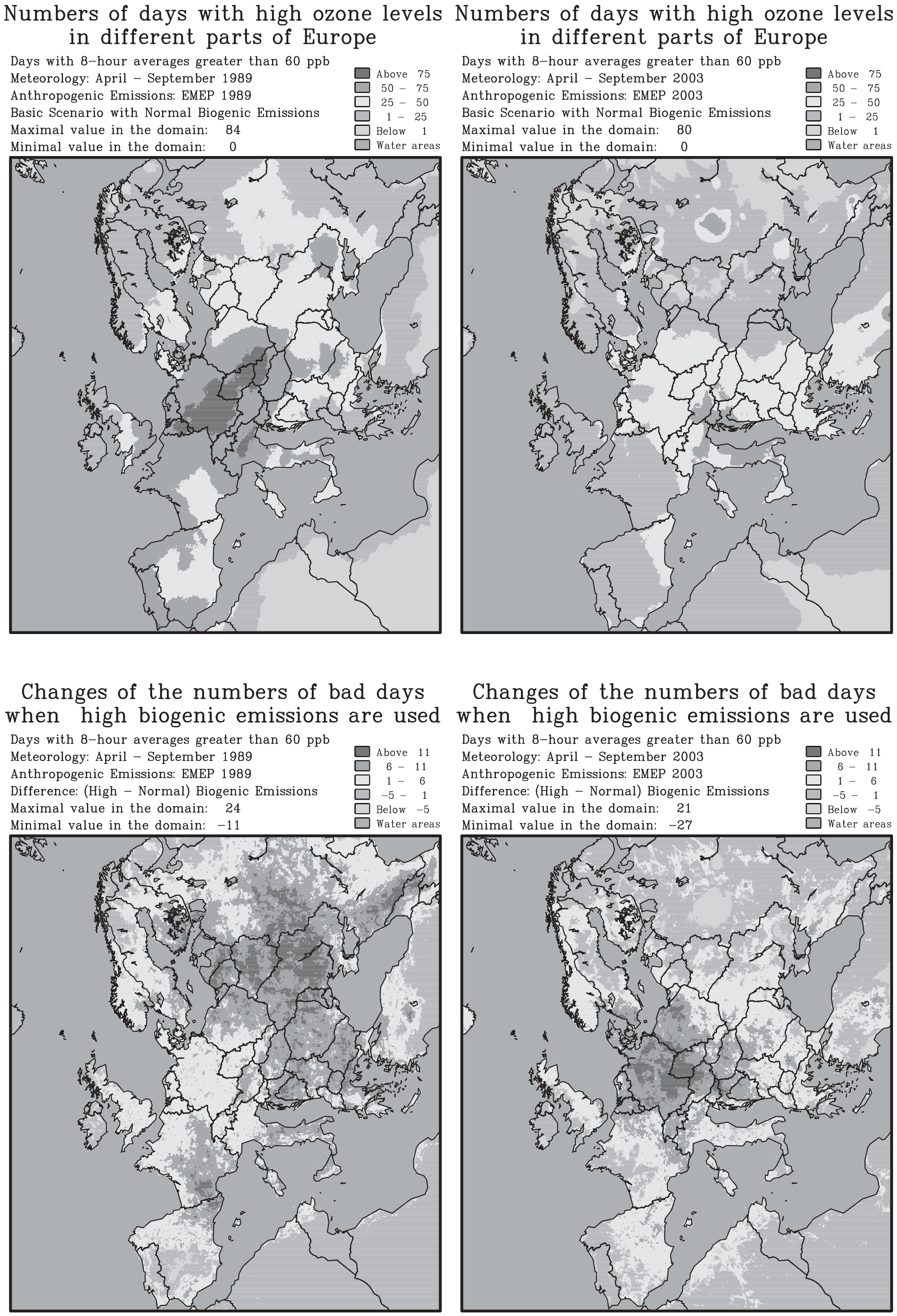

Numbers of days with high ozone levels in different parts of Europe
Days with 8–hour averages greater than 60 ppb
Meteorology: April – September 1989
Anthropogenic Emissions: EMEP 1989
Basic Scenario with Normal Biogenic Emissions
Maximal value in the domain: 84
Minimal value in the domain: 0
Above 75
50 – 75
25 – 50
1 – 25
Below 1
Water areas
Numbers of days with high ozone levels in different parts of Europe
Days with 8–hour averages greater than 60 ppb
Meteorology: April – September 2003
Anthropogenic Emissions: EMEP 2003
Basic Scenario with Normal Biogenic Emissions
Maximal value in the domain: 80
Minimal value in the domain: 0
Above 75
50 – 75
25 – 50
1 – 25
Below 1
Water areas
Changes of the numbers of bad days when high biogenic emissions are used
Days with 8–hour averages greater than 60 ppb
Meteorology: April – September 1989
Anthropogenic Emissions: EMEP 1989
Difference: (High – Normal) Biogenic Emissions
Maximal value in the domain: 24
Minimal value in the domain: –11
Above 11
6 – 11
1 – 6
–5 – 1
Below –5
Water areas
Changes of the numbers of bad days when high biogenic emissions are used
Days with 8–hour averages greater than 60 ppb
Meteorology: April – September 2003
Anthropogenic Emissions: EMEP 2003
Difference: (High – Normal) Biogenic Emissions
Maximal value in the domain: 21
Minimal value in the domain: –27
Above 11
6 – 11
1 – 6
–5 – 1
Below –5
Water areas

Figure 1. High versus normal biogenic emissions.

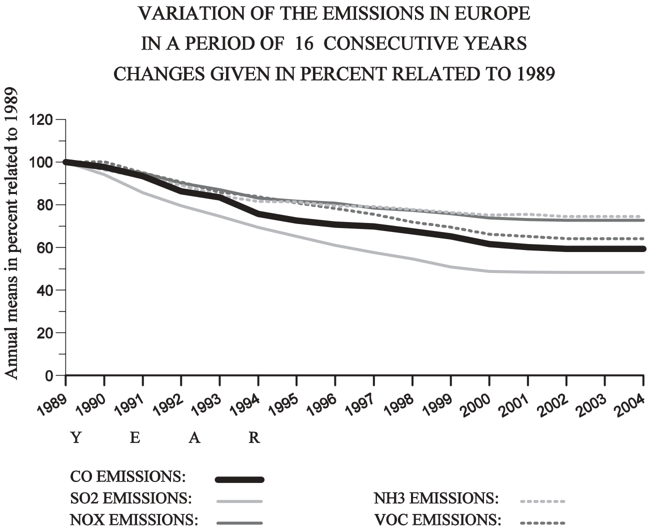


Figure 2. Reductions of the European human-made emissions in the period 1989–2004.

emissions were decreased (the biogenic emissions in the upper two plots are the same).

Other reasons can explain the different patterns in the two lower plots of Fig. 1. For example, the meteorological conditions may also play some role. More scenarios are needed to find the right answer of the question about the differences of the numbers of "bad days" in different parts of Europe. However, it is important to emphasize that the biogenic emissions:

- play an important role when the numbers of "bad days" are studied,
- must be studied more carefully (some scientists claim that these are underestimated). and
- requires much more scenarios, which have to be run over long time periods (many years). These scenerios are needed in the efforts to answer the questions that originated during the experiments.

The last requirement leads to huge computational tasks and great storage requirements.

Climatic Changes and "Bad Days"

Many programs are designed to predict the future state of the world's climate and to explore the effects of such changes on a variety of policy sectors (e.g., food, water, energy, socioeconomic development, and regional security). The major efforts in this direction are coordinated by the Intergovernmental Panel on Climate Change (IPCC). Environmental degradation is also of concern in future climate scenarios, and much effort has been dedicated to understand changing pressures on an already stressed system. Here, the attention will be focused on pollution in the future climate. Within many regions on planet Earth, air pollution policy has been regionalized to achieve two major aims: (*1*) to control and reduce transboundary pollution and (*2*) to meet policy objectives to limit air pollution impacts on human health and sensitive ecosystems. The effort to achieve these two aims has been a daunting task. Within Europe, the Convention on Long Range Transport of Air Pollution has been dedicated to establishing a legal framework for reducing air pollution and assuring the safety of humans and ecosystems within prescribed limits. Limit values for a variety of pollutants have also been established, as mentioned above, in the EU. However, in reaching compliance to the air quality directives, very few studies have considered the possibility that climate change *may induce* an additional controlling factor. Therefore, it is desirable to answer the question:

- *Will climate change add to the rate of reaching compliance to air quality policy objectives, or will it make the process of reaching compliance more difficult?*

This question can be considered in two aspects: qualitative and quantitative. If only a qualitative answer is needed, then the task becomes easy. The answer is obviously yes, because both the chemical reactions and the biogenic emissions depend on the temperature. Thus, the warming effect will certainly have some impact on the pollution levels. The second aspect is much more interesting, because it is important to produce some quantitative evaluation of the impact of the climate changes on the pollution levels. The predictions about the increase of the annual temperatures in Europe according to the IPCC SRES A2 Scenario as well as several other conclusions, which are related to the climatic changes in Europe and are discussed in the reports prepared by the scientists from IPCC, can be used to prepare three climatic scenarios related to pollution studies. These scenarios can also be used to obtain quantitative evaluation of the impact of future climatic changes on certain pollution levels.

Climate Scenario 1

The predicted, by the IPCC SRES A2 Scenario, annual changes of the temperature were used to produce this climatic scenario. Resulting from this scenario changes of the temperature in Europe are shown in Fig. 3. Consider any cell of the grid used to create the plot shown in Fig. 3

Table 1. Changes of Anthropogenic Emissions Participating in the Formation and Destruction of Ozone in Germany and Ukraine

	Emissions of nitrogen oxides			Emissions of volatile organic compounds		
Country	1989	2003	Reduction	1989	2003	Reduction
Germany	2989	1605	46.3%	3202	1272	60.3%
Ukraine	1065	936	12.1%	1512	704	53.4%

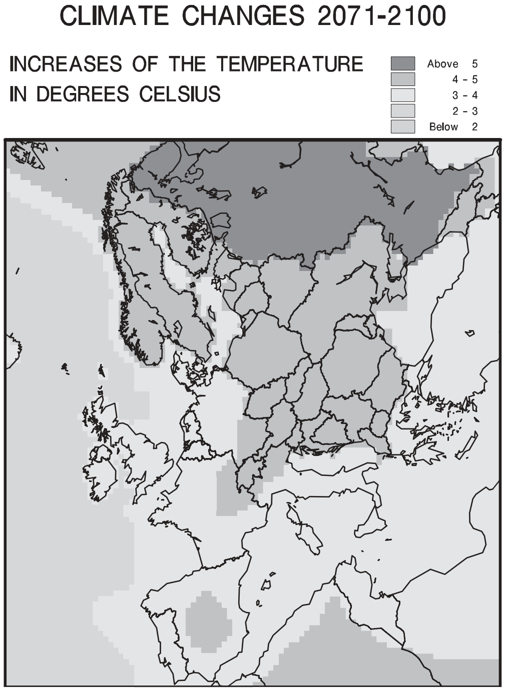


Figure 3. Future changes of the temperatures in Europe according to the IPCC SRES A2 Scenario.

and assume that this cell is located in a region where the increase of the temperature is in the interval [a,b]. The temperature at the chosen cell at hour n (where n is in the interval from 1989 to 2004) is increased by an amount $a + c(n)$, where $c(n)$ is randomly generated in the interval $[0, b - a]$. The mathematical expectation can increase the annual mean of the temperature at any cell of the space domain, where it is predicted that the increase of the temperature is in the interval [a,b], $(b - a)/2$. Only temperatures are varied in this scenario and the mean value of the annual change of the temperature at a given point will tend to be $(b - a)/2$ for each year of the chosen interval (from 1989 to 2004).

Climate Scenario 2

The extreme meteorological events will become even stronger in the future climate. It is also expected that: (*1*) higher maximum temperatures and more hot days will be observed in the land areas; (*2*) higher minimum temperatures, fewer cold days, and fewer frost days will be observed in nearly all land areas; and (*3*) the diurnal temperature range will be reduced over land areas. The temperatures during the night were increased with a factor larger than the factor by which the daytime temperatures were increased. In this way, the second and the third requirements are satisfied. The first requirement is satisfied as follows. During the summer periods, the daytime temperatures are increased by a larger amount in hot days. All these changes are observed only over land. Furthermore, the temperatures were varied in such a way that their annual means remained the same, at all cells, as those in the first climatic scenario. The cloud covers over land during the summer periods were also reduced.

Climate Scenario 3

It is also expected that there will be (*1*) more intense precipitation events and (*2*) increased summer drying and associated risk of drought. The precipitation events during winter were increased both over land and over water. The precipitation events in the continental parts of Europe were reduced during summer. Similar changes in the humidity data were made. The cloud covers during winter had increased, whereas the same cloud covers as in the second climatic scenario were applied during summer. As in the previous two climatic scenarios, the mathematical expectation of the annual means of the temperatures is the same as the predictions made in the IPCC SRES A2 Scenario.

Computational Difficulties. The computational requirements are enormous when the influence of the climatic changes on pollution levels is studied. The difficulties develop because:

- the models are to be run over a long time period (the time period was 16 years in the particular study discussed here, but it is perhaps more appropriate to use a longer period, say, 25 or even 50 years),
- many scenarios are to be used (14 scenarios were actually used, but again more scenarios will give better information about the studied phenomena),
- it is highly desirable to use fine resolution (the domain under consideration, containing the whole of Europe and its surroundings, was discretized by using a 480 × 480 × 10 grid).

The task of running 14 scenarios over a time period of 16 years on a fine grid (480 × 480 × 10 cells) is extremely time consuming. The storage requirements are also enormous. Therefore, the task of running so many scenarios over a long time period could be solved successfully only if several requirements are simultaneously satisfied: (*1*) fast but also sufficiently accurate numerical methods are to be implemented in the model, (*2*) the cache memories of the available computers have to be used efficiently, (*3*) codes that can be run in parallel have to be developed and used, and (*4*) reliable and robust splitting procedures have to be implemented. It must be emphasized that it is *impossible* to handle the 14 scenarios over a time period of 16 years on the available supercomputers if the subtasks (*1*)–(*4*) are not solved efficiently. Even when it was done, it took more than 2 years to compute data from all 2688 runs (14 scenarios × 16 years × 12 months) carried out in this particular study. This fact illustrates the great computational difficulties that are related to the investigation of impact of climatic changes on pollution levels.

Model Results

Some results related to the numbers of "bad days" are given in Fig. 4. Several conclusions can be drawn by investigating

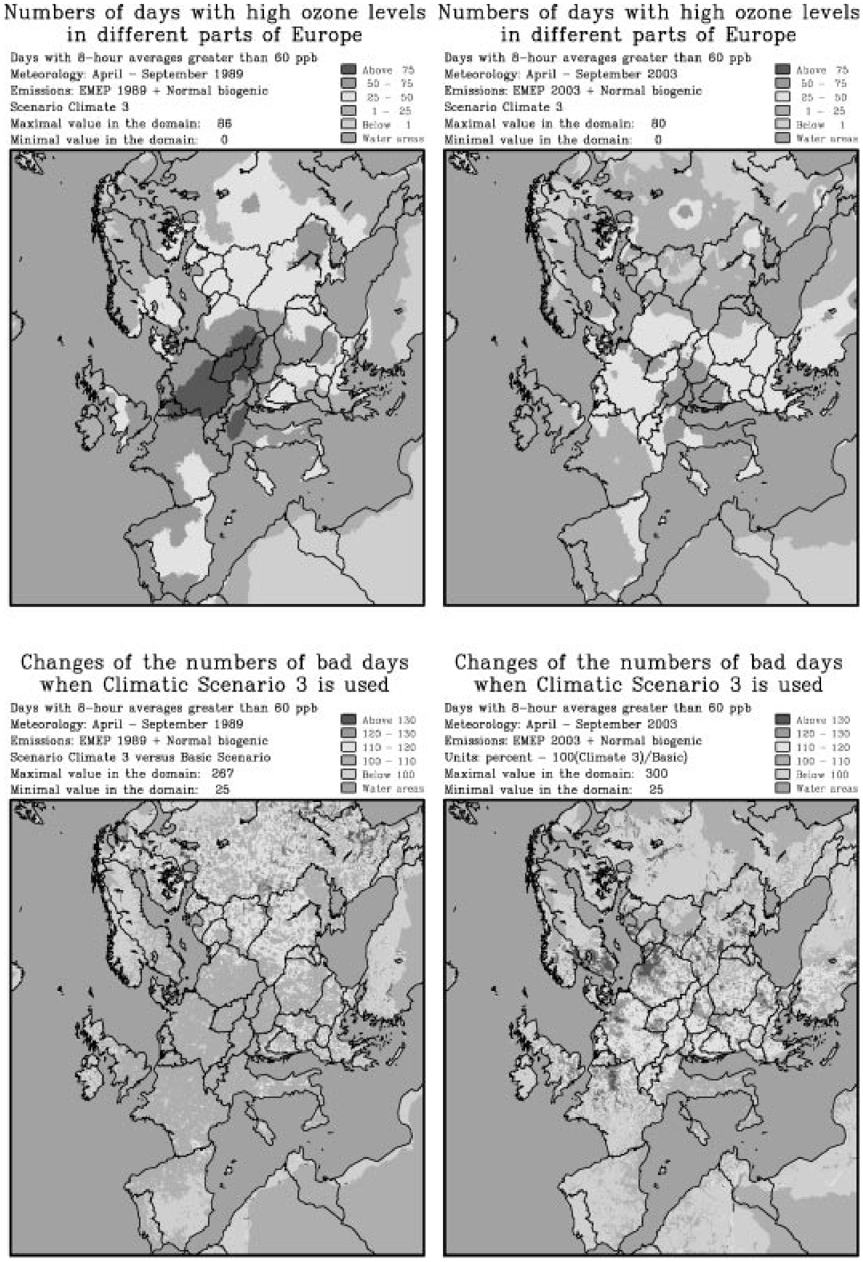

Numbers of days with high ozone levels in different parts of Europe
Days with 8-hour averages greater than 60 ppb
Meteorology: April – September 1989
Emissions: EMEP 1989 + Normal biogenic
Scenario Climate 3
Maximal value in the domain: 86
Minimal value in the domain: 0
Above 75
50 – 75
25 – 50
1 – 25
Below 1
Water areas
Numbers of days with high ozone levels in different parts of Europe
Days with 8-hour averages greater than 60 ppb
Meteorology: April – September 2003
Emissions: EMEP 2003 + Normal biogenic
Scenario Climate 3
Maximal value in the domain: 80
Minimal value in the domain: 0
Above 75
50 – 75
25 – 50
1 – 25
Below 1
Water areas
Changes of the numbers of bad days when Climatic Scenario 3 is used
Days with 8-hour averages greater than 60 ppb
Meteorology: April – September 1989
Emissions: EMEP 1989 + Normal biogenic
Scenario Climate 3 versus Basic Scenario
Maximal value in the domain: 267
Minimal value in the domain: 25
Above 130
120 – 130
110 – 120
100 – 110
Below 100
Water areas
Changes of the numbers of bad days when Climatic Scenario 3 is used
Days with 8-hour averages greater than 60 ppb
Meteorology: April – September 2003
Emissions: EMEP 2003 + Normal biogenic
Units: percent – 100(Climate 3)/Basic)
Maximal value in the domain: 300
Minimal value in the domain: 25
Above 130
120 – 130
110 – 120
100 – 110
Below 100
Water areas

Figure 4. Distributions of the "bad days" in different parts of Europe when Climate Scenario 3 is used (the two upper plots) and changes of the numbers of bad days (in percent) when Scenario Climate 3 is applied instead of the Basic Scenario (the lower two plots). Plots obtained by using meteorological data for 1989 are given on the left-hand side, whereas plots for 2003 are on the right-hand-side.

the results in Fig. 4 and comparing some of them with results from Fig. 1:

- Comparing the results shown in the upper plots of Fig. 4 (obtained by using Scenario Climate 3) with the corresponding plots in Fig. 1 (obtained with the Basic Scenario), it is observed that some regions where the limit of 25 "bad days" is exceeded when the Basic Scenario is used are enlarged when Scenario Climate 3 is applied.
- The previous conclusion was qualitative. Quantitative results can be obtained by taking the ratios of the results obtained by Climatic Scenario 3 and the Basic Scenario (multiplied by 100 in order to get the changes in percent). Results are shown in the lower plots of Fig. 4. It is observed that the climatic changes can cause significant increases of the "bad days" (in some areas of Europe by more than 30%). Therefore, the climatic changes must be taken into account in investigations carried out to define some measures that are to be taken to keep the pollution levels under prescribed acceptable levels.
- Many questions still must be answered. For example, it is not clear (as in the case where the relationship

between biogenic emissions and "bad days" was studied) why the biggest changes in year 1989 are mainly in Eastern Europe, whereas the changes are big also in Central and Western Europe in year 2003. Some reasons, similar to those given when the results in Fig. 1, can also be given here. However, it is more important to emphasize that it is necessary to carry out much more experiments with (*1*) more scenarios, (*2*) finer discretizations, (*3*) better input data, and (*4*) careful and detailed examination of the meteorological conditions in the studied period.

Much more results and conclusions, including results related to the first two climatic scenarios, are given at: http://www2.dmu.dk/atmosphericenvironment/Climate-%20and%20Pollution.

COMBINING THE ENVIRONMENTAL MODELS WITH OTHER LARGE-SCALE MODELS

The environmental phenomena are closely related to other phenomena. Therefore, the environmental models have to be considered together with other models. Several examples are given below to illustrate this statement.

Relationship Between Climatic Changes and Pollution Levels

Some results related to the impact of the climate changes on the "bad days" have been presented in one of the previous sections. It must also be mentioned here that the climate changes have significant influence also on other quantities related to damaging pollutants. Therefore, the climate changes must be considered as an important factor when decisions about measures, which will lead to reductions of harmful pollution levels, are to be taken. Furthermore, the increased pollution levels will cause some feedback effects on the climate. This finding implies that it is desirable to combine high-resolution climatic models with environmental models to study in detail the interrelations between these two processes.

Preparation of Pollution Forecasts on Different Scales

This task involves running several weakly connected large-scale models in order to handle a set of complex problems:

- weather forecasts on different regional scales (starting with results obtained on a global scale),
- weather forecasts on an urban scale (perhaps in parallel for several urban areas),
- pollution forecasts on different regional scales,
- pollution forecasts on an urban scale (perhaps in parallel for several urban areas),
- treatment of the output results in order to prepare them for the people who will use them (data mining algorithms and high-speed visualization tools must be applied at this stage),
- sending the relevant data to appropriate media (television stations, radio stations, Internet sites, GMSs, etc.).

It is clear that computer grids will be very powerful tools in the solution of the very challenging task related to the possibility to treat efficiently the set of problems described above. Currently, such sets of problems are solved only by imposing many simplifying (and very often not physical) assumptions. At the same time, it is also clear that a lot of difficulties must be overcome in the efforts to run such complex tasks efficiently on a computer grid. The greatest difficulties are the tasks of

- achieving reliable and robust transition from one scale to another,
- communicating relevant data from one part of the computational grid to another, and
- preparing the final results, which should be easily understandable by the recipients.

Coupling of Environmental Models with Economical Models

If some critical pollution levels are exceeded, then some measures are to be taken in an attempt to avoid damaging effects. The measures are normally related to some reductions of the human-made (anthropogenic) emissions. The emission reductions can be performed either directly (by using new technologies, filters, etc.) or indirectly (by introducing higher "green" taxes). It is clear, however, that the reductions of the emissions will as a rule cause economical problems; in the worst case, economical crises may be caused by putting too-heavy requirements on emission reductions. Therefore, it is necessary to combine the need to keep the pollution under the critical levels with the need to preserve the rates of the economical development of the region under consideration. It is worthwhile to combine the environmental models with economical models in the effort to achieve both lower pollution levels and sustainable development of the economy. The aim should be to optimize the process by finding out

- where to reduce the emissions and
- by how much to reduce them (the emissions should be reduced as much as needed, but no more than what is absolutely necessary)

to keep the balance between safe environment and sustainable development. The proper treatment of the task of searching an optimal solution with a combination of large-scale environmental and advanced economical models is both very time consuming, and the storage requirements are extremely high. The efficient solution of this task is a very challenging problem.

Formation and Transportation of Aerosol Particles

Aerosol particles are dangerous for health, modifying radiative fluxes, modifying cloud formation, and changing the chemical composition. Therefore, they have been recognized for their potentially negative impact on human health and ecosystems. This finding implied regulatory legislation regarding emissions and concentration levels of particulate matter all over the world. It was also acknowledged that particles play an important role in the global climate by their influence on earth's radiative balance.

The goal of aerosol modeling is to establish a detailed description of the aerosol particle concentrations, their composition, and size distribution. This model requires advanced modeling techniques and innovation as well as reliable validation data of particle characteristics.

Aerosol models may provide a predictive capability for future projections of the outcome of policy strategies on emissions. Consequently, aerosol models are needed that properly describe the cycle of formation, dispersion, and removal of particles. Such validated models can be used as cost-effective tools for reliable studies of the current status and predictions for various environmental and health impacts in the future.

Some extra mathematical terms, which described the transport and the transportation of aerosol particles, have to be added to the model described by Equation (1) when it has to be used in aerosol studies. Moreover, it is necessary to add some equations in Equation (1).

Studying Persistent Organic Pollutants

The persistent organic pollutants (POPs) are chemical compounds with different origins but common characteristics, such as semi-volatility, hydro-phobicity, bioaccumulation, toxicity, potential for long-range transport, and a tendency to accumulate in cold regions ("cold condensation"). POPs may have adverse health effects on humans and wildlife as well as harmful effects on the immune and reproductive systems. Several POPs currently are either banned or regulated through international treaties, but they are still found in the Arctic environment. Models similar to the model described by Equation (1) can be used to study POPs. The treatment of such models leads to huge computational tasks because (*1*) the spatial domains are normally very large (hemi-spherical models and, even better, global models are to be used) and (*2*) fine resolution of the systems of PDEs is highly desirable.

Implementation of Variational Data Assimilation in Environmental Models

Many uncertainties are related to large environmental models. The lack of reliable knowledge for some underlying physical and chemical processes is introducing great uncertainties, but also other reasons are for uncertainties suggested such as, inaccurate input data. Observations can be used to reduce the influence of the uncertainties. The variational data assimilation approach is becoming more and more popular. This approach could be viewed as an attempt to adjust globally the results obtained by a given model to a set of available observations. This approach has the theoretical advantage of providing consistency between the dynamics of the model and the final results of the assimilation. Variational data-assimilation procedures are based on the minimization of certain functional. Assume that (*1*) an improved initial concentration field must be found (it is important when pollution forecasts are to be computed) and (*2*) observations are available at time-points t_p, where $\mathrm{p} \in \{0, 1, 2, \ldots, \mathrm{P}\}$. These observations can be taken into account in an attempt to improve the results obtained by a given model. This effect can be achieved by minimizing the value of the following functional:

$$\mathrm{J}\{\mathrm{c}_0\} = \frac{1}{2}\sum_{\mathrm{p}=0}^{\mathrm{P}} \langle \mathrm{W}(\mathrm{t_p})(\bar{\mathrm{c}}_\mathrm{p} - \bar{\mathrm{c}}_\mathrm{p}^{\mathrm{obs}}),\, \bar{\mathrm{c}}_\mathrm{p} - \bar{\mathrm{c}}_\mathrm{p}^{\mathrm{obs}} \rangle \qquad (2)$$

where $\mathrm{J}\{\bar{\mathrm{c}}_0\}$ depends on the initial value $\bar{\mathrm{c}}_0$ of the vector of the concentrations, $\mathrm{W}(\mathrm{t_p})$ is a matrix containing some weights and $\langle , \rangle$ is an inner product in an appropriately defined Hilbert space (the usual vector space is normally used, i.e., $\bar{c} \in \mathcal{R}^q$ where q is the number of chemical species). The functional $\mathrm{J}\{\bar{\mathrm{c}}_0\}$ depends on both the weights and the differences between calculated by the model concentrations $\bar{c}_p$ and observations $\bar{c}_p^{\mathrm{obs}}$ at the time levels $\{0, 1, \ldots, P\}$ at which observations are available.

Data assimilation can be used not only to improve the initial values (as in the above example) but also in many other tasks (improving the emissions, boundary conditions, the calculated concentrations, etc.). The variational data-assimilation technique should not be considered as a universal tool that can be used in all situations. No measurements are available when different scenarios are used to study the response of the model to the variation of different parameters (emissions, meteorological conditions, climatic changes, etc). We have to rely only on the model in such studies. Therefore, it is absolutely necessary to use data-assimilation techniques not only to improve the model results but also to improve some physical and chemical mechanisms implemented in the model. The hope is that the improved model will provide more reliable results in situations where no measurements are available.

Using Ensembles

Ensembles (based on using results from several different models or results from the same model run by using different parameters, say, different initial values) can be applied in an attempt to improve the reliability of the model. The results are normally averaged. One should expect that the averaged results are better in some sense. This conclusion is based on an assumption (very often not explicitly stated) that no bias is observed in the errors of the results. If, for example, all the models participating in the ensemble procedure strongly overestimate the concentrations of some chemical species, then the results of the ensemble will not be much better because they will also overestimate the species under consideration. Similar conclusions can also be drawn in the case where one model is run with many values of a selected parameter (in this case the set of the values should be carefully chosen). Another problem may develop if one model produces very different results (say, 100 times higher than the other models). In such a case, the "bad" model will spoil the results of the ensemble (it should be eliminated, before the preparation of the ensemble). These examples are provided to show that one must be careful: It is necessary (*1*) to analyze somehow the properties of the models participating in the ensemble procedure or (*2*) to select a good set of values of the parameter which is varied.

The application of ensembles (for different purposes) requires increased computer power. Indeed, the performance of 50–100 runs and preparing an ensemble on the basis of all these runs is a challenging task even for the best high-performance computers currently available. However, the results obtained by using ensembles normally are more reliable than the results obtained in a single run.

READING LIST

Computational and Numerical Background of Environmental Models

V. Alexandrov, W. Owczarz, P. G. Thomsen and Z. Zlatev, Parallel runs of large air pollution models on a grid of SUN computers, *Mathemat. Comput. Simulat.*, **65**: 557–577, 2004.

Z. Zlatev, *Computer Treatment of Large Air Pollution Models*, Dordrecht, The Netherlands: 1995.

Z. Zlatev and I. Dimov, *Computational and Numerical Challenges in Environmental Modelling*, Amsterdam, The Netherlands: Elsevier, 2006.

Model Development, Validation, Policy Making

G. R. Carmichael, A. Sandu, F. A. Potra, V. Damian, and M. Damian, The current state and the future directions in air quality modeling. *Syst. Anal. Modell. Simula.*, **25**: 75–105, 1996.

C. Cuvelier, P. Thunis, R. Vautard, M. Amann, B. Bessagnet, M. Bedogni, R. Berkowicz, J. Brandt, F. Brocheton, P. Builtjes, A. Coppalle, B. Denby, G. Douros, A. Graf, O. Hellmuth, C. Honoré, A. Hodzic, J. Jonson, A. Kerschbaumer, F. de Leeuw, E. Minguzzi, N. Moussiopoulos, C. Pertot, G. Pirovano, L. Rouil, M. Schaap, F. Sauter, R. Stern, L. Tarrason, E. Vignati, M. Volta, L. White, P. Wind, and A. Zuber, CityDelta: A model intercomparison study to explore the impact of emission reductions in European cities in 2010, *Atmospher. Environ.*, **41**: 189–207, 2007.

A. Ebel, Chemical transfer and transport modelling. In: P. Borrell and P. M. Borrell (eds.), *Transport and Chemical Transformation of Pollutants* in the Troposphere, Berlin: Springer, 2000, pp. 85–128.

EMEP Home Page. Available: www.emep.int/emep_description.html.

EURAD - Project Homepage at University of Cologne. Available: www.eurad.uni-koeln.de/index_e.html.

IIASA Home Page. International Institute for Applied Systems Analysis, Laxenburg, Austria. Available: www.educations2u.biz/l12428/IIASA%20Home%20Page.

US Environmental Protection Agency. Available: www.epa.gov/.

Influence of the Biogenic Emission on the Pollution Levels

C. Anastasi, L. Hopkinson, and V. J. Simpson, Natural hydrocarbon emissions in the United Kingdom, *Atmospher. Environ.*, **25A**: 1403–1408, 1991.

V. S. Bouchet, R. Laprise, E. Torlaschi, and J. C. McConnel, Studying ozone climatology with a regional climate model 1. Model description and evaluation, *J. Geophys. Res.*, **104**: 30351–30371, 1999.

V. S. Bouchet, R. Laprise, E. Torlaschi, and J. C. McConnel, Studying ozone climatology with a regional climate model 2. Climatology, *J. Geophys. Res.*, **104**: 30373–30385, 1999.

G. Geernaert and Z. Zlatev, Studying the influence of the biogenic emissions on the AOT40 levels in Europe, *Internat. J. Environ. Pollut.*, **23** (1–2): 29–41, 2004.

D. Simpson, A. Guenther, C. N. Hewitt, and R. Steinbrecher, Biogenic emissions in Europe: I. Estimates and uncertainties, *J. Geophys. Res.*, **100**: 22875–22890, 1995.

Pollution and Climate Changes

IPCC – Intergovernmental Panel on Climate Change. Available: www.ipcc.ch/.

P. Csomos, R. Cuciureanu, G. Dimitriu, I. Dimov, A. Doroshenko, I. Farago, K. Georgiev, Á. Havasi, R. Horvath, S. Margenov, L. Moseholm, Tz. Ostromsky, V. Prusov, D. Syrakov, and Z. Zlatev, *Impact of Climate Changes on Pollution Levels in Europe*. 2006. Available:http://www2.dmu.dk/atmosphericenvironment/Climate%20and%20Pollution,
http://www.cs.elte.hu/~faragois/NATO.pdf,
http://www.umfiasi.ro/NATO.pdf,
http://www.personal.rdg.ac.uk/~sis04itd/MyPapers/climatic_scenarios_NATO.pdf,
http://www.softasap.net/ips/climatic_scenarios_NATO.pdf,
http://www.meteo.bg/bulair/NATO.pdf.

Pollution Forecasts

Air Quality Forecasts for Europe. Available: www.netherlands.globalbioweather.com/pollution.html.

AIRTEXT: Air Pollution Forecasts & Alerts. Available: www.airtext.info/howitworks.html.

EURAD Air Quality Forecast for the Northern Hemisphere, Europe, Germany and Sub-Regions. Available: www.eurad.univ-koeln.de.

PREV'/AIR Air Quality Forecast over the Globe, Europe and France. Available: http://prevair.ineris.fr/en/introduction.php.

Thor – An Integrated Air Pollution Forecast System. Available: http://www2.dmu.dk/1_viden/2_Miljoe-tilstand/3_luft/4_spredningsmodeller/5_Thor/default_en.asp.

UK National Air Quality Archive. Available: www.airquality.co.uk/archive/uk_forecasting/apfuk_home.php.

Environment and Economy

IIASA, International Institute for Applied Systems Analysis, Laxenburg, Austria, *Atmospheric Pollution and Economic Development*. Available: www.iiasa.ac.at/~rains.

Z. Zlatev, I. Dimov, Tz. Ostromsky, G. Geernaert, I. Tzvetanov, and A. Bastrup-Birk, Calculating losses of crops in Denmark caused by high ozone levels, *Environment. Model. Assessm.*, **6**: 35–55, 2001.

Persistent Organic Compounds in the Environment

R. E. Alcock, A. J. Sweetman, C.-Y. Juan, and K. C. Jones, A generic model of human lifetime exposure to persistent organic contaminants: development and application to PCB-101, *Environment. Pollut.*, **110**: 253–265, 2000.

E. Brorström-Lundén, Atmospheric transport and deposition of persistent organic compounds to the sea surface, *J. Sea Res.*, **35**: 81–90, 1996.

K. M. Hansen, J. H. Christensen, J. Brandt, L. M. Frohn, and C. Geels, Modelling atmospheric transport of α-hexachlorocyclohecane in the Northern Hemisphere with a 3-D dynamical model: DEHM-POP, *Atmospher. Chem. Phys.*, **4**: 1125–1137, 2004.

K. M. Hansen, J. H. Christensen, J. Brandt, L. M. Frohn, C. Geels, C. A. Skjøth, and Y.-F. Li, Modeling short-term variability of α-hexachorocyclohexane in Northern Hemispherical air, *J. Geopys. Res.*, **113: D02310, doi 10.1029/2007JD008492,** 2008.

J. A. Van Jaarsveld, W. A. J. van Pul, and F. A. A. M. de Leeuw, Modelling transport and deposition of persistent organic pollutants in the European region, *Atmospher. Environ.*, **31**: 1011–1024, 1997.

Formation and Transport of Aerosols

I. Ackermann, H. Hass, M. Memmesheimer, A. Ebel, F. S. Binkowski, and U. Shankar, Modal aerosol dynamics model for Europe: development and first applications, *Atmospher. Environ.* **32**: 2981–2999, 1998.

F. Binkowski and U. Shankar, The regional particulate model 1: Model description and preliminary results, *J. Geophys. Res.*, **100**: 26191–26209, 1995.

B. Schell, I. J. Ackermann, H. Hass, F. S. Binkowski, and A. Ebel, Modeling the formation of secondary organic aerosol within a comprehensive air quality model system, *J. Geophys. Res.*, **106**: 28275–28294, 2001.

Data Assimilation

D. N. Daescu and I. M. Navon, An analysis of a hybrid optimization method for variational data assimilation, *Internat. J. Computat. Fluid Dynam.*, **17**: 299–306, 2003.

H. Elbern and H. Schmidt, A four-dimensional variational chemistry data assimilation scheme for Eulerian chemistry transport modelling, *J. Geophys. Res.*, **104**: 18583–18598, 1999.

A. Sandu, D. N. Daescu, and G. R. Carmichael, Direct and adjoint sensitivity analysis of chemical kinetic systems with KKP: I. Theory and software tools, *Atmospher. Environ.*, **37**: 5083–5096, 2003.

Using Ensembles in Environmental Modeling

A. Becker, G. Wotawa, L.-E. DeGeer, P. Seibert, R. R. Draxler, C. Sloan, R. D'Amours, M. Hort, H. Glaab, Ph. Heinrich, Y. Grillon, V. Shershakov, K. Katayama, Y. Zhang, P. Stewart, M. Hirtl, M. Jean, and P. Chen, Global backtracking of anthropogenic radionuclides by means of a receptor oriented ensemble dispersion modelling system in support of Nuclear-Test-Ban Treaty verification, *Atmospher. Environ.*, **41**: 4520–4534, 2007.

L. Delle Monache and R. B. Stull, An ensemble air-quality forecast over western Europe during an ozone episode, *Atmospher. Environ.*, **37**: 3469–3474, 2003.

A. Riccio, G. Giunta, and S. Galmarini, Seeking the rational basis of the mefian model: the optimal combination of multi-model ensemble results, Atmospher. Chem. Phys. Discussi., **7**: 5701–5737, 2007.

Legislation Measures in Europe

European Parliament, Directive 2002/3/EC of the European Parliament and the Council of 12 February 2002 relating to ozone in ambient air. *Official J. European Commun.*, **L67**: 14–30, 2002.

UNECE. Protocol to the 1979 Convention on long-range air pollution to abate acidification, eutrophication and ground level ozone. EB.AIR/1999/1, Gothenburg, Sweden, 1999.

Zahari Zlatev
National Environmental
Research Institute
Aarhus University
Roskilde, Denmark

EXPERT DECISION SYSTEM FOR ROBOT SELECTION

INTRODUCTION

Over the past two decades, an upward trend has been observed in the use of industrial robots because of quality, productivity, flexibility, health, and safety reasons. Robots can help manufacturers in virtually every industry to stay globally competitive. Robots can be programmed to keep a constant speed and a predetermined quality when performing a task repetitively. Robots can manage to work under conditions hazardous to human health, such as excessive heat or noise, heavy load, and toxic gases. Therefore, manufacturers prefer to use robots in many industrial applications in which repetitive, difficult, or hazardous tasks need to be performed, such as spot welding, arc welding, machine loading, die casting, forging, plastic molding, spray painting, materials handling, assembly, and inspection. However, a wide selection of robot alternatives and the large number of performance attributes result in a major problem for potential robot users when deciding which robot to purchase.

In the absence of a robust decision aid, the robot selection decisions can be based on the recommendations of robot vendors, the recommendations of an expert hired for performing the evaluation task, or the user's own experience. The recommendations of robot vendors may be biased because they have an inherent interest in selling their product. Basing robot selection decisions on expert advice may be highly costly because experts usually charge considerable amounts for their valuations. Relying on personal experience generally results in selecting a robot with which the user is most familiar, ignoring other crucial factors.

A robot that has the capability of affording heavy load at high speed, as well as good repeatability and accuracy, will contribute positively to the productivity and flexibility of the manufacturing process, which are of high importance when competitive market forces require the introduction of new products into the market. When product design changes need to be made repeatedly, owning a high performing robot will avoid replacement or modification. Many studies reported in the literature address the development of a robust decision tool that enables the potential robot user to select a high-performing robot.

Although it is usually assumed that the specified performance parameters are mutually independent, in general performance parameters provided by robot vendors are not achievable simultaneously. For instance, Offodile and Ugwu (1) reported that for a Unimation PUMA 560 robot manufacturer-specified repeatability deteriorated as the speed increased beyond 50% of the status speed and the weight increased beyond 0.91 kg. Furthermore, it is very difficult to determine the functional relationship between these parameters; thus, making this assumption introduces a risk of selecting a robot that might fail to provide the required performance.

In this article, integration of an expert system and a decision-support system is proposed to enhance the quality and efficiency of the robot selection process. Rather than seeking the advice of an expert or group of experts, the user may consult an expert system to determine the key performance attributes and a short list of acceptable robot alternatives. Expert systems are used to perform a wide variety of complicated tasks that can only be performed by highly trained human experts. Although the problem domain for robot selection may be considered as narrow, which fits the expert system structure, it is also complex and requires a multicriteria decision making (MCDM) procedure. An expert system needs to access the database of a decision-making system to gather factual knowledge. Furthermore, the judgmental data obtained from experts can be incorporated into a decision-making system through an expert system.

A MCDM methodology is required in the expert decision system framework because the expert system provides a subset of robot alternatives based on the technical aspects, and an appropriate MCDM technique evaluates the short list of alternatives and determines the robot that best fits the user requirements.

The proposed decision-support system that employs quality function deployment and fuzzy linear regression integrates user demands with key specifications of robots. The developed decision-making approach has advantages compared with the techniques previously proposed for robot selection. Statistical methods, such as ordinary least squares, are based on determining a meaningful statistical relationship between performance parameters, which is difficult to achieve in practice and the problem aggravates for a small number of candidate robots. Multiattribute decision-making techniques such as multiattribute utility theory (MAUT), analytic hierarchy process (AHP), technique for order preference by similarity to ideal solution (TOPSIS), assume that preferential independence of the performance parameters hold. However, that is a very critical assumption that usually fails to hold in real-world applications. Although fuzzy multiattribute decision-making techniques enable qualitative attributes to be taken into account in an effective manner, they suffer from the same shortcoming as the other multiattribute decision-making techniques. Data envelopment analysis (DEA) does not require the preferential independence assumption of performance parameters. However, DEA assumes that every characteristic defined as output is related to every input. Profiling may be considered as an alternative to tackle that problem, but gathering the efficiency scores into a single efficiency score may also be problematic (2).

INDUSTRIAL ROBOTS

The Robot Institute of America has given the definition of a robot as a reprogrammable, multifunctional manipulator

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks (3).

Before going any further, main parameters used for robot classification are briefly introduced, and major benefits of robot installations are addressed. *Reprogrammability* has made industrial robots a key component of flexible automation. The robot's motion is controlled by a program that can be modified to change the motion of the robot arm significantly. The programmability provides the versatility of a robot.

The basic *geometric configurations* of the robots are usually classified as Cartesian, cylindrical, spherical, and jointed arm. A robot that conforms to a Cartesian geometry can move its gripper to any position within the cube or rectangle work envelope. A robot with a cylindrical geometry can move its gripper within a volume described by a cylinder. The spherical (polar) arm geometry positions the robot through two rotations and one linear actuation. Jointed arm, which are sometimes referred to as articulated robots, have an irregular work envelope. As more flexible and specialized coordinate systems are demanded through time, other robot coordinate systems such as selective compliance assembly robot arm (SCARA), which is particularly used in electronic circuit board assembly applications, have emerged.

Each joint on a robot introduces a *degree of freedom*. In general, a robot with 6 degrees of freedom is required for positioning the tool to a point in space with any orientation. Although a robot with the highest degrees of freedom can produce the most complex movement, one shall consider other factors such as range and quality of motion corresponding to a given degree of freedom.

The *work envelope* is a boundary for the region in which the robot operates determined by the extreme positions of the robot axes. The size of the work envelope defines the limits of reach; thus, it is a key characteristic that needs to be considered in robot selection. Although the reach for a Cartesian configuration is a rectangular-type space, the reach for a cylindrical configuration is a hollow cylindrical space, and the reach for a spherical configuration is a hollow spherical space, respectively; the reach for a jointed arm configuration does not have a specific shape.

The basic types of *power sources* (*drives*) for robots can be named as hydraulic, pneumatic, and electric. The main advantage of hydraulic actuators is that they can afford large load capacity, but they also have many drawbacks, such as possibility of leaks that may be hazardous in certain applications and a high noise level. An important application of hydraulic robots is in spray painting. The pneumatic power source is relatively inexpensive; it enables short cycle times, and leaks do not contaminate the work area, but it has limited positioning capability. Pneumatic robots are frequently used in pick-and-place operations and machine loading. Electric power results in uncontaminated work space, low noise level, and better positioning accuracy and repeatability; however, along with limited load capacity compared with the hydraulic power. Nowadays, the electric drive is the most popular type for general purpose industrial robots.

The *path control* is a means for describing the method that the robot controller employs to guide the tooling through the many points in the desired arm trajectory. The types of path control can be named as point-to-point, controlled path, and continuous path.

Load capacity (*payload*), which denotes the weight-lifting capacity of a robot, is a key parameter that requires careful analysis. In general, the weights that the robots can hold vary with respect to speed. Furthermore, the shape, surface conditions, and positioning of the object held are also important in terms of load capacity. The user of the robot should pay attention to the conditions under which the load capacity is determined by the manufacturer.

Repeatability and *accuracy* are the most easily confused attributes. Repeatability is a measure of the ability of the robot to return to the same position and orientation over and over again, whereas accuracy is a measure of closeness between the robot end-effector and the target point, and it is defined as the distance between the target point and the center of all points to which the robot goes on repeated trials. It is easier to correct poor accuracy than repeatability, and thus, repeatability is generally assumed to be a more critical attribute. Repeatability is a vital feature in justification and use of robots because although the accuracy of human workers may be higher, they tend to operate with less repeatability.

Even though robots have numerous advantages compared with humans in the workplace, one shall not consider a robot as a replacement for a worker in performing all manufacturing tasks. For instance, humans are superior to robots for manufacturing tasks that require intelligence and judgment capabilities. The robots are definitely more efficient, and in certain cases essential, for performing repetitive and highly fatiguing tasks, or for performing applications in environments that are hazardous or dangerous for a human worker to operate. It is also worth noting that a robot can operate three shifts per day for seven days a week with regular maintenance, whereas this schedule would have been impossible for a human worker.

Not only are robots efficient, and in certain cases essential, replacements for humans for performing fatiguing, hazardous, or dangerous tasks, but also they are important for preserving jobs for other workers by increasing productivity.

The major benefits of industrial robots can be named as

- Increased product and market flexibility,
- Increased productivity,
- Improved product quality,
- Shorter cycle times,
- Lower operating costs,
- Higher precision,
- Reduced floor space,
- Elimination of health and safety hazards.

Within the past two decades, the number of robot installations have increased with emphasis on the integration of robots into computer-integrated manufacturing systems.

JUSTIFICATION OF ADVANCED MANUFACTURING SYSTEMS

According to Meredith and Suresh (4), investment justification methods for advanced manufacturing technologies are classified into economic analysis techniques, analytical methods, and strategic approaches. These methods deviate from each other mainly because of the treatment of nonmonetary factors. Economic justification methods for manufacturing investments have been discussed thoroughly in the past couple of decades. Economic analysis methods are the basic discounted cash flow (DCF) techniques, such as present worth, annual worth, internal rate of return, and so on, and other techniques such as payback period and return on investment. It is well known by practitioners who follow the fundamental principles of engineering economy that accounting methods, which ignore time value of money, would produce inaccurate or at best approximate results.

The conventional DCF methods do not seem to be suitable on their own for the evaluation of an advanced manufacturing technology (AMT) investment because of the nonmonetary impacts posed by the system. Sullivan (5) points out the inadequacy of traditional financial justification measures of project worth such as return on investment, payback, and net present worth in considering the strategic merits of advanced manufacturing technologies. The results of the surveys conducted by Lefley and Sarkis (6) for appraisal of AMT investments in the United Kingdom and United States indicate the support for the difficulty in assessing AMT investments because of their nonquantifiable benefits. Because of this difficulty, over 80 % of the respondents in the United States and United Kingdom point out that not all potential benefits of AMT investments are considered in the financial justification process. Improvements in product quality, reliability, productivity, precision, cycle times, and competitiveness as a result of the versatility and flexibility of the system, are the focal points in the justification stage of an AMT investment. Productivity, quality, flexibility, and other intangibles should be examined in terms of potential returns through enhancement of long-term business competitiveness as well as in terms of a comprehensive evaluation of internal costs.

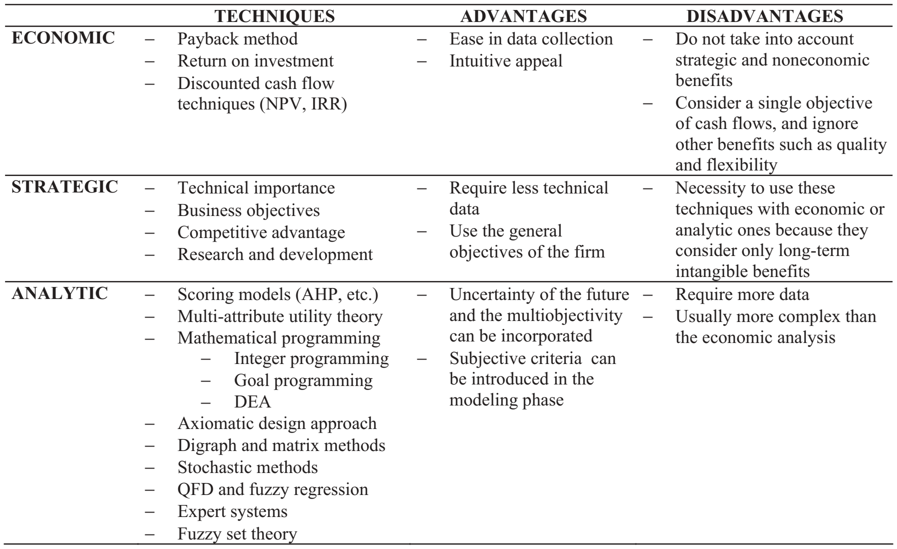
When flexibility, risk, and nonmonetary benefits are expected, and in particular if the probability distributions can be estimated subjectively, analytical procedures may be used. Strategic justification methods are qualitative in nature, and they are concerned with issues such as technical importance, business objectives, and competitive advantage (4). When strategic approaches are employed, the justification is made by considering long-term intangible benefits. Hence, using these techniques with economic or analytical methods would be more appropriate. Figure 1, which is an updated version of the classification provided in Karsak and Tolga (7), resumes the justification methods for advanced manufacturing technologies. Axiomatic design approach (8,9), digraph and matrix methods (10), and QFD and fuzzy regression (2) can be listed as the major updates to the classification given in Ref. 7.

Over the past several decades, manufacturing firms have assigned an increasing importance to robot selection because improper selection of robots can adversely affect their productivity and product quality along with profitability. The increased use of robots and the complexity of the robot evaluation and selection problem have motivated the researchers to develop models and methodologies for making sound decisions.

Mathematical programming, statistical procedures, and fuzzy set theoretic methods, as well as multiattribute and multiobjective decision-making methods can be listed among analytical methods used for robot selection. More recently, axiomatic design approach, digraph and matrix methods, and quality function deployment (QFD) and fuzzy regression have also been proposed as alternative decision aids. Many classifications for models developed for robot selection are available in the literature. For instance, Khouja and Offodile (11) classified models for robot selection as multicriteria decision-making models, performance-optimization models, computer-assisted models, statistical models, and other approaches. Here, we briefly review expert system applications and the use of multicriteria decision making (MCDM) techniques, which possess the potential for considering the multiple and conflicting criteria inherent in the robot selection problem.

Relatively few studies exist regarding the use of expert systems in robot selection. Fisher and Maimon (12) developed a two-phase model for robot selection. In phase 1, an expert system is used to obtain a list of tasks' requirements. The expert system determines a set of robotics technologies and engineering specifications that meet the requirements. In phase 2, a list of candidate robots is chosen and ranked. Boubekri et al. (13) developed a computer-aided system for robot selection, which includes an expert system that considers the functional and organizational parameters specified by the user, an economic feasibility analysis module based on payback period, and a module that provides the user with a detailed description of each robot in the knowledge base. Agrawal et al. (14) employed an expert system to determine the set of important attributes for the particular application and narrow down the robot alternatives, and then used a multiattribute decision-making approach named TOPSIS, which is based on the ideal solution concept, for ranking the shortlist of robot alternatives.

Several articles have focused on the use of MCDM techniques for justification of industrial robots. Imany and Schlesinger (15) compared linear goal programming and ordinary least-squares methods via a robot selection problem in which robots are evaluated based on cost and technical performance measures, which include load capacity, velocity, and repeatability. Liang and Wang (16) presented a robot selection procedure using the concepts of fuzzy set theory. Although providing a multicriteria tool that can incorporate subjective criteria, their approach suffers from the implicit assumption of mutual independence of technical performance parameters. Khouja (17) used DEA and MAUT in a two-phase procedure for robot selection. Baker and Talluri (18) addressed some limitations of the simple radial efficiency scores used for ranking industrial robot alternatives in Khouja's study and proposed a more robust evaluation procedure based on cross-efficiency analysis, which is an extension used for improving the discriminating power of DEA. Goh (19) presented an AHP model for a group of decision makers that considered



	TECHNIQUES	ADVANTAGES	DISADVANTAGES
ECONOMIC	– Payback method – Return on investment – Discounted cash flow techniques (NPV, IRR)	– Ease in data collection – Intuitive appeal	– Do not take into account strategic and noneconomic benefits – Consider a single objective of cash flows, and ignore other benefits such as quality and flexibility
STRATEGIC	– Technical importance – Business objectives – Competitive advantage – Research and development	– Require less technical data – Use the general objectives of the firm	– Necessity to use these techniques with economic or analytic ones because they consider only long-term intangible benefits
ANALYTIC	– Scoring models (AHP, etc.) – Multi-attribute utility theory – Mathematical programming – Integer programming – Goal programming – DEA – Axiomatic design approach – Digraph and matrix methods – Stochastic methods – QFD and fuzzy regression – Expert systems – Fuzzy set theory	– Uncertainty of the future and the multiobjectivity can be incorporated – Subjective criteria can be introduced in the modeling phase	– Require more data – Usually more complex than the economic analysis

Figure 1. Classification of justification methods for advanced manufacturing technologies (7).

both subjective and objective factors for robot selection. Karsak (20) developed a two-phase decision framework that employs DEA in the initial phase to determine the technically efficient robot alternatives and a fuzzy robot selection algorithm in the second phase to rank the technically efficient robots. Parkan and Wu (21) studied the robot selection problem using operational competitiveness rating, TOPSIS, and utility function model, and proposed to rank the robot alternatives based on the averages of the rankings obtained by these three decision aids. Braglia and Petroni (22) proposed the use of DEA with restricted multiplier weights for identifying the optimal robot by considering cost as the single input and engineering attributes as the outputs, and they addressed the advantages and drawbacks of using weight-restriction constraints compared with those of cross-efficiency analysis. Talluri and Yoon (23) proposed a cone-ratio DEA approach for robot selection, which made use of weight-restriction constraints to incorporate a priori information on the priorities of factors. More recently, a practical common weight MCDM methodology with an improved discriminating power has been developed for robot selection (24). The merits of the approach proposed in the paper compared with DEA-based models can be summarized as its ability to evaluate all robot alternatives by common weights for performance attributes that overcome the unrealistic weighting scheme common to DEA resulting from the fact that each decision-making unit (DMU) selects its own factor weights to lie on the efficient frontier, and to further rank DEA-efficient DMUs with a notable saving in computations compared with cross-efficiency analysis.

EXPERT SYSTEM

An expert system is a computer information system developed to act like a human expert in a specific area of knowledge. It is an interactive computer-based decision tool that uses both facts and heuristics to solve difficult decision problems based on an expert's knowledge. Because the knowledge of an expert tends to be domain-specific rather than general, expert systems that represent this knowledge usually reflect the specialized nature of such expertise. Expert systems provide the means for overcoming the shortcomings of conventional human decision-making processes and conventional software through integrating human expertise and power of computers.

Although a generally accepted view on traditional computer program is summarized as

$$\text{Traditional computer program} = \text{Data} + \text{Algorithm},$$

the expert system can be described as

$$\text{Expert system} = \text{Knowledge base} + \text{Inference engine}$$

An expert system consists typically of the following major components:

- *Knowledge base* comprises specific knowledge about the problem domain under consideration. It differs from a database because much knowledge in the knowledge base is represented implicitly. The knowledge is most commonly represented in terms of production rules. A production rule has the following structure:

  IF *conditions*
  THEN *conclusions*

- *Knowledge-acquisition interface* helps experts to express knowledge in a form that can be incorporated in a knowledge base. Determination of the problem domain and its characteristics, identifying the concepts that are used to describe the objects and their interrelationships; acquiring the knowledge; and representing it through suitable representation technique, such as production rules, implementation, and validation, can be listed as the stages of the knowledge acquisition process.

- *Inference engine* employs general rules of inference to arrive at logical conclusions according to the knowledge base of the expert system. Two main inference approaches are used by an inference engine to exploit the knowledge base: forward chaining (data driven) and backward chaining (goal-driven reasoning). Forward chaining begins with data input by the user and scans the rules to find those whose antecedent conditions are fulfilled by the data. It then fires those rules and deduces their consequences. The consequences are added to the knowledge base, and the rules are revisited to observe which new rules may now be fired. This process is repeated until all rules which may be fired have been fired (25). As opposed to forward chaining that is data driven, backward chaining is goal driven because the inference process is guided by the final goal or objective that should be reached rather than by the available information. The process identifies rules that have the goal as a consequence.
- *User interface* is responsible for the form of communicating with the user. User interface attempts to equip the user with most capabilities of interacting with a human expert. The user interface presents the conclusions and explains the reasoning for justification purposes. Most of them may provide sensitivity analysis and what-if analysis tools to observe the changes that would have occurred if the variables had taken different values.

Figure 2 illustrates the structure of an expert system and the interrelationship between its components. Particularly for the cases in which substantial information and data processing and analysis are required, expert systems derive conclusions at a much faster rate compared with human experts. Furthermore, expert systems are apt to deal with incomplete and uncertain information. However, the knowledge acquired in the expert system depends on the expert, and thus, the conclusions obtained are prone to change with knowledge elicited from a different human expert.

MULTICRITERIA DECISION MAKING APPROACH

MCDM addresses the problem of making decisions in the presence of multiple, usually conflicting criteria. MCDM includes two main fields, namely multiattribute decision making (MADM) and multi-objective decision making (MODM). While MADM refers to making preference decisions over a finite number of alternatives, considering multiple and possibly conflicting attributes, MODM aims to design the best alternative given a set of multiple and conflicting objectives. MCDM techniques are applied when a selection/design problem with multiple and usually conflicting attributes/objectives with incommensurable units is encountered. Conflict among attributes/objectives increases in the sense that a favorable value in one may have to be obtained with a poor value in the other. Incommensurable units refer to each criterion that has different units of measurement rendering a direct comparison between different criteria impossible. The basic differences in MADM and MODM problems are provided in Table 1.

Because the decision problem addressed here considers selection among a finite number of alternatives, in the presence of multiple and conflicting attributes with each attribute having different units of measurement, a MADM approach would be required.

In this section, a decision-making approach that integrates quality function deployment and fuzzy linear regression is presented to address the robot selection problem. The delineated procedure is based on the methodology developed by Karsak (2). QFD is a customer-oriented design tool that aims to meet customer requirements in a better way and enhance organizational capabilities while maximizing company goals. A key objective of QFD is to determine directly from the customer what they would

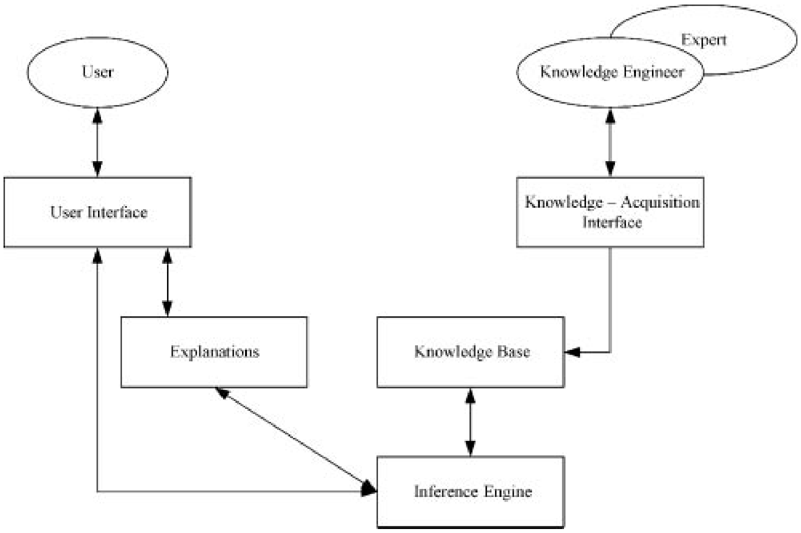


Figure 2. Structure of an expert system.

Table 1. Comparison of MADM and MODM Approaches

	MADM	MODM
Criteria defined by	Attributes	Objectives
Objectives	Implicit	Explicit
Attributes	Explicit	Implicit
Constraints	Implicit	Explicit
Number of alternatives	Finite (small)	Infinite (large)
Interaction with decision-maker	Limited	Significant
Utilization	Selection/evaluation	Design/search

expect from a specific product or service. QFD aims at delivering value by focusing on prioritized customer requirements, translating these into engineering characteristics (design requirements), and then communicating them throughout the organization in a way to assure that details can be quantified and controlled.

Relationships between customer requirements and engineering characteristics and among the engineering characteristics are defined by answering a specific question that corresponds to each cell in a matrix named the house of quality (HOQ). Hauser and Clausing (26) defined the HOQ as a kind of conceptual map that provides the means for interfunctional planning and communications.

Fuzzy linear regression was first introduced by Tanaka et al. (27). As opposed to statistical regression that is based on probability theory, fuzzy regression is founded on possibility theory and fuzzy set theory. In fuzzy regression, regression residuals that denote the deviations between observed values and estimated values are assumed to be caused by imprecise and vague nature of the system. Fuzzy regression has been reported as a more effective tool than statistical regression when the data set is insufficient to support statistical regression, human judgments are involved, and the degree of system fuzziness is high (27).

Fuzzy linear regression is selected as a decision tool for parameter estimation of functional relationships because of its aptness to deal with human expert knowledge, which is an important source in robot selection. First, the ratings for factors such as product quality, manufacturing flexibility, and vendor support that are listed among customer requirements are generally represented by expert judgment. Moreover, the fuzziness inherent in the relationships between customer requirements and robot characteristics, and the dependencies among robot's technical characteristics (i.e., the relationships between performance parameters, such as repeatability, velocity, and load capacity, which are difficult, if possible, to determine precisely) can be expressed effectively using expert judgment.

Over the past decade, some research has been performed on quantifying the planning issues in HOQ, mainly focusing on the interpretation of imprecise design information related to customer requirements and relationships between customer requirements and engineering characteristics. Many authors have used fuzzy set theory to consider the imprecision and vagueness in determining the importance of customer requirements and addressing the relationships between customer requirements and engineering characteristics (28, 29). Few researchers have addressed the development of procedures for setting target levels for engineering characteristics using fuzzy regression and fuzzy optimization (30, 31). Chan and Wu (32) presented a comprehensive review of QFD including quantitative methods applicable to it.

Similar to the process of setting target levels for engineering characteristics in QFD (30, 31), the target values for the robot characteristics can be determined by solving the following formulation:

$$\max z(y_1, y_2, \ldots, y_m) \tag{1}$$

subject to

$$\begin{aligned} y_i &= f_i(x_1, x_2, \ldots, x_n), & i &= 1, 2, \ldots, m \\ x_j &= g_j(x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n), & j &= 1, 2, \ldots, n \end{aligned}$$

where y_i denotes the customer perception of the degree of satisfaction of the ith customer requirement ($i = 1, 2, \ldots, m$), x_j is the normalized target value of the jth robot characteristic ($j = 1, 2, \ldots, n$), f_i represents the functional relationship between the ith customer requirement and the robot characteristics, and g_j denotes the functional relationship between the jth robot characteristic and other robot characteristics. The objective function of Equation (1) can be expressed as

$$z(y_1, y_2, \ldots, y_m) = \sum_{i=1}^{m} w_i (y_i - y_i^{\min})/(y_i^{\max} - y_i^{\min}) \tag{2}$$

where w_i is the relative importance weight of the ith customer requirement and is defined such that $0 < w_i \leq 1$ and $\sum_{i=1}^{m} w_i = 1$, and $y_i^{\min}$ and $y_i^{\max}$ denote the minimum and the maximum possible values, respectively, for the ith customer requirement. Because $(y_i - y_i^{\min})/(y_i^{\max} - y_i^{\min}) \in [0, 1]$, $z(y_1, y_2, \ldots, y_m)$ also takes values between 0 and 1, where 0 and 1 being the worst and the best values, respectively. Thus, Formulation (1) can be represented as

$$\max z(y_1, y_2, \ldots, y_m) = \sum_{i=1}^{m} w_i (y_i - y_i^{\min})/(y_i^{\max} - y_i^{\min}) \tag{3}$$

subject to

$$\begin{aligned} y_i &= f_i(x_1, x_2, \ldots, x_n), & i &= 1, 2, \ldots, m \\ x_j &= g_j(x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n), & j &= 1, 2, \ldots, n \\ y_i^{\min} &\leq y_i \leq y_i^{\max}, & i &= 1, 2, \ldots, m \end{aligned}$$

The information provided in the HOQ can be used to estimate the parameters of the functional relationships f_i and g_j. Because the relationships between customer requirements and robot characteristics and the interactions among robot characteristics are vague and generally difficult to define precisely, fuzzy regression seems to be a sound alternative approach for serving this purpose.

A fuzzy linear regression model is defined as follows (27):

$$\tilde{y} = \tilde{A}_0 + \tilde{A}_1 x_1 + \tilde{A}_2 x_2 + \cdots + \tilde{A}_n x_n \qquad (4)$$

where $\tilde{y}$ is the fuzzy output, x_j are the real-valued independent variables, and $\tilde{A}_j$ are the fuzzy parameters expressed as symmetrical triangular fuzzy numbers with centers α_j and spreads c_j, respectively, which have the membership function given as follows:

$$\mu_{\tilde{A}_j}(a_j) = \begin{cases} 1 - \dfrac{|\alpha_j - a_j|}{c_j}, & \alpha_j - c_j \le a_j \le \alpha_j + c_j \\ 0, & \text{otherwise} \end{cases}$$

Thus, the fuzzy linear regression model can be rewritten as

$$\tilde{y} = (\alpha_0, c_0) + (\alpha_1, c_1)x_1 + (\alpha_2, c_2)x_2 + \cdots + (\alpha_n, c_n)x_n \qquad (5)$$

Fuzzy linear regression determines the fuzzy parameters $\tilde{A}_j$ such that the estimated output has the minimum total spread while satisfying a target degree of belief H, where $0 \le H < 1$. The H value, which is selected by the decision maker, is referred to as the measure of goodness of fit of the estimated fuzzy linear regression model to the data set. To determine the fuzzy parameters $\tilde{A}_j$ the linear programming model given below is solved (33):

$$\min Z = \sum_{j=0}^{n} \left(c_j \sum_{k=1}^{s} |x_{jk}| \right) \qquad (6)$$

subject to

$$\sum_{j=0}^{n} \alpha_j x_{jk} + (1 - H) \left(\sum_{j=0}^{n} c_j |x_{jk}| \right) \ge y_k, k = 1, 2, \ldots, s$$

$$\sum_{j=0}^{n} \alpha_j x_{jk} - (1 - H) \left(\sum_{j=0}^{n} c_j |x_{jk}| \right) \le y_k, k = 1, 2, \ldots, s$$

$$x_{0k} = 1, k = 1, 2, \ldots, s$$

$$c_j \ge 0\,, j = 0, 1, \ldots, n$$

where x_{jk} is the value of the jth independent variable for the kth observation (here, the normalized value of the jth robot characteristic for the kth robot alternative), and y_k is the value of the dependent variable for the kth observation (here, the customer perception of the degree of satisfaction of the customer requirement for the kth robot alternative). The aim of Formulation (6) is to determine $\tilde{A}_j$ in a way to minimize the total fuzziness under the condition that each observation y_k has at least H degree of belonging to its fuzzy estimate [i.e., $\mu_{\tilde{y}_k}(y_k) \ge H$ for $k = 1, 2, \ldots, s$]. Here, both x_{jk} and y_k are crisp numbers for all j and k, and thus the resulting formulation is a conventional linear program. When no fuzziness is considered in the system parameters, only the center value estimates obtained from Formulation (6) are used in Formulation (3), whereas the spreads are ignored (30). Applications of fuzzy linear regression to QFD have been reported in the literature (30, 31).

INTEGRATED KNOWLEDGE-BASED DECISION FRAMEWORK FOR ROBOT SELECTION

Because of the wide selection of robot attributes and candidate robots, expert decision methods are a viable approach for robot selection. After gathering information about the robot application, an expert system is used to provide a list of pertinent attributes and their acceptable values. Based on the analysis of the entire production cell, some robot selection attributes may be ignored whereas others may be considered as critical. Through the integration of an expert system with the database of available robots, a shortlist of robots meeting the minimum acceptable values is determined. The selection of the most suitable robot cannot be accomplished through an inference procedure based on symbolic reasoning. Thus, an appropriate analytical decision making tool needs to be employed to determine the best robot, which results in moving from the expert system part to the decision model base. The basic structure of an integrated knowledge-based decision system is depicted in Fig. 3.

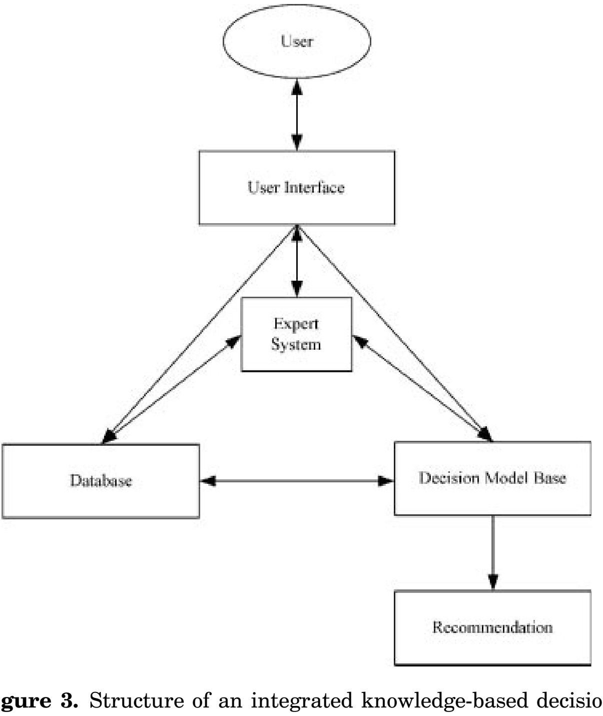


Figure 3. Structure of an integrated knowledge-based decision system.

The procedure is initiated by seeking information from the user about the application for which the robot is required. Potential applications are listed as spot welding, arc welding, machine loading, die casting, forging, plastic molding, spray painting, materials handling, assembly, and inspection. The list may be extended to cover more applications.

When the user identifies the type of application, the expert system is used to determine the set of key attributes for the particular application with their threshold values. Benefiting from the literature on robot selection (11–17) and the expert interviews, the robot parameters taken into consideration can be listed as

- Configuration
- Degrees of freedom
- Vertical reach
- Horizontal reach
- Power source (drive)
- Weight
- Control type
- Accuracy
- Repeatability
- Load capacity
- Velocity
- Programming method
- Programming language
- Memory size

The list of robot attributes given above is not meant to be comprehensive, and it can be easily expanded. Attributes with discrete states are assigned code numbers where each code number denotes the state of the attribute. For example, code numbers from 1 to 7 are used to represent degrees of freedom, where code numbers 1 to 6 denote the respective degrees of freedom, whereas code number 7 corresponds to degrees of freedom greater than or equal to 7. On the other hand, for attributes with values grouped as ranges, code numbers are assigned corresponding to the range in which the value of the respective attribute lies. For example, an attribute whose values are grouped as ranges is given in Table 2.

Table 2. Coding of the Parameter Repeatability

Repeatability (mm)	Code	Significance
Repeatability ≤ 0.025	1	Very good
$0.025 <$ Repeatability ≤ 0.050	2	Good-to-very good
$0.050 <$ Repeatability ≤ 0.100	3	Good
$0.100 <$ Repeatability ≤ 0.200	4	Average-to-good
$0.200 <$ Repeatability ≤ 0.300	5	Average
$0.300 <$ Repeatability ≤ 0.500	6	Poor-to-average
$0.500 <$ Repeatability ≤ 0.750	7	Poor
$0.750 <$ Repeatability ≤ 1.000	8	Very poor-to-poor
Repeatability > 1.000	9	Very poor

For the cases where the user does not specify a code number, the user is reminded by a prompt that an attribute value is not entered. If the user does not possess adequate information regarding the attribute value, then a default value is assigned by the expert system. An example for the list of robot parameters for the robot selection problem is depicted in Fig. 4.

For illustrative purposes, spray painting is considered as the application type. Spray painting, like many other robot applications, presents safety and health hazards while requiring precision. In general, hydraulic or pneumatic robots are employed for spray painting. Spray painting necessitates the use of continuous path control since the cavities of the painted piece-part must be reached by the robot. The expert system is employed to obtain a list of robot alternatives that meet the minimum performance requirements. If none of the robots satisfies the specified requirements, then the user is asked to revise the parameter values. Although the basic structure remains the same for all types of applications, the order of questions may vary for the considered application as key parameters for robot selection may differ according to the application type. A set of sample rules called for the spray painting application can be given as follows:

{Rule p}
IF application is spray painting
AND environment is unclean
THEN drive is hydraulic

{Rule q}
IF application is spray painting

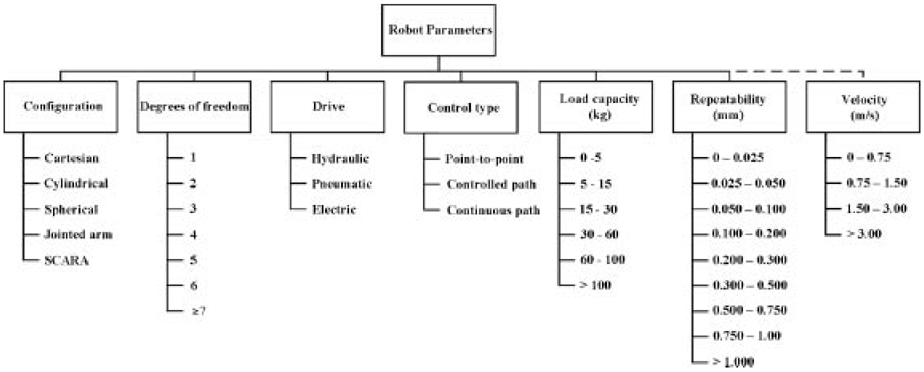


Figure 4. Example of parameters listing for the robot selection problem.

AND drive is hydraulic
THEN min number of degrees of freedom is 5

{Rule r}
IF degrees of freedom is 5
OR degrees of freedom is 6
OR degrees of freedom is ≥ 7
AND load capacity is average
AND repeatability is poor-to-average
AND velocity is average
THEN we have ROBOTS1

{Rule s}
IF we have ROBOTS1
AND configuration is jointed arm
AND control type is continuous path
AND programming is manual dry-run mode
THEN robot set is determined
AND display ROBOTS1A

The decision problem aims to determine the best robot alternative from the short-list of candidates determined using the expert system module by taking into consideration customer requirements (hereafter named as user requirements) and robot characteristics, the relationships between user requirements and robot characteristics, and the interactions between robot characteristics. The algorithm developed to rank the robot alternatives is summarized in Table 3.

Major user requirements for an industrial robot can be denoted as improved product quality, reduced cycle time, improved manufacturing flexibility, easier and standardized programming, improved precision, improved reliability and stability, improved potential to interface with existing equipment, reduced costs, and vendor support.

The prototype HOQ given in Fig. 5 illustrates the user requirements and robot characteristics, and the related data for the short list of robot alternatives. The relationship matrix in the HOQ is used to represent the relationships between the manufacturing firm's demands regarding product quality, manufacturing flexibility, vendor support, and so on, and robot characteristics such as repeatability, velocity, load capacity, and so on. The roof matrix in the HOQ is employed to denote the inner dependence among the robot characteristics. Below the relationship matrix, objective measures (i.e., data related to the abovementioned robot characteristics for each of the short-listed robot alternatives) are indicated. The rightmost part of the HOQ, which captures the user's perspective, presents the data that results from the competitive analysis of the robot alternatives with respect to user requirements. In accordance with customer requirement ratings in earlier studies, performance with respect to user requirements other than "cost" has been scaled from 1 to 5, where 1 and 5 represent the worst and the best, respectively.

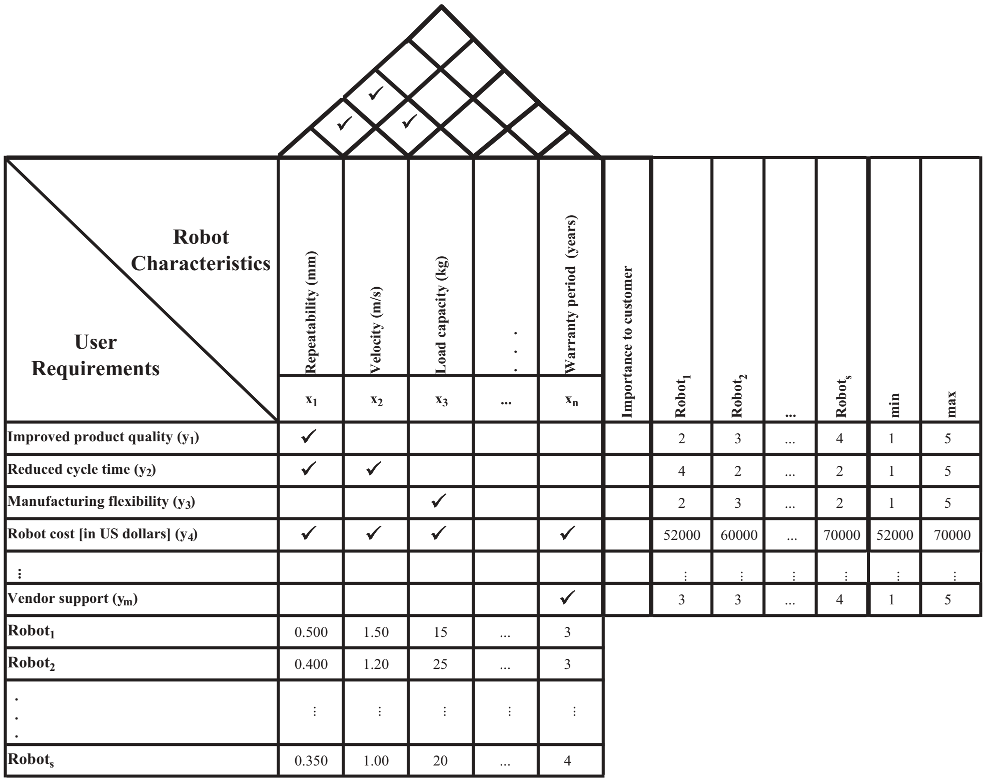
To avoid problems regarding scale differences, data concerning robot characteristics are normalized using a linear normalization procedure. It is obvious that the normalized data lie in the [0,1] interval, and the robot characteristic is more favorable as the normalized data approaches 1. To preserve conformity with other user requirements data that are denoted using a [1, 5] scale, cost data related to robot alternatives are normalized in a way to obtain a value of 5 for the lowest cost robot and a value of 1 for the robot with the highest cost.

The importance weights of the user requirements are determined using the AHP, which has been previously employed for prioritizing customer requirements within QFD framework (2). AHP is a multicriteria decision-making technique that is based on ratio scales and pairwise comparisons. In AHP, the relative importance values are determined using pairwise comparisons with a scale of 1 to 9, where a score of 1 indicates equal importance between the two elements, and 9 represents the extreme importance of one element compared to the other one. The values in between signify varying degrees of importance between these two extremes. Obviously, the weights for the user requirements may vary with respect to the application type of industrial robots.

Then, fuzzy linear regression is employed to estimate the parameters of the functional relationships between user requirements and robot characteristics, and among

Table 3. Stepwise Representation of the Algorithm Employed in the Decision Model Base to Rank the Short List of Robot Alternatives

Step 1.	Obtain the key robot attributes and their threshold values, and the shortlist of acceptable robot alternatives from the expert system module.
Step 2.	Identify the user requirements and other robot characteristics.
Step 3.	Obtain the pertinent robot attributes and other related robot characteristics data for the short list of robot alternatives.
Step 4.	Normalize the data concerning robot characteristics using a linear normalization procedure to avoid problems regarding scale differences.
Step 5.	Determine the relative importance weights of user requirements for the related application type employing the analytic hierarchy process.
Step 6.	Determine the preference ratings of robot alternatives with respect to user requirements.
Step 7.	Identify the relationships between user requirements and robot characteristics, and among the robot characteristics.
Step 8.	Estimate the parameters of the functional relationships between user requirements and robot characteristics, and of the functional relationships among robot characteristics.
Step 9.	Formulate the linear programming model to determine the target values of robot characteristics using the information obtained in previous steps.
Step 10.	Calculate the deviation of each robot alternative from the target robot possessing the optimal robot characteristic values computed in Step 9 using a distance metric based on the p-order relative lower partial moment.
Step 11.	Rank the robot alternatives according to the sum of deviations from the target robot characteristic values. Select the robot alternative that is closest to the target robot.



Robot Characteristics User Requirements	Repeatability (mm) x_1	Velocity (m/s) x_2	Load capacity (kg) x_3	. . . ...	Warranty period (years) x_n	Importance to customer	$Robot_1$	$Robot_2$	...	$Robot_s$	min	max
Improved product quality (y_1)	✓						2	3	...	4	1	5
Reduced cycle time (y_2)	✓	✓					4	2	...	2	1	5
Manufacturing flexibility (y_3)			✓				2	3	...	2	1	5
Robot cost [in US dollars] (y_4)	✓	✓	✓		✓		52000	60000	...	70000	52000	70000
⋮							⋮	⋮	⋮	⋮	⋮	⋮
Vendor support (y_m)					✓		3	3	...	4	1	5
$Robot_1$	0.500	1.50	15	...	3							
$Robot_2$	0.400	1.20	25	...	3							
. . .	⋮	⋮	⋮	⋮	⋮							
$Robot_s$	0.350	1.00	20	...	4							

Figure 5. House of quality for the robot selection problem.

robot characteristics themselves. As in several previous works on fuzzy regression, the H value can be set to 0.5 for the base case. When the data set is sufficiently large H could be set to 0, whereas a higher H value is suggested as the size of the data set becomes smaller (33).

Using the normalized data for the robot selection problem, parameter estimations are obtained by fuzzy linear regression. Because no fuzziness is considered in the system parameters, only the center value estimates obtained from fuzzy regression are employed in formulation (3) to determine the target values for robot characteristics, whereas the spread values are disregarded (31). To determine the ranking order of robot alternatives, the following distance metric is used:

$$d_k^p = \left[\sum_j (\max(0, (x_j^* - x_{jk})))^p \right]^{1/p}, \quad k = 1, 2, \ldots, s \quad (7)$$

where x_j^* is the normalized target value of the jth robot characteristic obtained by solving formulation (3), x_{jk} is the normalized value of the jth robot characteristic for the kth robot alternative, and d_k^p is the distance metric for the kth robot alternative that is based on the p-order relative lower partial moment. The robot alternative with the minimum value of the distance metric is determined as the best robot ($\min_k d_k^p$). Here, use of conventional distance metrics such as the city block distance or the Euclidean distance, which punish the desirable higher than optimal robot characteristics values of the robot alternatives as hard as the lower than optimal ones, would have been inappropriate.

CONCLUSION

An expert system can represent the logic of a human expert who possesses knowledge and experience in a specialized domain. An integrated use of MADM techniques and an expert system is deemed appropriate because the expert system framework typically results in multiple candidates, which needs to be evaluated by an appropriate decision making technique. The proposed integrated framework enables the use of structured decision modeling and techniques suitable for the complex nature of the robot selection problem by means of the decision support module.

The proposed decision framework possesses many advantages compared with the robot selection approaches developed in earlier studies. The primary advantage of expert decision models is to incorporate expert knowledge into a difficult problem. Second, by adopting the QFD principles, the decision-making system allows both user requirements that are generally qualitative and robot characteristics to be considered in the robot selection process. Third, using QFD enables incorporating not only the relationships between user requirements and robot characteristics but also the relationships between robot characteristics disregarding the unrealistic preferential independence assumption frequently encountered in earlier robot selection studies using multiattribute decision making techniques. Furthermore, the parameter estimation of the abovementioned functional relationships is performed using fuzzy regression, which is suitable for considering high system fuzziness.

Although the expert decision model is apt to take into account many robot attributes, one shall note that considering a wide selection of robot attributes results in several rules in the expert system that may prove to be devastating in application. Moreover, the expert system needs to be updated on a regular basis because of the technological changes. It is also worth noting that the expert system delineated in this manuscript illustrates the guidelines of constructing such a system for robot selection, and it by no means intends to serve as a thorough real-world application.

BIBLIOGRAPHY

1. O. F. Offodile and K. Ugwu, Evaluating the effect of speed and payload on robot repeatability, *Robot. Comput.-Integrat. Manufact.*, **8**: 27–33, 1991.
2. E. E. Karsak, Robot selection using an integrated approach based on quality function deployment and fuzzy regression, *Internat. J. Product. Res.*, **46**: 723–738, 2008.
3. W. R. Tanner, *Industrial Robots*, Vols. 1 and 2. Dearborn, MT: Society of Manufacturing Engineers, 1979.
4. J. R. Meredith and N. C. Suresh, Justification techniques for advanced manufacturing technologies, *Internat. J. Product. Res.*, **24**: 1043–1057, 1986.
5. W. G. Sullivan, Models IEs can use to include strategic, nonmonetary factors in automation decisions, *Indust. Engin.*, **42**: 42–50, 1986.
6. F. Lefley and J. Sarkis, Short-termism and the appraisal of AMT capital projects in the US and UK, *Internat. J. Product. Res.*, **35**: 341–368, 1997.
7. E. E. Karsak and E. Tolga, Fuzzy multi-criteria decision-making procedure for evaluating advanced manufacturing system investments, *Internat. J. Product. Econ.*, **69**: 49–64, 2001.
8. O. Kulak and C. Kahraman, Multi-attribute comparison of advanced manufacturing systems using fuzzy vs. crisp axiomatic design approach, *Internat. J. Product. Econ.*, **95**: 415–424, 2005.
9. O. Kulak, A decision support system for fuzzy multi-attribute selection of material handling equipments, *Exp. Sys. Applicat.*, **29**: 310–319, 2005.
10. R. V. Rao, A decision-making framework model for evaluating flexible manufacturing systems using digraph and matrix methods, *Internat. J. Adv. Manufact. Technol.*, **30**: 1101–1110, 2006.
11. M. Khouja and O. F. Offodile, The industrial robots selection problem: literature review and directions for future research, *IIE Trans.*, **26**: 50–61, 1994.
12. E. L. Fisher and O. Z. Maimon, Specification and selection of robots, in A. Kusiak (ed.), *Artificial Intelligence Implications for CIM*. Bedford, UK: IFS Publications, 1988.
13. N. Boubekri, M. Sahoui, and C. Lakrib, Development of an expert system for industrial robot selection, *Comput. Indust. Engineer.*, **20**: 119–127, 1991.
14. V. P. Agrawal, V. Kohli, and S. Gupta, Computer aided robot selection: the 'multiple attribute decision making' approach, *Internat. J. Product. Res.*, **29**: 1629–1644, 1991.
15. M. M. Imany and R. J. Schlesinger, Decision models for robot selection: A comparison of ordinary least squares and linear goal programming methods, *Decision Sci.*, **20**: 40–53, 1989.
16. G. S. Liang and M. J. J. Wang, A fuzzy multi-criteria decision-making approach for robot selection, *Robot. and Comput. Integrat. Manufact.*, **10**: 267–274, 1993.
17. M. Khouja, The use of data envelopment analysis for technology selection, *Comput. Indust. Engineer.*, **28**: 123–132, 1995.
18. R. C. Baker and S. Talluri, A closer look at the use of data envelopment analysis for technology selection, *Comput. Indust. Engineer.*, **32**: 101–108, 1997.
19. C. H. Goh, Analytic hierarchy process for robot selection, *J. Manufactur. Sys.*, **16**: 381–386, 1997.
20. E. E. Karsak, A two-phase robot selection procedure, *Product. Plan. Control*, **9**: 675–684, 1998.
21. C. Parkan and M. L. Wu, Decision-making and performance measurement models with applications to robot selection, *Comput. Indust. Engineer.*, **36**: 503–523, 1999.
22. M. Braglia and A. Petroni, Evaluating and selecting investments in industrial robots, *Internat. J. Product. Res.*, **37**: 4157–4178, 1999.
23. S. Talluri and K. P. Yoon, A cone-ratio DEA approach for AMT justification, *Internat. J. Product. Econ.*, **66**: 119–129, 2000.
24. E. E. Karsak and S. S. Ahiska, Practical common weight multi-criteria decision making approach with an improved discriminating power for technology selection, *Internat. J. Product. Res.*, **43**: 1537–1554, 2005.
25. R. E. Benfer, E. E. Brent, and L. Furbee, *Expert Systems* (Sage University Paper Series on Quantitative Applications in the Social Sciences, 07–077), Newbury Park, CA: Sage, 1991.
26. J. R. Hauser and D. Clausing, The house of quality, *Harvard Bus. Rev.*, **66**: 63–73, 1988.
27. H. Tanaka, S. Uejima, and K. Asai, Linear regression analysis with fuzzy model, *IEEE Trans. Sys., Man, Cybernet.*, **12**: 903–907, 1982.
28. L. K. Chan, H. P. Kao, A. Ng, and M. L. Wu, Rating the importance of customer needs in quality function deployment by fuzzy and entropy methods, *Internat. J. Product. Res.*, **37**: 2499–2518, 1999.
29. E. E. Karsak, Fuzzy multiple objective programming framework to prioritize design requirements in quality function deployment, *Comput. Indust. Engineer.*, **47**: 149–163, 2004.
30. K. J. Kim, H. Moskowitz, A. Dhingra, and G. Evans, Fuzzy multicriteria models for quality function deployment, *Euro. J. Operat. Res.*, **121**: 504–518, 2000.
31. Y. Chen, J. Tang, R. Y. K. Fung, and Z. Ren, Fuzzy regression-based mathematical programming model for quality function deployment, *Internat. J. Product. Res.*, **42**: 1009–1027, 2004.
32. L. K. Chan and M. L. Wu, Quality function deployment: a literature review, *Euro. J. Operat. Res.*, **143**: 463–497, 2002.
33. H. Tanaka and J. Watada, Possibilistic linear systems and their application to the linear regression model, *Fuzzy Sets Syst.*, **27**: 275–289, 1988.

FURTHER READING

C. R. Asfahl, *Robots and Manufacturing Automation*, New York: John Wiley & Sons, 1992.

S. Y. Nof, *Handbook of Industrial Robotics*, New York: John Wiley & Sons, 1999.

E. Ertugrul Karsak
Galatasaray University
Ortakoy, Istanbul, Turkey

GEOGRAPHIC INFORMATION SYSTEMS

A geographic information system (GIS) is a set of computer-based tools to collect, store, retrieve, manipulate, visualize, and analyze geo-spatial information (information identified by its location on the surface of reference, for example, the Earth). Some definitions of GIS include institutions, people, and data, besides the computer-based tools. These definitions refer more to a total GIS implementation than to the technology. Examples of GIS definitions can be found in Maguire (1), Chrisman (2), Foote and Lynch (3) among others. Our definition is discussed next.

Computer-based tools are hardware (equipment) and software (computer programs). *Geo-spatial information* describes facts about the Earth's features, for example, the location and characteristics of rivers, lakes, buildings, and roads. *Collection* of geo-spatial information refers to the process of gathering, in computer-compatible form, facts about features of interest. Facts usually collected are the location of features given by sets of coordinate values (such as latitude, longitude, and sometimes elevation), and attributes such as feature type (e.g., highway), name (e.g., Interstate 71), and unique characteristics (e.g., the northbound lane is closed). *Storing* of geo-spatial information is the process of electronically saving the collected information in permanent computer memory (such as a computer hard disk). Information is saved in structured computer files. These files are sequences of only two characters (0 and 1) called bits, organized into bytes (8 bits) and words (16–64 bits). These bits represent information stored in the binary system. *Retrieving* geo-spatial information is the process of accessing the computer-compatible files, extracting sets of bits, and translating them into information we can understand (for example, information given in our national language). *Manipulation* of geo-spatial data is the process of modifying, copying, or removing selected sets of information bits or complete files from computer permanent memory. *Visualization* of geo-spatial information is the process of generating and displaying a graphic representation of the information, complemented with text and sometimes with audio. *Analysis* of geo-spatial information is the process of studying, computing facts from the geo-spatial information, forecasting, and asking questions (and obtaining answers from the GIS) about features and their relationships. For example, what is the shortest route from my house to my place of work?

HARDWARE AND ITS USE

Computer hardware changes at a very fast pace most all the time. Better and better computers are available every year. This evolution impacts GIS and makes this description difficult in terms of covering what is the "state-of-art" in hardware. A good introduction to GIS hardware is given by UNESCO (4). Our goal here is to overview the major hardware components of GIS without trying to discuss any one in detail. The main component is the computer (or computers) on which the GIS run. Currently, GIS systems run on desktop computers mainframes (used as a stand-alone or as part of a network), and servers connected to the Internet. In general, GIS operations require handling large amounts of information (50 megabytes or larger file sizes are not uncommon), and in many cases, GIS queries and graphic displays must be generated very quickly. Therefore, important characteristics of computers used for GIS are processing speed, quantity of random access memory (RAM), size of permanent storage devices, resolution of display devices, and speed of communication protocols.

Several peripheral hardware components may be part of the system: printers, plotters, scanners, digitizing tables, and other data collection devices. *Printers and plotters* are used to generate text reports and graphics (including maps). High-speed printers with graphics and color capabilities are commonplace today. The number and sophistication of the printers in a GIS organization depend on the amount of text reports and small size (typically 8.5" by 11") maps and graphics to be generated. Plotters allow the generation of oversized graphics. The most common graphic products of a GIS system are maps. As defined by Thompson (5), "Maps are graphic representations of the physical features (natural, artificial, or both) of a part or the whole of the Earth's surface. This representation is made by means of signs and symbols or photographic imagery, at an established scale, on a specified projection, and with the means of orientation indicated." As this definition indicates, there are two different types of maps: (1) line maps, composed of lines, the type of map we are most familiar with, in paper form, for example a road map; and (2) image maps, which are similar to a photograph. A complete discussion of maps is given by Robinson et al. (6). Plotters able to plot only line maps usually are less sophisticated (and less expensive) than those able to plot high-quality line and image maps. Plotting size and resolution are other important characteristics of plotters. With some plotters, it is possible to plot maps with a size larger than 1 m. Higher plotting resolution allows plotting a greater amount of details. Plotting resolution is very important for images. Usually, the larger the map size needed, and the higher the plotting resolution, the more expensive the plotter.

Scanners are devices that sense and decompose a hardcopy image or scene into equal-sized units called pixels and store each pixel in computer-compatible form with corresponding attributes (usually a color value per pixel). The most common use of scanning technology is in fax machines. They take a hardcopy document, sense the document, and generate a set of electric pulses. Sometimes, the fax machine stores the pulses to be transferred later; other times they are transferred right away. In the case of scanners used in GIS, these pulses are stored as bits in a computer file. The image generated is called a raster image. A raster image is composed of pixels. Generally, pixels are

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

square units. Pixel size (the scanner resolution) ranges from a few micrometers (for example, 5 microns) to hundreds of microns (for example, 100 microns). The smaller the pixel size, the better the quality of the scanned images, but the larger the size of the computer file, the higher the scanner cost. Scanners are used in GIS to convert hardcopy documents to computer-compatible form, especially paper maps. Wempen (7) gives a complete discussion of scanning technology.

Some GISs cannot use raster images to answer geospatial questions (queries). Those GISs that can are usually limited in the types of queries they can perform (they can perform queries about individual locations but not geographic features). The reason of this limitation is the lack of explicit information in raster images. Only the location of each pixel in a grid array and a value per pixel (such as color) are the explicit information of raster images. Explicit information is the information that can be expressed without vagueness, implication, or ambiguity, leaving no quetion as to meaning or intent. Computer programs can recognize explicit information. Raster images mainly carry tacit information. Tacit information is information that is difficult to express, often personal or context-speciific, hard to communicate, and even harder to represent in a formal way. In general, computer programs cannot recognize tacit information. Most queries need information in vector form (that carries a lot more explicit information). Vector information represents individual geo-spatial features (or parts of features) and is an ordered list of vertex coordinates and alphanumeric and graphic attributes. Vector information is used for representation and analysis in most GIS. Figure 1 shows the differences between raster and vector.

Digitizing tables are devices that collect vector information from hardcopy documents (especially maps), and they consist of a flat surface on which documents can be attached, and a cursor or puck with several buttons, used to locate and input coordinate values (and sometimes attributes) into the computer. Attributes are commonly input via keyboard. The result of digitizing is a computer file with a list of coordinate values and attributes per feature. This method of digitizing is called “heads-down digitizing.” Digitizing tables were the most common tools for digitizing maps, but their use has decreased in the last decade.

Currently, there is a different technique to generate vector information. This method uses a raster image as a backdrop on the computer terminal. These images are the result of scanning paper maps or derive from digital photos. Usually, the image are geo-referenced (transformed into a coordinate system related in some way to the Earth). The raster images are displayed on the computer screen, and the operator uses the computer mouse to collect the vertices of a geo-spatial feature and to attach attributes (the keyboard or audio may be also used). As in the previous case, the output is a computer file with a list of coordinate values and attributes for each feature. This method is called “heads-up digitizing.” A more in-depth discussion on geospatial data acquisition in vector or raster format is given by GEOWEB (8).

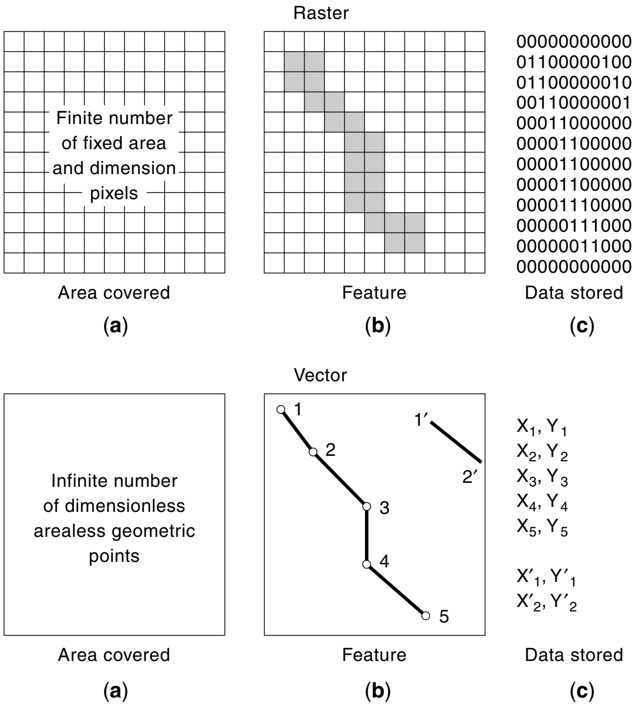


Figure 1. The different structures of raster and vector information, feature representation, and data storage.

SOFTWARE AND ITS USE

Software, as defined by the AGI dictionary (9), is the collection of computer programs, procedures, and rules for the execution of specific tasks on a computer system. A computer program is a logical set of instructions that tells a computer to perform a sequence of tasks. GIS software provides the functions to collect, store, retrieve, manipulate, query and analyze, and visualize geo-spatial information. An important component of software today is a graphical user interface (GUI). A *GUI* is set of graphic tools (icons, buttons, and dialog boxes) that can be used to communicate with a computer program to input, store, retrieve, manipulate, visualize, and analyze information and generate different types of output. Pointing with a device such as a mouse to select a particular software application operates most GUI graphic tools. Voice can also be used in a GUI to communicate with a computer program. Figure 2 shows a GUI.

GIS software can be divided into five major components (besides the GUI): input, manipulation, database management system, query and analysis, and visualization. *Input software* allows the import of geo-spatial information (location and attributes) into the appropriate computer-compatible format. Three different issues need to be considered: how to transform (convert) analog (paper-based) information into digital form, how to accept digital information collected by different devices, and how to store information in the appropriate format. Scanning, as well as heads-down and heads-up digitizing software with different levels of

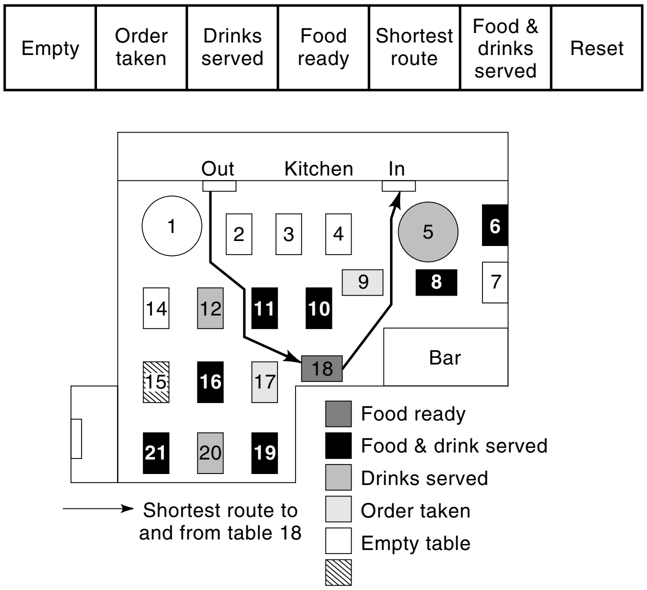


Figure 2. GUI for a GIS in a restaurant setting and the graphic answers to questions about table occupancy, service, and shortest route to Table 18.

automation, transforms paper-based information (especially graphic) into computer-compatible form. Text information (attributes) can be imported by a combination of scanning and character recognition software, or can be imported manually using keyboards or voice recognition software. In general, each commercial GIS software package has a proprietary format used to store locations and attributes. Only information in that particular format can be used in that particular GIS. When information is converted from paper into digital form using the tools from that GIS, the result is in the appropriate format. When information is collected using other devices, then a file format translation needs to be made. Translators are computer programs that take information stored in a given format and generate a new file (with the same or similar information) in a different format. In some cases, translation results in information loss.

Manipulation software allows changing the geo-spatial information by adding, removing, modifying, or duplicating pieces or complete sets of information. Many tools in manipulation software are similar to those in word processors, for example, create, open, and save a file; cut, copy, paste; and undo graphic and attribute information. Many other manipulation tools allow drafting operations of the information, such as drawing parallel lines, square, rectangles, circles, and ellipses; moving graphic elements; and changing colors, line widths, and line styles. Other tools allow the logical connection of different geo-spatial features. For example, geo-spatial features that are physically different and unconnected can be grouped as part of the same layer, level, or overlay (usually, these words have the same meaning), by which they are considered part of a common theme (for example, all rivers in a GIS can be considered part of the same layer: *hydrography*). Then, one can manipulate all features in this layer by a single command. For example, one could change the color of all rivers of the *hydrography* layer from light to dark blue by a single command.

Database management system (DBMS) is a collection of software for organizing information in a database. This software performs three fundamental operations: storage, manipulation, and retrieval of information from the database. A database is a collection of information organized according to a conceptual structure describing the characteristic of the information and the relationship among their corresponding entities (9). In a database, usually at least two computer files or tables and a set of known relationships, which allows efficient access to specific entities, exist. Entities in this concept are geo-spatial objects (such as a road, house, and tree). Multipurpose DBMS are classified into four categories: inverted list, hierarchical, network, and relational. Healy (10) indicates that there are two common approaches to DBMS for GIS: the hybrid and the integrated. The hybrid approach is a combination of a commercial DBMS (usually relational) and direct access operating system files. Positional information (coordinate values) is stored in direct access files and attributes in the commercial DBMS. This approach increases access speed to positional information and takes advantage of DBMS functions, minimizing development costs. Guptill (11) indicates that, in the integrated approach, the standard query language (SQL) used to ask questions about the database is replaced by an expanded SQL with spatial operators able to handle points, lines, polygons, and even more complex structures and graphic queries. This expanded SQL sits on top of the relational database, which simplifies geo-spatial information queries.

Query and analysis software provides new explicit information about the geo-spatial environment. The distinction between query and analysis is somewhat unclear. Maguire and Dangermond (12) indicate that the difference is a matter of emphasis: "Query functions are concerned with inventory questions such as 'Where is. . .?' Analysis functions deal with questions such as 'What if. . .?'." In general, query and analysis use the location of geo-spatial features, distances, directions, and attributes to generate results. Two characteristic operations of query and analysis are buffering and overlay. *Buffering* is the operation that finds and highlights an area of user-defined dimension (a buffer) around a geo-spatial feature (or a portion of a geo-spatial feature) and retrieves information inside the buffer or generates a new feature. *Overlay* is the operation that compares layers. Layers are compared two at a time by location or attributes. Query and analysis use mathematical or logical models to accomplish their objectives. Different GISs may use different mathematical or logical models and, therefore, the results of querying or analyzing the same geo-spatial data in two different GISs may be different.

Mathematical or logical models are of two kinds: (1) Embedded models and (2) external models. Embedded models are the kind of models that are used by any GIS user to perform query and analysis; they are an integral part of a GIS. For example, the models used to perform buffering and overlay are embedded models. Embedded

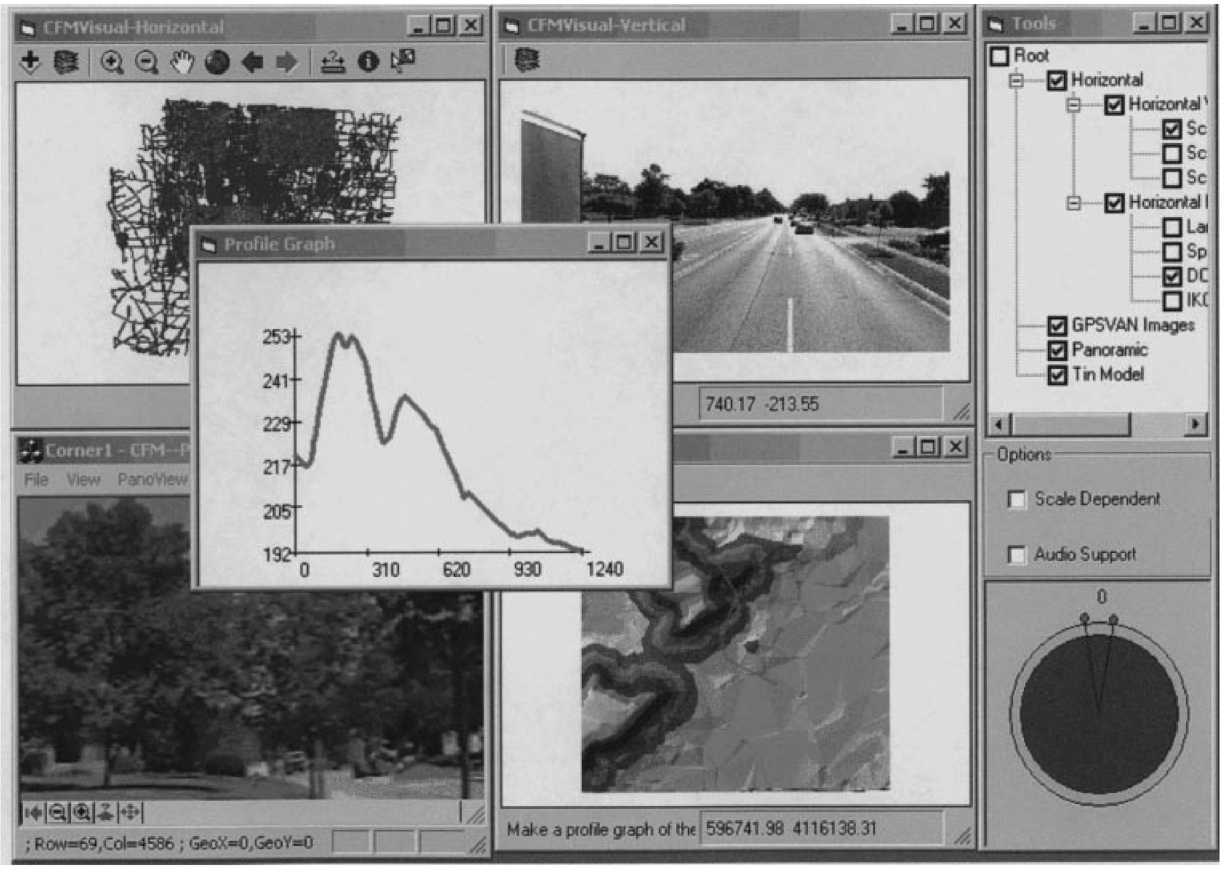


Figure 3.

models in many commercial systems are similar to black boxes: You input the data and you obtain results but, in general, you do not know how these results are generated. External models are mathematical or logical models provided by the user. In some quarters, the use of external models is known as GIS modeling.

There is not a clear distinction between the discipline of scientific modeling and GIS modeling. We would hypothesize that there are two instants of modeling in GIS: (1) when the input of scientific modeling is the outcome of GIS, and GIS is the only way to produce such outcome, and the scientific model can be programmed or interfaced with GIS; (2) When the input of scientific modeling can be collected or generated by means different than GIS, but GIS may be the simple way or the most cost-efficient way to provide the input data or the software implementation of the scientific model. In our opinion, only the first instant should be called GIS modeling. Todorov and Jeffress (13), White et al. (14), and Lauver et al. (15) present examples of GIS modeling. Wilson (16) presents an example of scientific modeling using GIS.

Query and analysis are the capabilities that differentiate GIS from other geographic data applications such as computer-aided mapping, computer-aided drafting (CAD), photogrammetry, and mobile mapping.

Visualization in this context refers to the software for visual representation of geo-spatial data and related facts, facilitating the understanding of geo-spatial phenomena, their analysis, and inter-relations. The term visualization in GIS encompasses a larger meaning. As defined by Buttenfield and Mackaness (17), "visualization is the process of representing information synoptically for the purpose of recognizing, communicating, and interpreting pattern and structure. Its domain encompasses the computational, cognitive, and mechanical aspects of generating, organizing, manipulating, and comprehending such representation. Representation may be rendered symbolically, graphically, or iconically and is most often differentiated from other forms of expression (textual, verbal, or formulaic) by virtue of its synoptic format and with qualities traditionally described by the term 'Gestalt,'" and it is the confluence of computation, cognition, and graphic design.

Traditional visualization in mapping and GIS is accomplished through maps, diagrams, and perspective views. A large amount of information is abstracted into graphic symbols. These symbols are endowed with visual variables (size, value, pattern, color, orientation, and shape) that emphasize differences and similarities among those facts represented. The joint representation of the facts shows explicit and tacit information. Explicit information can be accessed by other means such as tables and text. Tacit information requires, in some cases, performing operations with explicit information, such as computing the distance between two points on a road. In other cases, by looking at the graphic representation, we can access tacit information. For example, we can find an unexpected relationship between the relief and erosion that is not obvious from the explicit information. This example represents the power of visualization!

The most noticeable improvement in GIS recently is in visualization. Multimedia visualization that combines raster, vector, audio, panoramic views, digital video, and so on is gaining acceptance in the GIS community. Experimental systems with these capabilities are being demonstrated in university research centers and by some commercial vendors. Multimedia visualization systems offer the possibility of overcoming many of the problems of traditional visualizations. These systems allows dynamic, multisource, multisense, multiquality, representations of the environment instead of static, single-source, single-sense, single-quality representations. Figure 3 shows a prototype system developed by the Center for Mapping of The Ohio State University.

USING GIS

GIS is widely used. Users include national, state, and local agencies; private business (from delivery companies to restaurants, from engineering to law firms); educational institutions (from universities to school districts, from administrators to researchers); and private citizens. As indicated earlier, the full use of GIS requires software (that can be acquired from a commercial vendor), hardware (which allows running the GIS software), and data (with the information of interest). Partial use of GIS is possible today with access to the Internet. As indicated by Worboys (18), "data are only useful when they are part of a structure of interrelationships that form the context of the data. Such a context is provided by the *data model.*" Depending on the problem of interest, the data model maybe simple or complex. In a restaurant, information about seating arrangement, seating time, drinks, and food are well defined and easily expressed by a simple data model. Fundamentally, you have information for each table about its location, the number of people it seats, and the status of the table (empty or occupied). Once a table is occupied, additional information is recorded: How many people occupy the table? At what time was the table occupied? What drinks were ordered? What food was ordered?. What is the status of the order (drinks are being served, food is being prepared, etc.) Questions are easily answered from the above information with a simple data a model (see Fig. 2) such as follows: What table is empty? How many people can be seated at a table? What table seats seven people? Has the food ordered by table 11 been served? How long before table 11 is free again? Of course, a more sophisticated data model will be required if more complex questions are asked of the system. For example, What is the most efficient route to reach a table based on the current table occupancy? If alcoholic drinks are ordered at a table, how much longer will it be occupied than if nonalcoholic drinks are ordered? How long will it be before food is served to table 11 if the same dish has been ordered nine times in the last few minutes?

Many problems require a complex data model. A nonexhaustive list of GIS applications that require complex models is presented next. This list gives an overview of many fields and applications of GIS:

Siting of a store: Find, based on demographics, the best location in a region for a new store. Retailers collect ZIP codes information, the corresponding sale amount, and the store location for each transaction. This information can be used in a GIS to show the volume of sales coming from each ZIP code region. Using additional information for each ZIP code region, such as income, lifestyle retailers can determine how far a customer is willing to drive to go to a store. This information can be used to determine the best site for a new store.

Network analysis: Find, for a given school, the shortest bus routes to pick up students. School districts use the postal addresses of students, school locations, and student distribution to plan cost-efficient school bus routes. Some of the products of network analysis for school routing are find students homes, bus stops, and schools on maps; assigns students to closest stop; assign stops to a run and runs to a route; identify district boundaries, walk-zones, and hazardous streets; and generate stop times and driver directions for runs.

Utility services: Applications for utility services include service interruption management, emergency response, distribution, network operation, planning, research, sales, engineering, and construction. An electric company, for example, provides services to residential, commercial, government, nonprofit, and others clients. These services are location-based and require a fast response to irregular situations such as an outage. Outage are responded to by priority. Generally, an outage in a hospital requires a faster response than to a residence. Using GIS, this response is efficient and timely.

Land information system: Generate, using land parcels as the basic unit, an inventory of the natural resources of a region and the property-tax revenue. The geo-spatial description of each parcel, their attributes such as owner, area, number of rooms, value, use, and so on, together with the basic geographic features of the region, such as roads, rivers, streams, and lakes; vegetation; political boundaries; and so on, allows the study and analysis of the region.

Automated car navigation: Having a dataset with enough route information such as the geo-spatial description of roads, their speed limit, number of lanes, traffic direction, status of roads, construction projects,and so on, it is possible to use GIS for real-time car navigation. Questions such as: the recommended speeds, the path to be followed, street classification, and route restrictions to go from location A to location B can be answered during navigation.

Tourist information system: Integrating geo-spatial information describing roads and landmarks such as restaurants, hotels, motel gasoline stations, and so on, allows travelers to answer questions such as follows: What is the difference in driving time to go from location A to location B following the scenic route instead of the business route? Where, along the scenic route, are the major places of interest located? How far is the next four-star hotel? How far am I from the next gasoline station? Some systems allow to reserve a hotel room, rent a car, buy tickets to a concert or a movie, and so on, from the route.

Political campaigns: How to maximize funds and to reach the larger sympathetic audience is basic in a political campaign. Based on population information, political trends, cost, and social-economic level, it is possible, for example, to set the most time-efficient schedule to visit the largest possible number of cities where undecided voters could make the difference during the last week of a political campaign.

Marketing branch location analysis: Find, based on population density and consumer preferences, the location and major services to be offered by a new bank branch.

Terrain analysis: Find the most promising site in a region for oil exploration, based on topographic, geological, seismic, and geo-morphological information.

Driving directions: Find how to go from Point A to Point B based on the postal addresses, which is one of the most popular applications of GIS, and one that only requires access to the Internet. Most computer users are familiar with this application. You type the postal address of your departure place and the postal address of your destination. A computer program will generate a set of directions to travel. These instructions will be given by naming the major streets and highways you will drive, indicating how to connect from one to the next, and the distance to be traveled in each segment, and time of traveling (based on the legal speed limit). The program will provide you with written instructions or a map displaying the route to be traveled.

QUALITY AND ITS IMPACT IN GIS

The unique advantage of GIS is the capability to analyze and answer geo-spatial questions. If no geo-spatial data is available for a region, of course, it is not possible to use GIS. On the other hand, the validity of the analysis and quality of the answers in GIS are closely related to the quality of the geo-spatial data used and the quality of the embedded models and the external models. If poor quality or incomplete data were used, the query and analysis would provide poor or incomplete results. The same will happen if the quality of the models was poor. Therefore, it is fundamental to know the quality of the information in a GIS and the quality of the models. Generally, the quality of the embedded models in commercial GIS is unknown. In many cases, a GIS user has no way to know how good the embedded models of the system are, which is problematic in GIS because perfect geo-spatial data used with poor-quality embedded models generates poor results and the user may not be aware of that.

From the viewpoint of data, quality is defined by the U.S. National Committee Digital Cartographic Data Standard (NCDCDS)(19) as "fitness for use." This definition states that quality is a relative term: Data may be fit to use in a particular application but unfit for another. Therefore, we need to have a very good understanding of the scope of our application to judge the quality of the data to be used. The same committee identifies, in the Spatial Data Transfer Standard (SDTS), five quality components in the context of GIS: lineage, positional accuracy, attribute accuracy, logical consistency, and completeness.

SDTS is the U.S. Federal Information Processing Standard–173 and states "lineage is information about the sources and processing history of the data." Positional accuracy is "the correctness of the spatial (geographic) location of features." Attribute accuracy is "the correctness of semantic (nonpositional) information ascribed to spatial (geographic) features." Logical consistency is "the validity of relationships (especially topological ones) encoded in the data," and completeness is "the mapping and selection rules and exhaustiveness of feature representation in the data." The International Cartographic Association (ICA) has added two more quality components: semantic accuracy and temporal information. As indicated by Guptill and Morrison (20), "semantic accuracy describes the number of features, relationships, or attributes that have been correctly encoded in accordance with a set of feature representation rules." Guptill and Morrison (20) also indicate "temporal information describes the date of observation, type of update (creation, modification, deletion, unchanged), and validity periods for spatial (geographic) data records." Most of our understanding about the quality of geo-spatial information is limited to positional accuracy, specifically point positional accuracy. Schmidley (21) has conducted research in line positional accuracy. Research in attribute accuracy has been done mostly in the remote sensing area, and some in GIS (see Chapter 4 of Ref. 20). Very little research has been done in the other quality components (see Ref. 20).

To make the problem worse, because of limited digital vector geo-spatial coverage worldwide, GIS users combine, many times, different sets of geo-spatial information, each set of a different quality level. Most GIS commercial products have no tools to judge the quality of the data used; therefore, it is up to the GIS user to judge and keep track of information quality.

Another limitation of GIS technology today is the fact that GIS systems, including analysis and query tools, are sold as "black boxes." The user provides the geo-spatial data, and the GIS system provides results. In many cases, the methods, algorithms, and implementation techniques are considered proprietary and there is no way for the user to judge their quality. More and more users are starting to recognize the importance of quality GIS data. As result, many experts are conducting research into the different aspects of GIS quality.

Quality of external models usually can be evaluated. Generally, the user knows in detail the external model to be used and can derive means to evaluate its quality. Models can be evaluated by comparing their results with data of higher quality. For example, a rain prediction model can be evaluated by comparing the predicted rain with the actual rain. If this comparison is done enough times, it is possible to have a good estimator of the quality of the model.

THE FUTURE OF GIS

GIS is in its formative years. All types of users have accepted the technology, and it is a worldwide multibillion-dollar industry. This acceptance has created a great demand in digital geo-spatial information and improved technology to be satisfied in the near future. High-resolution (1 meter or less) commercial satellites and multisensor platforms (for example, global position system technology, inertial navigation systems, high-resolution digital images, laser scanners, multispectral, hyperspectral, etc.) generating high-resolution images, positions, attitude, and so on mobile mapping technology generating high-resolution images and geo-spatial positions and attitude; efficient analog-to-digital data conversion systems; and so forth are some of the promising approaches to the generation of geo-spatial data.

At the same time, the use of the Internet is creating new opportunities and new demands in GIS. Opportunities generated by the Internet include allowing access to a very large number of datasets all over the world and World Wide Web mapping. World Wide Web mapping is based on the easy-to-use browser-based format that is both simple and cost-effective to implement, which allows the common individual to use the Web to access maps and GIS-based data. Sophisticated GIS applications become usable by everyone over the Internet.

New demands in GIS generated by the Internet include better and faster analysis and query tools as well as better visualization systems; better tools to access and merge remote data without creating new datasets are needed; an integrated format for raster, vector, video, panoramic views, audio, spectral, multispectral data, and so on is fundamental, which will allow integration of multimedia data into a single format and will simplify the storage and manipulation of geo-spatial data. The Open GIS Consortium will help in satisfying some of the above demands. The Open GIS Consortium is an international industry consortium founded in 1994 by several GIS organizations. The purpose was to address the issue of incompatibility standards in GIS technology. Today, more than 220 companies, government agencies, and universities participate in a consensus process to develop publicly available specifications for interfaces and protocols that enable interoperable geo-processing services, data, and applications.

The vision of the Open GIS Consortium is a "world in which everyone benefits from geographic information and services made available across any network, application, or platform," and its mission "is to deliver spatial interface specifications that are openly available for global use" (22). The Open GIS Consortium envisions the integration of GIS data and technology into mainstream computing and the widespread use of standards-compliant GIS software throughout the information infrastructure. Current specifications from the Open GIS Consortium include (1) Reference Model; (2) Abstract Specification; (3) Implementation Specifications; (4) Recommendation Papers; (5) Discussion Papers; and (6) Conformant Products. The Open GIS Consortium is currently working on eight interoperability initiatives (22), and their effort will continue for several years to come.

GIS capabilities will improve, which is reflected in the large amount of ongoing research, published results, and products and services. This work includes visualization, user interfaces, spatial relation languages, spatial analysis methods, geo-spatial data quality, three-dimensional and spatio-temporal information systems, open GIS software design and access, and more. A search in the Internet of the topic "visualization research" produced than 300,000 hits. Noticeable among them are entries from AT&T Information Visualization Research Group (23) and the Stanford Computer Graphics Laboratory of Stanford University (24). In the field of "user interfaces," a search in the Internet found less than 200 hits. However, there are many professional associations such as the User Interface Engineering, which in 2003 had its eighth Conference. In the case of "Spatial Relation Languages, we received than 20,000 hits in our Internet search. Many interesting topics, such as visual languages for static and dynamic cases; Spatial Query Languages; Spatial reasoning; and so on are found under this topic. In the area of "Spatial Analysis Methods," we found more than 230,000 hits. Spatial analysis has been around for a long time, but GIS makes its use easy. Spatial data mining is a new topic in spatial analysis and generates a lot of interest among researchers. Data mining is discovering knowledge from large databases. As indicated by Ramirez (25), "simply put, data mining is basically a modeling activity. You need to *describe* the data, *build* a predictive model describing a situation you want to investigate based on patterns determined from known results, and *verify* the model. Once these things are done, the model is used to test the data to see what portions of the data satisfy the model. If you find that the model is satisfied, you have discovered something new about your data that is of value to you." We found more than 46,000 hits searching specifically for "Spatial Data Mining" on the Internet. This topic is of great interest that would provide a major payoff to the user of geo-spatial data. Searching for the topic "Geo-Spatial Data Quality," we found more than 2500 hits on the Internet. Many of these hits are related to metadata, but efforts in other aspects of data quality and visualization of geo-spatial quality were also found. The search of "Three-Dimensional and Spatio-Temporal Information Systems" on the Internet was conducted in two steps. We searched for "Three-Dimensional Information Systems" and received than 290,000 hits. We found a large variety of subjects such as machine vision, three-dimensional databases, and three-dimensional display systems that are more or less related to GIS. We also searched for "Spatio-Temporal Information Systems" and received than 16,000 hits. It is obvious that the subject of three-dimensional information systems is more advanced than spatio-temporal systems, but there is ongoing research in both subjects. Finally, in the topic of "Open GIS Software Design and Access," we discussed earlier the work of the Open GIS Consortium that is the best link to this topic. These research and development efforts will result in better, reliable, faster, and more powerful GIS.

Several peripheral hardware components may be part of the system: printers, plotters, scanners, digitizing tables, and other data collection devices. *Printers* and *plotters* are used to generate text reports and graphics (including maps). High-speed printers with graphics and color capabilities are commonplace today. The number and sophistication of the printers in a GIS organization depend on the amount of text reports to be generated. Plotters allow the generation of oversized graphics. The most common graphic products of a GIS system are maps. As defined by Thompson (1), "Maps are graphic representations of the physical features (natural, artificial, or both) of a part or the whole of the earth's surface. This representation is made by means of signs and symbols or photographic imagery, at an established scale, on a specified projection, and with the means of orientation indicated." As this definition indicates, there are two different types of maps: (1) line maps, composed of lines, the type of map we are most familiar with, usually in paper form, for example a road map; and (2) image maps, which are similar to a photograph. Plotters able to plot only line maps are usually less sophisticated (and less expensive) than those able to plot high-quality line and image maps. Plotting size and resolution are other important characteristics of plotters. With some plotters it is possible to plot maps with a size larger than one meter. Higher plotting resolution allows plotting a greater amount of details. Plotting resolution is very important for images. Usually, the larger the map size needed, and the higher the plotting resolution, the more expensive the plotter.

Scanners are devices that sense and decompose a hardcopy image or scene into equal-sized units called pixels and store each pixel in computer-compatible form with corresponding attributes (usually a color value per pixel). The most common use of scanning technology is in fax machines. They take a hardcopy document, sense the document, and generate a set of electric pulses. Sometimes, the fax machine stores the pulses to be transferred later; other times they are transferred right away. In the case of scanners used in GIS, these pulses are stored as bits in a computer file. The image generated is called a raster image. A raster image is composed of pixels. Generally, pixels are square units. Pixel size (the scanner resolution) ranges from a few micrometers (for example, five) to hundreds of micrometers (for example, 100 micrometers). The smaller the pixel size the better the quality of the scanned images, but the larger the size of the computer file and higher the scanner cost. Scanners are used in GIS to convert hardcopy documents to computer-compatible form, especially paper maps.

Some GIS cannot use raster images to answer geographic questions (queries). Those GIS that can are usually limited in the types of queries they can perform (they can perform queries about individual locations but not geographic features). Most queries need information in vector form. Vector information represents individual geographic features (or parts of features) and is an ordered list of vertex coordinates. Figure 1 shows the differences between raster and vector. *Digitizing tables* are devices that collect vector information from hard-copy documents (especially maps). They consist of a flat surface on which documents can be attached and a cursor or puck with several buttons, used to locate and input coordinate values (and sometimes attributes) into the computer. The result of digitizing is a computer file with a list of coordinate values and attributes per feature. This method of digitizing is called "heads-down digitizing."

Currently, there is a different technique to generate vector information. This method uses a raster image as a backdrop on the computer terminal. Usually, the image has been geo-referenced (transformed into a coordinate system related in some way to the earth). The operator uses the computer mouse to collect the vertices of a geographic feature and to attach attributes. As in the previous case, the output is a computer file with a list of coordinate values and attributes for each feature. This method is called "heads-up digitizing."

SOFTWARE AND ITS USE

Software, as defined by the AGI dictionary (2), is the collection of computer programs, procedures, and rules for the execution of specific tasks on a computer system. A computer program is a logical set of instructions, which tells a computer to perform a sequence of tasks. GIS software provides the functions to collect, store, retrieve, manipulate, query and analyze, and display geographic information. An important component of software today is a graphical user interface (GUI). A GUI is set of graphic tools (icons, buttons, and dialogue boxes) that can be used to communicate with a computer program to input, store, retrieve, manipulate, display, and analyze information and generate different types of output. Pointing with a device such as a mouse to select a particular software application operates most GUI graphic tools. Figure 2 shows a GUI.

GIS software can be divided into five major components (besides the GUI): input, manipulation, database management system, query and analysis, and visualization. *Input software* allows the import of geographic information (location and attributes) into the appropriate computer-compatible format. Two different issues need to be considered: how to transform (convert) analog (paper-based) information into digital form, and how to store information in the appropriate format. Scanning, and heads-down and heads-up digitizing software with different levels of automation, transforms paper-based information (especially graphic) into computer-compatible form. Text information (attributes) can be imported by a combination of scanning and character recognition software, and/ or by manual input using a keyboard and/or voice recognition software. In general, each commercial GIS software package has a proprietary format, used to store locations and attributes. Only information in that particular format can be used in that particular GIS. When information is converted from paper into digital form using the tools from that GIS, the result is in the appropriate format. When information is collected using other alternatives, then a file format translation needs to be made. Translators are computer programs that take information stored in a given format and generate a new file (with the same information) in a different format. In some cases, translation results in information loss.

Manipulation software allows changing the geographic information by adding, removing, modifying, or duplicating pieces or complete sets of information. Many tools in manipulation software are similar to those in word-processors: create, open, and save a file; cut, copy, paste, undo graphic and attribute information. Many other manipulation tools allow drafting operations of the information, such as: draw a parallel line, square, rectangle, circle, and ellipse; move a graphic element, change color, line width, line style. Other tools allow the logical connection of different geographic features. For example, geographic features that are physically different and unconnected, can be grouped as part of the same layer, level, or overlay (usually, these words have the same meaning). By doing this, they are considered part of a common theme (for example, all rivers in a GIS can be considered part of the same layer: *hydrography*). Then, one can manipulate all features in this layer by a single command. For example, one could change the color of all rivers of the hydrography layer from light to dark blue by a single command.

Database management system (DBMS) is a collection of software for organizing information in a database. This software performs three fundamental operations: storage, manipulation, and retrieval of information from the database. A database is a collection of information organized according to a conceptual structure describing the characteristic of the information and the relationship among their corresponding entities (2). Usually, in a database there are at least two computer files or tables and a set of known relationships, which allows efficient access to specific entities. Entities in this concept are geographic objects (such as a road, house, and tree). Multipurpose DBMS are classified into four categories: inverted list, hierarchical, network, and relational. Healy (3) indicates that for GIS, there are two common approaches to DBMS: the hybrid and the integrated. The hybrid approach is a combination of a commercial DBMS (usually, relational) and direct access operating system files. Positional information (coordinate values) is stored in direct access files and attributes, in the commercial DBMS. This approach increases access speed to positional information and takes advantage of DBMS functions, minimizing development costs. Guptill (4) indicates that in the integrated approach the Standard Query Language (SQL) used to ask questions about the database is

BIBLIOGRAPHY

1. D. J. Maguire, The history of GIS, in D. J. Maguire, M. F. Goodchild, and D. W. Rhind (eds.), *Geographical Information Systems*, Harlow, U.K.: Logman Scientific Group, l991.
2. Chrisman A Revised Information of Geographic Information Systems. University of Washington, 1998. Available: http://faculty.washington.edu/chrisman/G460/NewDef.html.
3. K. E. Foote and M. Lynch. Geographic Information Systems as an Integrating Technology: Context, Concepts, and Definitions, University of Texas, 1997. Available: http://www.colorado.edu/geography/gcraft/notes/intro/intro.html.
4. UNESCO. UNESCO Hardware Requirement, 1999. Available: http://gea.zyne.fer.hr/module_a/module_a6.html.
5. M. M. Thompson, *Maps for America*, 2nd ed. Reston, Virginia: U.S. Geological Suervey, 1981. p. 253.
6. A. H. Robinson, J. L. Morrison, P. C. Muehrcke, A. J. Kimerling, and S. C. Guptill, *Elements of Cartography* 6th ed. New York, Wiley, 1995.
7. F. Wempen. Unlock the secrets of scanner technology, 2002. Available: http://www.techrepublic.com/article_guest.jhtml?-id=r00320020311fair01.htm&fromtm=e015.
8. GEOWEB Spatial Data Acquisition – Specific Theory. Department of Geomatics, The University of Melbourne, 2000. Available: http://www.sli.unimelb.edu.au/gisweb/SDEModule/SDETheory.doc.
9. Association for Geographic Information *AGI GIS Dictionary* 2nd ed., 1993. Available: http://www.geo.ed.ac.uk/agidexe/term/638.
10. R. G. Healey, Database management systems, in D. J. Maguire, M. F. Goddchild, and D. W. Rhind (eds.), *Geographical Information Systems*, Harlow, U.K.: Logman Scientific Group, l991.
11. S. C. Guptill, Desirable characteristics of a spatial database management system, *Proceedings of AUTOCARTO 8*, ASPRS, falls Church, Virginia1987.
12. D. J. Maguire and J. Dangermond, The functionality of GIS, D. J. Maguire, M. F. Goodchild, and D. W. Rhind (eds.), *Geographical Information Systems*, Harlow U.K.: Logman Scientific Group, l991.
13. N. Todorov and G. Jeffress GIS Modeling of Nursing Workforce and Health-Risk Profiles, Available: http://www.spatial.maine.edu/ucgis/testproc/todorov.
14. W. S. White, P. J. Mizgalewich, D. R. Maidment, and M. K. Ridd, GIS Modeling and Visualization of the Water Balance During the 1993 Midwest Floods, *Proceedings AWRA Symposium on GIS and Water Resources*, Ft. Lauderdale, Florida, 1996.
15. C. L. Lauver, W. H. Busby, and J. L. Whistler, Testing a GIS model of habitat suitable for a decling grassland bird, *Environment. Manage.*, **30**(1): 88–97, 2002.
16. J. P. Wilson, GIS-based Land Surface/Subsurface Modeling: New Potential for New Models? *Proceedings of the Third International Conference/Workshop on Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, 1996.
17. B. P. Buttenfield and W. A. Mackaness, Visualization, in D. J. Maguire, M. F. Goodchild, and D. W. Rhind (ed), *Geographical Information Systems*, Harlow, U.K.: Logman Scientific Group, l991.
18. M. F. Worboys, *GIS: A Computing Perspective*, London: Taylor & Francis, 1995, p. 2.

19. Digital Cartographic Data Standard Task Force, The proposed standard for digital cartographic data, *The American Cartographer* **15**: 9–140, 1988.
20. S. C. Guptill and J. L. Morrison, *Elements of Spatial Data Quality*, Kidlington, U.K.: Elsevier Science, 1995.
21. R. W. Schmidley, *Framework for the Control of Quality in Automated Mapping*, Unpublished dissertation, The Ohio State University, Columbus, Ohio, 1996.
22. OGC. Open GIS Consortium (2003, June 28)). Open GIS Consortium, Inc., Available: http://www.opengis.org/.
23. AT&T. AT&T AT&T Information Visualization Research Group, 2003. Available: http://www.research,att.com/areas/visualization/projects_software/index.html, [2003. June 29].
24. Stanford University. Stanford University Stanford Computer Graphics Laboratory, 2003. Available: http://www.graphics.-stanford.edu/.
25. J. R. Ramirez, A user-friendly data mining system, *Proceedings 20th International Cartographic Conference*, Beijing, China, 2001, pp 1613–1622.

J. Raul Ramirez
The Ohio State University
Columbus, Ohio

HOME AUTOMATION

HOME AUTOMATION

It needs to be noted that home automation systems are intended for homes, so they do not usually address the issues of working environment, multiparty cooperation, ergonomics, and floor planning that are usually the problems addressed in the intelligent building design literature.

Home developers and builders are offering community linkage and links with schools in their new construction projects. Thus, the physical community is connected to the virtual community. The creation of community centers (let them be physical or virtual) is the end result of such efforts.

Home automation systems in various forms have appeared in the market for many years. Thus, we have seen many intelligent security systems, energy management units, lighting controllers, entertainment systems, and so on. Interfacing of these products has been limited, however, and has been usually rather costly, especially in the U.S. market. Some products have received a wide market acceptance and have become de facto standards in a limited home automation market.

Home automation products can, in general, be categorized as follows:

- Interactive smart products
- Intelligent subsystems
- Central automation systems

Most of us have extensively used interactive smart systems—that is, devices that previously required manual control but now have a wide set of programmable features. The cases of programmable video cassette recorders (VCRs), automated door openers, and automated sprinkler systems fall into this category. Intelligent subsystems consist of two or more interactive smart systems that are able to exchange information to accomplish more sophisticated tasks. The interaction between a TV and a programmable VCR falls into this category, as well as an interface of a telephone answering machine with the lighting or the security system. The ultimate and most comprehensive home automation system would be one that integrates a number of smart systems or intelligent subsystems into a system that can be thoroughly and seamlessly controlled by the home owner. Such a system would provide a comprehensive system of home information, telecommunication, entertainment, and control.

Several advantages are realized through the use of such an integrated system. A smart microwave can have its cooking schedule controlled through a central database that stores all the home inhabitants' schedules and habits. A VCR can record only the satellite or cable TV programs that the users like or allow to be viewed and then selectively broadcast them to the TV sets in the house. An integrated security system can be linked with video cameras, the VCR, the telephone network, and the local police station. A smoke detector can be linked to the heating, ventilating, and air conditioning system, and to lighting controls so that, in case a fire breaks out, smoke can be cleared and hallways can be appropriately illuminated to help people move out of the house.

Having such a system with so many differing applications brings forth a wealth of problems in terms of the required integration. High-definition video requires several megahertz of bandwidth, whereas a room thermostat requires a minimum bandwidth occasionally. High-fidelity audio or video traffic requires very strict limits on delays, whereas a washing machine control signal does not have these requirements.

From Home Automation to Intelligent Buildings

Advances in hardware and software technology have affected not only the home automation market but the market of intelligent buildings as well. Intelligent buildings is a term used to describe buildings that are not passive toward their occupants and the activities that take place in them but can program their own systems and manage the consumption of energy and materials. In an intelligent building, sensors receive information on the status of the building and, through the communication system of the building, transfer it to a central controller where, after the necessary comparisons and processing, actions are taken. An intelligent building consists of the peripheral units, the units that monitor the proper functioning of the equipment and regulate it if needed, and the field elements—that is, the sensors, indicators, and activators present in the building.

APPLICATIONS

Several applications have been envisioned by designers of home automation systems and standards organizations. The following categories of applications have been presented in the literature:

- Control of homes' heating, lighting, windows, doors, screens, and major appliances via a TV or TV-like screen.
- Remote control of the house environment via a touch-tone key telephone.
- Detectors to identify rooms that have been empty for more than a specified period of time and possibly transfer this information to the security system or regulate the heating of the room.
- Help for the elderly and disabled.

In the initial phases of research and development efforts, the following applications were identified:

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

- Load management;
- Domestic appliance system;
- Environment control;
- Lighting control;
- Security;
- Safety;
- Access control;
- Voice communication;
- Data communication (including telecontrol); and
- Entertainment.

Several other applications that can make use of the communications that exist outside the home include:

- Home banking;
- Information services;
- Working from home;
- Health monitoring (health check, health security);
- Telecontrol (appliances security heating, video recording); and
- Telemetering (gas, electricity, water).

Looking at the previously presented classifications of applications, one sees that there is a big difficulty in finding and imposing the most appropriate classification and identifying non-overlapping definitions, and then identifying functional links between different applications. Entertainment applications usually receive the most attention in standardization activities and market products because a large market already exists that has been accustomed to integration and common formats. Thus, the integration of audio devices such as DAT players, record players, cassette players, CD/DVD players, radio tuners, microphones, headphones, and remote controls has seen a very large market. The same concepts apply to video equipment; that is, the integration of TV display screens, VCRs, TV tuners, video cameras, video disk players, DVD players, video printers, and satellite dish platforms through a common interface has received considerable attention.

Security applications are the most advanced applications at homes today in terms of providing an integration of controller sensors, actuators, video camera, camera platform, microphones, door phone, push buttons/key access, and timers.

A considerable number of electric utilities have been involved with using advanced techniques of home automation for load management.

PRODUCTS AND STANDARDS

As in many other industries, home automation products were first introduced before a complete set of standards was specified. So in tracing the market and product development, we see a large number of products that do not follow any standard specifications but are absolutely proprietary.

Lonworks

For designers who will be involved in home automation designs, companies like Texas Instruments, Motorola, and Toshiba have been very active in developing the tools and components that will make this process easier.

Home automation systems have borrowed extensively from the developments in the networking community. The idea of using a local area network (LAN) to control and connect devices was implemented in Echelon's Lonworks. Lonworks is based on a distributed control LAN using its local operating network (LON). Communications media, network communication protocols, and application software are integrated. The LAN implements a predictive p-persistent CSMA protocol and can handle rates up to 1.25 Mbps. In the physical layer, transceivers for a variety of media are offered. The Neuron C application language, an extension of ANSI C, adds several features that allow efficient input/output (I/O) operations and efficient network management.

International efforts have been under way to develop standards covering the communication between home automation system modules. Most of these efforts use a LAN environment and follow standard layered approaches, such as the ones advocated by OSI.

CEBus

In the United States, the Electronic Industry Association (EIA) recognized the need to develop standards covering all aspects of home automation systems communication. A committee was organized in 1983 to carry out the task. In 1988, a home automation system communication standard known as CEBus (consumer electronic bus) was made available by the EIA committee for comments. It was upgraded and re-released in December 1989 after undergoing several changes. A final document became available in 1992 (1). The CEBus document covers the electrical and procedural characteristics of systems modules communication. The CEBus powerline technology was one of the first attempts to transport messages between household devices, using the 110–I20VAC electrical wiring in U.S. households. More than 400 companies have occasionally attended the CEBus committee meetings, providing a comprehensive standard, intended for the consumer electronics industry. The main objectives of CEBus have been:

- Low-cost implementation;
- Home automation for retrofit into existing cabling networks;
- To define minimum subsets per appliance intelligence and functional requirements;
- Distributed communication and control strategy;
- Basic plug-and-play functionality allowing devices to be added or removed from the network without interrupting the communication of other subsystems; and
- To accommodate a variety of physical media.

However, CEBus faced only the home automation area and never offered truly multimedia capabilities. In late

1995, CEBus became part of an umbrella standard known as Home Plug and Play (HPnP).

Home Plug and Play

Additions to the application layer of the original CEBus standards have been made in order to create the HPnP specification, transforming standalone products into interactive network products. This specification is expected to make systems easier to install and combine in a reliable in-home network. Among the objectives to be covered by HPnP standards is transport protocol independence, so more than one networking protocol can be used in the same home.

HPnP has three object types: status, listener, and request objects, which adapt the system in which the status information is given to the other systems. By the use of these objects, products from different producers can be used without detailed knowledge of their inner workings.

An important feature of HPnP is that it enables consumers to install more complex systems incrementally without complicating their use or requiring burdensome upgrades.

X.10

Like CEBus, the X.10 specification defines a communication "language" that allows compatible home appliances to talk to each other based on assigned addresses. X.10 is a broadcasting protocol. When an X.10 transmitter sends a message, any X.10 receiver plugged into the household power line tree receives and processes the signal, and responds only if the message carries its address. X.10 enables up to 256 devices to be uniquely addressed, while more than one device can be addressed simultaneously if they are assigned the same address.

HBS

The Japanese home bus system (HBS) has been developed as the national standard in Japan for home automation after several years of research and trials. HBS uses a frequency-division-multiplexing system using coaxial cable. Three bands are used for transmission of control signals: baseband, for high-speed data terminals; subband; and, for transmission of visual information, the FM-TV band. Recent efforts have concentrated on the expansion of the traditional idea of a home automation system into one that incorporates multimedia capabilities by using standard telecommunication services, such as ISDN BRI, and controls that provide low noise and low distortion.

EHS

The European home systems (EHS) specification has been developed under European Commission funding under the ESPRIT program. Its aim was to interconnect electrical and electronic appliances into the home in an open way so that different manufacturers can offer compatible products. An EHS product consists of three parts: a modem chip, a microcontroller, and a power supply. The main power cabling is used to carry the command and control signals at a speed of 2.4 kbps. Digital information is carried by a high-frequency signal superimposed on the voltage of the main. Sensitivity to electrical noise remains a problem, and filters are necessary to eliminate unwanted interference. Other media used include coaxial cable (to carry frequency-multiplexed TV/digital audio signals and control packets, 9.6 kbps), two twisted pair cables (telephone and general purpose, 9.6 kbps and 64 kbps), radio, and infrared (1 kbps).

EIBA Technologies

The European Installation Bus Association (EIBA) has assumed the role of the integrator in the European market. The EIB system for home and building automation is another topology-free, decentralized system with distributed intelligence, based on a CSMA/CA protocol for serial communication. Currently, various EIBA bus access units for twisted pair are commercially available. The bus access unit includes a transceiver; it locally implements the operating system and caters for user RAM and EEPROM space.

EIBA's objectives include the development of a unified concept for electrical fitting and home and building management. EIBA is a multivendor body that aims to establish a standard for building system technology on the European market. It makes the EIB system know-how available to members and licensees, provides members and licensees with support and documentation, establishes standards among its members, and specifies appropriate criteria for quality and compatibility, with the help of external test institutes. It also maintains the position of the EIB Tool Environment (ETE) as an unrivaled platform for open software tool development, at the heart of which is the EIB Tool Software (ETS), offering a common tool for the configuration of EIB installations.

EIB components, actuators, and monitoring and control devices communicate via a standardized data path or bus, along which all devices communicate. Little wiring is required, which in turn results in lower fire risk and minimized installation effort. Home automation systems provided by Siemens (see www.siemens.de) follow the EIBA standards and have several desirable features. Siemens' Home Electronic System (HES) provides:

- Security due to the continuous control of active processes around the house at the homeowner's fingertips;
- Economy in the use of utilities such as water, electricity, and heating energy;
- Convenience through simplifying operation and reducing the burden of routine tasks; and
- Communication by integrating the household management system into external communications facilities.

IEEE 1394

In order to combine entertainment, communication, and computing electronics in consumer multimedia, digital interfaces have been created. Such is the case of IEEE 1394, which was conceived by Apple Computer as a desktop LAN, and then was standardized by the IEEE 1394 working group.

IEEE 1394 can be described as a low-cost digital interface with the following characteristics:

- *High speed*. It is able to achieve 100 Mbit/s, 200 Mbit/s, and 400 Mbit/s; extensions are being developed to advance speeds to 1.6 Mbit/s and 3.2 Mbit/s and beyond.
- *Isochronous support*. Bandwidth for time-sensitive applications is guaranteed by a deterministic bandwidth allocation for applications such as real-time video feeds, which otherwise could be disrupted by heavy bus traffic.
- *Flexible topology*. There is no central bus supervision; therefore, it is possible to daisy-chain devices.
- *Hot-plug capability*. There is no need for the user to configure node IDs or unique termination schemes when new nodes are added; this action is done dynamically by the bus itself.
- *Cable power*. Peripherals of low cost can be powered directly from the IEEE 1394 cable.
- *Open standard*. The IEEE is a worldwide standards organization.
- *Consolidation of ports of PCs*. SCSI, audio, serial, and parallel ports are included.
- There is no need to convert digital data into analog data, and loss of data integrity can be tolerated.
- There are no licensing problems.
- A peer-to-peer interface can be provided.

The EIA has selected IEEE 1394 as a point-to-point interface for digital TV and a multipoint interface for entertainment systems; the European Digital Video Broadcasters (DVB) have selected it as their digital television interface. These organizations proposed IEEE 1394 to the Video Experts Standards Association (VESA) as the home network media of choice. VESA adopted IEEE 1394 as the backbone for its home network standard.

PLC

At the end of 1999, the Consumer Electronics Association (CEA) formed the Data Networking Subcommittee R7.3, and began work on a High-speed PowerLine Carrier (PLC) standard. PLC technology aims to deliver burst data rates up to 20 Mbps over powerline cables. However, like CEBus and X10, PLC shares the same power network with motors, switch-mode power supplies, fluorescent ballasts, and other impairments, which generate substantial impulse and wideband noise. To face this difficult environment, different technologies take widely differing approaches depending on the applications they are pursuing. Technologies and algorithms including orthogonal frequency-division multiplexing (OFDM), rapid adaptive equalization, wideband signaling, Forward Error Correction (FEC), segmentation and reassembly (SAR), and a token-passing MAC layer are employed over the powerline physical layer technologies in order to enhance transmission robustness, increase the required bandwidth, guarantee the quality, and provide both asynchronous and isochronous transmission.

HomePlug

The HomePlug Powerline Alliance is a rather newly founded nonprofit industry association established to provide a forum for the creation of an open specification for home powcrlinc networking products and services. The HomePlug mission is to promote rapid availability, adoption, and implementation of cost-effective, interoperable, and specifications-based home power networks and products enabling the connected home. Moreover, HomePlug aims to build a worldwide standard, pursuing frequency division for coexistence with access technologies in North America, Europe, and Asia. For medium access control, Homeplug 1.0 extends the algorithm used in IEEE 802.11 to avoid collisions between frames that have been transmitted by stations (2).

HomePNA

HomePNA is defined by the Home Phoneline Networking Association in order to promote and standardize technologies for home phone line networking and to ensure compatibility between home-networking products.

HomePNA takes advantage of existing home phone wiring and enables an immediate market for products with “Networking Inside.” Based on IEEE 802.3 framing and Ethernet CSMA/CD media access control (MAC), HomePNA v 1.0 is able to provide 1 Mbps mainly for control and home automation applications, whereas HomePNA v2.0 (3), standardized in 2001, provides up to 14 Mbps. Future versions promise bandwidths up to 100 Mbp/s.

COMMUNICATIONS AND CONTROL MEDIA

Several media, individually or in combination, can be used in a home automation system. Power line carrier, twisted pair, coaxial cable, infrared, radio communications, Digital Subscriber Loop (DSL) technologies, cable modems, and fiber optics have been proposed and investigated. Each medium has a certain number of advantages and disadvantages. In this section, we will present some of the most profound features of the media.

The power line carrier (PLC) or mains has been proposed in several applications. It is the natural medium of choice in load management applications. No special cables need to be installed because the power line is the bus itself. From one side, the power line medium already has a large number of appliances connected to it, but on the other side it is not a very friendly medium for transmission of communication signals because there is a fluctuation of the power line impedance and a high noise level on the line. There is also interference with communication caused by other houses. Spread spectrum or ASK techniques have been proposed for efficient modulation of the signal in PLC.

Recent advances in twisted pair (TP) transmissions, especially in telecommunications and computer networking applications, make it very attractive for applications that use standard computer interfaces. TP can be the generic system for the home system datagram services; if new communication technologies reach the home, TP can be used for high-bandwidth applications as well. TP can be easily assembled and installed, and connectors can be easily attached to it.

Coaxial cables have not been extensively—except for the Japanese market—used in home automation systems. Their high bandwidth and the experience technical people have amassed through the cable systems make them a very attractive medium. Retrofitting them in existing houses is one of their major disadvantages.

Infrared (IR)—that is, electromagnetic radiation with frequencies between 10^{10} and 10^{24} Hz—has been used extensively in remote control applications. Its use in home automation systems will require line-of-sight—that is, detectors in every single room so that there is a full coverage.

Radio waves—that is, electromagnetic signals whose frequency covers the range of 3 kHz to 300 MHz—do not need direct vision between the transmitter and the receiver, but there is a need for a license and problems with interference. Radio-frequency technology is being used for real-time data management in LANs in order to give free access to the host system from multiple mobile data input devices. Wireless home networking technology will operate in the large-bandwidth radio-frequency ranges and will use proprietary compression techniques. In the future, consumers might receive e-mail messages wirelessly from a compliant handheld device or view enhanced Web content on their connected television sets. The use of a radio frequency of 2.4 GHz will cut down on noise within the home and provide some security.

Home networking opens up new opportunities for cost-effective phones that include Internet capabilities. By sharing resources, manufacturers should be able to reduce the cost of an Internet phone by using the processor and modem of a connected PC. Currently, a number of major manufacturers are developing their own wireless home networking products. Two major industry groups, the Home Phoneline Networking Alliance (HPNA) and the HomeRF, are attempting to develop standards for two different technology sets.

The HomeRF Working Group (HRFWG) was formed to provide the foundation for a broad range of interoperable consumer devices by establishing an open industry specification for wireless digital communication between PCs and consumer electronic devices anywhere in and around the home. HRFWG, which includes the leading companies from the PC, consumer electronics, peripherals, communications, software, and semiconductor industries, has developed a specification for wireless communications in the home called the Shared Wireless Access Protocol (SWAP).

The specification developed by the HRFWG operates in the 2.4-GHz band and uses relaxed IEEE 802.11 wireless LAN and digital European cordless telephone (DECT) protocols. It also describes wireless transmission devices and protocols for interconnecting computers, peripherals, and electronic appliances in a home environment. Some examples of what users will be able to do with products that adhere to the SWAP specification include:

- Set up a wireless home network to share voice and data among peripherals, PCs, and new devices such as portable, remote display pads.
- Review incoming voice, fax, and e-mail messages from a small cordless telephone handset.
- Intelligently forward incoming telephone calls to multiple cordless handsets, fax machines, and voice mailboxes.
- Access the Internet from anywhere in and around the home from portable display devices.
- Activate other home electronic systems by simply speaking a command into a cordless handset.
- Share an ISP connection between PCs and other new devices.
- Share files, modems, and printers in multi-PC homes.
- Accommodate multiplayer games or toys based on PC or Internet resources.

Bluetooth

The Bluetooth program, backed by Ericsson, IBM, Intel, Nokia, and Toshiba, is already demonstrating prototype devices that use a two-chip baseband and RF module and hit data rates of 730 kbit/s at 2.4 GHz. Bluetooth uses a proprietary MAC that diverges from the IEEE 802.11 standard. Bluetooth has already managed to serve as a universal low-cost, user-friendly air interface that will replace the plethora of proprietary interconnect cables between a variety of personal devices and peripherals. Bluetooth is a short-range (10 cm to 10 m) frequency-hopping wireless system. There are efforts to extend the range of Bluetooth with higher-power devices.

Bluetooth supports both point-to-point and point-to-multipoint connections. Currently, up to 7 slave devices can communicate with a master radio in one device. It also provides for several piconets to be linked together in an ad hoc networking mode, which allows for extremely flexible configurations such as might be required for meetings and conferences.

The Bluetooth protocol stack architecture is a layered stack that supports physical separation between the Link Manager and the higher layers at the Host Interface, which is common in most Bluetooth implementations.

Bluetooth is ideal for both mobile office workers and small office/home office (SOHO) environment as a flexible cable replacement that covers the last meters. For example, once a voice over internet protocol (VoIP) call is established, a Bluetooth earphone may automatically switch between cellular and fixed telephone networks, when one enters his home or office. Of course, the low-bandwidth capability permits only limited and dedicated usage and inhibits Bluetooth from in-house multimedia networking.

IEEE 802.11

IEEE 802.11 is the most mature wireless protocol for wireless LAN communications, deployed for years in corporate, enterprise, private, and public environments (e.g., hot-spot areas). The IEEE 802.11 standards support several wireless LAN technologies in the unlicensed bands of 2.4 and 5 GHz, and share use of direct-sequence spread spectrum (DSSS) and frequency hopping spread spectrum (FHSS) physical layer RF technologies.

Initially, the IEEE 802.11 standard provided up to 2 Mbps at the 2.4-GHz band, without any inherent quality of service (QoS). The wide acceptance, however, initiated new versions and enhancements of the specification. The first and most important is the IEEE 802.11b specification, which achieves data rates of 5.5 and 11 Mbps. Recently, the IEEE 802.1lg task group has formed a draft standard that achieves data rates higher than 22 Mbps. In the 5-GHz band, the IEEE 802.1la technology supports data rates up to 54 Mbps using OFDM schemes. OFDM is very efficient in time-varying environments, where the transmitted radio signals are reflected from many points, leading to different propagation times before they eventually reach the receiver. Other 802.11 task groups targeting specific areas of the protocol are 802.11d, 802.11e, 802.11f, and 802.11h.

HIPERLAN/2

HIPERLAN/2 is a broadband wireless LAN technology that operates at rates as high as 54 Mbps in the 5-GHz frequency band. HIPERLAN/2 is a European proposition supported by the European Telecommunications Standards Institute (ETSI) and developed by the Broadband Radio Access Networks (BRAN) team. HIPERLAN/2 is designed in a flexible way so as to be able to connect with 3G mobile networks, IP networks, and ATM networks. It can be also used as a private wireless LAN network. A basic characteristic of this protocol is its ability to support multimedia traffic i.e., data, voice, and video providing quality of service. The physical layer uses OFDM, a technique that is efficient in the transmission of analog signals in a noisy environment. The MAC protocol uses a dynamic TDMA/TDD scheme with centralized control.

Universal Serial Bus (USB)

As most PCs today have at least 2 USB ports, accessible from outside the case, connecting new USB devices is a very simple Plug-n-Play process. Moreover, USB is able to cover limited power requirements of the devices, in many cases eliminating the need for additional power cables. USB 1.1 provides both asynchronous data transfer and isochronous streaming channels for audio/video streams, voice telephony, and multimedia applications, and bandwidth up to 12 Mbps adequate even for compressed video distribution. USB v2.0 transfers rates up to 460–480 Mbps, about 40 times faster than vl.l, covering more demanding consumer electronic devices such as digital cameras and DVD drives. USB may not dominate in the Consumer Electronics Networks in the short term, but it will certainly be among the major players.

Universal Plug-and-Play (UPnP)

UPnP aims to extend the simplicity and auto-configuration features from device PnP to the entire network, enabling the discovery and control of networked devices and services. UPnP in supported and promoted by the UPnP forum. UPnP is led by Microsoft, while some of the major UPnP forum members are HP, Honeywell, Intel, Mitsubishi, and Philips. The scope of UPnP is large enough to encompass many existing, as well as new and exciting, consumer electronics networking and automation scenarios including home automation/security, printing and imaging, audio/video entertainment, kitchen appliances, and automobile networks.

In order to ensure interoperability between vendor implementations and gain maximum acceptance in the existing networked environment, UPnP leverages many existing, mature, standard protocols used on the Internet and on LANs like IP, HTTP, and XML.

UPnP enables a device to dynamically join a network, obtain an IP address, convey its capabilities, and be informed about the presence and capabilities of other devices. Devices can automatically communicate with each other directly without any additional configuration. UPnP can be used over most physical media including Radio Frequency (RF, wireless), phone line, power line, IrDA, Ethernet, and IEEE 1394. In other words, any medium that can be used to network devices together can enable UPnP. Moreover, other technologies (e.g., HAVi, CeBus, orXlO) could be accessed via a UPnP bridge or proxy, providing for complete coverage.

UPnP vendors, UPnP Forum Working Committees, and the UPnP Device Architecture layers define the highest-layer protocols used to implement UPnP. Based on the device architecture specification, the working committees define information global to specific device types such as VCRs, HVAC systems, dishwashers, and other appliances. UPnP device vendors define the data specific to their devices such as the model name, URL, and so on.

DSL and Cable Modems

Digital subscriber line (DSL) is a modem technology that increases the digital speed of ordinary telephone lines by a substantial factor over common V.34 (33,600 bps) modems. DSL modems may provide symmetrical or asymmetrical operation. Asymmetrical operation provides faster downstream speeds and is suited for Internet usage and video on demand, where the heaviest transmission requirement is from the provider to the customer.

DSL has taken over the home network market. Chip sets will combine home networking with V.90 and ADSL modem connectivity into one system that uses existing in-home telephone wiring to connect multiple PCs and peripherals at a speed higher than 1 Mbps.

A cable modem is another option that should be considered in home network installations. Cable modem service is more widely available and significantly less expensive than DSL in some countries. Cable modems allow much faster Internet access than dial-up connections. As coaxial cable provides much greater bandwidth than telephone lines, a cable modem allows downstream data transfer speeds up to 3 Mbps. This high speed, combined with the fact that millions of homes are already wired for cable TV, has made the cable modem one of the top broadband contenders. The advent of cable modems also promises many new digital services to the home, including video on demand, Internet telephony and videoconferencing, and interactive shopping and games.

At first glance, xDSL (i.e., DSL in one of the available varieties) appears to be the frontrunner in the race between cable modems and DSL. After all, it can use the phone wire that is already in place in almost every home and business. Cable modems require a television cable system,which is also in many homes and businesses but does not have nearly the same penetration as basic telephone service. One important advantage that cable modem providers do have is a captive audience. All cable modem subscribers go through the same machine room in their local area to get Internet access.

In contrast to cable modem service, xDSL's flexibility and multi vendor support is making it look like a better choice for IT departments that want to hook up telecommuters and home offices, as well as for extranet applications. Any ISP will be able to resell xDSL connections, and those connections are open to some competition because of the Telecommunications Act of 1996. The competitive multi-vendor environment has led to a brisk commodity market for xDSL equipment and has made it a particularly attractive and low-cost pipe. Although new services are sure to be spawned by all that bandwidth, xDSL providers are able to depend on the guaranteed captive audience of their cable modem counterparts.

Fiber Optics

Fiber optics at home have also been evaluated in the literature. The well-known advantages of fiber, such as increased bandwidth, immunity to electromagnetic noise, security from wiretaps, and ease of installation, compete with its disadvantages, such as higher cost, difficulty in splicing, and requirement of an alternate power supply. A standard for a fiber optic CEBus (FOBus) has been developed.

One of the major drives behind the use of fiber optics is the ability to carry multimedia traffic in an efficient way. As telecommunication companies are planning to bring fiber to the home, a fiber optic network in the house will make the Internet working with places outside the house cost effective and convenient. Connection with multimedia libraries or with other places offering multimedia services will be easily accomplished to the benefits of the house occupants, especially students of any age who will be able to access, and possibly download and manage, these vast pools of information.

Several minimum requirements of a FOBus are set forth. In terms of service, the FOBus should provide the following services:

- Voice, audio, interactive, bulk data, facsimile, and video;
- One-way, two-way, and broadcast connectivity;
- Transport of continuous and bursty traffic;
- Interfaces to external networks and consumer products; and
- Multiple data channels and a single, digital control channel.

The network should meet the following physical requirements:

- Low installation costs and ease of installation;
- High reliability;
- Easy attachment of new devices;
- No interruption of service while a new node is being connected; and
- Access to the network via taps in each room.

The FOBus standard should also have a layered architecture in which layers above the physical layer are identical to the corresponding CEBus layers in other media.

Some of the applications of a fiber optic network in the home that will drive the design of the fiber optic home network are: the connection to emerging all-fiber networks, which will provide high-quality, high-bandwidth audio/visual/data services for entertainment and information; fiber network connection to all-fiber telephone networks to allow extended telephone services such as ISDN, videotelephone, and telecommuting; transport of high-quality audio/video between high-bandwidth consumer devices such as TVs and VCRs; and transport of control and data signals for a high degree of home automation and integration.

SECURITY

Security (the need to prevent unauthorized nodes from reading or writing information) is an issue of concern for every networking product. Many manufacturers have decided to create a security context on their products and have the key information on them, which means that one object of one context sends a message to another context object, and thus both have to be built by the same company so that the security encoding algorithm can be exchanged between them.

Security in the home automation systems literature is seen as follows:

- Security in terms of physical access control and alarm systems.
- Security in terms of the well being of house inhabitants through systems that monitor health status and prevent health problems.
- Security of the building itself in terms of a safe construction and the subsequent monitoring of this status.
- Security in terms of confidentiality of the information exchanged.

The latter is being achieved by the use of various security techniques in use, including message authentication algorithms, which are of two main types. Two-way authentication algorithms require the nodes involved in the checking to know the encoding algorithm, and each node must have an authentication key in order to accept the command issued. A one-way authentication algorithm verifies only the transmitter and the information that goes on the APDTU (packet in the application layer); it requires only one authentication key, but the encoding algorithm must be known by the nodes. Both types of algorithm require a random number that is encoded with the authentication keys.

Encryption is also used in order to obtain greater security in the message and in the data sent on the APDU. The algorithm or technique used has to be known by the receiver and transmitter. Encryption is implemented with the help of the authentication algorithm ID in the second byte.

FUTURE DIRECTION

Home automation systems have been presented as a promising technology for bringing the computer and communications revolution that has swept the office and industrial environments in the last decade to the home environment. However, we have not seen an use of home automation systems and an increase in the market share as predicted from market analysts. This lack of acceptance can be attributed to marketing problems, costs of installation and retrofitting, slow growth of new housing, and a lack of standards that synchronize with the developments in the other technological areas.

The wide availability of powerful computers at homes and the availability of high-speed telecommunications lines (in the form of cable TV, satellite channels, and, in the near future, fiber) make a redirection of the home automation industry necessary. More emphasis should be on applications that require access to external sources of information—such as video-on-demand and the Internet—or on access from outside the home to home services—such as the load management application discussed above from utilities or individuals and remote surveillance.

User-friendly customer interfaces combined with reasonable pricing will certainly move the industry ahead. The availability of the Internet and the World Wide Web should be exploited in different ways. First, the interfaces and the click-and-drag operations could be adopted and then the high use of bandwidth could be accomplished. The above considerations should be viewed in light of cost and retrofitting issues in existing dwellings and the availability of appliances that are compatible with standards and that can be purchased from multiple vendors.

Wireless technologies seem to dominate the future of home automation systems. With regard to the future of fiber optics at home, several observations can be made. External or non premises service providing networks, and second-generation television, receivers such as high-definition television (HDTV) are two main areas in which developing technologies will impact the design of the FOBus. One external network that the FOBus will have to accommodate is the public telephone network. The current public switched network uses copper wire in its local loop to provide service to a neighborhood; but in the future, the use of fiber in the loop (FITL) will be gradually phased in. Neighborhood curbside boxes will be replaced with optical network units (ONUs) that will provide plain old telephone service (POTS) as well as extended network services. Initially, the service to the home will be provided on copper medium, but it will eventually be replaced with fiber as well. The FITL system will support broadband communications, especially interactive applications.

Another external network that will impact the FOBus design is the cable television network, which is also gradually being replaced by fiber. The FOBus specification will have to accommodate the high-bandwidth services delivered by the cable network (generally in the form of broadcast channels); it may also have to support interactive services that are envisioned for the future.

The other developing technology that will impact the design of the fiber optic CEBus is the emerging advanced television (ATV) standard, which will most likely include HDTV. In the United States, the EIA is examining digital standards for HDTV transmission. Most require bandwidth of 20 Mbps, which the proponents of the standards claim can be transmitted on a standard 6-MHz channel using modulation techniques such as quadrature amplitude multiplexing. In addition, the ATV receiver will likely have separate input ports for RF, baseband digital, and baseband analog signals. The choice of which of these ports to use for the CEBus/ATV interface has not been made. Each has its own advantages. Using the RF port would allow a very simple design for the in-home fiber distribution network, and the interface would only have to perform optical-to-electrical conversion. The digital port would remove bandwidth constrictions from the broadcast signal and also allow for interactive programming and access to programming from various sources. The ATV could become the service access point for all audio/visual services in the home.

An important issue in home automation is the integration of Internet technologies in the house. Several companies have proposed technologies to embed network connectivity. The idea is to provide more control and monitoring capability by the use of a Web browser as a user interface. In this new technology, Java and http (standard Internet technologies) are accessed through a gateway that manages the communication between the Web browser and the device.

Among the advantages of this new technology are the following:

- Manufacturers can provide their products with strong networking capabilities, and increase the power of the Internet and the available intranets.
- The use of a graphical user interface (GUI) allows a simple display of the status, presence, and absence of devices from the network.
- Java, Visual Basic, and Active X development environments reduce the development time of device networking projects.
- Interface development is easy.
- Batch processes to gather data are easy and fast.

Standard technologies to network devices via the Internet provide for the development of internetworking solutions without the added time and costs of building proprietary connections and interfaces for electronic devices.

Manufacturers of home automation systems must take into account several factors. The users are the first to be considered. Their physiological and psychological capabilities as well as their socioeconomic characteristics must be considered before a new technology is adopted.

Another issue is the added value provided by such systems in terms of the reduction of repetitive tasks and the skills and knowledge required to operate them. Health and safety considerations must be taken into account. Also, one needs to examine the current status of technologies and the dynamics of these technologies in order to offer a successful product in the market and, mainly, in order to create a new healthy market sector.

The suggested technologies should be able to enhance the life in a household but certainly not dominate it. The systems should be reliable and controllable but also adaptive to specific user needs and habits. They should also be able to adapt to changing habits.

BIBLIOGRAPHY

1. *Draft CEBUS FO network requirements document*, Washington DC: EIA, May 15, 1992.
2. *HomePlug 1.0 Specification*, HomePlug Alliance, June 2001.
3. *Interface Specification for HomePNA 2.0: 10M8 technology*, December 1999.

FURTHER READING

The EIA/CEG Home Automation Standard, Electronics Industries Association, Wahsington, DC, Dec. 1989.

C. Douligeris, C. Khawand, and J. Khawand, Network layer design issues in a home automation system; *Int. J. Commun. Sys.*, **9**: 105–113, 1996.

C. Douligeris, Intelligent home systems, *IEEE Commun. Mag. (Special Issue on Intelligent Buildings: From Materials to Multimedia)*, **31**(10): 52–61, 1993.

M. Friedewald, O. Da Costa, Y. Punie, P. Alahuhta, and S. Heinonen, Perspectives of ambient intelligence in the home environment, *Telematics and Informatics*. New York: Elsevier, 2005.

C. Khawand, C. Douligeris, and J. Khawand, Common application language and its integration into a home automation system, *IEEE Trans. Consum. Electron.*, **37**(2): pp. 157–163, 1991.

J. Khawand, C. Douligeris, and C. Khawand, A physical layer implementation for a twisted pair home automation system; *IEEE Trans. Consum. Electron.*, **38**(3): 530–536, 1992.

B. Rose, Home networks: A standards perspective, *IEEE Commun. Mag.*, 78–85, 2001.

N. Srikanthan, F. Tan, and A. Karande, Bluetooth based home automation system, *Microprocessors Microsyst.*, **26**: 281–289, 2002.

N. C. Stolzoff, E. Shih, and A. Venkatesh, The home of the future: An ethnographic study of new information technologies in the home, Project Noah, University of California at Irvine.

T. Tamura, T. Togawa, M. Ogawa, and M. Yuda, Fully automated health monitoring system in the home, *Med. Eng. Phys.*, **20**: 573–579, 1998.

J. Tidd, Development of novel products through intraorganizational and interorganizational networks: The case of home automation, *J. Product Innovation Manag.*, **12**: 307–322, 1995.

T. B. Zahariadis, *Home Networking: Technologies and Standards*. Norwood, MA: Artech House, 2003.

T. Zahariadis, K. Pramataris, and N. Zervos, A comparison of competing broadband in-home technologies, *IEE Electron. Commun. Eng. J. (ECEJ)*, **14** (4): 133–142, 2002.

Christos Douligeris
University of Piraeus
Piraeus, Greece

H

HOME COMPUTING SERVICES

INTRODUCTION

Relevance of the Topic

The 1990s and the current decade have experienced tremendous growth in computers and telecommunications, and, for the first time, developments in technologies in the home followed in close proximity to their correlates in the corporate world. Notably, the diffusion of the Internet into the private sector has proceeded at enormous pace. Not only has the number of households with Internet access skyrocketed, but also access speed, number of users within the household, types of uses, and mobility of access have expanded.

In some cases, corporate use of technologies followed private home use (e.g., for Instant Messenger and other chat applications). Popular private applications such as music and video downloads initially required access to large corporate or academic networks because of capacity needs. Such applications encouraged the increasing diffusion of broadband into private homes.

Home and business technologies are increasingly intertwined because of the increasingly rapid pace of innovation. Also, home information technology (IT) may experience growth during times of economic slowdown because of price decline or network effects (DVD; Internet in the early 1990s; wireless today).

Although convergence is a predominant trend, a market for private IT applications separate from the corporate market is evolving as well. Price decline and miniaturization encourage the perspective of ubiquitous computing and of a networked society.

Definitions

A range of concepts have evolved that permit the conceptual separation of business/public computing services from those related to the home or private use. One definition points to *all the infrastructures and applications the private user can take advantage of for private uses*. This definition encompasses most applications discussed in this article, notably entertainment, information, communication, and shopping. Some other applications cross over into the public or business realm, in particular telework and distance learning. Although this article focuses on services in the home, more recently miniaturization and mobile technologies have blurred the line between home and other locations. Mobile phones, personal digital assistants (PDAs), personal entertainment technologies all are designed to extend applications that are conveniently available in the home to any location the user chooses.

Home computing trends revolve around various household functionalities, notably entertainment, information, purchasing, education, work, and health. During an age of networks, these applications are often no longer merely household related, but they require integration of home and business technologies. A key trend observed during the past decade has been the convergence of technologies, of content, and of applications.

Structure of this Article

Although understanding the technological advances in this area is important, much of the technology is derived from corporate computing applications and adopted for home use. Thus, this article will focus on content and usage of home computing more so than on technical details.

This article explores key issues pertaining to home computing products and services. In particular, it will discuss convergence of technology and other current technological trends related to end-user devices and networking. Selected services for the home will be addressed in light of technological changes.

As the technology becomes more available and common, concepts such as “computerized homes,” “Home-IT,” “information society,” or “networked society” are increasingly defined by the services with which they are associated.

The article concludes with future Home-IT trends.

DRIVERS OF TECHNOLOGY ADOPTION IN THE PRIVATE HOME

Convergence

Convergence of technologies has a critical impact on home computing as well as information and entertainment. Although analog technologies generally coincided with a limited one-on-one relationship of applications and appliances, digital technologies have made it possible to perform multiple functions with the same piece of equipment, which has lead to an increasing overlap between the telecommunications, television, and consumer electronics industries. For the user, it means that the same appliance can be used for work-at-home, chat, children’s entertainment, and online shopping or banking. Apart from technological innovation and cooperation among industry sectors, adoption of interactive media consumption patterns by the users is the third dimension of convergence. There is a continuing debate as to how rapidly convergence will be embraced by consumers. Although it has been technically feasible for some time, convergence is seen as limited because of demographics, lifestyle preferences, and other factors (1). For instance, the convergence of television (TV) and computers on the user side has not advanced as rapidly as expected, even though streaming video of television programming is available on the Internet, cable systems offer “Digital Cable,” and cell phones have cameras that permit instant e-mailing of pictures. Most Americans still watch television one program at a time, even though many rely increasingly on the Internet for news, weather, stock market, and other information.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

However, at least on the supply side, convergence is gradually advancing. Responding to digital satellite competition, cable companies have enhanced the existing fiber/coax physical plant of their systems with digital set-top boxes and digital distribution technology. These upgrades permit greater channel capacity, as well as interactive features. On-screen program guides, several dozen pay-pv-view (PPV) channels as well as multiplexed premium cable channels and digital music channels are common. Digital picture, flat-screen technology, surround sound, and high-definition television (HDTV) encourage the trend toward home theaters. In a typical digital cable offering interactivity is limited to two levels of information, which can be retrieved while watching a program or perusing the on-screen program guide; PPV ordering, as well as selection, programming, and recording of future programs through the on-screen guide are also interactive features. The systems are designed to allow for future expansion, especially online ordering of services as well as other purchases. Some systems offer Video on Demand (VoD), in which users can order movies and other videos from a large selection in a real-time setting. The more common "in-demand" offerings simulate a near-VoD experience, in which the most popular movies are available at half-hour starting times.

Several providers experiment with interactive applications that give the viewer options beyond simply choosing a program, including game show participation, choice of camera angles at sports games, access to background information for products advertised in commercials, and choice of plot lines and endings in movies. Other interactive uses of TV are calling up additional information on news and sports or TV/PC multitasking. Increasingly, TV and radio are supplemented by websites for information retrieval as well as audience feedback and service applications (such as buying tickets or merchandise).

In the consumer electronics sector, convergence is currently taking place both from computer companies and from home entertainment companies. Microsoft has developed a media player that allows integration of video, audio, photos, and even TV content, and Intel is making a significant investment in companies creating digital consumer products (2). On the other hand, Sharp is planning to debut liquid crystal display (LCD) TVs with PC card slots that enable the addition of "digital-video recording functions or a wireless connection to a home computer network" (3).

User Interface: TV, PC, Phone

Much discussion of Home-IT focuses on the Internet. Innovations associated with traditional media also offer considerable potential, in part because all electronic media are evolving rapidly, converging with other media, and becoming increasingly interactive. These hybrid media often reach the majority of the population (in some countries, a vast majority) that lacks regular, adequate Internet access (4, 5). Also, in spite of improvements in "user friendliness," many users see the PC as work-related, difficult to use (requires typing), and prone to breakdowns and viruses. PCs also tend to be outdated within a few years. By contrast, TV sets last for decades, they are easy to use, not prone to viruses, and are less expensive.

Worldwide, TV consumption is still the prevalent leisure activity, mainly because of its universal, low-cost accessibility and its ability to afford hours of entertainment and information with minimal effort. Although usage patterns are changing rapidly, for some time consumers may continue to choose TV for news and entertainment and PC for other sources of information and electronic commerce. Also, there seems to be a demographic pattern in that young viewers increasingly stray away from conventional TV news either to Internet news or entertainment/news programs (e.g., Comedy Central). Although it is a digital technology, the tremendously rapid adoption of the DVD player is largely a replacement for VHS home video with higher video quality.

Although the expectation was that video delivery would increasingly involve home computing devices, such as combination PC-TV or Web-TV and digital recording technology such as TiVo (5), most households invest in big-screen televisions and surround sound. TiVo was also adopted more slowly than expected.

A third popular user interface is the telephone. As a result of their rapid replacement cycle compared with regular-line phones, cellular phones in particular tend to be equipped with the latest technological gadgets. As prime value is placed on instant "24/7" communication, mobile technology epitomizes trends in personal technology. As a result of simple use, ubiquity, and compatibility with existing technology (i.e., the existing telephone network), adoption and upgrading of mobile phones are rapid. Besides regular voice use, text messaging has gained popularity among younger users, especially in Europe and Japan. Currently, web access is available via narrowband channels. However, the next generation of mobile broadband is currently being deployed. In concert with smartphones and wireless PDAs, broadband mobile networks (e.g., those based on the UMTS (Universal Mobile Telecommunications System) standard) provide multimedia services such as videophone or content streaming. The first rollout in Asia started in 2003. Pricing and compelling services are again key to success.

Interactive Entertainment

Content is the key to adoption of advanced interactive services. As a result of the high visibility of movies, the great public interest in this type of content, and their easy availability, Movies-on-Demand was the offering of choice for early interactive trials. Meanwhile, cable systems and satellite providers offer near PPV with 50–100 channels offering current movies as well as specialized (e.g., "adult") programming and sports or music events.

Music, sports, and special interest programming also have received their share of attention by the programmers of interactive cable systems. Interactive game channels are added to some systems. In-home gambling has strong economic appeal; regulatory barriers prevail, however. Anecdotal evidence suggests that participants in interactive trials enjoyed watching regular TV programs they missed during the week, newscasts tailored to individual preferences (6), as well as erotica.

Several television providers have experimented with interactive applications that give the viewer options beyond simply choosing a program, including participation in game shows such as *Wheel of Fortune* and *Jeopardy*, "pick-the-play" games for *Monday Night Football*, ordering pizza using Web-TV during a *Star Trek* marathon, access to background information for products advertised in commercials, and choice of plot lines and endings in movies.

Compared with the massive number of traditional movies available, interactive movies are few and far between. They are difficult to produce and require considerable technology. Even most sites for Internet video provide mainly repackaged conventional programming. Audience demand for interactivity is not yet understood. Many children and teens feel comfortable with it because of exposure to video and computer games; in fact, a considerable number of toys now include interactive components and interface with the world wide web (WWW) (7). Most likely the push for greater interactivity will come from advertising, which already relies on cross-promotion between different media including TV and Internet. As the marketing increasingly focuses on individualization, the ability to provide targeted advertising even within the same program is likely to have great appeal to advertisers. Also, because commercial avoidance is increasingly common, the push for product placement within programs may also lead to increasingly individualized product inserts.

Broadcast television stations are expanding their channel offerings as a result of conversion to HDTV and the resulting availability of greater channel capacity. However, the expectation is that they will, at least initially, offer greater selection and targeting rather than actual interactivity.

The Digital Home

The ultimate interactive experience may involve a home that is equipped with technology that can respond to the residents' needs. Smart house technology typically is developed for high-end or special needs homes, and these technologies filter down into existing and mid-level homes. Some smart-house solutions for the elderly use the TV set as an interface for appliance control and surveillance. A key feature of future smart-house technology is the ability of various appliances to "talk to the Internet and to each other" (8), which allows a maximum of control by the user, as well as coordination of technologies. In the long run, shifting control onto the Web could generate considerable cost savings by reducing the complexity of the technology within each device.

Especially home networking technologies such as the European EIBus or the US-led CEBus enable the interconnection of different household devices such as heating, shades, or lighting. In addition, wireless local area networks (LANs) are gaining ground in the private sphere, connecting IT devices. Eventually, audio/video, PC, and other household networks will converge (9,10) .

Although many such technologies are available, they have not been adopted on a broad scale. However, one might expect that demographic trends will drive such adoption: Aging baby boomers have an increased need for home-based conveniences and efficiencies; young home buyers have grown up with network technologies and may expect a high level of technology in their future homes. Also, elderly family members need increased attention, which may be facilitated via available technologies.

However, services to be delivered to the home not only require in-home technologies. Service providers such as banks or media firms need to prepare back-end infrastructures such as fault-tolerant servers, load-balancing access pipes, and real-time databases with information on availability or price quotes. Those out-of-home infrastructures are connected to the home via networks such as cable, telephone, powerline, or wireless connections.

SERVICES FOR THE HOME

Media attention has been focused on innovative infrastructures for the residential area such as wireless LAN in the home or broadband connections to the Internet.

However, the private household, even more than a corporate user, is interested in the application side (i.e., an easy-to-use, reasonably-priced, and fun service provision). Many applications exist in reality, yet they provide a quite unstructured picture.

Kolbe (11) proposed a classification scheme for analyzing and describing the respective classes of home applications in existence. According to Brenner and Kolbe (12), there are eight main services for the private household that can be supported by IT (see Fig. 1):

The basic services "information" and "communication" take mutual advantage of each other: There is no communication possible without at least basic information provided on one end, sometimes referred to as message or content. In turn, information needs to be conveyed in order to provide any benefit. For example, any news story posted by an Internet portal is meant as "communicating

Core services:

- Information
- Communication

Home services:

- Health
- Home services
- Travel
- Transactions
- Entertainment
- Education

Figure 1. IT-influenced services for of the private household.

information" to the (anonymous or personalized) users of that portal.

They are referred to as core *services* whereas the other ones are looked on as primary *home services* because they are based on information and communication features. Nevertheless, "communication" and "information" are described separately as some services exclusively provide bilateral or multilateral information (e.g., electronic books, news) or communication (e.g., e-mail, short message service (SMS)) benefits. Market revenues are most substantial in those basic areas.

Miles (13) and others (10) after him observed that more and more aspects of private life are affected by home services. We can differentiate three forms of usage according to the degree of networking. Prior to widespread networking, stand-alone applications such as an electronic encyclopedia or a game on a PC were common. The next step is locally interconnected applications within the confines of the private home such as entertainment networks for home cinema applications or controlling the heating via TV or PC. The third form is the out-of-the-home connected applications such as applications using the Internet for e-mail or shopping as well as remote monitoring services. All services can be structured a long these three areas.

In practice, these types of services are used in conjunction with each other, for example, during activities on the Internet the household seeks weather information (information and travel) for air travel via a portal or a price comparison for airfares (travel), then executes the purchase (transactions) using a travel portal and then pays online using a credit card (transactions), and finally gets an e-mail or mobile message confirmation of this order (communication). Another example is the 'Info- or Edutainment' area that unites information, entertainment, and education aspects (e.g., in interactive multimedia encyclopedias or electronic learning toys for children).

Work, transaction, and private aspects of life are converging as are technologies and applications. In some instances, private and business usage is almost indistinguishable (e.g., the use of an Internet portal or some smart phone features). Therefore, some of the services described below may also provide business value as selective business applications benefit the private user, especially in a home office environment.

Core Services

Information. *Information* is offered by all services in which the dissemination of information to the private household is central. Information provides the basis for more complex service types to be discussed later.

The following residential applications fall into this category:

- News portals providing up-to-date coverage such as news or weather information. Together with search capabilities, they provide access to the vast resources of the Internet to the private user. Interactive TV and multimedia broadband networks are prerequisites for customized individual news services that compile one's own newspaper on personal preferences and interests like sports or stock exchange news as examined by MIT's Media Lab.
- Electronic books and newspapers such as the electronic version of the *New York, Times*, which is available online for a fraction of the newsstand price. Electronic books with portable e-book players are one of the most notable examples for pure information. Encyclopedias, magazines, dictionaries, or special topics are available on different formats for proprietary players. Hyperlink functionality, connectivity to video printers, and find-and-select algorithms are advantages that traditional books do not share.
- Push services of events and product news: Mobile marketing is gaining ground fast. The latest research in Finland shows that 23% of all mobile-phone-using Finns (80% of all Finns) have received SMS push marketing (14).
- Information kiosks, which provide basic information for travelers or shoppers.

Communication. *Communication* enables the private household to establish bilateral or multilateral contact with the immediate or extended environment. This core service provides information as the basis for a variety of further services. However, communication as a basic need of users is evident in the residential home. Traditional media like telephone and fax have been complemented by innovative media such as e-mail or mobile communications, both text and voice.

SMS has achieved near 80% usage rates in some European countries, and SMS advertising has exploded. Mobile text messages generate a substantial part of telecom operators' revenue. In Europe, SMS revenues were at 12 billion Euros for 2002 (15).

Mobile phone users in the United Kingdom sent over one billion text messages during April 2002. The Mobile Data Association predicts that the total number of text messages for 2002 will reach 16 billion by the end of the year (16).

Home Services

Health. *Health* refers to all applications concerned with making provision for, maintaining, and monitoring the health of a person or social group.

Related services in the area are:

- Telemedicine with patient monitoring (surveillance of vital signs outside the hospital setting) and monitoring of dosage (including real-time adjustment based on the patient's response). Wireless sensors can be attached to the body and send signals to measurement equipment. They are popular in countries with widely dispersed populations (e.g., Norway) and increasingly developing countries.
- Electronic fitness devices that support training and wellness of the private user.
- Health-related websites.

Health applications for today's household are very limited in its range. In some countries, smart cards carry patients' data for billing and insurance companies or health consultancy software for private diagnosis and information about certain diseases. In the future, expert systems will enable medical advice from each home without leaving the private bed.

Home Services. *Home services* consist of systems that support home security, safety, meal preparation, heating, cooling, lighting, and laundry.

Currently, home services comprise only special devices such as those in a networked kitchen. Future scenarios project comprehensive home automation with interconnected kitchen appliances, audio and video electronics, and other systems like heating or laundry. Some prototypes by the German company Miele (called Miele@home) showed early in the development of "smart homes" that the TV can control the washing machine. The interconnection to out-of-home cable TV or telephone networks leads to the remote control services (e.g., security). Much media attention was received by the Internet refrigerator by NCR, which orders needed groceries without human interaction.

Key areas comprise:

- Central control of heating or air conditioning from home computer or TV.
- Lighting, shutters, and temperature control.
- Remote monitoring of home devices for security, laundry, refrigeration, or cooking.

Intelligent clothing and wearable computing are seen as emerging areas.

Travel. *Travel* includes all applications that support the selection, preparation, and undertaking of journeys. Travel applications make the central booking information systems for hotel or flight reservation accessible to the residential user. Individual preferences provide a search pattern for finding the places of interest. Future visions includes interactive, multimedia booking from the TV chair via broadband network with instant acknowledgements.

Main focus areas are:

- Travel planning on the Internet ranges from planning the entire trip via travel portals *Travelocity* or *Expedia* to selected information on public transportation or plane departures. These travel data can also be pushed to mobile devices or delivered according to the geographic position of the user.
- Automotive services. Increasingly, the car becomes an entertainment and information center with complete audio and video system. In addition, global positioning functionality helps planning and undertaking trips.
- Ticketless travel, such as e-ticket of airlines and ticketless boarding with contactless smart cards.

Transactions. *Transactions* combine all the administrative services and transactions, such as shopping and banking, of the private household.

The main applications of administration, e-banking, and e-shopping are applications serving "traditional" functions (17). Those services help the home to fulfill necessary administrative obligations with more efficiency and ease.

Using the PC and Internet connection, the private user can perform his bank business or order certain merchandise. Today's services (e.g., management of payments) will extend to a broader range (e.g., complete investment and mortgage affairs).

Of particular importance are the following transaction-oriented services:

- Electronic execution of administrative activities such as monitoring the household's budget with spreadsheets or planning software such as Quicken.
- Using personal information management (PIM) software such as scheduling, personal address book, or task lists, often provided in combination with PDAs or smart phone software.
- Deployment of productivity tools such as word processing, presentations, or spreadsheets for private letters, invitations, or planning purposes.
- Electronic banking and investing is the main service in this category. Although the focus is still on well-structured transactions such as payments (e.g., electronic bill presentment and payment (EBPP)), more complex tasks such as investment advice and research is delivered to private banking clients.

  In Switzerland, more than 50% of all private banking clients use the Internet for banking. Overall, 13% of all brokerage transactions and 26% of all payments are done via e-banking. Financial information is also accessed by the households. The big Swiss bank, UBS, lists prices of more than 370,000 stocks. Alerts can be sent to a mobile device. Some banks offer mobile banking services that resemble the features of the Internet offering.
- Shopping on the Internet has become an important service. Although purchases focus on standardized products, everything from furniture to groceries is available. The percentage of online purchases relative to total shopping revenue remains at moderate levels but is gradually increasing. The 2003 Christmas season experienced a strong increase in Internet sales: 18 billion (out of 217.4 billion total sales), up from 13.8 billion in the last quarter of 2002. More importantly, many retailers have offered a seamless shopping experience of catalogs, Internet, and stores (18). Especially auctions like eBay have received much attention from the private user: Amazon.com, a Fortune 500 company based in Seattle, opened its virtual doors on the World Wide Web in July 1995. Amazon.com and other sellers list millions of unique new and used items in categories such as apparel and accessories, sporting goods, electronics, computers, kitchenware and housewares, books, music, DVDs, videos, cameras and photo items, toys, baby items and baby registry, software, computer and video games, cell phones and service, tools and hardware,

Type of service	***Service area***	***Status quo 2004***	***Scenario 2007***	***Scenario 2010***
CORE SERVICES	*Information*	Electronic books, news portals	Fully electronic newspaper based on personalized profile	Electronic newspaper on e-paper
	Communication	Home-fax and mobile digital telephone	E-mail from every mobile device	Worldwide multimedia video communications
HOME SERVICES	*Health*	Consultancy software	Interactive, remote health services	Medicinal diagnostics at home by expert systems
	Home services	Only special interconnected household technologies, no standards, remote monitoring	Increased home automation via standard interfaces, entertainment and home services converge	All household equipment networked to in- and out-of-home devices, the "wired" home
	Travel	Travel portals, complete journey booking from home, GPS services	Intelligent guiding services for cars, location-based services, Internet access in cars	Automatic driving services, fully telematic information for the car
	Transactions	Home shopping over the Internet Integration of 'clicks and bricks'	Multimedia home shopping also for complex products	Virtual electronic shopping mall
		Home-banking for selected transactions	Home-banking for all activities	Multimedia banking, cybercash
	Entertainment	One way pay-TV, interactivity via telephone lines	Pay-per-view, limited number of services	Fully communicative TV (personal influence on action) and Video-on-demand
	Education	Computer Based Training software or Internet offerings	Distant multimedia learning at home, public electronic libraries	Individual virtual teachers using artificial intelligence and virtual reality simulations

Figure 2. The evolution of home computing services.

travel services, magazine subscriptions, and outdoor living items.

Entertainment. *Entertainment* includes those applications that can be used for leisure activities or for the purpose of entertaining household members.

Particular areas of entertainment services are:

- Home cinema with digital surround audio and home media server that connect flat plasma or LCD-TVs, audio systems, and multimedia PC environments with the Internet. In 2003, U.S. DVD sales surpassed videotape figures for the first time.
- On-demand digital TV with hundreds of channels of audio and video content.

- Games and gambling both via the Internet and mobile networks and in electronic stand-alone devices such as game boys and gambling machines.
- Digital toys such as Sony's smart dog or Lego's *Mindstorms* programmable brick sets developed in collaboration with MIT's MediaLab. Here, a close relationship to the learning component is evident.
- Using multimedia devices such as digital video cameras or digital photography in combination with home PCs and video authoring software for creating multimedia shows at home.
- Free and premium Internet radio with endless options of genres and downloadable music on portable devices such as MP3 players or smartphones.
- Adult content.

Education. *Education* refers to all applications that train and educate members of the household in special skills or knowledge. In an increasingly dynamic private environment, this function will gain in importance.

Distance Learning (DL) is frequently a self-selected activity for students with work and family commitments. Effects of social isolation should thus be limited. For instance, DL can facilitate daycare arrangements. In some circumstances, exclusion from the social network of the face-to-face classroom can be one of the drawbacks of DL (21). The private household uses this type of "education" for the training of special skills it is interested in using off-line computer-based training (CBT) software on CD-ROM or DVD to improve, for example, on a foreign language for the next holiday abroad or naval rules in order to pass the sailing exam. In addition, electronic accessible libraries and content on the Internet open the field for self-education processes to the private area. The usage artificial intelligence will substitute human teachers as far as possible and make them more efficient for special tasks. Virtual reality will help by visualization and demonstration of complex issues. Increasingly, colleges and universities offer DL classes based on strong demand from traditional and nontraditional students. Besides the added flexibility and benefit for students who are reluctant to speak up in class, DL benefits those students living far from the place of instruction. Dholakia et al. (22) found that DL has the potential to reduce or modify student commutes.

OUTLOOK

Figure 2 summarizesthehomeservicesandshowssomeofthe expected developments for the next several years. It summarizes three possible scenarios (status quo 2004, scenario 2007, and scenario 2010) based on the assessment of past, current, and future trends, and developments of services.

BIBLIOGRAPHY

1. H. Stipp, Should TV marry PC? *American Demographics*, July 1998, pp. 16–21.
2. Computer companies muscle into field of consumer electronics *Providence Sunday Journal*, January 11, 2004, 15.
3. Consumer electronics show is packed with Jetson-style gadgets *Providence Journal*, January 10, 2004, B1,8.
4. F. Cairncross, *The Death of Distance*, Boston, MA: Harvard Business School Press, 1997.
5. E. Schonfeld, Don't just sit there, do something. E-company, 2000, pp. 155–164.
6. Time Warner is pulling the plug on a visionary foray into Interactive TV, *Providence Journal*, May 11, 1997, A17.
7. N. Lockwood Tooher, The next big thing: Interactivity, *Providence Journal*, February 14, 2000, A1, 7.
8. S. Levy, The new digital galaxy, *Newsweek*, May 31, 1999,57–63.
9. B. Lee, Personal Technology, in *Red Herring*, **119**, 56–57, 2002.
10. A. Higgins, Jetsons, Here we come!, in *Machine Design*, **75** (7): 52–53, 2003.
11. L. Kolbe, *Informationstechnik für den privaten Haushalt (Information Technology for the Private Household)*, Heidelberg: Physica, 1997.
12. W. Brenner and L. Kolbe, Information processing in the private household, in *Telemat. Informat.* **12** (2): 97–110, 1995.
13. I. Miles, Home Informatics, Information Technology and the Transformation of Everyday Life, London, 1988.
14. A. T. Kearney, Cambridge business school mobile commerce study 2002. Available: http://www.atkearney.com/main.taf?p=1,5,1,106. Jan. 10. 2004.
15. Economist 2001: Looking for the pot of gold, in *Economist*, Special supplement: The Internet, untethered, October 13, 2001, 11–14.
16. O. Jüptner, Over five billion text messages sent in UK. Available: http://www.e-gateway.net/infoarea/news/news.cfm?nid=2415. January 9, 2003.
17. Jupiter Communications Company, *Consumer Information Appliance*, **5** (2): 2–23, 1994.
18. P. Grimaldi, Net Retailers have season of success. *Providence Journal*, December 27, 2003, B1, 2.
19. J. Lee, An end-user perspective on file-sharing systems, in *Communications of the ACM*, **46** (2): 49–53, 2003.
20. Mundorf, Distance learning and mobility, in IFMO, Institut für Mobilitätsforschung *Auswirkungen der virtuellen Mobilität* (Effects of virtual mobility), 2004, pp. 257–272.
21. N. Dholakia, N. Mundorf, R. R. Dholakia, and J. J. Xiao, Interactions of transportation and telecommunications behaviors, *University of Rhode Island Transportation Center* Research Report 536111.

FURTHER READING

N. Mundorf and P. Zoche, Nutzer, private Haushalte und Informationstechnik, in P. Zoche (ed.), *Herausforderungen für die Informationstechnik*, Heidelberg 1994, pp. 61–69.

A. Reinhardt, Building the data highway, in *Byte International Edition*, March 1994, pp. 46–74.

F. Van Rijn and R. Williams, Concerning home telematics, *Proc. IFIP TC* **9**, 1988.

NORBERT MUNDORF
University of Rhode Island
Kingston, Rhode Island
LUTZ KOLBE

REMOTE SENSING INFORMATION PROCESSING

INTRODUCTION

With the rapid advance of sensors for remote sensing, including radar, microwave, multispectral, hyperspectral, infrared sensors, and so on the amount of data available has increased dramatically from which detailed or specific information must be extracted. Information processing, which makes extensive use of powerful computers and techniques in computer science and engineering, has played a key role in remote sensing. In this article, we will review some major topics on information processing, including image processing and segmentation, pattern recognition and neural networks, data and information fusion, knowledge-based system, image mining, image compression, and so on. References (1–5) provide some useful references on information processing in remote sensing.

In remote sensing, the large amount of data makes it necessary to perform some type of transforms that preserve the essential information while considerably reducing the amount of data. In fact, most remote sensing image data are redundant, correlated, and noisy. Transform methods can help in three ways: by effective data representation, effective feature extraction, and effective image compression. Component analysis is key to transform methods. Both principal component analysis and independent component analysis will be examined for remote sensing.

IMAGE PROCESSING AND IMAGE SEGMENTATION

The motivation to enhance the noisy images sent back from satellites in the early 1960s has had significant impact on subsequent progress in digital image processing. For example, digital filtering such as Wiener filtering allows us to restore the original image from its noisy versions. Some new image processing, such as wavelet transforms and morphological methods, have been useful in remote sensing images. One important activity in remote sensing is the speckle reduction of SAR (synthetic aperture radar) images. Speckles appearing in SAR images is caused by the coherent interference of waves reflected from many elementary scatters. The statistics of SAR speckle has been well studied (6). Over a 100 articles have been published on techniques to remove the speckles. One of the most well-known techniques is the Lee's filter, which makes use of the local statistics (7). More recent studies of the subject are reported in Refs. 8 and 9.

Image restoration in remote sensing is required to remove the effects of atmospheric and other interference, as well as the noises presented by the sensors. A good example is the restoration of images from the Hubble Space Telescope.

Image segmentation attempts to define the regions and boundaries of the regions. Techniques are developed that preserve the edges and smooth out the individual regions. Image segmentation may involve pixel-by-pixel classification, which often requires using pixels of known classification for training. Segmentation may also involve region growing, which is essentially unsupervised. A good example is to extract precisely the lake region of Lake Mulargias on the island of Sardinia in Italy (10). The original image is shown in Fig. 1a. The segmentation result is shown in Fig.1b, for which the exact size of the lake can be determined. For land-based remote sensing images, pixel-by-pixel classification allows us to determine precisely the area covered by each crop and to assess the changes from one month to the next during the growing season. Similarly, flood damage can be determined from the satellite images. These examples are among the many applications of image segmentation.

PATTERN RECOGNITION AND NEURAL NETWORKS

A major topic in pattern recognition is feature extraction. An excellent discussion of feature extraction and selection problem in remote sensing with multispectral and hyperspectral images is given by Landgrebe (5). In remote sensing, features are usually taken from the measurements of spectral bands, which this means 6 to 8 features in multispectral data, but a feature vector dimension of several hundred in hyperspectral image data. With a limited number of training samples, increasing the feature dimension in hyperspectral images may actually degrade the classification performance, which is referred to as the Hughes phenomenon. Reference (5) presents procedures to reduce such phenomena. Neural networks have found many uses in remote sensing, especially with pattern classification. The back-propagation trained network, the radial basis function network, and the support vector machine are the three best-performing neural networks for classification. A good discussion on statistical and neural network methods in remote sensing classification is contained in Ref. 11 as well as many other articles that appear in Refs. 3, 4, and 12. A major advantage of neural networks is that learning is from the training data only, and no assumption of the data model such as probability density is required. Also, it has been found that combining two neural network classifiers such as combining SOM, the self-organizing map, with a radial basis function network can achieve better classification than either one used alone (13).

One problem that is fairly unique and significant to remote sensing image recognition is the use of contextual information in pattern recognition. In remote sensing

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

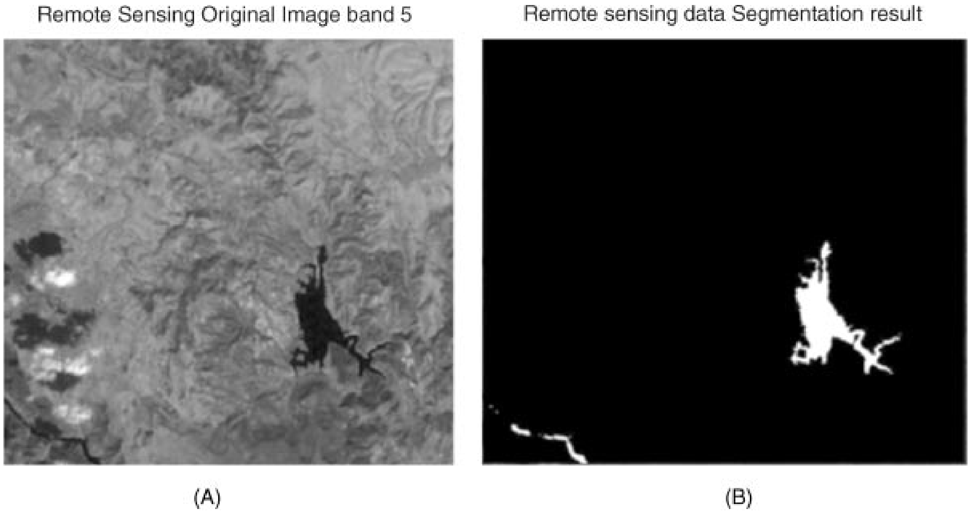

Remote Sensing Original Image band 5
Remote sensing data Segmentation result
(A)
(B)

Figure 1. Original image of Lake Mulargias region in Italy (A) and the result of region growing to extract the lake area (B).

image data, there is a large amount of contextual information that must be used to improve the classification. The usual procedure for contextual pattern recognition is to work with image models that exploit the contextual dependence. Markov random field models are the most popular, and with only a slightly increased amount of computation, classification performance can be improved with the use of such models (2,4,12).

Another pattern recognition topic is the change detection. The chapters by Serpico and Bruzzone (14) and Moser et al.(15) are recommended reading.

DATA FUSION AND KNOWLEDGE-BASED SYSTEMS

In remote sensing, there are often data from several sensors or sources. There is no optimum or well-accepted approach to the problem. Approaches can range from more theoretic, like consensus theory (16), to fuzzy logic, neural networks, multistrategy learning to fairly ad hoc techniques in some knowledge-based system to combine or merge information from different sources. In some cases, fusion at decision level can be more effective than fusion at data or feature level, but the other way can be true in other cases. Readers are referred to chapters by Solaiman (17), Benediktsson and Kanellopoulos (18), and Binaghi et al. (19) for detailed discussion.

IMAGE MINING, IMAGE COMPRESSION, AND WAVELET ANALYSIS

To extract certain desired information from a large remote sensing image database, we have the problem of data mining. For remote sensing images, Aksoy et al. (20) describe a probabilistic visual grammar to automatically analyze complex query scenarios using spatial relationships of regions, and to use it for content-based image retrieval and classification. Their hierarchical scene modeling bridges the gap between feature extraction and semantic interpretation. For image compression of hyperspectral images, Qian et al. (21) provide a survey of major approaches including vector quantization, discrete cosine transform and wavelet transform, and so on. A more comprehensive survey of remote sensing image compression is provided by Aiazzi et al.(22). Besides its important capability in image compression, wavelet transform has a major application in de-noising SAR images (23). The wavelet analysis of SAR images can also be used for near-real-time “quick look” screening of satellite data, data reduction, and edge linking (24).

COMPONENT ANALYSIS

Transform methods are often employed in remote sensing. A key to the transform methods is the component analysis, which, in general, is the subspace analysis. In remote sensing, component analysis includes the principal component analysis (PCA), curvilinear component analysis (CCA), and independent component analysis (ICA). The three component analysis methods are different conceptually. PCA is to look for the principal components according to the second-order statistics. CCA performs nonlinear feature space transformation while trying to preserve as much as possible the original data information in the lower-dimensional space see Ref. 25. ICA looks for independent components from the original data assumed to be linearly mixed from several independent sources. Nonlinear PCA that makes use of the higher-order statistical information (26) can provide an improvement over the linear PCA that employs only the second-order covariance information.

ICA is a useful extension of the traditional PCA. Whereas PCA attempts to decorrelate the components in a vector, ICA methods are to make the components as independent as possible. There are currently many approaches available for ICA (27). ICA applications in remote sensing study have become a new topic in recent years. S. Chiang et al. employed ICA in AVIRIS (airborne visible infrared imaging spectrometer) data analysis (28). T. Tu used a noise-adjusted version of fast independent component analysis (NAFICA) for unsupervised signature extraction and separation in hyperspectral images (29). With remote sensing in mind, we developed a new (ICA) method that makes use of the higher-order statistics The work is quite different from that of Cardoso (30). We name it the joint cumulant ICA (JC-ICA) algorithm (31,32). It can be implemented efficiently by a neural network. Experimental evidence (31) shows that, for the SAR image pixel classification, a small subset of ICA features perform a few percentage points better than the use of original data or PCA as features. The significant component images obtained by ICA have less speckle noise and are more informative. Furthermore, for hyperspectral images, ICA can be useful for selecting or reconfiguring spectral bands so that the desired objects in the images may be enhanced (32). Figures. 2 and 3 show, respectively, an original AVIRIS image and the enhanced image using the JC-ICA approach. The latter has more desired details.

CONCLUSION

In this article, an overview is presented of a number of topics and issues on information processing for remote sensing. One common theme is the effective use of computing power to extract the desired information from the large amount of data. The progress in computer science and engineering certainly presents many new and improved procedures for information processing in remote sensing.

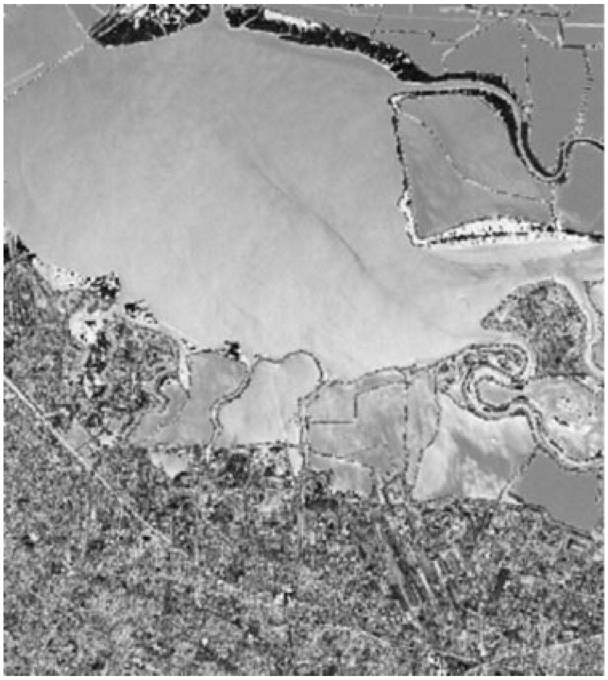


Figure 2. An AVIRIS image of Moffett field.

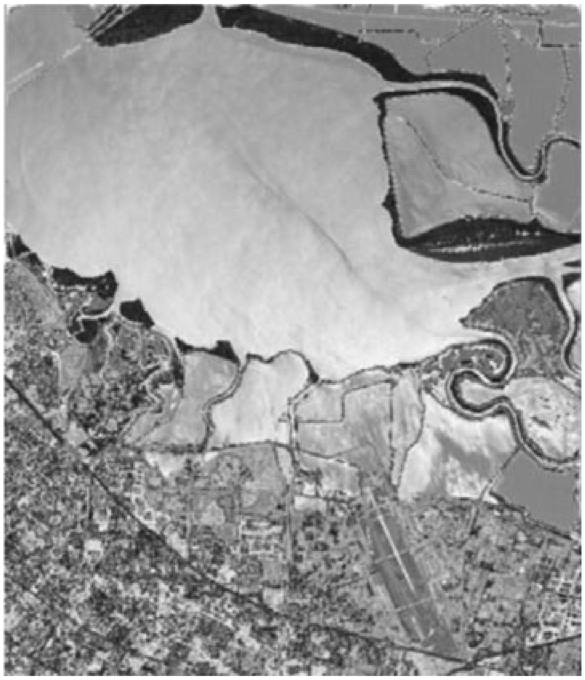


Figure 3. Enhanced image using JC-ICA.

BIBLIOGRAPHY

1. J. A. Richard and X. Jin, Remote sensing digital image analysis, 3rd ed., New York: Springer, 1991.
2. R. A. Schowengerdt, Remote sensing: Models and methods for image processing, New York: Academic Press, 1977.
3. C. H. Chen (ed.), Information processing for remote sensing, Singapore: World Scientific Publishing, 1999.
4. C. H. Chen (ed.), Frontiers of remote sensing information processing, Singapore, World Scientific Publishing, 2003.
5. D. Landgrebe, Signal theory methods in multispectral remote sensing, New York: Wiley, 2003.
6. J. S. Lee, et al., Speckle filtering of synthetic aperture radar images: a review, *Remote Sens. Rev.*, **8**: 313–340, 1994.
7. J. S. Lee, Digital image enhancement and noise filtering by use of local statistics, *IEEE Trans. Pattern Anal. Machine Intell.*, **2**(2): 165–168, 1980.
8. J. S. Lee, Speckle suppression and analysis for synthetic aperture radar images, *Op. Eng.*,**25**(5): 636–643, 1996.
9. J. S. Lee and M. Grunes, Polarimetric SAR speckle filtering and terrain classification-an overview, in C. H. Chen (ed.), *Information processing for remote sensing*, Singapore: World Scientific Publishing, 1999.
10. P. Ho and C. H. Chen, On the ARMA model based region growing method for extracting lake region in a remote sensing image, *SPIE Proc.*, Sept. 2003.
11. J. A. Benediktsson, On statistical and neural network pattern recognition methods for remote sensing applications, in C. H. Chen et al. (eds.), *Handbook of Pattern Recognition and Compter Vision*, 2nd ed., (ed.) Singapore: World Scientific Publishing, 1999.
12. E. Binaghi, P. Brivio, and S. B. Serpico, (eds.), *Geospatial Pattern Recognition*, Research Signpost, 2002.

13. C. H. Chen and B. Sherestha, Classification of multi-source remote sensing images using self-organizing feature map and radial basis function networks, *Proc. of IGARSS* 2000.
14. S. B. Serpico and L. Bruzzone, Change detection, in C. H. Chen (ed.), Information processing for remote sensing, Singapore: World Scientific Publishing, 1999.
15. G. Moser, F. Melgani and S. B. Serpico, Advances in unsupervised change detection, Chapter 18 in C. H. Chen (ed.), *Frontiers of remote sensing information processing*, Singapore: World Scientific Publishing, 2003.
16. J. A. Benediktsson and P. H. Swain, Consensus theoretic classification methods, *IEEE Trans. Syst. Man Cybernet.*, **22**(4): 688–704, 1992.
17. B. Solaiman, Information fusion for multispectral image classificatin post processing, in C. H. Chen (ed.), Information processing for remote sensing, Singapore: World Scientific Publishing, 1999.
18. J. A. Benediktsson and I. Kanellopoulos, Information extraction based on multisensor data fusion and neural networks, in C. H. Chen (ed.), Information processing for remote sensing, Singapore: World Scientific Publishing, 1999.
19. E. Binaghi, et al., Approximate reasoning and multistrategy learning for multisource remote sensing daa interpretation, in C. H. Chen (ed.), Information processing for remote sensing, Singapore: World Scientific Publishing, 1999.
20. S. Aksoy, et al., Scene modeling and image mining with a visual grammar, Chapter 3 in C. H. Chen (ed.), Frontiers of remote sensing information processing, Singapore, World Scientific Publishing, 2003.
21. S. Qian and A. B. Hollinger, Lossy data compression of 3-dimensional hyperspectral imagery, in C. H. Chen (ed.), Information processing for remote sensing, Singapore: World Scientific Publishing, 1999.
22. B. Aiazzi, et al., Near-lossless compression of remote-sensing data, Chapter 23 in C. H. Chen (ed.), Frontiers of remote sensing information processing, Singapore, World Scientific Publishing, 2003.
23. H. Xie, et al., Wavelet-based SAR speckle filters, Chapter 8 in C. H. Chen (ed.), Frontiers of remote sensing information processing, Singapore, World Scientific Publishing, 2003.
24. A. Liu, et al., Wavelet analysis of satellite images in ocean applications, Chapter 7 in C. H. Chen (ed.), Frontiers of remote sensing information processing, Singapore, World Scientific Publishing, 2003.
25. M. Lennon, G. Mercier, and M. C. Mouchot, Curvilinear component analysis for nonlinear dimensionality reduction of hyperspectral images, *SPIE Remote Sensing Symposium Conference 4541, Image and Signal Processing for Remote Sensing VII*, Toulouse, France, 2001.
26. E. Oja, The nonlinear PCA learning rule in independent component analysis, *Neurocomputing*, **17**(1): 1997.
27. A. Hyvarinen, J. Karhunen, and E. Oja, *Independent Component Analysis*, New York: Wiley, 2001.
28. S. Chiang, et al., Unsupervised hyperspectral image analysis using independent component analysis, *Proc. of IGARSS 2000*, Hawaii, 2000.
29. T. Tu, Unsupervised signature extraction and separation in hyperspectral images: A noise-adjusted fast independent component analysis approach, *Opt. Eng.*, **39**: 2000.
30. J. Cardoso, High-order contrasts for independent component analysis, *Neural Comput.*, **11**: 157–192, 1999.
31. X. Zhang and C. H. Chen, A new independent component analysis (ICA) method and its application to remote sensing images, *J. VLSI Signal Proc.*, **37** (2/3): 2004.
32. X. Zhang and C. H. Chen, On a new independent component analysis (ICA) method using higher order statistics with application to remote sensing image, *Opt. Eng.*, July 2002.

C. H. Chen
University of Massachusetts,
Dartmouth
North Dartmouth, Massachusetts

ROBOT KINEMATICS

INTRODUCTION

Robot kinematics is the study of the motion (kinematics) of robots. In a kinematic analysis the position, the velocity and acceleration of all links are calculated with respect to a fixed reference coordinate system without regard to the forces or moments that cause the motion. The relationship between motion and the associated forces and torques is studied in robot dynamics. Robot kinematics mainly includes two components: forward kinematics and inverse kinematics.

Forward kinematics is also known as direct kinematics. Forward kinematics is the static geometrical problem of computing the position and orientation of the end-effector of the manipulator. Specifically, given a set of joint motion, the forward kinematics problem is to compute the position and orientation of the tool frame relative to the base frame.

In inverse kinematics, given the position and orientation of the end-effector of the manipulator, all possible sets of joint motion are calculated that could be used to attain this given position and orientation. This issue is a fundamental problem in the practical use of manipulators.

From the viewpoint of robot mechanism, robots can be divided into two types, which are called serial robots and pareller robots. Serial robots are the most widely used robots in the industry. They can be observed throughout the manufacturing industry, as well as in automotive, aerospace, and commercial use. Serial robots can be manufactured for multiple operations that range from material processing operations to assembly operations. Serial manipulators are modeled and designed as anthropomorphic mechanical arms. Depending on the application, it can be outfitted with revolute and prismatic joints. This type of robot has what is called an open kinematic, chain which can be classified as either Articulated or Cartesian robots.

The parallel manipulators have some significant advantages over the conventional serial manipulators, such as more rigidity, accuracy, and high-force/torque capacity. They also have simpler inverse kinematics, which is an advantage in real-time control. They have been used in situations in which the demand on workspace and maneuverability is low but the dynamic loading is severe, and high-speed and precision motions are of primary concern. Recently, parallel manipulators have been developed for applications in aircraft simulators (1), telescopes (2), positioning trackers (3), micro-motion devices (4), and machine tools (5–8). However, because the theories and technologies for parallel manipulators are premature, most parallel manipulators that exist today are high-cost machines that provide less accuracy than conventional machines. Therefore, additional exploration is required to make parallel manipulators more attractive to the industry (9).

In this article, the basic mathematical concepts, including translational coordinate transformation, rotational coordinate transformation, and homogeneous transformation, are introduced in the section entitled Mathematical Fundamentals. The section on solving kinematics equation presents kinematics equation of robot manipulator focusing on the solution of the posture of joints and manipulator. The case study examines the kinematics analysis of a 3DOF parallel manipulator used for machining application. The last section concludes the article.

MATHEMATICAL FUNDAMENTALS

To explain the relationship between parts, tools, and manipulators, some concepts as position vector, plane, and coordinate frame should be used (10).

Presentation of Posture

The motion of robots can be described by its position and orientation, which is called its posture. Once the reference coordinate system has been built, any point in space can be expressed by one (3×1) vector. To an orthogonal coordinate system $\{O_a - x_a y_a z_a\}$, any point p in the space can be written as follows:

$$^{a}p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{1}$$

where $p_x,\ p_y,\ p_z$ denote the components of the vector p along the coordinate axis x_a, y_a, z_a, respectively. Here, p is called a position vector, which is shown in Fig. 1.

To research the motion and operation of robots, the expression of position and the orientation are needed. To prescribe the orientation of point b, one must assume an orthogonal coordinate system $\{O_b - x_b y_b z_b\}$ is attached to the point. Here, x_b, y_b, z_b denote the unit vectors of the coordinate axes. With respect to the reference coordinate system $\{O_a - x_a y_a z_a\}$, the orientation of point b is expressed as follows:

$$^{a}_{b}R = [\,^{a}x_b \quad ^{a}y_b \quad ^{a}z_b\,] = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2}$$

where $^{a}_{b}R$ is called the rotation matrix. $^{a}_{b}R$ has nine elements in total, but only three of them are independent. The following constraint conditions should be satisfied by the nine elements:

$$^{a}x_b \cdot {^{a}x_b} = {^{a}y_b} \cdot {^{a}y_b} = {^{a}z_b} \cdot {^{a}z_b} = 1 \tag{3}$$

$$^{a}x_b \cdot {^{a}y_b} = {^{a}y_b} \cdot {^{a}z_b} = {^{a}z_b} \cdot {^{a}x_b} = 0 \tag{4}$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

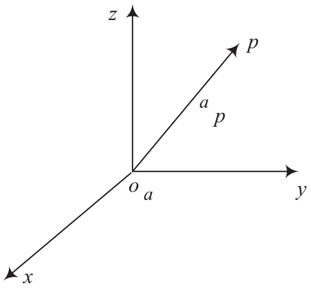


Figure 1. Presentation of position.

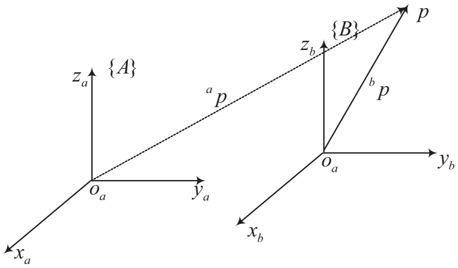


Figure 2. Translational transformation.

It can be concluded it the rotation matrix ${}^a_bR$ is orthogonal, and the following condition should be satisfied:

$$ {}^a_bR^{-1} = {}^a_bR^T; \quad |{}^a_bR| = 1 \tag{5}$$

The rotation matrix with respect to the rotation transformation by an angle θ about the axis x, y, z, respectively can be calculated as follows:

$$R(x,\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & -s\theta \\ 0 & s\theta & c\theta \end{bmatrix} \tag{6}$$

$$R(y,\theta) = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \tag{7}$$

$$R(z,\theta) = \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

where $s\theta = \sin\theta$ and $c\theta = \cos\theta$.

Coordinate Transformation

The description of any point p in space is different in different coordinate frames. To illustrate the relationship of the transformation from one coordinate frame to another, the mathematic expression will be used.

Translational Coordinate Transformation. Suppose that coordinate frames {B} and {A} have the same orientation. But the original points of the two coordinate frames do not overlap. The position vector $^a p_{o_b}$ is used to describe the position related to frame {A}. $^a p_{o_b}$ is called the *translational vector* of frame {B} with respect to frame {A}. If the position of point p in the coordinate frame {B} is written as $^b p$, then the position vector of p with respect to frame {A} can be written as follows:

$$^a p = {}^b p + {}^a p_{o_b} \tag{9}$$

It is called the equation of coordinate translation, which is shown in Fig. 2.

Rotational Coordinate Transformation. Suppose that coordinate frames {B} and {A} have the same original points, but their orientation is different. Using the rotation matrix ${}^a_bR$ to describe the orientation of frame {B} with respect to frame {A}, the transformation of point p in frame {A} and {B} can be deduced as:

$$^a p = {}^a_bR \cdot {}^b p \tag{10}$$

where $^a p$ denotes the position the p with the reference coordinate system {A}, and $^b p$ denotes the position the p with the reference coordinate system {B}. It is called the equation of coordinate rotation, which is shown in Fig. 3.

The following equations can be deduced:

$${}^b_aR = {}^a_bR^{-1} = {}^a_bR^T \tag{11}$$

Composite Transformation. For the common condition, neither the original points of frame {A} and {B} overlap, nor do they have the same orientation. Use the position vector $^a p_{o_b}$ to describe the original point of frame {B} with respect to frame {A}. Use the rotation matrix ${}^a_bR$ to describe the orientation of frame {B} respect to frame {A}. To any point in the space, the following transformation can be found:

$$^a p = {}^a_bR \cdot {}^b p + {}^a p_{o_b} \tag{12}$$

Homogeneous Coordinate Transformation

If the coordinates of any point in an orthogonal coordinate system are given, then the coordinates of this point in

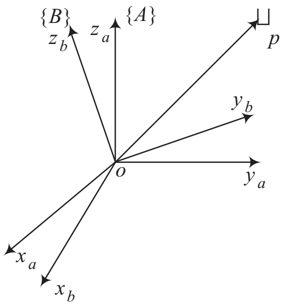


Figure 3. Rotational transformation.

another orthogonal coordinate system can be calculated by homogeneous coordinate transformation.

Homogeneous Transformation. The transformation formula in Equation (12) is unhomogeneous to point ${}^b p$, but it can be expressed by the following equivalent homogeneous transformation:

$$\begin{bmatrix} {}^a p \\ 1 \end{bmatrix} = \begin{bmatrix} {}^a_b R & {}^a p_{o_b} \\ 0_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} {}^b p \\ 1 \end{bmatrix} \tag{13}$$

where the vector (4×1) denotes the coordinates in three-dimensional space. It still can be noted as ${}^a p$ or ${}^b p$. The above equation can be written as the form of matrix:

$${}^a p = {}^a_b T \cdot {}^b p \tag{14}$$

where the vector (4×1) of ${}^a p$ and ${}^b p$ is called a homogeneous coordinate. Here:

$${}^a_b T = \begin{bmatrix} {}^a_b R & {}^a p_{o_b} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{15}$$

In fact, the transformation formula in Equation (13) is equivalent with Equation (12). The formula in Equation (13) can be written as follows:

$${}^a p = {}^a_b R \cdot {}^b p + {}^a p_{o_b;} \quad 1 = 1 \tag{16}$$

Are position vectors ${}^a p$ and ${}^b p$ the 3×1 orthogonal coordinate or the 4×1 homogeneous coordinate? It should be decided according to the context.

Translational Homogeneous Coordinate Transformation. Suppose that the vector $ai + bj + ck$ describes one point in space, where i, j, k is the unit vector of the axes x, y, z, respectively. This point can be expressed by translational homogeneous transformation:

$$\mathrm{Trans}(a, b, c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

where Trans denotes translational transformation.

Rotational Homogeneous Coordinate Transformation. Rotation about the x-axis, y-axis, and z-axis with θ, the following equations can be obtained:

$$\mathrm{Rot}(x, \theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta & -s\theta & 0 \\ 0 & s\theta & c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{18}$$

$$\mathrm{Rot}(y, \theta) = \begin{bmatrix} c\theta & 0 & s\theta & 0 \\ 0 & 1 & 0 & 0 \\ -s\theta & 0 & c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{19}$$

$$\mathrm{Rot}(z, \theta) = \begin{bmatrix} c\theta & -s\theta & 0 & 0 \\ s\theta & c\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

where Rot denotes rotational transformation.

Transformation of Object

The approach used to describe the posture of any point in the space can be used to describe the position and oriention in space. For example, the following Fig. 4(a) can be expressed by the six points attached to the reference frame.

If, at first, this object rotates 90 degrees is about the y-axis of the reference frame, then it translates 4 unit lengths. This transformation can be described as follows:

$$T = \mathrm{Trans}(4, 0, 0)\mathrm{Rot}(y, 90)\mathrm{Rot}(z, 90) = \begin{bmatrix} 0 & 0 & 1 & 4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The above matrix shows the operations of rotation and translation about the primary reference frame. The six

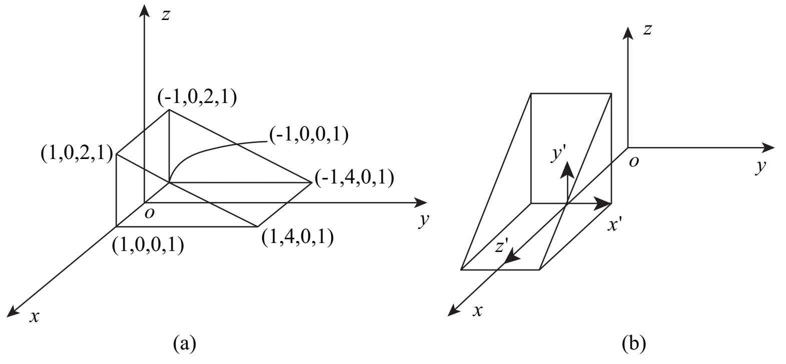


Figure 4. Transformation of wedge-shaped object.

points of the wedge-shaped object can be transformed as:

$$\begin{bmatrix} 0 & 0 & 1 & 4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 4 & 6 & 6 & 4 & 4 \\ 1 & -1 & -1 & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 4(b) shows the result of transformation.

General Rotation Transformation

In the above sections, the rotational transformation matrix with respect to rotation about x-axis, y-axis, and z-axis have been analyzed. Next, the focus is on the rotation matrix in the common situation: rotation about any vector (axis) with θ.

The Formula of General Rotation Transformation. Assume that f is the unit vector of the z-axis in coordinate frame {C}, namely:

$$C = \begin{bmatrix} n_x & o_x & a_x & 0 \\ n_y & o_y & a_y & 0 \\ n_z & o_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{21}$$

$$f = a_x i + a_y j + a_z k \tag{22}$$

Therefore, rotation about vector f is equivalent to rotation about z-axis in coordinate frame {C}, thus one obtains the following:

$$\mathrm{Rot}(f, \theta) = \mathrm{Rot}(c, \theta) \tag{23}$$

If the coordinate frame $\{T\}$ is known with respect to reference coordinate frame, then another coordinate frame $\{S\}$ can be calculated with respect to frame {C}, because:

$$T = \mathrm{C}S \tag{24}$$

Where, S expresses the relative position of T with respect to C. Then:

$$S = \mathrm{C}^{-1}T \tag{25}$$

The rotation of T about f is equivalent to the rotation of S about z-axis of frame {C}:

$$\mathrm{Rot}(f, \theta)T = \mathrm{CRot}(z, \theta)S \tag{26}$$

$$\mathrm{Rot}(f, \theta)T = \mathrm{CRot}(z, \theta)\mathrm{C}^{-1}T \tag{27}$$

Then the following equation can be derived:

$$\mathrm{Rot}(f, \theta) = \mathrm{CRot}(z, \theta)\mathrm{C}^{-1} \tag{28}$$

As f is the z-axis of frame {C}, it can be found that $\mathrm{Rot}(z, \theta)\mathrm{C}^{-1}$ is just the function of f, because:

$$\mathrm{C}\,Rot(z, \theta)\mathrm{C}^{-1}$$

$$= \begin{bmatrix} n_x & o_x & a_x & 0 \\ n_y & o_y & a_y & 0 \\ n_z & o_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta & -s\theta & 0 & 0 \\ s\theta & c\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_x & n_y & n_z & 0 \\ o_x & o_y & o_z & 0 \\ a_x & a_y & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} n_x & o_x & a_x & 0 \\ n_y & o_y & a_y & 0 \\ n_z & o_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_x c\theta - o_x c\theta & n_y c\theta - a_y s\theta & n_z c\theta - o_z s\theta & 0 \\ n_x s\theta + o_x c\theta & n_y s\theta + a_y c\theta & n_z s\theta + o_z c\theta & 0 \\ a_x & a_y & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} n_x n_x c\theta - n_x o_x s\theta + n_x o_x s\theta + o_x o_x c\theta + a_x a_x & n_x n_y c\theta - n_x o_y s\theta + n_y o_x s\theta + o_y o_x c\theta + a_x a_y & n_x n_z c\theta - n_x o_z s\theta - n_z o_x s\theta + o_z o_x c\theta + a_x a_z & 0 \\ n_y n_x c\theta - n_y o_x s\theta + n_x o_y s\theta + o_y o_x c\theta + a_y a_x & n_y n_y c\theta - n_y o_y s\theta + n_y o_y s\theta + o_y o_y c\theta + a_y a_y & n_y n_z c\theta - n_y o_z s\theta + n_z o_y s\theta + o_z o_y c\theta + a_y a_z & 0 \\ n_z n_x c\theta - n_z o_x s\theta + n_x o_z s\theta + o_z o_x c\theta + a_z a_x & n_z n_y c\theta - n_z o_y s\theta + n_y o_z s\theta + o_y o_z c\theta + a_z a_y & n_z n_z c\theta - n_z o_z s\theta + n_z o_z s\theta + o_z o_z c\theta + a_z a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{29}$$

Note that $z = a, vers\theta = 1 - c\theta,\ f = z$. Equation (29) can be simplified as follows:

$$\mathrm{Rot}(f, \theta) = \begin{bmatrix} f_x f_x vers\theta + c\theta & f_y f_x vers\theta - f_z s\theta & f_z f_x vers\theta + f_y s\theta & 0 \\ f_x f_y vers\theta + f_z s\theta & f_y f_y vers\theta + c\theta & f_z f_y vers\theta - f_x s\theta & 0 \\ f_x f_z vers\theta + f_z s\theta & f_y f_z vers\theta + f_x s\theta & f_z f_z vers\theta + c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{30}$$

Each basic rotation transformation can be derived from the general rotation transformation, that is, if $f_x = 1$, $f_y = 0$ and $f_z = 0$, then $\mathrm{Rot}(f, \theta) = \mathrm{Rot}(x, \theta)$. Equation (30) yields the following:

$$\mathrm{Rot}(x, \theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta & -s\theta & 0 \\ 0 & s\theta & c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{31}$$

which is identical with Equation (18)

The Equivalent Rotation Axis and Rotation Angle. Given any rotation transformationm, the equivalent rotation axis with the angle of θ can be calculated. Assume that the rotation transformationm matrix is as follows:

$$R = \begin{bmatrix} n_x & o_x & a_x & 0 \\ n_y & o_y & a_y & 0 \\ n_z & o_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{32}$$

If $R = \mathrm{Rot}(f, \theta)$, then

$$\begin{bmatrix} n_x & o_x & a_x & 0 \\ n_y & o_y & a_y & 0 \\ n_z & o_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x f_x vers\theta + c\theta & f_y f_x vers\theta - f_z s\theta & f_z f_x vers\theta + f_y s\theta & 0 \\ f_x f_y vers\theta + f_z s\theta & f_y f_y vers\theta + c\theta & f_z f_y vers\theta - f_x s\theta & 0 \\ f_x f_z vers\theta + f_z s\theta & f_y f_z vers\theta + f_x s\theta & f_z f_z vers\theta + c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{33}$$

By addition of the diagonal entry in two sides of the above equation, it is derived that:

$$n_x + o_y + a_z = (f_x^2 + f_y^2 + f_z^2) vers\theta + 3c\theta = 1 + 2c\theta \tag{34}$$

$$c\theta = \frac{1}{2}(n_x + o_y + a_z - 1) \tag{35}$$

By subtraction of the undiagonal entry, it yields the following:

$$\begin{aligned} o_z - a_y &= 2 f_x s\theta \\ a_x - n_z &= 2 f_y s\theta \\ n_y - o_x &= 2 f_z s\theta \end{aligned} \tag{36}$$

The sum of the squares of the above equation is as follows:

$$(o_z - a_y)^2 + (a_x - n_z)^2 + (n_y - o_x)^2 = 4s^2\theta \tag{37}$$

$$s\theta = \pm\frac{1}{2}\sqrt{(o_z - a_y)^2 + (a_x - n_z)^2 + (n_y - o_x)^2} \tag{38}$$

Define that the rotation mentioned before is the forward rotation with respect to vector f, and $0 \leq \theta \leq 180°$. In this case, the sign of Equation (38) is positive. Then, the rotation angle θ will be uniquely confirmed, as follows:

$$\tan\theta = \frac{\sqrt{(o_z - a_y)^2 + (a_x - n_z)^2 + (n_y - o_x)^2}}{n_x + o_y + a_z - 1} \tag{39}$$

Each component of vector f can be calculated by Equation (36):

$$\begin{aligned} f_x &= (o_z - a_y)/2s\theta \\ f_y &= (a_x - n_z)/2s\theta \\ f_z &= (n_y - o_x)/2s\theta \end{aligned} \tag{40}$$

SOLVING KINEMATICS EQUATION OF ROBOT MANIPULATOR

To operate and control a robot, we must have knowledge of both its spatial arrangement and a means of reference to the environment. Kinematics is the analytical study of the geometry of motion of a robot arm with respect to a fixed reference coordinate system without regard to the forces or moments that cause the motion. The issue of solving a kinematics equation of a robot manipulator focuses on the solution about the posture of joints and manipulators (10).

Posture Description of Motion

Description of Euler Angle. The Euler angle I, shown as Fig. 5, first defines a rotation of angle ϕ around the z-axis,

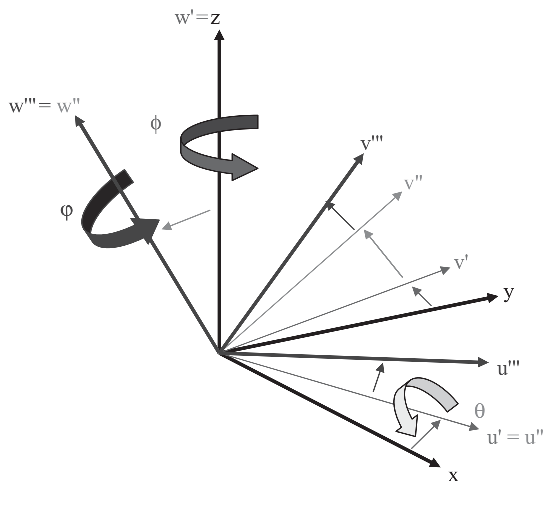


Figure 5. Euler angle I.

then a rotation of angle θ around the new x-axis, then a rotation of angle ψ around the new z-axis.

$$R_{z\phi} = \begin{pmatrix} \mathrm{c}\phi & -\mathrm{s}\phi & 0 \\ \mathrm{s}\phi & \mathrm{c}\phi & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$R_{u'\theta} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \mathrm{c}\theta & -\mathrm{s}\theta \\ 0 & \mathrm{s}\theta & \mathrm{c}\theta \end{pmatrix},$$

$$R_{w''\varphi} = \begin{pmatrix} \mathrm{c}\varphi & -\mathrm{s}\varphi & 0 \\ \mathrm{s}\varphi & \mathrm{c}\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{41}$$

The resultant Eulerian rotation matrix generates the following:

$$R = R_{2\phi}R_{u'\theta}R_{w''\varphi}\begin{pmatrix} \mathrm{c}\phi\,\mathrm{c}\varphi & -\mathrm{c}\phi\,\mathrm{s}\varphi & \\ -\mathrm{s}\phi\,\mathrm{s}\varphi\,\mathrm{c}\theta & -\mathrm{s}\phi\,\mathrm{c}\varphi\,\mathrm{c}\theta & \mathrm{s}\varphi\,\mathrm{s}\theta \\ \mathrm{s}\phi\,\mathrm{c}\varphi & -\mathrm{s}\phi\,\mathrm{s}\varphi & -\mathrm{c}\phi\,\mathrm{s}\theta \\ +\mathrm{c}\phi\,\mathrm{s}\varphi\,\mathrm{c}\theta & +\mathrm{c}\phi\,\mathrm{c}\varphi\,\mathrm{c}\theta & \\ \mathrm{s}\varphi\,\mathrm{s}\theta & \mathrm{c}\varphi\,\mathrm{s}\theta & \mathrm{c}\theta \end{pmatrix} \tag{42}$$

The Euler angle II, which is shown as Fig. 6, first defines a rotation of angle ϕ around the z-axis, then a rotation of angle θ around the new y-axis, then a rotation of angle ψ around the new z-axis.

Note the opposite (clockwise) sense of the third rotation, ϕ. The matrix with Euler angle II generates the following:

$$\begin{pmatrix} -\mathrm{s}\phi\,\mathrm{s}\varphi & -\mathrm{s}\phi\,\mathrm{c}\varphi & \\ +\mathrm{c}\phi\,\mathrm{c}\varphi\,\mathrm{c}\theta & -\mathrm{s}\phi\,\mathrm{c}\varphi\,\mathrm{c}\theta & \mathrm{c}\phi\,\mathrm{s}\theta \\ \mathrm{c}\phi\,\mathrm{s}\varphi & \mathrm{c}\phi\,\mathrm{c}\varphi & \mathrm{s}\varphi\,\mathrm{s}\theta \\ +\mathrm{s}\phi\,\mathrm{c}\varphi\,\mathrm{c}\theta & -\mathrm{s}\phi\,\mathrm{c}\varphi\,\mathrm{c}\theta & \\ -\mathrm{c}\varphi\,\mathrm{s}\theta & \mathrm{s}\varphi\,\mathrm{s}\theta & \mathrm{c}\theta \end{pmatrix} \tag{43}$$

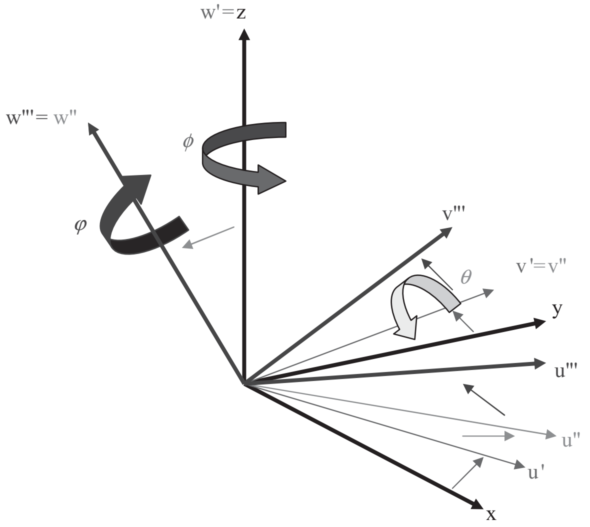


Figure 6. Euler angle II.

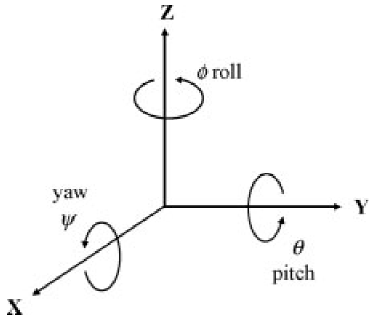


Figure 7. Rotation of raw, pitch and roll.

Description of Roll Pitch Yaw. Another common group of rotations is yaw, pitch, and roll, which are shown in Fig. 7. A rotation of ψ about the x-axis ($R_{x,\psi}$) is called yaw. A rotation of θ about the y-axis ($R_{y,\theta}$) is called pitch. A rotation of ϕ about the z-axis ($R_{z,\phi}$) is called roll.

$$\begin{aligned} &RPY(\phi,\theta,\psi) \\ &= \begin{bmatrix} c\phi & -s\phi & 0 & 0 \\ s\phi & c\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c\theta & 0 & s\theta & 0 \\ 0 & 1 & 0 & 0 \\ -s\theta & 0 & c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\psi & -s\psi & 0 \\ 0 & s\psi & c\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c\phi c\theta & c\phi s\theta s\psi - s\phi c\psi & c\phi s\theta c\psi + s\phi s\psi & 0 \\ s\phi c\theta & s\phi s\theta s\psi + c\phi c\psi & s\phi s\theta c\psi - c\phi s\psi & 0 \\ -s\phi & c\theta s\psi & c\theta c\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{44}$$

Link and Joint Parameters

The Denavit-Hartenberg representation is adopted to describe the motion parameters of link and joint. The following steps are used to confirm the parameters of link and joint:

- Number the joints from 1 to n starting with the base and ending with the end-effector.
- Establish the base coordinate system. Establish a right-handed orthonormal coordinate system (X_0, Y_0, Z_0) at the supporting base with the Z_0 axis lying along the axis of motion of joint 1.
- Establish joint axis. Align the Z_i with the axis of motion (rotary or sliding) of joint $i+1$.
- Establish the origin of the ith coordinate system. Locate the origin of the ith coordinate at the intersection of

the Z_i and Z_{i-1} or at the intersection of common normal between the Z_i and Z_{i-1} axes and the Z_i axis.

- Establish the X_i axis. Establish $X_i = \pm(Z_{i-1} \times Z_i)/\|Z_{i-1} \times Z_i\|$ or along the common normal between the Z_{i-1} and Z_i axes when they are parallel.
- Establish the Y_i axis. Assign $Y_i = +(Z_i \times X_i)/\|Z_i \times X_i\|$ to complete the right-handed coordinate system.
- Find the link and joint parameters.

Jacobian of Robot Manipulator

Different Motion of Robot. To describe the micromotion of a robot, differential coefficient is used for coordinate transformation. Given the coordinate frame {T},

$$T + \mathrm{d}T = \mathrm{Trans}(dx, dy, dz)\mathrm{Rot}(f, d\theta)T \tag{45}$$

where $\mathrm{Trans}(dx, dy, dz)$ denotes the differential translation of d_x, d_y, d_z, and $\mathrm{Rot}(f, d\theta)$ denotes the differential rotation about the vector f. Then dT can be calculated as follows:

$$dT = [\mathrm{Trans}(dx, dy, dz)\mathrm{Rot}(f, d\theta) - I]T \tag{46}$$

The homogeneous transformation that expresses differential translation is

$$\mathrm{Trans}(\mathrm{d}x, \mathrm{d}y, \mathrm{d}z) = \begin{bmatrix} 1 & 0 & 0 & \mathrm{d}x \\ 0 & 1 & 0 & \mathrm{d}y \\ 0 & 0 & 1 & \mathrm{d}z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{47}$$

For the formula of general rotation transformation

$$\mathrm{Rot}(f, \theta) = \begin{bmatrix} f_x\, f_x vers\theta + c\theta & f_y\, f_x vers\theta - f_z s\theta & f_z\, f_x vers\theta + f_y s\theta & 0 \\ f_x\, f_y vers\theta + f_z s\theta & f_y\, f_y vers\theta + c\theta & f_z\, f_y vers\theta - f_x s\theta & 0 \\ f_x\, f_z vers\theta + f_z s\theta & f_y\, f_z vers\theta + f_x s\theta & f_z\, f_z vers\theta + c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{48}$$

Because $\lim_{\theta\to 0}\mathrm{Sin}\,\theta = d\theta, \lim_{\theta\to 0}\cos\theta = 1, \lim_{\theta\to 0} vers\theta = 0$, the differential rotational homogeneous transformation can be expressed as follows:

$$\mathrm{Rot}(f, d\theta) = \begin{bmatrix} 1 & -fzd\theta & fyd\theta & 0 \\ fzd\theta & 1 & -fxd\theta & 0 \\ -fzd\theta & fxd\theta & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{49}$$

Because $\Delta = \mathrm{Trans}(dx, dy, dz)\mathrm{Rot}(f, d\theta)$, it yields the following:

$$\Delta = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -f_z d\theta & f_y d\theta & 0 \\ f_z d\theta & 1 & -f_x d\theta & 0 \\ -f_y d\theta & f_x d\theta & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -f_z d\theta & f_y d\theta & dx \\ f_z d\theta & 0 & -f_x d\theta & dy \\ -f_y d\theta & f_x d\theta & 0 & dz \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{50}$$

The differential rotation $\mathrm{d}\theta$ about vector f is equivalent to the differential rotation with respect to the x-axis, y-axis and z-axis, namely δ_x, δ_y, and δ_z, respectively. Then $f_x\mathrm{d}\theta = \delta_x$, $f_y\mathrm{d}\theta = \delta_y$, $f_z\mathrm{d}\theta = \delta_z$. Displace the above results into Equation (50), which yields the following:

$$\Delta = \begin{bmatrix} 0 & -\delta z & \delta y & dx \\ \delta z & 0 & -\delta x & dy \\ -\delta y & \delta x & 0 & dz \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{51}$$

If $d = d_x i + d_y j + d_z k, \delta = \delta_x i + \delta_y j + \delta_z k$, then the differential motion vector of rigid body or coordinate frame can be expressed as follows:

$$D = [dx \quad dy \quad dz \quad \delta x \quad \delta y \quad \delta z]^T = \begin{bmatrix} d \\ \delta \end{bmatrix} \tag{52}$$

Jacobian Matrix of Robot. The linear transformation between the motion speed of the manipulator and each joint can be defined as the Jacobian matrix of the robot. This Jacobian matrix indicates the drive ratio of motion velocity from the space of joints to the space of end-effector. Assume that the motion equation of the manipulator

$$x = x(q) \tag{53}$$

represents the displacement relationship between the space of the operation (end-effector) and the space of the joints. Differentiating Equation (53) with respect to time yields the following:

$$\dot{x} = J(q)\dot{q} \tag{54}$$

where $\dot{x}$ is the generalized velocity of end-effector in operating space. $\dot{q}$ is the joint velocity. $J(q)$ is $6 \times n$ partial derivative matrix, which is called the Jacobian Matrix. The component in line i and column j is as follows:

$$J_{ij}(q) = \frac{\partial x_i(q)}{\partial q_j}, i = 1, 2, \cdots, 6; j = 1, 2, \cdots, n \tag{55}$$

From Equation (55), it can be known that the Jacobian Matrix $J(q)$ is the linear transformation from the velocity of joint space and the velocity of operating space.

The generalized velocity $\dot{x}$ of a rigid board or a coordinate frame is a six-dimensional column vector composed of linear velocity v and angular velocity w.

$$\dot{x} = \begin{bmatrix} v \\ w \end{bmatrix} = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \begin{bmatrix} d \\ \delta \end{bmatrix} \tag{56}$$

Equation (56) yields:

$$D = \begin{bmatrix} d \\ \delta \end{bmatrix} = \lim_{\Delta t \to 0} \dot{x} \Delta t \tag{57}$$

Displace Equation (54) into the above equation, and the following is obtained:

$$D = \lim_{\Delta t \to 0} J(q)\dot{q}\Delta t \tag{58}$$

$$D = J(q)\mathrm{d}q \tag{59}$$

For a robot with n joints, its Jacobian matrix is a $6 \times n$ matrix, in which the first three lines denote the transferring rate of end-effector's linear velocity, and the last three lines denote the transferring rate of end-effector's angular effector velocity. The Jacobian matrix can be expressed as follows:

$$\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} J_{l1} & J_{l2} & \cdots & J_{ln} \\ J_{a1} & J_{a2} & \cdots & J_{an} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} \tag{60}$$

The linear velocity and angular velocity of end-effector can be expressed as the linear function of each joint velocity q

$$\left. \begin{aligned} v &= J_{l1}\dot{q}_1 + J_{l2}\dot{q}_2 + \cdots + J_{ln}\dot{q}_n \\ w &= J_{a1}\dot{q}_1 + J_{a2}\dot{q}_2 + \cdots + J_{an}\dot{q}_n \end{aligned} \right\} \tag{61}$$

where J_{li} and J_{ai} means the linear velocity and angular velocity of end-effector that results in joint i.

CASE STUDY: KINEMATIC ANALYSIS OF A PARALLEL MANIPULATOR

This example studies the kinematics analysis of a 3DOF parallel manipulator used for a machining application. The new design of this manipulator aims to achieve higher stiffness and to implement pure 3DOF motions (i.e., all side-effect motions can be eliminated). The manipulator consists of three identical legs with active ball screw actuators, and a passive leg is installed between the base and the moving platform. Each actuated leg is connected with the moving platform by a spherical joint, and the passive link is fixed on the ground and connected to the moving platform by a universal joint. Thus, the side-effect motions can be eliminated by the u-joint.

Description of the 3DOF Parallel Manipulator

Different from most existing 3DOF parallel manipulators, this design has a hybrid and uncoupled motion. The objective of the new design is to improve the system stiffness as well as to eliminate the coupled motions at the reference point to simplify the kinematic model and the control.

In Fig. 8, the novelty of the manipulator is as follows: (*1*) the universal joint of the passive link is located on the moving platform rather than on the base platform; thus, the motion along x and y translations and z rotation is eliminated; and (*2*) the reference point on the moving platform has a hybrid and uncoupled motion with x and y rotations and z translation.

The proposed manipulator has three platforms: base platform $B_1B_2B_3$, middle platform $M_1M_2M_3$, and moving platform $E_1E_2E_3$. The base platform is fixed on the ground. The middle platform is used to support the guideway B_iM_i of actuated links D_iE_i. The moving platform is used to mount a machine tool. The passive link is installed between the middle platform and the moving platform. Actuated link D_iE_i is connected to the moving platform by a spherical joint at E_i and to a slider connected to the active ball screw by a universal joint at D_i. The passive link is fixed on the

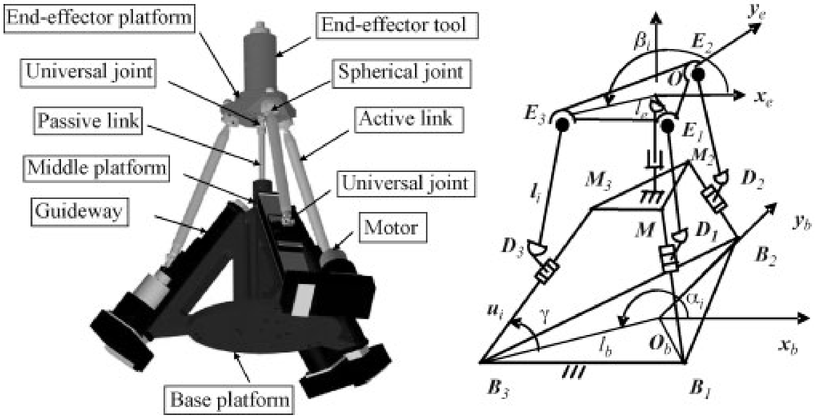


Figure 8. The CAD and Schematic model of 3DOF parallel manipulator.

middle platform at one end, and it is connected to the end-effector platform by a universal joint at the other end.

The following parameters are required for its description:

- the angle $\alpha_i (i = 1, 2, 3)$ between O_bB_i and x_b
- the angle $\beta_i (i = 1, 2, 3)$ between O_eE_i and x_e
- the size of the base platform l_b,
- the size of the end-effector platform l_e,
- the direction of a guideway γ,
- the length of an active link l_i, and
- the offset of the spherical joints on the platform z_0.

Inverse Kinematics

To describe the structure of the tripod system, two coordinate systems $\{\boldsymbol{O}_e - x_e y_e z_e\}$ and $\{\boldsymbol{O}_b - x_b y_b z_b\}$ are established, which are attached to the end-effector and base platforms, respectively. The following physical parameters are identified:

i) the angle α_i between $\boldsymbol{O}_b\boldsymbol{B}_i$ and $\boldsymbol{x}_b$,
ii) the angle β_i between $\boldsymbol{O}_e\boldsymbol{E}_i$ and $\boldsymbol{x}_e$,
iii) the dimension of the base platform l_b,
iv) the dimension of the end-effector platform l_e,
v) the direction of a guild bar γ, and
vi) the length of a support bar l_i.

For the original $\boldsymbol{O}_e$ of the end-effector, its translational motions along $\boldsymbol{x}_e$ and $\boldsymbol{y}_e$, and rotational motion along $\boldsymbol{z}_e$, are eliminated because of the usage of the passive leg, that is,

$$\left.\begin{aligned} x_e = y_e = 0 \\ z_\theta = 0 \end{aligned}\right\} \tag{62}$$

Therefore, the motions of $\boldsymbol{O}_e$ can be denoted by $(\theta_x, \theta_y, z_\theta)$, where θ_x and θ_y are the rotational motions along $\boldsymbol{x}_e$ and $\boldsymbol{y}_e$, and $\boldsymbol{z}_e$ is the translational motion along $\boldsymbol{z}_e$. The posture of the end-effector with respect to the coordinate system $\{\boldsymbol{O}_b - \boldsymbol{x}_b\boldsymbol{y}_b\boldsymbol{z}_b\}$ can be represented as follows:

$$T_e^b = \begin{bmatrix} \boldsymbol{R}_e & \boldsymbol{P}_e \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} c\theta_y & 0 & s\theta_y & 0 \\ s\theta_x s\theta_y & c\theta_x & -s\theta_x c\theta_y & 0 \\ -c\theta_x s\theta_y & s\theta_x & c\theta_x c\theta_y & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{63}$$

where

c, s denote the cosine and sine functions, respectively.
T_e^b is the posture of the end-effector with respect to the coordinate system $\{\boldsymbol{O}_b – \boldsymbol{x}_b\boldsymbol{y}_b\boldsymbol{z}_b\}$.
$\boldsymbol{R}_e$ is the 3×3 orientation matrix of the end-effector.
$\boldsymbol{P}_e$ is the location of $\boldsymbol{O_e}$.

Inverse kinematics are used to find the joint motions when the posture of the end-effector T_e^b is known. The joint motions are denoted by u_i, and the posture of the end-effector T_e^b is totally determined by the motions of $\boldsymbol{O_e}(\theta_x, \theta_y, z_\theta)$. To solve the inverse kinematic problem, one can simply apply the condition that the length of a support bar is constant.

The location of the connection between the end-effector platform and an active link is

$$\begin{aligned} \mathrm{p}_{e_i}^b &= \mathrm{R}_e \mathrm{p}_{e_i}^e + \mathrm{p}_e^b \\ &= \begin{bmatrix} l_e c\beta_i c\theta_y + z_0 s\theta_y \\ l_e c\beta_i s\theta_x s\theta_y + l_e s\beta_i c\theta_x - z_0 s\theta_x c\theta_y \\ -l_e c\beta_i c\theta_x s\theta_y + l_e s\beta_i s\theta_x + z_e + z_0 c\theta_x c\theta_y \end{bmatrix} \end{aligned} \tag{64}$$

where

$$\mathrm{p}_{e_i}^b = \begin{bmatrix} x_{e_i}^b \\ y_{e_i}^b \\ z_{e_i}^b \end{bmatrix} \quad \text{and} \quad \mathrm{p}_{e_i}^e = \begin{bmatrix} l_e c\beta_i \\ l_e s\beta_i \\ z_0 \end{bmatrix}$$

z_0 is the offset of the spherical joint with respect to O_e.

The derivative of Equation (64) with the time yields

$$\begin{bmatrix} \delta x_{e_i}^b \\ \delta y_{e_i}^b \\ \delta z_{e_i}^b \end{bmatrix} = [\mathrm{J}_i]_{3\times 3} \begin{bmatrix} \delta\theta_x \\ \delta\theta_y \\ \delta z_e \end{bmatrix} \tag{65}$$

where

$$\mathrm{J}_i = \begin{bmatrix} 0 & -l_e c\beta_i s\theta_y + z_0 c\theta_y & 0 \\ (l_e c\beta_i s\theta_y - z_0 c\theta_y)c\theta_x - l_e s\beta_i s\theta_x & (l_e c\beta_i c\theta_y + z_0 s\theta_y)s\theta_x & 0 \\ (l_e c\beta_i s\theta_y - z_0 c\theta_y)s\theta_x + l_e s\beta_i c\theta_x & -(l_e c\beta_i c\theta_y + z_0 s\theta_y)c\theta_x & 1 \end{bmatrix}$$

Because the active links have a fixed length, one has

$$|\mathrm{O}_b\mathrm{E}_i - \mathrm{O}_b\mathrm{B}_i - \mathrm{B}_i\mathrm{D}_i| = |\mathrm{D}_i\mathrm{E}_i| \quad (i = 1, 2, 3) \tag{66}$$

Equation (66) yields

$$k_{i1}^2 + k_{i2}^2 + k_{i3}^2 = l_i^2 \tag{67}$$

where

$$\begin{aligned} k_{i1} &= x_{e_i}^b - (l_b - u_i c\gamma)c\alpha_i \\ k_{i2} &= y_{e_i}^b - (l_b - u_i c\gamma)s\alpha_i \\ k_{i3} &= z_{e_i}^b - u_i s\gamma \end{aligned}$$

Assuming only torsion is in the linear actuator of each active link, the derivation of Equation (67) with respect to the time is then

$$\delta u_i = \begin{bmatrix} -\dfrac{k_{i1}}{k_{i4}} & -\dfrac{k_{i2}}{k_{i4}} & -\dfrac{k_{i3}}{k_{i4}} \end{bmatrix} \cdot \begin{bmatrix} \delta x_{e_i}^b \\ \delta y_{e_i}^b \\ \delta z_{e_i}^b \end{bmatrix} \quad (i = 1, 2, 3) \tag{68}$$

where

$$k_{i4} = k_{i1}c\gamma c\alpha_i + k_{i2}c\gamma s\alpha_i - k_{i3}s\gamma$$

Substituting Equation (65) into Equation (68), one has

$$\begin{bmatrix} \delta u_1 \\ \delta u_2 \\ \delta u_3 \end{bmatrix} = (\mathrm{J}_t)_{3\times 3} \begin{bmatrix} \delta\theta_x \\ \delta\theta_y \\ \delta z_e \end{bmatrix} = \begin{bmatrix} (\mathrm{J}_{t,1})_{1\times 3} \\ (\mathrm{J}_{t,2})_{1\times 3} \\ (\mathrm{J}_{t,3})_{1\times 3} \end{bmatrix} \begin{bmatrix} \delta\theta_x \\ \delta\theta_y \\ \delta z_e \end{bmatrix} \tag{69}$$

where

$$\mathrm{J}_{t,i} = \begin{bmatrix} -\dfrac{k_{i1}}{k_{i4}} & -\dfrac{k_{i2}}{k_{i4}} & -\dfrac{k_{i3}}{k_{i4}} \end{bmatrix} \cdot J_i$$

The active link is a two-force component; therefore, only axial deformation occurs. Differentiating Equation (67) with respect to time yields the following:

$$\delta l_i = \begin{bmatrix} \dfrac{k_{i1}}{l_i} & \dfrac{k_{i2}}{l_i} & \dfrac{k_{i3}}{l_i} \end{bmatrix} \cdot \begin{bmatrix} \delta x^b_{e_i} \\ \delta y^b_{e_i} \\ \delta z^b_{e_i} \end{bmatrix} (i = 1, 2, 3) \tag{70}$$

Substituting Equation (65) into (70)

$$\dot{\boldsymbol{\rho}} = \mathbf{Jt} \tag{71}$$

where vectors $\dot{\boldsymbol{\rho}}$ and $\mathbf{t}$ are the joint velocity and the twist of the platform defined as follows:

$$\dot{\boldsymbol{\rho}} = [\,\delta l_1 \quad \delta l_2 \quad \delta l_3\,]^T$$

$$\mathbf{t} = [\,\boldsymbol{\omega}^T \quad \dot{\mathbf{p}}^T\,]^T = [\,\delta\theta_x \quad \delta\theta_y \quad \delta z_e\,]$$

that is,

$$\begin{bmatrix} \delta l_1 \\ \delta l_2 \\ \delta l_3 \end{bmatrix} = (\mathrm{J}_a)_{3\times 3} \begin{bmatrix} \delta\theta_x \\ \delta\theta_y \\ \delta z_e \end{bmatrix} = \begin{bmatrix} (\mathrm{J}_{a,1})_{1\times 3} \\ (\mathrm{J}_{a,2})_{1\times 3} \\ (\mathrm{J}_{a,3})_{1\times 3} \end{bmatrix} \begin{bmatrix} \delta\theta_x \\ \delta\theta_y \\ \delta z_e \end{bmatrix}$$

where

$$\mathrm{J}_{a,i} = \begin{bmatrix} \dfrac{k_{i1}}{l_i} & \dfrac{k_{i2}}{l_i} & \dfrac{k_{i3}}{l_i} \end{bmatrix} \cdot \mathrm{J}_i$$

Direct Kinematics

The direct kinematics is to solve the posture of the end-effector T^b_e when the joint motion u_i ($i = 1$, 2, 3) is known. The solution of a direct kinematic problem can also be derived from Equation (65). At this moment, the motions of the end-effector $(\theta_x, \theta_y, z_\theta)$ are unknown, and the joint motion u_i ($i = 1$, 2, 3) is given. To solve direct kinematic problem, z_e and θ_y could be represented by θ_x, therefore, Equation (65) can be rewritten as

$$\left.\begin{aligned} z_e^2 + (A_i s\theta_y + B_i)z_e + (C_i c\theta_y + D_i s\theta_y + E_i) = 0 \\ (i = 1, 2, 3) \end{aligned}\right\} \tag{72}$$

where the coefficients $A_i \sim E_i$ are the functions of θ_x, they are

$$\begin{aligned} A_i &= -2l_e c\beta_i c\theta_x \\ B_i &= 2(l_e s\beta_i s\theta_x - u_i s\gamma) \\ C_i &= -2(l_b - u_i c\gamma)l_e c\alpha_i c\beta_i \\ D_i &= 2l_e c\beta_i(u_i s\gamma c\theta_x - (l_b - u_i c\gamma)s\alpha_i s\theta_x) \\ E_i &= l_e^2 + l_b^2 + u_i^2 - l_i^2 - 2(l_e s\beta_i(u_i s\gamma s\theta_x \\ &\quad + (l_b - u_i c\gamma)s\alpha_i c\theta_x) + l_b u_i c\gamma) \end{aligned}$$

From Equation (67), one has

$$c\theta_y = -\frac{Fz_e^2 + Gz_e + H}{Kz_e + L} \tag{73}$$

$$s\theta_y = -\frac{Iz_e + J}{Kz_e + L} \tag{74}$$

where the coefficients F–L are the functions of θ_x, and expressed by

$$\begin{aligned} F &= B_{12}A_{13} - B_{13}A_{12} \\ G &= E_{12}A_{13} + B_{12}D_{13} - E_{13}A_{12} - B_{13}D_{12} \\ H &= E_{12}D_{13} - E_{13}D_{12} \\ I &= C_{12}B_{13} - C_{13}B_{12} \\ J &= C_{12}E_{13} - C_{13}E_{12} \\ K &= C_{12}A_{13} - C_{13}A_{12} \\ L &= C_{12}D_{13} - C_{13}D_{12} \end{aligned}$$

and

$$\begin{aligned} &A_{ij} = A_i - A_j, B_{ij} = B_i - B_j, C_{ij} = C_i - C_j \\ &D_{ij} = D_i - D_j, E_{ij} = E_i - E_j \end{aligned}$$

Because $c^2\theta_y + s^2\theta_y = 1$, substituting Equation (68) and (69) into this expression generates

$$M_4 z_e^4 + M_3 z_e^3 + M_2 z_e^2 + M_1 z_e + M_0 = 0 \tag{75}$$

where

$$\begin{aligned} M_4 &= F^2 \\ M_3 &= 2FG \\ M_2 &= G^2 + I^2 + K^2 + 2FH \\ M_1 &= z_e^2 + 2(GH + IJ + KL) \\ M_0 &= H^2 + J^2 + L^2 \end{aligned}$$

One can observe that Equation (67) includes three independent equations, and two independent equations have been derived [Equations (68) and (69)]. Another equation can be derived by substituting Equations (68) and (69) into any one of the equations in Equation (67), for example, the equation when $i = 1$. Thus one obtains

$$N_3 z_e^3 + N_2 z_e^2 + N_1 z_e + N_0 = 0 \tag{76}$$

where

$$\begin{aligned} N_3 &= K \\ N_2 &= B_1 K + L - A_1 I - C_1 F \\ N_1 &= E_1 K + B_1 L - A_1 J - D_1 I - C_1 G \\ N_0 &= E_1 L - D_1 J - C_1 H \end{aligned}$$

If the direct kinematic problem is solvable for the given design, then Equations (70) and (71) should possess a common solution of z_e. Based on Bezout's method, the following condition should be satisfied:

$$\begin{vmatrix} M_4 & M_3 & M_2 & M_1 & M_0 & 0 & 0 \\ 0 & M_4 & M_3 & M_2 & M_1 & M_0 & 0 \\ 0 & 0 & M_4 & M_3 & M_2 & M_1 & M_0 \\ N_3 & N_2 & N_1 & N_0 & 0 & 0 & 0 \\ 0 & N_3 & N_2 & N_1 & N_0 & 0 & 0 \\ 0 & 0 & N_3 & N_2 & N_1 & N_0 & 0 \\ 0 & 0 & 0 & N_3 & N_2 & N_1 & N_0 \end{vmatrix} = 0 \tag{77}$$

Equation (72) becomes an equation about θ_x when the joint motion u_i is given. Equation (72) is converted using the standard transformation formulas

$$c\theta_x = (1 - t^2)/(1 + t^2), s\theta_x = 2t/(1 + t^2), (t = \tan(\theta_x/2))$$

It is a polynomial equation with the order of 40.

After θ_x is obtained from Equation (72), z_e and θ_y can be calculated sequentially from Equations (70) and (71), and Equations (68) and (69).

Velocity Analysis

Suppose that the velocities of the end-effector are $(\dot{\theta}_x, \dot{\theta}_y, \dot{z}_e)$. Differentiating Equation (66) with respect to time, one has the velocity of an active joint as

$$\dot{\boldsymbol{u}}_i = -\frac{\dot{k}_b u_i + \dot{k}_c}{2u_i + k_b} \cdot z_{d_i} \tag{78}$$

where

$\dot{\boldsymbol{u}}_i$ is the velocity of joint i, and

$$z_{d_i} = (c\gamma c\alpha_i, c\gamma s\alpha_i, s\gamma)^T$$

$$\begin{aligned} \dot{k}_b &= 2(c\gamma(\dot{x}^b_{e_i} c\alpha_i + \dot{y}^b_{e_i} s\alpha_i) - \dot{z}^b_{e_i} s\gamma) \\ \dot{k}_c &= 2(x^b_{e_i}\dot{x}^b_{e_i} + y^b_{e_i}\dot{y}^b_{e_i} + z^b_{e_i}\dot{z}^b_{e_i}) - 2l_b(\dot{x}^b_{e_i} c\alpha_i + \dot{y}^b_{e_i} s\alpha_i) \end{aligned}$$

$$(\dot{x}_{e_o}, \dot{y}_{e_o}, \dot{z}_{e_o})^T = (0_x, 0_y, \dot{z}_e)^T + \boldsymbol{\omega}_e \times (\boldsymbol{R}_e \cdot \boldsymbol{r}^T_{ei})$$

$$\boldsymbol{\omega}_e = (\dot{\theta}_x, \dot{\theta}_y, 0)^T$$

$$\boldsymbol{r}_{ei} = (x^b_{e_i}, y^b_{e_i}, z^b_{e_i})$$

To determine the kinematic behavior of the rigid body of a support bar, its angular velocity should be known, which can be determined from

$$\boldsymbol{v}_{e_i} = \boldsymbol{v}_{d_i} + \omega_{d_i e_i} \times \boldsymbol{r}_{d_i e_i} \tag{79}$$

Therefore

$$\boldsymbol{\omega}_{d_i e_i} = \frac{\boldsymbol{r}_{d_i e_i} \times (\boldsymbol{v}_{e_i} - \boldsymbol{v}_{d_i})}{l_i^2} \tag{80}$$

where

$\boldsymbol{\omega}_{d_i e_i}$ is the angular velocity of support bar i, and

$$\begin{aligned} \boldsymbol{v}_{e_i} &= (\dot{x}^b_{e_i}, \dot{y}^b_{e_i}, \dot{z}^b_{e_i})^T \\ \boldsymbol{v}_{d_i} &= \dot{\boldsymbol{u}}_i \\ \boldsymbol{r}_{d_i e_i} &= (x^b_{e_i} - (l_b - u_i c\gamma)c\alpha_i, y^b_{e_i} - (l_b - u_i c\gamma)s\alpha, z^b_{e_i} - u_i s\gamma)^T \end{aligned}$$

Acceleration Analysis

Assume the accelerations of the end-effector are $(\ddot{\theta}_x, \ddot{\theta}_y, \ddot{z}_e)$. Differentiating Equation (73) with respect to time, one has the acceleration of an active joint as follows:

$$\ddot{\boldsymbol{u}}_i = -\frac{2\dot{u}_i^2 + 2\dot{k}_b \dot{u}_i + \ddot{k}_b u_i + \ddot{k}_c}{2u_i + k_b} z_{d_i} \tag{81}$$

where $\ddot{\boldsymbol{u}}_i$ is the acceleration of joint i, and

$$\begin{aligned} \ddot{k}_b &= 2(c\gamma(\ddot{x}^b_{e_i} c\alpha_i + \ddot{y}^b_{e_i} s\alpha_i) - \ddot{z}^b_{e_i} s\gamma) \\ \ddot{k}_c &= 2(x^b_{e_i}\ddot{x}^b_{e_i} + y^b_{e_i}\ddot{y}^b_{e_i} + z^b_{e_i}\ddot{z}^b_{e_i} + (\dot{x}^b_{e_i})^2 + (\dot{y}^b_{e_i})^2 + (\dot{z}^b_{e_i})^2) \\ &\quad -2l_b(\ddot{x}^b_{e_i} c\alpha_i + \ddot{y}^b_{e_i} s\alpha_i) \\ (\ddot{x}_{e_o}, \ddot{y}_{e_o}, \ddot{z}_{e_o})^T &= (0, 0, \ddot{z}_e)^T + (\ddot{\theta}_x, \ddot{\theta}_y, 0)^T \times (\boldsymbol{R}_e \boldsymbol{r}_{ei}) + \boldsymbol{\omega}_e \\ &\quad \times (\boldsymbol{\omega}_e \times (\boldsymbol{R}_e \boldsymbol{r}_{ei})) \end{aligned}$$

Similar to Equation (73), to obtain the angular acceleration of a support bar, the following relationship can be applied:

$$\boldsymbol{a}_{e_i} = \boldsymbol{a}_{d_i} + \boldsymbol{\omega}_{d_i e_i} \times (\boldsymbol{\omega}_{d_i e_i} \times \boldsymbol{r}_{d_i e_i}) + \varepsilon_{d_i e_i} \times \boldsymbol{r}_{d_i e_i} \tag{82}$$

Therefore,

$$\boldsymbol{\varepsilon}_{d_ie_i} = \boldsymbol{r}_{d_ie_i} \times (\boldsymbol{a}_{e_i} - \boldsymbol{a}_{d_i} - \boldsymbol{\omega}_{d_ie_i} \times (\boldsymbol{\omega}_{d_ie_i} \times \boldsymbol{r}_{d_ie_i}))/l_i^2 \tag{83}$$

where

$\boldsymbol{\varepsilon}_{d_ie_i}$ is the angular acceleration of support bar i,
$\boldsymbol{\omega}_{d_ie_i}$ is calculated from Equation (75)

$$\boldsymbol{a}_{d_i} = \ddot{\boldsymbol{u}}_i \boldsymbol{z}_d$$

CONCLUSIONS

In this article, the kinematics of robot manipulators is introduced, which includes the conceptions of reference coordinate frame, translational transformation, rotational transformation, and homogeneous transformation as well as the preliminary knowledges in robot kinematics, such as Euler angle, Denavit-Hartenberg representation and Jacobian matrix of robot. An example is given. Both direct and inverse kinematics are conducted.

BIBLIOGRAPHY

1. D. Steward, A platform with six degrees of freedom, *Proc. Instn. Mech. Engrs.*, **180**(5): 371–386, 1965.
2. J. A. Carretero, R. P. Podhorodeski, M. N. Nahon, and C. M. Gosselin, Kinematic analysis and optimization of a new three degree-of-freedom spatial parallel manipulator, *ASME J. Mechanical Design*, **122**: 17–24, 2000.
3. G. R. Dunlop and T. P. Jones, Position analysis of a two DOF parallel mechanism – Canterbury tracker, *Mech. Mach. Theory*, **34**: 599–614, 1999.
4. K.-M. Lee and S. Arjunan, A three-degrees-of-freedom micromotion in-parallel actuated manipulator, *IEEE Trans. Robot. Automat.*, **7**(5): 634–641, 1991.
5. D. Fedewa, M. G. Mehrabi, S. Kota, N. Orlandea, and V. Gopalakrishran, Parallel structures and their applications in reconfigurable machining systems, *Proc. of the 2000 Parallel Kinemati Machines*-International Conference, Ann-Arbor, MI, 2000, pp. 87–97.
6. C. C. Nguyen, Z.-L. Zhou, and M. Bryfogis, A Robotically assisted munition loading system, *J. Robotic Sys.*, **12**(12): 871–881, 1995.
7. J. A. Soons, On the geometric and thermal errors of a Hexapod machine tools, In: C. R. Molinari-Tosatti and K. S. Smith, (eds.), *Parallel Kinematic Machines: Theoretical Aspects and Industrial Requirements*, Advanced Manufacturing Series, London: Springer-Verlag, 1999, pp. 151–170.
8. K. H. Wurst, LINAPOD-Machine tools as parallel link systems based on a modular design, In: C. R. Molinari-Tosatti and K. S. Smith, (eds.), *Parallel Kinematic Machines: Theoretical Aspects and Industrial Requirements*, Advanced Manufacturing Series, London: Springer-Verlag, 1999, pp. 377–394.
9. Y. Koren, Will industry adopt PKMs? *Manufact. Engineer.*, 1999, pp. 240.
10. Z. X. Cai, *Robotics,* Beijing: Press of Tsinghua University, 2000.

Dan Zhang
Zhuming Bi
University of Ontario Institute
of Technology
Oshawa, Ontario, Canada

Zhen Gao
University of Science and
Technology of China
Hefei, China

ROBOT MOTION PLANNING

INTRODUCTION

The aim in robot motion planning is to be able to specify a task in a high-level, expressive language and have the robot(s) automatically convert the specification into a set of low-level primitives, such as feedback controllers and communication protocols, to accomplish the task (1,2). The robots can vary from manipulator arms used in manufacturing or surgery, to autonomous vehicles used in search and rescue or in planetary exploration, and to smart wheelchairs for disabled people. They are subject to mechanical constraints (e.g., a car-like robot cannot move sideways and an airplane cannot stop in place) and have limited computation, sensing, and communication capabilities. The environments can be cluttered with possibly moving and shape-changing obstacles and can contain dynamic (moving, appearing, or disappearing) targets. The challenge in this area is the development of a computationally efficient framework accommodating both the robot constraints and the complexity of the environment, while allowing for a large spectrum of task specifications.

A robot combines moving mechanical pieces such as wheels, gears, and breaks with digital devices such as processors and sensing and communication devices, in continuous interaction with a possibly changing environment. Therefore, motion planning is a highly interdisciplinary area, combining tools from computer science, mechanics, control theory, and differential geometry. Given the variety of applications, many motion planning approaches have been developed over the years. Depending on the task they address, motion planning problems can roughly be divided into four main groups: *navigation, coverage, mapping, and localization*(3).

In *navigation,* the problem is to find a collision-free motion between two configurations of the robot. *Coverage* is the problem of moving a robot sensor or end effector in such as way that it reaches all points in a target space (e.g., painting a surface). *Mapping* involves exploring an environment with the goal of producing a representation that can be used, for example, in navigation and coverage. Finally, in *localization,* the problem is to use a map and sensor data to determine the configuration (state) of the robot. Localization and mapping are sometimes performed simultaneously, such as in simultaneous localization and mapping (SLAM)(4).

Motion planners also differ depending on the robot model they consider. For example, it is much easier to plan the motion of a robot that is free to move instantaneously in all directions of its configuration space (omnidirectional robot), rather than generating motion for a car-like or an airplane-like vehicle that cannot move sideways (i.e., nonholonomic robot; see Ref. 5 for a collection of motion planning approaches for such robots). Motion planning can be performed for kinematic robot models, which capture only the configuration and velocity of the robot, or for dynamic robot models, which capture forces and accelerations.

Motion planning approaches can also be classified depending on the properties of the underlying algorithms. A motion plan is optimal if the produced motion minimizes energy consumption, execution time, trajectory length, and so on. Computational complexity is also a determining factor. For example, in most cases, it is desired that the amount of necessary memory and running time scale polynomially with the size of the input of the planner, which can be the number of obstacles, the number of degrees of freedom of the robot, and so on. Finally, a planner is complete if it always finds a path if one exists. Others are resolution complete, if a solution is found whenever one exists at a given discretization resolution, or probabilistic complete, if the probability of finding a solution during an iterative discretization process converges to 1 when the solution exists.

WORKSPACE AND CONFIGURATION SPACE

Given a robotic system, a *configuration* is a complete description that determines the position of every point of the system uniquely. Its *configuration space,* called for simplicity C-space, is the set of all possible configurations of the system. The number of degrees of freedom of a robotic system is the dimension of its minimal configuration space or, in other words, the minimum number of parameters needed to describe the system completely. The space in which the robotic system does work is called the *workspace*, which can be seen as the Euclidean space $\mathbb{R}^2$ or $\mathbb{R}^3$, depending on whether the motion is in plane or space. Most often, however, the workspace is defined more precisely as the subset of $\mathbb{R}^2$ or $\mathbb{R}^3$ that can be reached by a point of interest on the robot, such as the end effector of a manipulator arm.

Consider, for example, a two-joint planar robot arm, where a point on the first link is pinned to the ground, and the base of the second link is pinned to the end of the first, such that the only possible motion of the second link is a rotation about the (revolute) joint [Fig. 1(a)]. If we denote by θ_1 and θ_2 the angles formed by the two links with the horizontal, then (θ_1, θ_2) can be seen as coordinates of the configuration space, which is $S^1 \times S^1 = T^2$, where S^1 and T^2 denote the unit circle and the torus, respectively [Fig. 1(c)]. The workspace of this robot, however, is an annulus, with the outer radius determined by the sum of the lengths of the two links, and the inner radius is given by the difference between their lengths [Fig. 1(b)].

The configuration space of a planar square robot (or any other rigid body) that can only translate without rotation (see Fig. 2) is $\mathbb{R}^2$. For a planar robot that can only rotate about a fixed point, the configuration space is $SO\,(2)$, called the Special Orthogonal group in the plane, which is

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

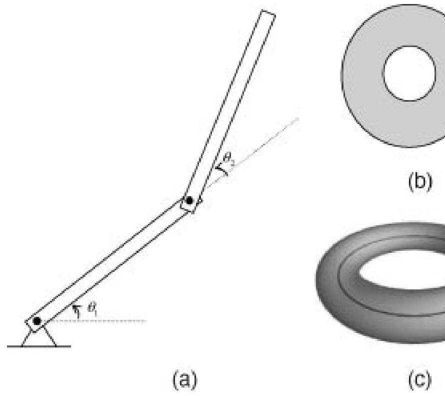


Figure 1. A two-link manipulator (a), its workspace (b), and its configuration space (c).

isomorphic to S^1. The configuration space of a robot allowed to translate and rotate in the plane is $SE(2)$, called the Special Euclidean group in the plane, and defined as $SE(2) = \mathbb{R}^2 \times SO(2)$. In space, a rotating and translating robot modeled as a rigid body evolves in $SE(3) = \mathbb{R}^3 \times SO(3)$, where $SO(3)$ is the group of rotations in space (3).

If obstacles are present in the workspace, it is useful to define explicitly the set of robot configurations for which collisions occur. For each obstacle in the workspace, the *configuration space obstacle* is defined as the set of all configurations at which the robot intersects the obstacle in the workspace. The free configuration space, also called free C-space, is the set of configurations at which the robot does not intersect any obstacle. Figure 2(b) shows the free C-space for the square robot moving in the environment shown in Fig. 2(a), if only translational motion is allowed (no rotation). The reader is referred to Ref. 3, p. 509 for an example of the free C-space construction for a polytopal robot translating and rotating in a polytopal environment with polytopal obstacles. In this setup, a navigation problem, as defined above, translates to finding a continuous curve between the initial and the final configuration in the free C-space.

RIGID BODY MOTION INTERPOLATION

A navigation problem for a robot, represented simply as a rigid body moving in space or plane with no obstacles, is also called rigid body motion interpolation. In the configuration space of the robot, which is $SE(3)$ or $SE(2)$ as defined above, this problem translates to generating a (possibly smooth) curve interpolating between two points. The problem of

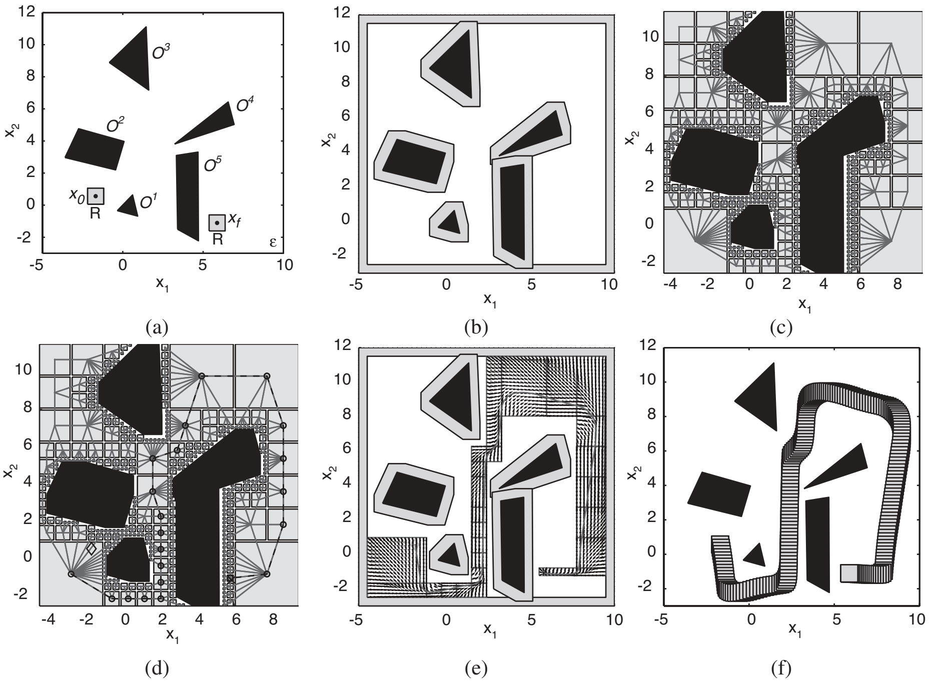


Figure 2. Cell decomposition and simultaneous planning and control for a square robot translating (no rotation) in a 2-D rectangular environment with polyhedral obstacles. The observable is the centroid of the robot: (a) initial (left) and final (right) positions of the robot. (b) The free C-space is obtained by enlarging the obstacles, shrinking the environment boundaries, and reducing the robot to its observable point. (c) Rectangular (quadtree) partition of the free C-space and the quotient graph of the partition. (d) Optimal path from initial to final node (rectangle) in the quotient graph; ◇ and × denote the initial and final position of the robot observable, respectively. (e) Vector field assignment (used in simultaneous planning and control) and the resulting trajectory of the observable. (f) Robot motion in the initial environment.

finding a smooth interpolating curve is well understood in Euclidean spaces (e.g., a line segment is a smooth interpolating curve between two points), but it is not easy to generalize such techniques to curved spaces, such as $SE(3)$ and $SE(2)$. Most work in this area proposes to generalize the notion of interpolation from the Euclidean space to a curved space. For example, in Ref. 6, Bezier curves are used for interpolating rotations based on a spherical analog of the well-known de Casteljau algorithm. Other examples include spatial rational B-splines and Hermite interpolation (see Ref. 7 for an overview).

The above methods find immediate applications in computer graphics (e.g., generate a “natural” motion for an object thrown from one place to another). However, to generate a robot motion plan, two more issues have to be taken into consideration: optimality of the trajectory and invariance with respect to the choice of a world frame. The optimality requirement is particularly relevant in applications such as deep space formations. For example, to achieve interferometry, a group of satellites is required to maintain a rigid body formation. A reconfiguration demands a fuel-optimal trajectory to preserve mission life and is constrained by the limited thrust available.

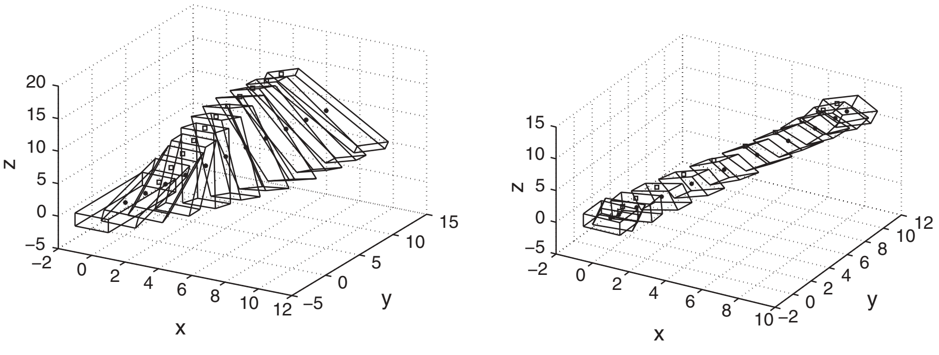
Coordinate-free approaches leading to trajectories that are invariant to reference frames exist for the generation of shortest paths and minimum acceleration trajectories on $SO(3)$ (the set of all rotations in $\mathbb{R}^3$) and $SE(3)$ (the set of all poses in $\mathbb{R}^3$) (see Ref. 8 for an overview) (see Fig. 3 for examples). However, analytical solutions are available only in the simplest cases, and the procedure for solving optimal motions, in general, is computationally intensive. A relaxation based on the generation of optimal curves in an embedding Euclidean space and subsequent projection, which leads to suboptimal solutions, is proposed in Ref. 7.

POTENTIAL-BASED PLANNERS

Potential-based motion planners are based on the idea that a robot configuration can be driven to a desired value in the same way in which a particle moves in a force field. More precisely, a potential ϕ is a differentiable, real-valued function defined on the configuration space. Its gradient $\nabla\phi$ is a vector that points in the direction of maximum (local) increase of ϕ. This gradient can be used to define a vector field, (i.e., an assignment of a vector to each point of the configuration space). Guiding a robot through an environment with obstacles toward a goal can, therefore, be achieved by constructing a potential function in the configuration space, with a minimum value at the goal and high values at the obstacles, and by setting the velocity of the robot configuration equal to the negative of the gradient of the potential. Such an approach can be used directly to accommodate kinematic robot models. However, this approach can be extended for dynamic models, by designing control laws guaranteeing the convergence of the velocities to a desired vector field (9).

The robot motion terminates at a point of zero velocity or, equivalently, at a point where the gradient of the potential function vanishes, which is a critical point for the potential function. This point can be, in general, a minimum, a maximum, or a saddle point, and it can be degenerate or nondegenerate (isolated), depending on the Hessian matrix of the potential function. Explicitly, a critical point is degenerate if and only if the Hessian i.e., the matrix of second derivatives; see Ref. 3 is singular at that point. A positive-definite Hessian indicates a local minimum, a negative-definite Hessian indicates a local maximum, and a Hessian of indefinite sign indicates a saddle point. The goal in potential-based motion planning is to have the robot stop at the minimum corresponding to the goal. Although the robot can, in theory, stop at a local maximum or at a saddle point, this is impractical, because such points are unstable, and the probability that this happens is basically zero. Other possible local minima, on the other hand, are attractive, and the robot can get “caught” into such undesirable points in its way to the goal. Most of the existing potential-based planners, where the potential function is constructed through the superposition of attractive (to the goal) functions and repulsive (from the obstacles) functions, suffer from this problem.

To address the local minima problem, two types of approaches have been developed (see Ref. 3 for a detailed overview). In the first approach, the potential field is augmented with a search-based planner. For example, the randomized path planner (RPP)(3) uses a variety of potential



(a)

(b)

Figure 3. A geodesic (minimum length, or energy) curve for a cuboid and a minimum acceleration curve for a cube.

functions, and when stuck at a local minimum, it performs a random walk, with the goal of escaping the local minimum. In the second approach, a special type of potential function, called a navigation function, is constructed. Although guaranteed to have exactly one minimum, a navigation function can only be applied to a limited class of configuration spaces, which are diffeomorphic to sphere spaces.

ROADMAPS

If several navigation tasks are expected to occur in an environment, then building a map of the environment and then performing navigation using the map can prove to be more efficient than performing navigation from scratch every time such a request occurs. The most used of such maps are topological, or graph-like, structures, where nodes correspond to "interesting features" and the edges show adjacency between nodes. For example, the nodes can be points of interest for a specific task such as targets or intersections, whereas the edges can label actions required from the robot to move from a location to another.

Roadmaps (3) are topological maps embedded in the free space. In other words, in a roadmap, the nodes correspond to physical locations in the environment, and the edges correspond to paths between different locations. A roadmap is, therefore, both a graph and a collection of one-dimensional manifolds (curves). Robots use roadmaps in the same way drivers use the interstates. Instead of planning a trip from point A to point B on small streets, a driver would plan her trip from A to a close interstate, then on the interstate for as long as possible, and then from the interstate to the destination B. Similarly, if a roadmap is available, a robot planner would find a collision-free path to the roadmap, then travel on the roadmap until close to the destination, and then find another collision-free path from the exit point on the roadmap to the destination. Most motion occurs on the roadmap, which is low dimensional, as opposed to the motion to and from the roadmap, which occurs in a possibly high-dimensional configuration space.

Several types of roadmaps have been developed over the years, which include visibility maps, deformation retracts, and silhouettes. In visibility maps, which work for polygonal environments with polygonal obstacles, the nodes are the vertices of the polygons, and an edge between two nodes means that a line of sight exists between the nodes. Deformation retracts capture the "essential" topology of an environment, and they include generalized Voronoi diagrams(3). Finally, silhouette methods are based on repeated projection of the robot-free configuration space onto lower dimensional spaces until a one dimensional representation is reached.

SAMPLING-BASED ALGORITHMS

The construction of roadmaps, as presented above, is based on an explicit representation of the free C-space. As a result, as the dimension of the configuration space increases (e.g., a manipulator arm with several joints and a gripper with fingers can have tens of degrees of freedom), motion planners based on roadmaps become computationally infeasible. In such cases, sampling-based approaches are more appropriate. In short, a sampling-based planner generates samples (i.e., collision-free configurations of the robot) and then interpolating paths for the samples. The latter process is also often achieved through sampling but at a finer rate.

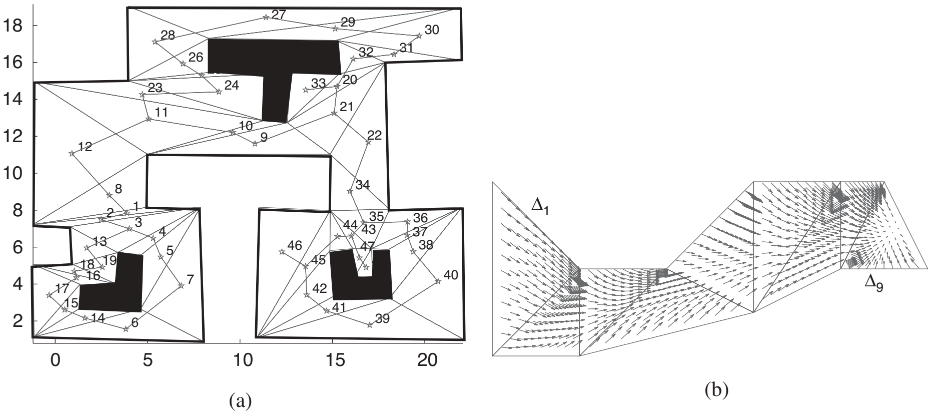
The most representative sampling-based algorithm is the Probabilistic Road Map Planner (PRM) (3). The main idea behind PRM is that it is easy (and cheap) to check for collisions with obstacles. In other words, it is easy to see whether a sample is in the free C-space. PRM uses coarse sampling to obtain the nodes of the roadmap and fine sampling to construct its edges. Once the roadmap is constructed, it is used in exactly the same way as the "classic" roadmap presented in the previous section. The existing PRMs differ by the way samples and interpolating paths are generated. The basic PRM uses uniform distribution for node sampling. Other, more sophisticated, PRMs use sampling schemes such as importance sampling in areas that are difficult to explore and deterministic sampling such as sampling on grids and quasirandom sampling.

As PRM is a roadmap planner, it is optimized for several queries. For single queries, other sampling-based algorithms are effective, such as the rapidly exploring random tree planner (RRT)(3). A combination of multiple-query and single-query methods, such as the sampling-based roadmap of trees (SRT)(3), tries to find a compromise between using a roadmap versus a large sampling tree in very difficult planning problems. Other developments in this area led to sampling-based planners that take into account kinematic and dynamic constraints, stability requirements, energy constraints, visibility constraints, and contact constraints. Sampling-based algorithms opened a new direction in robot motion planning, by making it possible to approach very high-dimensional robotic systems.

CELL DECOMPOSITIONS

Cell decompositions are among the most used techniques for robot motion planning. To illustrate the main ideas, we assume for simplicity that the motion task is a navigation task. The computation behind most of the existing approaches consists of three main steps(3). In the first step, the free configuration space is partitioned, and the quotient graph of the partition is constructed [see Fig. 2(c)]. In this graph, a node labels a free cell, and an edge between two nodes shows an adjacency relation between the corresponding cells. In the second step, the cells corresponding to the initial and the final configurations are determined, and a path is determined between the corresponding nodes in the quotient graph [see Fig. 2(d)]. This path can be optimal with respect to some cost, which in the simplest cases penalizes the distance traveled by the robot. Alternatively, the cost can prevent from generating paths "too close" to the obstacles. Finally, in the third step, a desired robot trajectory is constructed inside the configuration-space tube determined by the path in the quotient graph, and a trajectory-following controller is synthesized for the robot.

The several cell-decomposition methods can be classified according to the underlying partition scheme. The most popular cell decompositions are trapezoidal decomposi-



(a)

(b)

Figure 4. (a) A triangulation of the free space in a polygonal environment and the corresponding quotient graph. (b) A sequence of triangles (such as resulting from a path on the quotient graph of a triangulation) is executed by constructing affine vector fields in each triangle.

tions, triangulations, and rectangular grids. For example, Fig. 2(c) shows a rectangular partition, whereas Fig. 4(a) shows a triangulation of a polygonal environment cluttered with polygonal obstacles. Note that although efficient algorithms exist for planar trapezoidal partitions and triangulations, these procedures become cumbersome in higher dimensional spaces. Rectangular partitions, although not particularly efficient in plane, have the advantage of working in higher dimensions, especially when a 2^n-trees (i.e., quad-trees in plane and oct-trees in space) are used during the partition process.

The three-step, top-down process presented above has two main disadvantages. First, because no robot control and/or actuation constraints are taken into account during the decomposition, it is possible that the reference trajectory generated in the last step cannot be followed by the robot. To deal with this issue, in recent years, researchers proposed approaches for the so-called *simultaneous planning and control*. In these studies, the environment partitioning is performed at the same time with the assignment of robot-compatible controllers for each region. For example, polygonal partitions of planar environments followed by assignment of vector fields obtained as solutions of Laplaces equation in each of the regions were considered in Ref. 10. Triangular partitions and rectangular partitions can be also accompanied by the assignment of vector fields with arbitrary polyhedral bounds, if the robot dynamics are restricted to affine and multi-affine (see, for example, Ref. 11). In Figs. 2(e) and 4(b), we show how vector fields are assigned in each rectangle and triangle from a path in the quotient graph. In this setup, the "execution" of a "discrete" path is provably correct (i.e., regardless of the actual position of the robot inside each region), therefore avoiding the trajectory generation and following process.

Trapezoidal, triangular, and rectangular decompositions, as presented above, are mostly used for navigation tasks, and they are not appropriate for coverage(3). Indeed, even if coverage of cells can be efficiently achieved through coverage algorithms on graphs, covering the space inside each cell might be difficult because of the size and shape of the resulting cells. If coverage is the task at hand, then Boustrophedon decompositions and Morse Cell decompositions are more appropriate(3). Roughly put, Boustrophedon decompositions start from a trapezoidal decomposition and reorganize cells such that shorter and more efficient paths can cover the same area. Morse decompositions are based on the same idea, but they allow us to achieve coverage in non polygonal environments. Finally, for a special class of motion planning problems, called pursuit/evasion problems (games), a visibility-based cell decomposition is more appropriate. Roughly, moving from one cell to an adjacent one in this decomposition corresponds to a change in visibility (i.e., target or obstacles appear or disappear).

SYMBOLIC APPROACHES TO MOTION PLANNING AND CONTROL

The current approaches to robot motion planning and control presented above have two main limitations. First, the specification language is not rich enough for a large spectrum or robotic applications. For example, a navigation task is always formulated as "go from A to B and avoid obstacles." Nevertheless, the accomplishment of a mission might require the attainment of either A or B, convergence to a region ("reach A eventually and stay there for all future times"), visiting targets sequentially ("reach A, and then B, and then C"), surveillance ("reach A and then B infinitely often"), and so on. Second, as mentioned, some of the approaches above, such as cell decomposition, do not explicitly take into account the control, sensing, and communication constraints of the robot.

Symbolic approaches to robot motion planning and control have been developed recently to address these limitations. They draw on well-established concepts in related areas, such as behavior-based robotics and hybrid control systems. As the specification language is enriched and real-world robot control, sensing, and communication constraints are taken into account, concepts and tools from the theory of computation such as automata and languages develop naturally, hence the name "symbolic" (see Ref. 12

for a more detailed overview of these methods and the main challenges in the area).

To introduce the main ideas, note that the typical cell-decomposition approach to the navigation problem is a hierarchical, three-level process. At the first (top-most) level, the specification "go from A to B and avoid obstacles" is given, the obstacle-free configuration space of the robot is partitioned into cells, and the quotient graph is constructed (see Figs. 2(c) and 4(a) for examples). As any path connecting the cell containing A to the cell containing B in this graph satisfies the specification (i.e., it avoids obstacles), this is called the specification level. In the second step, a path on this graph is chosen, which can be seen as a "discrete" execution of the robot, hence, the name *execution level* for this step. Finally, in the third step, called the *implementation level*, a reference trajectory traversing the sequence of cells given by the path is generated, and robot controllers are constructed so that the reference trajectory is followed.

Symbolic approaches to motion planning fit into the three-level hierarchy described above, and they can be divided into two main groups: top-down approaches and bottom-up approaches. In top-down approaches (also referred to as middle-out approaches(13)), the focus is on the expressivity of the specification language, and the hope is that, while going down the three-level hierarchy presented above, implementations are possible for real-world robots. In bottom-up approaches, the starting point is a careful analysis of the control and sensing communication of the robot, possible executions are generated at the execution level, and the hope is that the set of such robot-compatible executions give rise to an expressive specification language. However, a significant gap exists between these two approaches. Bridging in this gap is one of the main challenges in the area(12).

Top-Down Symbolic Approaches

It was recently suggested that, to define a rich specification language for robot motion, inspiration can be taken from temporal logics, which are commonly used for specifying and verifying the correctness of digital circuits and computer programs. Roughly, any rich, human-like, temporal, and logic statement about the reachability of regions of interest by the robot (including the ones given as examples above) translate naturally to a formula in such a logic. Consider, for example, that a robot moves in an environment with three obstacles o_1, o_2, and o_3 and three targets r_1, r_2, and r_3 that need to be surveyed (visited infinitely many times). In other words, the task can be formulated as "*Always avoid obstacles o_1, o_2, o_3 and visit regions r_1, r_2, r_3, in this order, infinitely often.*" This specification immediately translates to the following formula of linear temporal logic (LTL) over the set of symbols o_1, o_2, o_3, r_1, r_2, r_3: $\mathbf{G}(\mathbf{F}(r_1 \wedge \mathbf{F}(r_2 \wedge \mathbf{F}r_3)) \wedge \neg(o_1 \vee o_2 \vee o_3))$, where $\neg$ and $\wedge$ stand for Boolean negation and disjunction and $\mathbf{G}$ and $\mathbf{F}$ are temporal operators that mean "always" and "eventually," respectively.

The semantics of LTL formulas are given over labeled transition graphs (also called Kripke structures or transition systems). Such a transition system can be obtained from the dual graph of the partition induced by the regions of interest, if the nodes are labeled according to their being part of obstacles or of targets, and if the edges are viewed as transitions that a robot can take. To compute a transition between two nodes (or a self-transition), one could proceed by checking for the existence of robot feedback controllers taking the robot from one region to another in finite time (or keeping the robot inside the region forever), regardless of the initial position of the robot. If this is achieved, then a certain type of equivalence relation exists between the initial control system describing the motion of the robot in the environment and the finite transition system, called bisimulation, which guarantees that the two systems satisfy the same LTL formula. Therefore, provided that the two types of controllers can be constructed, the motion planning problem is reduced to a classic model checking procedure, for which exist several off-the-shelf tools developed by the formal verification community(14).

Currently, two classes of systems are available for which such quotients can be efficiently constructed: affine systems with polyhedral partitions, and multi-affine systems (i.e., polynomial systems where the maximum power at which a variable can occur is one) with rectangular partitions. Although these two classes of systems seem restrictive for robot dynamics, it is important to note that multi-affine dynamics capture vector cross products, and they can therefore accommodate dynamics of aircraft with gas-jet actuators and underwater vehicles. In addition, differential- drive and car-like vehicles can be easily accommodated by solving an additional input–output feedback linearization. Fully automatic computational frameworks for control of affine and multi-affine dynamics from rich specifications given as arbitrary LTL formulas over linear and rectangular predicates were developed in Ref. 15 and 16. A related approach was used in Ref. 17 to control a nonholonomic robot model. In Ref. 18, it is shown that a significant decrease in computation can be achieved if the specifications are restricted to a fragment of LTL.

Bottom-Up Symbolic Approaches

The top-down symbolic approaches presented above use environment discretization to capture the complexity of the environment. While allowing for a rich specification language over the partition regions, they are (in current form) restricted to static, *a priori* known environments and simple robot dynamics, such as fully actuated or affine dynamics with polyhedral speed constraints. As suggested, robots with more complex dynamics such as helicopter-like vehicles might not be able to implement executions strings over partition regions. In this situation, the discretization may be more appropriate at the level of controllers rather than environments. The argument behind such a control-driven discretization is that the global control task can be broken down into more easily defined behavioral building blocks, each defined with respect to a particular subtask, sensing modality, or operating point. Strings over such behaviors make up words in so-called motion description languages (MDLs)(19). An example of such a string is $(k_{i_1}, \xi_{i_1}), \ldots, (k_{i_q}, \xi_{i_q})$, where $k_{i_j} : \mathcal{R}^+ \times X \rightarrow U$ are feedback control laws and $\xi_{i_j} : \mathcal{R}^+ \times X \rightarrow \{0, 1\}$ are temporal or environmentally driven interrupt conditions, $j = 1, \ldots, q$. The robot "parses" such words as $\dot{x} = f(x, k_{i_1}(t,x))$ until $\xi_{i_1}(t,x) = 1$, at which point the timer t is reset to 0, and $\dot{x} = f(x, k_{i_2}(t,x))$ until $\xi_{i_2}(t,x) = 1$, and so on.

An attractive alternative to MDL is to use motion primitives. The idea is that, instead of using controllers chosen from a collection of controls, one could think of simplifying a robot control problem by piecing together, in an appropriate way, a set of elementary trajectories chosen from a small "library"—that are themselves guaranteed to satisfy the constraints. Such feasible trajectories that can be combined sequentially to produce more complicated trajectories are called "motion primitives"(20). The compatibility rules between such primitives can be, as above, modeled as finite-state machines, called Maneuver Automata. Motion primitives can be generated in several ways, for example, by recording the actions of a human pilot; if an accurate model of the robot' s dynamics is available, model-based approaches are also possible (e.g., to design optimal maneuvers).

Although the symbolic approaches to motion planning described in this section have been applied successfully to challenging problems in autonomous mobile robotics, including acrobatic aircraft, and off-road races, several challenges still need to be addressed. For example, the problem of choosing the control modes (quanta) or motion primitives for achieving a given task is not at all obvious. One way of addressing it is by letting the control mode selection be driven by experimental data. For instance, one can envision a scenario in which a human operator is controlling a mobile platform and then, through an analysis of the input–output sample paths, construct motion description languages that reproduce the human-driven robot behavior.

BIBLIOGRAPHY

1. J. C. Latombe, *Robot Motion Planning*, Boston, MA: Kluger Academic Publishers., 1991.
2. S. M. LaValle, *Planning Algorithms*, Cambridge, UK: Cambridge University Press, 2006.
3. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, Boston, MA: MIT Press, 2005.
4. S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: The MIT Press, 2005.
5. Z. Li and J. F. Canny, (eds.), *Nonholonomic Motion Planning*, Norwell, MA: Kluwer Academic Publishers, 1992.
6. K. Shoemake, Animating rotation with quaternion curves, *ACM Siggraph*, **19** (3): 245–254, 1985.
7. C. Belta, Geometric methods for multi-robot motion planning and control, PhD thesis, Philadelphia, PA, University of Pennsylvania, 2003.
8. M. Žefran, V. Kumar, and C. Croke, On the generation of smooth three-dimensional rigid body motions, *IEEE Trans. Robotics Auto.*, **14** (4): 579–589, 1995.
9. E. Rimon and D. E. Koditschek, Exact robot navigation using artificial potential functions, *IEEE Trans. Robotics Auto.*, **8** (5): 501–518.
10. D. C. Conner, A. A. Rizzi, and H. Choset, Composition of local potential functions for global robot control and navigation, *Proc. of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, 2003.
11. C. Belta, V. Isler, and G. J. Pappas, Discrete abstractions for robot planning and control in polygonal environments, *IEEE Trans. Robotics*, **21** (5): 864–874, 2005.
12. C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, Symbolic planning and control of robot motion, *IEEE Robotics Auto. Mag.*, **14** (1): 61–71, 2007.
13. M. S. Branicky, T. A. Johansen, I. Petersen, and E. Frazzoli, On-line techniques for behavioral programming, *Proc. of the IEEE Conference on Decision and Control*, Sydney, Australia, 2000.
14. E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA: The MIT Press, 2000.
15. L. C. G. J. M. Habets, P. J. Collins, and J. H. van Schuppen, Reachability and control synthesis for piecewise-affine hybrid systems on simplices, *IEEE Trans. Aut. Control*, **51**: 938–948, 2006.
16. M. Kloetzer and C. Belta, A fully automated framework for control of linear systems from temporal logic specifications, *IEEE Trans. Auto. Cont.*, **53** (1): 287–297, 2008.
17. D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas, Valet parking without a valet, *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, 2007.
18. G. Fainekos, S. Loizou, and G. J. Pappas, Translating temporal logic to controller specifications, *Proc. 45th IEEE Conference on Decision and Control*, San Diego, CA, 2006.
19. M. Egerstedt and R. W. Brockett, Feedback can reduce the specification complexity of motor programs, *IEEE Trans. Auto. Cont.*, **48** (2): 213–223, 2003.
20. E. Frazzoli, M. A. Dahleh, and E. Feron, Maneuver-based motion planning for nonlinear systems with symmetries, *IEEE Trans. Robotics*, **21** (6): 1077–1091, 2005.

Calin Belta
Boston University
Brookline, Massachusetts

TRANSACTION PROCESSING

A business transaction is an interaction in the real world, usually between an enterprise and a person, in which something, such as money, products, or information, is exchanged (1). It is often called a computer-based transaction, or simply a transaction, when some or the whole of the work is done by computers. Similar to the traditional computer programs, a transaction program includes functions of input and output and routines for performing requested work. A transaction can be issued interactively by users through a Structured Query Language (SQL) or some sort of forms. A transaction can also be embedded in the application program written in a high-level language such as C, Pascal, or COBOL.

A transaction processing (TP) system is a computer system that processes the transaction programs. A collection of such transaction programs designed to perform the functions necessary to automate given business activities is often called an application program (application software). Figure 1 shows a transaction processing system. The transaction programs are submitted to clients, and the requests will be scheduled by the transaction processing monitor and then processed by the servers. A TP monitor is a piece of software that connects multiple clients to multiple servers to access multiple data resources (databases) in TP systems. One objective of the TP monitor is to optimize the use of system and network resources when clients and servers execute on different processors.

TP is closely associated with database systems. In fact, most earlier TP systems, such as banking and airlines reservation systems, are database systems, in which data resources are organized into databases and TP is supported by database management systems (DBMSs). In traditional database systems, transactions are usually simple and independent, and they are characterized as short duration in that they will be finished within minutes (probably seconds). Traditional transaction systems have some limitations for many advanced applications such as cooperative work, in which transactions need to cooperate with each other. For example, in cooperative environments, several designers might work on the same project. Each designer starts up a cooperative transaction. Those cooperative transactions jointly form a transaction group. Cooperative transactions in the same transaction group may read or update each other's uncommitted (unfinished) data. Therefore, cooperative transactions may be interdependent. Currently, some research work on advanced TP has been conducted in several related areas such as computer-supported cooperative work (CSCW) and groupware, workflow, and advanced transaction models (2–6). In this article, we will first discuss traditional transaction concepts and then examine some advanced transaction models.

Because of recent developments in laptop or notebook computers and low-cost wireless digital communication, mobile computing began to emerge in many applications. As wireless computing leads to situations where machines and data no longer have fixed locations in the network, distributed transactions will be difficult to coordinate, and data consistency will be difficult to maintain. In this article, we will also briefly discuss the problems and possible solutions in mobile transaction processing.

This paper is organized as follows. First, we will introduce traditional database TP, including concurrency control and recovery in centralized database TP. The next section covers the topics on distributed TP. Then, we discuss advanced TP and define an advanced transaction model and a correctness criterion. Mobile TP is also presented. Finally, future research directions are included.

DATABASE TRANSACTION PROCESSING

As database systems are the earlier form of TP systems, we will start with database TP.

Databases Transactions

A database system refers to a database and the access facilities (DBMS) to the database. One important job of DBMSs is to control and coordinate the execution of concurrent database transactions.

A database is a collection of related data items that satisfy a set of integrity constraints. The database should reflect the relevant state as a snapshot of the part of the real world it models. It is natural to assume that the states of the database are constrained to represent the legal (permissible) states of the world. The set of *intintegrity constraints* such as functional dependencies, referential integrity, inclusion, exclusion constraints, and some other user-defined constraints are identified in the process of information analysis of the application domain. These constraints represent real-world conditions or restrictions (7). For example, functional dependencies specify some constraints between two sets of attributes in a relation schema, whereas referential integrity constraints specify constraints between two sets of attributes from different relations. For detailed definitions and discussions on various constraints, we refer readers to Refs. 7 and 8. Here, we illustrate only a few constraints with a simple example.

Suppose that a relational database schema has the following two table structures for Employee and Department with attributes like Name and SSN:

Employee (Name, SSN, Bdate, Address, Dnumber)
Department (Dname, Dnumber, Dlocation).

Name = employee name
SSN = social security number
Bdate = birth date
Address = living address
Dnumber = department number

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

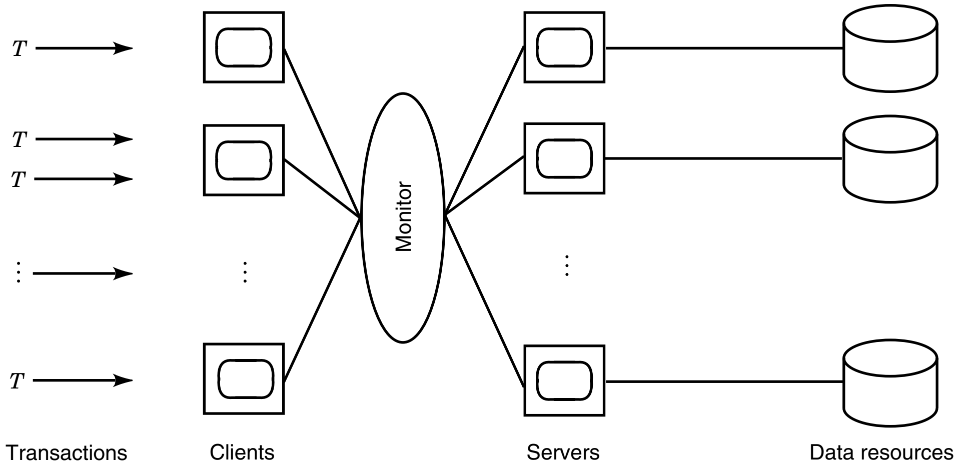


Figure 1. TP monitor between clients Data resources and data resources.

Dname = department name

Dlocation = department location

Each employee has a unique social security number (SSN) that can be used to identify the employee. For each SSN value in the Employee table, there will be only one associated value for Bdate, Address, and Dnumber in the table, respectively. In this case, there are functional dependencies from SSN to Bdate, Address, Dnumber. If any Dnumber value in the Employee relation has the same Dnumber value in the Department relation, there will be a referential integrity constraint from *Employee*'s Dnumber to *Department*'s Dnumber.

A database is said to be "consistent" if it satisfies a set of integrity constraints. It is assumed that the initial state of the database is consistent. As an empty database always satisfies all constraints, often it is assumed that the initial state is an empty database. It is obvious that a database system is not responsible for possible discrepancies between a state of the real world and the corresponding state of the database if the existing constraints were inadequately identified in the process of information analysis. The values of data items can be queried or modified by a set of application programs or transactions. As the states of the database corresponding to the states of the real world are consistent, a transaction can be regarded as a transformation of a database from one consistent state to another consistent state. Users' access to a database is facilitated by the software system called a DBMS, which provides services for maintaining consistency, integrity, and security of the database. Figure 2 illustrates a simplified database system. The transaction scheduler provides functions for transaction concurrency control, and the recovery manager is for transaction recovery in the presence of failures, which will be discussed in the next section.

The fundamental purpose of the DBMS is to carry out queries and transactions. A query is an expression, in a suitable language, that determines a portion of the data contained in the database (9). A query is considered as a read-only transaction. The goal of query processing is extracting information from a large amount of data to assist a decision-making process. A transaction is a piece of programming that manipulates the database by a sequence of read and write operations.

read(X) or R(X), which transfers the data item *X* from the database to a local buffer of the transaction

write(X) or W(X), which transfers the data item *X* from the local buffer of the transaction back to the database

In addition to *read* and *write* operations, a transaction starts with a *start* (or *begin*) operation and ends with a *commit* operation when the transaction succeeds or an *abort* when the transaction fails to finish. The following example shows a transaction transferring funds between two bank accounts (*start* and *end* operations are omitted).

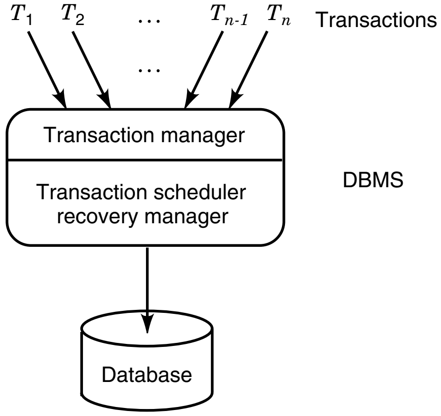


Figure 2. Database system and DBMS.

Example 1. Bank transfer transaction.

```
read($X$)
$X \rightarrow X + 100$
write($X$)
read($Y$)
$Y \rightarrow Y - 100$
write($Y$)
```

Here, X and Y stand for the balances of savings and credit accounts of a customer, respectively. This transaction transfers some money ($100) from the savings account to the credit account. It is an atomic unit of database work. That is, all these operations must be treated as a single unit.

Many database systems support multiple user accesses or transactions to the database. When multiple transactions execute concurrently, their operations are interleaved. Operations from one transaction may be executed between operations of other transactions. This interleaving may cause inconsistencies in a database, even though the individual transactions satisfy the specified integrity constraints. One such example is the *lost update* phenomenon.

Example 2. For the lost update phenomenon, assume that two transactions, crediting and debiting the same bank account, are executed at the same time without any control. The data item being modified is the account balance. The transactions read the balance, calculate a new balance based on the relevant customer operation, and write the new balance to the file. If the execution of the two transactions interleaves in the following pattern (supposing the initial balance of the account is $1500), the customer will suffer a loss:

Debit Transaction	Credit Transaction
read balance ($1500)	read balance ($1500)
withdraw ($1000)	deposit ($500)
balance := $1500 − $1000	balance := $1500 + $500
Write balance ($500)	Write balance ($2000)

The final account balance is $500 instead of $1000. Obviously, these two transactions have produced an inconsistent state of the database because they were allowed to operate on the same data item and neither of them was completed before another. In other words, neither of these transactions was treated as an atomic unit in the execution. Traditionally, transactions are expected to satisfy the following four conditions, known as ACID properties (9–11):

- *Atomicity* is also referred to as the all-or-nothing property. It requires that either all or none of the transaction's operations are performed. Atomicity requires that if a transaction fails to commit, its partial results cannot remain in the database.
- *Consistency* requires a transaction to be correct. In other words, if a transaction is executed alone, it takes the database from one consistent state to another. When all the members of a set of transactions are executed concurrently, the DBMS must ensure the consistency of the database.
- *Isolation* is the property that an incomplete transaction cannot reveal its results to other transactions before its commitment, which is the requirement for avoiding the problem of *cascading abort* (i.e., the necessity to abort all the transactions that have observed the partial results of a transaction that was later aborted).
- *Durability* means that once a transaction has been committed, all the changes made by this transaction must not be lost even in the presence of system failures.

The ACID properties are also defined in RM-ODP (Reference Model of Open Distributed Processing) (12). ODP is a standardization in a joint effort of the International Standardization Organization (ISO) and International Telecommunication Union (ITU), which describes systems that support heterogeneous distributed processing both within and between organizations through the use of a common interaction model.

Consistency and isolation properties are taken care of by the concurrency control mechanisms, whereas the maintenance of atomicity and durability are covered by the recovery services provided in transaction management. Therefore, concurrency control and recovery are the most important tasks for transaction management in a database system.

Concurrency Control and Serializability

The ACID properties can be trivially achieved by the sequential execution of transactions, which, however, is not a practical solution because it severely damages system performance. Usually, a database system is operating in a multiprogramming, multiuser environment, and the transactions are expected to be executed in the database system concurrently. In this section, the concepts of transaction concurrency control, the schedule of transactions, and the correctness criterion used in concurrency control are discussed.

A database system must monitor and control the concurrent executions of transactions so that overall correctness and database consistency are maintained. One of the primary tasks of the DBMS is to allow several users to interact with the database simultaneously, giving users the illusion that the database is exclusively for their own use (13). This feat is accomplished through a concurrency control mechanism.

Without a concurrency control mechanism, numerous problems can occur: the lost update (illustrated earlier in an example), the temporary update (or the uncommitted dependency), and the incorrect summary problems (7,14). The unwanted results may vary from annoying to disastrous in the critical applications. Example 3 shows a problem of temporary updates where a transaction T_B updates a data item f_1 but fails before completion. The value of f_1 updated by T_B has been read by another transaction T_A.

Example 3. Consider an airline reservation database system for customers booking flights. Suppose that a transaction A attempts to book a ticket on flight F_1 and on flight F_2 and that a transaction B attempts to cancel a booking on flight f_1 and to book a ticket on flight F_3.

Let f_1, f_2, and f_3 be the variables for the seat numbers that have been booked on flights F_1, F_2, and F_3, respectively. Assume that transaction B has been aborted for some reason so that the scenario of execution is as follows:

Transaction A	Transaction B
$R[f_1]$	$R[f_1]$
$f_1 = f_1 + 1$	$f_1 = f_1 - 1$
$W[f_1]$	$W[f_1]$
$R[f_2]$	$R[f_3]$
$f_2 = f_2 + 1$	$f_3 = f_3 + 1$
$W[f_2]$	$W[f_3]$
Commit transaction A	Abort transaction B

It is obvious that both transactions are individually correct if they are executed in a serial order (i.e., one commits before another). However, the interleaving of the two transactions shown here causes a serious problem in that the seat on fight F_1 canceled by transaction B may be the last seat available and transaction A books it before transaction B aborts, which results in one seat being booked by two clients.

Therefore, a database system must control the interaction among the concurrent transactions to ensure the overall consistency of the database. The execution sequence of operations from a set of transactions is called a *schedule*(15,16). A schedule indicates the interleaved order in which the operations of transactions were executed. If the operations of transactions are not interleaved (i.e., the executions of transactions are ordered one after another) in a schedule, the schedule is said to be serial. As we mentioned earlier, the serial execution of a set of correct transactions preserves the consistency of the database. As serial execution does not support concurrency, the equivalent schedule has been developed and applied for comparisons of a schedule with a serial schedule, such as view equivalence and conflict equivalence of schedules. In general, two schedules are *equivalent* if they have the same set of operations producing the same effects in the database (15).

Definition 1. Two schedules S_1, S_2 are *view equivalent* if

1. for any transaction T_i, the data items read by T_i in both schedules are the same; and
2. for each data item x, the latest value of x is written by the same transaction in both schedules S_1 and S_2.

Condition 1 ensures that each transaction reads the same values in both schedules, and

Condition 2 ensures that both schedules result in the same final systems.

In conflict equivalence, only the order of conflict operations needs to be checked. If the conflict operations follow the same order in two different schedules, the two schedules are conflict equivalent.

Definition 2. Two operations are in *conflict* if

1. they come from different transactions and
2. they both operate on the same data item and at least one of them is a write operation.

Definition 3. Two schedules S_1 and S_2 are *conflict equivalent* if for any pair of transactions T_i and T_j in both schedules and any two conflicting operations $O_{ip} \in T_i$ and $O_{jq} \in T_j$, when the execution order O_{ip} precedes O_{jq} in one schedule, say S_1, the same execution order must exist in the other schedule, S_2.

Definition 4. A schedule is *conflict serializable* if it is conflict equivalent to a serial schedule. A schedule is *view serializable* if it is view equivalent to a serial schedule.

A conflict serializable schedule is also view serializable but not vice versa because definition of view serializability accepts a schedule that may not necessarily be conflict serializable. There is no efficient mechanism to test schedules for view serializability. It was proven that checking for view serializability is an NP-complete problem (17). In practice, the conflict serializability is easier to implement in the database systems because the serialization order of a set of transactions can be determined by their conflicting operations in a serializable schedule.

The conflict serializability can be verified through a conflict graph. The conflict graph among transactions is constructed as follows: For each transaction T_i, there is a node in the graph (we also name the node T_i). For any pair of conflicting operations (o_i, o_j), where o_i from T_i and o_j from T_j, respectively, and o_i comes before o_j, add an arc from T_i to T_j in the conflict graph.

Examples 4 and 5 present schedules and their conflict graphs.

Example 4. A nonserializable schedule is shown here. Its conflict graph is shown in Fig. 3.

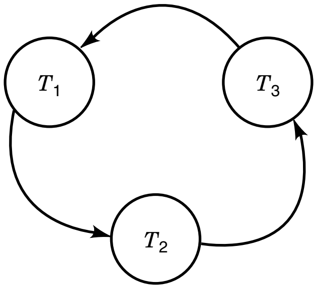


Figure 3. Conflict graph 1 (with a cycle).

Schedule	T_1	T_2	T_3
read(A)	read(A)		
read(B)		read(B)	
$A \leftarrow A + 1$	$A \leftarrow A + 1$		
read(C)			read(C)
$B \leftarrow B + 2$		$B \leftarrow B + 2$	
write(B)		write(B)	
$C \leftarrow C * 3$			$C \leftarrow C * 3$
write(C)			write(C)
write(A)	write(A)		
read(B)			read(B)
read(A)		read(A)	
$A \leftarrow A - 4$		$A \leftarrow A - 4$	
read(C)	read(C)		
write(A)		write(A)	
$C \leftarrow C - 5$	$C \leftarrow C - 5$		
write(C)	write(C)		
$B \leftarrow 6 * B$			$B \leftarrow 6 * B$
write(B)			write(B)

Example 5. A serializable schedule is shown here. Its conflict graph is shown in Fig. 4.

Schedule	T_1	T_2	T_3
read(A)	read(A)		
$A \leftarrow A + 1$	$A \leftarrow A + 1$		
read(C)	read(C)		
write(A	write(A)		
$C \leftarrow C - 5$	$C \leftarrow C - 5$		
read(B)		read(B)	
write(C)	write(C)		
read(A)		read(A)	
read(C)			read(C)
$B \leftarrow B + 2$		$B \leftarrow B + 2$	
write(B)		write(B)	
$C \leftarrow 3 * C$			$C \leftarrow 3 * C$
read(B)			read(B)
write(C)			write(C)
$A \leftarrow A - 4$		$A \leftarrow A - 4$	
write(A)		write(A)	
$B \leftarrow 6 * B$			$B \leftarrow 6 * B$
write(B)			write(B)

The following theorem shows how to check the serializability of a schedule.

Theorem 1. A schedule is conflict serializable if and only if its conflict graph is acyclic (15).

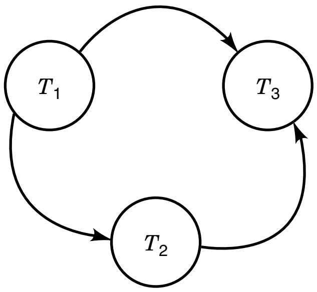


Figure 4. Conflict graph 2 (without cycle).

Intuitively, if a conflict graph is acyclic, the transactions of the corresponding schedule can be topologically sorted such that conflict operations are consistent with this order, and therefore equivalent to a serial execution in this order. A cyclic graph implies that no such order exists. The schedule in Example 4 is not serializable because there is cycle in the conflict graph; however, the schedule in Example 5 is serializable. The serialization order of a set of transactions can be determined by their conflicting operations in a serializable schedule.

In order to produce conflict serializable schedules, many concurrency control algorithms have been developed such as two-phase locking, timestamp ordering, and optimistic concurrency control.

The Common Concurrency Control Approaches

Maintaining consistent states in a database requires such techniques as semantic integrity control, transaction concurrency control, and recovery. Semantic integrity control ensures database consistency by rejecting update programs that violate the integrity constraints of the database, which is done by specifying the constraints during the database design. Then the DBMS checks the consistency during transaction executions. Transaction concurrency control monitors the concurrent executions of programs so that the interleaved changes to data items still preserve the database consistency. Recovery of a database system ensures that the system can cope with various failures in the system and recover the database to a consistent state.

A number of concurrency control algorithms have been proposed for the DBMSs. The most fundamental algorithms are two-phase locking (18,19), timestamp ordering (20,21), optimistic concurrency control (22), and serialization graph testing (23,24).

Two-phase locking (2PL) is one of the most popular concurrency control algorithms based on the locking technique. The main idea of locking is that each data item must be locked before a transaction accesses it (i.e., if conflicting operations exist, only one of them can access the data at a time, and the other must wait until the previous operation has been completed and the lock has been released). A transaction may involve accesses to many data items. The rule of 2PL states that all locks of the data items needed by a transaction should be acquired before a lock is released. In other words, a transaction should not release a lock until it is certain that it will not request any more locks. Thus, each transaction has two phases: an *expanding phase* during which new locks on data items can be acquired but none can be released and a *shrinking phase* in which the transaction releases locks and no new locks are required.

The 2PL algorithm is a very secure way to ensure that the order of any two transactions is compatible with the order of their conflicting operations. More precisely, if $o_{ip} \in T_i$ precedes $o_{jq} \in T_j$ in the schedule and o_{ip} is in conflict with o_{jq}, then all other conflicting operations of T_i, T_j must have the same order of precedence. The 2PL algorithms guarantee the conflict serializability of a schedule for concurrent transactions. However, 2PL algorithms may lead to deadlocks when a set of transactions wait for each other in a circular way. For example, two transactions T_1 and T_2 both write data items a and b. T_1 holds a lock on a and waits for a lock on b, whereas T_2 holds a lock on b and waits for a lock on a. In this case, T_1 and T_2 will be waiting for each other, and a deadlock occurs. When a deadlock occurs, some transactions need to be aborted to break the cycle.

Timestamp ordering (TO) is used to manage the order of the transactions by assigning timestamps to both transactions and data items. Each transaction in the system is associated with a unique timestamp, assigned at the start of the transaction, which is used to determine the order of conflicting operations between transactions. Each data item is associated with a read timestamp, which is the timestamp of the latest transaction that has read it, and a write timestamp, which is the timestamp of the latest transaction that has updated it. Conflicting operations must be executed in accordance with their corresponding transaction timestamps. A transaction will be aborted when it tries to read or write on a data item whose timestamp is greater than that of the transaction. The serializable order of transactions is the order of their timestamps.

Both 2PL and TO concurrency control algorithms are considered pessimistic approaches. The algorithms check every operation to determine whether the data item is available according to the locking or timestamp, even though the probability of conflicts between transactions is very small. This check represents significant overhead during transaction execution, with the effect of slowing down the TP.

Optimistic concurrency control (OCC) (22) is another approach in which no check is done while the transaction is executing. It has better performance if it is used in the environment where conflicts between transactions are rare. During transaction execution, each transaction executes three phases in its life time. The following three phases are used in the OCC protocol:

1. *Read Phase*. The values of the data items are read and stored in the local variables of the transaction. All modifications on the database are performed on temporary local storage without updating the actual database.
2. *Validation Phase*. According to the mutually exclusivity rules, a validation test is performed to determine whether the updates can be copied to the actual database.
3. *Write Phase*. If the transaction succeeds in the validation phase, the actual updates are performed to the database; otherwise, the transaction is aborted.

Optimistic approaches are generally used in conjunction with timestamps. A timestamp is assigned to a transaction at the end of its read phase or before the validation phase. The serialization order of transactions is then validated using the timestamps.

In a *serialization graph*-based concurrency control protocol, an online serialization graph (conflict graph) is explicitly maintained. The serialization graph testing (SGT) scheduler maintains a serialization graph for the history that represents the execution it controls. When a SGT scheduler receives an operation o_i of transaction T_i from the transaction manager, it first adds a node for T_i in the serialization graph (SG). The scheduler then checks whether there exists a previously scheduled operation o_k of transaction T_k conflicting with o_i. If there is one, an arc from T_k to T_i is added to the SG. The operations of transaction T_i can be executed as long as the graph is acyclic. Otherwise, the transaction, which causes a cycle in the graph, is aborted. As the acyclic serialization graph guarantees the serializability of the execution, the SGT scheduler produces the correct schedules for the concurrent transactions. However, it is not necessarily recoverable and is much less cascadeless or strict (14) as defined later.

A schedule S is said to be recoverable if, for every transaction T_i that reads data items written by another transaction T_j in S, T_i can be committed only after T_j is committed. That is, a recoverable schedule avoids the situation where a committed transaction reads the data items from an aborted transaction. A recoverable schedule may still cause cascading aborts because it allows the transactions to read from uncommitted transactions. For example, a transaction T_2 reads a data item x after x is updated by a transaction T_1, which is still active in an execution. If T_1 is aborted during the processing, T_2 must be aborted. Cascading aborts are undesirable.

To avoid cascading abortion in a schedule S, every transaction should read only those values written by committed transactions. Thus, a cascadeless schedule is also a recoverable schedule.

As a cascadeless schedule allows a transaction to write data from an uncommitted transaction, an undesirable situation may occur (14). For instance, consider the scenario of an execution

$$W_{T_1}[x,\ 2]W_{T_2}[x,\ 4].\,\mathrm{Abort}(T_1)\mathrm{Abort}(T_2)$$

where two transactions T_1 and T_2 write the same data item x, with values 2 and 4, respectively, and both are aborted later. The value of the data item x is called a before image if it will be replaced by a new value. The *before image* is saved in the log. In this case, the before image of data item x for transaction T_2 is 2 written by an aborted transaction T_1.

The term *strict schedule* was introduced in Ref. 14 to describe a very important property from a practical viewpoint. A schedule of transactions is called strict if the transactions read or write data items only from committed transactions. Strict schedules avoid cascading aborts and are recoverable. They are conservative and offer less concurrency.

The concurrency control algorithms presented above, such as 2PL, TO, and SGT, do not necessarily produce strict schedules by themselves.

If a strict schedule using 2PL algorithm is required, the locks being held by any transaction can be released only after the transaction is committed.

A TO approach with a strict schedule will not allow a transaction T to access the data items that have been updated by a previous uncommitted transaction even if transaction T holds a greater timestamp.

SGT can produce a strict schedule in such a way that each transaction cannot be committed until it is a source node of the serialization testing graph. That is, a transaction T could not be involved in a cycle of the serializable testing graph if previous transactions that T reads or writes from have all been committed.

Recoverability of Transactions

In addition to concurrency control, another important goal of transaction management is to provide a reliable and consistent database in the presence of various failures. Failures may corrupt the consistency of the database because the execution of some transactions may be only partially completed in the database. In general, database systems are not failure-free systems. A number of factors cause failures in a database system (9) such as:

1. *Transaction Abortions*. The situation can be caused by the transaction itself, which is caused by some unsatisfactory conditions. Transaction abortion can also be forced by the system. These kinds of failure do not damage the information stored in memory, which is still available for recovery.
2. *System Crashes*. The typical examples of this type of failure are system crashes or power failures. These failures interrupt the execution of transactions, and the content of main memory is lost. In this case, the only available accessible information is from a stable storage, usually a disk.
3. *Media Failures*. Failures of the secondary storage devices that store the database are typical of media failure. As the content of stable storages has been lost, the system cannot be recovered by the system software only. The common technique to prevent such unrecoverable failures is to replicate the information on several disks.

The first two types of failures are considered in the recovery of transactions. Transactions represent the basic units of recovery in a database system. If the automicity and durability of the execution of each transaction have been guaranteed in the presence of failures, the database is considered to be consistent.

Typically, the piece of software responsible for recovery of transactions is called the recovery manager (RM). It is required to ensure that whenever a failure occurs, the database is brought back to the consistent state it was in before the failure occurred. In other words, the RM should guarantee that updates of the database by the committed transactions are permanent, in contrast to any partial effects of uncompleted transactions that should be aborted.

The basic technique for implementing transactions in the presence of failures is based on the use of logs. A log is a file that records all operations on the database carried out by all transactions. It is supposed that a log is accessible after the failures occur. The log is stored in *stable storage*, which is the most resilient storage medium available in the system. Stable storage is also called *secondary storage*. Typically, it is implemented by means of duplexed magnetic tapes or disks that store duplicate copies of the data. The replicated stable storage is always kept mutually consistent with the primary copy of the disk or tape. The database is stored permanently on stable storage. The updates on a database by a transaction are not directly written into the database immediately. The operations of the transactions are implemented in the *database buffer* located in main memory (also referred to as volatile storage). It is only when the contents of the database buffer have been *flushed* to stable storage that any update operation can be regarded as durable.

It is essential that the log record all the updates on the database that have been carried out by the transactions in the system before the contents of the database buffer have been written to database, which is the rule of *write-ahead log*.

A log contains the information for each transaction as follows:

- transaction identifier;
- list of update operations performed by the transaction (for each update operation, both the old value and the new value of the data items are recorded); and
- status of the transaction: tentative, committed, or aborted.

The log file records the required information for undoing or redoing the transaction if a failure occurs. As the updates were written to the log before flushing the database buffer to the database, the RM can surely preserve the consistency of the database. If a failure occurs before the commit point of a transaction is reached, the RM will abort the transaction by *undo*ing the effect of any partial results that have been flushed into the database. On the other hand, if a transaction has been committed but the results have not been written into the database at the time of failure, the RM would have to *redo* the transaction, using the information from the log, in order to ensure transaction durability.

DISTRIBUTED TRANSACTION PROCESSING

In many applications, both data and operations are often distributed. A database is considered distributed if a set of data that belongs logically to the same system is physically spread over different sites interconnected by a computer network. A site is a host computer and the network is a computer-to-computer connection via the communication system. Although the software components that are typically necessary for building a DBMS are also the principal components for a distributed DBMS (DDBMS), some additional capacities must be provided for a distributed database, such as the mechanisms of distributed concurrency control and recovery.

One of the major differences between a centralized and a distributed database system lies in the TP. In a distributed database system, a transaction might involve data residing on multiple sites (called a global transaction). A global transaction is executed on more than one site. It consists of a set of subtransactions, each subtransaction involving data residing on one site. As in centralized databases, global transactions are required to preserve the ACID properties. These properties must be maintained individually on each site and also globally. That is, the concurrent global transactions must be serializable and recoverable in the distributed database system. Consequently, each subtransaction of a global transaction must be either performed in its entirety or not performed at all.

Serializability in a Distributed Database

Global transactions perform operations at several sites in a distributed database system (DDBS). It is well understood that the maintenance of the consistency of each single database does not guarantee the consistency of the entire distributed database. It follows, for example, from the fact that serializability of executions of the subtransactions on each single site is only a necessary (but not sufficient) condition for the serializability of the global transactions. In order to ensure the serializability of distributed transactions, a condition stronger than the serializability of single schedule for individual sites is required.

In the case of distributed databases, it is relatively easy to formulate a general requirement for correctness of global transactions. The behavior of a DDBS is the same as a centralized system but with distributed resources. The execution of the distributed transactions is correct if their schedule is serializable in the whole system. The equivalent conditions are:

- Each local schedule is serializable, and
- The subtransactions of a global transaction must have a compatible serializable order at all participating sites.

The last condition means that, for any two global transactions G_i and G_j, their subtransactions must be scheduled in the same order at all the sites on which these subtransactions have conflicting operations. Precisely, if G_{ik} and G_{jk} belongs to G_i and G_j, respectively, and the local serializable order is G_{ik} precedes G_{jk} at site k, then all the subtransactions of G_i must precede the subtransactions of G_j at all sites where they are in conflict.

Various concurrency control algorithms such as 2PL and TO have been extended to DDBS. As the transaction management in a DDBS is implemented by a number of identical local transaction managers, the local transaction managers cooperate with each other for the synchronization of global transactions. If the timestamp ordering technique is used, a global timestamp is assigned to each subtransaction, and the order of timestamps is used as the serialization order of global transactions. If a two-phase locking algorithm is used in the DDBS, the locks of a global transaction cannot be released at all local sites until all the required locks are granted. In distributed systems, the data item might be replicated. The updates to replicas must be atomic (i.e., the replicas must be consistent at different sites). The following rules may be used for locking with n replicas:

1. Writers need to lock all n replicas; readers need to lock one replica.
2. Writers need to lock all m replicas ($m > n/2$); readers need to lock $n - m + 1$ replicas.
3. All updates are directed first to a primary copy replica (one copy has been selected as the primary copy for updates first and then the updates will be propagated to other copies).

Any one of these rules will guarantee consistency among the duplicates.

Atomicity of Distributed Transactions

In a centralized system, transactions can either be processed successfully or aborted with no effects left on the database in the case of failures. In a distributed system, however, additional types of failure may happen.

For example, network failures or communication failures may cause network partition, and the messages sent from one site may not reach the destination site. If there is a partial execution of a global transaction at a partitioned site in a network, it would not be easy to implement the atomicity of a distributed transaction. To achieve an atomic commitment of a global transaction, it must be ensured that all its subtransactions at different sites are capable and available to commit. Thus, an agreement protocol has to be used among the distributed sites. The most popular atomic commitment protocol is the two-phase commitment (2PC) protocol.

In the basic 2PC, there is a *coordinator* at the originating site of a global transaction. The participating sites that execute the subtransactions must commit or abort the transaction unanimously. The coordinator is responsible for making the final decision to terminate each subtransaction. The first phase of 2PC is to request from all *participants* the information on the execution state of subtransactions. The participants report to the coordinator, which collects the answers and makes the decision. In the second phase, that decision is sent to all participants. In detail, the 2PC protocol proceeds as follows for a global transaction T_i(9):

Two-Phase Commit Protocol Phase 1: Obtaining a Decision.

1. Coordinator asks all participants to prepare to commit transaction T_i:
   a. Add [prepare T_i] record to the log.
   b. Send [prepare T_i] message to each participant.
2. When a participant receives [prepare T_i] message, it determines if it can commit the transaction:
   a. If T_i has failed locally, respond with [abort T_i].
   b. If T_i can be committed, send [ready T_i] message to the coordinator.
3. Coordinator collects responses:
   a. All respond "ready"; decision is commit.
   b. At least one response is "abort"; decision is abort.
   c. At least one fails to respond within time-out period, decision is abort.

Phase 2: Recording the Decision in the Database1.

1. Coordinator adds a decision record ([abort T_i] or [commit T_i]) in its log.
2. Coordinator sends a message to each participant informing it of the decision (commit or abort).
3. Participant takes appropriate action locally and replies "done" to the coordinator.

The *first phase* is that the coordinator initiates the protocol by sending a "prepare-to-commit" request to all participating sites. The "prepare" state is recorded in the log, and the coordinator is waiting for the answers. A participant will reply with a "ready-to-commit" message and record the "ready" state at the local site if it has finished the operations of the subtransaction successfully. Otherwise, an "abort" message will be sent to the coordinator, and the subtransaction will be rolled back accordingly.

The *second phase* is that the coordinator decides whether to commit or abort the transaction based on the answers from the participants. If all sites answered "ready-to-commit," then the global transaction is to be committed. The final "decision-to-commit" is issued to all participants. If any site replies with an "abort" message to the coordinator, the global transaction must be aborted at all the sites. The final "decision-to-abort" is sent to all the participants who voted the "ready" message. The global transaction information can be removed from the log when the coordinator has received the "completed" message from all the participants.

The basic idea of 2PC is to make an agreement among all the participants with respect to committing or aborting all the subtransactions. The atomic property of global transaction is then preserved in a distributed environment.

The 2PC is subject to the blocking problem in the presence of site or communication failures. For example, suppose that a failure occurs after a site has reported "ready-to-commit" for a transaction, and a global commitment message has not yet reached this site. This site would not be able to decide whether the transaction should be committed or aborted after the site is recovered from the failure. A three-phase commitment (3PC) protocol (14) has been introduced to avoid the blocking problem. But 3PC is expensive in both time and communication cost.

Transaction Processing in Heterogeneous Systems

Traditional DDBS are often homogeneous because local database systems are the same, using the same data models, the same languages, and the same transaction managements. However, in the real world, data are often partitioned across multiple database systems, file systems, and applications, all of which may run on different machines. Users may run transactions to access several of these systems as single global transactions. A special case of such systems are multidatabase systems or federated database systems.

As the 2PC protocol is essential to support the atomicity of global transactions and, at the same time, the local systems may not provide such support, layers of software are needed to coordinate and the execution of global transactions (25) for transactional properties of concurrency and recovery. A TP monitor is a piece of software that connects multiple clients to multiple servers to access multiple databases/data resources as shown in Fig. 1. Further discussions on TP monitors can be found in Ref. 1.

ADVANCED TRANSACTION PROCESSING

In traditional database applications such as banking and airline reservation systems, transactions are short and noncooperative and usually can be finished in minutes. The serializability is a well-accepted correctness criterion for these applications. TP in advanced applications such as cooperative work will have different requirements, need different correctness criteria, and require different systems supports to coordinate the work of multiple designers/users and to maintain the consistency. Transactions are often called advanced transactions if they need nonserializable correctness criteria. Many advanced transaction models have been discussed in the literature 2–5. In this section, we will briefly examine some advanced transaction models and then present a general advanced transaction model and its correctness criterion.

Advanced Transaction Model

In addition to advanced transactions, we can also see other similar terms such as nontraditional transactions, long transactions, cooperative transactions, and interactive transactions. We will briefly list some work on advanced TP or cooperative TP in advanced database transaction models (2,3), groupware (4,26,27), and workflow systems (5,28).

Advanced Database Transaction Models (3).

1. Saga (29). A transaction in Saga is a long-lived transaction that consists of a set of relatively independent steps or subtransactions, T_1, T_2 ,..., T_n. Associated with each subtransaction T_i is a compensating transaction C_i, which will undo the effect of T_i. Saga is based on the compensation concept. Saga relaxes the property of isolation by allowing a Saga transaction to reveal its partial results to other transactions before it completes. As a Saga transaction can interleave its subtransactions with subtransactions of other sagas in any order, consistency or serializability is compromised. Saga preserves atomicity and durability of traditional transaction by using forward and backward recoveries.
2. Cooperative Transaction Hierarchy (30). This model supports cooperative applications like computer-aided design (CAD). It structures a cooperative application as a rooted tree called a cooperative transaction hierarchy. The external nodes represent the transactions associated with the individual designers. An internal node is called a transaction group. The term cooperative transaction refers to transactions with the same parent in the transaction tree. Cooperative transactions need not to be serial-

izable. Isolation is not required. Users will define correctness by a set of finite automata to specify the interaction rules between cooperative transactions.

3. Cooperative SEE Transactions (31). This model supports cooperative work in software engineering environments (SEEs). It uses nested active transactions with user-defined correctness. ACID properties are not supported.
4. DOM Transaction Model for distributed object management (32). This model uses open and closed nested transactions and compensating transactions to undo the committed transactions. It also use contingency transactions to continue the required work. It does not support ACID properties.
5. Others (3). Open nested transactions, ConTract, Flex, S, and multilevel transactions models use compensating transactions and contingency transactions. The ACID properties are compromised. The polytransaction model uses user-defined correctness. Tool Kit also uses user-defined correctness and contingency transactions to achieve the consistency.

Groupware (2,26,33). Most groupware systems synchronize cooperative access to shared data in a more or less ad hoc manner. Groupware systems involve multiple concurrent users or several team members at work on the same task. The members, or users, are often in different locations (cities or even countries). Each team member starts up a cooperative transaction, each cooperative transaction should be able to see the intermediate result of other cooperative transactions, and these cooperative transactions jointly form a cooperative transaction group. When they read or update the uncommitted data from other cooperative transactions, nonserializable synchronization and concurrency mechanisms are required to maintain consistency. A cooperative editing system is an example.

Workflow Applications (5). Workflow is used to analyze and control complicated business processes. A large application often consists of a collection of tasks. Each task can be viewed as a cooperative transaction processed by one user or designer, and these tasks are partially ordered by control and data flow dependencies. The workflow supports the task coordination specified in advance through the control flow. Serializability is not preserved either.

These applications have some common properties: (1) users are often distributed; (2) they conduct some cooperative work in an interactive fashion; and (3) this interactive cooperative work may take a long time. These applications have the following special consistency requirements:

1. A transaction may read intermediate results produced by other transactions.
2. The consistency between individual and group needs to be maintained.

Based on this summary, we give the following definition.

Definition 5. An advanced transaction (cooperative transaction group) is defined as a set (group) of cooperative transactions $T_1, T_2, \ldots, T_n$, with the following properties:

1. Each cooperative transaction is a sequence (or partial order) of read(x) and write(y) operations.
2. For the same data item, there might be more than one read(x), written as read^1(x), read^2(x), . . ., in a cooperative transaction, and each read(x) will get a different value depending on the time and interaction with other transactions.
3. Similarly, for each y, there might be more than one write(y), written as write^1(y),,write^2(Y), . . ., each of which will produce an individual version of data item y.

The first part shows that an advanced transaction is a cooperative transaction group. If the size of the group is one, it will become a single transaction. The property 1 is the same as that in traditional transactions. The second and third properties indicate some cooperative features. The first read(x) may read other transaction's committed or uncommitted data depending on the concurrency control employed. After the first read operation on x, the data item might be updated by another transaction or another cooperative transaction; then it can read the new value in the next read(x). Similarly, after the first write operation on x, because of the cooperative feature, a transaction may read some new data from other transactions and then issue another write(x) to incorporate it to the current processing. The later write(x) can undo the previous write or do a further update to show the new semantics.

To further justify the second and third properties of the definition, we discuss their compatibilities with interactive and noninteractive transactions in advanced transaction applications.

Interactive Transactions. A cooperative transaction can be formed with great flexibility because a user can dynamically issue an operation depending on the most current information. If a data item has been updated recently after the first read, the cooperative transaction may wish to read the data again because of the cooperative feature. In order to incorporate the recent changes in to its own transaction, it can perform additional operations or compensate for the previous operations, which is also the flexibility of interactive work.

Noninteractive Transactions. In some database transaction models, the transactions are not as interactive as those online transactions from groupwares and transaction workflow applications (3). To maintain system consistency and meet the application requirements, all of them use compensating transactions, contingency transactions, or triggers, where a compensating transaction is a transaction undoing the effect of a previous transaction; a contingency transaction is a transaction to continue or extend a previous transaction; and the trigger is a mechanism to invoke another transaction (if the trigger condition is true) to restore the consistency. A compensating transaction, a contingency transaction, or a trigger can be viewed as an extension of a transaction that violates the consistency requirements during the execution, and the extended part will have the read and write operations on some data items in common. They are another type of interaction. These interactions need to be programmed in advance; therefore, they are not as flexible as interactive transactions. But the interactive features are still required even for these noninteractive database transaction applications.

Similar to distributed database transactions, the advanced transaction definition could be extended to a distributed advanced transaction as follows:

Definition 6. A distributed advanced transaction (distributed cooperative transaction group) is defined as a set (group) of cooperative transactions T_1, T_2,. . ., T_n, with the following properties:

1. Each transaction T_i consists of a set of subtransactions T_i^j at site j, $j \in [1 \cdot \cdot m]$, m is the number of sites in a distributed system. Some T_i^j might be empty if T_i has no subtransaction at site j.
2. Each subtransaction is a sequence (or partial order) of read(x) and write(y) operations.
3. For the same data item x, there might be more than one read(x), denoted as read^1(x), read^2(x),. . ., in a cooperative transaction, each read(x) will get a different value depending on the time and interaction with other transactions.
4. Similarly, for each y, there might be more than one write(y), denoted as write^1(y), write^2(y),. . ., each of which will produce an individual version of data item y.

Just as the serializability theory plays an important role in the traditional transaction model in developing concurrency control and recovery algorithms, a general correctness theory for advanced transactions is also required to guide transaction management for advanced applications. In the next subsection, we will present such a correctness criterion.

f-Conflict Serializability

As in the traditional transactions, we can assume that, for write operations on x, there must be a read operation before the first write in a cooperative transaction. It is natural to read the data first before the update [i.e., one's update may depend on the read value or one may use a read operation to copy the data into the local memory, then update the data and write it back (when the transaction commits)].

In advanced transaction applications, cooperative transactions could read and write a data item more than once, which is different from traditional transactions. The reason for reading a data item more than once is to know the recent result and therefore make the current transaction more accurate, which, however, will violate the serializability, because a cooperative transaction may read a data item before another transaction starts and also read the data updated by the same transaction. If so, the schedule between these two transactions will not be serializable. However, from the semantic point of view, the most important read or write on the same data item will be the last read or write. If we give high priority to the last read or write conflicts in developing the correctness criteria, we could have an f-conflict (final conflict) graph, based on which we will present an f-conflict serializability theorem as a general correctness criterion for advanced TP.

Definition 7. The f-conflict graph among transactions is constructed as follows. For each transaction T_i, there is a node in the graph (we also name the node T_i). For any pair of final conflicting operations (O_i, O_j), where O_i from T_i and O_j from T_j, respectively, and O_i comes earlier than O_j, add an arc from T_i to T_j in the conflict graph.

Definition 8. A schedule is f-conflict serializable if and only if its f-conflict graph is acyclic. The f-conflict serialization order of a set of transactions can be determined by their f-conflicting operations in an f-conflict serializable schedule. From the definitions, we can see the relationship between conflict serializability and f-conflict serializability.

Theorem 2. If a schedule is conflict serializable, it is also f-conflict serializable; the reverse is not true.

The conflict serializability is a special case of f-conflict serializability in traditional TP.

Definition 9. A schedule of distributed advanced transactions is f-conflict serializable if and only if

1. the schedule of subtransactions at each site is f-conflict serializable, and
2. the f-conflict serialization order at all sites are the same.

Advanced transactions or cooperative transactions may have different application-dependent requirements and require different system supports to coordinate the work of multiple users and to maintain the consistency. As a result, different synchronization, coordination, and control mechanisms within a cooperative transaction group are developed. The f-conflict serializability in conjunction with application-dependent semantics could be used for designing and testing advanced TP approaches. The application-dependent requirements can be reflected in the detailed transaction structures. For example, when there are several write operations on the same x, the later write might be to undo and then redo the operation (or perform a different operation). The undo operations might be reversing operations or compensating operations, and the redo operations could be contingency operations or new operations that may need to keep the intention (user intention) of the original write (6,27) or to incorporate the new semantics.

In a recent work, we have verified a cooperative editing system, REDUCE, according to this theory, and have shown that the schedules from this system is f-conflict serializable (34).

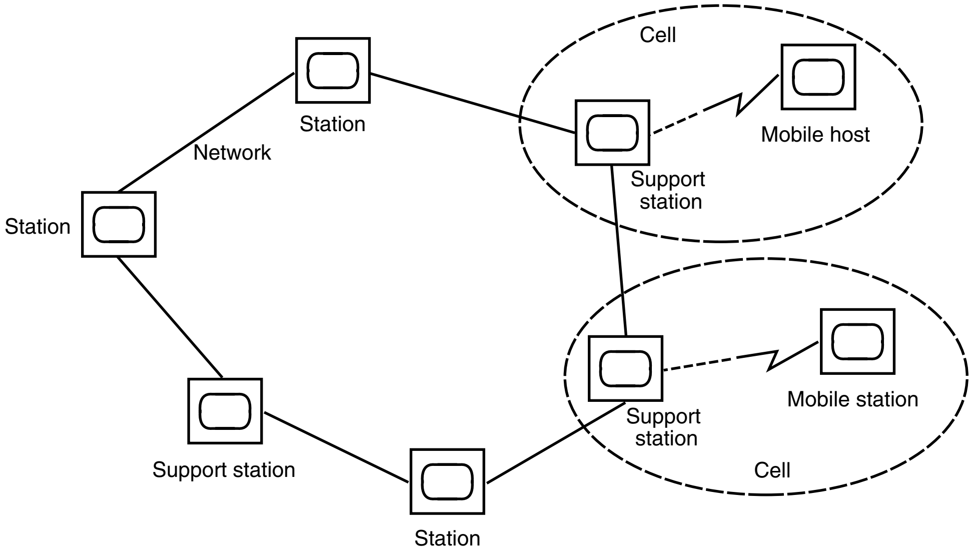


Figure 5. Wired and wireless net-working environment.

Advanced transactions are very long when compared with traditional transactions. The arbitrary abortion of such long transactions is not appropriate because aborting long transactions means increasing the processing cost and response time. In an environment with short (traditional) transactions and long/cooperative transactions, long/cooperative transactions should not be aborted because of conflict operations with short transactions. On the other hand, because the quick response is often required or preferred for short transactions, long transactions should not block the short transactions.

Based on the f-conflict serializability, a timestamp ordering concurrency control algorithm (35) is developed to support both traditional short transactions and long cooperative transactions. With this new timestamp ordering method, short transactions can be processed in the traditional way, as if there are no cooperative transactions. Therefore, they will not be blocked by long transactions; a cooperative transaction will not be aborted when there is a conflict with short transactions, rather, it will incorporate the recent updates into its own processing. The serializabilities, among short transactions, and between a cooperative transaction (group) and other short transactions, are all preserved.

Mobile Transaction Processing

In both centralized and DDBS, data and machines have fixed locations. As a result of recent advances in the development of portable computing devices and wireless communication networks, mobile computing began to emerge in many database applications. The mobile computing environment consists of mobile computers, known as mobile hosts, and a wired network of computers, some of which are mobile support stations through which mobile hosts can communicate with the wired network. Each mobile support station manages those mobile hosts within its cell, the geographical area it covers. Figure 5 shows both a wired and wireless connected networking environment.

Mobile computing systems can be viewed as an extension of distributed systems (36). However, to support TP in the mobile computing environment, physical limitations imposed by the nature of the networking environment have to be taken into consideration (37,38).

- Communication between mobile hosts and mobile support stations is asymmetric. Bandwidth in the upstream direction from mobile hosts to mobile support stations is low, resulting in excessive latency.
- Portable computing devices have a limited battery life, processing capability, and storage capacity.
- Most mobile hosts do not stay continuously connected, for a number of reasons, including reducing connection charges and saving power. Mobile hosts can also move between cells, disconnecting one cell to connect to another.

In such an environment, the characteristics of mobile transactions can differ in a number of ways from transactions in distributed systems (39,40).

- When a mobile host moves to a new cell during the execution of a transaction, it might need to continue its execution in another cell. Therefore, a mobile transaction might have to split its computation in that some parts of the computation are executed on the mobile host and others on different fixed hosts.
- A mobile transaction tends to be long-lived because of the high latency of wireless communication and long disconnection time.
- A mobile transaction tends to be prone to failure.

- A mobile transaction may be running in a distributed and heterogeneous system.

Traditional TP protocols may not address these distinctive characteristics of mobile computing systems and mobile transactions. To support TP in a mobile computing environment efficiently and effectively, a number of desirable features should be supported.

- Operations on shared data must ensure correctness of transactions executed on both mobile hosts and fixed hosts.
- Transaction aborts and blocking should be minimized to save resources and to increase concurrency. Early detection of data conflicts leading to transaction restarts is required.
- Communication between mobile hosts and support stations should be minimized and adaptable to the network connectivity.
- Autonomy for mobile transactions to be processed locally during disconnection should be supported.

A traditional distributed transaction consists of a set of subtransactions that are executed concurrently at multiple sites and there is one coordinator to coordinate the execution and commitment of these subtransactions. A mobile transaction is another kind of distributed transaction. The entire transaction can be submitted in a single request from the mobile host, or the operations of a transaction are submitted in multiple requests, possibly to different support stations in different cells. The former method involves a single coordinator for all the operations of the transaction, whereas the latter may involve multiple coordinators. For example, after submitting some operations (and getting partial results back), the mobile host might need to submit the remaining operations to another cell because it has moved to a new cell. The execution of the mobile transaction is not fully coordinated by a single coordinator because, to a certain extent, it depends on the movement of the mobile computer. The kangaroo transaction model (41) uses a split operation to create a new subtransaction when the mobile computer hops from one cell to another. A subtransaction is a global or a local transaction that can be committed independently and the failure of one may result in the entire kangaroo transaction being undone. To manage the execution of a kangaroo transaction, a data structure is maintained between the mobile support stations involved.

In typical multidatabase systems where users may simultaneously access heterogeneous data from different local databases, a global locking table can be maintained for correct execution of concurrent global and local transactions. In the mobile environment, intensive communication of locking information between the local sites and the global transaction manager is impractical because of the physical limitations of the networking environment. A hierarchical concurrency control algorithm using global locking table with semantic information contained within the hierarchy can be used to dynamically adjust the amount of communication required to detect and resolve data conflicts (42). To reduce the impact on local transactions due to the processing of the long-lived global transactions submitted by mobile users, the *Pre-Serialization* technique allows global transactions to establish their serialization order before completing execution (43). In this way, subtransactions of a global transaction can be committed independently at local sites and resources may be released in a timely manner.

Guaranteeing the consistency of data processed by mobile hosts is harder because mobile hosts are often disconnected from the rest of the network while still in operation. For instance, if a data item cached in a mobile computer is updated by another computer while the mobile computer is disconnected, the cached data will become inconsistent or out of date. If a conventional lock-based approach is adopted in the mobile computing environment to maintain data consistency, the system could suffer from significant performance degradation as the data items held by a long-lived mobile transaction could not be released until the transaction commits. To improve data availability, a transaction can *pre-commit* at the mobile host (44) so that the future value of a data object can be made visible to other transactions before the delayed final commit of the transaction at the mobile support station, which reduces the blocking of other transactions to increase concurrency and costly transaction aborts can also be avoided as a pre-committed transaction is guaranteed to commit.

During disconnection, mobile host users may issue query or update transactions on the data that reside locally. Data are often replicated or cached at mobile hosts for reasons of performance and availability. To support TP in a networking environment with intermittent links, *weak transactions*(45) let users access local data in mobile computing applications where bounded inconsistency is acceptable. In a weak transaction, weak read operations read local, potentially inconsistent copies and weak write operations perform tentative updates. Data reconciliation can be activated when the mobile computer is reconnected to the wired network.

In mobile computing systems, the number of mobile hosts is far greater than the number of support stations, and support stations have a relatively abundant downstream bandwidth. The pull-based architecture in traditional distributed systems, where data items are delivered from servers to clients on a demand basis, is no longer a good match in mobile computing systems. In contrast, push-based data delivery fits well the inherent communication asymmetry to exploit the abundant downstream bandwidth in mobile computing systems. In the push-based architecture called *Broadcast Disks*(46), data items are continuously and repetitively broadcast to mobile hosts without any specific request, and the mobile hosts listen to the broadcast channel and retrieve data of their interest. Data dissemination can be found in many applications, including stock trading and electronic auctions. In these applications, data updates must be disseminated promptly and consistently to a large community of mobile users.

In the broadcast environment, data items may be updated by transactions executed at the server while they are being broadcast. To ensure the consistency of mobile transactions, the broadcast channel can be used to transmit concurrency control-related information to the mobile hosts to perform all or part of the transaction validation function (47–49). In this way, data conflicts can be detected earlier at the mobile hosts to avoid any computing and communication resources being wasted, as well as helping to improve the performance of mobile transactions. In addition, transaction restarts are more costly in the mobile environment. Excessive transaction aborts because of ineffectiveness of the concurrency control mechanisms or unnecessary restrictive correctness criteria should be avoided (50).

To increase the concurrency of mobile transactions, multiple versions of data items can be broadcast (51). A mobile read-only transaction can choose to read the data versions, if they exist, that correspond to a single database state. With multiversioning, mobile transactions can resume execution after temporary disconnection, as long as the required versions are still on the broadcast. To provide better currency, additional information in the form of an invalidation report consisting of a list of data items that have been updated can periodically be broadcast to the mobile hosts.

Mobile transactions also introduce some other new problems, such as awareness of location. In wired DDBSs, location transparency is an important feature. However, mobile applications may be location-dependent, for instance, the current position of a mobile host may be accessed by a mobile transaction. Moreover, failures occur more often in mobile computing because of the frequent switching on and off of mobile computers and the frequent handoff when mobile computers move across the boundary of cells. Another new challenge in the mobile computing environment is failure handling and recovery.

FUTURE RESEARCH DIRECTIONS

The future work on TP will continue in the direction on new transaction models. Although the advanced transaction model and f-conflict serializability provide a guideline for advanced application, many particular applications still need user-defined correctness and often employ the semantic information for semantic serializability and semantic atomicity.

In advanced database applications such as CAD and cooperative work, the transactions are often cooperative or interactive or online analysis processing. We need to design mechanisms for advanced models to support partial rollbacks, reread, and rewrite operations to reflect the cooperative features.

As database systems are being deployed in more and more complex applications, the traditional data model (e.g., the relational model) has been found to be inadequate and has been extended (or replaced) by object-oriented data models. Related to this extension is another research direction: TP in object-oriented databases, including semantic-based concurrency control and recovery in object-oriented databases. Ref. 52 presents a brief introduction and some future research topics on this area as well as a comprehensive list of references on advanced TP.

ACKNOWLEDGMENT

We thank Anne Fuller for her comments and review on an earlier version of this article.

BIBLIOGRAPHY

1. P. A. Bernstein and E. Newcomer, *Principles of Transaction Processing*, San Mateo, CA: Morgan Kaufmann, 1997.
2. K. Abrer et al., Transaction models supporting cooperative work-TransCoop experiences, in Y. Kambayashi and K. Yokota (eds.), *Cooperative Databases and Applications*, Singapore: World Scientific, 1997, pp. 347–356.
3. A. K. Elmagarmid, *Database Transaction Models for Advanced Applications*, San Mateo, CA: Morgan Kaufmann, 1992.
4. C. A. Ellis and S. J. Gibbs, Concurrency control in groupware systems, *Proc. ACM SIGMOD*, 1989, pp. 399–407.
5. M. Rusinkiewicz and A. Sheth, Specification and execution of transactional workflows, in W. Kim (ed.), *Modern Database Systems*, Reading, MA: Addison-Wesley, 1994, pp. 592–620.
6. C. Sun et al., A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems, *Proc. ACM Group97*, Phoenix, AZ, 1997, pp. 425–434.
7. R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, Menlo Park, CA: Benjamin/Cummins, 1989.
8. A. Silberschatz , H. Korth, and S. Sudarshan, *Database Systems Concepts*, New York: McGraw-Hill, 1991.
9. S. Ceri and G. Pelagate, *Distributed Databases: Principles and Systems*, New York: McGraw-Hill, 1984.
10. T. Haerder and A. Reuter, Principles of transaction-oriented database recovery, *ACM Comput. Surv.*, **15** (4): 287–317, 1983.
11. J. N. Gray, The transactions concept: Virtues and limitations, *Proc. 7th Int. Conf. Very Large Data Base*, 1981, pp. 144–154.
12. ISO/IEC DIS 10746-2, Basic reference model of open distributed Processing - Part 2: descriptive model [Online]. Available: http://www.dstc.edu.au/AU/ODP/standards.html.
13. D. Agrawal and A. El. Abbadi, Transaction management in database systems, *Database Trans. Models Adv. Appl.*, 1–32, 1992.
14. C. J. Date, *An Introduction to Database System*, Vol. 2, Reading, MA: Addison-Wesley, 1982.
15. P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Reading, MA: Addison-Wesley, 1987.
16. H. Korth, A. Silberschatz, *Database Systems Concepts*, 2nd ed. New York: McGraw-Hill, 1991.
17. C. Papadimitriou, *The Theory of Database Concurrency Control*, Rockville MD: Computer Science Press, 1986.
18. K. P. Eswaran et al., The notions of consistency and predicate locks in a database system, *Commun. ACM*, **19** (11): 624–633, 1976.
19. J. N. Gray, Notes on database operating systems, *Lect. Notes Comput. Sci.*, **6**: 393–481, 1978.
20. P. A. Bernstein and N. Goodman, Timestamp based algorithms for concurrency control in distributed database systems, *Proc. 6th Int. Conf. VLDB*, 285–300, 1980.
21. L. Lamport, Time, clocks and the ordering of events in a distributed system, *Commun. ACM*, **21** (7): 558–565, 1978.
22. H. T. Kung and J. T. Robinson, On optimistic methods for concurrency control, *Proc. Conf. VLDB*, 1979.

23. D. Z. Badal, Correctness of concurrency control and implications in distributed databases, *COMPSAC Conf.*, 1979, pp. 588–593.

24. M. A. Casanova, Concurrency control problem of database systems, *Lect. Notes Comput. Sci.*, **116**: 1981.

25. A. Silberschatz, H. Korth, and S. Sudarshan, *Database Systems Concepts*, 3rd ed., New York: McGraw-Hill, 1991.

26. S. Greenberg and D. Marwood, Real time groupware as a distributed system: Concurrency control and its effect on the interface, *Proc. ACM Conf. CSCW'94*, 1994, pp. 207–217.

27. C. Sun et al., Achieving convergency, causality-preservation and intention preservation in real-time cooperative editing systems, *ACM Trans. Comput.-Hum. Interact.*, **5** (1): 1–42, 1998.

28. D. Jean, A. Cichock, and M. Rusinkiewicz, A database environment for workflow specification and execution, in Y. Kambayashi and K. Yokota (eds.), *Cooperative Databases and Applications*, Singapore: World Scientific, 1997, pp. 402–411.

29. H. Garcia-Molina and K. Salem, *Sagas, Proc. ACM SIGMOD Conf. Manage. Data*, 1987, pp. 249–259.

30. M. Nodine and S. Zdonik, Cooperative transaction hierarchies: A transaction model to support design applications, in A. K. Elmagarmid (ed.), *Database Transaction Models for Advanced Applications*, San Mateo, CA: Morgan Kaufmann, 1992, pp. 53–86.

31. G. Heiler et al., A flexible framework for transaction management in engineering environments, in A. Elmagarmid (ed.), *Transaction Models for Advanced Applications*, San Mateo, CA: Morgan Kaufmann, 1992, pp. 87–112.

32. A. Buchmann, M. T. Ozsu, and M. Hornick, A transaction model for active distributed object systems, in A. Elmagarmid (ed.), *Transaction Models for Advanced Applications*, San Mateo, CA: Morgan Kaufmann, 1992, pp. 123–158.

33. C. A. Ellis, S. J. Gibbs, and G. L. Rein, Groupware: Some issues and experiences, *Commun. ACM*, **34** (1): 39–58, 1991.

34. Y. Zhang et al., A novel timestamp ordering approach for co-existing traditional and cooperation transaction processing, to appear in*Int. J. Intell. and Cooperative Inf. Syst.*, an earlier version in *Proc. 3rd IFCIS Conf. Cooperative Information Systems*, New York, 1998.

35. Y. Zhang, Y. Kambayashi, X. Jia, Y. Yang, and C. Sun, On interactions between coexisting traditional and cooperative transactions, *Int. J. Coop. Inform. Syst.*, **8** (2,3): 87–109, 1999.

36. M. H. Dunham and A. Helal, Mobile computing and databases: Anything new?*SIGMOD Rec.*, **24** (4): 5–9, 1995.

37. E. Pitoura and G. Samaras, *Data Management for Mobile Computing*, Dordrecht, the Netherlands: Kluwer Academic Publishers, 1998.

38. D. Barbara, Mobile computing and databases – a survey, *IEEE Trans. Knowledge Data Eng.*, **11** (1): 108–117, 1999.

39. A. K. Elmagarmid, J. Jing, and T. Furukawa, Wireless client/server computing for personal information services and applications, *SIGMOD Rec.*, **24** (4): 16–21, 1995.

40. S. Madria et al., Data and transaction management in a mobile environment, in S. Upadhyaya, A. Chaudhury, K. Kwiat, and M. Weiser (eds.), *Mobile Computing Implementing Pervasive Information and Communications Technologies*, Dordrecht, the Netherlands Kluwer Academic Publishers, 2002, pp. 167–190.

41. M. H. Dunham, A. Hedal, and S. Balakrishnan, A mobile transaction model that captures both the data and movement behavior, *Mobile Networks Applicat.*, **2**: 149–162, 1997.

42. J. B. Lim and A. R. Hurson, Transaction processing in mobile, heterogeneous database systems, *IEEE Trans. Knowledge Data Eng.*, **14** (6): 1330–1346, 2002.

43. R. A. Dirckze and L. Gruenwald, A pre-serialization transaction management technique for mobile multidatabases, *Mobile Networks Applicat.*, **5**: 311–321, 2000.

44. S. Madria and B. Bhargava, A transaction model to improve data availability in mobile computing, *Distributed Parallel Databases*, **10**: 127–160, 2001.

45. E. Pitoura and B. Bhargava, Data consistency in intermittently connected distributed systems, *IEEE Trans. Knowledge Data Eng.*, **11** (6): 896–915, 1999.

46. S. Acharya et al., Broadcast disks: Data management for aymmetric communication environments, *ACM SIGMOD Record, Proc. 1995 ACM SIGMOD Int. Conf. Management of Data*, **24** (2): 199–210, 1995.

47. D. Barbara, Certification reports: Supporting transactions in wireless systems, *Proc. 17th Int. Conf. Distributed Computing Systems*, 1997, pp. 466–473.

48. E. Pitoura and P. K. Chrysanthis, Scalable processing of read-only transactions in broadcast push, *Proc. 19th IEEE Int. Conf. Distributed Computing Systems*, 1999, pp. 432–439.

49. V. C. S. Lee et al., On transaction processing with partial validation and timestamp ordering in mobile broadcast environments, *IEEE Trans. Comput.*, **51** (10): 1196–1211, 2002.

50. J. Shanmugasundaram et al., Efficient concurrency control for broadcast environments, *ACM SIGMOD Record, Proc. 1999 ACM SIGMOD Int. Conf. Management of Data*, **28** (2): 85–96, 1999.

51. E. Pitoura and P. K. Chrysanthis, Multiversion data broadcast, *IEEE Trans. Compu.*, **51** (10): 1196–1211, 2002.

52. K. Ramamritham and P. K. Chrysanthis, *Advances in Concurrency Control and Transaction Processing*, Los Alamitos, CA: IEEE Computer Society Press, 1997.

FURTHER READING

R. Alonso, H. Garcia-Molina, and K. Salem, Concurrency control and recovery for global procedures in federated database systems, *Q. Bull. Comput. Soc. IEEE Tech. Comm. Database Eng.*, **10** (3): 5–11, 1987.

P. A. Bernstein and N. Goodman, Concurrency control in distributed database systems, *Comput. Surv.*, **13** (2): 188–221, 1981.

J. Cao, Transaction management in multidatabase systems. Ph.D. thesis, Department of Mathematics and Computing, University of Southern Queensland, Australia, 1997.

U. Dayal, M. Hsu, and R. Latin, A transactional model for long running activities, *Proc. 17th Conf. Very Large Databases*, 1991, pp. 113–122.

C. A. Ellis, S. J. Gibbs, and G. L. Rein, Design and use of a group editor, in G. Cockton (ed.), *Enginering for Human Computer Interaction*, Amsterdam: North-Holland, 1990, pp. 13–25.

J. N. Gray, *Transaction Processing: Implementation Techniques*, San Mateo, CA: Morgan Kaufmann, 1994, pp. 207–217.

G. Kaiser and C. Pu, Dynamic restructuring of transactions, in A. Elmagarmid (ed.), *Transaction Models for Advanced Applications*, San Mateo, CA: Morgan Kaufmann, 1992.

M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

Y. Kambayashi and K. Yokota (eds.), *Cooperative Databases and Applications*, Singapore: World Scientific, 1997.

C. Mohan et al., ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging, *ACM Trans. Database Syst.*, **17** (1): 94–162, 1992.

C. Pu, G. Kaiser, and N. Huchinson, Split transactions for open-ended activities, *Proc. 14th Conf. Very Large Databases*, Los Angeles, CA, 1988, pp. 26–37.

T. Rodden, A survey of CSCW systems, *Interact. Comput. Interdisc. J. Hum.-Compu. Interac.*, **3** (3): 319–353, 1991.

Y. Zhang and Y. Yang, On operation synchronization in cooperative editing environments, in *IFIP Transactions A-54 on Business Process Re-engineering*, 1994, pp. 635–644.

Y. Zhang
Victoria University
Melbourne, Australia
X. Jia
V. C. S. Lee
City University of Hong Kong
Hong Kong

ACTIVE CONTOURS: SNAKES

The shape of a real-world object can be represented by its outline in the image plane. In computer vision, the outline of the object is referred to as the *object contour*. A fundamental approach to finding automatically the object contour is the “snakes framework,” which was introduced by the seminal work of Kass et al. in 1987 (1). For the last two decades, snakes have been used successfully in the context of facial animation, visual speech analysis, traffic monitoring, surveillance, medical imaging (tracking and segmentation of organs), and blue screening in Hollywood movies.

A snake is an elastic model of a continuous and flexible curve that is fitted on the boundary between the object and the rest of the image by analyzing the visual image content. The process of fitting iteratively an initial snake to the object, such that the snake encloses the object tightly, is called “snake evolution.” During its evolution, the snake imposes continuity and smoothness constraints on the evolved contour, which relax the requirement of a noise-free image. In addition to the continuity and smoothness constraints, snake have the capability to be attracted to certain shape configurations known a priori. The evolution of a snake from one configuration to another in consecutive frames of a video clip attributes a dynamic behavior to the contour and provides object-tracking capabilities. The snake performing object tracking is considered a *dynamic contour* moving from frame to frame.

THE SNAKE FORMULATION

The snake is composed of a set of control points marked in the spatial image coordinates (x,y). The control points initially can reside inside or outside the object region. From its initial configuration, the snake evolves by changing the positions of the control points while minimizing an associated cost (energy) function evaluated on the contour:

$$E_{\text{snake}} = \int_0^1 (\alpha E_{\text{image}} + \beta E_{\text{internal}} + \gamma E_{\text{external}})ds \qquad (1)$$

Where E denotes energy, s denotes the contour arc-length and α, β, and γ are the control parameters (1). The final position of the control points provides the final configuration of the snake which is obtained by the equilibrium of all three terms, E_{image}, E_{internal}, and E_{external}, in the snake energy [Equation (1)] (2). In particular, the image energy term, E_{image}, attracts the snake to a desired configuration by evaluating the visual features in the image. During its evolution, the internal energy term, E_{internal}, imposes a regularity constraint to enforce the contour continuity and smoothness. The last term in the energy function, E_{external}, accounts for the user-defined constraints. Traditionally, researchers define E_{external} in terms of a known set of shapes the object can have.

VISUAL CUES

The snake’s attraction to distinctive local features on the object boundary signifies the role of feature extraction in the snake framework. Traditionally, feature extraction is achieved by convolving an image with a mask. In its simplest form, the convolution mask H can be considered a small image, usually an $n \times n$ matrix, and the convolution operation $*$ between the image I and the mask H is performed by

$$I(x,y) * H = \sum_{i=1}^{n}\sum_{j=1}^{n} I(x + i/2, y + j/2)H(i,j) \qquad (2)$$

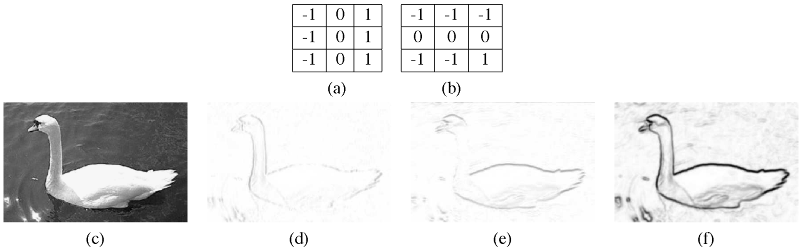
The convolution of the image with a filter generates a feature image in which the boundaries are expected to be highlighted while the other regions are suppressed. For instance, convolving an image shown in Fig. 1(c) using the vertical and horizontal edge filters shown in Fig. 1(a) and Fig. 1(b) produces the edge responses shown in Fig. 1(d) and Fig. 1(e). The gradient magnitude feature computed from these edge responses emphasizes the object boundary to which the snake will be attracted.

The convolution operation is a local operation, which does not guarantee the generation of expressive features. This operation can be exemplified as shown in Fig. 2(a) where the background clutter and the object texture generate ambiguous edges, causing the snake to get attracted to the wrong configurations. A solution to this problem is to use global features computed in the regions defined by the inside and the outside of the object contour (4). The similarity between the colors observed inside and outside the object is a common measure used by researchers (see Fig. 2(b) for the definition of the snake inside and outside). This similarity measure can be computed by means of the distance between the probability distribution functions (pdf) associated with the inside and outside regions. Based on this distance, the snake evolves by moving the control points inward or outward.

CURVE SMOOTHNESS

In the case when the object is not distinctive from its background or when the image contains noise, the snake may not converge to a final configuration, which represents the object shape. To overcome this problem, it is necessary to stabilize the contour evolution to keep the shape of the snake intact and to not resonate from one configuration to another. Stabilization of the contour is achieved by the internal energy term, E_{internal}, given in Equation (1).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.



-1	0	1
-1	0	1
-1	0	1

(a)

-1	-1	-1
0	0	0
-1	-1	1

(b)

Figure 1. The convolution mask to detect (a) the vertical edges and (b) the horizontal edges, (c) An input image. Resulting (d) vertical edges after convolving (a) with (c), and (e) horizontal edges after convolving (b) with (c). (f) The gradient magnitude image generated using (d) and (e) for highlighting the boundary of the object.

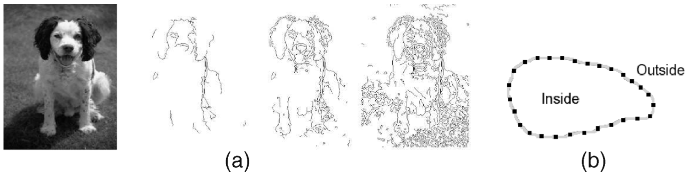


Figure 2. (a) The edges obtained by applying the Canny edge detector (3) with different thresholds. Note the ambiguity of the features that will guide the snake evolution. (b) The inside and outside regions defined by the snake.

This term includes a weighted combination of a membrane function and a thin plate function shown in Fig. 3:

$$E_{\text{internal}} = w_1 \underbrace{\Gamma(s)\left|\frac{\partial\Gamma(s)}{\partial s}\right|^2}_{\text{membrane}} + w_2 \underbrace{\Gamma(s)\left|\frac{\partial^2\Gamma(s)}{\partial s^2}\right|^2}_{\text{thin plate}} \tag{3}$$

where $\Gamma(s)$ denotes the curve and w_1 and w_2 are the weights. Practically, when $w_1 \gg w_2$, the curve is allowed to kink, whereas $w_1 \ll w_2$ forces the curve to bend slowly. A common practice is to use different weights for each control point, so that both w_1 and w_2 become functions of s. This approach allows parts of the snake to have corners and allows the others parts to be smooth.

SNAKE EVOLUTION

The motion of each control point, which is governed by Equation (1), evolves the underlying curve to new a con-

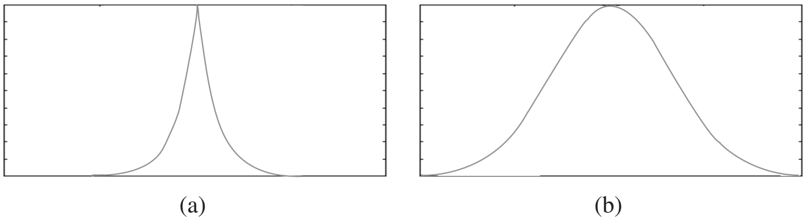


Figure 3. (a) The membrane function and (b) the thin plate function, which are a regularization filter of order 1 and 2, respectively.

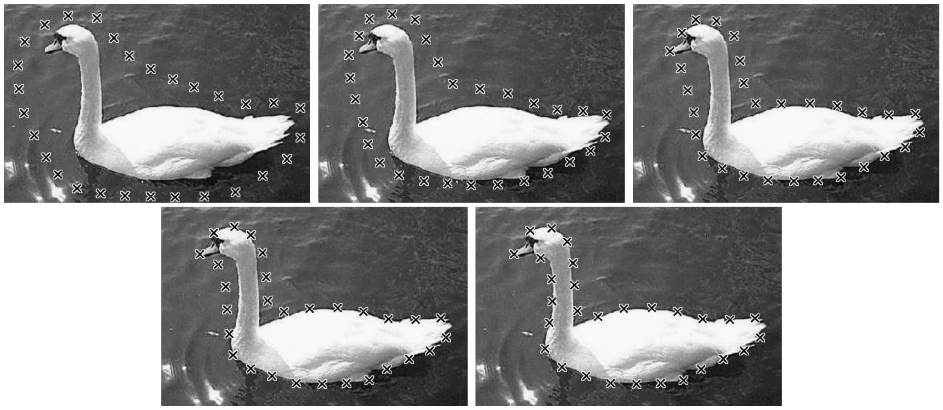


Figure 4. The evolution of an initial snake using the gradient magnitude image shown in Fig. 1(f) as its feature.

figuration. This process is shown in Fig. 4. Computing the motion of a control point s_i requires the evaluation of the first- and the second- order curve derivatives in a neighborhood $\Gamma(s_i)$. An intuitive approach to evaluate the curve derivatives is to use the finite difference approximation:

$$\frac{\partial\Gamma(s_i)}{\partial s} = \frac{\Gamma(s_i) - \Gamma(s_{i-1})}{d} \tag{4}$$

$$\frac{\partial^2\Gamma(s_i)}{\partial s^2} = \frac{\Gamma(s_{i+1}) - 2\Gamma(s_i) + \Gamma(s_{i-1})}{d^2} \tag{5}$$

The finite difference approximation, however, is not applicable in regions where two control points overlap, resulting in zero Euclidean distance between the neighboring control points: $d = 0$. Hence, a special handling of the displacement between the control points is required, so they do not overlap during their motion. Another approach to compute the derivatives is to fit a set of polynomial functions to neighboring control points and to compute the derivatives from these continuous functions. In the snake literature, the parametric spline curve is the most common polynomial approximation to define the contour from a set of control points. As shown in Fig. 5, the spline curve provides naturally a smooth and continuous approximation of the object contour. This property waives the requirement to include regularization terms in the energy function. Hence, the complexity of the snake energy formulation is simplified.

The complexity of the snake formulation can also be reduced by using a greedy algorithm (4,5). The greedy algorithms move the control points on an individual basis by finding a set of local solutions to the regular snake energy Equation (1). computing locally the energy at each control point requires analytical equations to evaluate the snake regularity and curvature (5). An alternative greedy formulation is to move each control point individually based on similarity in appearance between local regions defined inside and outside of the curve (4). Practically, if the appearance of the outside is similar to that of the inside, the control point is moved inside; if not, it is moved outside. In either approach, the assumption is that each local solution around a control point is correct and contributes to the global solution defined by the object boundary.

DISCUSSION

The snakes framework has been very useful to overcome the limitations of the segmentation and tracking methods for cases when the features generated from the image are not distinctive. In addition, its parametric form results in a compact curve representation that provides a simple technique to compute geometric features of the curve, such as the curvature, and the moments of the object region, such as the object area.

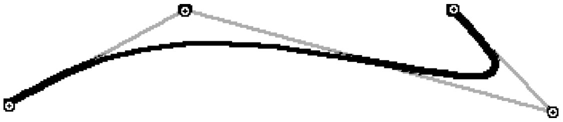


Figure 5. Spline function estimated from four control points. The gray lines denote the control polygons connecting the control points.

The algorithms developed for the snake framework perform near real time when the initial snake is placed close to the objects of interest. Their performance, however, degrades in the presence of background clutter. To overcome this limitation, researchers have proposed various shape models to be included in the external energy term.

One of the main concerns about the snakes is the amount of control points chosen initially to represent the object shape. Selection of a few control points may not define the object, whereas selecting too many control points may not converge to a solution. For instance, if the object circumference is 50 pixels and the snake is initialized with 100 control points, snake iterations will enter a resonant state caused by the regularity constraint, which prevents the control points from overlapping. A heuristic solution to this problem would be to add or remove control points when such cases are observed during the snake evolution.

Images composed of multiple objects require initialization of several independent snakes surrounding each object. Multiple snakes are required because both the finite difference approximation and the splines prevents the snake from changing its topology by splitting of one curve into two or merging of two curves into one. For topology changing curves, we refer the reader to the article on the "Level Set Methods."

BIBLIOGRAPHY

1. M. Kass, A. Witkin, and D. Terzopoulos, Snakes: active contour models, *Internation. Conf. of Comp. Vision*, London, UK, pp.259–268, 1987.
2. A. Blake and M. Isard, *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion.*, New York: Springer, 2000.
3. J. Canny, A computational approach to edge detection, *Putt. that. Machine Intell.*, **8** (6): 679–698, 1986.
4. R. Ronfard, Region based strategies for active contour models, *Iternat. J. Comp. Vision*, **13** (2): 229–251, 1994.
5. D. Williams and M. Shah, A fast algorithm for active contours and curvature estimation, *Comp. Vision Graphics Imag. Process*, **55** (1): 14–26, 1992.

ALPER YILMAZ
The Ohio State University
Columbus, Ohio

C

COLOR: COLOR MODELS

INTRODUCTION

With the widespread use of color-imaging and display systems, the ability to describe and specify color has become increasingly important. These models are used primarily to understand or characterize the behavior of color stimuli or abstract signals that are representative of color. In general, color models may be divided into four families as follows:

- Physics-based models,
- Models based on human perception,
- Models used for imaging devices, and
- Hybrid models that are used in imaging devices but are loosely based on human perception.

PHYSICS-BASED MODELS

Sources of light have a spectral power distribution that may be represented by $i(\lambda)$, where λ denotes wavelength (with units of say, nanometers). Light interacts with an inhomogeneous surface of an object (see Fig. 1) and undergoes two distinct processes, which are described by the dichromatic reflection model (1). The first process occurs at the interface between the medium in which the incident light is traveling and the surface of the object, which causes some reflection and refraction. The refracted light passes through the interface and interacts with the particles in the medium. It is then scattered repeatedly and absorbed selectively by the pigments or colorants in the medium and is in part transmitted through the transparent surface with its spectral composition altered by the colorants. This model pertains to dielectrics, not metals. The *dichromatic reflection model* states that each of the two components—the first denoted the "interface" reflection and the second denoted the "body" reflection—may be factored into the product of the spectral composition (c), depending on wavelength only and a geometric factor (m) that depends on the angular arrangement of the illumination and viewing geometry. This model may be represented as

$$\begin{aligned} i_r(\lambda, \theta) &= i_{\text{interface}}(\lambda, \theta) + i_{\text{body}}(\lambda, \theta) && (1) \\ &= m_{\text{interface}}(\theta) c_{\text{interface}}(\lambda) + m_{\text{body}}(\theta) c_{\text{body}}(\lambda) && (2) \end{aligned}$$

The wavelength-related terms depend in turn on both the spectral composition of the illuminant and on the object. This color formation model is used in machine vision-related works and has found wide acceptance.

In a somewhat simplified framework in which viewing geometry is not of much concern—especially when the surface is Lambertian (reflects light equally in all directions, independent of viewing angle)—the viewing geometry is constant. This model may then be restated as: The light that reaches a surface gets spectrally selectively absorbed/transmitted/reflected by the surface depending on the colorants that are in the body. The energy exiting the surface may now be represented by

$$i_r(\lambda) = i(\lambda) r(\lambda) \tag{3}$$

Here, $r(\lambda)$ denotes the spectral reflectance function of the surface (note that the interface and the body are treated as one). This simplistic model discounts viewing geometry, that is, the relative position of the source of light and its direction, the surface orientation, and also the position of the observer relative to each of these.

MODELS BASED ON HUMAN PERCEPTION

A copublished article titled "Color Perception" gives a brief description of the mechanisms of color perception. In that article, we introduced the following equation regarding formation of color stimuli in the retina-based cone functions, which is represented as functions of wavelength by $l(\lambda)$, $m(\lambda)$, and $s(\lambda)$: These symbols give the spectral sensitivities of the three different cones in the human retina, denoting long, medium, and short human cone sensors (i.e., roughly red, green, and blue). In the presence of a light source (illuminant) represented by $i(\lambda)$, the reflectance function of a surface (described by $r(\lambda)$ in [0..1]) is modified in a wavelength-selective fashion to create a stimulus $i_r(\lambda)$ to the eye given by Equation (3). Let us denote the cone functions as a vector given by

$$\mathbf{lms}(\lambda) = [l(\lambda), m(\lambda), s(\lambda)] \tag{4}$$

The signals measured in the cones of the eye are then a three-vector $\mathbf{c} = [c_l, c_m, c_s]$, given by:

$$\mathbf{c} = \int_{\lambda=380\,\text{nm}}^{\lambda=830\,\text{nm}} \mathbf{lms}(\lambda) i_r(\lambda) d\lambda \tag{5}$$

In the case of emissive sources of light, $i(\lambda)$ in Equation 3 is replaced by the spectral content of the light itself. In the case of transmission through translucent materials, $r(\lambda)$ is replaced by the transmission function with respect to wavelength. These equations are founded on Grassmann's laws (2), which state that a color (**C**) that is matched with a weighted combination of three independent primaries (**R, G, B**) will maintain the laws of linearity. In other words, consider two colors **C1** and **C2**, each created with two different weightings of the primaries, given by

$$\begin{aligned} \mathbf{C1} &= \alpha_1 \mathbf{R} + \alpha_2 \mathbf{G} + \alpha_3 \mathbf{B} \\ \mathbf{C2} &= \beta_1 \mathbf{R} + \beta_2 \mathbf{G} + \beta_3 \mathbf{B} \end{aligned} \tag{6}$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

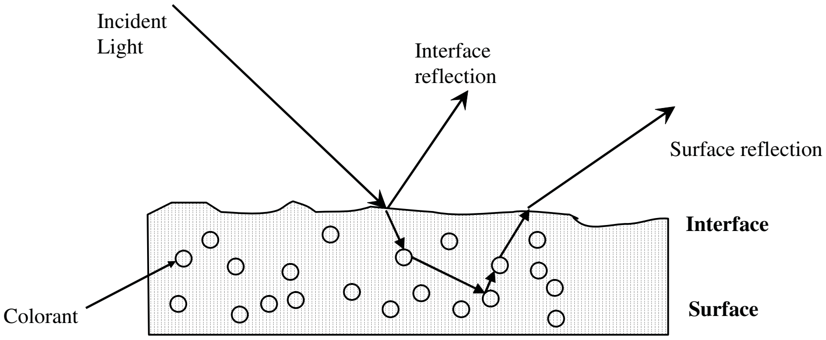


Figure 1. Dichromatic reflection model.

A combination of these two stimuli will result in a color given by

$$\mathbf{C1} + \mathbf{C2} = (\alpha_1 + \beta_1)\mathbf{R} + (\alpha_2 + \beta_2)\mathbf{G} + (\alpha_3 + \beta_3)\mathbf{B} \qquad (7)$$

Hence, with knowledge of the trichromacies of the three primaries at every wavelength, the net result of a linear combination of these primaries as they vary in their spectral content is given by Equation 2. (In computer computations as opposed to nature, integrals are actually replaced by summations based on samplings of continuous functions.) Of course, knowledge of the weights is needed and in the case of Equation 2, the various weights are given by the cone functions.

The ability to measure directly the cone functions of an observer is only a recent development. However, researchers had inferred the cone functions based on what are known as *color matching functions* and the International Commission on Illumination [Commission International de L'Eclairage, (CIE)] had standardized them long before the cone functions could be measured directly. The color-matching functions are determined by a process that involves an observer viewing a bipartite field, one half of which uses a reference color and another uses a color made by adjusting the strengths of three independent sources of illumination until a perfect match is made (illustrated in Reference 3). This model suggests that it is possible to plot the individual weights of the three primaries on a wavelength scale, which in turn gives us the weight our visual system applies to a given set of spectral stimuli to match a particular color. These weights, the color-matching functions, were determined by two independent researchers (John Guild and William D. Wright) in the 1920s. In a document published in 1931, the CIE combined these results to publish a set of RGB color-matching functions with RGB primaries standardized to those at 700 nm, 546.1 nm and 435.8 nm, respectively (4). These are shown in Fig. 2. These color-matching functions have negative excursions because some colors reside outside the triangle formed by these three primaries, and a negative excursion in the primaries' weights is the only way to represent a color outside a triangle. To address this, the CIE also published a set of color-matching-functions with non-negative values, which are known as $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$ (note that these are now imaginary, nonphysical primaries). In addition to being non-negative, the new-color matching functions (see Fig. 3) were created such that $\dot{y}(\lambda)$ matched the photopic luminance response that was standardized by the CIE earlier, in 1924 (5). These color-matching functions are to be used in the same manner as the cone primaries and the RGB primaries. The equations for

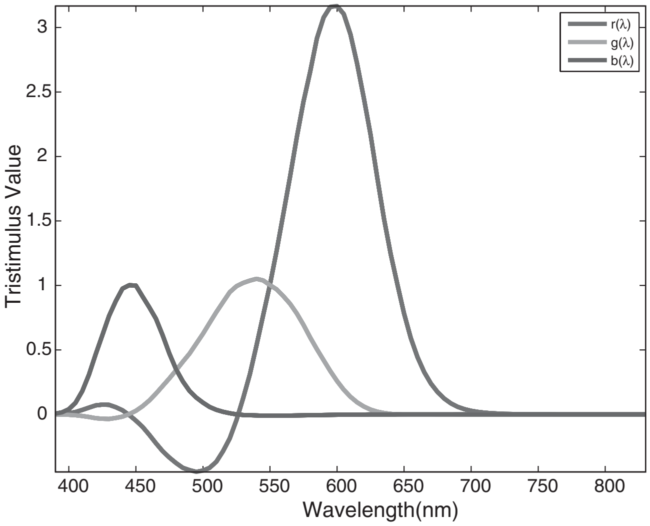


Figure 2. RGB color matching functions.

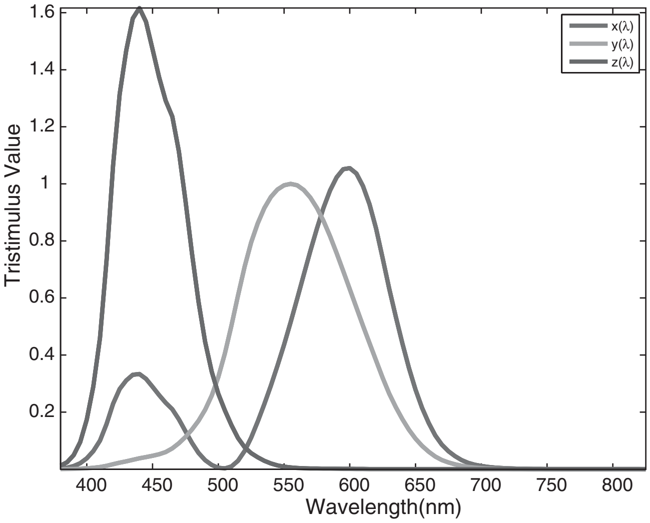


Figure 3. XYZ color matching functions with Judd-Vos modifications.

computing these special "tristimulus" values, in XYZ color space, are analogous to Equation 5 but with respect to the standardized color-matching functions as follows:

$$\begin{aligned} X &= k\int_{\lambda} \bar{x}(\lambda) i_r(\lambda) d\lambda \\ Y &= k\int_{\lambda} \bar{y}(\lambda) i_r(\lambda) d\lambda \\ Z &= k\int_{\lambda} \bar{z}(\lambda) i_r(\lambda) d\lambda \end{aligned} \tag{8}$$

In Equation 8, k denotes a normalization factor that is set to 683 lumens/Watt in the case of absolute colorimetry and to $100/\int_{\lambda} \bar{y}(\lambda) i(\lambda) d\lambda$ for relative colorimetry. In the case of relative colorimetry, this means a value of $Y = 100$ denotes the brightest color—the illuminant that reflects off a perfect-reflecting diffuser.

The first published set of color-matching functions by the CIE were originally empirically determined for a 2° field—the bipartite field used for matching subtended a 2° angle on the observers' retina. After the 1931 publication, W. S. Stiles and J. M. Burch conducted experiments (6) to measure color-matching functions for larger fields of view. This research was combined with the findings of Speranskaya (7) into the publication of a 10° observer in 1964 (8). The differences between these two standard observers is significant enough to warrant a clear specification of which observer color-matching functions are used in experimental work. More specifically, the 10° observer has noticeable shifts of the color-matching functions in the blue direction because the subtense of the stimulus encompasses a larger portion of the retina and hence, more S cones as well as increased macular pigment absorption.

In the CIE colorimetric system, an XYZ tristimulus value uniquely specifies a color. However, a convenient two-dimensional representation of the tristimulus values led to the projection of the tristimulus values by normalizing by the sum of the three values. These "chromaticity" values are given by

$$\begin{aligned} x &= \frac{X}{X+Y+Z} \\ y &= \frac{Y}{X+Y+Z} \end{aligned} \tag{9}$$

A third chromaticity value z can be analogously defined, but it is not independent because $z = 1 - x - y$. Specifying a color by its (x, y) chromaticity coordinates and its luminance Y also uniquely specifies a color, and it is often used to describe a color because the tristimulus values are straightforward to obtain from the (x, y, Y) values. The biggest advantage of the (x, y) chromaticity coordinates is that they specify a magnitude-independent hue and purity of a color. A *chromaticity diagram* (see Fig. 4) is typically used to specify a color using its chromaticity coordinates. Unique specification, however, requires the luminance to be specified as well. Figure 4 also shows the locus of illuminants standardized as daylight illuminants (per the CIE standard), specifically denoting the location D55 (standard mid-morning daylight), D65 (standard noon daylight), and also for reference, the location of illuminant A (tungsten lamps) and the equi-energy point, E ($x = y = 0.33$). Here, D65 stands for a standard daylight with a "correlated color temperature" of T = 6500 K, as the closest point on the locus of all Plankian lights, which are specified analytically as a function of T. In Fig. 4, the horseshoe-shaped locus denotes the locus of monochromatic stimuli visible to the standard 2° observer (the gamut of visible colors). Shorter wavelength stimuli (starting at 380 nm, which elicit a relatively strong blue response) reside in the lower left of this horseshoe shape, whereas the longer wavelengths (ending at 830 nm, which elicit a relatively strong red response) reside on the lower right, with the top of the horseshoe curve around 520 nm (eliciting a strong green response). The line that connects the blue and red corners is referred to as the *line of purples*. Colors on this line, although on the border of the gamut, have no counterpart in monochromatic light. The (x,y) chromaticity diagram is perceptually nonuniform: Unit vectors in the chromaticity space do not correspond to a unit change in perception even if the luminance is kept constant. In an attempt to improve the uniformity of the chromaticity diagram, in 1976 the CIE published a Uniform Chromaticity Scale (UCS) diagram that scaled and normalized the XYZ tristimulus values (9). This chromaticity diagram is denoted by u', v' axes that are related to the XYZ tristimulus values by the following equations:

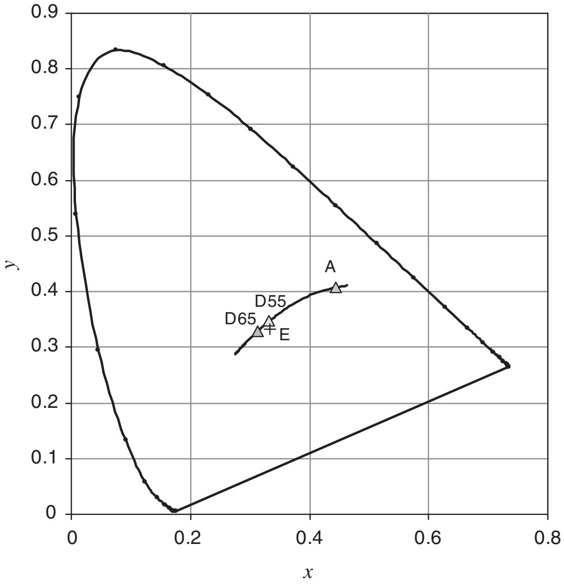


Figure 4. CIE xy chromaticity diagram that shows illuminants D55, D65, A, and E ('+', equi-energy illuminant).

$$\begin{aligned} u' &= \frac{4X}{X + 15Y + 3Z} \\ v' &= \frac{9Y}{X + 15Y + 3Z} \end{aligned} \tag{10}$$

Figure 5 shows the UCS along with the standard illuminant locations and the D-illuminant locus. As in Fig. 4, in this diagram, the horseshoe-shaped locus represents the gamut of visible colors.

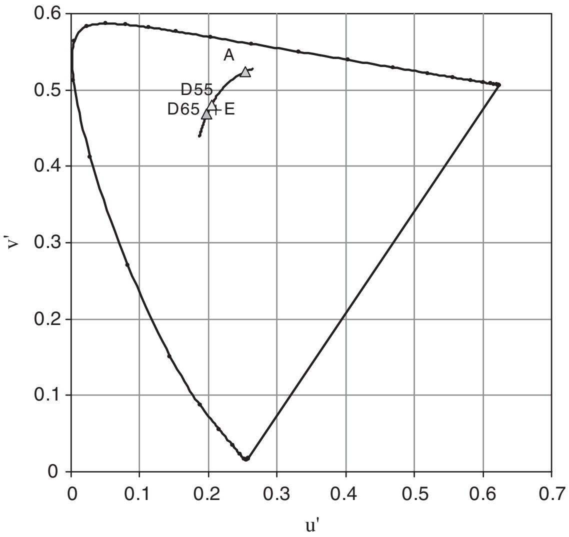


Figure 5. CIE UCS u′v′ chromaticity diagram showing illuminants D55, D65, A, and E (equi-energy illuminant).

The CIEXYZ color space, albeit powerful, does not represent colors in a perceptually uniform fashion; although the Y axis maps luminance, the X, Z axes have no perceptual correlates. To address these concerns and in turn enable a mechanism to incorporate the nonlinearity of the human visual system and furthermore provide a means of obtaining measures of differences between colors, in 1976 the CIE proposed the CIELAB and CIELUV color spaces, for subtractive and additive systems, respectively.

Lightness-Chroma-Hue Color Spaces

The CIEXYZ color space does not have perceptual correlates that would make it useful for common use. In an attempt to add perceptual behavior to color spaces, based on earlier works of many researchers, the CIE proposed a lightness scale along with two chromatic scales. In the CIELAB color space, the axes are denoted by L^* (Lightness), a^*(redness-greenness) and b*(yellowness-blueness). For a stimulus given by a tristimulus value of X, Y, and Z, the CIELAB coordinates are given by:

$$\begin{aligned} L^* &= 116\, f(Y/Y_n) - 16 \\ a^* &= 500[f(X/X_n) - f(Y/Y_n)] \\ b^* &= 200[f(Y/Y_n) - f(Z/Z_n)] \end{aligned}$$

where

$$f(t) = \begin{cases} t^{1/3}, & \text{for } t > 0.008856 \\ 7.787t + 16/116 & \text{otherwise} \end{cases} \tag{11}$$

In the above equations, the subscript n denotes the tristimulus values that correspond to the reference white—note that therefore the CIELAB color space is a relative color space. Given the CIELAB coordinates in a three–dimensional space, correlates of chroma and hue may be derived as follows:

$$C^*_{ab} = (a^{*2} + b^{*2})^{1/2} \tag{12}$$

$$h^*_{ab} = \tan^{-1}(b^*/a^*) \tag{13}$$

Under highly controlled viewing conditions, a CIELAB ΔE difference of 1 correlates with a single just noticeable difference in color. It is to be noted that the CIELAB color-difference measure was designed for color differences between uniform color patches in isolation. It has however been used for image differences measures as well. In complex imagery, difference up to 3 is not significant (10).

In a similar construct, the CIE also recommended a CIELUV color space based on the uniform chromaticity scale (UCS), which uses subtractive shift from the reference white instead of the normalization based on division that is used in the CIELAB space. The equations to transform a tristimulus value from $u\prime, v\prime$ coordinates to CIELUV are given by:

$$\begin{aligned} L^* &= 116\, f(Y/Y_n) - 16 \\ u^* &= 13\, L^*(u' - v'_u) \\ v^* &= 13\, L^*(v' - v'_n), \end{aligned}$$

where

$$f(t) = \begin{cases} t^{1/3}, & \text{for } t > 0.008856 \\ 7.787t + 16/116 & \text{otherwise} \end{cases} \tag{14}$$

The u', v' coordinates for a tristimulus value are computed using Equation 10. As in the CIELAB definitions, the subscript n denotes the u', v' coordinates of the reference white being used. For example, for viewing a computer screen, the XYZ for standard light D65 is used; for viewing hardcopy, D55 or D50 is used. The implications of the u* and v* axes are similar to those in CIELAB, which approximate redness-greenness and yellowness-blueness directions.

Based on these correlates, the CIE recommends that color-difference measures in the two uniform-perception spaces CIELAB and CIELUV be given by the Euclidean difference between the coordinates of two color samples as follows:

$$\begin{aligned} \Delta E^*_{ab} &= \left[(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2\right]^{1/2} \\ \Delta E^*_{uv} &= \left[(\Delta L^*)^2 + (\Delta u^*)^2 + (\Delta v^*)^2\right]^{1/2} \end{aligned} \tag{15}$$

where the differences are given between the corresponding color coordinates in the CIELAB and CIELUV spaces between the standard and the test samples. Many improvements to this basic color-difference measure have been proposed and adopted over the years, which involve scaling the lightness, chroma, and hue differences appropriately based on the application and the dataset of samples to which the color difference measure has been adapted or improved (11). Typically, color-difference thresholds are dependent on the application, and thresholds for perceptibility judgments are significantly lower than thresholds for acceptability judgments. Many other color-difference measures have been proposed, and more recently, the CIE DE2000 has been adopted as a measure of color difference, again for uniform color patches under highly controlled viewing conditions and is slowly gaining acceptance (11,12).

In the case of both these uniform color spaces, note that

- The color space is only a rough approximation of a color-appearance space [predictors of correlates to brightness and colorfulness attributes are better modeled with color appearance models specifically designed for this purpose (13)].
- The chroma axes do not correspond to the location of unique hues. This subject is entirely different subject and is explored by Kuehni in Reference 14.
- The color differences are not to be used for samples across different viewing conditions—reference viewing conditions need to be used.
- The lightness scale closely matches the Munsell Value scale, and the two chromatic axes are used for describing redness-greenness and yellowness-blueness perception. These color axes are clearly modeled after the human vision opponent color theory.
- These color spaces were designed for threshold color differences and their application to supra-threshold (larger than about 5 units of ΔE) color differences is to be handled with care (15).

These color spaces provide a powerful tool to model and quantify color stimuli and are used in color-difference modeling for color patches. They have more recently been used to describe color appearances (see Reference 13). Models for describing colors based on lightness, chroma, and hue are powerful in their abilities to enable communication of color stimuli as well.

MODELS USED IN IMAGING DEVICES

In linear-additive combinations of colors, a unit input of a color corresponds to a globally constant unit of the output signal, whereas in nonlinear combinations, a transfer function would determine the input–output relationship. This nonlinearity is often used in quantized systems with limited available bits and in signal compression systems to take advantage of the available output signal bit-depth by stretching small input codes over a larger range of output codes and compressing the larger input codes into a smaller output dynamic range. This model is referred to as *gamma encoding*. From an encoding perspective, in its simplest form the input–output relationship is typically given by a gain-offset-gamma model, which is given by the following:

$$y = \text{round}\left[(2^N - 1)(\alpha x + \beta)^\gamma\right] \tag{16}$$

where α denotes a scalar gain, N denotes the number of bits in a system, β is a scalar offset, γ denotes a power law with values larger than 1 (typically around 2.2), and x and y denote the normalized input and the output signals, respectively (16). In encoding systems, the three channels typically have the same parameters. Display systems based on cathode ray tubes (CRTs) have an inherent response that follows the inverse relationship—large steps in input signal at the low end of the input signal cause a small change in output, whereas at the upper end of the signal range, small steps caused large output changes. It so happens that gamma encoding (using a power-law of γ on the linear luminance input) the input prior to transmitting the data to a CRT display causes the display luminance to follow similar steps that result in a net unity transfer function. This model is also a useful means of encoding data to maximize bit-depth usage while reducing visibly apparent contouring on the output data and display (3,17). In the case of a quantized color space, for reasons of perceptual uniformity, it is preferable to establish a nonlinear relationship between color values and intensity or luminance.

RGB Color Model

The specification of a color in the RGB color space implies a linear (after gamma is removed) or nonlinear (gamma applied) combination of the red, green, and blue primaries in varying strengths. In RGB color spaces, the manner in which colors are reproduced varies from device to device. For example, a color specified as an RGB triplet is more than likely going to look different from one display device to another when the exact same RGB triplet is provided as input because of differences in the "color" of the primaries and gamma curves. This finding makes the RGB space a device-dependent color space. Specifying colors in device-dependent color spaces, although not preferred from a color-reproduction perspective, is often resorted to because of its ease in comprehension.

An RGB color model can represent any color within an RGB color cube, as shown in Fig. 6. This color model is most

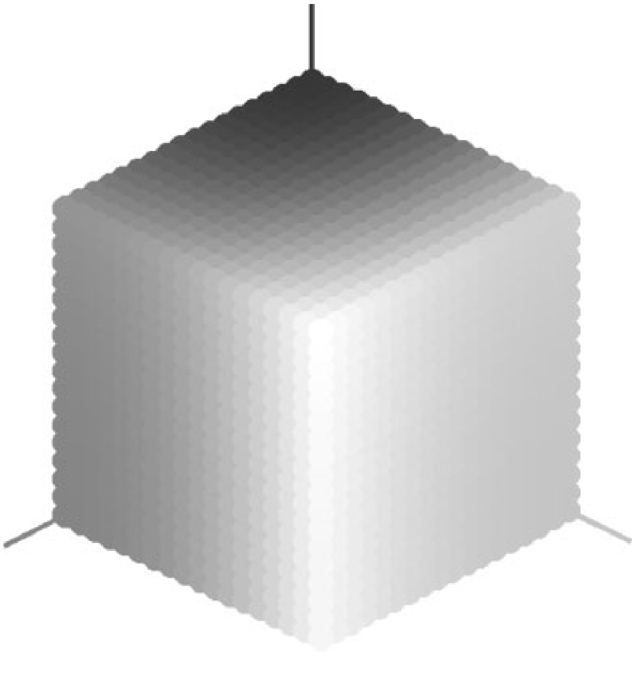


Figure 6. RGB color cube commonly used in display applications.

commonly used in display applications in which data is additive in nature. For example, a full-strength yellow color is specified by (1.0, 1.0, 0.0), denoting the use of the red and green primaries at full strength and the blue primary completely turned off. In an 8-bit system, this will correspond to a code value of (255,255,0). A three-primary display with three independent color primaries (typically denoted by their CIE x,y chromaticity values along with that of white) is specified by

$$\begin{bmatrix} x_R & x_G & x_B & x_W \\ y_R & y_G & y_B & y_W \end{bmatrix} \qquad (17)$$

From this primary set and the CIEXYZ tristimulus value of white computed as $[x_W/y_W, 1, (1-x_W-y_W)/y_W]$, a 3 × 3 system matrix **A** given by

$$\mathbf{A} = \begin{bmatrix} X_R & X_G & X_B \\ Y_R & Y_G & Y_B \\ Z_R & Z_G & Z_B \end{bmatrix} \qquad (18)$$

is computed. Using the XYZ tristimulus of white (with $\mathbf{c} = [1.0,1.0,1.0]$), **A** is computed using the following arithmetic:

$$\begin{bmatrix} x_R & x_G & x_B \\ y_R & y_G & y_B \\ z_R & z_G & z_B \end{bmatrix} \begin{bmatrix} k_R & 0 & 0 \\ 0 & k_G & 0 \\ 0 & 0 & k_B \end{bmatrix} \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} \qquad (19)$$

A is now given by

$$\mathbf{A} = \begin{bmatrix} x_R & x_G & x_B \\ y_R & y_G & y_B \\ z_R & z_G & z_B \end{bmatrix} \begin{bmatrix} k_R & 0 & 0 \\ 0 & k_G & 0 \\ 0 & 0 & k_B \end{bmatrix} \qquad (20)$$

Accordingly, the CIE tristimulus $\mathbf{t} = [X, Y, Z]$ that is generated from an RGB triple $\mathbf{c} = [R, G, B]$ is given by

$$\mathbf{t} = \mathbf{A}\mathbf{c}^t \qquad (21)$$

Consider for example, two displays (with the same input–output transfer functions, same absolute white luminance levels) with the following primary and white chromaticities:

$$\begin{bmatrix} 0.6400 & 0.3000 & 0.1500 & 0.3127 \\ 0.3300 & 0.6000 & 0.0600 & 0.3290 \end{bmatrix} \qquad (22)$$

and,

$$\begin{bmatrix} 0.6400 & 0.2100 & 0.1500 & 0.3127 \\ 0.3300 & 0.7100 & 0.0600 & 0.3290 \end{bmatrix} \qquad (23)$$

The system defined by Equation 22 has a system matrix given by

$$\mathbf{A}_1 = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \qquad (24)$$

and that defined by Equation 23 has a system matrix given by

$$\mathbf{A}_2 = \begin{bmatrix} 0.5767 & 0.1856 & 0.1882 \\ 0.2973 & 0.6274 & 0.0753 \\ 0.0270 & 0.0707 & 0.9913 \end{bmatrix} \qquad (25)$$

A yellow color code (1,1,0) in each case will correspond to a CIEXYZ of [0.7700,0.9278,0.1385] and [0.7622,0.9247, 0.0977], corresponding to a yellow CIE x, y coordinate of (0.4193,0.5053) and (0.4271,0.5181), respectively. The exact same representation of yellow results in different colors on screen—a consequence of using a device-dependent representation. Consequently, even though different display systems using an RGB color space define a cube in a Euclidean signal color space, their individual rendered color gamuts may very likely be significantly different. For a more rigorous comparison, the perceptual correlates (CIELAB, CIELUV or some such representation) of the colors need to also be considered.

The color-mixing matrix for additive colors mixing is shown in Table 1, which states for example that a cyan color would be created using maximum intensity of green and blue primaries and none of the red primary.

CMY/CMYK Color Model

Printers, on the other hand, create colors using inks that are deposited on paper, in which case the manner in which they create color is called subtractive color mixing. The inks selectively absorb wavelengths of incident light and reflect the remainder. As a beam of light passes through an absorbing medium, the amount of light absorbed is proportional to the intensity of the incident light times the coefficient of absorption (at a given wavelength). This model is often referred to as *Beer–Lambert–Bouguer law* and is given by the following:

$$A(\lambda) = \log_{10}\varepsilon(\lambda)c(\lambda)l(\lambda) \qquad (26)$$

Table 1. Color Mixing Matrix for Additive Primaries

	Color Displayed							
Primary Used	Red	Green	Blue	Cyan	Yellow	Magenta	White	Black
Red	1	0	0	0	1	1	1	0
Green	0	1	0	1	1	0	1	0
Blue	0	0	1	1	0	1	1	0

Table 2. Color-mixing Matrix for Subtractive Primaries

	Color Displayed							
Primary Used	Red	Green	Blue	Cyan	Yellow	Magenta	White	Black
Cyan	0	1	1	1	0	0	0	1
Yellow	1	1	0	0	1	0	0	1
Magenta	1	0	1	0	0	1	0	1

where $\varepsilon(\lambda)$ is denotes absorptivity, $c(\lambda)$ denotes the concentration, and $l(\lambda)$ denotes the path length for the beam of light. Stated differently, the higher the concentration or thickness or absorptivity of a certain absorptive material, the higher is absorption—the intensity of reflected or transmitted beam of light will be reduced (18). The simplest model for printer inks is called the *block-dye model*. In such a setup, different thicknesses of the three primary inks may be deposited on top of each other to result in a final color to the observer. The colorant amounts required to print a stimulus designated by RGB emissions is given by $Y = 1 - X$, where $Y \in \{C, M, Y\}$ and $X \in \{R, G, B\}$ all normalized to unity. Real primary inks however do not correspond to these ideal functions, and hence, more sophisticated models need to include not just the spectral absorptions/reflectances of the inks, but the density (or area) of the inks and the characteristics of the media (paper) involved. The *Kubelka-Munk equations* describe the absorption and scattering of light as it passes through layers of ink and the substrate (e.g., paper). Various extensions are used in practice that account for the shortcomings of the basic Kubelka-Munk analysis, which consider issues such as nonlinearities in ink deposition, interactions between inks, and so on (18,19). In subtractive color mixing, the primaries are typically cyan (C), yellow (Y), and magenta (M). The color-mixing matrix for subtractive color mixing is shown in Table 2.

Often, the amount of ink to be used is defined in terms of its optical density D which is given by:

$$D = -\log_{10} R \tag{27}$$

where R denotes a weighted measure of the reflectance of the ink being used. Printers commonly use halftone patterns (spatial patterns that are approximations of continuous tones of a certain color) to create shades between full-on and full-off. Consider the sample example of a black and white picture printed on the front page of a newspaper: This image may appear to be be a continuous-tone (smooth shades) grayscale picture, but a closer analysis reveals that the printer is simply a black-and-white printer with different-sized dots representing different gray shades as shown in Fig. 7. Various models are used to describe the area of a certain ink needed to create a mid-tone color (19). Digital printing uses patterns of binary dots to substitute for varying sized dots. The printing industry has long used dot-area models that describe the relationship between the area A of an ink on a medium to its reflectance function R. The simplest form is the *Murray-Davies equation* given by the following:

$$A = \frac{R_w - R}{R_w - R_s} \tag{28}$$

where R_w denotes the reflectance of the medium (unprinted), R_s denotes the reflectance of the solid ink, and R denotes the desired reflectance for which the dot-area is to be calculated. Equation 28 may also be written in terms of density using Equation 27. More complex models such as the *Yule–Nielsen model*, the *Clapper–Yule model*, the *Neugebauer model*, and its various extensions are used to better assess the amounts of the three inks (C,M,Y) needed and for the tonal response of the desired and actual ink-area, which may differ because of the characteristics of the medium being used (for example, absorption of liquid inks by paper) (20). Much like RGB color systems, in which the reference white made a difference in the appearance of a certain color, depending on the kind of paper and inks used for printing, the rendered color can be significantly different from one printer to another.

Most printers use a "K" channel, which denotes black ink, primarily because a black generated by mixing cyan, yellow, and magenta is not black enough in appearance.

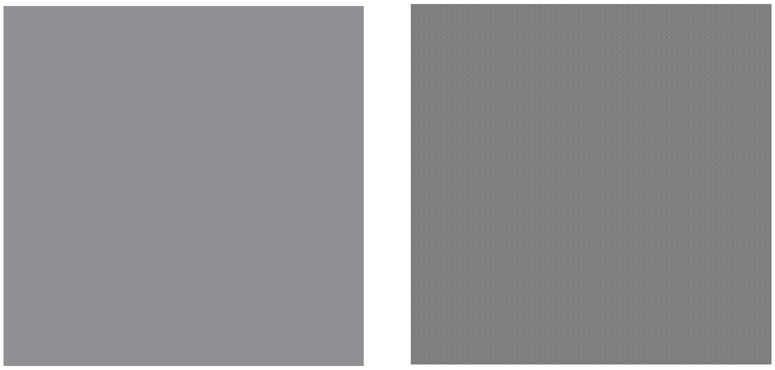


Figure 7. A continuous-tone representation of 50% gray and a halftone representation of the same code.

Additionally, to complicate matters, to print black, a printer would need to lay cyan, magenta, and yellow inks on top of each other, which makes ink drying a cause for concern and limits of ink absorption by the substrate (e.g., paper). Additionally, using one unit of black ink instead of one unit each of cyan, yellow and magenta inks can lead to significant cost savings.

HSL/HSV Color Model

To make the representation of colors intuitive, colors may be ordered along three independent dimensions that correspond to the perceptual correlates of lightness, hue, and chroma. In device-dependent color spaces, many variants of these perceptual correlates are commonly used: HSV is by far the most common. H stands for the perceptual correlate of hue; S stands for the saturation of a color, defined by the chroma of a color divided by its luminance (the more desaturated the color the closer it is to gray); and V stands for value (a perceptual correlate of lightness). This color model is commonly used in image-processing and editing software. However, the HSV color model has two visualization representations, one of which is a cylinder with black at the bottom and pure full-intensity colors on the top, and the other is representation by a cone, with black at the apex and white on the base. The equations used to convert RGB data into the HSV color space are given by the following:

$$V = \max \tag{29}$$

$$S = \begin{cases} 0 & \text{if } V = 0 \\ (V - \min)/V & \text{if } V > 0 \end{cases} \tag{30}$$

$$H = \begin{cases} 0 & \text{if } S = 0 \\ 60(G - B)/(\max - \min) & \text{if } (\max = R \text{ and } G \geq B) \\ 60(G - B)/(\max - \min) + 360 & \text{if } (\max = R \text{ and } G < B) \\ 60(B - R)/(\max - \min) + 120 & \text{if } \max = G \\ 60(R - G)/(\max - \min) + 120 & \text{if } \max = B \end{cases} \tag{31}$$

where max and min denote the maximum and minimum of the (*R,G,B*) triplet. These representations are shown in Fig. 8 on the left. From the figure, it is apparent that saturation is not dependent on the intensity of the signal. It is, however, often useful in image-processing applications to have an indicator of saturation given by a function of the intensity of the signal, which results in a conical-shaped HSV color space (Fig. 8). When the conical representation is preferred, S is given by (max—min)/(2^N−1) where 2^N−1 denotes the largest possible value for R, G, or B. Other variants of the HSV color space also exist and are used as an intuitive link to RGB color spaces (HSB, or HLS denoting various correlates of hue, saturation, and brightness/lightness).

HYBRID MODELS

Color models that are designed for imaging devices and communication needs are typically formulated such that colors are encoded/transmitted in the color space of a reference device. Colors spaces that fit such a description include the sRGB color space, the YCC, YUV, YIQ color transmission spaces, the SWOP CMYK color space, Adobe RGB (Adobe Systems, Inc., San Jose, CA), and ProPhoto RGB (Kodak, Rochester, NY), to list a few.

A popular mechanism to standardize colors across electronic devices, such as printers, monitors and the Internet, is the use of the *sRGB color space*. Originally, this was proposed by Hewlett-Packard and Microsoft and later standardized by the International Electrotechnical Commission under IEC 61966-2-1 (21). The sRGB standard has two primary parts, the viewing conditions and the necessary colorimetric definitions and transformations. The sRGB reference viewing environment corresponds to conditions typical of monitor display viewing conditions and thus may not be as well suited for print material, because of the various proprietary gamut-mapping algorithms in most printers that take advantage of each printer's color gamut. The colorimetric definitions provide the transforms necessary to convert between the sRGB color space and the CIEXYZ tristimulus color space as defined for a standard two-degree observer. More specifically, the standard is written for a standard reference monitor that has Rec. 709 primaries and a D65 white point. An overview of the technical advantages and challenges of the sRGB color space may be found in references 22 and 23. As was mentioned earlier, color spaces for video directly make use of the gamma-corrected signals, denoted as R′, G′, B′, from camcorders, without any attempt to correlate to the linear signals used in color science, such as those in Equation 5. For still imaging as well as video, this problem can be

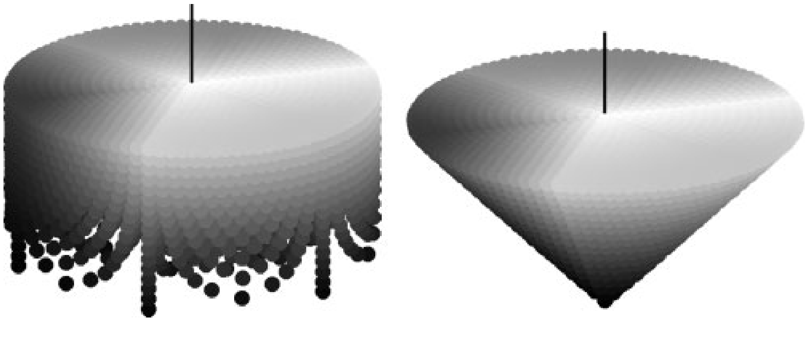


Figure 8. The HSV color model represented as a cylinder and a cone.

mitigated by the use of the transform built in to the sRGB standard, which includes a function for transforming from nonlinear signals I' to linear ones. On a scale of 0.0 to 1.0, for each of $I = R, G, B$, we apply a function

$$I = \begin{cases} I'/12.92, & \text{if } I' < 0.04045; \\ ((I' + 0.055)/1.055)^{2.4} & \text{otherwise} \end{cases} \tag{32}$$

In the video industry, a common mode of communication is the YCbCr color space (YPbPr in the analog domain) that converts RGB signal information into an opponent luma-chroma color space. A nonlinear transfer function is applied to linear-light *R, G, B* values and a weighted sum of the resulting *R′,G′,B′* values is used in the Y, Cb, and Cr signals. In the television space, these signals have dynamic ranges (on an 8-bit scale) of 16–235 for the luma signal and 16–240 in the Cb and Cr signals. This range is to allow for signal noise and potential signal processing noise, giving some head room and foot room. The weights are different depending on the color space that the data is being created for. For example, encoding $R^{'}, G^{'}, B^{'}$ signals with a 16–235 dynamic range into a color space defined by the NTSC primaries (often referred to as ITU-R BT.601), is given by the following:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{33}$$

whereas when using HDTV (referred to as ITU-R BT.709) primaries, is given by the following:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.213 & 0.715 & 0.072 \\ -0.117 & -0.394 & 0.511 \\ 0.511 & -0.464 & -0.047 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{34}$$

Keith Jack's book (24) is a useful reference for more details on the values of these coefficients for computer-systems that have *R′, G′, B′* data in the range of 0–255 and for details and considerations on color conversion issues. The *Y* channel typically contains most information in the image, as defined by spatial frequencies, and is hence sampled at much higher rates than the chroma signals. This information greatly helps in the ability of the transmission system to compress luma and chroma data with low overheads when compared with luma-only systems. To aid compression formats, color images in the JPEG and JPEG2000 file formats also convert the *R′, G′, B′* information into the YCbCr color space prior to compression.

In the printing industry, a commonly specified color space is the SWOP (Specifications for Web Offset Publications) CMYK color space. The SWOP CMYK (25) is a proofing specification that has a well-established relationship between the CMYK input to a standard printer and its CIELAB values (an approximation of the perceptual coordinates of a color) and for a standardized dataset. Specifying images in the SWOP CMYK color space allows the printing house and the content creator to preview images on a common baseline prior to printing. Most image-editing software available nowadays allows the user to preview images in the SWOP CMYK color space.

Depending on the application, color models have their individual uses. Device-independent color models like the CIEXYZ, CIELAB, CIELUV, and their other derivatives are used most often to communicate color either between devices or between different color processing teams across the world. The International Color Consortium (ICC) has been extremely successful in standardizing device-independent color spaces between displays, printers, and capture devices (26,27). The color profiles that are stored and communicated in ICC profiles use an intermediate profile connection space (PCS) like CIEXYZ or CIELAB. ICC profiles also store color transformation profiles to and from different color devices (say, from an input device such as a scanner to CIEXYZ, and from CIEXYZ to an output device such as a printer). For example, an sRGB ICC profile incorporates the color space transform from sRGB to the PCS, and a SWOP CMYK ICC profile would incorporate the color space transform from the PCS to the CMYK output color space for a printer. Furthermore, depending on the rendering intent (how the colors need to be represented on the output device), different transformations may be specified in the ICC profile.

CONCLUSIONS

Typically, devices that use additive color mixing are those used for display and use red, green, and blue primaries, although recently multiprimary displays (more than three) are also becoming available (28,29). The multiprimary displays allow for an improved color rendition with regard to natural colors. Recently, multiprimary acquisition systems (as opposed to three-primary acquisition systems) are gaining importance for archiving applications (30,31). The Digital Cinema Initiative (DCI), which standardizes the various aspects of digital cinema, has specified the data format for communication of digital film content to be in the CIEXYZ format, making the transmission of the information completely device independent (32). As far as subtractive color mixing devices are concerned, printers with more than three or even four inks have been available for some time now. It is fairly common for printer manufacturers to use six or seven inks—two types of cyans, magentas, and yellows, and of course a black ink. The two types of color inks allow for an increased range of ink densities to be deposited on the print media. Spectrally accurate printing is a field that is receiving attention as well and is a field of rapid advancements (30). Models of color are also becoming increasingly complex, which accounts for viewing conditions and the appearance of colors in spatial contexts of complex images as opposed to the relatively simplified constructs used in this article. All these examples are indicators that the study of color models is still a very active field.

The interested reader is referred to more detailed discussions of this subject such as the comprehensive books by Hunt (17), Berns (16), Kuehni (33), Fairchild (13), Sharma (31), and Green and MacDonald (19).

BIBLIOGRAPHY

1. G. Healey, S. Shafer, and L. Wolff, eds., *Physics-Based Vision: Principles and Practice*, vol 2. Boston, MA: Jones Bartlett Publishers Inc., 1992.
2. G. Wyszecki and W. S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*, 2nd ed. New York: Wiley Interscience, 2000.
3. Z.-N. Li and M. Drew, *Fundamentals of Multimedia*. Englewood Cliffs, NJ: Prentice Hall, 2004.
4. Commission Internationale de l'Eclairage, *Proceedings International Congress on Illumination*. Cambridge, UK: Cambridge University Press, 1931.
5. Commission Internationale de l'Eclairage, *The Basis of Physical Photometry, CIE Proceedings 1924*. Cambridge, UK: Cambridge University Press, 1926.
6. W. S. Stiles and J. M. Burch, NPL colour-matching investigation: Final report (1958). *Optica Acta*, **6**: 1–26, 1959.
7. N. I. Speranskaya, Determination of spectrum color co-ordinates for twenty-seven normal observers, *Optics Spectrosc.*, **7**: 424–428, 1959.
8. Commission Internationale de l'Eclairage, *CIE Proceedings (1964) Vienna Session, Committee Report E-1.4.1.*Paris, France: Bureau Central de la CIE, 1964.
9. Commission Internationale de l'Eclairage, *CIE Publication 15.2, Colorimetry*. Vienna, Austria: Central Bureau CIE, 1986.
10. M. Stokes, M. Fairchild, and R. Berns, Precision requirements for digital color reproduction. *ACM Trans. Graph.*, **11** (4): 406–422, 1992.
11. Commission Internationale de l'Eclairage, *Colorimetry, 3rd Edition, Publication CIE 15:2004*. Vienna, Austria: Bureau CIE, 2004.
12. R. G. Kuehni, CIEDE2000, milestone or a final answer? *Color Res. Applicat.*, **27** (2): 126–127, 2002.
13. M. D. Fairchild, *Color Appearance Models*. 2nd ed.New York: John Wiley and Sons, 2005.
14. R. G. Kuehni, Variability in unique hue selection: a surprising phenomenon. *Color Res. Applicat.*, **29** (2): 158–162, 2004.
15. R. G. Kuehni, *Color: An Introduction to Practice and Principles*, 2nd ed. New York: Wiley Interscience, 2004.
16. R. Berns, *Billmeyer and Saltzman's Principles of Color Technology*, 3rd ed. New York: Wiley Interscience, 2000.
17. R. W. G. Hunt, *The Reproduction of Color,* 6th ed. New York: Wiley, 2004.
18. R. McDonald, *Colour Physics for Industry*, 2nd ed. Bradford, UK: Society of Dyers and Colourists, 1997.
19. P. Green and L. MacDonald, ed., *Colour Engineering: Achieving Device Independent Colour*. New York: John Wiley & Sons, 2002.
20. P. Emmel and R. Hersch, A unified model for color prediction of halftoned prints, *J. Imag. Sci. Technol.*, **44**: 351–359, 2000.
21. International Electrotechnical Commission, *IEC 61966-2-1: Multimedia Systems and Equipment - Colour Measurement and Management - Part 2-1: Colour Management -Default RGB Colour Space - sRGB*. Geneva, Switzerland: IEC, 1999.
22. M. Stokes, M. Anderson, S. Chandrasekar, and R. Motta, A standard default color space for the Internet: sRGB. Available: http://www.color.org/sRGB.html, 1996.
23. Microsoft Corporation. Colorspace interchange using srgb. Available: http://www.microsoft.com/whdc/device/display/color/sRGB.mspx, 2001.
24. K. Jack, *Video Demystified - A Handbook for the Digital Engineer*, 3rd ed. Eagle Rock, VA: LLH Technology Publishing 2001.
25. CGATS TR 001, *Graphic Technology - Color Characterization Data for Type 1 Printing.*Washington, DC: American National Standards Institute, 1995.
26. International Organization for Standardization, *ISO 15076-1, Image technology colour management - Architecture, Profile Format and Data Structure - Part 1: Based on ICC 1:2004-10*. 2005.
27. International Color Consortium: Available: http://www.color.org.
28. D.-W. Kang, Y.-H. Cho, Y.-T. Kim, W.-H. Choe, and Y.-H. Ha, Multiprimary decomposition method based on a three-dimensional look-up table in linearized Lab space for reproduction of smooth tonal change, *J. Imag. Sci. Technol.*, **50** (4): 357–367, 2006.
29. M. Brill and J. Larimer, Avoiding on-screen metamerism in N-primary displays, *J. Soc. Informat. Disp.*, **13**: 509–516, 2005.
30. Art Spectral Imaging: Available: http://www.art-si.org/.
31. CIE Division 8: Image Technology: Multispectral Imaging. Available: http://www.colour.org/tc8-07/.
32. DCI System Requirements and Specifications for Digital Cinema v 1.1. Available: http://www.dcimovies.com/. Hollywood, CA: Digital Cinema Initiatives, LLC.
33. R. G. Kuehni, *Color Space and Its Divisions: Color Order from Antiquity to the Present*. New York: Wiley-Interscience, 2003.
34. G. Sharma, ed., *Digital Color Imaging Handbook*. Boca Raton, FL: CRC Press, 2003.

Rajeev Ramanath
Texas Instruments Incorporated
Plano, Texas
Mark S. Drew
Simon Fraser University
Vancouver, Vancouver, BC
Canada

COLOR PERCEPTION

INTRODUCTION

Color as a human experience is an outcome of three contributors: light, the human eye, and the neural pathways of the human brain. Factors such as the medium through which the light is traveling, the composition of the light itself, and anomalies in the human eye/brain systems are important contributors. The human visual system, which includes the optical neural pathways and the brain, responds to an extremely limited part of the electromagnetic (EM) spectrum, approximately 380 nm to 830 nm but concentrated almost entirely on 400 nm to 700 nm. We are blind, basically, to the rest of the EM spectrum, in terms of vision. For normal observers, this wavelength range roughly corresponds to colors ranging from blue to red (as shown in Fig. 1). The red end of the spectrum is associated with long wavelengths (toward 700 nm) and the blue end with short wavelengths (400 nm).

COLOR VISION

The structure in the eye that enables color vision is the retina, which contains the necessary color sensors. Light passes through the cornea, lens, and iris; the functionality of these is roughly comparable with the similar parts of most common cameras. The pupillary opening functions in a fashion similar to the aperture in a camera and results in the formation of an upside-down image of the outside world on the back face of the eye, the retina—a dense collection of photoreceptors. Normal human color vision is enabled by four different photoreceptors in the retina. They are called the rods and the L, M, and S cones (for long, medium, and short wavelength sensitive); each has a different spectral sensitivity within the range of approximately 400 nm to 700 nm (1). Figure 2 shows the (normalized) spectral sensitivities of the cones. Note that color as we know it is specifically a human experience and that a different species of animals respond differently to spectral stimuli. In other words, a bee would see the same spectral stimulus dramatically differently than a human would. In fact, bees are known to have their spectral sensitivities shifted toward the lower wavelengths, which gives them the ability to "see" ultraviolet light. The following salient points must be considered for a holistic understanding of human color perception:

1. The rods are activated for vision at low luminance levels (about 0.1 lux) at significantly lower spatial resolution than the cones. This kind of vision is called *scotopic vision*. The corresponding spectral sensitivity function is shown in Fig. 3. At these luminances, normal humans do not have any perception of color. This lack is demonstrated easily by trying to look at a colored painting at low luminance levels. Moreover, at these luminance levels, our visual acuity is extremely poor. This specific property of the visual system is a function of the low spatial density of the rod photoreceptors in the *foveal* region of the retina.
2. The cones are activated only at significantly higher luminance levels (about 10 lux and higher), at which time, the rods are considered to be bleached. This type of vision is referred to as *photopic vision*. The corresponding sensitivity function that corresponds to luminance sensitivities is shown in Fig. 3. The green curve is called the *luminous efficiency curve*. In this article we will consider photopic vision only. Interestingly, the retinal density of the three types of cones is not uniform across the retina; the S cones are more numerous than the L or M cones. The human retina has an L, M, S cone proportion as high as 40:20:1, respectively, although some estimates (2,3) put them at 12:6:1. This proportion is used accordingly in combining the cone responses to create the luminous efficiency curve. However, the impact of these proportions to visual experiences is not considered a significant factor and is under investigation (4). What is referred to as *mesopic vision* occurs at mid-luminance levels, when the rods and cones are active simultaneously.
3. The pupillary opening, along with independently scalable gains on the three cone output, permits operation over a wide range of illuminant variations, both in relative spectral content and in magnitude.
4. Color stimuli are different from color experiences. For the purposes of computational color science, the differences between color measurements and color experiences sometimes may not be considered, but often the spatial and temporal relations among stimuli need to be taken into account.

L, M, and S cone functions may be represented as a function of wavelength as $l(\lambda)$, $m(\lambda)$, and $s(\lambda)$. In the presence of a light source (illuminant), represented by $i(\lambda)$, the reflectance function of an arbitrary surface [described by $r(\lambda)$] is modified in a wavelength-selective fashion to create a stimulus $i_r(\lambda)$ to the eye given by

$$i_r(\lambda) = i(\lambda)r(\lambda) \tag{1}$$

Let us denote the cone functions as a vector given by

$$\mathrm{lms}(\lambda) = [l(\lambda), m(\lambda), s(\lambda)] \tag{2}$$

We denote the signals measured in the cones of the eye by $\mathrm{c} = [c_l, c_m, c_s]$, given by

$$\mathrm{c} = \int_{\lambda = 380\ \mathrm{nm}}^{\lambda = 830\ \mathrm{nm}} \mathrm{lms}(\lambda) i_r(\lambda) d\lambda \tag{3}$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

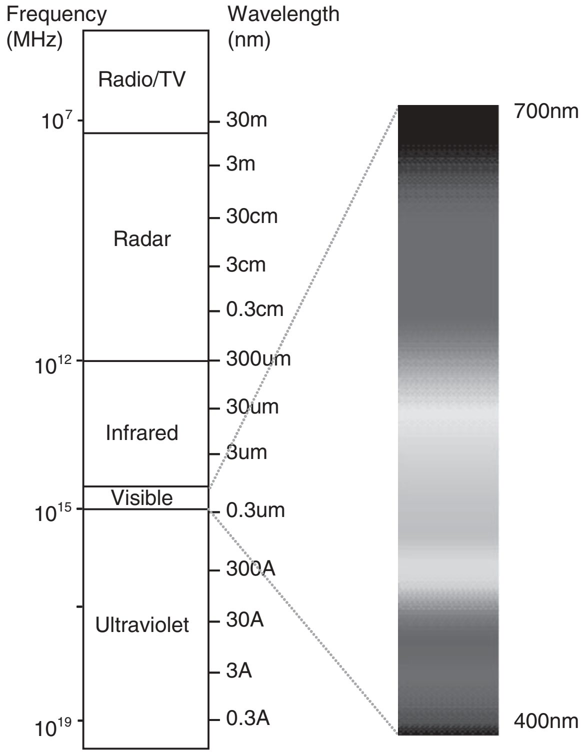


Figure 1. Electromagnetic spectrum, showing the limited range of human vision and color perception.

On an aside, although it is shown here that a color stimulus is formed by the process of reflection of the spectral energy $i(\lambda)$ of an illuminant off a surface $r(\lambda)$, it is only one manifestation of the cause of a color stimulus. The source for all colors at an elemental level may be grouped into 15 basic causes in five groups: (1) vibrations, rotations, and excitations, as in flames, neon lights, and so on; (2) ligand-field effects in transition metal compounds like turquoise, including impurities, as in rubies or emeralds; (3) molecular orbitals of organic compounds like chlorophyll and charge-transfer compounds like sapphire; (4) energy-bands in brass, gold, diamonds, and so on; and (5) geometric and physical optical effects like interference, diffraction, scattering, refraction, and so forth. The Nassau book on this topic is a thorough reference (5). In the case of emissive sources of stimuli (e.g. traffic lights, or television sets), Equation (3) is rewritten as

$$c = \int_{\lambda=380\,\mathrm{nm}}^{\lambda=830\,\mathrm{nm}} \mathrm{lms}(\lambda)e(\lambda)d\lambda \qquad (4)$$

where $e(\lambda)$ denotes the spectral stimulus that excites the cones.

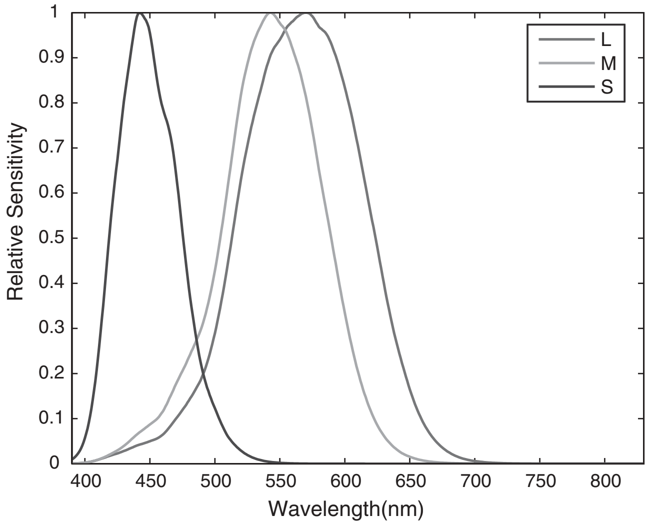


Figure 2. Cone sensitivity functions of normal human observers.

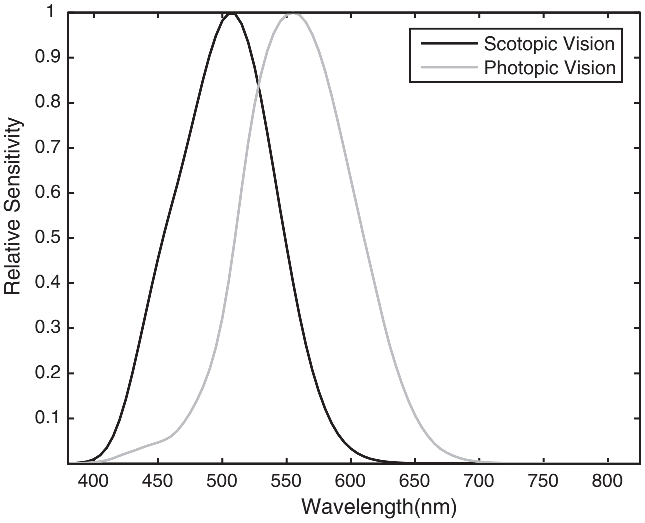


Figure 3. Sensitivity functions for photopic and scotopic vision in normal human observers.

A glance at the L, M, and S cone functions in Fig. 2 clearly highlights that the three measurements c, resulting from stimulating these cones, is not going to reside in an orthogonal three-dimensional color space—there will be correlation among L, M, and S. As a result of psychophysical testing, it is understood that human color vision operates on the basis of opponent color theory, which was first proposed by Ewald Hering in the latter half of the nineteenth century (2,6,7). Hering used a simple experiment to provide the primary proof of opponent color processes in the human visual system. An older hypothesis about the human visual system (works of Hermann von Helmholtz and James Maxwell in the mid-nineteenth century) suggested that the human visual system perceives colors in three independent dimensions (each corresponding to the three known color-sensitive pigments in the eye, roughly approximated by red, green, and blue axes). Although conceptually correct, this hypothesis could not explain some of the effects (unique hues and afterimages) that Hering observed. In a series of published works, Hering suggested that the visual system does not see colors as a combination of red and green (a reddish-green color). In fact, a combination of a red stimulus with a green stimulus produces no hue sensation at all. He suggested also that the

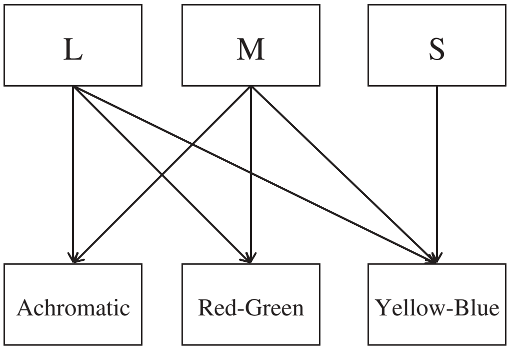


Figure 4. Weighted linear combinations of the L, M, S cone stimulations result in opponent color functions and an achromatic signal.

human visual system has two different chromatic dimensions (one corresponding to an orthogonal orientation of a red–green axis and another with a yellow–blue axis), not three. These concepts have been validated by many researchers in the decades since and form the framework of modern color theories. Similarly, staring at the bright set of headlights of an approaching car leaves a black or dark image after the car passes by, which illustrates that humans also see colors along a luminance dimension (absence or presence of white.) This finding has formed the backbone of modern color theory. Opponent color theory suggests that at least at the first stage, human color vision is based on simple linear operations on the signals measured by the L, M, and S cones. In other words, from the L, M, and S cone stimulations c, three resulting signals are computed that perform the task of reducing the interdependence between the measurements, somewhat orthogonalizing the space and hence reducing the amount of information transmitted through the neural system from the eye to the brain (see Fig. 4). This functionality is enabled by the extensive neural system in the retina of the eye. The opponent colors result in opponent cone functions (plotted in Fig. 5), which clearly suggests a fundamental conclusion of modern human color perception research: The three independent axes are luminance, redness–greenness, and yellowness–blueness (8). In other words, we have three sets of opposing color perception: black and white, red and green, and yellow and blue. In the figure, the red–green process appears as it does because colors on the ends of the spectrum appear similar (deep red is similar to purple)—the hue wraps around and often is portrayed in a circle.

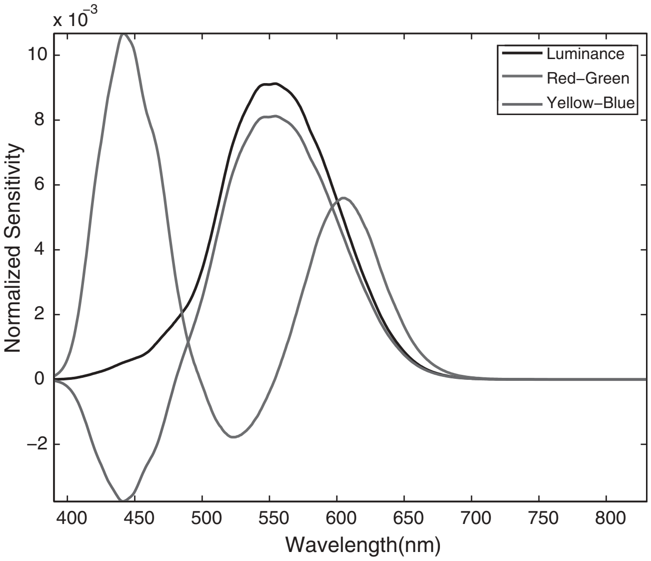


Figure 5. Normalized functions showing resultant red–green and yellow–blue sensitivities, along with the luminance channel.

Not surprisingly, much of the field of color science has been involved in trying to determine relationships between stimuli entering the eye and overall color experiences. This determination requires the ability to isolate color stimuli not just from their spatial relationships but also from the temporal relationships involved. Additionally, it requires a clear understanding of perceptual color appearance phenomena. As data have become available via a variety of experiments, the linear relationships between cone signals and color specifications has needed to be revised into a complex set of nonlinear relationships. Note that most color appearance phenomena have been developed for simplistic viewing fields, whereas all of our color experiences are based on complex images that involve complex illumination settings. It is instructive, nonetheless, to visit some common local color appearance descriptors that do not take into account spatial relationships such as the surroundings around a viewed scene. The terminology used below is used commonly in the color appearance work published by the Commission Internationale del' Eclairage in its standard documents (e.g. See Ref. 9) and in some popular textbooks on this subject (2,10). Groups involved in color ordering work and computational color technology, however, may define these terms differently based on their specific needs.

Hue

Hue is defined as the property of a color that describes it as a red, green, yellow, or blue or a combination of these unique hues. By definition, grays are not associated with any hue. A hue scale is defined typically as an angle. Figure 6 has magenta/violet hues at one end and reds at the other.

Brightness

The property of a color that makes it appear to emit more or less light.

Lightness

The property of a color that describes its brightness relative to that of the brightest white object in the visual field. A typical lightness scale is shown in Fig. 7.

Figure 6. A typical hue scale.

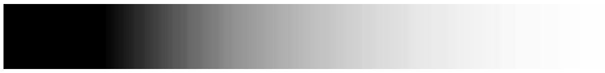


Figure 7. A typical lightness scale, with black at one end and the brightest possible white at the other.

Colorfulness

The property of a color that makes it appear more or less chromatic.

Chroma

The property of a color that describes its colorfulness relative to the brightness of the brightest white object in the visual field. In general, the relationship that exists between brightness and lightness is comparable with the relationship between colorfulness and chroma. Figure 8 shows a chroma scale for a hue of red and yellow. Note that in this example the chroma of yellows extends much farther than that of reds, as yellows appear much brighter than reds in nature and in most display systems.

Saturation

The property of a color that describes its colorfulness in proportion to its brightness. A typical saturation scale (shown here for a red and yellow hue) is displayed in Fig. 9. Note that by definition, saturation is normalized and hence unlike chroma; the same scale exists for both reds and yellows (and for other hues as well).

To aid in understanding these properties, Fig. 10 shows the locus of lines with constant hue, saturation, lightness, and chroma if we fix the brightness of white. By definition, saturation and hue are independent of the lightness of white.

Related and Unrelated Colors

In its simplest form, a color can be assumed to be independent of everything else in the viewing field. Consider, for illustrative purposes, a small patch of a color displayed in a dark room on a monitor with black background. This color is observed devoid of any relationships. This setup is typically the only one where colors are unrelated and are associated with attributes like brightness, hue, and saturation. Related colors, on the other hand, are observed in relationship with their surrounding and nearby colors. A simple example involves creating an image with a patch of brown color on a background with increasing white brightness, from black to white. The brown color is observed as a bright yellow color when on a black background but as a muddy brown on the brightest white background, which illustrates its relationship to the background (for neutral background colors). In practice, related colors are of great importance and are associated with perceptual attributes such as hue, lightness, and chroma, which are attributes that require relationships with the brightness of white. To specify a color completely, we need to define its brightness, lightness, colorfulness, chroma, and hue.

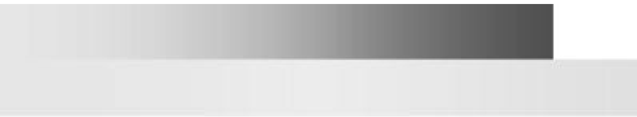


Figure 8. A typical chroma scale for red and yellow, starting with zero chroma on left and moving to maximum chroma on the right.

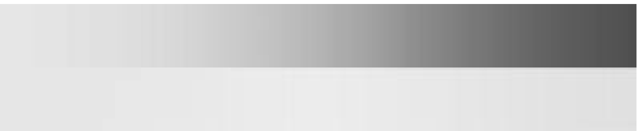


Figure 9. A typical saturation scale for red, starting with zero saturation on the left and moving to a saturation of 1 for pure color on the right.

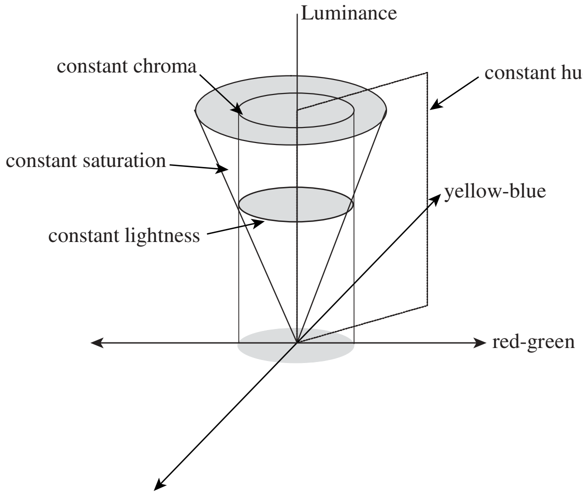


Figure 10. Loci of constant hue, saturation, lightness, and chroma shown in a perceptual color space.

Metamerism

According to Equation (3) , if we can control the stimulus that enters the eye for a given color, then to *match* two colors we merely need to match their resulting stimulus measurements c. In other words, two different spectra can be made to appear the same. Such stimuli are called *metamers*. If $c_1 = c_2$, then

$$\int_{\lambda\,=\,380\text{ nm}}^{\lambda\,=\,830\text{ nm}} \text{lms}(\lambda) i_{r_1}(\lambda) d\lambda = \int_{\lambda\,=\,380\text{ nm}}^{\lambda\,=\,830\text{ nm}} \text{lms}(\lambda) i_{r_2}(\lambda) d\lambda \tag{5}$$

Different manifestations of the above equality carry different names: “observer”, “illuminant”, and “object” metamerism, depending on whether equal stimuli c result from changing the sensor functions **lms**(λ), the light $i(\lambda)$, or the surface $r(\lambda)$. So, two completely different spectral stimuli can be made to generate the same cone stimuli for the same observer—a property that color engineers in fields ranging from textiles to televisions have considered a blessing for decades, because changes of pigments, dyes, phos-

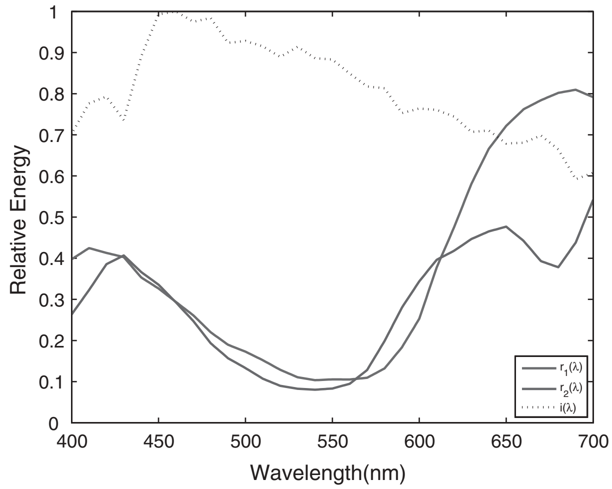


Figure 11. Metameric reflectances $r_1(\lambda)$ and $r_2(\lambda)$. Although their reflectance functions differ, under the illuminant $i(\lambda)$, their stimulation of the cones is identical and color perceived is the same.

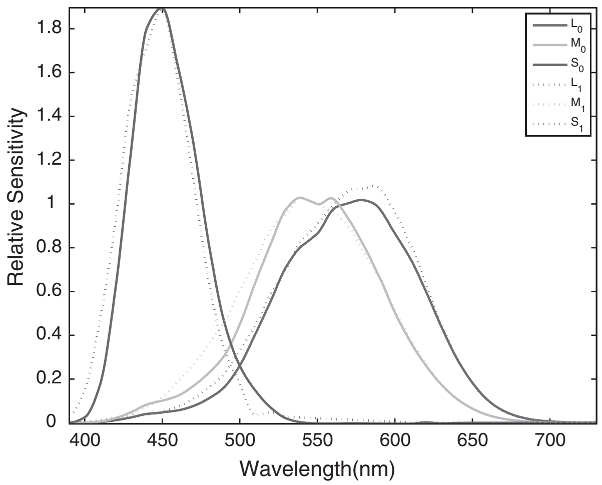


Figure 12. Different observer cone functions showing observer variances.

phors, and color filters can achieve a consistent perception of colors across various media. Equal colors are called metameric. Consider, for example, two color samples that have reflectance functions $r_1(\lambda)$ and $r_2(\lambda)$, as in Fig. 11. When plotted on a wavelength scale, it may appear that these two reflectance functions must result in completely different perceptions to the observer. However, if we were to apply the same illuminant and observer sensitivity functions to these otherwise different colors, they result in identical colors being perceived by the eye. These two colors (reflectance functions) hence are called *metamers*, and this is an example of object metamerism.

On a similar note, consider two patches of color with reflectance functions $r_3(\lambda)$ and $r_4(\lambda)$ being viewed under identical illumination conditions by two different observers (observers whose cone functions are not the same), as shown in Fig. 12. One observer would view these patches as being the same (they are metameric), whereas the other would view this exact same pair as distinctly different—resulting in observer metamerism. This kind of metamerism is relatively common, because most, if not all, concepts related to color are built around a "standard" or "average" observer, whereas in fact significant variation exists between observers. The final type of metamerism, illuminant metamerism, consists of metameric colors that arise from the same observer and reflectance but different lights.

Adaptation

Arguably, the most remarkable capability of the human visual system is its ability to adapt to changes in the illuminant. This ability may be classified broadly as lightness and chromatic adaptation. The resulting effect is that despite changes in the spectral content of the light or its absolute power, the visual system maintains quite constant overall perception. However, certain limitations apply to these abilities; these changes are limited mainly to changes in *natural* illuminants and objects. This occurance is to be expected given the types of illuminants with which humans have been most familiar.

This ability is explained best by means of an example. As one walks out of a relatively dark movie theater to bright afternoon sunlight, it takes only a few seconds for the visual system to adapt to as much as two orders of magnitude change in the intensity of the illuminant, without change in visual experience. Here, the cones are the dominant photoreceptors, and the rods have become bleached (unable to replenish their photopigments). This type of adaptation is referred to as *luminance* or *light adaptation*. Similarly, entering a dimly lit movie theater from the bright sunny outdoors again requires time to adapt to the dark conditions, after which our visual system has adapted well to the surroundings. This, however, takes slightly longer than in the former situation because now the rods need to become active, requiring them to unbleach, which is a comparatively longer process. This kind of adaptation is called *dark adaptation*. The ability to dark and light adapt gives us the ability to have reasonable visual capability in varying illuminant conditions while taking maximal advantage of the otherwise limited dynamic range of the photoreceptors themselves.

A second, and perhaps the most fascinating, mode of adaptation is called *chromatic adaptation*. This term refers to the ability of the visual system to maintain color perception under small, but significant, changes in the spectral content of the illuminant. A newspaper seems to maintain its mostly white background independent of whether we look at it outdoors under an overcast sky, indoors under a fluorescent lamp, or under an incandescent lamp. Consider looking at a bowl of fruit that contains a red apple, a yellow banana, and other fruit under an incandescent illuminant. The apple will appear red and the banana yellow. Changing the illuminant to a typical fluorescent lamp, which greatly alters the spectral content of the light, does not appear to change the color of the apple or the banana, after a few seconds of adaptation. The human visual system maintains its perception; our visual system has adapted chromatically. Interestingly, our ability to adapt to changes in spectral content of the illuminant are limited mostly to changes in natural illuminants such as sunlight.

Color Constancy

The phenomenon of objects maintaining their appearance under varying illuminants is referred to as *color constancy*. For example, the appearance of a dress that looks red in the store might look nothing like a red under street lighting (e.g., sodium-vapor lamps); the visual system cannot adapt as well to a sodium-vapor lamp as it can to a fluorescent or incandescent lamp; thus it inconsistently renders the perception of this dress fabric color. This subject is interesting given the formation model described in Equation (3). The information the eye receives from the object changes with the illuminant athough, given the premise of color constancy, the net result of the visual system needs to stay the same. This occurrence is known however, to be untrue for humans as we take informational cues from the color of the illuminant and perform some form of chromatic adaptation (11). The study of color constancy provides us with clues that describe how the human visual system operates and is used often by computational color technologists in maintaining numerical color constancy.

Color inconstancy is a battle that textile and paint manufacturers, camera and display manufacturers, and printer manufacturers regularly have to fight because significant changes may take place between illuminants in retail stores and in the home or between hardcopy and on-screen imaging. In each case, the color data reside in a different color space with its own color appearance models. Moreover, illuminants are difficult to control because we typically have mixed illuminants (not just one specific type) in whatever surrounding we are in.

After Images

When stimulated for an extended period of time by a strong stimulus, the human visual system adapts, and when the source of this stimulus is removed, a negative after image appears for a short period of time, most commonly attributed to sensor fatigue. Many forms of after images have been shown to be valid. The most commonly known type of after images is the one formed via color responses, known as chromatic after images. For example, if individuals fix their gaze on a picture of a brightly colored set of squares for some time and then a plain white stimulus is presented quickly to the eye, the individuals experience a negative image of corresponding opponent colors. Other forms of after images and visual stimulus adaptation that may be of interest to the reader have been demonstrated by Fairchild and Johnson (12).

SIMPLE COLOR APPEARANCE PHENOMENA

Specifying a color merely by its physical attributes has its advantages but has drawbacks, too, when describing the appearance of colors, especially in somewhat complex scenes. We illustrate these difficulties via some examples.

Simultaneous Lightness and Color Contrast

Consider an achromatic color placed in a relatively simple scene as shown in the upper half of Fig. 13. Both central squares seem to have the same luminance. However, if we reduce the luminance of the background in one half and increase it in the other while keeping the same central achromatic patches as shown in the lower half of the figure, one patch appears brighter than the other although in fact they have exactly the same luminance. This occurrence may be attributed to the presence of reinforcing and inhibiting ON–OFF receptive fields that work locally to enhance differences. A similar phenomenon occurs when we change the color of the background. The achromatic patches would seem to have the opponent color of the background and no longer retain their achromatic nature. In Fig. 14, all four inner squares are the same achromatic color. However, each square contrasts with its surrounding, resulting in the appearance of opponent colors. Note that the upper left square has a greenish tint, the upper right a bluish tint, the

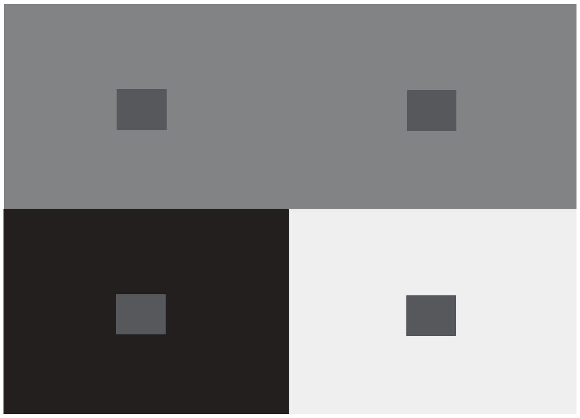


Figure 13. A simple example that demonstrates simultaneous lightness contrast. Notice that the same gray patches as in the upper half of the image seem to have different brightnesses when the background luminance is changed.

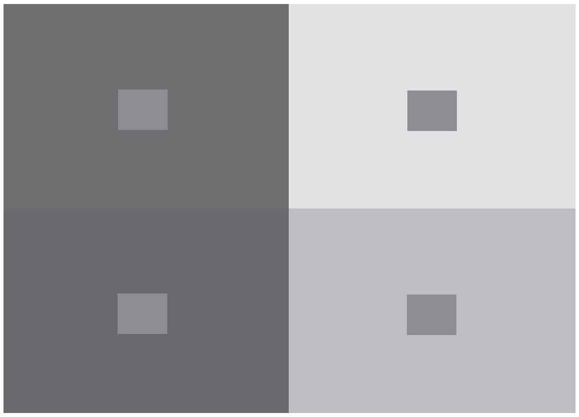


Figure 14. A simple example that demonstrates simultaneous color contrast. Notice that the same gray patches as in the upper half of the image seem to have different hues when the background luminance is changed.

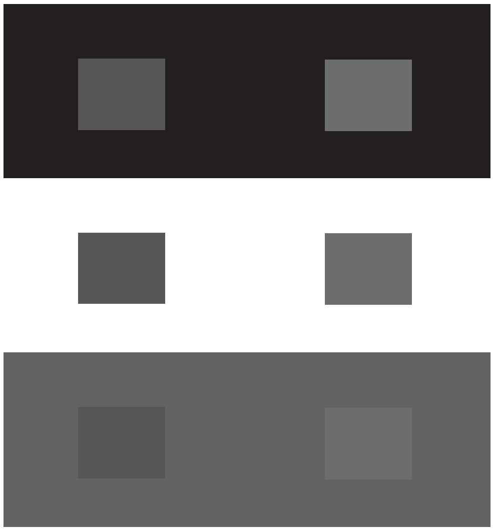


Figure 15. An example that demonstrates lightness crispening. Notice that the difference between the gray patches in the white and black backgrounds is hardly noticeable, but when the lightness of the background is in between that of the two patches the appearance of the difference is accentuated.

bottom left a yellowish tint, and the bottom right a reddish tint.

Lightness and Chromatic Crispening

The difference between two colors that are only slightly different is heightened if the color of the background that surrounds them is such that its color lies between those of the two patches. Figure 15 illustrates this phenomenon for lightness. Note that the difference between the two patches is hardly noticeable when the background is white or black, but when the luminance of the background is in between the colors of the patches, the difference is accentuated greatly.

Only a few color appearance phenomena have been addressed in this article. We have looked at some phenomena that are easy to observe and do not need in-depth study. Color appearance phenomena are the primary drivers of image quality assessments in the imaging industry. The interested reader is referred to books on this topic included in the Bibliography.

ORDER IN COLOR PERCEPTION

From the preceding sections on the various color appearance terms, one may gather that many potential candidates exist for ordering color perceptions. One means is based on the somewhat-orthogonal dimensions of redness–greenness, yellowness–blueness, and luminance. These axes may be placed along Euclidean axes, as shown in Fig. 16. Color spaces with such an ordering of axes form the basis of all computational color science.

Another method for ordering color perceptions could be based on hue, chroma, and lightness, which again could be placed along the Euclidean axes as in Fig. 17. It turns out that the two orderings are related to each other: The hue–chroma–lightness plot is simply another representation of the opponent-color plot. Such a relationship was found also by extensive studies on color orderings performed by Munsell in the early 1900s. In his publications, Munsell proposed a color ordering in which the spacing between each color and its neighbor would be perceived as equal. This resulted in a color space referred to as the Munsell color solid, which to date is the most organized, successful, and widely used color order system. Munsell proposed a notation for colors that specifies their exact location in the color solid. A vertical value (V) scale in ten steps denotes the luminance axis. Two color samples along the achromatic axis (denoted by the letter N for neutrals) are ordered such that they are spaced uniformly in terms of our perception; for example, a sample with a value of 4 would correspond to one that is half as bright as one with a value of 8. Munsell defined basic hues (H) of red (R), yellow (Y), green (G), blue (B), and purple (P) and combinations (RP for red-purples and so on) that traverse the circumference of a circle, as shown in Fig. 18. A circle of constant radius defines

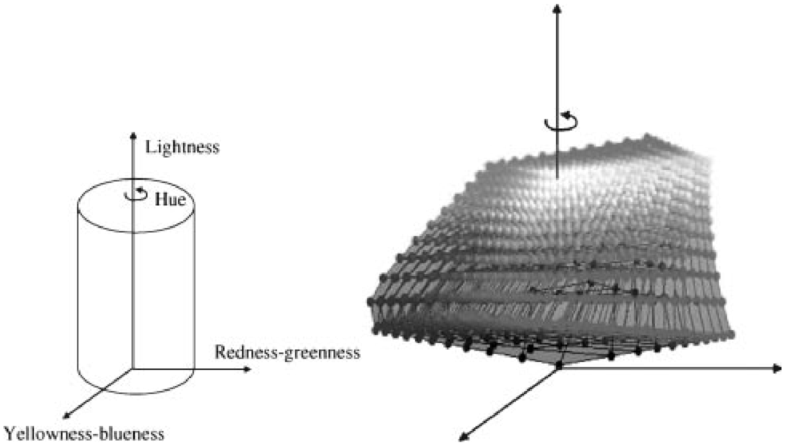


Figure 16. A plot of lightness, redness–greenness, and yellowness–blueness ordered along Euclidean axes.

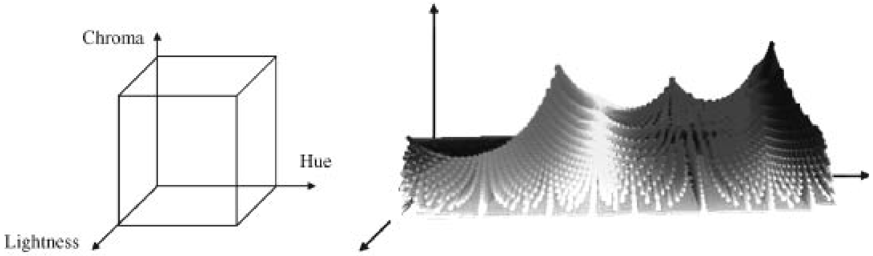


Figure 17. A plot of lightness, chroma, and hue ordered along the Euclidean axes.

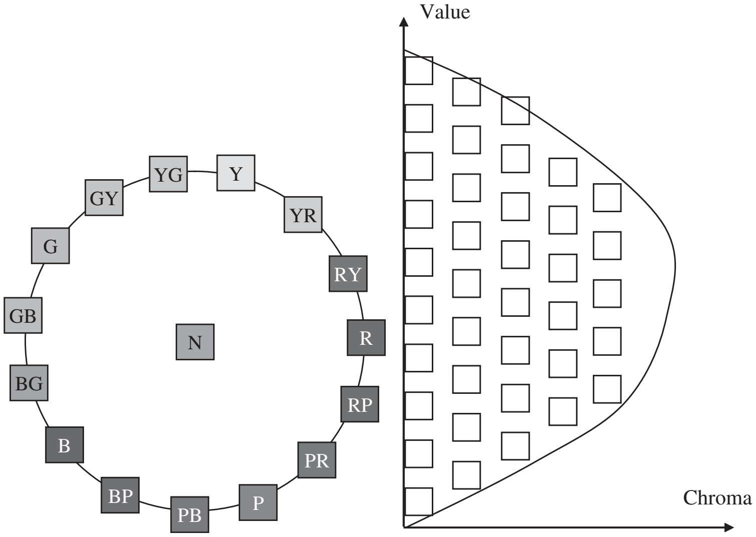


Figure 18. A plot of a constant value plane (left) that shows the various hue divisions of a constant chroma circle in the Munsell notation, alongside a constant hue plane (right).

the locus of colors with the same chroma (C) or deviations from the achromatic axis.

Increasing radii denote higher chroma colors on an open-ended scale. In this fashion, a color is denoted by H V/C (Hue Value/Chroma). For example, 5GY6/10 denotes a hue of 5GY (a green–yellow midway between a green and a yellow) at value 6 and chroma 10. Most modern computational color models and color spaces are based on the fundamentals of the Munsell color order system.

The NCS color order system is another ordering scheme, much more recent and gaining acceptance (13). The NCS color ordering system is based on the work of the Hering opponent color spaces. The perceptual axes used in the NCS are blackness–whiteness, redness–greenness, and yellowness–blueness; these colors are perceived as being "pure" (see Fig. 19). The whiteness–blackness describes the *z*-dimension, whereas the elementary colors (red, green–yellow, and blue) are arranged such that they divide the x–y plane into four quadrants. Between two unique hues, the space is divided into 100 steps. A color is identified by its blackness (s), its chromaticness (c), and its hue. For example, a color notated by 3050-Y70R denotes a color with a blackness value of 30 (on a scale of 0 to 100), a chromaticness of 50 (an open-ended scale), and a hue described as a yellow with 70% red in its mixture. A good reference that details the history and science of color order systems was published recently by Kuehni (14).

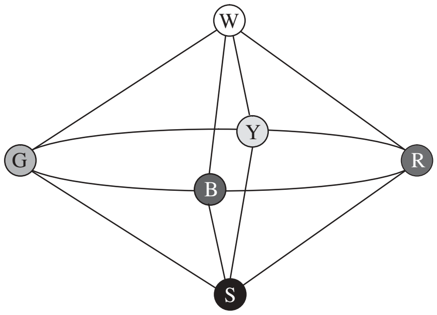


Figure 19. A schematic plot of the NCS color space.

CONCLUSIONS

The multitide of effects and phenomena that need to be explored in color vision and perception is profound. One would imagine that color science, a field with such everyday impact and so interwoven with spoken and written languages, would be understood thoroughly by now and formalized. But the mechanisms of vision and the human brain are so involved that researchers only have begun unraveling the complexities involved. Starting from the works of ancient artisans and scientists and passing through the seminal works of Sir Issac Newton in the mid-1600s to the works of the most recent researchers in this field, our knowledge of the complexities of color has increased greatly, but much remains to be understood.

BIBLIOGRAPHY

1. G. Wyszecki and W. S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*, 2nd ed. New York: Wiley-lnterscience, 2000.
2. M. D. Fairchild, *Color Appearance Models,* 2nd ed. New York: John Wiley & Sons, 2005.
3. A. Roorda and D. Williams, The arrangement of the three cone classes in the living human eye, *Nature*, **397**: 520–522, 1999.
4. D. Brainard, A. Roorda, Y. Yamauchi, J. Calderone, A. Metha, M. Neitz, J. Neitz, D. Williams, and G. Jacobs, Consequences of the relative numbers of 1 and m cones, *J. Optical Society of America A*, **17**: 607–614, 2000.
5. K. Nassau, *The Physics and Chemistry of Color: The Fifteen Causes of Color*. New York: John Wiley & Sons, 1983.
6. R. G. Kuehni, *Color: An Introduction to Practice and Principles*, 2nd ed. New York: Wiley-Interscience, 2004.
7. R. S. Berns, *Billmeyer and Saltzman' s Principles of Color Technology*, 3rd ed. New York: John Wiley & Sons, 2000.
8. P. K. Kaiser and R. Boynton, *Human Color Vision*, 2nd ed. Optical Society of America, 1996.
9. Commission Internationale de l' Eclairage, *A Color Appearance Model for Colour Management Systems: CIECAM02*. CIE Pub. 159, 2004.
10. R. Hunt, *The Reproduction of Colour*, 6th ed. New York: John Wiley & Sons, 2004.
11. D. Jameson and L. Hurvich, Essay concerning color constancy, *Ann. Rev. Psychol.*, **40**: 1–22, 1989.
12. M. Fairchild and G. Johnson, On the salience of novel stimuli: Adaptation and image noise, *IS&T 13th Color Imaging Conference*, 2005, pp. 333–338.
13. A. Hård and L. Sivik, NCS-natural color system: A Swedish standard for color notation, *Color Res. Applicat.*, **6**(3): 129–138, 1981.
14. R. G. Kuehni, *Color Space and Its Divisions: Color Order from Antiquity to the Present*. New York: Wiley-Interscience, 2003.

RAJEEV RAMANATH
Texas Instruments Incorporated
Plano, Texas
MARK S. DREW
Simon Fraser University
Vancouver, British Columbia,
Canada

CONTOUR TRACKING

Object tracking is a fundamental area of research that finds application in a wide range of problem domains including object recognition, surveillance, and medical imaging. The main goal of tracking an object is to generate a trajectory from a sequence of video frames. In its simplest form, an object trajectory is constituted from the spatial positions of the object centroid and resides in a three-dimensional space defined by the image and time coordinates. In the case when the changes in the object size and orientation are tracked also, such as by a bounding box around the object, the dimensionality of the trajectory is increased by two and includes the scale and orientation, in addition to time and image dimensions.

A trajectory in a higher dimensional space provides a more descriptive representation of the object and its motion. Depending on the application domain, an increase in the trajectory dimensionality may be desirable. For instance, in the context of motion-based object recognition, a trajectory that encodes the changes in the object shape over a time period increases the recognition accuracy. The additional information encoded in the trajectory also provides a means to identify the actions performed by the objects, such as sign language recognition, where the shape of the hands and their interactions define the sign language vocabulary.

The most informative trajectory is the one that encodes the deformation in the object shape. This task requires tracking the area occupied by the object from one video frame to the next. A common approach in this regard is to track the contour of an object, which is known also as the contour evolution. The contour evolution process is achieved by minimizing a cost function that is constituted of competing forces trying to contract or expand the curve. The equilibrium of the forces in the cost function concludes the evolution process. These forces include regularization terms, image-based terms, and other terms that attract the contour to a desired configuration. The latter of these terms traditionally encodes a priori shape configurations that may be provided ahead of time.

REPRESENTING THE OBJECT AND ITS CONTOUR

The object contour is a directional curve placed on the boundary of the object silhouette [see Fig. 1(c) and (d)] . The contours are used either in a contour-based representation or in a boundary condition in a region-based representation. The region-based representation uses the distance transform, Poisson equation, of the medial axis. The distance transform assigns each silhouette pixel with its shortest distance from the object contour (1). In a similar vein, the Poisson equation assigns the mean of the distances computed by random walks reaching the object contour (2). The medial axis generates skeletal curve segments that lie on the locus of circles that are tangent to the object contour at two or more points [see in Fig. 1(c)]. Use of the contour as a boundary condition requires explicit detection of the object and prohibits defining a cost function that evolves an initial contour to its final configuration. Hence, in the remainder of the text, we will discuss the contour-based representation and related contour evolution techniques.

A contour can be represented explicitly or implicitly. Explicit representations define the underlying curve para-metrically and perform tracking by changing the parameters that, in turn, evolve the contour. Parametric representations require analytical expressions that provide a means to compute the geometric features used during the contour evolution. The most common parametric representation in the context of contour tracking uses a set of control points positioned on the object boundary. The use of different control points for different objects generates a unique coordinate system for each object, which is referred to as the Lagrangian coordinates [see Fig. 1(e)]. In the Lagrangian coordinates, the relations between the control points play an important role for computing the geometric properties of the underlying curve. These relations can be realized by either the finite difference approximation or the finite element analysis. The finite difference approximation treats each control point individually and assumes that they are connected by lines. On the contrary, the finite element analysis defines the relations by a linear combination of a set of functions referred to as the splines. The splines generate continuous curves that have parametric forms. Their parametric nature permits the computation of the geometric curve features analytically. The contour tracking in these representations is achieved by moving the control points from one place to another. For more information, we refer the reader to the article on “Snakes: Active Contours.”

Contrary to the explicit representation, the implicit contour representations for different objects lie in the same Cartesian coordinates, namely the Eulerian coordinates (grid). The contour in the Eulerian coordinates is defined based on the values for the grid positions. For instance, one common approach used in fluid dynamics research, which investigates the motion of a fluid in an environment, is to use volumetric representation. In volumetric representation, each grid is considered a unit volume that is filled with water, such that inside the contour (or surface in higher dimensions) the unit volumes are filled, whereas for outside they are empty. In the field of computer vision, the most common implicit representation is the level-set method. In the level-set method, the grid positions are assigned a signed Euclidean distance from the closest contour point. This method is similar to the distance transformation discussed for representing regions, with the difference of including a sign. The sign is used to label inside and outside the contour, such that grid positions inside the closed contour are positive, whereas the outside grid positions are negative. The signed distances uniquely

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

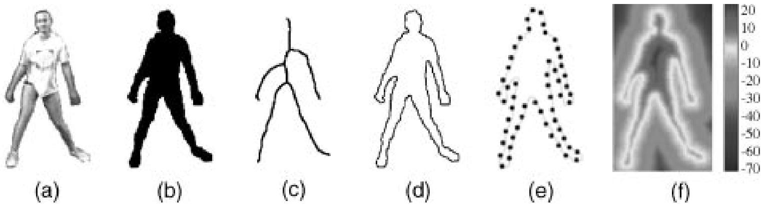


Figure 1. Possible representations for the object shape given in (a):(b) object silhouette, (c) skeleton and (d) its contour. Representing the contour by using (e) a set of control points in the Lagrangian coordinates and (f) level-sets in the Eulerian coordinates.

locate the contour, such that it resides on the zero-crossings in the grid. The zero-crossings are referred to as the zero level-set. The evolution of the contour is governed by changing the grid values based on the speed computed at each grid position. For more information, we refer the reader to the article on the "Level-Set Methods."

THE STATE SPACE MODELS FOR CONTOUR TRACKING

The state space models define the object contour by a set of states, $X^t : t = 1, 2 \ldots$. Tracking then is achieved by updating the contour state in every frame:

$$X^t = f^t(X^{t-1}) + W^t \tag{1}$$

where W^t is the white noise. This update eventually maximizes the posterior probability of the contour. The posterior probability depends on the prior contour state and the current likelihood, which is defined in terms of the image measurements Z^t. A common measurement used for contour tracking is the distance of the contour from the edges in the image.

The state space models-based contour tracking involve two major steps. The first step predicts the current location of the contour, such as the new position of each control point, and the second step corrects the estimated state, according to the image observations. The state prediction and correction is performed by using various statistical tools. Among others, the Kalman filtering and the particle filtering are the most common statistical tools. Computationally, the Kalman filter is more attractive because only one instance of the object state is required to perform prediction and correction. However, the Kalman filter assumes that the object state is distributed by a Gaussian, which may result in a poor estimation of the state variables that are not Gaussian distributed. The particle filtering overcomes this limitation by representing the distribution of the object state by a set of samples, referred to as the particles (3). Each particle has an associated weight that defines the importance of that particle. Keeping a set of samples for representing the current state requires maintaining and updating all the instances during the correction step, which is a computationally complex task.

Tracking the object contour using the state space methods involves careful selection of the state variables that represent the object shape and motion. For this purpose, Terzopoulos and Szeliski (4) use a spring model to govern the contour motion. In this model, the state variables include the stiffness of the spring placed at each control point. Once the object state is estimated, a correction is made by evaluating the gradient magnitude from the image. Isard and Blake (5) model the shape and rigid motion by using two state variables that correspond to the spline parameters and the affine motion. The image measurements used to correct the estimated state include the edges in the image observed in the normal direction to the contour [see Fig. 2(a)]. This approach has been extended recently to include nonrigid contour deformations that are computed after the rigid object state is recovered (6).

DIRECT MINIMIZATION OF THE COST FUNCTION

The methods falling under this category iteratively evolve the contour by minimizing an associated cost function. The cost function is constituted of the optical flow field or the appearance observed inside and outside the object and is minimized by a greedy algorithm or a gradient descent method.

The contour tracking based on the optical flow field exploits the constancy of the brightness of a pixel in time:

$$I^{t+1}(x,y) - I^t(x-u, y-v) = 0 \tag{2}$$

where I is the imaging function, t denotes the frame number, and (u,v) is the optical flow vector. The optical flow during the contour evolution can be computed by searching for similar color in a neighborhood of each pixel (7). Once the flow vectors for all the object pixels are computed, the cost of moving the contour can be evaluated by accumulating the brightness similarities using Equation (2). Tracking

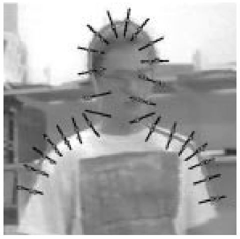


Figure 2. Edge observations along the contour normals. (Reprinted with permission from the IEEE.)

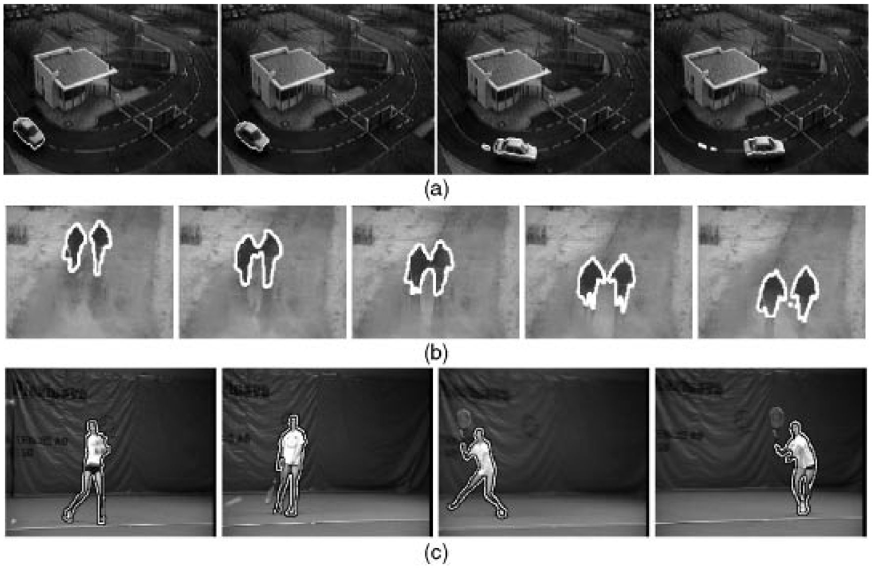


Figure 3. Tracking results of the methods proposed in (a) Ref. 7, (b) Ref. 8, and (c) Ref. 9. (Reprinted with permission from the IEEE.)

results of this approach are shown in Fig. 3(a). An alternative approach to computing the optical flow is to adopt a morphing equation that morphs the intensities in the previous frame to the intensities in the current frame (8). The intensity morphing equation, however, needs to be coupled with a contour tracking function, such that the intensities are morphed for the contour pixels in the previous and the current frame. The speed of the contour is computed according to the difference between the intensities of the corresponding pixels. For instance, if the difference is high, then the contour moves with the maximum speed in its normal direction and while the morphing function is evaluated by considering the new position of the contour. The tracking results using this approach are shown in Fig. 3(b). The cost function based on the optical flow also can be written in terms of the common motion constraint (10). The common motion constraint assumes that the motion inside the contour is homogenous, such that the contour is evolved to a new position if the difference between neighboring motion vectors is high.

In contrast to the cost functions using brightness constancy, the statistics computed inside and outside the object contour impose a less strict constraint. An important requirement of statistics-based methods is the initialization of the contour in the first frame to generate the appearance statistics. Region statistics can be computed by piecewise stationary color models generated from the subregions around each control point (11). This model can be extended to include the texture statistics generated from a band around the contour (9). Using a band around the contour combines image gradient-based and region statistics-based contour tracking methods into a single framework, such that when the width of the band is set to one, the cost function is evaluated by image gradients. The contour tracking results using region statistics is shown in Fig. 3(c).

THE SHAPE PRIORS

Including a shape model in the contour cost function improves the estimated object shape. A common approach to generate a shape model of a moving object is to estimate the shape distribution associated with the contour deformations from a set of contours extracted online or off line. The shape distribution can be in the form of a Gaussian distribution, a set of eigenvectors, or a kernel density estimate. The cost function associated with these distributions contains contour probabilities conditioned on the estimated shape distribution.

For the explicit contour representations, the shape model is generated using the spatial-position statistics of the control points. A simple shape prior in this context is to use a Gaussian distribution (10):

$$p(x_i) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(x_i - \mu_{x_i})^2}{2\sigma_{x_i}^2} - \frac{(y_i - \mu_{y_i})^2}{2\sigma_{y_i}^2}) \qquad (3)$$

where μ denotes the mean, σ denotes the standard deviation, and $\mathrm{x}_i = (x_i, y_i)$ is the position of the ith control point. Before modeling, this approach requires registration of the contours to eliminate the translational effects. Registration can be performed by mean normalization of all the contours.

An alternative shape model can be computed by applying the principal component analysis (PCA) to the vectors of the control points. The PCA generates a new coordinate system that emphasizes the differences between the contours, such that selecting a subset of principal components (eigenvectors with the highest eigenvalues) models the underlying contour distribution. Given an input contour, the distance is computed by first reconstructing the input using a linear combination of the selected principal components and then evaluating the Euclidean distance

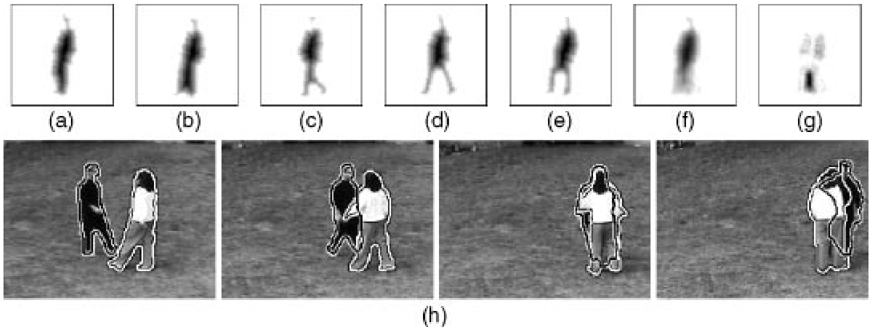


Figure 4. (a–e) A sequence of level-sets generated from walking action. (f) Mean level-set and (g) standard deviation level-set. (h) Tracking results for occluded person using the shape model given in (f) and (g). (Reprinted with permissions from the IEEE.)

between the input vector and the reconstructed contour. The weights in the linear combination are computed by projecting the input contour to the principal components.

The shape priors generated for implicit contour representations do not model explicitly the contour shape. This property provides the flexibility to model the objects with two or more split regions. Considering the level-set representation, which defines the contour by zero crossings on the level-set grid, a shape model can be generated by modeling distance values in each grid position by a Gaussian distribution (9). This modeling two level-set functions for each set of contours, as shown in Fig. 4(a–g), that correspond to the mean and the standard deviation of the distances from the object boundary.

DISCUSSION

Compared with tracking the centroid, a bounding box, or a bounding ellipse, contour tracking provides more detailed object shape and motion that is required in certain application domains. For instance, the contour trackers commonly are used in medical imaging, where a more detailed analysis of the motion of an organ, such as the heart, is required. This property, however, comes at the expense of computational cost, which is evident from the iterative updates performed on all the grid positions or the control points, depending on the contour representation chosen. In cases when the domain of tracking does not tolerate high computational costs, such as real-time surveillance, the contour trackers may be less attractive. This statement, however, will change in coming years, considering the ongoing research on developing evolution strategies that will have real-time performance (12).

The design of a contour tracker requires the selection of a contour representation. Depending on the application domain, both the implicit and explicit representations have advantages and disadvantages. For instance, although the implicit representations, such as the level-set method, inherently can handle breaking and merging of the contours, the explicit representations require including complex mechanisms to handle topology changes. In addition, the implicit representations naturally extend tracking two-dimensional contours to three or more-dimensional surfaces. The implicit representation, however, requires re-initialization of the grid at each iteration, which makes it a computationally demanding procedure compared with an explicit representation.

The choice of the cost function is another important step in the design of a contour tracker and is independent of the contour representation. The cost functions traditionally include terms related to the contour smoothness, image observations, and additional constraints. Among these three terms, recent research concentrates on developing cost functions that effectively use the image observations while adding additional constraints such as shape priors. Especially, the research on the use of innovative constraints to guide the evolution of the contour is not concluded. One such constraint is the use of shape priors, which becomes eminent in the case of an occlusion during which parts of the tracked object are not observed. Improved tracking during an occlusion is shown in Fig. 4(h) where using the shape priors successfully resolves the occlusion.

As with other object tracking approaches, in a contour tracking framework, the start or end of an object trajectory plays a critical role in its application to real-world problems. The starting of a contour trajectory requires segmentation of the object when it first is observed. The segmentation can be performed by using a contour segmentation framework, as discussed in the chapter on “Level-Set Methods”, or by using the background subtraction method, which labels the pixels as foreground or background depending on their similarity to learned background models. Most segmentation approaches, however, do not guarantee an accurate object shape and, hence, may result in poor tracking performance.

BIBLIOGRAPHY

1. A. Rosenfeld and J. Pfaltz, Distance functions in digital pictures, *in Pattern Recognition*, vol. l. 1968, pp. 33–61.
2. L. Gorelick, M. Galun, W. Sharon, R. Basri, and A. Brandt, Shape representation and classification using the poisson equation, *IEEE Conf. an Computer Vision and Pattern Recognition*, 2004.
3. H. Tanizaki, Non-gaussian state-space modeling of nonstationary time series, *J. American Statistical Association*, **82**: 1032–1063, 1987.

4. D. Terzopoulos and R. Szeliski, Tracking with kalman snakes, in A. Blake and A. Yuille (eds.) *Active Vision*. MIT Press, 1992.

5. M. Isard and A. Blake, Condensation—conditional density propagation for visual tracking, *Int. Jrn. on Computer Vision*, **29**(1): 5–28, 1998.

6. J. Shao, F. Porikli, and R. Chellappa, A particle filter based non-rigid contour tracking algorithm with regulation, *Int. Conf. on Image Processing*, 2006, pp. 34–41.

7. A. Mansouri, Region tracking via level set pdes without motion computation, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **24**(7): pp. 947–961, 2002.

8. M. Bertalmio, G. Sapiro, and G. Randall, Morphing active contours, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **22**(7): pp. 733–737, 2000.

9. A. Yilmaz, X. Li, and M. Shah, Contour based object tracking with occlusion handling in video acquired using mobile cameras, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **26**(11): pp. 1531–1536, 2004.

10. D. Cremers and C. Schnorr, Statistical shape knowledge in variational motion segmentation, *Elsevier Jrn. on Image and Vision Computing*, **21**: pp. 77–86, 2003.

11. R. Ronfard, Region based strategies for active contour models. *Int. Jrn. on Computer Vision*, **13**(2): pp. 229–251, 1994.

12. Y. Shi and W. Karl, Real-time tracking using level sets, *IEEE Conf. on Computer Vision and Pattern Recognition*, 2005, pp. 34–41.

ALPER YILMAZ
The Ohio State University
Columbus, Ohio

E

EDGE DETECTION IN GRAYSCALE, COLOR, AND RANGE IMAGES

INTRODUCTION

In digital images, edge is one of the most essential and primitive features for human and machine perception; it provides an indication of the physical extent of objects in an image. Edges are defined as significant local changes (or discontinuities) in brightness or color in an image. Edges often occur at the boundaries between two different regions. Edge detection plays an important role in compute vision and image processing. It is used widely as a fundamental preprocessing step for many computer vision applications, including robotic vision, remote sensing, fingerprint analysis, industrial inspection, motion detection, and image compression (1,2). The success of high-level computer vision processes heavily relies on the good output from the low-level processes such as edge detection. Because edge images are binary, edge pixels are marked with value equal to “1,” whereas others are “0”; edge detection sometimes is viewed as an information reduction process that provides boundary information of regions by filtering out unnecessary information for the next steps of processes in a computer vision system (3).

Many edge detection algorithms have been proposed in the last 50 years. This article presents the important edge detection techniques for grayscale, color, and range images.

EDGE AND EDGE TYPES

Several definitions of edge exist in computer vision literature. The simplest definition is that an edge is a sharp discontinuity in a gray-level image (4). Rosenfeld and Kak (5) defined an edge as an abrupt change in gray level or in texture where one region ends and another begins. An edge point is a pixel in an image where significant local intensity change takes place. An edge fragment is an edge point and its orientation. An edge detector is an algorithm that produces a set of edges from an image. The term “edge” is used for either edge points or edge fragments (6–8).

Edge types can be classified as step edge, line edge, ramp edge, and roof edge (7 –9). Step edge is an ideal type that occurs when image intensity changes significantly from one value on one side to a different value on the other. Line edges occur at the places where the image intensity changes abruptly to a different value, stays at that value for the next small number of pixels, and then returns to the original value. However, in real-world images, step edge and line edge are rare because of various lighting conditions and noise introduced by image-sensing devices. The step edges often become ramp edges, and the line edges become the roof edges in the real-world image (7,8). Figure 1(a)–(d) show one-dimensional profiles of step, line, ramp, and roof edge, respectively.

EDGE DETECTION METHODS IN GRAY-LEVEL IMAGES

Because edges are, based on the definition, image pixels that have abrupt changes (or discontinuities) in image intensity, the derivatives of the image intensity function (10) can be used to measure these abrupt changes. As shown in Fig. 2(a) and (b), the first derivative of the image intensity function has a local peak near the edge points. Therefore, edges can detect by thresholding the first derivative values of an image function or by the zero-crossings in the second derivative of the image intensity as shown in Fig. 2(c).

Edge detection schemes based on the derivatives of the image intensity function are very popular, and they can be categorized into two groups: gradient-based and zero-crossing-based (or called Laplacian) methods. The gradient-based methods find the edges by searching for the maxima (maximum or minimum) in the first derivatives of the image function. The zero-crossing (Laplacian) methods detect the edges by searching for the zero crossings in the second derivatives of the image function.

Gradient-Based Edge Detection

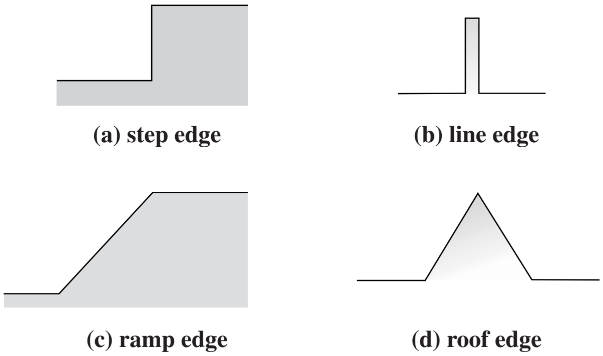
An edge is associated with a local peak in the first derivative. One way to detect edges in an image is to compute the gradient of local intensity at each point in the image. For an image, *f(x, y)* with x and y, the row and the column coordinates, respectively, its two-dimensional gradient is defined as a vector with two elements:

$$G = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} [f(x+dx, y) - f(x, y)]/dx \\ [f(x, y+dy) - f(x, y)]/dy \end{bmatrix} \tag{1}$$

where G_x and G_y measure the change of pixel values in the x- and y-directions, respectively. For digital images, dx and dy can be considered in terms of number of pixels between two points, and the derivatives are approximated by differences between neighboring pixels. Two simple approximation schemes of the gradient for $dx = dy = 1$ are

$$\begin{aligned} &G_x \approx f(x+1,y) - f(x,y)\,,\; G_y \approx f(x,y+1) - f(x,y) \\ &G_x \approx f(x+1,y) - f(x-1,y),\; G_y \approx f(x,y+1) - f(x,y-1) \end{aligned} \tag{2}$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.



(a) step edge

(b) line edge

(c) ramp edge

(d) roof edge

Figure 1. One-dimensional profiles of four different edge types.

These derivative approximation schemes are called the first difference and the central difference, respectively. The convolution masks for the first difference and the central difference can be represented as follows, respectively (11):

$$G_x = \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, \; G_y = \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix},$$

$$G_x = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \; G_y = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

The first difference masks cause the edge location bias because the zero crossings of its vertical and horizontal masks lie at different positions. On the other hand, the central difference masks avoid this position mismatch problem because of the common center of horizontal and vertical masks (11). Many edge detectors have been designed using convolution masks using 3 × 3 mask sizes or even larger.

Two important quantities of the gradient are the magnitude and the direction of the gradient. The magnitude of the gradient G_m is calculated by

$$G_m = \sqrt{G_x^2 + G_y^2} \tag{3}$$

To avoid the square root computation, the gradient magnitude is often approximated by

$$G_m \approx |G_x| + |G_y| \text{ or} \tag{4}$$

$$G_m \approx \max(|G_x|, |G_y|) \tag{5}$$

The direction of the gradient is given by

$$\theta_g = \begin{cases} \tan^{-1}\left(\dfrac{G_y}{G_x}\right) & \text{if} \quad G_x \neq 0 \\ 0^0 & \text{if} \quad G_x = 0 \cap G_y = 0 \\ 90^0 & \text{if} \quad G_x = 0 \cap G_y \neq 0 \end{cases} \tag{6}$$

where the angle θ_g is measured with respect to the x-axis.

In general, a gradient-based edge detection procedure consists of the following three steps:

1. *Smoothing filtering:* Smoothing is used as a preprocessing step to reduce the noise or other fluctuations in the image. Gaussian filtering (10–13) is a well-known low-pass filter, and σ parameter controls the strength of the smoothing. However, smoothing filtering can also blur sharp edges, which may contain important features of objects in the image.
2. *Differentiation:* Local gradient calculation is implemented by convolving the image with two masks, G_x and G_y, which are defined by a given edge detector. Let us denote the convolution of a mask M_{kxk} and an

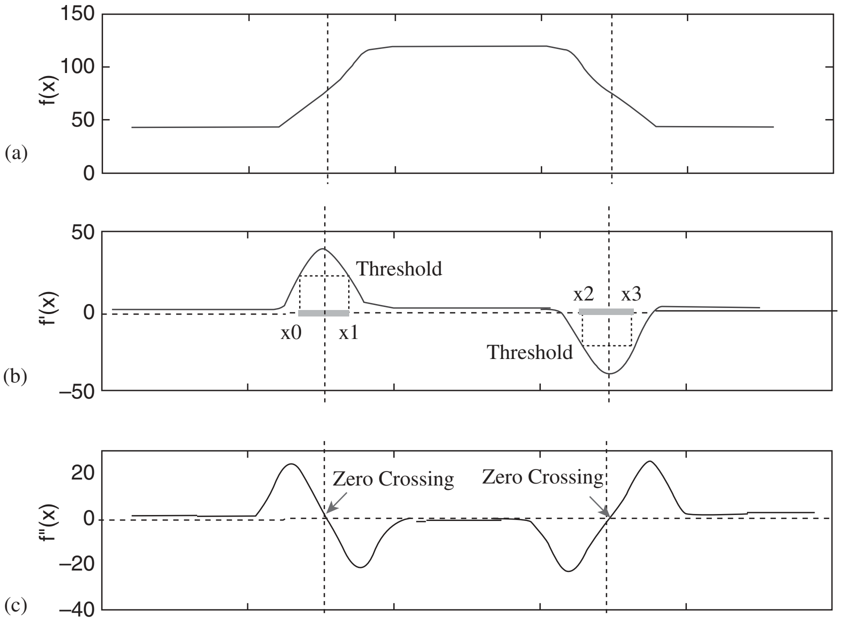


Figure 2. Edge detection by the derivative operators: (a) 1-D profile of a smoothed step edge, (b) the first derivative of a step edge, and (c) the second derivative of a step edge.

image f_{mxn} as N_{mxn}. For every pixel (i,j) in the image f, we calculate:

$$N(i,j) = M \oplus f(i,j) = \sum_{k2=-\frac{k}{2}}^{\frac{k}{2}} \sum_{k1=-\frac{k}{2}}^{\frac{k}{2}} M\,(k1,k2) \times f(i+k1, j+k2 \qquad (7)$$

for $1 \leq i \leq m,\ 1 \leq j \leq n$

3. *Detection:* Detecting edge points based on the local gradients. The most common approach is to apply a threshold to image gradients. If the magnitude of the gradient of an image pixel is above a threshold, the pixel is marked as an edge. Some techniques such as the hysteresis method by Canny (3) use multiple thresholds to find edges. The thinning algorithm is also applied to remove unnecessary edge points after thresholding as necessary (14,15).

Roberts Cross Edge Detector. The Roberts cross edge detector is the simplest edge detector. It rotated the first difference masks by an angle of 45°. Its mathematic expressions are (10,12)

$$G_x = f(x, y+1) - f(x+1, y),$$
$$G_y = f(x, y) - f(x+1, y+1) \qquad (8)$$

G_x and G_y can be represented in the following convolution masks:

$$G_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix},\ G_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

These two convolution masks respond maximally to edges in diagonal directions (45°) in the pixel grid. Each mask is simply rotated 90° from the other. The magnitude of the gradient is calculated by Equation (3). To avoid the square root computation, the computationally simple form of the Robert cross edge detector is the Robert's absolute value estimation of the gradient given by Equation (4).

The main advantage of using Roberts cross edge operator is its fast computational speed. With a 2 × 2 convolution mask, only four pixels are involved for the gradient computation. But the Roberts cross edge operator has several undesirable characteristics. Because it is based on the first difference, its 2 × 2 diagonal masks lie off grid and cause edge location bias. The other weak points are that it is sensitive to noise and that its response to the gradual(or blurry) edge is weak. Figure 3(b)–(g) show the Roberts cross edge maps with various threshold values. The experiment shows that the Roberts cross edge detector is sensitive to the noise with low threshold values. As the threshold value increases, noise pixels are removed, but also real edge pixels with weak response are removed (11).

Prewitt Edge Operator. The Prewitt edge operator uses the central difference to approximate differentiation. Two Prewitt convolution masks at x- and y-directions are shown below:

$$G_x = \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix},\ G_y = \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Because it has the common center of G_x and G_y masks, the Prewitt operator has less edge-location bias compared with the first difference-based edge operators. It also accomplishes noise reduction in the orthogonal direction by means of local averaging (11).

The local gradient at pixel (x, y) can be estimated by convolving the image with two Prewitt convolution masks, G_x and $G_{y,}$ respectively. Mathematically we have

$$\begin{aligned} G_x &= \frac{1}{3}([f(x-1, y+1) + f(x, y+1) + f(x+1, y+1)] - \\ &\quad [f(x-1, y-1) + f(x, y-1) + f(x+1, y-1)]) \\ G_y &= \frac{1}{3}([f(x+1, y-1) + f(x+1, y) + f(x+1, y+1)] - \\ &\quad [f(x-1, y-1) + f(x-1, y) + f(x-1, y+1)]) \end{aligned} \qquad (9)$$

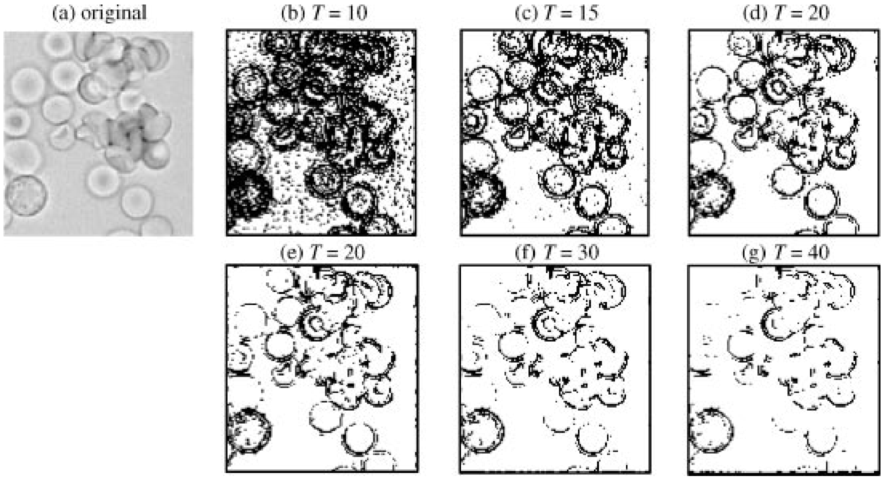


Figure 3. Roberts cross edge maps by using various threshold values: as threshold value increases, noise pixels are removed, but also real edge pixels with weak responses are removed.

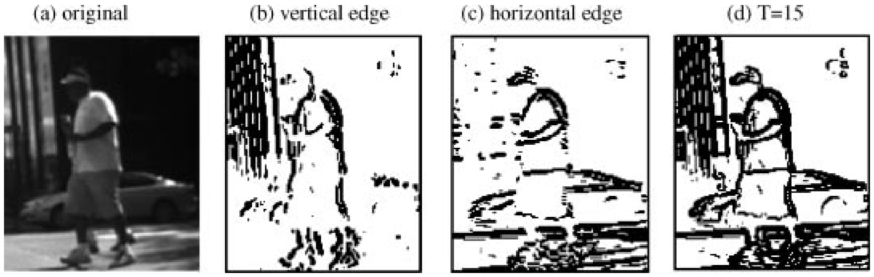


Figure 4. Prewitt edge maps: (a) original image, (b) vertical edge map generated by G_x, (c) horizontal edge map generated by G_y, and (d) complete edge map, $T = 15$.

The Prewitt operators can be extended to detect edges tilting at 45° and 135° by using the following two masks.

$$\begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & +1 \\ 0 & +1 & +1 \end{bmatrix}, \begin{bmatrix} 0 & +1 & +1 \\ -1 & 0 & +1 \\ -1 & -1 & 0 \end{bmatrix}$$

Figure 4(b) shows the vertical edge map generated by the Prewitt G_x mask, and the horizontal edge map generated by the Prewitt G_y mask is shown in Fig. 4(c). Combining these two horizontal and vertical edge maps, the complete edge map is generated as shown in Fig. 4(d).

Sobel Edge Detector. The Sobel gradient edge detector is very similar to the Prewitt edge detector except it puts emphasis on pixels close to the center of the masks. The Sobel edge detector (10,12) is one of the most widely used classic edge methods. Its x- and y-directional 3×3 convolution masks are as follows:

$$G_x = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \; G_y = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The local gradient at pixel (x,y) can be estimated by convolving the image with two Sobel convolution masks, G_x and $G_{y,}$ respectively,

$$\begin{aligned} G_x &= \frac{1}{4}([f(x-1,y+1) + 2\times f(x,y+1) + f(x+1,y+1)] - \\ &\quad [f(x-1,y-1) + 2 \times f(x,y-1) + f(x+1,y-1)]) \\ G_y &= \frac{1}{4}([f(x+1,y-1) + 2\times f(x+1,y) + f(x+1,y+1)] - \\ &\quad [f(x-1,y-1) + 2 \times f(x-1,y) + f(x-1,y+1)]) \end{aligned} \tag{10}$$

The Sobel operator puts more weights on the pixels near the center pixel of the convolution mask. Both masks are applied to every pixel in the image for the calculation of the gradient at the x- and y-directions. The gradient magnitude is calculated by Equation (3). The Sobel edge detectors can be extended to detect edges at 45° and 135° by using the two masks below:

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & +1 \\ 0 & +1 & +2 \end{bmatrix}, \begin{bmatrix} 0 & +1 & +2 \\ -1 & 0 & +1 \\ -2 & -1 & 0 \end{bmatrix}$$

Figure5(j)–(l) show the edge detection results generated by the Sobel edge detector with the threshold value, $T = 20$. Figure 5 also shows the performance analysis of each gradient-based edge detector in the presence of noises. To evaluate the noise sensitivity of each edge detector, 5% and 10% Gaussian noise are added into the original image as shown in Fig. 5(b) and (c), respectively. For fair comparisons, a fixed threshold value is used ($T = 20$) for all edge detectors. Figure 5(e) and (f) show that the Roberts cross edge detector is sensitive to noises. On the other hand, the Sobel and the Prewitt edge detectors are less sensitive to noises. The Sobel operators provide both differencing and smoothing effects at the same time. Because of these characteristics, the Sobel operators are widely used for edge extraction. The smoothing effect is achieved through the involvement of 3×3 neighbors to make the operator less sensitive to noises. The differencing effect is achieved through the higher gradient magnitude by involving more pixels in convolution in comparison with the Roberts operator. The Prewitt operator is similar to the Sobel edge detector. But the Prewitt operator's response to the diagonal edge is weak compared with the response of the Sobel edge operator. Prewitt edge operators generate slightly less edge points than do Sobel edge operators.

Non-Maximum Suppression—a Postprocessing After Gradient Operation. One difficulty in gradient edge detectors (and also in many other edge detectors) is how to select the best threshold (16) used to obtain the edge points. If the threshold is too high, real edges in the image can be missed. If the threshold is too low, nonedge points such as noise are detected as edges. The selection of the threshold critically affects the edge output of an edge operator. Another problem related with the gradient edge detectors is that edge outputs from the gradient-based method appear to be several pixels wide rather than a single pixel (see Figs. 3–5). This problem is because most edges in the real-world images are not step edges and the grayscales around the edges change gradually from low to high intensity, or vice versa. So a thinning process such as nonmaximum suppression (14,15,17–19) may be needed after the edge detection.

The method of nonmaximum suppression is used to remove weak edges by suppressing the pixels with nonmaximum magnitudes in each cross section of the edge direction (15). Here, we introduce the algorithm proposed by Rosenfeld and Thursten(14,15,17–19). Let $\theta(i)$ denote the edge direction at pixel i, and let $G_m(i)$ denote the gradient magnitude at i.

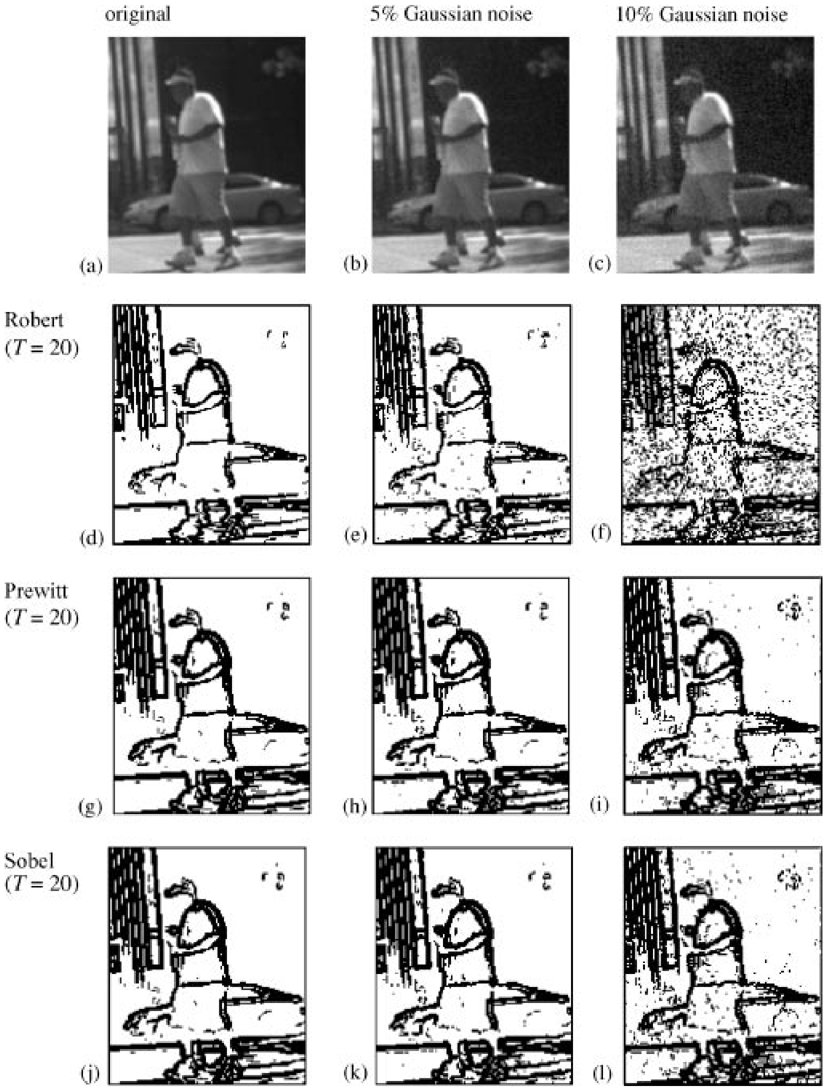


Figure 5. Performance evaluation in the presence of noises for gradient-based edge operators: (a) original image, (b) add 5% Gaussian noise, (c) add 10% Gaussian noise, (d)–(f) Roberts edge maps, (g)–(i) Prewitt edge maps, and (j)–(l) Sobel edge maps. Used the same threshold ($T = 20$) for fair comparisons.

Step 0: For every edge point, *p(x,y)*, do the following steps.

Step 1: Find two nearest neighboring pixels q_1 and q_2 that are perpendicular to the edge direction associated with the edge pixel p.

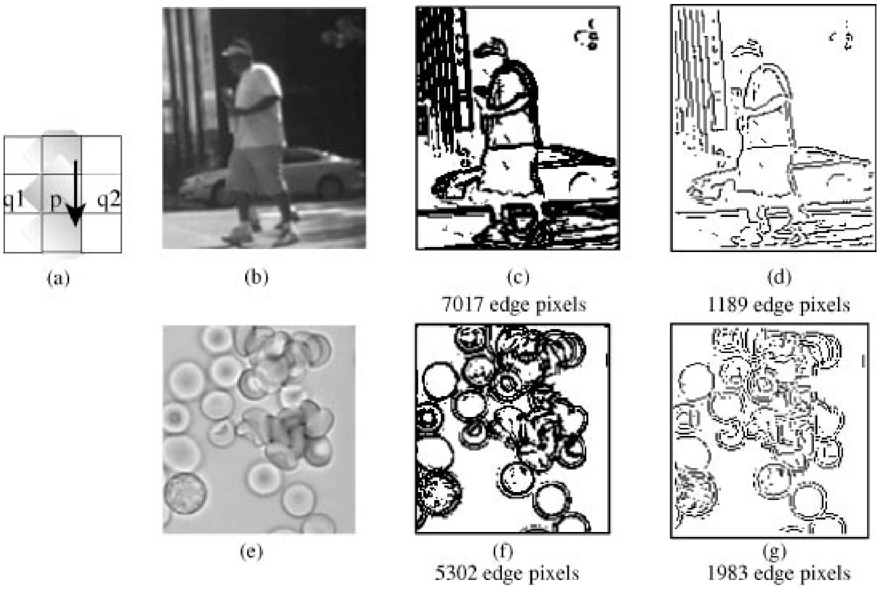
Step 2: If ($|\theta(p) - \theta(q_1)| \leq \alpha$ and $|\theta(p) - \theta(q_2)| \leq \alpha$), then go to ***Step 3*** else return to ***Step 0***.

Step 3: If ($G_m(p) \leq G_m(q_1)$ or $G_m(p) \leq G_m(q_2)$), then suppress the edge at pixel p.

Figure 6(a) shows how to choose q_1 and q_2 when edge direction at pixel p is vertical (top-to-down), which is shown in an arrow. Four edge directions are often used in the nonmaximum suppression method: 0°, 45°, 90°, and 135°, respectively, and all edge orientations are quantized to these four directions. Figure 5(d) and (g) shows the results after the nonmaximum suppression is applied to the edge images in Fig. 5(c) and (f), respectively. Figure 5(d) generated 71.6% less edge pixels compared with the edge pixels in Fig. 5(c). Figure 5(g) has 63.6% less edge pixels compared with the edge pixels in Fig. 5(f). In our experiments, more than 50% of the edge points that were generated by the gradient-based edge operator are considered as nonlocal maximal and suppressed.

Second-Derivative Operators

The gradient-based edge operators discussed above produce a large response across an area where an edge presents. We use an example to illustrate this problem. Figure 2(a) shows a cross cut of the smooth step edge, and its first derivative (or gradient) is shown in Fig. 2(b). After a threshold is applied to the magnitude of the gradient and to all pixels above the



(a) (b) (c) 7017 edge pixels (d) 1189 edge pixels

(e) (f) 5302 edge pixels (g) 1983 edge pixels

Figure 6. Experiments with nonmaximum suppression: (a) an example of how to select q_1, and q_2 when edge direction is top-to-down, (b) and (e) original input images, (c) and (f) Sobel edge maps ($T = 20$) before nonmaximum suppression, and (d) and (g) edge maps after nonmaximum suppression is applied to (c) and (f), respectively.

threshold, for example, the pixels in Fig. 2(b) between $x0 \sim x1$ and $x2 \sim x3$ are considered as edge points. As a result, too many edge points occur, which causes an edge localization problem: Where is the true location of the edge? Therefore, a good edge detector should not generate multiple responses to a single edge. To solve this problem, the second-derivative-based edge detectors have been developed.

Laplacian Edge Detector. The Laplacian of a function *f(x, y)* is given by Equation (11)

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \tag{11}$$

Because the Laplacian edge detector defines an edge as the zero crossing of the second derivative, a single response is generated to a single edge location as observed in Fig. 2(c). The discrete Laplacian convolution mask is constructed as follows:

For a digital image, G_x is approximated by the difference

$$\frac{\partial f(x,y)}{\partial x} = G_x \approx f(x+1,y) - f(x,y), \text{ so}$$

$$\begin{aligned}
\frac{\partial^2 f(x,y)}{\partial x^2} &= \frac{\partial}{\partial x}\left(\frac{\partial f(x,y)}{\partial x}\right) = \frac{\partial G_x}{\partial x} \approx \frac{\alpha(f(x+1,y) - f(x,y))}{\partial x} \\
&= \frac{\alpha f(x+1,y)}{\partial x} - \frac{\partial f(x,y)}{\partial x} \\
&= [f(x+2,y) - f(x+1,y)] - [f(x+1,y) - f(x,y)] \\
&= f(x+2,y) - 2f(x+1,y) + f(x,y)
\end{aligned} \tag{12}$$

This approximation is centered at the pixel $(x + 1, y)$. By replacing $x + 1$ with x, we have

$$\frac{\partial^2 f(x,y)}{\partial x^2} = f(x+1,y) - 2f(x,y) + f(x-1,y) = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

Similarly,

$$\begin{aligned}
\frac{\partial^2 f(x,y)}{\partial y^2} &= f(x,y+1) - 2f(x,y) + f(x,y-1) \\
&= [1 \quad -2 \quad 1]
\end{aligned} \tag{13}$$

By combining the x and y second partial derivatives, the Laplacian convolution mask can be approximated as follows:

$$\nabla^2 f(x,y) = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} + [1 \quad -2 \quad 1] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Other Laplacian convolution masks are constructed similarly by using the different derivative estimation and different mask size (11). Two other 3×3 Laplacian masks are

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & 1 \end{bmatrix}$$

After the convolution of an image with a Laplacian mask, edges are found at the places where convolved image values change sign from positive to negative (or vice versa) passing through zero. The Laplacian edge detector is omnidirectional (isotropic), and it highlights edges in all directions. Another property of the Laplacian edge detector is that it generates closed curves of the edge contours. But the Laplacian edge detector alone is seldom used in real-world computer vision applications because it is too sensitive to image noise. As shown in Fig. 7(c), the very small local peak in the first derivative has its second derivative cross through zero. The Laplacian edge detector may generate spurious edges because of image noise. To avoid the effect of noise, a Gaussian filtering is often applied to an image before the Laplacian operation.

Marr Hildreth—Laplacian of Gaussian. Marr and Hildreth (13) combined the Gaussian noise filter with the Laplacian into one edge detector called the Laplacian of Gaussian (LoG). They provided the following strong argument for the LoG:

1. Edge features in natural images can occur at various scales and different sharpness. The LoG operator can be applied to detecting multiscales of edges.
2. Some form of smoothing is essential for removing noise in many real-world images. The LoG is based on the filtering of the image with a Gaussian smoothing filter. The Gaussian filtering reduces the noise sensitivity problem when zero crossing is used for edge detection.
3. A zero crossing in the LoG is isotropic and corresponds to an extreme value in the first derivative.

Theoretically the convolution of the input image *f(x,y)* with a two-dimensional (2-D) Gaussian filter *G(x,y)* can be expressed as

$$S(x,y) = G(x,y) \times f(x,y), \text{ where } G(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x^2+y^2)}{\sigma^2}} \tag{14}$$

Then, the Laplacian (the second derivative) of the convolved image S is obtained by

$$\nabla^2 S(x,y) = \nabla^2[G(x,y) \times f(x,y)] = [\nabla^2 G(x,y)] \times f(x,y) \tag{15}$$

The computation order of Laplacian and convolution operations can be interchanged because these two operations are linear and shift invariant (11) as shown in Equation (15). Computing the Laplacian of the Gaussian filtered image $\nabla^2[G(x,y) \times f(x,y)]$ yields the same result with convolving the image with the Laplacian of the Gaussian filter ($[\nabla^2 G(x,y)] \times f(x,y)$). The Laplacian of the Gaussian filter $\nabla^2 G(x,y)$ is defined as follows:

$$\mathrm{LoG}(x,y) = \nabla^2 G(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{(x^2+y^2)}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{16}$$

where σ is the standard deviation, which also determines the spatial scale of the Gaussian. Figure 8(a) shows a one-dimensional (1-D) LoG profile and (b) shows a 2-D LoG profile. A 2-D LoG operation can be approximated by a convolution kernel. The size of the kernel is determined by the scale parameter σ. A discrete kernel that approximates the LoG with $\sigma = 1.4$ is shown in Fig. 9. In summary, edge detection using the LoG consists of the following two steps:

1. Convolve the image with a 2-D LoG mask at a selected scale σ.
2. Consider as edges only those zero-crossing pixels whose corresponding first derivatives are above a threshold. To find a zero crossing, we need to check four cases for the signs of the two opposing neighboring pixels: up/down, right/left, and the two diagonals.

Note that results of edge detection depend largely on the σ value of the LoG. The convolution mask is larger for a larger σ, which is suitable for detecting large-scale edges. Figure 10 shows edge maps generated by the

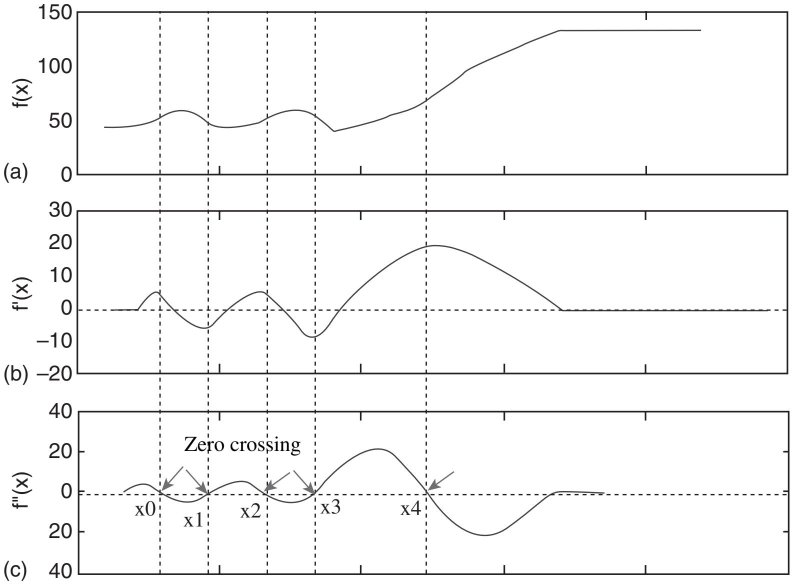


Figure 7. Illustration of spurious edges generated by zero crossing: (a) 1-D profile of a step edge with noise, (b) the first derivative of a step edge, and (c) the second derivative of a step edge. The zero crossing of *f(x)* creates several spurious edges points (x_0, x_1, x_2, and x_3).

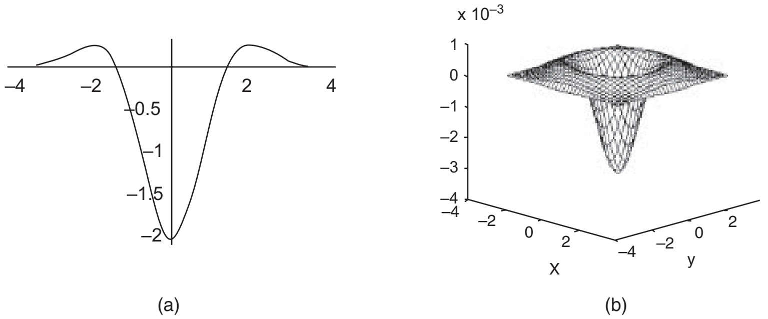


Figure 8. (a) a 1-D LoG profile and (b) a 2-D LoG profile.

LoG operator with various scales. Figure 10(b)–(e) show the LoG edge maps with $\sigma = 1.0, 1{,}5, 2.0$, and 2.5, respectively. In Fig. 10(f), two different scales $\sigma_1 = 0.7, \sigma_2 = 2.3$ are used, and the result is obtained by selecting the edge points that occurred in both scales. Figure 10(g) and (h) use the gradient magnitude threshold to reduce noise and break contours.

In comparison with the edge images based on the gradient methods in Figs. 3–5, the edge maps from the LoG is thinner than the gradient-based edge detectors. Because of the smoothing filtering in the LoG operator, its edge images (Fig. 10) are robust to noise; however, sharp corners are lost at the same time. Another interesting feature of the LoG is that it generates the closed-edge contours. However, spurious edge loops may appear in outputs and edge locations may shift at large scales (8). Both the LoG and the Laplacian edge detectors are isotropic filters, and it is not possible to extract directly the edge orientation information from these edge operators. Postprocesses such as nonmaxima suppression and hysteresis thresholding are not applicable.

Advanced Edge Detection Method—Canny Edge Detection

The Canny edge detector (3,16,20) is one of the most widely used edge detectors. In 1986, John Canny proposed the following three design criteria for an edge detector:

1. *Good localization*: Edge location founded by the edge operator should be as close to the actual location in the image as possible.
2. *Good detection with low error rate*: An optimal edge detector should respond only to true edges. In other words, no spurious edges should be found, and true edges should not be missed.
3. *Single response to a single edge*: The optimal edge detector should not respond with multiple edge pixels to the place where only a single edge exists.

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Figure 9. A discrete kernel that approximates the LoG with $\sigma = 1.4$.

Following these design criteria, Canny developed an optimal edge detection algorithm based on a step edge model with white Gaussian noise. The Canny edge detector involves the first derivative of a Gaussian smoothing filter with standard deviation σ. The choice of σ for the Gaussian filter controls the filter width and the amount of smoothing (11). Steps for the Canny edge detector are described as follows:

1. *Smoothing using Gaussian filtering*: A 2-D Gaussian filter *G(x,y)* by Equation (14) is applied to the image to remove noise. The standard deviation σ of this Gaussian filter is a scale parameter in the edge detector.
2. *Differentiation*: Compute the gradient G_x in the *x*-direction and G_y in the *y*-direction using any of the gradient operators (Roberts, Sobel, Prewitt, etc.). The magnitude G_m and direction θ_g of the gradient can be calculated as

$$G_m = \sqrt{G_x^2 + G_y^2}, \ \theta_g = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{16}$$

3. *Nonmaximum suppression:* Apply the nonmaximum suppression operation (see the section on “Nonmaximum Suppression” for details) to remove spurious edges.
4. *Hysteresis process*: The hysteresis step is the unique feature of the Canny edge operator (20). The hysteresis process uses two thresholds, a low threshold t_{low}, and a high threshold t_{high}. The high threshold is usually two or three times larger than the low threshold. If the magnitude of the edge value at the pixel *p* is lower than t_{1ow}, then the pixel *p* is marked immediately as a non-edge point. If the magnitude of the edge value at pixel *p* is greater than t_{high}, then it is immediately marked as an edge. Then any pixel that is connected to the marked edge pixel *p* and its

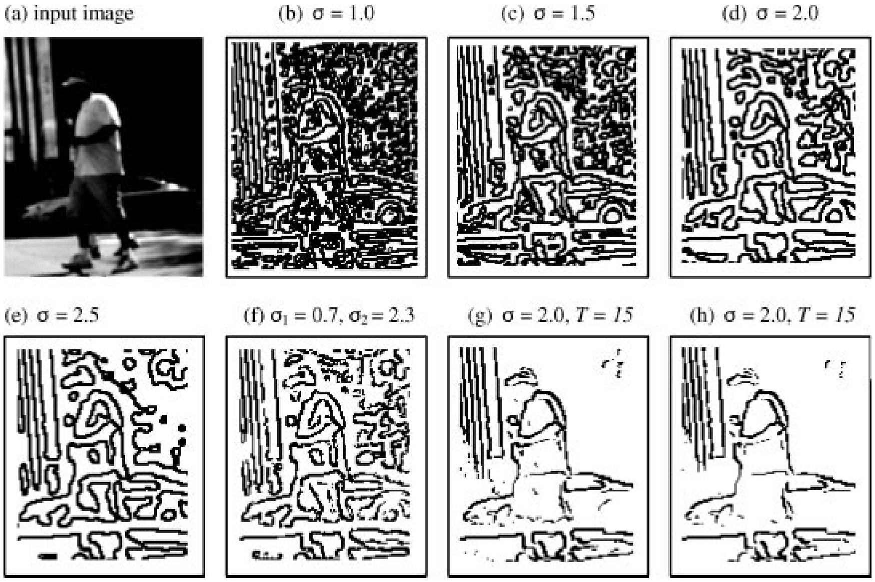


Figure 10. The LoG operator edge maps: (b) $\sigma = 1.0$, (c) $\sigma = 1.5$, (d) $\sigma = 2.0$, (e) $\sigma = 2.5$, (f) two scales used $\sigma_1 = 0.7$ and $\sigma_2 = 2.3$, (g) $\sigma = 2.0$ and $T = 15$, and (h) $\sigma = 2.0$ and $T = 20$.

magnitude of the edge value is greater than a low threshold t_{low}, it is also marked as an edge pixel. The edge marking of the hysteresis process is implemented recursively. This hysteresis producer generates the effects of reducing the broken edge contours and of removing the noise in adaptive manner (11).

The performance of the Canny operator is determined by three parameters: the standard deviation σ of the Gaussian filter and the two thresholds t_{low} and t_{high}, which are used in the hysteresis process. The noise elimination and localization error are related to the standard deviation σ of the Gaussian filter. If σ is larger, the noise elimination is increased but the localization error can also be more serious. Figures 11 and 12 illustrate the results of the Canny edge operator. Figure 11 demonstrate the effect of various σ values with the same upper and lower threshold values. As the σ value increases, noise pixels are removed but the sharp corners are lost at the same time. The effect of the hysteresis threshold is shown in Fig. 12. Figure 12(a) and (c) are edge maps with the hysteresis threshold. Edge maps with a hysteresis threshold have less broken contours than edge maps with one threshold: Compare Fig. 12(a) with Fig. 12(b). Table 1 summarized the computational cost of each edge operators used in gray images. It is noticed that the computational cost of the Canny operator is higher than those of other operators, but the Canny operator generates the more detailed edge map in comparison with the edge maps generated by other edge operators.

EDGE DETECTION IN COLOR IMAGES

What is the Color Image?

An image pixel in the color image is represented by a vector that is consisted of three components. Several different ways exist to represent a color image, such as RGB, YIQ,

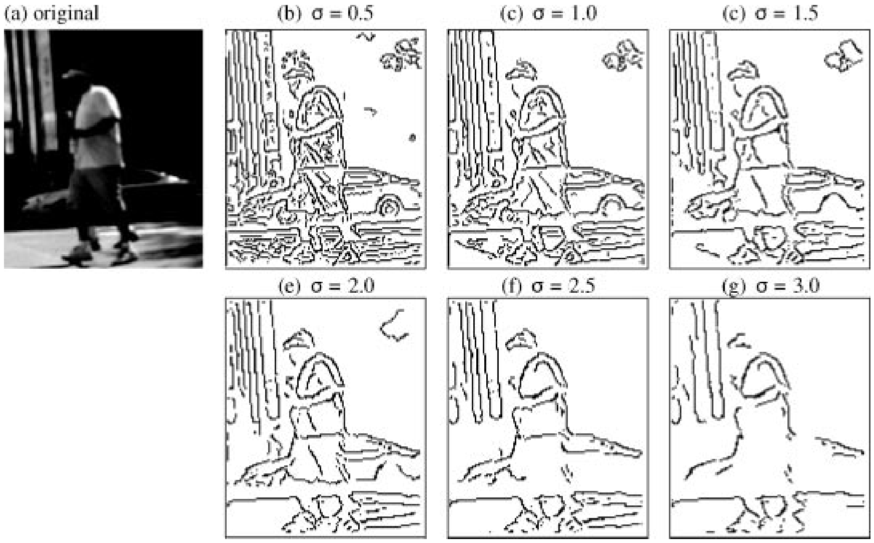


Figure 11. The effect of the standard deviation σ of the Gaussian smoothing filter in the Canny operator: (b) $\sigma = 0.5$, (c) $\sigma = 1.0$, (d) $\sigma = 1.5$, (e) $\sigma = 2.0$, (f) $\sigma = 2.5$, and (g) $\sigma = 3.0$, $T_{\mathrm{high}} = 100$, $T_{\mathrm{low}} = 40$.

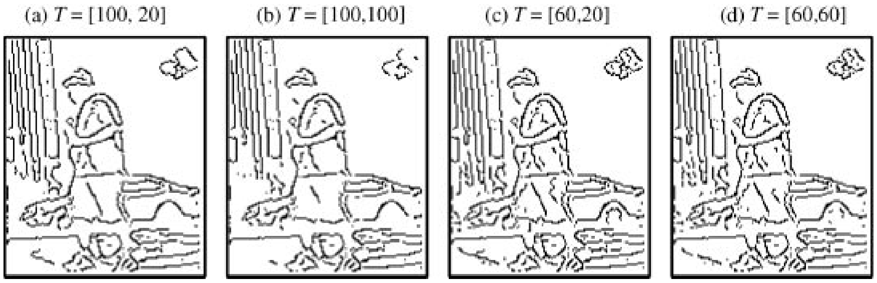

(a) $T = [100, 20]$ (b) $T = [100,100]$ (c) $T = [60,20]$ (d) $T = [60,60]$

Figure 12. The effect of the hysteresis threshold in the Canny operator: fixed $\sigma = 1.5$ (a) $t_{\mathrm{high}} = 100$, $t_{\mathrm{low}} = 20$; (b) $t_{\mathrm{high}} = t_{\mathrm{low}} = 100$; (c) $t_{\mathrm{high}} = 60$, $t_{\mathrm{low}} = 20$; (d) $t_{\mathrm{high}} = 60$, $t_{\mathrm{low}} = 60$.

HIS, and CIE L*u*v*. The RGB, the YIQ, and the HSI are the color models most often used for image processing (10).

The commonly known RGB color model consists of three colors components: Red(R), Green (G) and Blue (B). The RGB color model represents most closely to the physical sensors for colored light in most color CCD sensors (21). RGB is a commonly used color model for the digital pictures acquired by digital cameras (22). The three components in the RGB model are correlated highly, so if the light changes, all three components are changed accordingly (23).

The YIQ color space is the standard for color television broadcasts in North America. The YIQ color space is obtained from the RGB color model by linear transformation as shown below (10):

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{17}$$

In the YIQ color space, Y measures the luminance of the color value and I and Q are two chromatic components called *in-phase* and *quadrature*. The main advantage of YIQ color model in image processing is that luminance (Y) and two chromatic components (I and Q) are separated (10,24).

The HSI (hue, saturation, intensity), also known as HSB (hue, saturation, brightness), and its generalized form HSV (hue, saturation, value) models are also used frequently in color image processing (24–26). The HSI model corresponds more accurately to the human perception of color qualities. In the HSI model, hue, the dominant color, is represented by an angle in a color circle where primary colors are separated by 120° with Red at 0°, Green at 120°, and Blue at 240°. Saturation is the purity of the color. The high saturated values are assigned for pure spectral colors and the low values for the mixed shades. The intensity is associated with the overall brightness of a pixel (21).

Color Edge Detection versus Gray-Level Edge Detection

The use of color in edge detection implies that more information for edges is available in the process, and the edge detection results should be better than those of the gray-level images. Novak and Shafer (27) found that 90% of edge pixels are about the same in edge images obtained from gray-level and from color images. But 10% of the edge pixels left are as undetected when gray-level information is used (28). It is because edge may not be detected in the intensity component in low contrast images, but it can be detected in the chromatic components. For some applications, these undetected edges may contain crucial information for a later processing step, such as edge-based image segmentation or matching (29). Figure 13 demonstrates the typical example of the differences between color edge detectors and gray-level edge detectors. The gray-level Sobel edge detector missed many (or all) real edge pixels because of the low contrast in the intensity component as shown in Fig. 13(c) and (d).

A color image is considered as a two-dimensional array *f(x,y)* with three components for each pixel. One major concern in color edge detection is the high computational complexity, which has increased significantly in comparison with the edge detection in gray value images (see Table 2 for the computational cost comparison between the color Sobel operator and the gray-level Sobel operator with various size of images).

Definition of Edge in Color Images

The approaches for detecting edges in color images depend on the definition of an edge. Several definitions have been

Table 1. Computational cost comparison

		Image Size		
Computational Time (Seconds)	680 by 510	1152 by 864	1760 by 1168	
	Robert	0.032	0.125	0.219
	Prewitt	0.062	0.234	0.422
	Sobel	0.047	0.187	0.343
Gray image	Robert+NMS*	0.141	0.438	0.1
Edge operator	Prewitt+NMS*	0.14	0.516	1.188
	Sobel+NMS*	0.109	0.531	1.234
	LOG	0.5	1.453	2.984
	Canny	0.469	1.531	3.172

*NMS = Nonmaximum suppression.

Table 2. Computational cost comparison: the gray-level Sobel edge detector versus the color-Sobel edge detector

	Image Size		
Computational time (Seconds)	680 by 510	1152 by 864	1760 by 1168
Gray-level Sobel operator	0.047	0.187	0.343
Color Sobel operator	0.172	0.625	1.11

proposed, but the precise definition of color edge has not been established for color images so far (30). G. S. Robinson (24) defined a color edge as the place where a discontinuity occurs in the image intensity. Under this definition, edge detection would be performed in the intensity channel of a color image in the HIS space. But this definition provides no explanation of possible discontinuities in the hue or saturation values. Another definition is that an edge in a color image is where a discontinuity occurs in one color component. This definition leads to various edge detection methods that perform edge detection in all three color components and then fuses these edges to an output edge image (30). One problem facing this type of edge detection methods is that edges in each color component may contain inaccurate localization. The third definition of color edges is based on the calculation of gradients in all three color components. This type of multidimensional gradient methods combines three gradients into one to detect edges. The sum of the absolute values of the gradients is often used to combine the gradients.

Until now, most color edge detection methods are based on differential grayscale edge detection methods, such as finding the maximum in the first derivative or zero crossing in the second derivative of the image function. One difficulty in extending these methods to the color image originates from the fact that the color image has vector values. The monochromatic-based definition lacks the consideration about the relationship among three color components. After the gradient is calculated at each component, the question of how to combine the individual results remains open (31).

Because pixels in a color image are represented by a vector-valued function, several researchers have proposed vector-based edge detection methods (32–36). Cumani (32,35) and Zenzo (36) defined edge points at the locations of directional maxima of the contrast function. Cumani suggested a method to calculate a local measure of directional contrast based on the gradients of the three color components.

Color Edge Detection Methods

Monochromatic-Based Methods. The monochromatic-based methods extend the edge detection methods developed for gray-level images to each color component. The results from all color components are then combined to generate the final edge output. The following introduces commonly used methods.

Method 1: the Sobel operator and multidimensional gradient method

(i) Apply the Sobel operator to each color component.

(ii) Calculate the mean of the gradient magnitude values in the three color components.

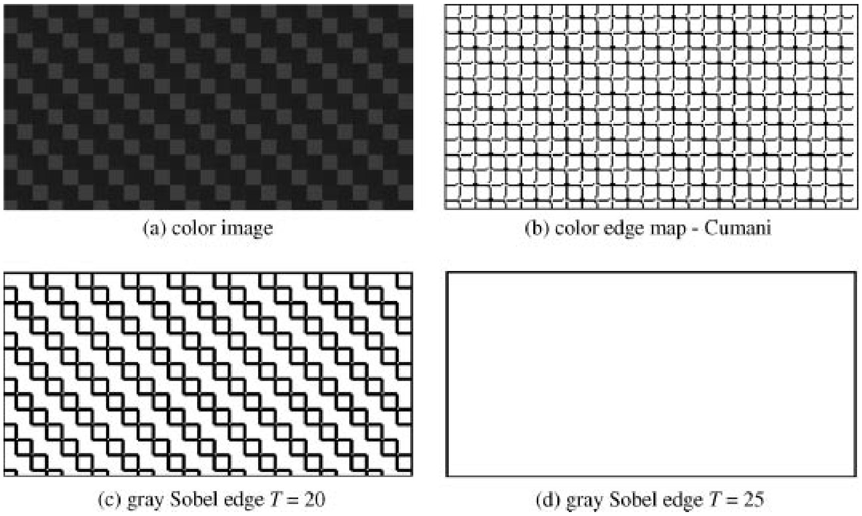
(iii) Edge exists if the mean of the gradient magnitude exceeds a given threshold (28,30).

Note the sum of the gradient magnitude in the three color components can also be used instead of the mean in Step ii).

Method 2: the Laplacian and fusion method

(i) Apply the Laplacian mask or the LoG mask to each color component.

(ii) Edge exists at a pixel if it has a zero crossing in at least one of the three color components (28,31).


(a) color image

(b) color edge map - Cumani

(c) gray Sobel edge $T = 20$

(d) gray Sobel edge $T = 25$

Figure 13. Color versus gray edge detectors: (a) color image [used with permission from John Owens at the University of California, Davis], (b) edge map generated by the color edge detector, and (c) and (d) edge map generated by the gray-level Sobel operator.

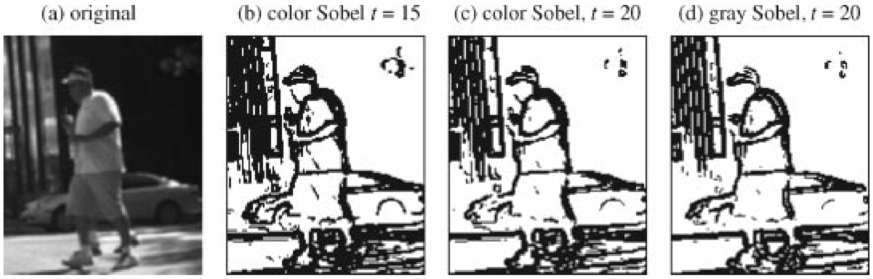


Figure 14. Experiments with the color Sobel operator: (a) color input; (b) and (c) color Sobel operator with $T = 15$, and $T = 20$, respectively; and (d) gray Sobel operator with $T = 20$.

Experimental results with multidimensional Sobel operator are shown in Fig. 14(b)–(c). The color Sobel operator generates the more detailed edge maps [Fig. 14(b) and (c)] compared with the edge map generated by the gray-level Sobel operator in Fig. 14(d). But the computational cost of the color Sobel operator increases three times more than the cost of the gray-level Sobel operator as shown in Table 2.

Vector-Based Methods. ***Color Variants of the Canny Edge Operator.*** Kanade (37) introduced an extension of the Canny edge detector (3) for edge detection in color images. Let a vector *C(r(x,y),g(x,y),b(x,y))* represent a color image in the RGB color space. The partial derivatives of the color vector can be expressed by a Jacobian matrix J as below:

$$J = \left(\frac{\partial C}{\partial x}, \frac{\partial C}{\partial y}\right) = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \\ \frac{\partial b}{\partial x} & \frac{\partial b}{\partial y} \end{bmatrix} = (G_x, G_y) \qquad (18)$$

The direction θ and the magnitude G_m of a color edge are given by

$$\tan(2\theta) = \frac{2 \cdot G_x \cdot G_y}{\|G_x\|^2 - \|G_y\|^2} \qquad (19)$$

$$G_m = \sqrt{\|G_x\|^2 \cos^2(\theta) + 2 \cdot G_x \cdot G_y \cdot \sin(\theta) \cdot \cos(\theta) + \|G_y\|^2 \sin^2(\theta)} \qquad (20)$$

where G_x, G_y are the partial derivatives of the three color components and $\|\cdot\|$ is a mathematical norm. Several variations exist based on different mathematical norms such as the L_1-norm (sum of the absolute value), L_2-norm (Euclidean distance), and L_∞-norm (maximum of the absolute value). Kanade (37) summarizes detailed experimental results obtained with various mathematical norms. After the edge direction and magnitude of the edge have been calculated for each pixel, the rest of the steps are the same with the Canny operator for gray-level images (see the section on the "advanced edge detection method"). Figure 15 shows edge maps generated by the color Canny operator with various scales and threshold values. The edge maps generated by the color Canny operator have the more detailed edge images compared with the edge map [Fig. 15(d)] generated by the gray- level Canny operator.

Cumani Operator. Cumani (32,35) proposed a vector-based color edge detection method that computes the zero crossings in the second directional derivatives of a color image. He defined a local measure of directional contrast based on the gradients of the image components. Then, edge points are detected by finding zero crossings of the first derivative of the contrast function in the direction of the maximal contrast.

Let a three-channel color image be represented by a two-dimensional vector field as follows:

$$f(x,y) = (\ f_1(x,y), \quad f_2(x,y), \quad f_3(x,y))$$

The squared local contrast S of *f(x,y)* at point $P = (x,y)$ in the direction of the unit vector $\vec{u} = (u_1, u_2)$ is

$$\begin{aligned} S(P,u) &= u^t J u = (u_1, u_2)\begin{pmatrix} E & F \\ F & G \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \\ &= Eu_1^2 + 2Fu_1u_2 + Gu_2^2 \end{aligned}$$

where

$$\begin{aligned} J &= \nabla f (\nabla f)^T = \begin{pmatrix} E & F \\ F & G \end{pmatrix}, E = \sum_{i=1}^{3}\left(\frac{\partial f_i}{\partial x}\right)^2, \\ F &= \sum_{i=1}^{3}\left(\frac{\partial f_i}{\partial x}\right)\left(\frac{\partial f_i}{\partial y}\right), \text{ and } G = \sum_{i=1}^{3}\left(\frac{\partial f_i}{\partial y}\right)^2 \end{aligned} \qquad (21)$$

The maximal value λ of *S(P, u)* is

$$\lambda_{\pm} = (E + G)_z \pm \sqrt{\frac{(E - G)^2}{4} + F^2} \qquad (22)$$

This maximum λ occurs when $\vec{u}$ is the corresponding eigenvector (35)

$$\begin{aligned} \vec{u} &= \left(\sqrt{\frac{1 + C}{2}}, \sqrt{\frac{1 - C}{2}}\right), \text{ where } C \\ &= \frac{E - G}{\sqrt{(E - G)^2 + (2F)^2}} \end{aligned} \qquad (23)$$

Edge points are defined at the locations where λ has a local maximum along the direction of the unit vector $\vec{u}$. So the

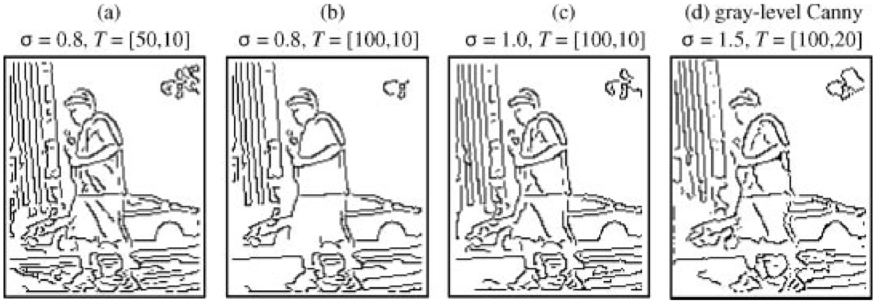


Figure 15. Experiments with the color Canny operator: color Canny edge maps (a) $\sigma = 0.8$, $t_{\text{high}} = 50$, $t_{\text{low}} = 10$; (b) $\sigma = 0.8$, $t_{\text{high}} = 100$, $t_{\text{low}} = 10$; (c) $\sigma = 1.0$, $t_{\text{high}} = 100$, $t_{\text{low}} = 10$; and (d) the edge map generated by the gray-level Canny operator $\sigma = 1.5$, $t_{\text{high}} = 100$, $t_{\text{low}} = 20$.

zero crossings of λ in the direction of $\vec{u}$ are candidates of edges (33). To find the zero crossing of λ, the directional derivative is defined as

$$\begin{aligned}\nabla\lambda_+ \cdot u_+ &= \nabla S(P, u_+) \cdot u_+ \\ &= \frac{\partial E}{\partial x} u_1^3 + \left(\frac{\partial E}{\partial y} + 2\frac{\partial F}{\partial x}\right) u_1^2 u_2 \\ &\quad + \left(2\frac{\partial F}{\partial y} + \frac{\partial G}{\partial x}\right) u_1 u_2^2 + \frac{\partial G}{\partial y} u_2^3 \qquad (24)\end{aligned}$$

where E, F, and G are defined in Equation (21). Finally, edge points are determined by computing the zero crossing of $\nabla\lambda_+ \cdot u_+$ and the sign of $\nabla\lambda_+ \cdot u_+$ along a curve tangent to u_+ at point P.

Cumani tested this edge detector with color images in the RGB space with the assumption that the Euclidean metric exists for the vector space (32). Figure 16(b), (c), and (d) show the edge detection results generated by the Cunami color edge operator at different scales. It seems that the Cunami color edge operator generated the more detailed edge images in comparison with the edge images generated by the monochromatic-based methods.

EDGE DETECTION METHOD IN RANGE IMAGES

Range images are a special class of digital images. The range images encode the distance information from the sensor to the objects. Pixel values in the range images are related to the positions of surface geometry directly. Therefore, range images provide an explicit encoding of the local structure and geometry of the objects in the scene. Edge detection methods developed for intensity images mainly focused on the detection of step edges. In the range imagers, it is possible to detect correctly both the step edges and the roof edges because of the available depth information.

Hoffman and Jain (38) described three edge types in range images: step edges, roof edges, and smooth edges. Step edges are those composed pixels in which the actual depth values are significantly discontinuous as compared with their neighboring pixels. Roof edges are where the depth values are continuous, but the directions of the surface normal change abruptly. Smooth edges are related with discontinuities in the surface curvature. But smooth edges relatively seldom occur in range images. Step edges in range images can be detected with ordinary gradient edge operators, but roof edges are difficult to be detected (39). Thus, an edge detection method for a range image must take into account these two types of edges such as discontinuities in depth and discontinuities in the direction of surface normal.

Edge Detection Using Orthogonal Polynomials

Besl and Jain (40,41) proposed a method that uses orthogonal polynomials for estimating the derivatives in range images. To estimate the derivatives, they used the locally fit discrete orthogonal polynomials with an unweighted least-squares technique. Using smooth second-order polynomials, range images were approximated locally. This method provided the smoothing effect and computational efficiency by obtaining the coefficient of the polynomials directly. But unweighted least squares could cause errors in image differentiation. To overcome this problem, a weighted least-squares approach was proposed by Baccar and Colleagues (42,43).

Extraction of Step Edges. The step edge detection method proposed by Baccar and Colleagues (42) is based on the use of locally fit discrete orthogonal polynomials with a weighted least-squares technique. For the weighted least-squares approximation *W(x)*, a one-dimensional Gaussian kernel of unit amplitude with zero mean and

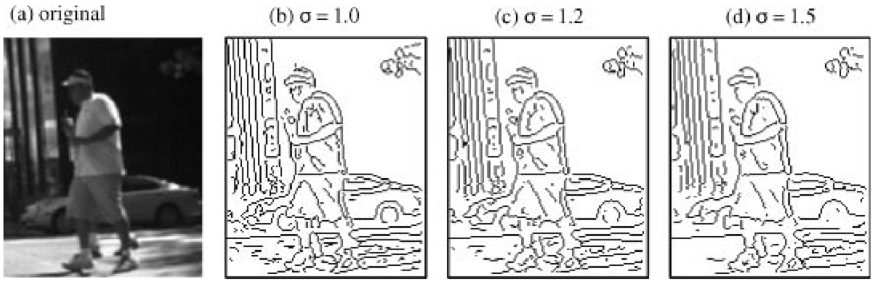


Figure 16. Experiments with the Cumani operator: (a) color input image and the Cumani edge maps (b) $\sigma = 1.0$, (c) $\sigma = 1.2$, and (d) $\sigma = 1.5$.

standard deviation σ is used. The two-dimensional Gaussian kernel at the center of the window can be represented by the product of *W(x)* and *W(y)*. Let $W(x) = e^{\frac{-x^2}{2\sigma^2}}$. Then a one-dimensional set of second-degree orthogonal polynomials φ_0, φ_1, φ_2 is defined as

$$\varphi_0(x) = 1,\ \varphi_1(x) = x,\ \varphi_2(x) = x^2 - A, \quad \text{where} \\ A = \frac{\sum xW(x)\varphi_1(x)\varphi_0(x)}{\sum W(x)\varphi_0^2(x)} \tag{25}$$

A locally approximated range image $\hat{r}(x,y)$ is calculated with a second-degree Gaussian weighted orthogonal polynomial as follows (42):

$$\begin{aligned} \hat{r}(x,y) &= \sum_{i+j\le 2} a_{ij}\varphi_i(x)\varphi_j(y) \\ &= a_{00} + a_{10}\varphi_1(x) + a_{01}\varphi_1(y) + a_{11}\varphi_1(x)\varphi_1(y) \\ &\quad + a_{20}\varphi_2(x) + a_{02}\varphi_2(y) \\ &= a_{00} + a_{10}x + a_{01}y + a_{11}xy + a_{20}(x^2 - a_{20}) \\ &\quad + a_{02}(y^2 - a_{02}) \\ &= a_{10}x + a_{01}y + a_{11}xy + a_{20}x^2 + a_{02}y^2 \\ &\quad + a_{00} - A(a_{02} + a_{20}) \end{aligned} \tag{26}$$

At a differentiable point of the surface, the quantity of the surface normal is defined as

$$a_{ij} = \frac{1}{\partial_i\partial_j}\sum_{x,y} r(x,y)W(x)W(y)\varphi_i(x)\varphi_j(y), \quad \text{where} \\ \partial_i = \sum_{x=-M}^{M} W(x)\varphi_i^2(x) \tag{27}$$

The partial derivatives of the approximated range image $\hat{r}(x,y)$ are defined by the following equations:

$$\hat{r}_x(x,y) = a_{10} + a_{11}y + 2a_{20}x, \\ \hat{r}_y(x,y) = a_{01} + a_{11}x + 2a_{02}\,y \tag{28}$$

At the center of the discrete window for $(x, y) = (0, 0)$, the partial derivatives are computed by

$$\hat{r}_x(0,0) = a_{10},\ \hat{r}_y(0,0) = a_{01} \tag{29}$$

The gradient magnitude at the center of this discrete window is $\sqrt{a_{10}^2 + a_{01}^2}$. The step edge image, $g_{step}(x,y)$, is obtained directly from the coefficients of the polynomials.

Extraction of Roof Edges. The roof edges are defined as the discontinuities in the orientation of the surface normal of objects in a range image. The quantity that defines the surface normal at a differentiable point of a surface is defined in Equation (27). The approximation to the surface normal, which is the angle between the two projections of the normal vector **n** on the (*x*, *z*)- and (*y*, *z*)-planes, is computed using

$$\gamma(x,y) = \tan^{-1}(r_y/r_x), \quad \text{where} \\ n = (-r_x, -r_y, 1)^T/(1 + r_x^2 + r_y^2)^{1/2} \tag{30}$$

The partial derivatives r_x and r_y of the function *r(x, y)* are calculated using the same Gaussian weighted least squares in Equation (28). The quantity $\gamma(x,y)$ represents the surface normal, and a 5 × 5 median filtering is applied to produce the final surface normal image. The roof edge image $g_{roof}(x,y)$ is computed from the final surface image by using the weighted Gaussian approach (42,43). The final edge map is generated after implementing a fusion step that combined step edge image $g_{step}(x,y)$ and roof edge image $g_{roof}(x,y)$ and a subsequent morphological step (42).

Edge Detection via Normal Changes

Sze et al. (44) as well as Mallet and Zhong (45) presented an edge detection method for range images based on normal changes. They pointed out that depth changes of a point in a range image with respect to its neighbors are not sufficient to detect all existent edges and that normal changes are much more significant than those of depth changes. Therefore, the step and roof edges in range images are identified by detecting significant normal changes.

Let $\vec{p}(u,v)$ be a point on a differentiable surface S and $\vec{p}(u,v) = (u, v, f(u,v))$. If we denote $\vec{p}_u$ and $\vec{p}_v$ as the partial derivatives of $\vec{p}(u,v)$ at u- and v-directions, respectively; then the partial derivatives of $\vec{p}(u,v)$ are given as follows (44):

$$\begin{aligned} \vec{p}_u &= \frac{\partial\vec{p}(u,v)}{\partial u} = (1, 0,\ f_u(u,v)) \\ \vec{p}_v &= \frac{\partial\vec{p}(u,v)}{\partial v} = (0, 1,\ f_v(u,v)) \end{aligned} \tag{31}$$

The normal of a tangent plane at $\vec{p}(u,v)$ is defined as

$$\vec{N}(u,v) = \frac{\vec{p}_u \times \vec{p}_v}{\|\vec{p}_u \times \vec{p}_v\|} \tag{32}$$

If we replace Equation (32) with Equation (31), the value of norm $\vec{N}(u,v)$ can be rewritten as below:

$$\begin{aligned} \vec{N}(u,v) &= \left(\frac{-f_u}{\sqrt{1+f_u^2+f_v^2}}, \frac{-f_v}{\sqrt{1+f_u^2+f_v^2}}, \frac{1}{\sqrt{1+f_u^2+f_v^2}}\right) \\ &= (n_1(u,v), n_2(u,v), n_3(u,v)), \end{aligned} \\ \text{where } f_u = \partial f(u,v)/\partial u \text{ and } f_v = \partial f(u,v)/\partial v \tag{33}$$

Steps for edge detection in range images via normal changes are summarized as follows (44):

1. *Calculate the normal of every point in a range image:* the partial derivatives are essential to derive the normal value at each data point as shown in Equation (33).

However, the partial derivative shown in Equation (33) cannot be computed directly because the range image data points are discrete. To calculate the normal on a set of discrete data points, the locally fit discrete orthogonal polynomials, originally proposed by Besl and Jain (40,41) and explained earlier, can be used. Other exiting methods are the orthogonal wavelet-based approach (46) and the nonorthogonal wavelet-based approach (45).

2. *Find the significant normal changes as edge point:* Using the dyadic wavelet transform proposed by Mallat and Zhong (45), the significant normal changes (or local extrema) are selected as edge points. The dyadic wavelet transform of a *f(x,y)* at scale 2^j along the x- and y-directions can be expressed by

$$\begin{aligned} W^1_{2j} f(x,y) &= f * (1/2^{2j})\varphi^1(x/2^j, y/2^j) \\ W^2_{2j} f(x,y) &= f * (1/2^{2j})\varphi^2(x/2^j, y/2^j) \end{aligned} \tag{34}$$

where $\varphi^1(x,y) = \partial\theta(x,y)/\partial x$, $\varphi^2(x,y) = \partial\theta(x,y)/\partial y$, and $\theta(x,y)$ is a smoothing function that satisfies the following conditions: Its integration over the full domain is equal to 1 and converges to 0 at infinity. The dyadic wavelet transformation of the vector of normal changes $\vec{N}(u,v)$ at scale 2^j is given by

$$W_j\vec{N}(u,v) = W^1_{2j}\vec{N}(u,v)du + W^2_{2j}\vec{N}(u,v)dv \tag{35}$$

where

$W^i_{2j}\vec{N}(u,v) = (W^i_{2j}n_1(u,v), W^i_{2j}n_2(u,v), W^i_{2j}n_3(u,v))$, $i = 1,2$.

Their associated weights can be the normal changes (or gradient) of $\vec{N}(u,v)$ along the du- and dv-directions. Two important values for edge detection can be calculated as follows: The magnitude of the dyadic wavelet transformation of $W_j\vec{N}(u,v)$ at scale 2^j is computed as

$$M_{2j}\overrightarrow{N}(u,v) = \sqrt{\|W^1_{2j}\vec{N}^{\rightarrow}(u,v)\|^2 + \|W^2_{2j}\overrightarrow{N}(u,v)\|^2} \tag{36}$$

and the angle with respect to the du-direction is

$$A_{2j}\overrightarrow{N}(u,v) = \text{argument}\left(\|W^1_{2j}\vec{N}(u,v)\| + i\|W^2_{2j}\overrightarrow{N}(u,v)\|\right) \tag{37}$$

Every point in the range image will be associated with two important values: magnitude of normal changes with respect to its certain neighbors and the direction tendency of the point (44). Edge points can be detected by thresholding the normal changes. Experimental results are provided for synthetic and real 240 × 240 range images in Ref 44. Three different methods such as quadratic surface fitting and orthogonal and nonorthogonal wavelet-based approaches are used to calculate the normal values for the comparison purpose. After the normal values are decided, the dyadic transforms proposed by Mallat and Zhong (45) are applied to detect the normal changes at every point in a range image. In their experiments, the nonorthogonal wavelet-based approach used to estimate the normal values generated the best results in comparison with the other methods.

CONCLUSION

Edge detection has been studied extensively in the last 50 years, and many algorithms have been proposed. In this article, we introduced the fundamental theories and the popular technologies for edge detection in grayscale, color, and range images. More recent edge detection work can be found in Refs.16, 30, 47 and 48. We did not touch on the topic of evaluating edge detectors. Interested readers can find such research work in Refs. 49–53.

BIBLIOGRAPHY

1. Z. He and M. Y. Siyal, Edge detection with BP neural networks, *Signal Processing Proc. ICSP'98*, **2**: 382–384, 1988.
2. E. R. Davies, Circularity- a new principle underlying the design of accurate edge orientation operators, *Image and Vision computing*, **2**(3): 134–142, 1984.
3. J. Canny, A computational approach to edge detection, *IEEE Trans. Pattern Anal. Machine Intell.*, **8**(6): 679–698, 1986.
4. Z. Hussain, *Digital Image Processing- Partial Application of Parallel Processing Technique*, Cluchester: Ellishorwood, 1991.
5. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, New York: Academic Press, 1982.
6. R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, Reading, MA: Addison-Wesley Publishing Company, 1992.
7. R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*, New York: McGraw-Hill, Inc., 1995.
8. Y. Lu and R. C. Jain, Behavior of edges in scale space, *IEEE Trans. on Pattern Anal. Machine Intell.*, **11**(4): 337–356, 1989.
9. M. Shah, A. Sood, and R. Jain, Pulse and staircase edge models, *Computer Vis. Graphics Image Proc.*, **34**: 321–343, 1986.
10. R. Gonzalez and R. Woods, *Digital Image Processing*, Reading, MA: Addison Wesley, 1992.
11. P. Mlsna and J. Rodriguez, *Gradient and Laplacian-Type Edge Detection, Handbook of Image and Video Processing*, New York: Academic Press, 2000.
12. E. R. Davies, *Machine Vision*, New York: Academic Press, 1997.
13. D. Marr and E. C. Hildreth, Theory of edge detection, *Proc. Roy. Society, London, Series B*, vol. **207**(1167): 187–217, 1980.
14. L. Kitchen and A. Rosenfeld, Non-maximum suppression of gradient magnitude makes them easier to threshold, *Pattern Recog. Lett.*, **1**(2): 93–94, 1982.
15. K. Paler and J. Kitter, Grey level edge thinning: a new method, *Pattern Recog. Lett.*, **1**(5): 409–416, 1983.
16. S. Wang, F. Ge, and T. Liu, Evaluation edge detection through boundary detection, *EURASIP J. Appl. Signal Process.*, **2006**: 1–15, 2006.
17. T. Pavlidis, *Algorithms for Graphics and Image Processing*, New York: Springer, 1982.
18. L. Kitchen and A. Rosenfeld, Non-maximum suppression of gradient magnitude makes them easier to threshold, *Pattern Recogn. Lett.*, **1**(2): 93–94, 1982.

19. J. Park, H. C. Chen, and S. T. Huang, A new gray-level edge thinning method, *Proc. of the ISCA 13th International Conference: Computer Applications in Industry and Engineering*, 2000, pp. 114–119.
20. J. R. Parker, Home page. University of Calgary. Available: http://pages.cpsc.ucalgary.co/~parker/501/edgedetect.pdf.
21. S. Wesolkowski and E. Jernigan, Color edge detection in RGB using jointly Euclidean distance and vector angle, *Vision Interface'99: Troi-Rivieres,* Canada, 1999, pp. 9–16.
22. H. D. Cheng, X. H. Jiang, Y. Sun, and J. Wang, Color image segmentation: advance and prospects, *Pattern Recogn.*, **34**: 2259–2281, 2001.
23. M. Pietikäinen, S. Nieminen, E. Marszalec, and T. Ojala, Accurate color discrimination with classification based on feature distributions, *Proc. 13th International Conference on Pattern Recognition*, Vienna, Austria, **3**, 1996, pp. 833–838.
24. G. Robinson, Color edge detection, *Optical Engineering*, **16**(5): 126–133, 1977.
25. T. Carron and P. Lambert, Color edge detector using jointly hue, saturation and intensity, ICIP 94, Austin, Texas, 1994, pp. 977–981.
26. P. Tsang and W. Tang, Edge detection on object color, *IEEE International Conference on Image Processing, C*, 1996, pp. 1049–1052.
27. C. Novak and S. Shafer, Color edge detection, *Proc. of DARPA Image Understanding Workshop, vol. I*, Los Angeles, CA, 1987, pp. 35–37.
28. A. Koschan, A comparative study on color edge detection, *Proc. 2nd Asian Conference on Computer Vision ACCV'95*, vol III, Singapore, 1995, pp. 574–578.
29. J. Fan, W. Aref, M. Hacid, and A. Elmagarmid, An improved isotropic color edge detection technique, *Pattern Recogn. Lett.*, **22**: 1419–1429, 2001.
30. A. Koshen and M. Abidi, Detection and classification of edges in color images, *IEEE Signal Proc. Mag.*, Jan: 64–73, 2005.
31. T. Huntsberger and, M. Descalzi, Color edge detection, *Pattern Recogn. Lett.*, **3**: 205–209, 1985.
32. A. Cumani, Edge detection in multispectral images, *CVGIP: Grap. Models Image Proc.*, **53**(I): 40–51, 1991.
33. L. Shafarenko, M. Petrou, and J. Kittler, Automatic watershed segmentation of randomly textured color images, *IEEE Trans. Image Process.*, **6**: 1530–1544, 1997.
34. Y. Yang, Color edge detection and segmentation using vector analysis, Master's Thesis, University of Toronto, Canada, 1995.
35. A. Cumani, Efficient contour extraction in color image, *Proc. of 3rd Asian Conference on Computer Vision*, vol. 1, 1998, pp. 582–589.
36. S. Zenzo, A note on the gradient of a multi-image, *CVGIP*, **33**: 116–125, 1986.
37. T. Kanade, Image understanding research at CMU, *Proc. Image Understading Workshop*, vol II, 1987, pp. 32–40.
38. R. Hoffman and A. Jain, Segmentation and classification of range image, *IEEE Trans. On PAMI 9-5*, 1989, pp. 643–649.
39. N. Pal and S. Pal, A review on Image segmentation technique, *Pattern Recogn.*, **26**(9): 1277–1294, 1993.
40. P. Besl and R. Jain, Invariant surface characteristics for 3D object recognition in range images, *Comp. Vision, Graphics Image Image Process.*, **33**: 33–80, 1984.
41. P. Besl and R. Jain, Segmentation through variable-order surface fitting, *IEEE Trans. Pattern Anal. Mach. Intell.*, **10**(3): 167–192, 1988.
42. M. Baccar, L. Gee, R. Gonzalez, and M. Abidi, Segmentation of range images via data fusion and Morphological watersheds, *Pattern Recogn.*, **29**(10): 1673–1687, 1996.
43. R. G. Gonzalez, M. Baccar, and M. A. Abidi, Segmentation of range images via data fusion and morphlogical watersheds, *Proc. of the 8th Scandinavian Conf. on Image Analysis*, vol. 1, 1993, pp. 21–39.
44. C. Sze, H. Liao, H. Hung, K. Fan, and J. Hsieh, Multiscale edge detection on range images via normal changes, *IEEE Trans. Circuits Sys. II: Analog Digital Signal Process.*, vol. **45**(8): 1087–1092, 1998.
45. S. Mallat and S. Zhong, Characterization of signal from multiscale edges, *IEEE Trans. Pattern Anal. Machine Intell.*, **14**(7): 710–732, 1992.
46. J. W. Hsieh, M. T. Ko, H. Y. Mark Liao, and K. C. Fan, A new wavelet-based edge detector via constrained optimization, *Image Vision Comp.*, **15**: 511–527, 1997.
47. R. Zhang, G. Zhao, and L. Su, A new edge detection method in image processing, *Proceedings of ISCIT 2005*, 2005, pp. 430–433.
48. S. Konishi, A. L. Yuille, J. M. Coughlan, and S. C. Zhu, Statistical edge detection: learning and evaluating edge cues, *IEEE Trans. Pattern Anal. Machine Intell.*, **25**(1): 57–73, 2003.
49. M. C. Shin, D. B. Goldgof, K. W. Bowyer, and S. Nikiforou, Comparison of edge detection algorithms using a structure from motion task, *IEEE Trans. on System, Man, and Cyberne.* –Part B: Cybernetics, **31**(4): 589–601, 2001.
50. T. Peli and D. Malah, A study of edge detection algorithms, *Comput. Graph. Image Process.*, **20**(1): 1–21, 1982.
51. P. Papachristou, M. Petrou, and J. Kittler, Edge postprocessing using probabilistic relaxation, *IEEE Trans. Syst., Man, Cybern. B*, **30**: 383–402, 2000.
52. M. Basu, Gaussian based edge detection methods—a survey, *IEEE Trans. Syst., Man, Cybern.-part C: Appl. Rev.*, **32**(3): 2002.
53. S. Wang, F. Ge, and T. Liu, Evaluating Edge Detection through Boundary Detection, *EURASIP J. Appl. Signal Proc.*, Vol. 2006, pp. 1–15.

JUNG ME PARK
YI LU MURPHEY
University of Michigan—Dearborn
Dearborn, Michigan

F

FACE RECOGNITION TECHNIQUES

INTRODUCTION TO FACE RECOGNITION

Biometrics[1] is becoming a buzzword due to increasing demand for user-friendly systems that provide both secure and efficient services. Currently, one needs to remember numbers and/or carry IDs all the time, for example, a badge for entering an office building, a password for computer access, a password for ATM access, and a photo-ID and an airline ticket for air travel. Although very reliable methods of biometric personal identification exist, e.g., fingerprint analysis and iris scans, these methods rely on the cooperation of the participants, whereas a personal identification system based on analysis of frontal or profile images of the face is often effective without the participant's cooperation or knowledge. It is due to this important aspect, and the fact that humans carry out face recognition routinely, that researchers started an investigation into the problem of machine perception of human faces. In Fig. 1, we illustrate the face recognition task of which the important first step of detecting facial regions from a given image is shown in Fig. 2.

After 35 years of investigation by researchers from various disciplines (e.g., engineering, neuroscience, and psychology), face recognition has become one of the most successful applications of image analysis and understanding. One obvious application for face recognition technology (FRT) is law-enforcement. For example, police can set up cameras in public areas to identify suspects by matching their imagaes against a watch-list facial database. Often, low-quality video and small-size facial images pose significant challenges for these applications. Other interesting commercial applications include intelligent robots that can recognize human subjects and digital cameras that offer automatic focus/exposure based on face detection. Finally, image searching techniques, including those based on facial image analysis, have been the latest trend in the booming Internet search industry. Such a wide range of applications pose a wide range of technical challenges and require an equally wide range of techniques from image processing, analysis, and understanding.

The Problem of Face Recognition

Face perception is a routine task of human perception system, although building a similar robust computer system is still a challenging task. Human recognition processes use a broad spectrum of stimuli, obtained from many, if not all, of the senses (visual, auditory, olfactory, tactile, etc.).

In many situations, contextual knowledge is also applied (e.g., the context plays an important role in recognizing faces in relation to where they are supposed to be located). However, the human brain has its limitations in the total number of persons that it can accurately "remember." A key advantage of a computer system is its capacity to handle large numbers of facial images.

A general statement of the problem of the machine recognition of faces can be formulated as follows: Given still or video images of a scene, identify or verify one or more persons in the scene using a stored database of faces. Available collateral information, such as race, age, gender, facial expression, or speech, may be used to narrow the search (enhancing recognition). The solution to the problem involves face detection (recognition/segmentation of face regions from cluttered scenes), feature extraction from the face regions (eyes, nose, mouth, etc.), recognition, or identification (Fig. 3).

Brief Development History

The earliest work on face recognition can be traced back at least to the 1950s in psychology (4) and to the 1960s in the engineering literature (5). Some of the earliest studies include work on facial expression of emotions by Darwin (6) [see also Ekman (7) and on facial profile-based biometrics by Galton (8)]. But research on automatic machine recognition of faces really started in the 1970s after the seminal work of Kanade (9) and Kelly (10). Over the past 30 years, extensive research has been conducted by psychophysicists, neuroscientists, and engineers on various aspects of face recognition by humans and machines.

Psychophysicists and neuroscientists have been concerned with issues such as whether face perception is a dedicated process [this issue is still being debated in the psychology community(11,12)], and whether it is done holistically or by local feature analysis. With the help of powerful engineering tools such as functional MRI, new theories continue to emerge (13). Many of the hypotheses and theories put forward by researchers in these disciplines have been based on rather small sets of images. Nevertheless, many of the findings have important consequences for engineers who design algorithms and systems for the machine recognition of human faces.

Until recently, most of the existing work formulates the recognition problem as recognizing 3-D objects from 2-D images. As a result, earlier approaches treated it as a 2-D pattern recognition problem. During the early and middle 1970s, typical pattern classification techniques were used that measured attributes of features (e.g., the distances between important points) in faces or face profiles (5,9,10). During the 1980s, work on face recognition remained largely dormant. Since the early 1990s, research interest in FRT has grown significantly. One can attribute this growth to several reasons: the increase in interest in commercial opportunities, the availability of real-time hardware, and the emergence of surveillance-related applications.

[1] Biometrics: the study of automated methods for uniquely recognizing humans based on one or more intrinsic physical or behavior traits.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

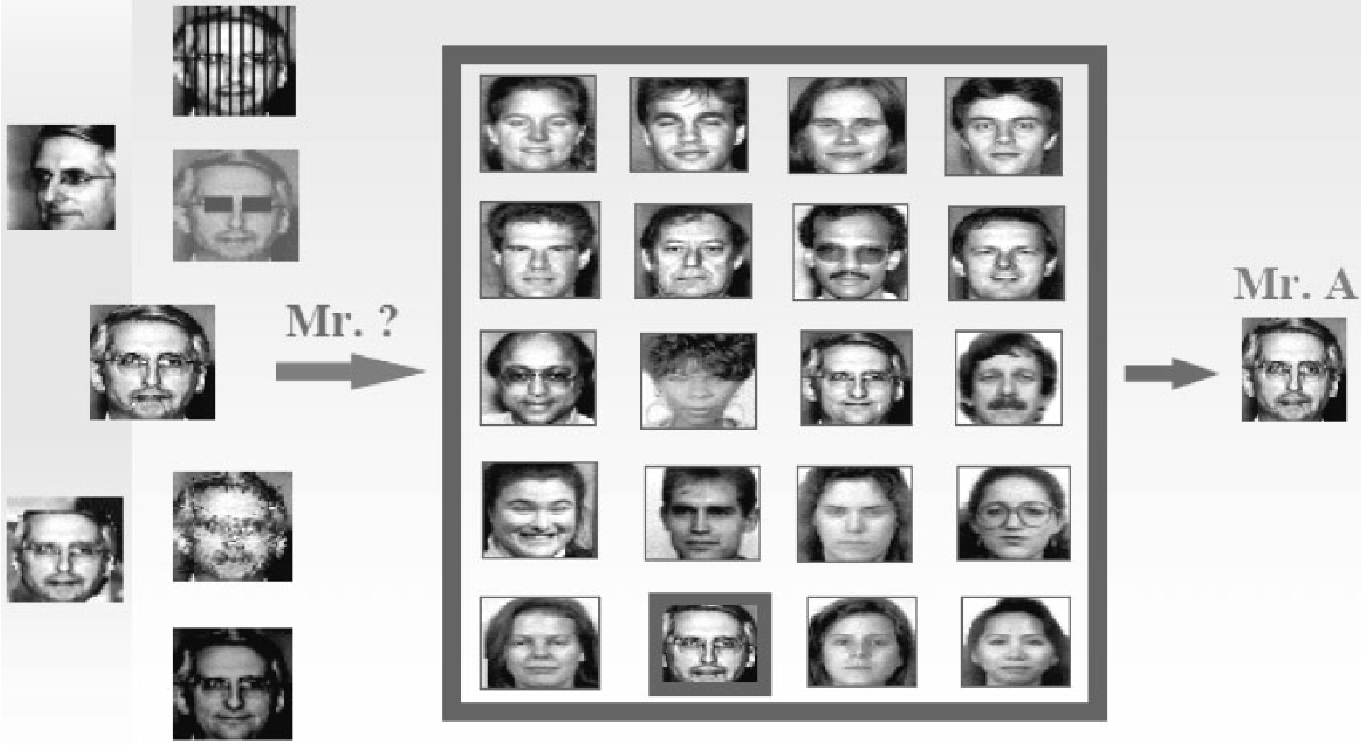


Figure 1. An illustration of the face recognition task (1): given an input facial image (left column: many variants of the facial image are used to illustrate image appearance change due to natural variations in lighting and pose, and electronic modifications that simulate more complex variations), matching it against a database of facial images (center column), and finally outputting the matched database image and/or the ID of the input image (right column).

Over the past 18 years, research has focused on how to make face recognition systems fully automatic by tackling problems such as localization of a face in a given image or a video clip and by extracting features such as eyes, mouth, and so on. Meanwhile, significant advances have been made in the design of classifiers for successful face recognition. Among appearance-based holistic approaches, eigenfaces (14,15) and Fisherfaces (16–18) have proved to be effective in experiments with large databases. Feature-based graph matching approaches (19) have also been successful. Compared with holistic approaches, feature-based methods are less sensitive to variations in illumination and viewpoint and to inaccuracy in face localization. However, the feature extraction techniques needed for this type of approach are still not sufficiently reliable or accurate (20).

During the past 8–15 years, much research has been concentrated on video-based face recognition. The still image problem has several inherent advantages and disadvantages. For applications such as airport surveillance, the automatic location and segmentation of a face could pose serious challenges to any segmentation algorithm if only a static picture of a large, crowded area is available. On the other hand, if a video sequence is available, segmentation of a moving person can be accomplished more easily using motion as a cue. In addition, a sequence of images might help to boost the recognition performance if we can use all these images effectively. But the small size and low image quality of faces captured from video can increase significantly the difficulty in recognition.

More recently, significant advances have been made on 3-D based face recognition. Although it is known that face recognition using 3-D images has many advantages than face recognition using a single or sequence of 2-D images, no serious effort was made for 3-D face recognition until recently. This delay was mainly caused by the feasibility, complexity, and computational cost to acquire 3-D data in real-time. Now, the availability of cheap, real-time 3-D sensors (21) makes it much easier to apply 3-D face recognition.

Recognizing a 3-D object from its 2-D images poses many challenges. The illumination and pose problems are two prominent issues for appearance-based or image-based approaches (22). Many approaches have been proposed to handle these issues, and the key here is to model the 3-D geometry and reflectance properties of a face. For example, 3-D textured models can be built from given 2-D images, and the images can then be used to synthesize images under various poses and illumination conditions for recognition or animation. By restricting the image-based 3-D object modeling to the domain of human faces, fairly good reconstruction results can be obtained using the state-of-the-art algorithms. Other potential applications in which modeling is crucial includes computerized aging, where an appropriate model needs to be built first and then a set of model parameters are used to create images that simulate the aging process.

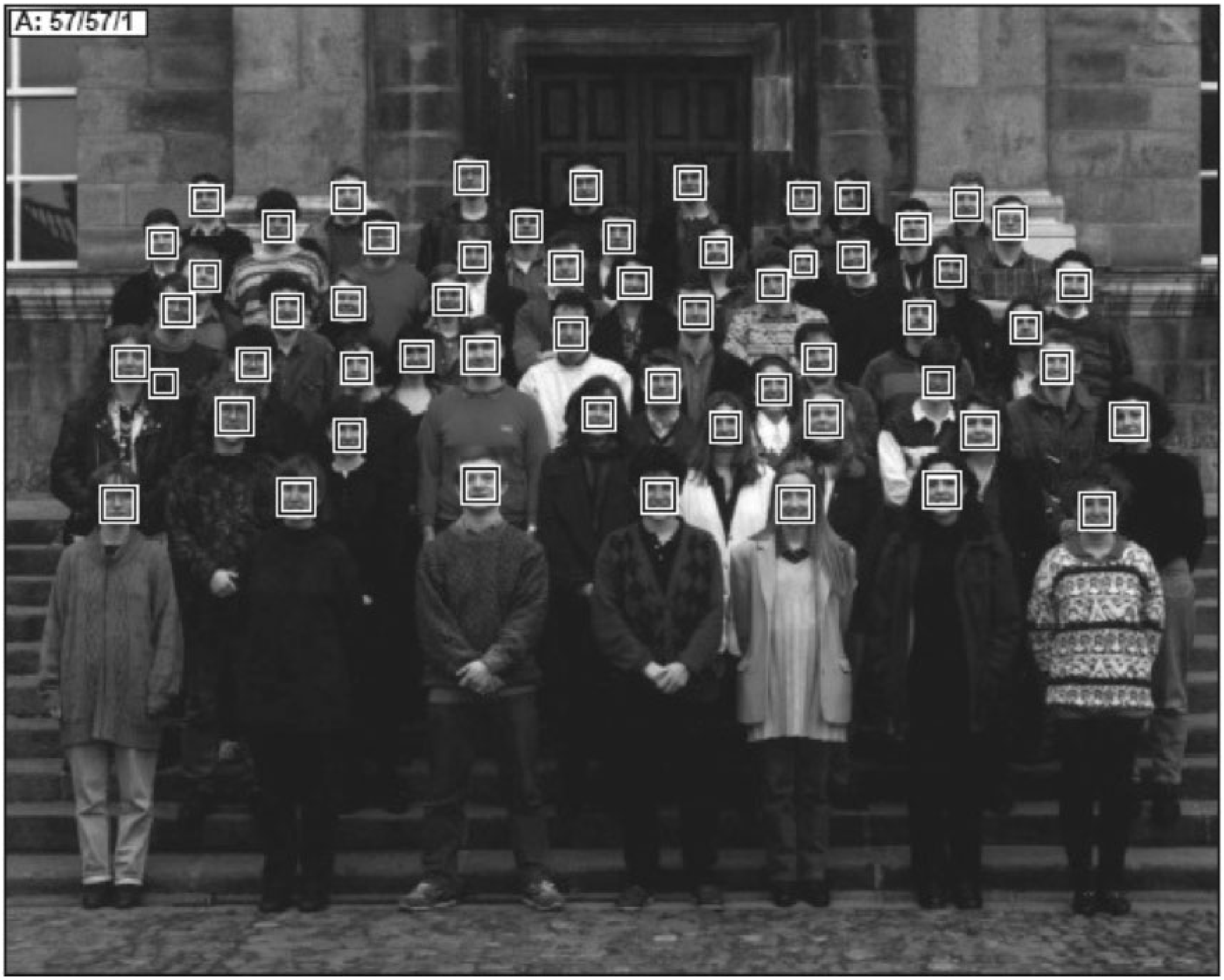


Figure 2. Detection/Segmentation/Recognition of facial regions from an image (2).

Methods for Machine Recognition of Faces

As illustrated in Fig. 4, the problem of automatic face recognition involves three key steps/subtasks:

1. Detection and coarse normalization of faces
2. Feature extraction and accurate normalization of faces
3. Identification and/or verification

Sometimes, different subtasks are not totally separated. For example, facial features (eyes, nose, mouth) are often used for both face recognition and face detection. Face detection and feature extraction can be achieved simultaneously as indicated in Fig. 4. Depending on the nature of the application, e.g., the sizes of the training and testing databases, clutter and variability of the background, noise, occlusion, and speed requirements, some subtasks can be very challenging. A fully automatic face recognition system must perform all three subtasks, and research on each subtask is critical. This is not only because the techniques used for the individual subtasks need to be improved, but also because they are critical in many different applications (Fig. 3). For example, face detection is needed to initialize face tracking, and extraction of facial features is needed for recognizing human emotion, which in turn is essential in human–computer interaction (HCI) systems. Without considering feature locations, face detection is declared as successful if the presence and rough location of a face has been correctly identified.

Face Detection and Feature Extraction

Segmentation/Detection. Up to the mid-1990s, most work on segmentation was focused on single-face segmentation from a simple or complex background. These approaches included using a whole-face template, a deformable feature-based template, skin color, and a neural network.

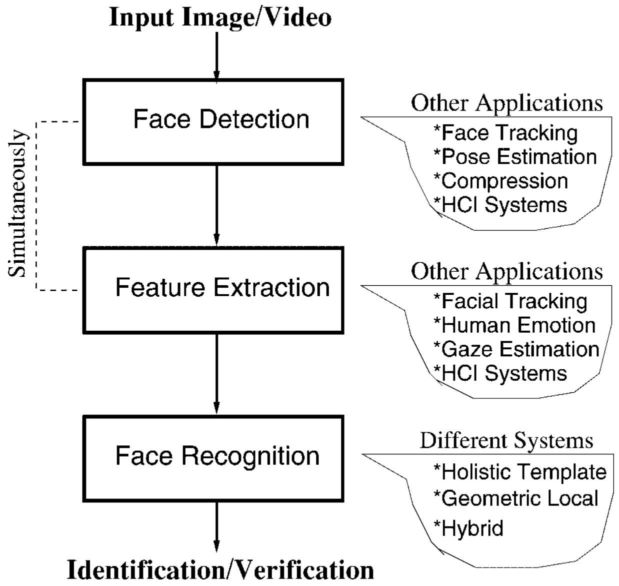


Figure 3. Configuration of a generic face recognition/processing system. We use a dotted line to indicate cases when both face detection and feature extraction work together to achieve accurate face localization and reliable feature extraction [e.g. (3)].

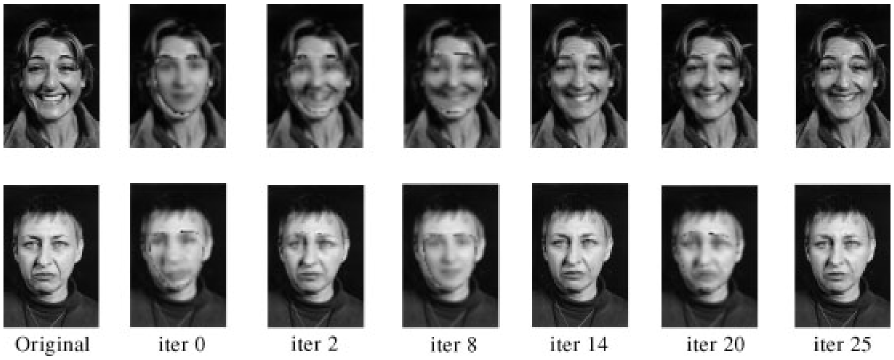


Figure 4. Mutiresolution seach from a displaced position using a face model (30).

Significant advances have been made in recent years in achieving automatic face detection under various conditions. Compared with feature-based methods and template-matching methods, appearance, or image-based methods (2, 23) that train machine systems on large numbers of samples have achieved the best results (refer to Fig. 4). This may not be surprising since complicated face objects are different from non-face objects, although they are very similar to each other. Through extensive training, computers can be good at detecting faces.

Feature Extraction. The importance of facial features for face recognition cannot be overstated. Many face recognition systems need facial features in addition to the holistic face, as suggested by studies in psychology. It is well known that even holistic matching methods, e.g., eigenfaces (15) and Fisherfaces (16), need accurate locations of key facial features such as eyes, nose, and mouth to normalize the detected face (24–26).

Three types of feature extraction methods can be distinguished:

1. Generic methods based on edges, lines, and curves
2. Feature-template-based methods that are used to detect facial features such as eyes
3. Structural matching methods that take into consideration geometrical constraints on the features

Early approaches focused on individual features; for example, a template-based approach is described in Ref. 27 to detect and recognize the human eye in a frontal face. These methods have difficulty when the appearances of the features change significantly, e.g., closed eyes, eyes with glasses, or open mouth. To detect the features more reliably, recent approaches use structural matching methods, for example, the active shape model (ASM) that represents any face shape (a set of landmark points) via a mean face shape and principle components through training (3).

Compared with earlier methods, these recent statistical methods are much more robust in terms of handling variations in image intensity and in feature shape. The advantages of using the so-called “analysis through synthesis”

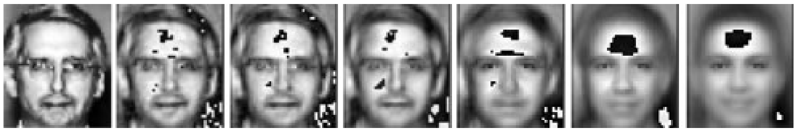


Figure 5. Original image [size 48 × 42 (i.e., 2016)] and the reconstructed image using 300, 200, 100, 50, 20, and 10 leading components, respectively (32).

approach come from the fact that the solution is *constrained* by a *flexible* statistical model. To account for texture variation, the ASM model has been expanded to statistical appearance models including a flexible appearance model (28) and an active appearance model (AAM)(29). In Ref. 29, the proposed AAM combined a model of shape variation (i.e., ASM) with a model of the appearance variation of shape-normalized (shape-free) textures. A training set of 400 images of faces, each labeled manually with 68 landmark points and approximately 10,000 intensity values sampled from facial regions were used. To match a given image with a model, an optimal vector of parameters (displacement parameters between the face region and the model, parameters for linear intensity adjustment, and the appearance parameters) are searched by minimizing the difference between the synthetic image and the given image. After matching, a best-fitting model is constructed that gives the locations of all the facial features so that the original image can be reconstructed. Figure 4 illustrates the optimization/search procedure to fit the model to the image.

Face Recognition

As suggested, three types of FRT systems have been investigated:recognition based on still images, recognition based on a sequence of images, and, more recently, recognition based on 3-D images. All types of FRT technologies have their advantages and disadvantages. For example, video-based face recognition can use temporal information to enhance recognition performance. Meanwhile, the quality of video is low and the face regions are small under typical acquisition conditions (e.g., in surveillance applications). Rather than presenting all three types of FRT systems, we focus on still-image-based FRT systems that form the foundations for machine recognition of faces. For details on all three types of FRT systems, please refer to a recent review article (the first chapter in Ref. 31).

Face recognition is such an interesting and challenging problem that it has attracted researchers from different fields: psychology, pattern recognition, neural networks, computer vision, and computer graphics. Often, a single system involves techniques motivated by different principles. To help readers that are new to this field, we present a class of linear projection/subspace algorithms based on image appearances. The implementation of these algorithms is straightforward, yet they are very effective under constrained situations. These algorithms helped to revive the research activities in the 1990s with the introduction of eigenfaces (14,15) and are still being researched actively for continuous improvements.

Eigenface and the Projection-Based Appearance Methods. The first successful demonstration of the machine recognition of faces was made by Turk and Pentland (15) using eigenpictures (also known as eigenfaces) for face detection and identification. Given the eigenfaces, every face in the database is represented as a vector of weights obtained by projecting the image into a subset of all eigenface components (i.e., a subspace) by a simple inner product operation. When a new test image whose identification is required is given, the new image is represented by its vector of weights. The test image is identified by locating the image in the database whose weights are the closest (in Euclidean distance) to the weights of the test image. By using the observation that the projection of a facial image and a nonface image are different, a method to detect the presence of a face in a given image is obtained. Turk and Pentland illustrate their method using a large database of 2500 facial images of 16 subjects, digitized at all combinations of three head orientations, three head sizes, and three lighting conditions.

In a brief summary, eigenpictures/eigenfaces are effective low-dimensional representations of facial images based on Karhunen–Loeve (KL) or principal component analysis projection (PCA)(14). Mathematically speaking, sample facial images (2-D matrix format) can be converted into vector representations (1-D format). After collecting enough sample vectors, one can perform statistical analysis (i.e., PCA) to construct new orthogonal bases and then can represent these samples in a coordinate system defined by these new bases. More specifically, mean-subtracted sample vectors x can be expressed as a linear combination of the orthogonal bases Φ_i (typically $m \ll n$):

$$\mathrm{x} = \sum_{i=1}^{n} a_i \Phi_i \approx \sum_{i=1}^{m} a_i \Phi_i \tag{1}$$

via solving an eigenproblem

$$C\Phi = \Phi\Lambda \tag{2}$$

where C is the covariance matrix for input x and Λ is a diagonal matrix consisting of eigenvalues λ_i. The main point of eigenfaces is that it is an efficient representation of the original images (i.e., from n coefficients to m coefficient). Figure 5 shows a real facial image and several reconstructed images based on several varying number of leading principal components Φ_i that correspond to the largest eigenvalues λ_i. As can be seen from the plots, 300 leading principal components are sufficient to represent the original image of size 48 × 42 (=2016).[2]

[2] Please note that the reconstructed images are obtained by converting from 1-D vector format back to 2-D matrix format and adding back the average/mean facial image.

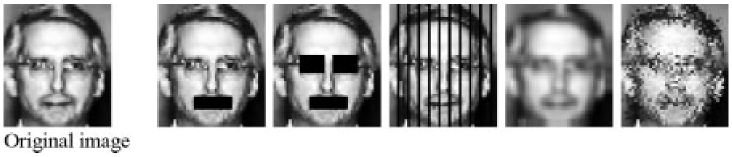


Figure 6. Electronically modified images that have been identified correctly using eigenface representation (18).

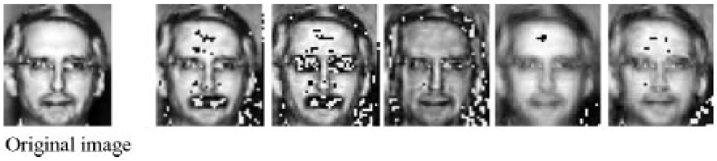


Figure 7. Reconstructed images using 300 PCA projection coefficients for electronically modified images (Fig. 7) (26).

Another advantage of using such a compact representation is the reduced sensitivity to noise. Some of the noise could be caused by small occlusions as long as the *topologic structure* does *not* change. For example, good performance against blurring, partial occlusion has been demonstrated in many eigenpicture based systems and was also reported in Ref. 18 (Fig. 6), which should not come as a surprise because the images reconstructed using PCA are much better than the original distorted images in terms of the global appearance (Fig. 7).

In addition to eigenfaces, other linear projection algorithms exist, including ICA (independent component analysis) and LDA/FLD (linear discriminant analysis/Fisher's linear discriminant analysis) to name a few. In all these projection algorithms, classification is performed 1) first by projecting the input x into a subspace via a projection/basis matrix $\mathbf{P}_{roj}$ ($\mathbf{P}_{roj}$ is Φ for eigenfaces, W for Fisherfaces with pure LDA projection, and $W\Phi$ for Fisherfaces with sequential PCA and LDA projections; these three bases are shown for visual comparison in Fig. 8),

$$\mathbf{z} = \mathbf{P}_{\mathrm{ro}j}\mathbf{x} \tag{3}$$

2) then by comparing the projection coefficient vector $\mathbf{z}$ of the input with all the pre-stored projection vectors of labeled classes to determine the input class label. The vector comparison varies in different implementations and can influence the system's performance dramatically (33).

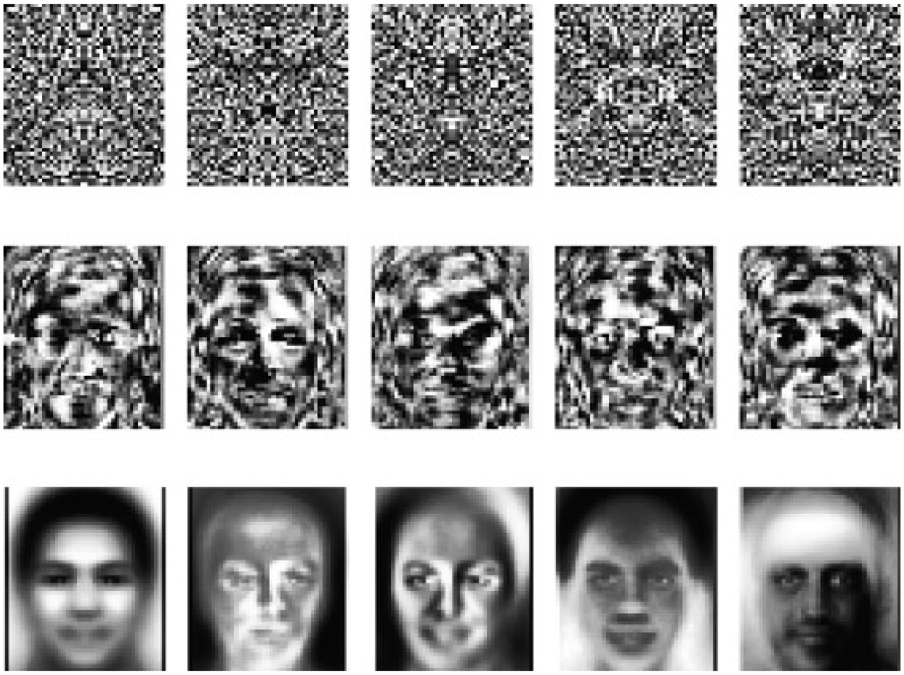


Figure 8. Different projection bases constructed from a set of 444 individuals, where the set is augmented via adding noise and mirroring. (Improved reconstruction for facial images outside the training set using an extended training set that adds mirror-imaged faces was suggested in Ref. 14.).The first row shows the first five pure LDA basis images W, the second row shows the first five subspace LDA basis images $W\Phi$, the average face and first four eigenfaces Φ are shown on the third row (18).

For example, PCA algorithms can use either the angle or the Euclidean distance (weighted or unweighted) between two projection vectors. For LDA algorithms, the distance can be unweighted or weighted.

Face recognition systems using LDA/FLD (called Fisherfaces in Ref. 16) have also been very successful (16,17,34). LDA training is carried out via scatter matrix analysis (35). For an M-class problem, the within-class and between-class scatter matrices S_w, S_b are computed as follows:

$$S_w = \sum_{i=1}^{M} Pr(w_i)C_i \tag{4}$$

$$S_b = \sum_{i=1}^{M} Pr(w_i)(\mathbf{m}_i - \mathbf{m}_0)(\mathbf{m}_i - \mathbf{m}_0)^T$$

where $Pr(\omega_i)$ is the prior class probability and usually is replaced by $1/M$ in practice with the assumption of equal priors. Here S_w is the *within-class scatter matrix*, showing the average scatter C_i of the sample vectors x of different classes ω_i around their respective means m_i: $C_i = E[(\mathrm{x}(\omega) - \mathrm{m}_i)\ (\mathrm{x}(\omega) - \mathrm{m}_i)^T|\omega = \omega_i]$. Similarly, S_b is the *between-class scatter matrix*, which represents the scatter of the conditional mean vectors m_i around the overall mean vector m_0. A commonly used measure to quantify the discriminatory power is the ratio of the determinant of the between-class scatter matrix of the projected samples to the determinant of the within-class scatter matrix: $\mathcal{J}(T) = |T^T S_b T|/|T^T S_w T|$. The optimal projection matrix W that maximizes $\mathcal{J}(T)$ can be obtained by solving a generalized eigenvalue problem:

$$S_b W = S_w W \Lambda_L \tag{5}$$

There are several ways to solve the generalized eigenproblem of Equation (5). One is to compute directly the inverse of S_w and solve a nonsymmetric (in general) eigenproblem for matrix $S_w^{-1}S_b$. But this approach is unstable numerically because it involves the direct inversion of a potentially very large matrix that probably is close to being singular. A stable method to solve this equation is to solve the eigen-problem for S_w first (32,35), [i.e., to remove the within-class variations (whitening)]. Because S_w is a real symmetric matrix, orthonormal W_w and diagonal Λ_w exist such that $S_w W_w = W_w \Lambda_w$. After whitening, the input $\mathbf{x}$ becomes $\mathbf{y}$:

$$\mathbf{y} = \Lambda_w^{-1/2}\mathbf{W}_w^T\mathbf{x} \tag{6}$$

The between-class scatter matrix for the new variable $\mathbf{y}$ can be constructed similar to Equation (5).

$$S_b^y = \sum_{i=1}^{M} Pr(\omega_i)(\mathbf{m}_i^y - \mathbf{m}_0^y)(\mathbf{m}_i^y - \mathbf{m}_0^y)^T \tag{7}$$

Now the purpose of FLD/LDA is to maximize the class separation of the now whitened samples $\mathbf{y}$, which leads to another eigenproblem: $S_b^y W_b = W_b \Lambda_b$. Finally, we apply the change of variables to $\mathbf{y}$:

$$\mathbf{z} = \mathbf{W}_b^T\mathbf{y} \tag{8}$$

Combining Equations (6) and (8), we have the following relationship: $\mathbf{z} = W_b^T \Lambda_w^{-1/2} W_w^T \mathbf{x}$ and $\mathbf{W}$ simply is

$$W = W_w \Lambda_w^{-1/2} W_b \tag{9}$$

To improve the performance of LDA-based systems, a regularized subspace LDA system that unifies PCA and LDA was proposed in Refs. 26 and 32. Good generalization capability of this system was demonstrated by experiments on new classes/individuals without retraining the PCA bases Φ, and sometimes even the LDA bases W. Although the reason for not retraining PCA is obvious, it is interesting to test the adaptive capability of the system by fixing the LDA bases when images from new classes are added.[3] The fixed PCA subspace of dimensionality 300 was trained from a large number of samples. An augmented set of 4056 mostly frontal-view images constructed from the original 1078 images of 444 individuals by adding noisy and mirrored images was used in Ref. 32. At least one of the following three characteristics separates this system from other LDA-based systems (16,34)): (1) the unique selection of the universal face subspace dimension, (2) the use of a weighted distance measure, and (3) a regularized procedure that modifies the within-class scatter matrix S_w. The authors selected the dimensionality of the universal face subspace based on the characteristics of the eigenvectors (face-like or not) instead of the eigenvalues (18), as is done commonly. Later, it was concluded in Ref. 36 that the global face subspace dimensionality is on the order of 400 for large databases of 5000 images. The modification of S_w into $S_w + \delta I$ has two motivations (1): first, to resolve the issue of small sample size (37) and second, to prevent the, significantly discriminative information[4] contained in the null space of S_w(38) from being lost.

To handle the non-linearity caused by pose, illumination, and expression variations presented in facial images, the above linear subspace methods have been extended to kernel faces (39–41) and tensorfaces (42).

Categorization of Still-Image Based FRT. Many methods of face recognition have been proposed during the past 30 years from researchers with different backgrounds. Because of this fact, the literature on face recognition is vast and diverse. To have a clear and high-level categorization, we follow a guideline suggested by the psychological study of how humans use holistic and local features. Specifically, we have the following categorization (see table 1 for more information):

[3] This makes sense because the final classification is carried out in the projection space z by comparison with pre-stored projection vectors with nearest-neighbor rule.

[4] The null space of S_w contains important discriminant information because the ratio of the determinants of the scatter matrices would be maximized in the null space.

1. **Holistic matching methods.** These methods use the whole face region as the raw input to a recognition system. One of the widely used representations of the face region is eigenpictures (14), which are based on principal component analysis.
2. **Feature-based (structural) matching methods.** Typically, in these methods, local features such as the eyes, nose, and mouth are first extracted and their locations and local statistics (geometric and/or appearance) are fed into a structural classifier.
3. **Hybrid methods.** Just as the human perception system uses both local features and the whole face region to recognize a face, a machine recognition system should use both. One can argue that these methods could offer potentially the best of the two types of methods

Within each of these categories, additional classification is possible. Using subspace analysis, many face recognition techniques have been developed: eigenfaces (15), which use a nearest-neighbor classifier; feature-line-based methods, which replace the point-to-point distance with the distance between a point and the feature line linking two stored sample points (46); Fisherfaces (16,18,34), which use Fisher's Linear Discriminant Analysis (57); Bayesian methods, which use a probabilistic distance metric (43); and SVM methods, which use a support vector machine as the classifier (44). Using higher-order statistics, independent component analysis is argued to have more representative power than PCA, and, in theory, it can provide better recognition performance than PCA (47). Being able to offer potentially greater generalization through learning, neural networks/learning methods have also been applied to face recognition. One example is the probabilistic decision-based neural network method (48) and the other is the evolution pursuit method (45).

Most earlier methods belong to the category of structural matching methods, which use the width of the head, the distances between the eyes and from the eyes to the mouth, and so on (10), or the distances and angles between eye corners, mouth extrema, nostrils, and chin top (9). Recently, a mixture-distance-based approach using manually extracted distances was reported (20). Without finding the exact locations of facial features, hidden markov model based methods use strips of pixels that cover the forehead, eye, nose, mouth, and chin (51,52). Reference 52 reported better performance than Ref. 51 by using the KL projection coefficients instead of the strips of raw pixels. One of the most successful systems in this category is the graph matching system (19), which is based on the Dynamic Link Architecture (58). Using an unsupervised learning method based on a self-organizing map, a system based on a convolutional neural network has been developed (53).

In the hybrid method category, we have the modular eigenface method (54), a hybrid representation based on PCA and local feature analysis (55), a flexible appearance model-based method (28), and a recent development (56) along this direction. In Ref. 54, the use of hybrid features by combining eigenfaces and other eigenmodules is explored: eigeneyes, eigenmouth, and eigennose. Although experiments show only slight improvements over holistic eigenfaces or eigenmodules based on structural matching, we believe that these types of methods are important and deserve further investigation. Perhaps many relevant problems need to be solved before fruitful results can be expected (e.g., how to arbitrate optimally the use of holistic and local features).

Many types of systems have been applied successfully to the task of face recognition, but they all have some advantages and disadvantages. Appropriate schemes should be chosen based on the specific requirements of a given task.

COMMERCIAL APPLICATIONS AND ADVANCED RESEARCH TOPICS

Face recognition is a fascinating research topic. On one hand, many algorithms and systems have reached a certain level of maturity after 35 years of research. On the other hand, the success of these systems is limited by the conditions imposed by many real applications. For example, automatic focus/exposure based on face detection has been built into digital cameras. However, recognition of face images acquired in an outdoor environment with changes in illumination and/or pose remains a largely unsolved problem. In other words, current systems are still far away from the capability of the human perception system.

Commercial Applications of FRT

In recent years, we have seen significant advances in automatic face detection under various conditions. Consequently, many commercial applications have emerged. For example, face detection has been employed for automatic exposure/focus in digital cameras, such as Powershot SD800 IS from Canon and FinePix F31fd from Fuji. These smart cameras can zero in automatically on faces, and photos will be properly exposed. In general, face detection technology is integrated into the camera's processor for increased speed. For example, FinePix F31fd can identify faces and can optimize image settings in as little as 0.05 seconds.

One interesting application of face detection technology is the passive driver monitor system installed on 2008 Lexus LS 600hL. The system uses a camera on the steering column to keep an eye on the driver. If he or she should be looking away from the road ahead and the pre-collision system detects something beyond the car (through stereo vision and radar), the system will sound a buzzer, flash a light, and even apply a sharp tap on the brakes.

Finally, the popular application of face detection technology is an image-based search of Internet or photo albums. Many companies (Adobe, Google, Microsoft, and many start-ups) have been working on various prototypes. Often, the bottle-neck for such commercial applications is the difficulty to recognize and to detected faces. Despite the advances, today's recognition systems have limitations. Many factors exist that could defeat these systems: facial expression, aging, glasses, and shaving.

Nevertheless, face detection/recognition has been critical for the success of intelligent robots that may provide

Table 1. Categorization of Still-Image-Based Face Recognition Techniques

Approach	Representative Work
Holistic methods	
Principal Component Analysis	
Eigenfaces	Direct application of PCA (15)
Probabilistic Eigenfaces	Two-class problem with prob. measure (43)
Fisherfaces/Subspace LDA	FLD on eigenspace (16,18,34)
SVM	Two-class problem based on SVM (44)
Evolution Pursuit	Enhanced GA learning (45)
Feature Lines	Point-to-line distance based (46)
ICA	ICA-based feature analysis (47)
Other Representations	
LDA/FLD	FLD/LDA on raw image (17)
PDBNN	Probabilistic decision-based NN (48)
Kernel faces	Kernel methods (39–41)
Tensorfaces	Multilinear analysis (42,49)
Feature-based methods	
Pure geometry methods	Earlier methods (9,10); recent methods (20,50)
Dynamic Link Architecture	Graph matching methods (19)
Hidden Markov Model	HMM methods (51,52)
Convolution Neural Network	SOM learning based CNN methods (53)
Hybrid methods	
Modular Eigenfaces	Eigenfaces and eigenmodules (54)
Hybrid LFA	Local feature method (55)
Shape-normalized	Flexible appearance models (28)
Component-based	Face region and components (56)

important services in the future. Prototypes of intelligent robots have been built, including Honda's ASIMO and Sony's QRIO.

Advanced Research Topics

To build a machine perception system someday that is close to or even better than the human perception system, researchers need to look at both aspects of this challenging problem: (*1*) the fundamental aspect of how human perception system works and (*2*) the systematic aspect of how to improve system performance based on best theories and technologies available. To illustrate the fascinating characteristics of the human perception system and how it is different from currently available machine perception systems, we plot the negative and upside-down photos of a person in Fig. 9. It is well known that negative or upside-down photos make human perception of faces more difficult (59)). Also we know that no difference exists in terms of information (bits used to encode images) between a digitized normal photo and a digitized negative or upside-down photo (except the sign and orientation information).

From the system perspective, many research challenges remain. For example, recent system evaluations (60) suggested at least two major challenges: the illumination variation problem and the pose variation problem. Although many existing systems build in some sort of performance invariance by applying pre-processes, such as histogram equalization or pose learning, significant illumination or pose change can cause serious performance degradation. In addition, face images could be partially occluded, or the system needs to recognize a person from an image in the database that was acquired some time ago. In an extreme scenario, for example, the search for missing children, the time interval could be up to 10 years. Such a scenario poses a significant challenge to build a robust system that can tolerate the variations in appearances across many years.

Real problems exist when face images are acquired under uncontrolled and uncooperative environments, for example, in surveillance applications. Although illumination and pose variations are well-defined and well-researched problems, other problems can be studied systematically, for example, through mathematical modeling. Mathematical modeling allows us to describe physical entities mathematically and hence to transfer the physical

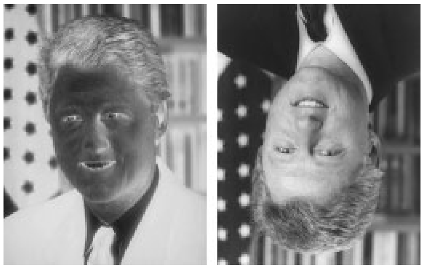


Figure 9. Typical limitation of human perception system with negative and upside-down photos: which makes it difficult or takes much longer for us to recognize people from the photos. Interestingly, we can manage eventually to overcome this limitation when recognizing famous people (President Bill Clinton in this case).

phenomena into a series of numbers (31). The decomposition of a face image into a linear combination of eigenfaces is a classic. example of mathematical modeling. In addition, mathematical modeling can be applied to handle the issues of occlusion, low resolution, and aging.

As an application of image analysis and understanding, machine recognition of faces benefits tremendously from advances in many relevant disciplines. To conclude our article, we list these disciplines for further reading and mention their direct impact on face recognition briefly

- **Pattern recognition.** The ultimate goal of face recognition is recognition of personal ID based on facial patterns, including 2-D images, 3-D structures, and any pre-processed features that are finally fed into a classifier.
- **Image processing.** Given a single or a sequence of raw face images, it is important to normalize the image size, enhance the image quality, and to localize local features before recognition.
- **Computer vision.** The first step in face recognition involves the detection of face regions based on appearance, color, and motion. Computer vision techniques also make it possible to build a 3-D face model from a sequence of images by aligning them together. Finally, 3-D face modeling holds great promises for robust face recognition.
- **Computer graphics.** Traditionally, computer graphics are used to render human faces with increasingly realistic appearances. Combined with computer vision, it has been applied to build 3-D models from images.
- **Learning.** Learning plays a significant role to building a mathematical model. For example, given a training set (or bootstrap set by many researchers) of 2-D or 3-D images, a generative model can be learned and applied to other novel objects in the same class of face images.
- **Neuroscience and Psychology.** Study of the amazing capability of human perception of faces can shed some light on how to improve existing systems for machine perception of faces.

BIBLIOGRAPHY

1. W. Zhao, Tutorial on face recognition, *European Conference on Computer Vision*, 2004.
2. H. Rowley, S. Baluja, and T. Kanade, Neural network based face detection, *IEEE Trans. Patt. Anal. Mach. Intell.*, **20**: 39–51, 1998.
3. T. Cootes, C. Taylor, D. Cooper, and J. Graham, Active shape models–their training and application, *Comp. Vis. Image Understand.*, **61**: 18–23, 1995.
4. I. Bruner and R. Tagiuri, The perception of people, In G. Lindzey (ed.), *Handbook of Social Psychology*, Reading, MA: Addision-Wesley, 1954.
5. M. Bledsoe, The model method in facial recognition, Technical Report PRI 15, Palo Alto, CA: Panoramic Research Inc., 1964.
6. C. Darwin, *The Expression of the Emotions in Man and Animals*, London: John Murray, 1872.
7. P. Ekman, (ed.), *Charles Darwin's THE EXPRESSION OF THE EMOTIONS IN MAN AND ANIMALS*, London and New York: 1998.
8. F. Galton, Personal identification and description, *Nature*, (June 21): 173–188, 1888.
9. T. Kanade, Computer Recognition of Human Faces, *Basel*, Switzerland: Birkhauser, 1973.
10. M. Kelly. Visual identification of people by computer. Technical Report AI 130, Stanford, CA, 1970.
11. I. Biederman and P. Kalocsai, Neural and psychophysical analysis of object and face recognition. In H. Wechsler, P. J. Phillips, V. Bruce, F. Soulie, and T. S. Huang (eds), *Face Recognition: From Theory to Applications, Berlin*: springer-Verlag, 1998, pp. 3–25.
12. I. Gauthier and N. Logothetis, Is face recognition so unique after all? *J. Cognit. Neuropsychol*. **17**: 125–142, 2000.
13. J. Haxby, M. I. Gobbini, M. Furey, A. Ishai, J. Schouten, and P. Pietrini, Distributed and overlapping representations of faces and objects in ventral temporal cortex, *Science*, **293**: 425–430, 2001.
14. M. Kirby and L. Sirovich, Application of the karhunen-loeve procedure for the characterization of human faces, *IEEE Trans. on Patt. Analy. Mach. Intell.*, **12**: 103–108, 1990.
15. M. Turk and A. Pentland. Eigenfaces for recognition. *J. Cognitive Neurosc.*, **3**: 72–86, 1991.
16. P.N. Belhumeur, J. Hespanha, and D. J. Kriegman, Eigenfaces vs. fisherfaces: Recognition using class specific linear projection, *IEEE: Trans. Pa. Anal. Mac. Intell.*, **19**: 711–720, 1997.
17. K. Etemad and R. Chellap, Discriminant analysis for Recognition of human face images, *J. Optical Soc. Amer.*, **14**: 1724–1733, 1997.
18. W. Zhao, R. Chellappa, and A. Krishnaswamy, Discriminant analysis of principal components for face recognition, Proc. of International Conference on Automatic Face and Gesture Recognition, 1998 pp. 336–341.
19. L. Wiskott, J.-M. Fellous, and C. v. d. Malsburg, Face recognition by elastic bunch graph matching, *IEEE Trans. Patt. Anal. Mach. Intell.*, **19**: 775–779, 1997.
20. I. Cox, J. Ghosn, and P. Yianilos, Feature-based face recognition using mixture-distance, *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 209–216, 1996.
21. International Workshop on Real Time 3D Sensor and Their Use 2004.
22. W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, Face recognition: A literature survey, *ACM Comput. Surv.***35**: 399–458, 2003.
23. K. Sung and T. Poggio, Example-based learning for view-based human face detection, *IEEE Trans. Patt. Anal. Mach.*, **20**: 39–51, 1997.
24. A. Martinez, Recognizing imprecisely localized, partially occluded and expression variant faces from a single sample per class, *Trans. on Patt. Anal. Mach. Intel.*, **24**: 748–763, 2002.
25. M. H. Yang, D. Kriegman, and N. Ahuja, Detecting faces in images: A survey, *Trans. Patt. Anal. Mach. Intell.*, **24**: 34–58, 2002.
26. W. Zhao, *Robust Image Based 3D Face Recognition*, PhD thesis, College Park, MD: University of Maryland, 1999.
27. P. Hallinan, Recognizing human eyes, *SPIE Proc. of Vol. 1570: Geometric Methods In Computer Vision*, 1991. pp. 214–226,

28. A. Lanitis, C. Taylor, and T. Cootes, Automatic face identification system using flexible appearance models, *Image Vision Comput.*, **13**: 393–401, 1995.

29. T. Cootes, G. Edwards, and C. Taylor, Active appearance models, *IEEE Trans. Patt. Anal. Mach. Intell.* **23**: 681–685, 2001.

30. T. Cootes, K. Walker, and C. Taylor, View-based active appearance models, *Proc. of International Conference on Automatic Face and Gesture Recognition*, 2000.

31. W. Zhao and R. Chellappa, eds. *Face Processing: Advanced Modeling and Methods*, Burlington, VT: Academic Press, 2006.

32. W. Zhao, R. Chellappa, and P. Phillips, Subspace linear discriminant analysis for face recognition. Technical Report CAR-TR 914, College Park, MD: University of Maryland, 1999.

33. H. Moon and P. Phillips, Computational and performance aspects of pca-based face recognition algorithrms, *Perception*, **30**: 301–321, 2001.

34. D. Swets and J. Weng, Using discriminant eigenfeatures for image retrieval, *IEEE Trans. Patt. Anal. Mach. Intell.*, **18**: 831–836, 1996.

35. K. Fukunaga, *Statistical Pattern Recognition*, Academic Press, New York: 1989.

36. P. Penev and L. Sirovich, The global dimensionality of face space, Proc. of the 4th International Conference on Automatic Face and Gesture Recognition, 2000, pp. 264.

37. Z. Hong and J. Yang, Optimal disciminant plane for a small number of samples and design method of classifier on the plane, *Patt. Recog.*, **24**: 317–324, 1991.

38. L. Chen, H. Liao, M. Ko, J. Lin, and G. Yu, A new lda-based face recognition system which can solve the small sample size problem, *Patt. Recogn.*, **33**: 1713–1726, 2000.

39. M.-H. Yang, Kernel eigenfaces vs. kernel fisherfaces: Face recognition using kernel methods, *Proc. of International Conference on Automatic Face and Gesture Recognition*, 2002 pp. 215–220.

40. B. Schlkopf, A. Smola, and K.-R. Muller, Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computat.*, **10**: 1299–1319, 1998.

41. S. Mika, G. Rätsch, J. Weston, B. Schlkopf, and K.-R. Muller, Fisher discriminant analysis with kernels, *Proc. of Neural Networks for Signal Processing IX* 1999, pp. 41–48.

42. M. A. Vasilescu and D. Terzopoulos, Multilinear analysis of image ensembles: Tensorfaces, *Proc. of European Conference on Computer Vision*, 2002 pp. 447–460

43. B. Moghaddam and A. Pentland, Probabilistic visual learning for object representation, *IEEE Trans. Patt. Anal. Mach. Intell.***19**: 696–710, 1997.

44. P. Phillips, Support vector machines applied to face recognition, Proc. of Neural Information Processing Systems, 1998, pp. 803–809.

45. C. Liu and H. Wechsler, Evolutionary pursuit and its application to face recognition, *IEEE Trans. Patt. Anal. Mach. Intell.* **22**: 570–582, 2000.

46. S. Z. Li and J. Lu, Face recognition using the nearest feature line method, *IEEE Trans. Neural Netw.***10**: 439–443, 1999.

47. M. Bartlett, H. Lades, and T. Sejnowski, Independent component representation for face recognition, *Proc. of SPIE Symposium on Electronic Imaging: Science and Technology*, pp. 528–537. 1998.,

48. S. Lin, S. Kung, and L. Lin, Face recognition/detection by probabilistic decision-based neural network, *IEEE Trans. Neural Netw.*, **8**: 114–132, 1997.

49. L. R. Tucker, Some mathetical notes on three-mode factor analysis, *Psychometrika*, **31**: 279–311, 1996.

50. B. Majunath, R. Chellappa, and C. v. d. Malsburg, A feature based approach to face recognition, Proc. of IEEE Conference on Computer Vision and Pattern Recognition, 1992, pp. 373–378.

51. F. Samaria and S. Young, HMM based architecture for face identification, *Image and Vision Computing*, **12**: 537–583, 1994.

52. A. Nefian and M. Hayes III, Hidden markov models for face recognition, Proc. of International Conference on Acoustics, Speech, and Signal Proceeding, 1998, pp. 2721–2724.

53. S. Lawrence, C. Giles, A. Tsoi, and A. Back, Face recognition: A convolutional neural-network approach, *IEEE Trans. Neural Netw.*, **8**: 98–113, 1997.

54. A. Pentland, B. Moghaddam, and T. Straner, View-based and modular eignespaces for face recognition, Proc. of IEEE Conference on Computer Vision and Pattern Recognition, 1994, pp. 84–91.

55. P. Penev and J. Atick, Local feature analysis: A general statistical theory for object representation, *Network: Computat. Neural Sys.*, **7**: 477–500, 1996.

56. J. Huang, B. Heisele, and V. Blanz, Component-based face recognition with 3d morphable models, Proc. of International Conference on Audio- and Video-Based Person Authentication, 2003.

57. R. Fisher, The statistical utilization of multiple measuremements, *Annals Eugen.*, **8**: 376–386, 1938.

58. M. Lades, J. Vorbruggen, J. Buhmann, J. Lange, C. V. Malsburg, R. Wurtz, and W. Konen, Distortion invariant object recognition in the dynamic link architecture, *IEEE Trans. Comp.*, **42**: 300–311, 1993.

59. R. Yin, Looking at upside-down faces, *J. Experim. Psychol.*, **81**: 141–151, 1969.

60. P. J. Phillips, H. Moon, S. Rizvi, and P. Rauss, The feret evaluation methodology for face-recognition algoithms, *IEEE Trans. Patt. Analy. Mach. Intell.*, **22**: 1090–1104, 2000.

WENYI ZHAO
Institutive Surgical, Inc.
Sunnyvale, California

FINGERPRINT IDENTIFICATION

HISTORY OF FINGERPRINTS

Biometrics such as fingerprint, palm, face, gait, ear, signature, and speech are used to ensure a high confidence in the recognition of an individual for high-security applications (1). Among these biometric traits, fingerprints have been used for a long period of time because of their distinctiveness and immutability. The study of fingerprint characteristics can be traced back to about 4500 years, the era of the pyramid-building in Egypt. However, the use of fingerprints for identification began in the mid-1800s.

Sir William Herschel discovered that fingerprints remain stable over time and distinct across individuals. In 1877, he commenced placing the inked palm and thumb impressions of some members of the local population on contracts. These prints were used as a form of signature on the documents. However, Herschel never claimed that he had developed a method to identify criminals.

In the late 1800s, the most advanced findings in fingerprint study was made by Dr. Henry Faulds. He found that fingerprints will not change even with superficial injury and that the latent prints left on objects can be used to identify criminals. In 1892, Sir Francis Galton published an accurate and in-depth study of fingerprint science in a book called *Finger Prints,* in which he described an attempt at a fingerprint classification system to facilitate the handling of large collections of fingerprints. Although the work of Galton proved to be sound and became the foundation of modern fingerprint science, his approach to classification was inadequate. Juan Vucetich, an Argentinian police officer who corresponded with Galton, devised his own fingerprint classification system, which was put into practice in September 1891. In 1897, Sir Edward Henry established the famous *Henry System*(2), which is a systematic and effective method of classifying fingerprints. He published the book *Classification and Uses of Fingerprints* in 1900. About 10 years later, his classification system was being used widely by police forces and prison authorities in the English-speaking world.

Since the early 1960s, researchers have begun to develop an *automatic* fingerprint identification system (AFIS) to improve the efficiency of fingerprint recognition. Today, almost all law enforcement agencies around the world use an AFIS. And fingerprint science is a well-researched field with research and development activities worldwide. Several publicly available databases (3,4) exist for evaluating the performance of various fingerprint recognition algorithms. New high-resolution electronic sensors, which are quite affordable (5,6), are available for use in portable laptop computers, mobile phones, and personal digital assistants (PDAs).

FINGERPRINT FEATURES

Fingerprints have two fundamentally important characteristics: permanence and distinctiveness. It has been found that fingerprints are formed in the fetal stage and do not change naturally. Also, no two fingerprints ever are found to be exactly the same. Because of these interesting characteristics, fingerprints are used widely for human recognition and identification.

Fingerprints are represented by features that are classified at three levels.

- Level 1 features describe the patterns of the fingerprints, which include ridge flow, core and delta, and pattern type. Ridge flow is the orientation image of the fingerprint. This feature is commonly used for classification. Figure 1 shows an example of the original image and the orientation image. Core and delta are singular points that are defined as the points where the orientation field of a fingerprint is discontinuous. Core is the topmost point on the innermost recurving ridge, and delta is the center of a triangular region where flows from three different directions meet. Figure 2 is an example of the core and delta in a fingerprint. Fingerprints are generally classified into five classes: right loop (R), left loop (L), whorl (W), arch (A), and tented arch (T). Figure 3 shows examples of the fingerprints from these five classes.
- Level 2 features are the points of the fingerprints. They include minutiae, scar, crease, and dot (7). A fingerprint consists of white and dark curves. The white curves are called the *valley* and the dark curves are called the *ridge*. Minutiae features are the ridge characteristics that correspond to the crossings and endings of ridges. They include endpoint, bifurcation, forks, island, and enclosures. The endpoint and bifurcation are used commonly in fingerprint recognition. Figure 4 is an example of the endpoint and bifurcation. Scar is the crossing of two or more adjacent ridges. Figure 5 (a) shows an example of a scar. A crease appears as a white line in a fingerprint. It is a linear depression (or grooves) in the skin. Figure 5 (b) shows an example of a crease. A dot is an isolated ridge unit with a pore on it. Figure 5 (c) shows an example of a dot.
- Level 3 features (5) describe the fingerprint shape that refers to pores and ridge contours. Pores are small openings on ridges. We need a high resolution sensor ($\geq$1000 pixels per inch (ppi)) to get this feature. Figure 6 is an example of sweat pores. Ridge contours are morphological features that include ridge width, shape, path deviation, and so forth.

Fingerprint sensors, the very front end of the fingerprint recognition systems, are used to capture the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

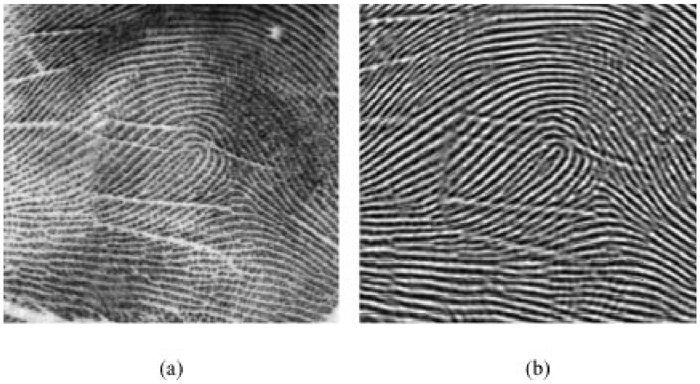


Figure 1. An example of the original image and the orientation image: (a) original image, (b) orientation image. The orientation is shown by the arrows above the ridges.

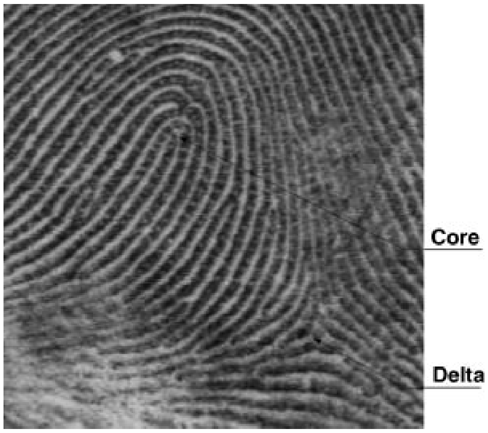


Figure 2. Level 1 features: core and delta.

fingerprint images. The kinds of fingerprint sensors are: optical sensors, semiconductor sensors, and ultrasound sensors. Among these sensors, optical sensors are considered to be stable and reliable, semiconductor sensors are considered to be low cost and portable, and ultrasound sensors are considered to be accurate but more expensive.

FINGERPRINT RECOGNITION

Depending on an application two kinds of fingerprint recognition systems exist: verification systems and identification systems (8). A verification system generally stores the fingerprint images or feature sets of users in a database. At a future time, it compares the fingerprint of a person with her/his own fingerprint image or feature set to verify that this person is, indeed, who she/he claims to be. This problem is a one-to-one matching problem. The system can accept or reject this person, according to the verification result. An identification system is more complex where, for a query fingerprint, the system searches the entire database to find out if any fingerprint images or feature sets saved in the database can match it. It conducts one-to-many matching (8). Two kinds of identification systems exists: the closed-set identification system and the open-set identification system (9). The closed-set identification system is the identification system for which all potential users are enrolled in the system. Usually, the closed-set identification is used for research purposes. The open-set identification system is the identification system for which some potential users are not enrolled in the system. The open-set identification is performed in real operational systems. The verification and the closed-set identification are special cases of the open-set identification.

Three kinds of approaches (see Fig. 7) exist to solve the fingerprint identification problem (10): (1) Repeat the verification procedure for each fingerprint in the database and select the best match; (2) use fingerprint classification followed by verification; and (3) use fingerprint indexing followed by verification (10,11). Fingerprint matching, classification, and indexing are three basic problems in fingerprint identification.

Fingerprint Matching

A fingerprint matching algorithm aligns the two given fingerprints, finds the correspondences between them,

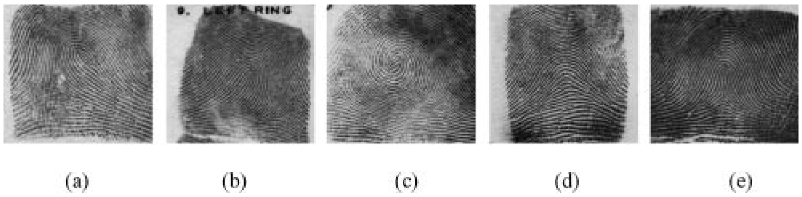


Figure 3. Examples of fingerprints for each class based on the *Henry System:* (a) right loop, (b) left loop, (c) whorl, (d) arch, and (e) tented.

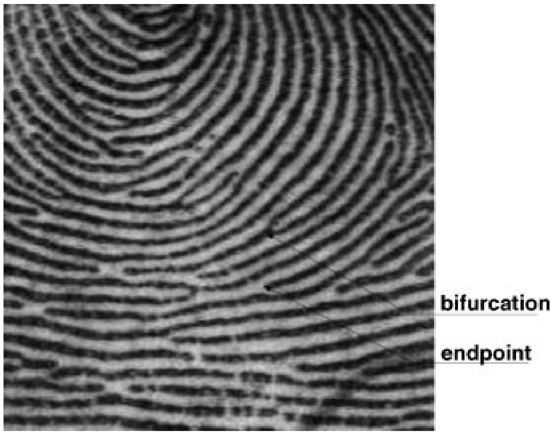


Figure 4. Minutiae: endpoint and bifurcation.

and returns a measure of the degree of similarity. Usually, the similarity score is used to represent the degree of similarity between the two given fingerprints. Fingerprint matching is a challenging problem because different impressions of the same finger could be very different because of distortion, displacement, rotation, noise, skin condition, pressure, noise, and so forth (8). Furthermore the impressions from different fingers could be quite similar. Figure 8 shows two impressions of one fingerprint from the NIST-4 database (10). The fingerprint matching algorithms can be classified into three different types: (1) the correlation-based approach, (2) the minutiae-based approach, and (3) the ridge feature-based approach.

1. *Correlation-based matching:* The correlation-based approach uses the gray-level information of fingerprints. For a template fingerprint (in the database) and a query fingerprint, it computes the sum of the squared differences in gray values of all the pixels to evaluate the diversity of the template fingerprint and the query fingerprint. To deal with the distortion problem, it computes the correlation in local regions instead of the global correlation on the entire image. In Bazen et al. (12), the correlation-based evaluation is used to find the distinctive regions of the template fingerprint. These local regions fit very well at the original locations and much worse at other locations. During the matching, they compute the gray-level distance between the distinctive regions of a template and the corresponding areas in the query fingerprints. Then, they sum up the squared gray-level difference for each local region. The position of a region of the template with the minimal distance is considered as the corresponding region in the query fingerprint.

   Compared with the other matching algorithms described below, correlation-based matching approaches use gray-level information of the fingerprint. When the quality of the fingerprint image is not good, especially when a large number of minutiae are missing, the correlation-based matching algorithm may be considered. However, it is expensive computationally.

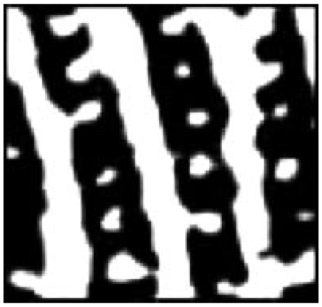


Figure 6. Example of sweat pores.

2. *Minutiae-based matching:* Minutiae-based matching is the most commonly used method for fingerprint recognition systems. In this approach, a fingerprint is represented by a set of minutiae features. Thus, the fingerprint recognition problem is reduced to a point-matching problem. Therefore, any point matching approach, such as the relaxation algorithms, can be used to recognize the fingerprints (13,14).

- Feature extraction: The first step of the minutiae-matching algorithm is the minutiae extraction. Figure 9 is the block diagram of the minutiae-based feature extraction procedure, which is used widely in most fingerprint recognition systems. As an example, Bhanu and Tan present a learned template-based algorithm for feature extraction (15). Templates are learned from examples by optimizing a criterion function using the Lagrange method. To detect the presence of minutiae in fingerprints, templates for endpoints and bifurcations are applied with appropriate orientation to the binary fingerprints at selected potential minutiae locations.

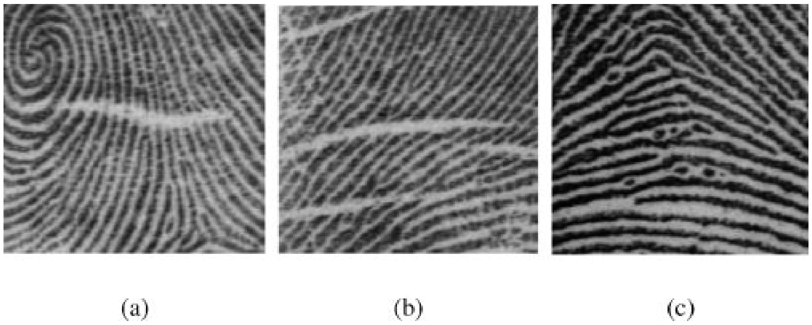
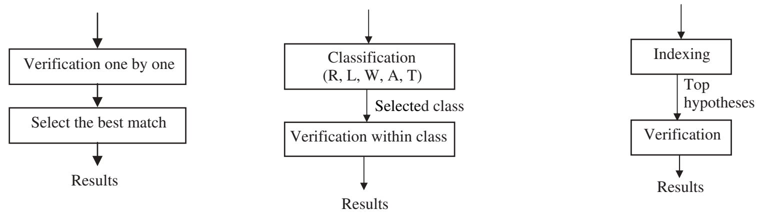


Figure 5. Level 2 features: (a) scar, (b) crease, and (c) dot.



(1) Repeat verification (2) Classification followed by verification (3) Indexing followed by verification

Figure 7. Block diagram of three kinds of approaches to solve the identification problem.

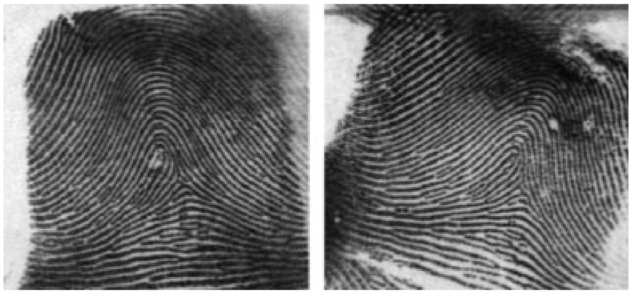


Figure 8. Two impressions of one fingerprint.

- Matching: Tan and Bhanu (13,14) present a fingerprint-matching approach, based on genetic algorithms (GA). This method can achieve a globally optimized solution for the transformation between two sets of minutiae extracted from two different fingerprints. In their approach, the fitness function is based on the local properties of triplets of minutiae, such as minimum angle, maximum angle, triangle handedness, triangle direction, maximum side, minutiae density, and ridge counts. These features are described in the "Fingerprint Indexing" section below.

  Jiang and Yau (16) use both the local and global structures of minutiae in their minutiae-matching approach. The local structure of a minutia describes the features independent of the rotation and translation in its l-nearest neighborhood. The global structure is variant with the rotation and translation. Using the local structure, the best matched minutiae pair is found and used to align the template and query fingerprint. Then, the elastic bounding box of the global features is used for the fingerprint matching.

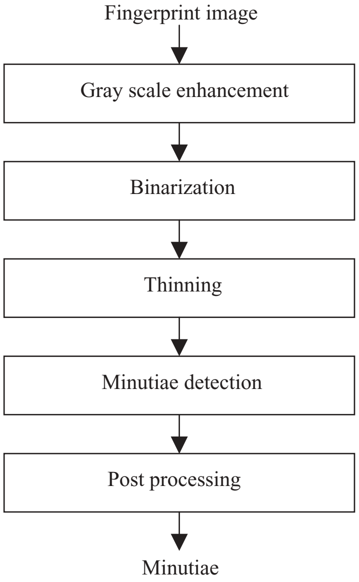


Figure 9. Block diagram for minutiae-based feature extraction.

  Kovacs-Vajna (17) used triangular matching to deal with deformations of fingerprints. In this approach, the minutiae regions of the template fingerprint are moved around the query fingerprint to find the possible correspondence. The triangular matching algorithm is used to obtain the matching minutiae set. Then, the dynamic time-warping algorithm is applied to validate the final matching results.

3. *Ridge feature-based matching:* For a good quality fingerprint with size 480 × 512 [500 pixels per inch (ppi)], about 80 minutiae features could exist. Thus, for the triangular minutiae matching, hundreds of thousands of triangles could exist. So, the minutiae matching approach needs high computational power. However, when the image quality is not good, the minutiae extraction would be difficult. Because fingerprints consist of natural valley and ridges, researchers have used them for fingerprint matching. Maltoni et al. (8) present a filter-based algorithm for fingerprint recognition. It is based on the grayscale image. First, they determine a reference point with the maximum curvature of the concave ridges and an interest region in the fingerprint. After tessellating the interest region around the reference point, they use a bank of Gabor filters to capture both local and global details in a fingerprint. Then, they compute the average absolute deviation from the mean to define the compact fixed-length FingerCode as the feature vector. Finally, they match the fingerprint by computing the Euclidean distance between the corresponding FingerCode between the template and query fingerprints.

With the improvement of the fingerprint sensor technology, now it is possible to extract features at a high resolution. In Jain et al. (5), authors use pores and ridge contours combined with minutiae features to improve fingerprint recognition performance. In their approach, they use Gabor filter and wavelet transform to extract pores and ridge contours. During the matching process, they extract orientation field and minutiae features and establish alignment between the template and query fingerprint. If the orientation fields match, then the system uses a minutiae-based matching algorithm to verify the query fingerprint or to reject the query fingerprint. If the number of the corresponding minutiae between the template fingerprint and query fingerprint is greater than a threshold, then these two fingerprints match; if not, then the system extracts pores and ridge contours and they use the Iterative Closest Point (ICP) algorithm to match these features. This hierarchical matching system requires 1000 ppi resolution for the sensor.

FINGERPRINT CLASSIFICATION

Most fingerprint classification systems use the Henry system for fingerprint classification, which has five classes as shown in Fig. 3. The most widely-used approaches for fingerprint classification are based on the number and relations of the singular points, including the core and the delta. Karu and Jain (24) present a classification approach based on the structural information around the singular points. Three steps are in this algorithm: (1) Compute the ridge direction in a fingerprint image; (2) find the singular points based on the changes in the directional angle around the curve; and (3) classify the fingerprints according to the number and locations of the core and delta. Other researchers use a similar method: first, find the singular point; then use a classification algorithm to find the difference in areas, which are around the singular points for different classes. Several representations based on principal component analysis (PCA) (3), a self-organizing map (18), and Gabor filters (8) are used. The problems with these approaches are:

- It is not easy to detect singular points, and some fingerprints do not have singular points.
- Uncertainty in the location of the singular points is large, which has a great effect on the classification performance because the features around the singular points are used.

Cappelli et al. (19) present a structural analysis of the orientation field of a fingerprint. In their approach, the directional image is calculated and enhanced. A set of dynamic masks is used in the segmentation step, and each dynamic mask is adapted independently to best fit the directional image according to a cost function. The resulting cost constitutes a basis for the final classification (3). Based on the orientation field, Cappelli et al. also present a fingerprint classification system based on the multispace KL transform (20). It uses a different number of principal components for different classes. Jain and Minut (31) propose a classification algorithm based on finding the kernel that best fits the flow field of a given fingerprint. For each class, a kernel is used to define the shape of the fingerprint in that class. In these approaches, it is not necessary to find the singular points.

Researchers also have tried different methods to combine different classifiers to improve the classification performance. Senior (21) combines the hidden Markov model (HMM), decision trees, and PCASYS (3). Yao et al. (22) present new fingerprint classification algorithms based on two machine learning approaches: support vector machines (SVMs) and recursive neural networks (RNNs). In their approach, the fingerprints are represented by the relational graphs. Then, RNNs are used to train these graphs and extract distributed features for the fingerprints. SVMs integrated with distributed features are used for classification. To solve the ambiguity problem in fingerprint classification, an error-correcting code scheme is combined with SVMs.

Tan et al. (23) present a fingerprint classification approach based on genetic programming (GP) to learn composite operators that help to find useful features. During the training, they use GP to generate composite operators. Then, the composite operators are used to generate the feature vectors for fingerprint classification. A Bayesian classifier is used for classification. Fitness values are computed for the composite operators based on the classification results. During the testing, the learned composite operator is applied directly to generate feature vectors. In their approach, they do not need to find the reference points. Table 1 summarizes representative fingerprint classification approaches.

Table 1. Representative fingerprint classification approaches

Authors	Approach
Candela et al. (3), 1995	Probabilistic neural network (PNN)
Karu and Jain (24), 1996	Rule-based classification
Halici and Ongun (18), 1996	Neural network based on self-organizing feature maps (SOM)
Cappelli et al. (19), 1997	Multispace principal component analysis
Qi et al. (25), 1998	Probabilistic neural network based on genetic algorithm (GA) and feedback mechanism
Jain et al. (8), 1999	K-nearest neighbor and neural network based on Gabor features (FingerCode)
Cappelli et al. (26), 1999	Classification based on partitioning of orientation image
Kamijo (27), 1999	A four-layered neural network integrated in a two-step learning method
Su et al. (28), 2000	Fractal analysis
Pattichis et al. (29), 2001	Probabilistic neural network and AM–FM representation for fingerprints
Bernard et al. (30), 2001	Kohonen topological map
Senior (21), 2001	Hidden Markov model and decision tree and PCASYS
Jain and Minut (31), 2002	Model-based method based on hierarchical kernel fitting
Mohamed and Nyongesa (32), 2002	Fuzzy neural network
Yao et al. (22), 2003	Support vector machine and recursive neural network based on FingerCode
Tan et al. (23), 2005	Genetic programming

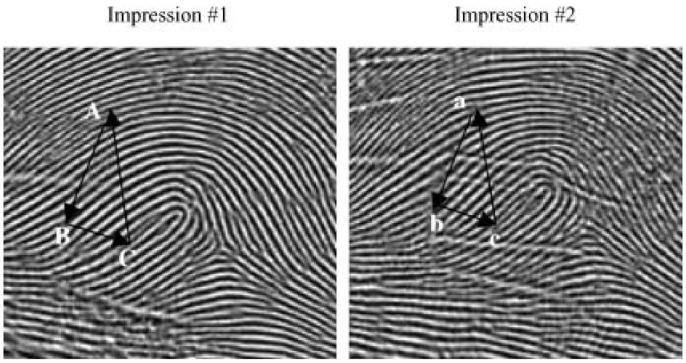


Figure 10. An example of two corresponding triangles in a pair of fingerprints.

FINGERPRINT INDEXING

The purpose of indexing algorithms is to generate, in an efficient manner, a set of hypotheses that is a potential match to a query fingerprint. Indexing techniques can be considered as front-end processing, which then would be followed by back-end verification processing in a complete fingerprint recognition system.

A prominent approach for fingerprint indexing is by Germain et al. (11). They use the triplets of minutiae in their indexing procedure. The features they use are: the length of each side, the ridge count between each pair of vertices, and the angles that the ridges make with respect to the x-axis of the reference frame. The number of corresponding triangles is defined as the similarity score between the query and the template fingerprints. In their approach, a hash table is built where all possible triplets are saved. For each triplet, a list of IDs, including the fingerprints that have this triplet, is saved. During the identification process, the triplets of the query fingerprint are extracted and—by a hashing process described below—the potential IDs of the query fingerprint are determined.

Because some features in Ref. (11) may not be reliable, Bhanu and Tan (33) use a novel set of features of a triplet for fingerprint indexing. These features are:

- Minimum angle α_{min} and median angle α_{med}. Assume α_i are three angles in a triplet, where i = 1, 2, 3. $\alpha_{min} = min\{\alpha_i\}$, $\alpha_{max} = max\{\alpha_i\}$, $\alpha_{med} = 180° - \alpha_{min} - \alpha_{max}$.
- Triangle handedness ϕ. Let $Z_i = x_i + jy_i$ be the complex number corresponding to the location (x_i, y_i) of point P_i, i = 1,2, 3. Define $Z_{21} = Z_2 - Z_1$, $Z_{32} = Z_3 - Z_2$, and $Z_{13} = Z_1 - Z_3$. Let triangle handedness $\phi = sign\ (Z_{21} \times Z_{32})$, where *sign* is the signum function and × is the cross product of two complex numbers. Points P_1, P_2, and P_3 are noncolinear points, so ϕ = 1 or -1.
- Triangle direction η. Search the minutiae in the image from top to bottom and left to right. If the minutiae is the start point, then $\upsilon = 1$; otherwise $\upsilon = 0$. Let $\eta = 4\upsilon_1 + 2\upsilon_2 + \upsilon_3$, where υ_i is the υ value of point P_i, i = 1,2,3, and $0 \leq \eta \leq 7$.
- Maximum side λ. Let $\lambda = max\{L_i\}$, where $L_1 = |Z_{21}|, L_2 = |Z_{32}|$, and $L_3 = |Z_{13}|$.
- Minutiae density χ. In a local area (32 × 32 pixels) centered at the minutiae P_i, if χ_i minutiae exists then the minutiae density for P_i is χ_i. Minutiae density χ is a vector consisting of all χ_i.
- Ridge counts ξ. Let ξ_1, ξ_2, and ξ_3 be the ridge counts of sides P_1P_2, P_2P_3, and P_3P_1, respectively. Then, ξ is a vector consisting of all ξ_i.

During the offline hashing process, the above features for each template fingerprint (33) are computed and a hash table $H(\alpha_{min}, \alpha_{med}, \phi, \eta, \lambda, \chi, \xi)$ is generated. During the online hashing process, the same features are computed for each query fingerprint and compared with the features represented by H. If the difference in features is small enough, then the query fingerprint is probably the same as the "stored" fingerprints that have similar features. Figure 10 is an example of two corresponding triangles in a pair of fingerprints that are two impressions of one fingerprint. In the first impression, three noncolinear minutiae A, B, and C are picked randomly to form a triangle ΔABC. The features in this triangle are $\{\alpha_{min} = 30°, \alpha_{med} = 65°, \phi = 1, \eta = 6, \lambda = |AC|, \chi = \{0,0,0\}, \Delta\xi = \{6,5,12\}\}$. Similarly, three noncolinear minutiae a, b, and c in the second impression form Δabc. Its features are $\{\alpha_{min} = 31°, \alpha_{med} = 63°, \phi = 1, \eta = 6, \lambda = |ac|, \chi = \{0,2,0\}, \xi = \{6,5,12\}\}$. If the error between these two triangles are within the error tolerance (34), then these two triangles are considered the corresponding triangles. The output of this process, carried out for all the triplets, is a list of hypotheses, which is sorted in the descending order of the number of potential corresponding triangles. Top T hypotheses are the input to the verification process.

PERFORMANCE EVALUATION

Two classes in the fingerprint recognition systems exist: match and nonmatch. Let s and n denote match and non-match. Assume that x is the similarity score. Then, $f(x\,|\,s)$ is the probability density function given s is true, and $f(x\,|\,n)$ is the probability density function given n is true. Figure 11 is an example of these two distributions. For a criterion k, one can define

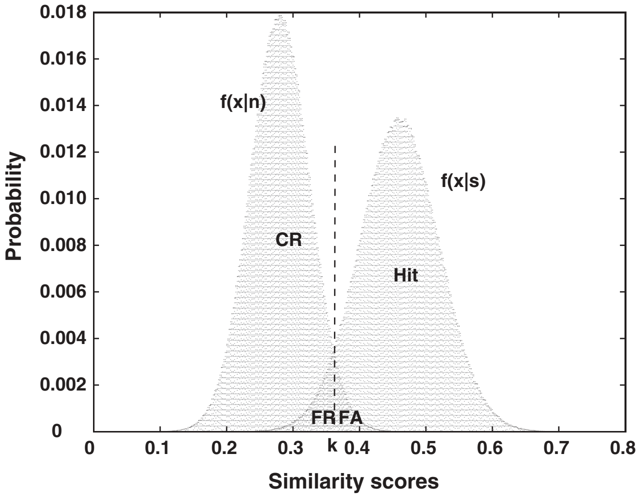


Figure 11. Densities of the match and nonmatch scores.

- Hit: the probability that x is above k given s, where $Hit = \int_k^\infty f(x|s)dx$
- False alarm: the probability that x is above k given n, where $FA = \int_k^\infty f(x|n)dx$
- False rejection: the probability that x is below k given s, where $FR = \int_{-\infty}^k f(x|s)dx$
- Correct rejection: the probability that x is below k given n, where $CR = \int_{-\infty}^k f(x|n)dx$

A *receiver operating characteristic (ROC)* curve is a graphical plot whose x-axis is the *false alarm (FA)* rate and y-axis is the *hit (Hit)* rate. A *ROC* curve is used to evaluate the performance of a recognition system because it represents the changes in *FA* rate and *Hit* rate with different discrimination criteria (thresholds). A *detection error tradoff (DET)* curve plots the *FA* rate and the *false rejection (FR)* rate with the change of discrimination criteria. A confidence interval is an interval within which the estimation is likely to be determined. The ISO standard performance testing report gives the detail of recommendations and requirements for the performance evaluations (9).

Public fingerprint databases are used to evaluate the performance of different fingerprint recognition algorithms and fingerprint acquisition sensors. The National Institute of Standards and Technology (NIST) provides several special fingerprint databases with different acquisition methods and scenarios. Most images in these databases are rolling fingerprints that are scanned from paper cards. The *NIST* special database 24 is a digital video of live-scan fingerprint data. The detail of the *NIST* special databases can be found in Ref. (35). Since 2000, fingerprint verification competition (FVC) has provided public databases for a competition that is held every 2 years. For each competition, four disjoint databases are created that are collected with different sensors and technologies. The performance of different recognition algorithms is addressed in the reports of each competition (36–39). The fingerprint vendor technology evaluation (F$_p$VTE) 2003 is conducted by *NIST* to evaluate the performance of the fingerprint recognition systems. A total of 18 companies competed in the *FpVTE*, and 34 systems from U.S government were examined. The performance of different recognition algorithms is discussed in Ref. (40).

PERFORMANCE PREDICTION

Several research efforts exist for analyzing the performance of fingerprint recognition. Galton (41) assumes that 24 independent square regions could cover a fingerprint and that he could reconstruct correctly any of the regions with a probability of 1/2 by looking at the surrounding ridges. Accordingly, the Galton formulation of the distinctiveness of a fingerprint is given by $(1/16) \times (1/256) \times (1/2)^{24}$, where 1/16 is the probability of the occurrence of a fingerprint type and 1/256 is the probability of the occurrence of the correct number of ridges entering and exiting each of the 24 regions. Pankanti et al. (8) present a fingerprint individuality model that is based on the analysis of feature space and derive an expression to estimate the probability of false match based on the minutiae between two fingerprints. It measures the amount of information needed to establish correspondence between two fingerprints. Tan and Bhanu (42) present a two-point model and a three-point model to estimate the error rate for the minutiae-based fingerprint recognition. Their approach not only measures the position and orientation of the minutiae but also the relations between different minutiae to find the probability of correspondence between fingerprints. They allow the overlap of uncertainty area of any two minutiae. Tabassi et al. (43) and Wein and Baveja (44) use the fingerprint image quality to predict the performance. They define the quality as an indication of the degree of separation between the match score and nonmatch score distributions. The farther these two distributions are from each other, the better the system performs.

Predicting large population recognition performance based on a small template database is another important topic for the fingerprint performance characterization. Wang and Bhanu (45) present an integrated model that considers data distortion to predict the fingerprint identification performance on large populations. Learning is incorporated in the prediction process to find the optimal small gallery size. The Chernoff and Chebychev inequalities are used as a guide to obtain the small gallery size given the margin of error and confidence interval. The confidence interval can describe the uncertainty associated with the estimation. This confidence interval gives an interval within which the true algorithm performance for a large population is expected to fall, along with the probability that it is expected to fall there.

FINGERPRINT SECURITY

Traditionally, cryptosystems use secret keys to protect information. Assume that we have two agents, called Alice and Bob. Alice wants to send a message to Bob over the public channel. Eve, the third party, eavesdrops over the public channel and tries to figure out what Alice and Bob are saying to each other. When Alice sends a message to Bob, she uses a secret encryption algorithm to encrypt the message. After Bob gets the encrypted message, he will use a secret decryption algorithm to decrypt the message. The secret keys are used in the encryption and decryption processes. Because the secret keys can be forgotten, lost, and broken, biometric cryptosystems are possible for security.

Uludag et al. (46) propose a cryptographic construct that combines the fuzzy vault with fingerprint minutiae data to protect information. The procedure for constructing the fuzzy vault is like what follows in the example of Alice and Bob. Alice places a secret value k in a vault and locks it using an unordered set A of the polynomial coefficients. She selects a polynomial p of variable x to encode k. Then, she computes the polynomial projections for the elements of A and adds some randomly generated chaff points that do not lie on p to arrive at the final point set R. Bob uses an unordered set B of the polynomial coefficients to unlock the vault only if B overlaps with A to a great extent. He tries to learn k, that is to find p. By using error-correction coding, he can reconstruct p. Uludag et al. (46) present a curve-based transformed minutia representation in securing the fuzzy fingerprint vault.

We know that the biometrics is permanently related with a user. So if the biometrics is lost, then the biometrics recognition system will be compromised forever. Also, a user can be tracked by cross-matching with all the potential uses of the fingerprint biometrics, such as access to the house, bank account, vehicle, and laptop computer. Ratha et al. (47) presents a solution to overcome these problems in the fingerprint recognition systems. Instead of storing the original biometrics image, authors apply a one-way transformation function to the biometrics. Then, they store the transformed biometrics and the transformation to preserve the privacy. If a biometrics is lost, then it can be restored by applyings a different transformation function. For different applications of the same biometrics, they use different transformation functions to avoid a user being tracked by performing a cross match.

Like other security systems, fingerprint sensors are prone to spoofing by fake fingerprints molded with artificial materials. Parthasaradhi et al. (48) developed an anti spoofing method that is based on the distinctive moisture pattern of live fingers contacting fingerprint sensors. This method uses the physiological process of perspiration to determine the liveness of a fingerprint. First, they extract the gray values along the ridges to form a signal. This process maps a 2 fingerprint image into a signal. Then, they calculate a set of features that are represented by a set of dynamic measurements. Finally, they use a neural network to perform classification (live vs. not live).

CONCLUSIONS

Because of their characteristics, such as distinctiveness, permanence, and collectability, fingerprints have been used widely for recognition for more than 100 years. Fingerprints have three levels of features: pattern, point, and shape. With the improvement of the sensor resolution, more and better fingerprint features can be extracted to improve the fingerprint recognition performance. Three kinds of approaches exist to solve the fingerprint identification problem: (1) Repeat the verification procedure for each fingerprint in the database and select the best match; (2) perform fingerprint classification followed by verification; and (3) create fingerprint indexing, followed by verification. Fingerprint verification is done by matching a query fingerprint with a template. The feature extraction, matching, classification, indexing and performance prediction are the basic problems for fingerprint recognition.

Fingerprint prediction, security, liveness detection, and cancelable biometrics are the important current research problems. The area of biometric cryptosystems, especially the fingerprint cryptosystem, is an upcoming area of interest because the traditional secret key can be forgotten, lost, and broken.

BIBLIOGRAPHY

1. R. Wang and B. Bhanu, Predicting fingerprint biometric performance from a small gallery, *Pattern Recognition Letters*, **28** (1): 40–48, 2007.
2. E. R. Henry, *Classification and Uses of Fingerprints*. George Routledge and Sons, 1900.
3. G. T. Candela, P. J. Grother, C. I. Watson, R. A. Wilkinson, and C. L. Wilson, PCASYS-A pattern-level classification automation system for fingerprints, *NIST Technical Report. NISTIR 5467*, 1995.
4. D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain, FVC2000: fingerprint verification competition, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **24** (3): 402–412, 2002.
5. A. K. Jain, Y. Chen, and M. Demirkus, Pores and ridges: High resolution fingerprint matching using level 3 features, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **29** (1): 15–27, 2007.
6. http://authentec.com.
7. Scientific Working Group on Friction Ridge Analysis, Study and Technology (SWGFAST), *Available*: http://fingerprint.nist.gov/standard/cdeffs/Docs/SWGFAST_Memo.pdf.
8. D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar, *Handbook of Fingerprint Recognition*. New York: Springer, 2003.
9. ISO/IEC19795-1, Information Technology-Biometric Performance Testing and Reporting-Part 1: Principles and Framework. ISO/IEC JTC1/SC37 N908, 2006.
10. X. Tan and B. Bhanu, A robust two step approach for fingerprint identification, *Pattern Recognition Letters*, **24** (13): 2127–2134, 2003.
11. R. S. Germain, A. Califano, and S. Colville, Fingerprint matching using transformation parameter clustering, *IEEE Computational Science and Engineering*, **4** (4): 42–49, 1997.
12. A. M. Bazen, G. T. B. Verwaaijen, S. H. Gerez, L. P. J. Veelenturf, and B. J. vander Zwaag, A correlation-based fingerprint verification system, *Proc. IEEE Workshop on Circuits Systems and Signal Processing*, Utrecht, Holland, 2000, pp. 205–213.
13. X. Tan and B. Bhanu, Fingerprint matching by genetic algorithm, *Pattern Recognition*, **39** (3): 465–477, 2006.

14. B. Bhanu and X. Tan, *Computational Algorithms for Fingerprint Recognition*. Kluwer Academic Publishers, 2003.
15. B. Bhanu and X. Tan, Learned templates for feature extraction in fingerprint images, *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, Hawaii, 2001, Vol 2, pp. 591–596.
16. X. Jiang and W. Y. Yau, Fingerprint minutiae matching based on the local and global structures, *Proc. IEEE Int. Conf. Pattern Recognition*, Barcelona, Spain, 2000, pp. 1038–1041.
17. Z. M. Kovacs-Vajna, A fingerprint verification system based on triangular matching and dynamic time warping, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **22** (11): 1266–1276, 2000.
18. U. Halici and G. Ongun, Fingerprint classification through self-organizing feature maps modified to treat uncertainties, *Proc. IEEE*, **84** (10): 1497–1512, 1996.
19. R. Cappelli, D. Maio, and D. Maltoni, Fingerprint classification based on multi-space KL, *Proc. Workshop Autom. Identific. Adv. Tech.*, 1999, pp. 117–120.
20. N. K. Ratha and R. Bolle, *Automatic Fingerprint Recognition Systems*. Springer, 2003.
21. A. Senior, A combination fingerprint classifier, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **23** (10): 1165–1174, 2001.
22. Y. Yao, G. L. Marcialis, M. Pontil, P. Frasconi, and F. Roli, Combining flat and structured representations for fingerprint classification with recursive neural networks and support vector machines, *Pattern Recognition*, **36**, (2): 397–406, 2003.
23. X. Tan, B. Bhanu, and Y. Lin, Fingerprint classification based on learned features, *IEEE Trans. on Systems, Man and Cybernetics, Part C, Special issue on Biometrics*, **35**, (3): 287–300, 2005.
24. K. Karu and A. K. Jain, Fingerprint classification, *Pattern Recognition*, **29**, (3): pp. 389–404, 1996.
25. Y. Qi, J. Tian and R. W. Dai, Fingerprint classification system with feedback mechanism based on genetic algorithm, *Proc. Int. Conf. Pattern Recog.*, **1**: 163–165, 1998.
26. R. Cappelli, A. Lumini, D. Maio, and D. Maltoni, Fingerprint classification by directional image partitioning, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **21** (5): 402–421, 1999.
27. M. Kamijo, Classifying fingerprint images using neural network: Deriving the classification state, *Proc. Int. Conf. Neural Network*, **3**: 1932–1937, 1993.
28. F. Su, J. A. Sun, and A. Cai, Fingerprint classification based on fractal analysis, *Proc. Int. Conf. Signal Process.*, **3**: 1471–1474, 2000.
29. M. S. Pattichis, G. Panayi, A. C. Bovik, and S. P. Hsu, Fingerprint classification using an AM-FM model, *IEEE Trans. Image Process.*, **10** (6): 951–954, 2001.
30. S. Bernard, N. Boujemaa, D. Vitale, and C. Bricot, Fingerprint classification using Kohonen topologic map, *Proc. Int. Conf. Image Process.,***3**: 230–233, 2001.
31. A. K. Jain and S. Minut, Hierarchical kernel fitting for fingerprint classification and alignment, *Proc. IEEE Int. Conf. Pattern Recognition*, **2**: 469–473, 2002.
32. S. M. Mohamed and H. O. Nyongesa, Automatic fingerprint classification system using fuzzy neural techniques, *Proc. Int. Conf. Fuzzy Systems*, **1**: 358–362, 2002.
33. B. Bhanu and X. Tan, Fingerprint indexing based on novel features of minutiae triplets, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **25**, (5): 616–622, 2003.
34. X. Tan and B. Bhanu, Robust fingerprint identification, *Proc. IEEE Int. Conf. on Image Processing*, New York, 2002, pp. 277–280.
35. http://www.itl.nist.gov/iad/894.03/databases/defs/dbases.html.
36. http://bias. csr. unibo. it/fvc2000/.
37. http://bias.csr.unibo.it/fvc2002/.
38. http://bias. csr. unibo. it/fvc2004/.
39. http://bias.csr.unibo.it/fvc2006/.
40. C. Wilson, R. A. Hicklin, M. Bone, H. Korves, P. Grother, B. Ulery, R. Micheals, M. Zoepfl, S. Otto, and C. Watson, *Fingerprint Vendor Technology Evaluation 2003: Summary of Results and Analysis Report*, 2004.
41. F. Galton, *Finger Prints*. McMillan, 1892.
42. X. Tan and B. Bhanu, On the fundamental performance for fingerprint matching, *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, Madison, Wisconsin, 2003, pp. 18–20.
43. E. Tabassi, C. L. Wilson, and C. I. Watson, Fingerprint image quality, *National Institute of Standards and Technology International Report 7151*, 2004.
44. L. M. Wein and M. Baveja, Using fingerprint image quality to improve the identification performance of the U.S. visitor and immigrant status indicator technology program, *The National Academy of Sciences*, **102** (21): 7772–7775, 2005.
45. R. Wang and B. Bhanu, Learning models for predicting recognition performance, *Proc. IEEE Int. Conf. on Computer Vision*, Beijing, China, 2005, pp. 1613–1618.
46. U. Uludag, S. Pankanti, and A. K. Jain, Fuzzy vault for fingerprints, *Proc. Audio- and Video-based Biometric Person Authentication*, Rye Brook, New York, 2005, pp. 310–319.
47. N. K. Ratha, S. Chikkerur, J. H. Connell, and R. M. Bolle, Generating cancelable fingerprint templates, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **29** (4): 561–572, 2007.
48. S. T. V. Parthasaradhi, R. Derakhshani, L. A. Hornak, and S. A. C. Schuckers, Time-series detection of perspiration as a liveness test in fingerprint devices, *IEEE Trans. on System, Man, and Cybernetics-Part C: Applications and Reviews*, **35** (3): 335–343, 2005.

Rong Wang
Bir Bhanu
University of California
Riverside, California

GEOMETRIC CAMERA CALIBRATION

INTRODUCTION

Geometric camera calibration is the process of determining geometric properties of a camera. Here, the camera is considered as a ray-based sensing device, and the camera geometry defines how the observed rays of light are mapped onto the image. The purpose of calibration is to discover the mapping between the rays and image points. Hence, a calibrated camera can be used as a direction sensor for which both the forward-projection and back-projection are known, that is, one may compute the image point corresponding to a given projection ray and vice versa.

The geometric calibration of a camera is usually performed by imaging a calibration object whose geometric properties are known. The calibration object often consists of one to three planes that contain visible control points in known positions. The calibration is achieved by fitting a camera model to the observations, which are the measured positions of the control points in the calibration images. The camera model contains two kinds of parameters: The *external* parameters relate the camera orientation and position to the object coordinate frame, and the *internal* parameters determine the projection from the camera coordinate frame onto image coordinates. Typically, both the external and the internal camera parameters are estimated in the calibration process, which usually involves nonlinear optimization and minimizes a suitable cost function over the camera parameters. The sum of squared distances between the measured and modeled control point projections is used frequently as the cost function because it gives the maximum-likelihood parameter estimates assuming isotropic and independent normally distributed measurement errors.

Calibration by nonlinear optimization requires a good initial guess for the camera parameters. Hence, various methods have been proposed for the direct estimation of the parameters. Most methods deal with conventional perspective cameras, but recently effort has been made in developing models and calibration methods for more general cameras. In fact, the choice of a suitable camera model is an important issue in camera calibration. For example, the pinhole camera model, which is based on the ideal perspective projection model and often used for conventional cameras, is not a suitable model for omnidirectional cameras that have a very large field of view. Hence, a recent trend has occurred towards generic calibration techniques that would allow the calibration of various types of cameras.

In this article, we will provide an overview into geometric camera calibration and its current state. However, because the literature for camera calibration is vast and ever-evolving, it is not possible to cover all aspects in detail. Nevertheless, we hope that this article serves as an introduction to the literature in which more details can be found. The article is structured as follows. First, in the Background section, we describe some historical background for camera calibration. Thereafter, we review different camera models with an emphasis on central cameras. After describing camera models we discuss methods for camera calibration. The focus is on our previous works (1, 2). Finally, in the calibration examples section, we present some calibration examples with real cameras.

BACKGROUND

Geometric camera calibration is a prerequisite for image-based metric three-dimensional measurements, and it has a long history in photogrammetry and computer vision. One of the first references is by Conrady (3), who derived an analytical expression for the geometric distortion in a decentered lens system. Conrady's model for decentering distortion was used by Brown (4), who proposed a plumb line method for calibrating radial and decentering distortion. Later the approach used by Brown was commonly adopted in photogrammetric camera calibration (5).

In photogrammetry, the emphasis has traditionally been in the rigorous geometric modeling of the camera and optics. On the other hand, in computer vision it is considered important that the calibration procedure is automatic and fast. For example, the well-known calibration method developed by Tsai (6) was designed to be an automatic and efficient calibration technique for machine vision metrology. This method uses a simpler camera model than Brown (4) and avoids the full-scale nonlinear search by using simplifying approximations. However, because of the increased processing power of personal computers, the nonlinear optimization is not as time-consuming now as it was before. Hence, when the calibration accuracy is important the camera parameters are usually refined by a full-scale nonlinear optimization.

Besides increasing the theoretical understanding, the advances in geometric computer vision have also affected the practice of image-based 3-D reconstruction during the last two decades (7,8). For example, although the traditional photogrammetric approach assumes a precalibrated camera, an alternative approach is to compute a projective reconstruction with an uncalibrated perspective camera. The projective reconstruction is defined up to a 3-D projective transformation, and it can be upgraded to a metric reconstruction by self-calibration (8). In self-calibration, the camera parameters are determined without a calibration object; feature correspondences over multiple images are used instead. However, the conventional calibration is typically more accurate and stable than self-calibration. In fact, self-calibration methods are beyond the scope of this article.

CAMERA MODELS

In this section, we describe several camera models that have appeared in the literature. We concentrate on central

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

cameras, that is, cameras with a single effective viewpoint. Single viewpoint means that all the rays of light that arrive onto the image travel through a single point in space.

Perspective Cameras

The pinhole camera model is the most common camera model, and it is a fair approximation for most conventional cameras that obey the perspective model. Typically these conventional cameras have a small field of view, up to 60°. The pinhole camera model is widely used and simple; essentially, it is just a perspective projection followed by an affine transformation in the image plane (8).

The pinhole camera geometry is illustrated in Fig. 1. In a pinhole camera, the projection rays meet at a single point that is the camera center **C** and its distance from the image plane is the focal length f. By similar triangles, it may be seen in Fig. 1 that the point $(X_c, Y_c, Z_c)^{\mathrm{T}}$ in the camera coordinate frame is projected to the point $(fX_c/Z_c,\ fY_c/Z_c)^{\mathrm{T}}$ in the image coordinate frame. In terms of homogeneous coordinates, this perspective projection can be represented by a 3×4 projection matrix,

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \simeq \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix}$$

where $\simeq$ denotes equality up to scale.

However, instead of the image coordinates $(x, y)^{\mathrm{T}}$, the pixel coordinates $(u, v)^{\mathrm{T}}$ are usually used, and they are obtained by the affine transformation

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} m_u & -m_u \cot\alpha \\ 0 & \dfrac{m_v}{\sin\alpha} \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \tag{1}$$

where $(u_0, v_0)^{\mathrm{T}}$ is the principal point, α is the angle between u and v axis, and m_u and m_v give the number of pixels per unit distance in u and v directions, respectively. The angle α is $\frac{\pi}{2}$ in the conventional case of orthogonal pixel coordinate axes.

In practice, the 3-D point is expressed in some world coordinate system that is different from the camera coordinate system. The motion between these coordinate systems is given by a rotation **R** and translation **t**. Hence, in homogeneous coordinates, the mapping of the 3-D point **X** to its image **m** is given by

$$\begin{aligned} \mathbf{m} &\simeq \begin{bmatrix} m_u & -m_u \cot\alpha & u_0 \\ 0 & \dfrac{m_v}{\sin\alpha} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \mathbf{X} \\ &= \begin{bmatrix} m_u f & -m_u f \cot\alpha & u_0 \\ 0 & \dfrac{m_v}{\sin\alpha} & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{R} \quad \mathbf{t}] \mathbf{X} \\ &= \begin{bmatrix} m_u f & m_u s f & u_0 \\ 0 & m_u \gamma f & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{R} \quad \mathbf{t}] \mathbf{X} \end{aligned} \tag{2}$$

where we have introduced the parameters $\gamma = \frac{m_v}{m_u \sin\alpha}$ and

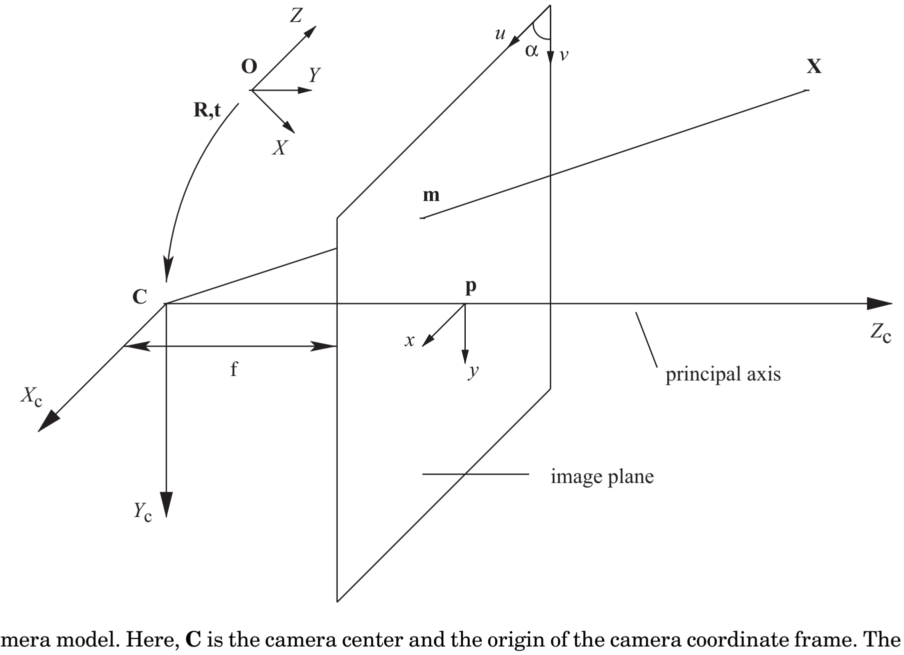


Figure 1. Pinhole camera model. Here, **C** is the camera center and the origin of the camera coordinate frame. The principal point **p** is the origin of the normalized image coordinate system (x, y). The pixel image coordinate system is (u, v).

$s = -\cot\alpha$ to simplify the notation. Because a change in the focal length and a change in the pixel units are indistinguishable above, we may set $m_u = 1$ and write the projection equation in the form

$$\mathbf{m} \simeq \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X} \tag{3}$$

where the upper triangular matrix

$$\mathbf{K} = \begin{bmatrix} f & sf & u_0 \\ 0 & \gamma f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

is the *camera calibration matrix* and contains the five internal parameters of a pinhole camera.

It follows from Eq. (3) that a general pinhole camera may be represented by a homogeneous 3 × 4 matrix

$$\mathbf{P} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} \tag{5}$$

which is called the *camera projection matrix*. If the left hand submatrix **KR** is nonsingular, as it is for perspective cameras, the camera **P** is called a *finite projective camera*.

A camera represented by an arbitrary homogeneous 3 × 4 matrix of rank 3 is called a *general projective camera*. This class covers the *affine cameras*, which have a projection matrix whose last row is (0, 0, 0, 1) up to scale. A common example of an affine camera is the orthographic camera in which the scene points are orthogonally projected onto the image plane (8).

Lens Distortion. The pinhole camera is an idealized mathematical model for real cameras that may often deviate from the ideal perspective imaging model. Hence, the basic pinhole model is often accompanied with lens distortion models for more accurate calibration of real lens systems. The most important type of geometric distortion is the radial distortion that causes an inward or outward displacement of a given image point from its ideal location. Decentering of lens elements causes additional distortion, which also has tangential components.

A commonly used model for lens distortion accounts for radial and decentering distortion (4, 5). According to this model, the corrected image coordinates x', y' are obtained by

$$\begin{aligned} x' &= x + \bar{x}(\kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6 + \cdots) \\ &\quad + (\rho_1(r^2 + 2\bar{x}^2) + 2\rho_2 \bar{x}\bar{y})(1 + \rho_3 r^2 + \cdots) \\ y' &= y + \bar{y}(\kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6 + \cdots) \\ &\quad + (2\rho_1 \bar{x}\bar{y} + \rho_2(r^2 + 2\bar{y}^2))(1 + \rho_3 r^2 + \cdots) \end{aligned} \tag{6}$$

where x and y are the measured coordinates, and

$$\begin{aligned} \bar{x} &= x - x_p \\ \bar{y} &= y - y_p \\ r &= \sqrt{(x - x_p)^2 + (y - y_p)^2} \end{aligned}$$

Here, the center of distortion (x_p, y_p) is a free parameter in addition to the radial distortion coefficients κ_i and decentering distortion coefficients ρ_i. In the traditional photogrammetric approach, the values for the distortion parameters are computed by least-squares adjustment by requiring that images of straight lines are straight after the correction (4). However, the problem with this approach is that not only the distortion coefficients but also the other camera parameters are initially unknown. For example, the formulation above requires that the scales in both coordinate directions are equal, which is not the case with pixel coordinates unless the pixels are square.

In Ref. 1, the distortion model of Eq. (6) was adapted and combined with the pinhole model to make a complete and accurate model for real cameras. This camera model has the form

$$\mathbf{m} = \mathcal{P}(\mathbf{X}) = \mathbf{P}\mathbf{X} + \mathcal{C}(\mathbf{P}\mathbf{X}) \tag{7}$$

where $\mathcal{P}$ denotes the non linear camera projection and **P** is the camera projection matrix of a pinhole camera. The nonlinear part $\mathcal{C}$ is the distortion model derived from Eq. (6). In Ref. 1, two parameters were used for both the radial and decentering distortion (i.e., the parameters κ_1, κ_2 and ρ_1, ρ_2), and it was assumed that the center of distortion coincides with the principal point of the pinhole camera.

Central Omnidirectional Cameras

Although the pinhole model accompanied with lens distortion models is a fair approximation for most conventional cameras, it is not a suitable model for omnidirectional cameras whose field of view is over 180°. The reason is that, when the angle between the incoming light ray and the optical axis of the camera approaches 90°, the perspective projection maps the ray infinitely far in the image, and it is not possible to remove this singularity with the distortion model described above. Hence, more flexible models are needed, and below we discuss different models for central omnidirectional cameras.

Catadioptric Cameras. In a catadioptric omnidirectional camera, the wide field of view is achieved by placing a mirror in front of the camera lens. In a central catadioptric camera, the shape and configuration of the mirror are such that the complete catadioptric system has a single effective viewpoint. It has been shown that the mirror surfaces that produce a single viewpoint are surfaces of revolution whose two-dimensional profile is a conic section (9). Practically useful mirror surfaces used in real central catadioptric cameras are planar, hyperbolic, elliptical, and parabolic. However, a planar mirror does not change the field of view of the camera (9).

The central catadioptric configurations with hyperbolic, elliptical, and parabolic mirrors are illustrated in Fig. 2. To satisfy the single viewpoint constraint, the parabolic mirror is combined with an orthographic camera, whereas the other mirrors are combined with a perspective camera. In each case, the effective viewpoint of the catadioptric system is the focal point of the mirror denoted by F in Fig. 2.

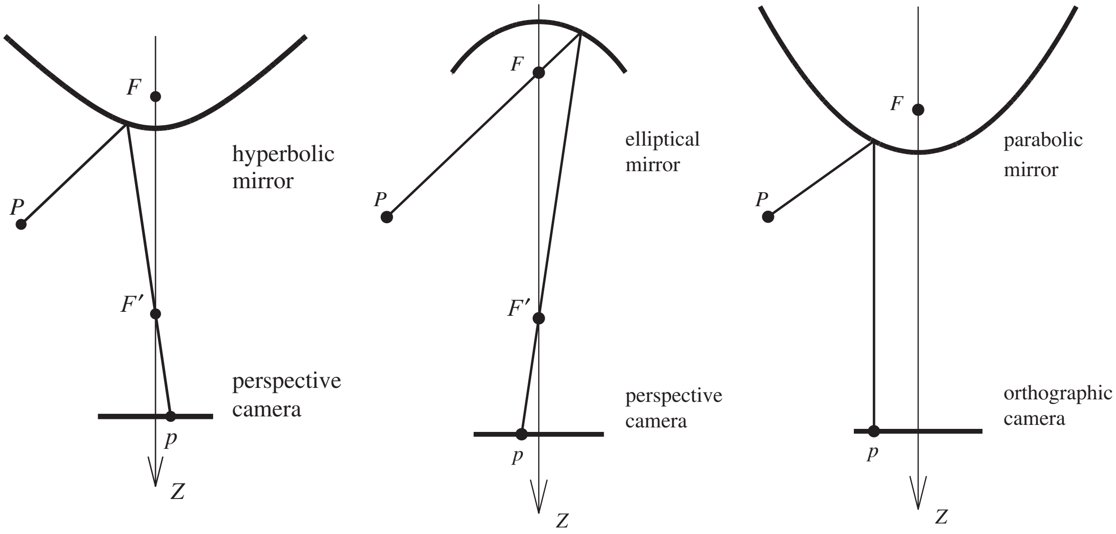


Figure 2. Central catadioptric camera with a hyperbolic, elliptical and parabolic mirror. The Z-axis is the optical axis of the camera and the axis of revolution for the mirror surface. The scene point P is imaged at p. In each case, the viewpoint of the catadioptric system is the focal point of the mirror denoted by F. In the case of hyperbolic and elliptical mirrors, the effective pinhole of the perspective camera must be placed at the other focal point, which is here denoted by F'.

Single-viewpoint catadioptric image formation is well studied (9, 10), and it has been shown that a central catadioptric projection, including the cases shown in Fig. 2, is equivalent to a two-step mapping via the unit sphere (10, 11). As described in Refs. 11 and 12, the unifying model for central catadioptric cameras may be represented by a composed function $\mathcal{H} \circ \mathcal{F}$ so that

$$\mathbf{m} = (\mathcal{H} \circ \mathcal{F})(\mathbf{\Phi}) \tag{8}$$

where $\mathbf{\Phi} = (\theta, \varphi)^{\mathrm{T}}$ defines the direction of the incoming light ray, which is mapped to the image point $\mathbf{m} = (u, v, 1)^{\mathrm{T}}$. Here $\mathcal{F}$ first projects the object point onto a virtual image plane, and then the planar projective transformation $\mathcal{H}$ maps the virtual image point to the observed image point $\mathbf{m}$. The two-step mapping $\mathcal{F}$ is illustrated in Fig. 3(a), in which the object point $\mathbf{X}$ is first projected to $\mathbf{q} = (\cos\varphi \sin\theta, \sin\varphi \sin\theta, \cos\theta)$ on the unit sphere, whose center O is the effective viewpoint of the camera. Thereafter, the point $\mathbf{q}$ is perspectively projected to $\mathbf{x}$ from another point Q so that the line determined by O and Q is perpendicular to the image plane. The distance $l = |OQ|$ is a parameter of the catadioptric camera.

Mathematically the function $\mathcal{F}$ has the form

$$\mathbf{x} = \mathcal{F}(\mathbf{\Phi}) = r(\theta)\begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix} \tag{9}$$

where the function r is the radial projection that does not depend on φ because of radial symmetry. The precise form of r as a function of θ is determined by the parameter l, that is,

$$r = \frac{(l+1)\sin\theta}{l + \cos\theta} \tag{10}$$

This follows from the fact that the corresponding sides of similar triangles must have the same ratio, thus $\frac{r}{\sin\theta} = \frac{l+1}{l+\cos\theta}$, as Fig. 3(a) illustrates.

In a central catadioptric system with a hyperbolic or elliptical mirror, the camera axis does not have to be aligned with the mirror symmetry axis. The camera can be rotated with respect to the mirror as long as the camera center is at the focal point of the mirror. Hence, in the general case, the mapping $\mathcal{H}$ from the virtual image plane to the real image plane is a planar projective transformation (12). However, often the axes of the camera and mirror are close to collinear so that the mapping $\mathcal{H}$ can be approximated with an affine transformation $\mathcal{A}$ (11). That is,

$$\mathbf{m} = \mathcal{A}(\mathbf{x}) = \mathbf{K}\begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} \tag{11}$$

where the upper triangular matrix $\mathbf{K}$ is defined in Eq. (4) and contains five parameters. (Here the affine transformation $\mathcal{A}$ has only five degrees of freedom because we may always fix the camera coordinate frame so that the x-axis is parallel to the u-axis.)

Fish-Eye Lenses. Fish-eye cameras achieve a large field of view by using only lenses, whereas the catadioptric cameras use both mirrors and lenses. Fish-eye lenses are designed to cover the whole hemispherical field in front of the camera, and the angle of view is very large, possibly over 180°. Because it is impossible to project the hemispherical field of view on a finite image plane by a perspective projection, the fish-eye lenses are designed to obey some other projection model (13).

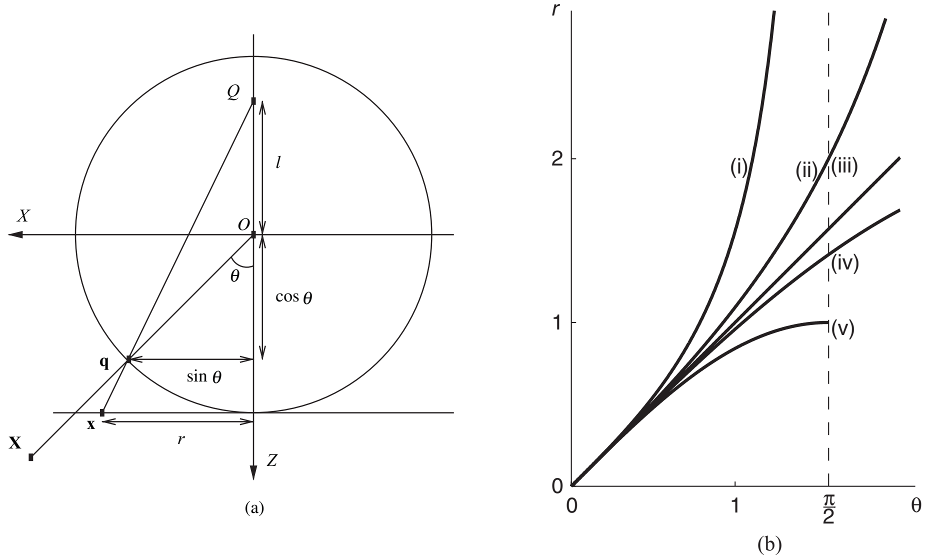


Figure 3. (a) A generic model for a central catadioptric camera (11). The Z-axis is the optical axis, and the plane $Z = 1$ is the virtual image plane. The object point $\mathbf{X}$ is first projected to $\mathbf{q}$ on the unit sphere, and thereafter $\mathbf{q}$ is perspectively projected to $\mathbf{x}$ from Q. (b) The projections of Eqs. (12)–(16).

The perspective projection of a pinhole camera can be represented by the formula

$$r = \tan\theta \qquad \text{(i. perspective projection)} \tag{12}$$

where θ is the angle between the principal axis and the incoming ray, and r is the distance between the image point and the principal point measured on a virtual image plane that is placed at a unit distance from the pinhole. Fish-eye lenses instead are usually designed to obey one of the following projections:

$$r = 2\tan(\theta/2) \qquad \text{(ii. stereographic projection)} \tag{13}$$

$$r = \theta \qquad \text{(iii. equidistance projection)} \tag{14}$$

$$r = 2\sin(\theta/2) \qquad \text{(iv. equisolid angle projection)} \tag{15}$$

$$r = \sin(\theta) \qquad \text{(v. orthogonal projection)} \tag{16}$$

where the equidistance projection is perhaps the most common model. The behavior of the different projection models is illustrated in Fig. 3(b).

Although the central catadioptric cameras and fish-eye cameras have a different physical construction, they are not too different from the viewpoint of mathematical modeling. In fact, the radial projection curves defined by Eq. (10) are similar to those shown in Fig. 3(b). In particular, when $l = 0$ Eq. (10) defines the perspective projection, $l = 1$ gives the stereographic projection (because $\tan\frac{\theta}{2} = \frac{\sin\theta}{1+\cos\theta}$), and on the limit $l \to \infty$ we obtain the orthogonal projection. Hence, the problem of modeling radially symmetric central cameras is essentially reduced to modeling radial projection functions such as those in Fig. 3(b).

Generic Model for Central Cameras. Here we describe a generic camera model, which was proposed in Ref. 2 and is suitable for central omnidirectional cameras as well as for conventional cameras. As discussed above, the radially symmetric central cameras may be represented by Eq. (10) where the function $\mathcal{F}$ is given by Eq. (9). The radial projection function r in $\mathcal{F}$ is an essential part of the model. If r is fixed to have the form of Eq. (12), then the model is reduced to the pinhole model. However, modeling of omnidirectional cameras requires a more flexible model and here we consider the model

$$r = k_1\theta + k_2\theta^3 + k_3\theta^5 + k_4\theta^7 + k_5\theta^9 + \cdots \tag{17}$$

which allows good approximation of all the projections in Fig. 3(b). In Ref. 2 it was shown that the first five terms, up to the ninth power of θ, give enough degrees of freedom for accurate approximation of different projection curves. Hence, the generic camera model used here contains five parameters in the radial projection function r.

However, real lenses may deviate from precise radial symmetry and, therefore, the radially symmetric model above was supplemented with an asymmetric part in Ref. 2. Hence, instead of Eq. (10) the camera model proposed in Ref. 2 has the form

$$\mathbf{m} = (\mathcal{A} \circ \mathcal{D} \circ \mathcal{F}) \times (\boldsymbol{\Phi}) \tag{18}$$

where $\mathcal{D}$ is the asymmetric distortion function so that $\mathcal{D} \circ \mathcal{F}$ gives the distorted image point x_d, which is then transformed to pixel coordinates by the affine transformation $\mathcal{A}$. In detail,

$$\begin{aligned}\mathbf{x}_\mathrm{d} &= (\mathcal{D} \circ \mathcal{F})(\boldsymbol{\Phi}) \\ &= r(\theta)\mathbf{u}_r(\varphi) + \Delta_\mathrm{r}(\theta, \varphi)\mathbf{u}_r(\varphi) + \Delta_\mathrm{t}(\theta, \varphi)\mathbf{u}_\varphi(\varphi)\end{aligned} \tag{19}$$

where $\mathbf{u}_r(\varphi)$ and $\mathbf{u}_\varphi(\varphi)$ are the unit vectors in the radial and tangential directions and

$$\begin{aligned}\Delta_\mathrm{r}(\theta, \varphi) = (g_1\theta + g_2\theta^3 + g_3\theta^5)(i_1\cos\varphi + i_2\sin\varphi + i_3\cos 2\varphi \\ + i_4\sin 2\varphi)\end{aligned} \tag{20}$$

$$\begin{aligned}\Delta_\mathrm{t}(\theta, \varphi) = (h_1\theta + h_2\theta^3 + h_3\theta^5)(j_1\cos\varphi + j_2\sin\varphi + j_3\cos 2\varphi \\ + j_4\sin 2\varphi)\end{aligned} \tag{21}$$

Here both the radial and tangential distortion terms contain seven parameters.

The asymmetric part in Eq. (19) models the imperfections in the optical system in a somewhat similar manner as the distortion model by Brown (4). However, instead of rigorous modeling of optical distortions, here the aim is to provide a flexible mathematical distortion model that is just fitted to agree with the observations. This approach is often practical because several possible sources of imperfections may exist in the optical system, and it is difficult to model all of them in detail.

The camera model defined above is denoted by $\mathcal{M}_{24}$ in the following because the number of parameters is 24: $\mathcal{F}$ and $\mathcal{A}$ have both 5 parameters and $\mathcal{D}$ has 14 parameters. However, often it is assumed that the pixel coordinate system is orthogonal [i.e., $s = 0$ in Eq. (4)], so that the number of parameters in $\mathcal{A}$ is only four. This model is denoted by $\mathcal{M}_{23}$. In addition, sometimes it may be useful to leave out the asymmetric part to avoid overfitting. The corresponding radially symmetric models are here denoted by $\mathcal{M}_9$ and $\mathcal{M}_6$. The model $\mathcal{M}_6$ contains only two terms in Eq. (17) whereas $\mathcal{M}_9$ contains five.

Other Distortion Models. In addition to the camera models described in the previous sections, also several other models have appeared in the literature. For example, the so-called division model for radial distortion is defined by

$$r = \frac{r_\mathrm{d}}{1 - cr_\mathrm{d}^2} \tag{22}$$

where r_d is the measured distance between the image point and the distortion center and r is the ideal undistorted distance (14, 15). A positive value of the distortion coefficient c corresponds to the typical case of barrel distortion (14). However, the division model is not suitable for cameras whose field of view exceeds 180°. Hence, other models must be used in this case and, for instance, the two-parametric projection model

$$r = \frac{a - \sqrt{a^2 - 4b\theta^2}}{2b\theta} \tag{23}$$

has been used for fish-eye lenses (16). Furthermore, a parameter-free method for determining the radial distortion was proposed in Ref. 17.

Noncentral Cameras

Most real cameras are strictly speaking noncentral. For example, in the case of parabolic mirror in Fig. 2, it is difficult to align the mirror axis and the axis of the camera precisely. Likewise, in the hyperbolic and elliptic configurations, the precise positioning of the optical center of the perspective camera in the focal point of the mirror is practically infeasible. In addition, if the shape of the mirror is not a conic section or the real cameras are not truly orthographic or perspective the configuration is noncentral. However, in practice the camera is usually negligibly small compared with the viewed region so that it is effectively point-like. Hence, the central camera models are widely used and tenable in most situations so that also here, in this article, we concentrate on central cameras.

Still, some works suggest the single viewpoint constraint is relaxed and a noncentral camera model is used. For example, a completely generic camera calibration approach was discussed in Refs. 18 and 19, in which a nonparametric camera model was used. In this model, each pixel of the camera is associated with a ray in 3-D and the task of calibration is to determine the coordinates of these rays in some local coordinate system. In addition, work has been published about designing mirrors for noncentral catadioptric systems that are compliant with predefined requirements (20).

Finally, as a generalization of central cameras, we would like to mention the axial cameras in which all the projection rays go through a single line in space (19). For example, a catadioptric camera that consists of a mirror and a central camera is an axial camera if the mirror is any surface of revolution and the camera center lies on the mirror axis of revolution. A central camera is a special case of an axial camera. The equiangular (21, 22) and equiareal (23) catadioptric cameras are other classes of axial cameras. In equiareal cameras, the projection is area preserving, whereas the equiangular mirrors are designed so that the radial distance measured from the center of symmetry in the image is linearly proportional to the angle between the incoming ray and the optical axis.

CALIBRATION METHODS

Camera calibration is the process of determining the parameters of the camera model. Here we consider conventional calibration techniques that use images of a calibration object that contains control points in known positions. The choice of a suitable calibration algorithm depends on the camera model, and below we describe methods for calibrating both perspective and omnidirectional central cameras.

Although the details of the calibration procedure may differ depending on the camera, the final step of the procedure is usually the refinement of camera parameters by nonlinear optimization regardless of the camera model. The cost function normally used in the minimization is the sum of squared distances between the measured and modeled control point projections, that is

$$\sum_{j=1}^{N}\sum_{i=1}^{N}\delta_j^i d(\mathbf{m}_j^i, \hat{\mathbf{m}}_j^i)^2 \tag{24}$$

where $\mathbf{m}_j^i$ contains the measured image coordinates of the control point i in the view j, the binary variable δ_j^i indicates whether the control point i is observed in the view j and

$$\hat{\mathbf{m}}_j^i = \mathcal{P}_j(\mathbf{X}^i) \tag{25}$$

is the projection of the control point $\mathbf{X}^i$ in the view j. Here, $\mathcal{P}_j$ denotes the camera projection in the view j and it is determined by the external and internal camera parameters. The justification for minimizing Eq. (24) is that it gives the maximum likelihood solution for the camera parameters when the image measurement errors obey a zero-mean isotropic Gaussian distribution. However, the successful minimization of Eq. (24) with standard local optimization methods requires a good initial guess for the parameters. Methods for computing such an initial guess are discussed below.

Perspective Cameras

In the case of a perspective camera, the camera projection $\mathcal{P}$ is represented by a 3×4 matrix $\mathbf{P}$ as described in the section on perspective cameras. In general, the projection matrix $\mathbf{P}$ can be determined from a single view of a noncoplanar calibration object using the Direct Linear Transform (DLT) method, which is described below. Then, given $\mathbf{P}$, the parameters $\mathbf{K}$ and $\mathbf{R}$ in Eq. (5) are obtained by decomposing the left 3×3 submatrix of $\mathbf{P}$ using the QR-decomposition whereafter also $\mathbf{t}$ can be computed (8).

On the other hand, if the calibration object is planar and the internal parameters in $\mathbf{K}$ are all unknown, several views are needed. In this case, the constant camera calibration matrix $\mathbf{K}$ can be determined first using the approach described below in the section on planar calibration object. Thereafter, the view dependent parameters $\mathbf{R}_j$ and $\mathbf{t}_j$ can be computed and used for initializing the nonlinear optimization. If the perspective camera model is accompanied with a lens distortion model, then the distortion parameters in Eq. (7) may be initialized by setting them to zero (24).

Noncoplanar Calibration Object. Assuming that the known space points $\mathbf{X}^i$ are projected at the image points $\mathbf{m}^i$, the unknown projection matrix $\mathbf{P}$ can be estimated using the DLT method (8, 25, 26). The projection equation gives

$$\mathbf{m}^i \simeq \mathbf{P}\mathbf{X}^i \tag{26}$$

which can be written in the equivalent form

$$\mathbf{m}^i \times \mathbf{P}\mathbf{X}^i = 0 \tag{27}$$

where the unknown scale in Eq. (26) is eliminated by the cross product. The equations above are linear in the elements of $\mathbf{P}$ so they can be written in the form

$$\mathbf{A}^i\mathbf{v} = 0 \tag{28}$$

where

$$\mathbf{v} = (P_{11}\,P_{12}\,P_{13}\,P_{14}\,P_{21}\,P_{22}\,P_{23}\,P_{24}\,P_{31}\,P_{32}\,P_{33}\,P_{34})^{\mathrm{T}} \tag{29}$$

and

$$\mathbf{A}^i = \begin{bmatrix} 0^{\mathrm{T}} & -m_3^i\mathbf{X}^{i^{\mathrm{T}}} & m_2^i\mathbf{X}^{i^{\mathrm{T}}} \\ m_3^i\mathbf{X}^{i^{\mathrm{T}}} & 0^{\mathrm{T}} & -m_1^i\mathbf{X}^{i^{\mathrm{T}}} \\ -m_2^i\mathbf{X}^{i^{\mathrm{T}}} & m_1^i\mathbf{X}^{i^{\mathrm{T}}} & 0^{\mathrm{T}} \end{bmatrix} \tag{30}$$

Thus, each point correspondence provides three equations, but only two of them are linearly independent. Hence, given $M \geq 6$ point correspondences, we get an overdetermined set of equations $\mathbf{A}\mathbf{v} = 0$, where the matrix $\mathbf{A}$ is obtained by stacking the matrices $\mathbf{A}^i, i = 1, \cdots, M$. In practice, because of the measurement errors no exact solution to these equations exist, but the solution $\mathbf{v}$ that minimizes $||\mathbf{A}\mathbf{v}||$ can be computed using the singular value decomposition of $\mathbf{A}$ (8). However, if the points $\mathbf{X}^i$ are coplanar, then ambiguous solutions exist for $\mathbf{v}$, and hence the DLT method is not applicable in such case.

The DLT method for solving $\mathbf{P}$ is a linear method that minimizes the algebraic error $||\mathbf{A}\mathbf{v}||$ instead of the geometric error in Eq. (24). This method implies that, in the presence of noise, the estimation result depends on the coordinate frames in which the points are expressed. In practice, it has been observed that a good idea is to normalize the coordinates in both $\mathbf{m}^i$ and $\mathbf{X}^i$ so that they have zero mean and unit variance. This kind of normalization may significantly improve the estimation result in the presence of noise (8).

Planar Calibration Object. In the case of a planar calibration object, the camera calibration matrix $\mathbf{K}$ can be solved by using several views. This approach was described in Refs. 24 and 27, and it is briefly summarized in the following.

The mapping between a scene plane and its perspective image is a planar homography. Because one may assume that the calibration plane is the plane $Z = 0$, the homography is defined by

$$\mathbf{m} \simeq \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\begin{pmatrix}X\\Y\\0\\1\end{pmatrix} = \mathbf{H}\begin{pmatrix}X\\Y\\1\end{pmatrix} \tag{31}$$

where the 3×3 homography matrix

$$\mathbf{H} = \mathbf{K}[\,\mathbf{r}^1 \quad \mathbf{r}^2 \quad \mathbf{t}\,] \tag{32}$$

where the columns of the rotation matrix $\mathbf{R}$ are denoted by $\mathbf{r}^i$. The outline of the calibration method is to first determine the homographies for each view and then use Eq. (32) to derive constraints for the determination of $\mathbf{K}$. The constraints for $\mathbf{K}$ are described in more detail below. Methods for determining a homography from point correspondences are described, for example, in Ref. 8.

Denoting the columns of $\mathbf{H}$ by $\mathbf{h}^i$ and using the fact that $\mathbf{r}^1$ and $\mathbf{r}^2$ are orthonormal one obtains from Eq. (32) that

$$\mathbf{h}^{1^{\mathrm{T}}}\mathbf{K}^{-\mathrm{T}}\mathbf{K}^{-1}\mathbf{h}^2 = 0 \tag{33}$$

$$\mathbf{h}^{1^{\mathrm{T}}}\mathbf{K}^{-\mathrm{T}}\mathbf{K}^{-1}\mathbf{h}^1 = \mathbf{h}^{2^{\mathrm{T}}}\mathbf{K}^{-\mathrm{T}}\mathbf{K}^{-1}\mathbf{h}^2 \tag{34}$$

Thus, each homography provides two constraints that may be written as linear equations on the elements of the homogeneous symmetric matrix $\boldsymbol{\omega} = \mathbf{K}^{-\mathrm{T}}\mathbf{K}^{-1}$. Hence, the system of equations, derived from Eqs. (33) and (34) above, is of the form $\mathbf{Av} = 0$, where the vector of unknowns $\mathbf{v} = (\omega_{11}, \omega_{12}, \omega_{13}, \omega_{22}, \omega_{23}, \omega_{33})^{\mathrm{T}}$ consists of the elements of $\boldsymbol{\omega}$. Matrix $\mathbf{A}$ has $2N$ rows, where N is the number of views. Given three or more views, the solution vector $\mathbf{v}$ is the right singular vector of $\mathbf{A}$ that corresponds to the smallest singular value. When $\boldsymbol{\omega}$ is solved (up to scale), one may compute the upper triangular matrix $\mathbf{K}$ by Cholesky-factorization. Thereafter, given $\mathbf{H}$ and $\mathbf{K}$, the external camera parameters can be retrieved from Eq. (32). Finally, the obtained estimates should be refined by minimizing the error of Eq. (24) in all views.

Omnidirectional Cameras

In this section, we describe a method that uses a planar calibration pattern to calibrate the parameters of the camera model introduced in the section entitled "Generic Model for Central Cameras" (2). Planar calibration patterns are very common because they are easy to create. In fact, often also the noncoplanar calibration objects contain planar patterns because they usually consist of two or three different planes.

The calibration procedure consists of four steps that are described below. We assume that M control points are observed in N views so that, for each view j, a rotation matrix $\mathbf{R}_j$ and a translation vector $\mathbf{t}_j$ exist, which describe the orientation and position of the camera with respect to the calibration object. In addition, we assume that the object coordinate frame is chosen so that the plane $Z = 0$ contains the calibration pattern and the coordinates of the control point i are denoted by $\mathbf{X}^i = (X^i, Y^i, 0)^{\mathrm{T}}$. The corresponding homogeneous coordinates in the calibration plane are denoted by $\mathbf{x}_{\mathrm{p}}^i = (X^i, Y^i, 1)^{\mathrm{T}}$ and the observed image coordinates in the view j are $\mathbf{m}_{\mathrm{j}}^{\mathrm{i}} = (u_j^i, v_j^i, 1)^{\mathrm{T}}$.

Step 1: Initialization of internal parameters.
In the first three steps of the calibration procedure, we use the camera model $\mathcal{M}_6$, which contains only six nonzero internal parameters [i.e., the parameters $(k_1, k_2, f, \gamma, u_0, v_0)$]. These parameters are initialized using a priori knowledge about the camera. For example, the principal point (u_0, v_0) is usually located close to the image center, γ has a value close to 1, and f is the focal length in pixels. The initial values for k_1 and k_2 can be obtained by fitting the model $r = k_1\theta + k_2\theta^3$ to the desired projection curve in Fig. 3(b).

Step 2: Back-projection and computation of homographies.
Given the internal parameters, we may back-project the observed points $\mathbf{m}_j^i$ onto the unit sphere centered at the camera origin. For each $\mathbf{m}_j^i$ the back-projection gives the direction $\boldsymbol{\Phi}_j^i = (\theta_j^i, \varphi_j^i)^{\mathrm{T}}$ and the points on the unit sphere are defined by $\mathbf{q}_j^i = (\sin\varphi_j^i \sin\theta_j^i,\ \cos\varphi_j^i \sin\theta_j^i,\ \cos\theta_j^i)^{\mathrm{T}}$. Because the mapping between the points on the calibration plane and on the unit sphere is a central projection, a planar homography $\mathbf{H}_j$ exists so that $\mathbf{q}_j^i \simeq \mathbf{H}_j\mathbf{x}_{\mathrm{p}}^{\mathrm{i}}$. For each view j, the homography $\mathbf{H}_j$ is estimated from the correspondences $(\mathbf{q}_j^i, \mathbf{x}_{\mathrm{p}}^{\mathrm{i}})$. In detail, the initial estimate for $\mathbf{H}_j$ is computed by the linear algorithm (8), and it is then refined by minimizing $\sum_i \sin^2\alpha_j^i$,, where α_j^i is the angle between the unit vectors $\mathbf{q}_j^i$ and $\mathbf{H}_j\mathbf{x}_{\mathrm{p}}^{\mathrm{i}}/||\mathbf{H}_j\mathbf{x}_{\mathrm{p}}^{\mathrm{i}}||$.

Step 3: Initialization of external parameters.
The initial values for the external camera parameters are extracted from the homographies $\mathbf{H}_j$. It holds that

$$\mathbf{q}_j^i \simeq \begin{bmatrix}\mathbf{R}_j & \mathbf{t}_j\end{bmatrix}\begin{pmatrix}X^i\\Y^i\\0\\1\end{pmatrix} = [\,\mathbf{r}_j^1 \quad \mathbf{r}_j^2 \quad \mathbf{t}_j\,]\begin{pmatrix}X^i\\Y^i\\1\end{pmatrix}$$

which implies $\mathbf{H}_j \simeq [\mathbf{r}_j^1\,\mathbf{r}_j^2\,\mathbf{t}_j]$. Hence,

$$\mathbf{r}_j^1 = \lambda_j\mathbf{h}_j^1, \quad \mathbf{r}_j^2 = \lambda_j\mathbf{h}_j^2, \quad \mathbf{r}_j^3 = \mathbf{r}_j^1 \times \mathbf{r}_j^2, \quad \mathbf{t}_j = \lambda_j\mathbf{h}_j^3$$

where $\lambda_j = \pm \,||\, \mathbf{h}_j^1 \,||^{-1}$. The sign of λ_j can be determined by requiring that the camera is always on the front side of the calibration plane. However, the obtained rotation matrices may not be orthogonal because of estimation errors. Hence, the singular value decomposition is used to compute the closest orthogonal matrices in the sense of Frobenius norm, which are then used for initializing each $\mathbf{R}_j$.

Step 4: Minimization of projection error.

If a camera model with more than six parameters is used, then the additional camera parameters are initialized to zero at this stage. As we have the estimates for the internal and external camera parameters, we may compute the imaging function $\mathcal{P}_j$ for each camera, in which a control point is projected to $\hat{\mathbf{m}}^i_j = \mathcal{P}_j(\mathbf{X}^i)$. Finally, all the camera parameters are refined by minimizing Eq. (24) using nonlinear optimization, such as the Levenberg-Marquardt algorithm.

Precise Calibration with Circular Control Points

To achieve an accurate calibration, we have used a calibration plane with circular control points because the centroids of the projected circles can be detected with a sub-pixel level of accuracy (28). However, in this case the problem is that the centroid of the *projected* circle is not the image of the center of the original circle. Therefore, because $\mathbf{m}^i_j$ in Eq. (24) is the measured centroid, we should not project the centers as points $\hat{\mathbf{m}}^i_j$ because it may introduce bias in the estimates. Of course, it is not an issue if the control points are really pointlike, such as the corners of a checkerboard pattern.

In the case of a perspective camera, the centroids of the projected circles can be solved analytically given the camera parameters and the circles on the calibration plane (1). However, in the case of more generic camera models, the projection is more complicated and the centroids of the projected circles must be solved numerically (2).

CALIBRATION EXAMPLES

In this section, we illustrate camera calibration with real examples involving different kinds of cameras.

Conventional Cameras with Moderate Lens Distortion

The first calibrated camera was a Canon S1 IS digital camera with a zoom lens whose focal length range is 5.8–58.0 mm, which corresponds to a range of 38–380 mm in the 35-mm film format. The calibration was performed with the zoom fixed to 11.2 mm. Hence, the diagonal field of view was about 30°, which is a relatively narrow angle. The other camera was a Sony DFW-X710 digital video camera equipped with a Cosmicar H416 wide-angle lens. The focal length of this wide-angle lens is 4.2 mm, and it produces a diagonal field of view of about 80°.

Both cameras were calibrated by using a planar calibration pattern that contains white circles on black background. The pattern was displayed on a digital plasma display (Samsung PPM50M6HS) whose size is 1204 × 724 mm^2. A digital flat-screen display provides a reasonably planar object and because of its self-illuminating property, it is easy to avoid specular reflections that might otherwise hamper the accurate localization of the control points. Some examples of the calibration images are shown in Fig. 4. The image in Fig. 4(a) was taken with the narrow-angle lens and the image in Fig. 4(b) with the wide-angle lens. The resolution of the images is 2048 × 1536 pixels and 1024 × 768 pixels, respectively. The lens distortion is clearly visible in the wide-angle image.

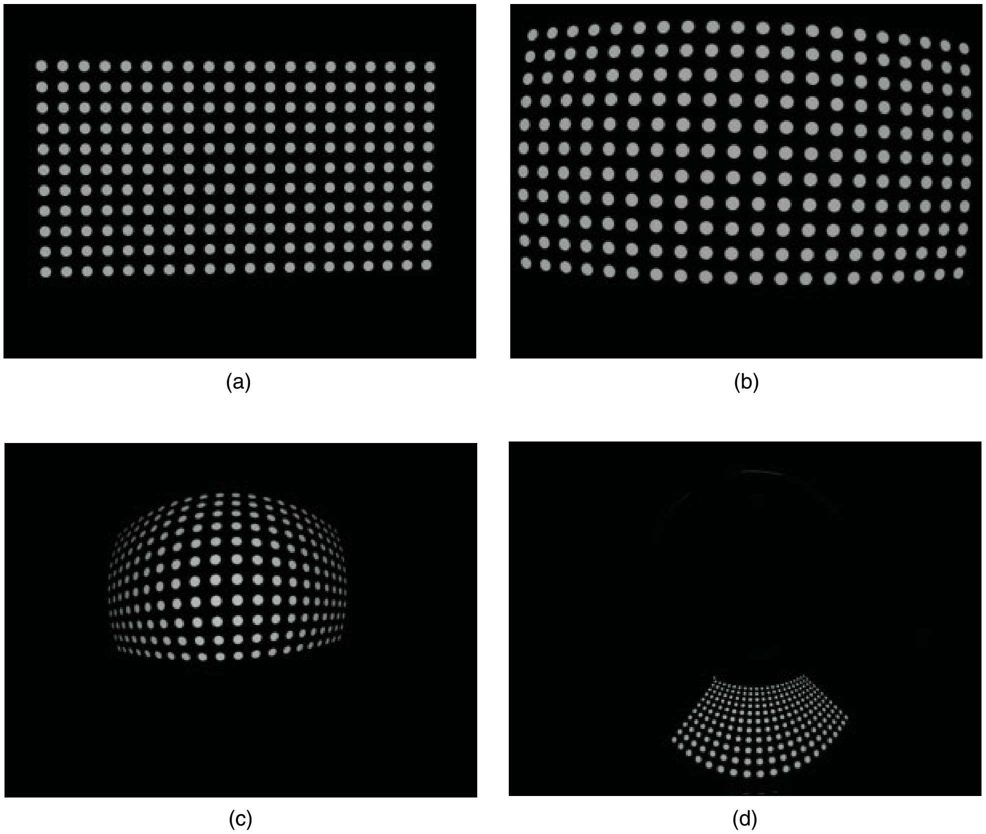
The number of calibration images was six for both cameras, and each image contained 220 control points. The images were chosen so that the whole image area was covered by the control points. In addition to the set of calibration images we took another set of images that likewise contained six images in which the control points were distributed onto the whole image area. These images were used as a test set to validate the results of calibration. In all cases, the control points were localized from the images by computing their grayscale centroids (28).

The cameras were calibrated using four different camera models. The first model, denoted by $\mathcal{M}_\mathrm{p}$, was the skew-zero pinhole model accompanied with four distortion parameters. This model was used in Ref. 1, and it is described in the section on lens distortion. The other three models were the models $\mathcal{M}_6$, $\mathcal{M}_9$, and $\mathcal{M}_{23}$ defined in the section entitled "Generic Model for Central Cameras." All calibrations were performed by minimizing the sum of squared projection errors as described in the section on calibration methods. The computations were carried out by using the publicly available implementations of the calibration procedures proposed in Refs. 1 and 2.[1]

The calibration results are shown in Table 1 in which the first four rows give the figures for the narrow-angle and wide-angle lenses introduced above. The first and third row in Table 1 contain the RMS calibration errors, that is, the root-mean-squared distances between the measured and modeled control point projections in the calibration images. The second and fourth row show the RMS projection errors in the test images. In the test case, the values of the internal camera parameters were those estimated from the calibration images, and only the external camera parameters were optimized by minimizing the sum of the squared projection errors.

The results illustrate that the most flexible model $\mathcal{M}_{23}$ generally performs the best. However, the difference between the models $\mathcal{M}_6$, $\mathcal{M}_9$, and $\mathcal{M}_{23}$ is not large for the narrow-angle and wide-angle lens. The model $\mathcal{M}_\mathrm{p}$ performs well with the narrow-angle lens, but it seems that the other models are better in modeling the severe radial distortion of the wide-angle lens. The relatively low values of the test set error indicate that the risk of overfitting is small. This risk could be decreased even more by using more calibration images. The RMS error is somewhat larger for the narrow-angle lens than for the wide-angle lens, but it may be caused by the fact that the image resolution is higher in the narrow-angle case. Hence, the pixel units are not directly comparable for the different cameras.

[1] http://www.ee.oulu.fi/mvg/page/downloads.



(a) (b)

(c) (d)

Figure 4. Images of the calibration pattern taken with different types of cameras, a) narrow-angle lens, b) wide-angle lens, c) fish-eye lens, d) hyperbolic mirror combined with a narrow-angle lens.

Omnidirectional Cameras

We calibrated a fish-eye lens camera and two different catadioptric cameras. The fish-eye lens that was used in the experiments is the ORIFL190-3 lens manufactured by Omnitech Robotics (Englewood, CA). This lens has a 190° field of view, and it was attached to a PointGrey Dragonfly (PointGrey, Vancouver, BC, Canada) color video camera, which has a resolution of 1024 × 768 pixels. The catadioptric cameras were constructed by placing two different mirrors in front of the Canon S1 IS camera (Canon, Tokyo, Japan), which has a resolution of 2048 × 1536 pixels. The first mirror was a hyperbolic mirror from Eizoh (Tokyo, Japan) and the other mirror was an equiangular mirror from Kaidan (Feasterville, PA). The field of view provided by the hyperbolic mirror is such that when the mirror is placed above the camera so that the optical axis is vertical, the camera observes about 30° above and 50° below the horizon (the region directly below the mirror is obscured by the camera). The equiangular mirror by Kaidan provides a slightly larger view of field because it observes about 50° above and below the horizon. In the azimuthal direction the viewing angle is 360° for both mirrors.

Table 1. The RMS projection errors in pixels

	$\mathcal{M}_p$	$\mathcal{M}_6$	$\mathcal{M}_9$	$\mathcal{M}_{23}$
narrow-angle lens	0.293	0.339	0.325	0.280
test set error	0.249	0.309	0.259	0.236
wide-angle lens	0.908	0.078	0.077	0.067
test set error	0.823	0.089	0.088	0.088
fish-eye lens	-	0.359	0.233	0.206
test set error	-	0.437	0.168	0.187
hyperbolic mirror	-	4.178	1.225	0.432
test set error	-	3.708	1.094	0.392
equiangular mirror	-	2.716	0.992	0.788
test set error	-	3.129	1.065	0.984

Because the field of view of all three omnidirectional cameras exceeds a hemisphere, the calibration was not performed with the model $\mathcal{M}_p$, which is based on the perspective projection model. Hence, we only report the results obtained with the central camera models $\mathcal{M}_6$, $\mathcal{M}_9$, and $\mathcal{M}_{23}$. The calibration experiments were conducted in a similar way as for the conventional cameras above, and the same calibration object was used. However, here the number of images was 12 both in the calibration set and the test set. The number of images was increased to have a better coverage for the wider field of view.

The results are illustrated in Table 1, where it can be observed that the model $\mathcal{M}_{23}$ again shows best performance. The radially symmetric model $\mathcal{M}_9$ performs almost equally well with the fish-eye camera and the equiangular camera. However, for the hyperbolic camera, the additional degrees of freedom in $\mathcal{M}_{23}$ clearly improve the calibration accuracy. This improvement might be an indication that the optical axis of the camera is not aligned precisely with the mirror axis. Nevertheless, the asymmetric central camera model $\mathcal{M}_{23}$ provides a good approximation for this catadioptric camera. Likewise, here the central model seems to be tenable also for the equiangular catadioptric camera, which is strictly speaking noncentral. Note that the sensor resolution of the fish-eye camera was different than that of the catadioptric cameras.

CONCLUSION

Geometric camera calibration is a prerequisite for image-based, accurate 3-D measurements, and it is therefore a fundamental task in computer vision and photogrammetry. In this article, we presented a review of calibration techniques and camera models that commonly occur in applications. We concentrated on the traditional calibration approach in which the camera parameters are estimated by using a calibration object whose geometry is known. The emphasis was on central camera models, which are the most common in applications and provide a reasonable approximation for a wide range of cameras. The process of camera calibration was additionally demonstrated with practical examples in which several different kinds of real cameras were calibrated.

Camera calibration is a wide-ranging topic and much research exists that was not possible to be covered here. For example, recently research efforts have focused toward completely generic camera calibration techniques that could be used for all kinds of cameras, including the noncentral ones. In addition, camera self-calibration is an active research area that was not discussed in the scope of this article. However, camera calibration using a calibration object and a parametric camera model, as discussed here, is the most viable approach when a high level of accuracy is required.

BIBLIOGRAPHY

1. J. Heikkilä, Geometric camera calibration using circular control points, *IEEE Trans. Patt. Anal. Mach. Intell.*, **22**(10): 1066–1077, 2000.
2. J. Kannala and S. S. Brandt, A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses, *IEEE Trans. Patt. Anal. Mach. Intell.*, **28**(8): 1335–1340, 2006.
3. A. Conrady, Decentering lens systems, *Monthly Notices Roy. Astronom. Soc.*, **79**: 384–390, 1919.
4. D. C. Brown, Close-range camera calibration, *Photogramm. Engineer.*, **37**(8): 855–866, 1971.
5. C. C. Slama, ed., *Manual of Photogrammetry*, Am. Soc. Photogrammetry, 4th ed., Falls Church, VA, 1980.
6. R. Tsai, A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses, *IEEE J. Robot. Automat.*, **RA-3**(4): 323–344, 1987.
7. O. Faugeras and Q.-T. Luong, *The Geometry of Multiple Images*, Cambridge, MA: The MIT Press, 2001.
8. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge: Cambridge University Press, 2003.
9. S. Baker and S. K. Nayar, A theory of single-viewpoint catadioptric image formation, *Internat. J. Comp. Vis.*, **35**(2): 11–22, 1999.
10. C. Geyer and K. Daniilidis, Catadioptric projective geometry, *Internat. J. Comp. Vis.*, **45**(3): 223–243, 2001.
11. X. Ying and Z. Hu, Catadioptric camera calibration using geometric invariants, *IEEE Trans. Patt. Ana. Mach. Intell.*, **26**(10): 1260–1271, 2004.
12. J. P. Barreto and H. Araujo, Geometric properties of central catadioptric line images and their application in calibration, *IEEE Trans. Patt. Anal. Mach. Intell.*, **27**(8): 1327–1333, 2005.
13. K. Miyamoto, Fish eye lens, *J. Optical Soc. Am.*, **54**(8): 1060–1061, 1964.
14. C. Bräuer-Burchardt and K. Voss, A new algorithm to correct fish-eye- and strong wide-angle-lens-distortion from single images, *Proc. International Conference on Image Processing*, 2001.
15. A. Fitzgibbon, Simultaneous linear estimation of multiple view geometry and lens distortion, *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2001.
16. B. Mičušík and T. Pajdla, Structure from motion with wide circular field of view cameras, *IEEE Trans. Patt. Anal. Mach. Intell.*, **28**(7): 1135–1149, 2006.
17. R. Hartley and S. B. Kang , Parameter-free radial distortion correction with center of distortion estimation, *IEEE Trans. Patt. Anal. Mach. Intell*. 1309–1321, 2008.
18. M. D. Grossberg and S. K. Nayar, A general imaging model and a method for finding its parameters, *Proc. International Conference on Computer Vision*, 2001, pp. 108–115.
19. S. Ramalingam, Generic Imaging Models: Calibration and 3D Reconstruction Algorithms, Ph.D. dissertation. Institut National Polytechnique de Grenoble, 2006.
20. R. Swaminathan , M. D. Grossberg , and S. K. Nayar , Designing mirrors for catadioptric systems that minimize image errors, *Proc. Workshop on Omnidirectional Vision*, 2004.
21. J. S. Chahl, and M. V. Srinivasan, Reflective surfaces for panoramic imaging, *Applied Optics*, **36**(31): 8275–8285, 1997.
22. M. Ollis, H. Herman, and S. Singh, Analysis and design of panoramic stereo vision using equi-angular pixel cameras, Pittsburgh, PA: Carnegie Mellon University, CMU-RI-TR-99-04, 1999.
23. R. A. Hicks and R. K. Perline, Equi-areal catadioptric sensors, *Proc. Workshop on Omnidirectional Vision*, 2002.
24. Z. Zhang, A flexible new technique for camera calibration, *IEEE Trans. Patt. Anal. Mach. Intell*. **22**(11): 1330–1334 2000.
25. Y. I. Abdel-Aziz and H. M. Karara , Direct linear transformation from comparator to object space coordinates in close-range photogrammetry, *Symposium on Close-Range Photogrammetry*, 1971.
26. I. Sutherland, Three-dimensional data input by tablet, *Proc. IEEE*. **62**: 453–461, 1974.
27. P. Sturm and S. Maybank, On plane based camera calibration: A general algorithm, singularities, applications, *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1999, pp. 432–437.
28. J. Heikkilä, and O. Silvén, Calibration procedure for short focal length off-the-shelf CCD cameras, *Proc. International Conference on Pattern Recognition*, 1996, pp. 166–170.

Juho Kannala
Janne Heikkilä
Sami S. Brandt
University of Oulu
Oulu, Finland

L

LEVEL SET METHODS

The shape of a real-world object can be represented by a parametric or a nonparametric contour in the image plane. The parametric contour representation is referred to as an explicit representation and is defined in the Lagrangian coordinate system. In this coordinate system, two different objects have two different representations stemming from the different sets of control points that define the object contours. The control points constitute the finite elements, which is a common formalism used to represent shapes in the Lagrangian coordinates. In contrast to the parametric Lagrangian representation, the nonparametric representation defines the object contour implicitly in the Eulerian coordinates, which remains constant for two different objects. The level set method is a nonparametric representation defined in the Eulerian coordinate system and is used commonly in the computer vision community to represent the shape of an object in an image.

The level set method has been introduced in the field of fluid dynamics by the seminal work of Osher and Sethian in 1988 (1). After its introduction, it has been applied successfully in the fields of fluid mechanics, computational physics, computer graphics, and computer vision. In the level set representation, the value of each grid point (pixel) is set traditionally to the Euclidean distance between the grid point and the contour. Hence, moving the contour from one configuration to the other is achieved by changing the distance values in the grid points. During its motion, the contour can change implicitly its topology by splitting into two disjoint contours or by merging from two contours to one.

The implicit nature of the representation becomes essential to handle the topology changes for the case when an initial configuration is required to solve a time-dependent problem. Upon initialization, the level set converges to a solution by re-evaluating the values at the grid points iteratively. This iterative procedure is referred to as the "contour evolution."

CONTOUR EVOLUTION

Without loss of generality, I will discuss the level set formalism in the two-dimensional image coordinates (x, y), which can be extended to higher dimensions without complicating the formulation. Let there be a closed contour Γ defined in the image coordinates as shown in Fig. 1(a). The contour can be visualized as the boundary of an object in an image. To represent the evolution of the contour, the level set formalism introduces two additional dimensions that define the surface in z and the time t: $[0,T)$:

$$z = \Phi(x, y, t)$$

The surface dimension, $z \in R$, encodes the signed Euclidean distance from the contour. More specifically, the value of z inside the closed contour has a negative sign, and outside the contour has a positive sign [see Fig. 1(b)]. At any given time instant t, the cross section of the surface at $z = 0$ corresponds to the contour, which is also referred to as the zero-level set.

The time dimension t is introduced as an artificial dimension to account for the iterative approach to finding the steady state of an initial configuration. In computer vision, the initial configuration of the level set surface relates to an initial hypothesis of the object boundary, which is evolved iteratively to the correct object boundary. The iterations are governed by a velocity field, which specifies how the contour moves in time. In general, the velocity field is defined by the domain-related physics. For instance, in fluid mechanics, the velocity of the contour is defined by the physical characteristics of the fluid and the environment in which it will dissolve; whereas in computer vision, the velocity is defined by the appearance of the objects in the image.

At each iteration, the equations that govern the contour evolution are derived from the zero-level set at time t: $\Phi(\mathbf{x}(t), t) = 0$, where $\mathbf{x} = (x, y)$. Because the moving contour is always at the zero level, the rate of contour motion in time is given by:

$$\frac{\partial \Phi(x(t), t)}{\partial t} = 0 \tag{1}$$

By applying the chain rule, Equation (1) becomes $\Phi_t + \Delta\Phi(\mathbf{x}(t), t)\mathbf{x}'(t) = 0$. In Fig. 2, a contouris present that evolves using the curvature flow, which moves rapidly in the areas where the curvature is high.

An important aspect of the level set method is its ability to compute the geometric properties of the contour Γ from the level set grid by implicit differentiation. For instance, the contour normal is computed by $\vec{n} = -\frac{\Delta\Phi}{|\Delta\Phi|}$. Hence, dividing both sides by $|\Delta\Phi|$ and replacing $\mathbf{x}(t)\vec{n}$ by F results in the well-known level set evolution equation:

$$\Phi_t + |\Delta\Phi| F = 0 \tag{2}$$

which evolves the contour in the normal direction with speed F. The sign of F defines whether the contour will move inward or outward. Particularly, a negative F value shrinks the contour and a positive F value expands the contour. In computer vision, F is computed at each grid point based on its appearance and the priors defined from the inside and the outside of the contour. As Equation (2) includes only first-order derivatives, it can be written as a Hamilton–Jacobi equation $\Phi_t + H(\phi(x), \phi(y)) = 0$, where $H(\phi(x), \phi(y)) = |\Delta\Phi| F$. Based on this observation, the numerical approximations of the implicit derivatives can be computed by using the forward Euler time-discretization (see Ref. 2 for more details).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

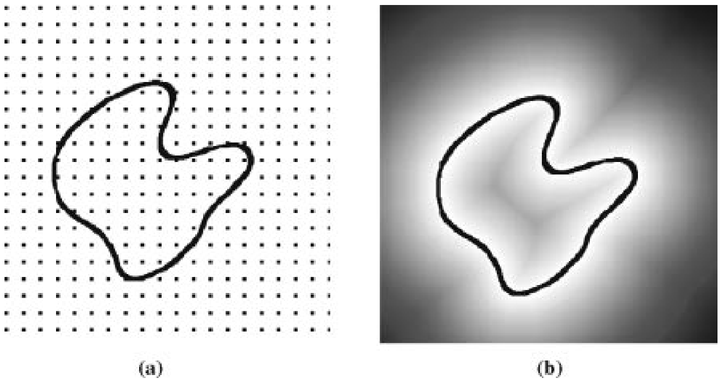


Figure 1. (a) The contour is denned in the spatial image coordinates, *(x,y)*. Each black dot represents a grid point on the plane, (b) The level set function denning the contour given in (a). The blue-colored grid points denote positive values, and the red-colored grid points denote negative values. The distance of a grid point from the contour is expressed by the variation in the color value from light to dark.

REINITIALIZATION OF THE LEVEL SET FUNCTION

To eliminate numeric instabilities during the contour evaluation, it is necessary to reinitialize the level set function at each evolution iteration. This requirement, however, is a major limitation of the level set framework when fast convergence is required, such as in video-based surveillance. An intuitive approach to reinitialize the level set grid is to follow a two-step approach. The first step is to recover the contour by detecting the zero crossings. The second step is to regenerate the level set surface to preserve the geometric properties of the new contour. In computer vision, the second step is to apply the Euclidean distance transform to find the distance of each grid point from the recovered contour.

Application of the distance transform to update the level set is a time-consuming operation, especially considering that it needs to be done at every iteration. One way to reduce the complexity of the level set update is to apply the "narrow band" approach (3). The narrow band approach performs reinitialization only in a neighborhood denned by a band. This band also defines the limits in which the distance transform is applied. The procedure involves recovering the contour, positioning the band-limits from the extracted contour, reinitializing the values residing in the band, and updating the level set bounded by the band. The narrow band approach still must recover the contour and reinitializate the level set function before a solution is reached; hence, the error during the evolution may accumulate from one iteration to the next.

An alternate approach to level set reinitialization is to use an additional partial differential equation (PDE) evaluated on the level set function. In this approach, after the PDE, which operates on the level set, moves the contour, a second PDE re-evaluates the grid values while preserving the zero crossings and geometric properties of the contour (4). The level set reinitialization PDE is given by:

$$\phi_t = \text{sign}(\phi_0)(1 - |\Delta\phi|)$$

where the sign function defines the update of all the levels except for the zero-level. An example sign function is

$$\text{sign}_{\varepsilon}(\phi_0) = \frac{\phi_0}{\sqrt{\phi_0^2 + \varepsilon^2}}$$

FAST MARCHING LEVEL SET

Constraining the contour to to either shrink or to expand eliminates the limitations posed by the level set reinitialization. In this case, the sign of *F* in Equation (2) is constant. Practically, the constant sign guarantees a single visit to each level set grid point. This property results in a very useful formulation referred to as the "fast marching level set."

Compared with the aforementioned approaches, the fast marching method moves the contour one pixel at a time. Hence the distance traveled by the contour becomes constant, $d = 1$. The traveled distance also is attributed to the speed of the contour, $F(x, y)$, as well as the time it takes to travel it, $T(x, y)$. Constrained by these observations, the

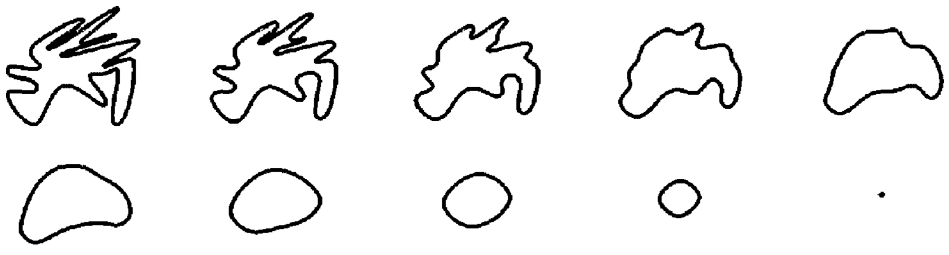


Figure 2. A contour moving with the curvature flow in the level set framework. Note the fast evolution in the regions where the contour bends more than the other regions.

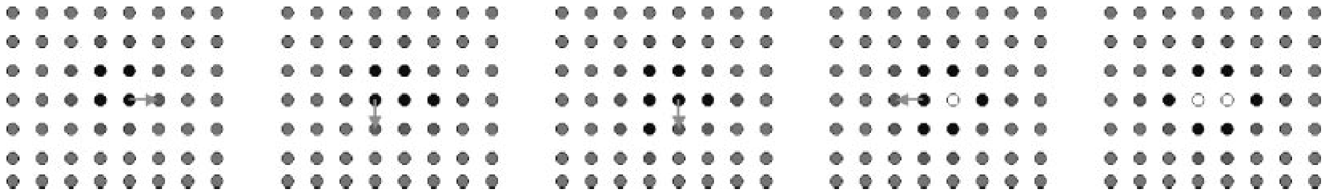


Figure 3. The fast marching level set iterations. The black nodes denote the active contour pixels, the red nodes denote the alive level set grid points, the blue nodes are the nodes that are not considered during the iteration (faraway), and the white nodes denote already visited nodes.

level set becomes a stationary formulation given by $1 = F|\Delta T|$, such that at time t the contour is denned by $\{\Gamma | T(x,y) = t\}$. Based on the stationary formation, the time required to move the contour is computed by $T(x,y) = \frac{1}{F(x,y)}$.

Comparatively, the implementation of the fast matching method is easier than its alternatives. Given an initial contour, the pixels on the contour are labeled as *active*, the pixels within the reach of the contour in the next iteration are labeled as *alive* (pixels with distance of ≤ 1), and the pixels that the contour cannot reach are labeled as *faraway*. The iterations start by computing the time required $T(x, y)$ to travel to each alive pixel. From among alive pixels, the pixel with the lowest arrival time changes its status to active (new contour position). This change introduces new set of alive points. This process is iteratively performed until $F = 0$ for all live pixels. See Fig. 3 for a visualization of this iterative process.

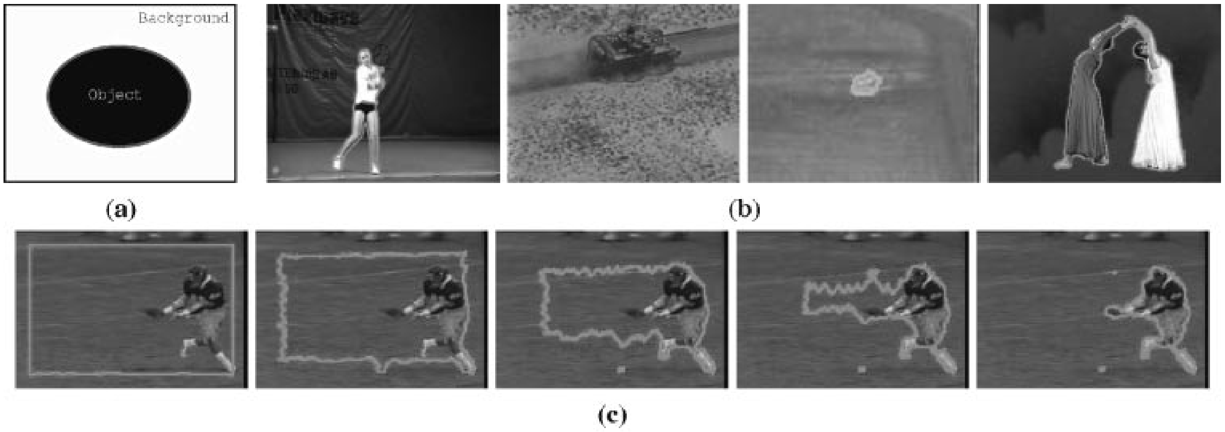
AN EXAMPLE: OBJECT SEGMENTATION

The level set method has become a very successful tool to solve problems ranging from noise removal, image segmentation, registration, object tracking, and stereo. A common property in all these domains is to formulate the problem by a partial differential equation that is solved using the level set formalism.

In the context of level sets, the cost or gain function is required to be in the form of a Hamilton–Jacobi equation, $\phi_t + H(\phi_x, \phi_y) = 0$ that allows only the first-order differentiation of (ϕ). Hence, given a task, the first goal is to come up with a cost function that can be converted to a Hamiltonian $H(\phi_x, \phi_y)$. Let us consider the object segmentation problem as an example. In the following discussion, a region-based object segmentation approach will be considered that is formulated in terms of the appearance similarity inside and outside of the contour (see Fig. 4). Practically, the appearance of the object inside and outside the contour should be different, hence, to minimize this similarity would result in the segmentation of the object. Let us assume the contour is initialized outside the object (inside initialization is also possible), such that the appearance of the region outside the contour serves as a prior. Using this prior, the likelihood of object boundary can be denned in terms of the probability of inside pixels given the outside prior: $p(I(R_{\text{inside}})|R_{\text{outside}})$. Maximizing the gain function formulated using this likelihood measure segments the object as follows:

$$E(\Gamma) = -\int_{x \in R_{\text{inside}}} \log p(I(\mathbf{x}))|R_{\text{outside}})d\mathbf{x} \qquad (3)$$

Equation (3), however, is not in the form of a Hamiltonian. Application of Green's theorem converts this gain function



(a)

(b)

(c)

Figure 4. (a) The inside and outside regions denned by the object contour, (b) Some segmentation results, (c) The zero level set during the segmentation iterations based on the appearance similarity between the inside and outside regions.

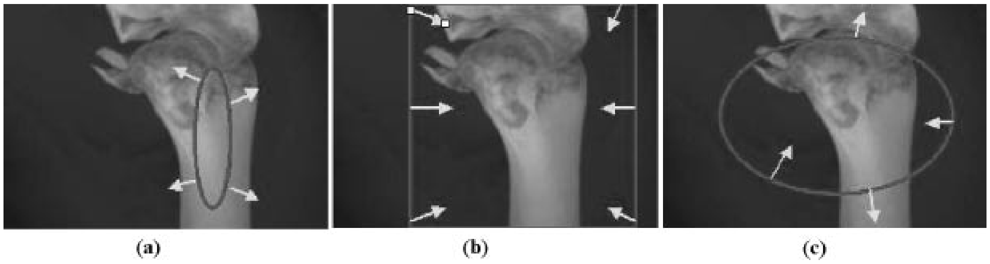


Figure 5. Initialization of the contour, (a) The contour is completely inside the object, (b) outside the object, (c) includes both inside and outside regions of the object.

to a Hamiltonian and results in the level set propagation speed given by $F(x,y) = -\log p(I(x,y)|R_{\text{outside}})$ (see Ref. (5), Chapter 5] for application of Green's theorem). In Fig. 5, several segmentation results are shown for different domains.

DISCUSSION

Because of its robustness, its efficiency, and its applicability to a wide range of problems, the level set method has become very popular among many researchers in different fields. The Hamilton–Jacobi formulation of the level set can be extended easily to higher dimensions. It can model any topology including sharp corners, and can handle the changes to that topology during its evolution. The level set method has overcome the contour initialization problems associated with the classic active contour approach to segment images, such that the initial contour can include part of the object and part of the background simultaneously (see Fig. 5 for possible contour initialization). These properties, however, come at the price of requiring careful thinking to formulate the problem to be solved using the level set framework. Especially, converting a computer vision problem to a PDE may require considerable attention.

On the down side, because of the reinitialization of the level set after each iteration, the level set method becomes a computationally expensive minimization technique. The narrow band approach is proposed to overcome this limitation by bounding the region for reinitialization to a band around the contour. Despite reduced complexity, the narrow band method remains unsuitable for tasks that require realtime processing, such as object tracking in a surveillance scenario. The other possible solution to the complexity problem is the fast marching approach, which works in real time when implemented carefully. However, the fast marching method evolves the contour only in one direction. Hence, the initial position of the contour becomes very important. For instance, the contour has to be either outside or inside the object of interest, and the initializations that involve both inside and outside of the object may not converge.

BIBLIOGRAPHY

1. S. Osher and J. Sethian, Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulation, *Computat. Phys.*, **79**: 12–49, 1988.
2. J. Sethian, *Level Set methods: Evolving Interfaces in Geometry, Fluid mechanics Computer Vision and Material Sciences*, Cambridge, UK: Cambridge University Press, 1996.
3. D. Chop, Computing minimal surfaces via level set curvature flow, *J. Computat. Phys.*, **106**: 77–91, 1993.
4. I. Sussman, P. Smereka, and S. Osher, A level set approach for computing solutions to incompressible two-phase flow, *J. Computat. Phys.*,**114**: 146–159, 1994.
5. A. Yilmaz, Object Tracking and Activity Recognition in Video Acquired Using Mobile Cameras. PhD Thesis, Orlando, FL: University of Central Florida, 2004.

ALPER YILMAZ
The Ohio State University
Columbus, Ohio

MEDICAL IMAGE PROCESSING

INTRODUCTION

The discovery of *x rays* in 1895 by Professor Wilhelm Conrad Röntgen led to a transformation of medicine and science. In medicine, x rays provided a noninvasive way to visualize the internal parts of the body. A beam of radiation passing through the body is absorbed and scattered by tissue and bone structures in the path of the beam to varying extents depending on their composition and the energy level of the beam. The resulting absorption and scatter patterns are captured by a film that is exposed during imaging to produce an image of the tissues and bone structures. By using varying amounts of energy levels of different sources of radiant energy, radiographic images can be produced for different tissues, organs and bone structures.

The simple planar x-ray imaging, the main radiologic imaging method used for most of the last century, produced high-quality analog two-dimensional (2-D) projected images of three-dimensional (3-D) organs. Over the last few decades, increasingly sophisticated methods of diagnosis have been made possible by using different types of radiant energy, including X rays, gamma rays, radio waves, and ultrasound waves. The introduction of the first *x-ray computed tomography* (x-ray CT) scanner in the early 1970s totally changed the medical imaging landscape. The CT scanner uses instrumentation and computer technology for image reconstruction to produce images of cross sections of the human body. With the clinical experience accumulated over the years and the establishment of its usefulness, the CT scanner became very popular.

The exceptional multidimensional digital images of internal anatomy produced by medical imaging technology can be processed and manipulated using a computer to visualize subtle or hidden features that are not easily visible. *Medical image analysis and processing* algorithms for enhancing the features of interest for easy analysis and quantification are rapidly expanding the role of medical imaging beyond noninvasive examination to a tool for aiding surgical planning and intraoperative navigation. Extracting information about the shape details of anatomical structures, for example, enables careful preoperative planning of surgical procedures.

In medical image analysis, the goal is to accurately and efficiently extract information from medical images to support a range of medical investigations and clinical activities from diagnosis to surgery. The extraction of information about anatomical structures from medical images is fairly complex. This information has led to many algorithms that have been specifically proposed for biomedical applications, such as the quantification of tissue volumes (1), diagnosis (2), localization of pathology (3), study of anatomical structures (4), treatment planning (5), partial volume correction of functional imaging data (6), and computer-integrated surgery (7,8).

Technological advances in medical imaging modalities have provided doctors with significant capabilities for accurate noninvasive examination. In modern medical imaging systems, the ability to effectively process and analyze medical images to accurately extract, quantify, and interpret information to achieve an understanding of the internal structures being imaged is critical in order to support a spectrum of medical activities from diagnosis, to radiotherapy, to surgery. Advances in computer technology and microelectronics have made available significant computational power in small desktop computers. This capability has spurred the development of software-based biomedical image analysis methods such as *image enhancement and restoration, segmentation of internal structures and features of interest, image classification,* and *quantification.*

In the next section, different medical imaging modalities are discussed before the most common methods for medical image processing and analysis are presented. An exhaustive survey of such methods can be found in Refs. 9 and 10.

ACQUISITION OF MEDICAL IMAGES

A biomedical image analysis system comprises three major elements: an image acquisition system, a computer for processing the acquired information, and a display system for visualizing the processed images. In medical image acquisition, the primary objective is to capture and record information about the physical and possibly functional properties of organs or tissues by using either external or internal energy sources or a combination of these energy sources.

Conventional Radiography

In *conventional radiography,* a beam of x rays from an external source passing through the body is differentially absorbed and scattered by structures. The amount of absorption depends on the composition of these structures and on the energy of the x ray beam. Conventional imaging methods, which are still the most commonly used diagnostic imaging procedure, form a projection image on standard radiographic film. With the advent of digital imaging technology, radiographs, which are x-ray projections, are increasingly being viewed, stored, transported, and manipulated digitally. Figure 1 shows a normal chest x-ray image.

Computed Tomography

The realization that x-ray images taken at different angles contain sufficient information for uniquely determining the internal structures, led to the development of x-ray CT scanners in the 1970s that essentially reconstruct accurate cross-sectional images from x-ray radiographs. The

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

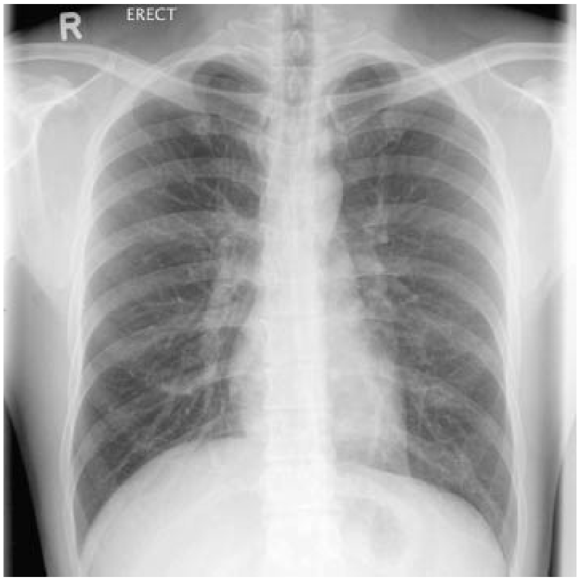
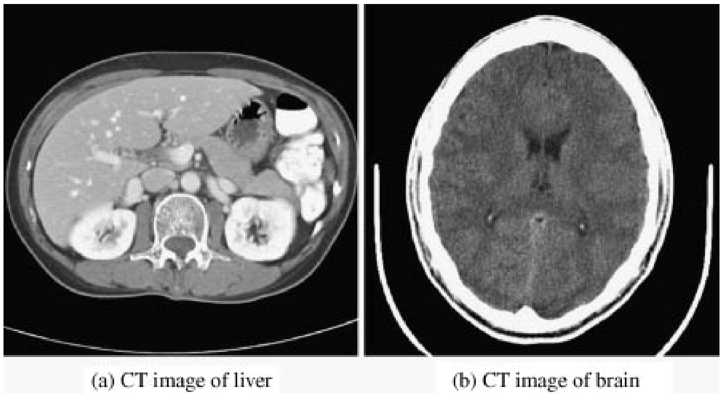


Figure 1. X-ray image of a chest.


(a) CT image of liver

(b) CT image of brain

Figure 2. CT images of normal liver and brain.

conventional x-ray CT consists of a rotating frame that has an x-ray source at one end and an array of detectors that accurately measure the total attenuation along the path of the x ray at the other end. A fan beam of x rays is created as the rotating frame spins the x-ray tube and detectors around the patient. During the 360° rotation, the detector captures numerous snapshots of the attenuated x-ray beam corresponding to a single slice of tissue whose thickness is determined by the collimation of the x-ray beam. This information is then processed by a computer to generate a 2-D image of the slice. Multiple slices are obtained by moving the patient in incremental steps. In the more recent spiral CT (also known as the helical CT), projection acquisition is carried out in a spiral trajectory as the patient continuously moves through the scanner. This process results in faster scans and higher definition of internal structures, which enables greater visualization of blood vessels and internal tissues. CT images of a normal liver and brain are shown in Fig. 2. In comparison with conventional x-ray imaging, CT imaging is a major breakthrough. It can image the structures with subtle differences in x-ray absorption capacity even when almost obscured by a structure with a strong ability on x-radiation absorption. For example, the CT can image the internal structures of the brain, which is enclosed by the skull [as shown in Fig. 2(b)], whereas the x ray fails to do so.

Magnetic Resonance Imaging

Medical imaging methods using magnetic resonance include MRI, PET, and SPECT. MRI is based on the principles of nuclear magnetic resonance (NMR), which is a spectroscopic technique used to obtain microscopic chemical and physical information about molecules. An MRI can produce high-quality multidimensional images of the inside of the human body, as shown in Fig. 3, providing both structural and physiologic information of internal organs and tissues. Unlike the CT, which depicts the x-ray opacity of the structure being imaged, MRIs depict the density as well as biochemical properties based on physiologic function, including blood flow and oxygenation. A major advantage of the MRI is the fast signal acquisition with a very high spatial resolution.

Radiographic imaging modalities such as those based on x rays provide anatomical information about the body but not the functional or metabolic information about an organ or tissue. In addition to anatomical information, MRI methods are capable of providing some functional and metabolic information. *Nuclear medicine*-based imaging systems image the distribution of radioisotopes distributed within specific organs of interest by injection or inhalation of radio-pharmaceuticals that metabolize the tissue, which makes them a source of radiation. The images acquired by these systems provide a direct representation of the metabolism or function of the tissue or organ being imaged as it becomes a source of radiation that is used in the imaging

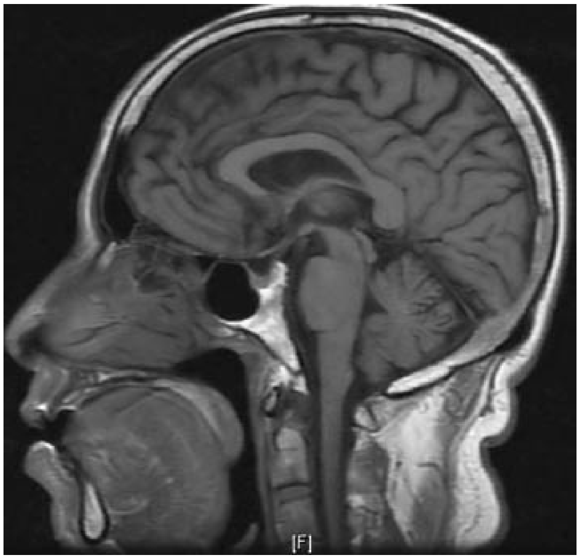
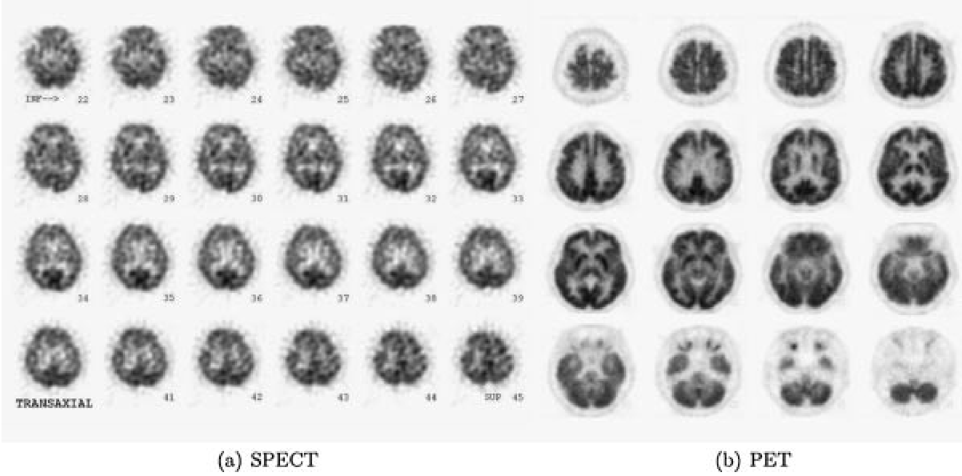


Figure 3. MRI of the brain.



(a) SPECT

(b) PET

Figure 4. SPECT and PET sequences of a normal brain.

process. SPECT and PET are nuclear medicine-based imaging systems.

SPECT systems use gamma cameras to image photons that are emitted during decay. Like the x-ray CT, many SPECT systems rotate a gamma camera around the object being imaged and process the acquired projections using a tomographic reconstruction algorithm to yield a 3-D reconstruction. SPECT systems do not provide good resolution images of anatomical structures like CT or MR images, but they show distribution of radioactivity in the tissue, which represents a specific metabolism or blood flow as shown in Fig. 4(a). PET systems, like SPECT, also produce images of the body by detecting the emitted radiation, but the radioactive substances used in PET scans are increasingly used with CT or MRI scans so as to provide both anatomical and metabolic information. Some slices of a PET brain image are shown in Fig. 4(b).

Ultrasound Imaging

Ultrasound or *acoustic imaging* is an external source-based imaging method. Ultrasound imaging produces images of organs and tissues by using the absorption and reflection of ultrasound waves traveling through the body (Fig. 5). It has been successfully used for imaging of anatomical structures, blood flow measurements, and tissue characterization. A major advantage of this method, which does not involve electromagnetic radiation, is that it is almost nonintrusive, and hence, the examined structures can be subjected to uninterrupted and long-term observations while the subject does not suffer any ill effects.

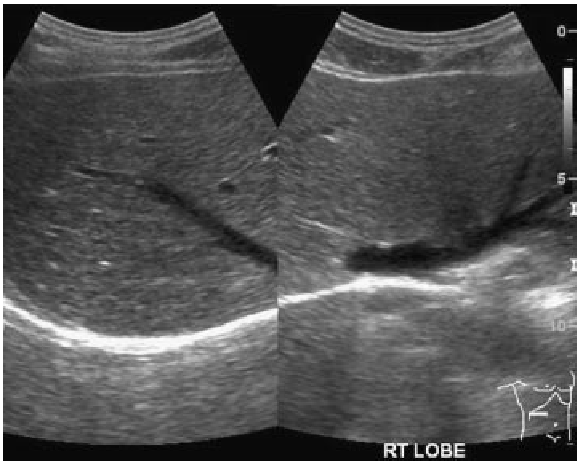


Figure 5. Ultrasound image of a normal liver.

IMAGE ENHANCEMENT AND RESTORATION

In image enhancement, the purpose is to process an acquired image to improve the contrast and visibility of the features of interest. The contrast and visibility of images actually depend on the imaging modality and on the nature of the anatomical regions. Therefore, the type of image enhancement to be applied has to be suitably chosen. *Image restoration* also leads to image enhancement, and generally, it involves mean-squared error operations and other methods that are based on an understanding of the type of degradation inherent in the image. However, procedures that reduce noise tend to reduce the details, whereas those that enhance details also increase the noise. Image enhancement methods can be broadly classified into two categories: *spatial-domain methods*(11) and *frequency-domain methods*(12).

Spatial-domain methods involve manipulation on a pixel-to-pixel basis and include histogram-based methods, spatial filtering, and so on. The *histogram* provides information about the distribution of pixel intensities in an image and is normally expressed as a 2-D graph that provides information of the occurrence of specific gray-level pixel values in the image. *Histogram equalization* is a commonly used technique that involves "spreading out" or "stretching" the gray levels to ensure that the gray levels are redistributed as evenly as possible (12,13). Minor variations in structures or features are better visualized within regions that originally looked uniform by this operation. Figure 6 shows the result of applying histogram equalization to a CT image.

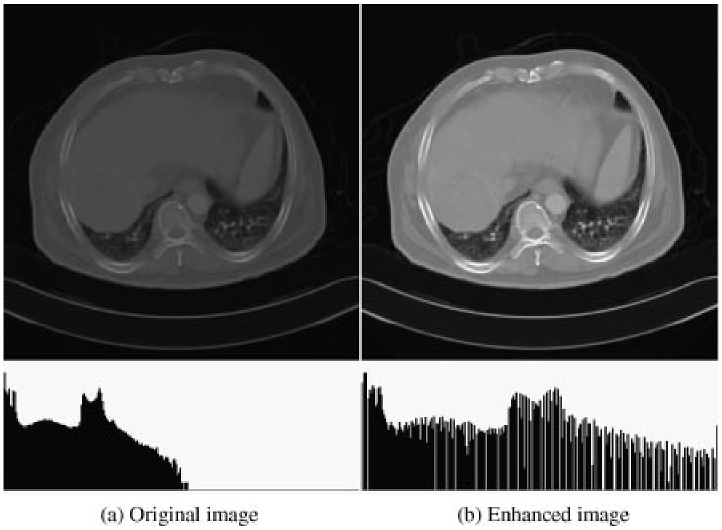


Figure 6. CT images of the liver and the associated histograms: (a) original image and (b) enhanced image.

In some instances, however, global equalization across all possible pixel values could result in loss of important details and/or high-frequency information. For such situations, local histogram modifications, including localized or regional histogram equalization, can be applied to obtain good results (14). In spatial-domain methods, pixel values in an image are replaced with some function of the pixel and its neighbors. *Image averaging* is a form of spatial filtering where each pixel value is replaced by the mean of its neighboring pixels and the pixel itself. Edges in an image can be enhanced easily by subtracting the pixel value with the mean of its neighborhood. However, this approach usually increases the noise in the image. Figure 7 shows the results of applying local spatial-domain methods.

Frequency-domain methods are usually faster and simpler to implement than spatial domain methods (12). The processing is carried out in the Fourier domain for removing or reducing image noise, enhancing edge details, and improving contrast. A *low-pass filter* can be used to suppress noise by removing high frequencies in the image, whereas a *high-pass filter* can be used to enhance the high frequencies resulting in an increase of detail and noise. Different filters can be designed to selectively enhance or suppress features or details of interest.

MEDICAL IMAGE SEGMENTATION

A fundamental operation in medical image analysis is the *segmentation* of anatomical structures. It is not surprising that segmentation of medical images has been an important research topic for a long time. It essentially involves partitioning an image into distinct regions by grouping together neighboring pixels that are related. The extraction or segmentation of structures from medical images and reconstructing a compact geometric representation of these structures is fairly complex because of the complexity and variability of the anatomical shapes of interest. Inherent shortcomings of the acquired medical images, such as sampling artifacts, spatial aliasing, partial volume effects, noise, and motion, may cause the boundaries of structures to be not clearly distinct. Furthermore, each imaging modality with its own characteristics could produce images that are quite different when imaging the same structures. Thus, it is challenging to accurately extract the boundaries of the same anatomical structures. Traditional image processing methods are not easily applied for analyzing medical images unless supplemented with considerable amounts of expert intervention (15).

There has been a significant body of work on algorithms for the segmentation of anatomical structures and other

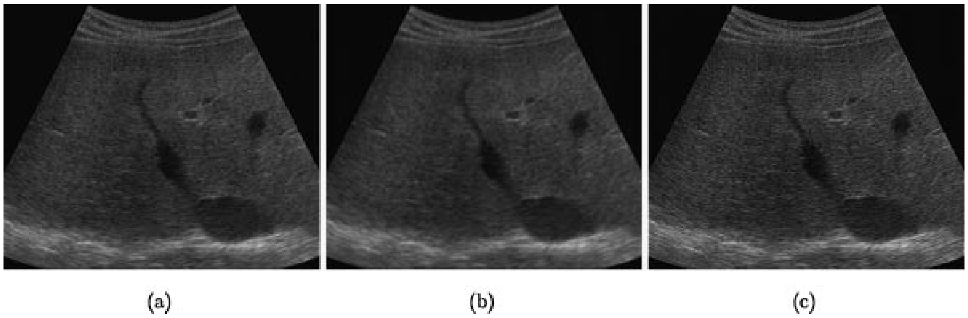
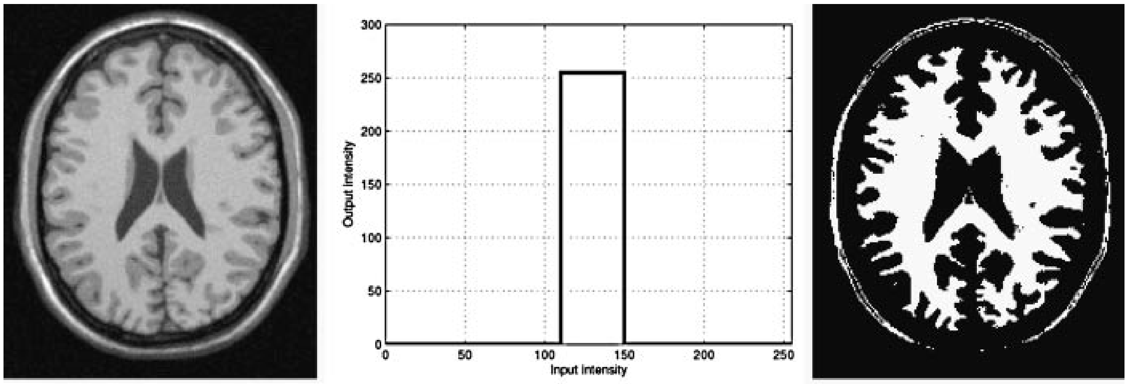


Figure 7. Ultrasound images of liver. (a) Original image. (b) The result of image averaging where the image becomes smoother. (c) The image of edge enhancement where the image becomes sharper but with increased noise levels.



(a) Input image (b) Thresholding function (c) Output image

Figure 8. Binary thresholding of MR image of the brain.

regions of interest that aims to assist and automate specific radiologic tasks (16). They vary depending on the specific application, imaging modality, and other factors. Currently, no segmentation method yields acceptable results for different types of medical images. Although general methods can be applied to a variety of data (10,15,17), they are specific for particular applications and can often achieve better performance by taking into account the specific nature of the image modalities. In the following subsections, the most commonly used segmentation methods are briefly introduced.

Thresholding

Thresholding is a very simple approach for segmentation that attempts to partition images by grouping pixels that have similar intensities or range of intensities into one class and the remaining pixels into another class. It is often effective for segmenting images with structures that have contrasting intensities (18,19). A simple approach for thresholding involves analyzing the histogram and setting the threshold value to a point between two major peaks in the distribution of the histogram. Although there are automated methods for segmentation, thresholding is usually performed interactively based on visual assessment of the resulting segmentation (20). Figure 8 shows the result of a thresholding operation on an MR image of brain where only pixels within a certain range of intensities are displayed.

The main limitations of thresholding are that only two classes are generated, and that it typically does not take into account the spatial characteristics of an image, which causes it to be sensitive to noise and intensity inhomogeneities which tend to occur in many medical imaging modalities. Nevertheless, thresholding is often used as an initial step in a sequence of image-processing operations.

Region-Based Segmentation

Region growing algorithms (21) have proven to be an effective approach for image segmentation. The basic approach in these algorithms is to start from a seed point or region that is considered to be inside the object to be segmented. Neighboring pixels with similar properties are evaluated to determine whether they should also be considered as being part of the object, and those pixels that should be are added to the region. The process continues as long as new pixels are added to the region. Region growing algorithms differ in that they use different criteria to decide whether a pixel should be included in the region, the strategy used to select neighboring pixels to evaluate, and the stopping criteria that stop the growing. Figure 9 illustrates several examples of region growing-based segmentation. Like thresholding-based segmentation, region growing is seldom used alone but usually as part of set of image-processing operations, particularly for segmenting small and simple structures such as tumors and lesions (23,24). Disadvantages of region growing methods include the need for manual intervention to specify the initial seed point and its sensitivity to noise, which causes extracted regions to have holes or to even become disconnected.

Region splitting and merging algorithms (25,26) evaluate the homogeneity of a region based on different criteria such as the mean, variance, and so on. If a region of interest is found to be inhomogeneous according to some similarity constraint, it is split into two or more regions. Since some neighboring regions may have identical or similar properties after splitting, a merging operation is incorporated that compares neighboring regions and merges them if necessary.

Segmentation Through Clustering

Segmentation of images can be achieved through clustering pixel data values or feature vectors whose elements consist of the parameters to be segmented. Examples of *multidimensional feature vectors* include the red, green, and blue (RGB) components of each image pixel and different attenuation values for the same pixel in dual-energy x-ray images. Such datasets are very useful as each dimension of data allows for different distinctions to be made about each pixel in the image.

In clustering, the objective is to group similar feature vectors that are close together in the *feature space* into a single cluster, whereas others are placed in different clusters. Clustering is thus a form of *classification*. Sonka and

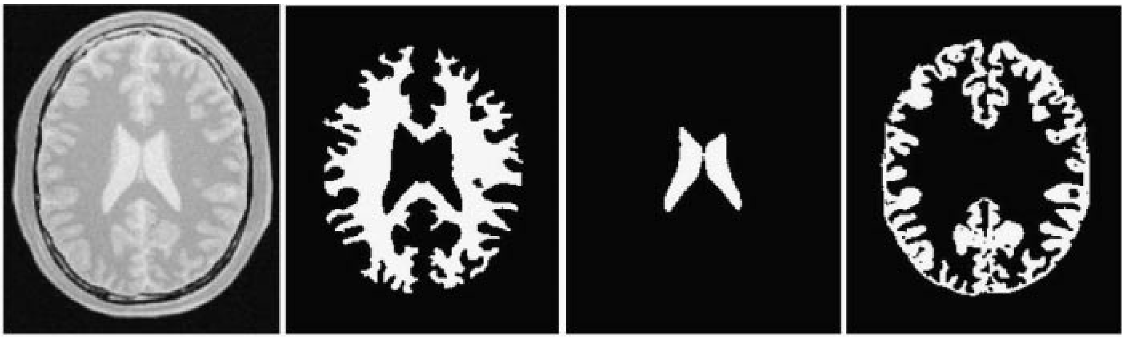


Figure 9. Segmentation results of region growing with various seed points obtained by using Insight Toolkit (22).

Fitzpatrick (27) provide a thorough review of classification methods, many of which have been applied in object recognition, registration, segmentation, and feature extraction. Classification algorithms are usually categorized as *unsupervised* and *supervised.* In supervised methods, sample feature vectors exist for each class (i.e., a priori knowledge) and the classifier merely decides on how to classify new data based on these samples. In unsupervised methods, there is no a priori knowledge and the algorithms, which are based on clustering analysis, examine the data to determine natural groupings or classes.

Unsupervised Clustering. Unlike supervised classification, very few inputs are needed for unsupervised classification as the data are clustered into groupings without any user-defined training. In most approaches, an initial set of grouping or classes is defined. However, the initial set could be inaccurate and possibly split along two or more actual classes. Thus, additional processing is required to correctly label these classes.

The *k-means clustering algorithm*(28,29) partitions the data into *k* clusters by optimizing an objective function of feature vectors of clusters in terms of similarity and distance measures. The objective function used is usually the sum of squared error based on the Euclidean distance measure. In general, an initial set of *k* clusters at arbitrary centroids is first created by the k-means algorithm. The centroids are then modified using the objective function resulting in new clusters. The k-means clustering algorithms have been applied in medical imaging for segmentation/classification problems (30,31). However, their performance is limited when compared with that achieved using more advanced methods.

While the k-means clustering algorithm uses fixed values that relate a data point to a cluster, the *fuzzy k-means* clustering algorithm (32) (also known as the *fuzzy c-means)* uses a membership value that can be updated based on distribution of the data. Essentially, the fuzzy k-means method enables any data sample to belong to any cluster with different degrees of membership. Fuzzy partitioning is carried out through an iterative optimization of the objective function, which is also a sum of squared error based on the Euclidean distance measure factored by the *degree of membership* with a cluster. Fuzzy k-means algorithms have been applied successfully in medical image analysis (33–36), most commonly for segmentation of MRI images of the brain. Pham and Prince (33) were the first to use adaptive fuzzy k-means in medical imaging. In Ref. 34, Boudraa et al. segment multiple sclerosis lesions, whereas the algorithm of Ahmed et al. (35) uses a modified objective function of the standard fuzzy k-means algorithm to compensate for inhomogeneities. Current fuzzy k-means methods use adaptive schemes that iteratively vary the number of clusters as the data are processed.

Supervised Clustering. Supervised methods use sample feature vectors (known as *training data*) whose classes are known. New feature vectors are classified into *one of the known classes* on the basis of how similar they are to the known sample vectors. Supervised methods assume that classes in multidimensional feature spaces can be described by *multivariate probability density functions*. The probability or *likelihood* of a data point belonging to a class is related to the distance from the class center in the feature space. *Bayesian classifiers* adopt such probabilistic approaches and have been applied to medical images usually as part of more elaborate approaches (37,38). The accuracy of such methods depends very much on having good estimates for the mean (center) and covariance matrix of each class, which in turn requires large training datasets.

When the training data are limited, it would be better to use *minimum distance* or *nearest neighbor classifiers* that merely assign unknown data to the class of the sample vector that is closest in the feature space, which is measured usually in terms of the Euclidean distance. In the *k-nearest neighbor* method, the class of the unknown data is the class of majority of the k-nearest neighbors of the unknown data. Another similar approach is called the *Parzen windows* classifier, which labels the class of the unknown data as that of the class of the majority of samples in a volume centered about the unknown data that have the same class. The nearest neighbor and Parzen windows methods may seem easier to implement because they do not require a priori knowledge, but their performance is strongly dependent on the number of data samples available. These supervised classification approaches have been used in various medical imaging applications (39,40).

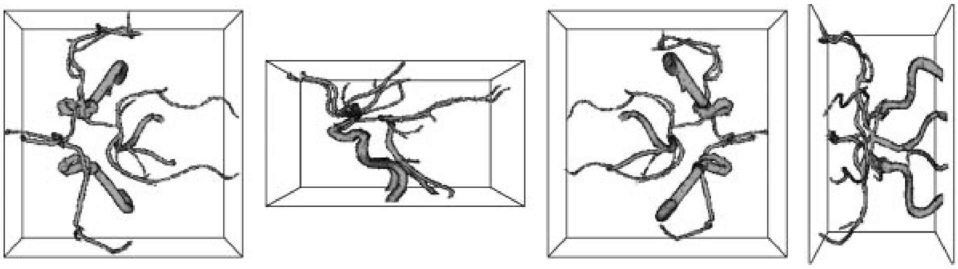


Figure 10. Different views of the MRA segmentation results using the capillary active contour.

Model Fitting Approaches

Model fitting is a segmentation method where attempts are made to fit simple geometric shapes to the locations of extracted features in an image (41). The techniques and models used are usually specific to the structures that need to be segmented to ensure good results. Prior knowledge about the anatomical structure to be segmented enables the construction of shape models. In Ref. 42, *active shape models* are constructed from a set of training images. These models can be fitted to an image by adjusting some parameters and can also be supplemented with textural information (43). Active shape models have been widely used for medical image segmentation (44–46).

Deformable Models

Segmentation techniques that combine *deformable models* with local edge extraction have achieved considerable success in medical image segmentation (10,15). Deformable models are capable of accommodating the often significant variability of biological structures. Furthermore, different regularizers can be easily incorporated into deformable models to get better segmentation results for specific types of images. In comparison with other segmentation methods, deformable models can be considered as “high-level segmentation” methods (15).

Deformable models (15) are referred by different names in the literature. In 2-D segmentation, deformable models are usually referred to as snakes (47,48), active contours (49,50), balloons (51), and deformable contours (52). They are usually referred to as active surfaces (53) and deformable surfaces (54,55) in 3-D segmentation.

Deformable models were first introduced into computer vision by Kass et al. (47) as “snakes” or active contours, and they are now well known as *parametric deformable models* because of their explicit representation as parameterized contours in a Lagrangian framework. By designing a global shape model, boundary gaps are easily bridged, and overall consistency is more likely to be achieved. Parametric deformable models are commonly used when some prior information of the geometrical shape is available, which can be encoded using, preferably, a small number of parameters. They have been used extensively, but their main drawback is the inability to adapt to topology (15,48). *Geometric deformable models* are represented implicitly as a level set of higher dimensional, scalar-level set functions, and they evolve in an Eulerian fashion (56,57). Geometric deformable models were introduced more recently by Caselles et al. (58) and by Malladi et al. (59). A major advantage of these models over parametric deformable models is topological flexibility because of their implicit representation. During the past decade, tremendous efforts have been made on various medical image segmentation applications based on level set methods (60). Many new algorithms have been reported to improve the precision and robustness of level set methods. For example, Chan and Vese (61) proposed an active contour model that can detect objects whose boundaries are not necessarily defined by gray-level gradients.

When applying for segmentation, an initialization of the deformable model is needed. It can be manually selected or generated by using other low-level methods such as thresholding or region growing. An energy functional is designed so that the model lies on the object boundary when it is minimized. Yan and Kassim (62,63) proposed the *capillary active contour* for *magnetic resonance angiography* (MRA) image segmentation. Inspired by capillary action, a novel energy functional is formulated, which is minimized when the active contour snaps to the boundary of blood vessels. Figure 10 shows the segmentation results of MRA using a special geometric deformable model, the capillary active contour.

MEDICAL IMAGE REGISTRATION

Multiple images of the same subject, acquired possibly using different medical imaging modalities, contain useful information usually of complementary nature. Proper integration of the data and tools for visualizing the combined information offers potential benefits to physicians. For this integration to be achieved, there is a need to *spatially align* the separately acquired images, and this process is called *image registration*. Registration involves determining a *transformation* that can relate the position of features in one image with the position of the corresponding features in another image. To determine the transformation, which is also known as *spatial mapping,* registration algorithms use geometrical features such as points, lines, and surfaces that correspond to the same physical entity visible in both images. After accurate registration, different images will have the same coordinate system so that each set of points in one image will occupy the same volume as the corresponding set of points in another image. In addition to combining images of the same subject from different modalities, the other applications of image registration include aligning temporal image sequences to compensate for motion of the subject between scans and image guidance during medical procedures.

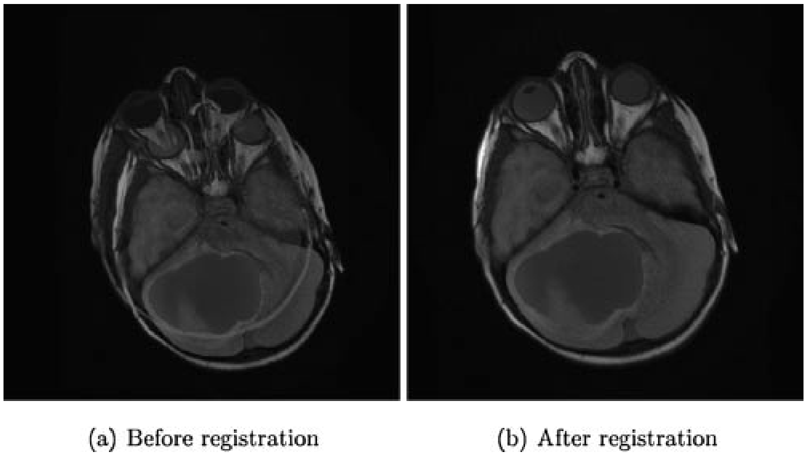


Figure 11. (a) Result of direct overlapping of brain MR images without registration. (b) Result of overlapping of images after rigid registration.

The evaluation of the transformation parameters can be computationally intensive but can be simplified by assuming that the structures of interest do not deform or distort between image acquisitions. However, many organs do deform during image acquisition, for example, during the cardiac and respiratory cycles. Many medical image registration algorithms calculate *rigid body* or *affine* transformations, and thus, their applicability is restricted to parts of the body where deformation is small. As bones are rigid, *rigid body registration* is widely used where the structures of interest are either bone or are enclosed in bone. The brain, which is enclosed by the skull, is reasonably nondeformable, and several registration approaches have been applied to the rigid body registration of brain images. Figure 11 shows an example of rigid registration of brain MR images. Image registration based on rigid body transformations is widely used for aligning multiple 3-D tomographic images of the same subject acquired using different modalities *(intermodality registration*) or using the same modality *(intramodality registration*) (64–66).

The problem of aligning images of structures with deformed shapes is an active area of research. Such *deformable (nonaffine)* registration algorithms usually involve the use of an initial rigid body or affine transformation to provide a starting estimate (65–68).

After registration, *fusion* is required for the integrated display. Some fusion methods involve direct combination of multiple registered images. One example is the superimposition of PET data on MRI data to provide a single image containing both structure and function (69). Another example involves the use of MR and CT combined to delineate tumor tissues (70).

Atlas-Based Approaches

In atlas-based registration and segmentation of medical images, prior anatomical and/or functional knowledge is exploited. The atlas is actually a reference image in which objects of interest have been carefully segmented. In this method (65,66,71), the objective is essentially to carry out a *nonrigid registration* between the image of the patient and the atlas. The first step, known as atlas warping, involves finding a transformation that maps the atlas image to the target image to be segmented. The warping usually consists of a combination of linear and nonlinear transformations to ensure good registration despite anatomical variability. Even with nonlinear registration methods, accurate atlas-based segmentation of complex structures is difficult because of anatomical variability, but these approaches are generally suited for segmentation of structures that are stable in large numbers of people. Probabilistic atlases 72–75 help to overcome anatomical variability.

CONCLUDING REMARKS

Significant advances made in medical imaging modalities have led to several new methodologies that provide significant capabilities for noninvasive and accurate examination of anatomical, physiological, metabolic, and functional structures and features. three- and four dimensional medical images contain a significant amount of information about the structures being imaged. Sophisticated software-based image processing and analysis methods enhance the information acquired by medical imaging equipment to improve the visibility of features of interest and thereby enable visual examination, diagnosis, and analysis. There are, however, several *challenges* in medical imaging ranging from accurate analysis of cardiac motion and tumors to nonaffine registration applications involving organs other than the brain and so on (76,77). Also, new imaging modalities, such as optical (78), microwave, and electrical impedance (79) imaging methods hold the promise of breakthroughs when new associated algorithms for processing and analyzing the resulting information acquired via these modalities are available.

ACKNOWLEDGMENTS

The authors would like to thank the Department of Nuclear Medicine at the Singapore General Hospital for providing the PET image and Prof. S. C. Wang of the Department of Diagnostic Imaging at the National University Hospital for providing the images used in this chapter. The authors are also grateful to Dr. P. K. Sadasivan for his valuable inputs.

BIBLIOGRAPHY

1. S. M. Lawrie and S. S. Abukmeil, Brain abnormality in schizophrenia: A systematic and quantitative review of volumetric magnetic resonance imaging studies, *Br. J. Psychiat.*, **172**: 110–120, 1998.
2. P. Taylor, Computer aids for decision-making in diagnostic radiology–a literature review, *Br. J. Radiol.* **68** (813): 945–957, 1995.
3. A. P. Zijdenbos and B. M. Dawant, Brain segmentation and white matter lesion detection in MR images, *Crit. Rev. Biomed. Engineer.*, **22** (6): 401–465, 1994.
4. A. Worth, N. Makris, V. Caviness, and D. Kennedy, Neuroanatomical segmentation in MRI: technological objectives, *Internat. J. Patt. Recog. Artificial Intell.*, **11**: 1161–1187, 1997.
5. V. Khoo, D. P. Dearnaley, D. J. Finnigan, A. Padhani, S. F. Tanner, and M. O. Leach, Magnetic resonance imaging (MRI): Considerations and applications in radiotherapy treatment planning, *Radiother. Oncol.*, **42**: 1–15, 1997.
6. H. Muller-Gartner, J. Links, J. Prince, R. Bryan, E. McVeigh, J. P. Leal, C. Davatzikos, and J. Frost, Measurement of tracer concentration in brain gray matter using positron emission tomography: MRI-based correction for partial volume effects, *J. Cerebral Blood Flow Metabol.*, **12** (4): 571–583, 1992.
7. N. Ayache, P. Cinquin, I. Cohen, L. Cohen, F. Leitner, and O. Monga, Segmentation of complex three-dimensional medical objects: A challenge and a requirement for computer-assisted surgery planning and performance, in R. Taylor, S. Lavallee, G. Burdea, and R. Mosges, (eds), *Computer-Integrated Surgery*, Cambridge MA: The MIT Press, 1996, pp. 59–74.
8. W. E. L. Grimson, G. J. Ettinger, T. Kapur, M. E. Leventon, W. M. Wells, and R. Kikinis, Utilizing segmented MRI data in image-guided surgery, *Internat. J. Patt. Recog. Artificial Intell.*, **11** (8): 1367–1397, 1997.
9. D. L. Pham, C. Xu, and J. L. Prince, Current methods in medical image segmentation, *Annu. Rev. Biomed. Eng.*, **2**: 315–337, 2000.
10. J. S. Duncan and N. Ayache, Medical image analysis: Progress over two decades and the challenges ahead, *IEEE Trans. Pattern Anal. and Machine Intell.*, **22** (1): 85–106, 2000.
11. P. Perona and J. Malik, Scale-space and edge detection using anisotropic diffusion, *IEEE Trans. Pattern Anal. and Machine Intell.*, **12**: 629–639, 1990.
12. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd ed. Upper Saddle River: Prentice Hall, 2002.
13. T. Acharya and A. K. Ray, *Image Processing: Principles and Applications*, New York: Wiley-Interscience, 2005.
14. K. Zuiderveld, Contrast limited adaptive histogram equalization. P. Heckbert in (ed.), Graphic Gems IV, New York: Academic Press, 1994.
15. T. McInerney and D. Terzopoulos, Deformable models in medical image analysis: A survey, *Med. Image Anal.*, **1** (2): 91–108, 1996.
16. A. Dhawan, *Medical Image Analysis*. New York: Wiley, 2003.
17. J. Suri, K. Liu, S. Singh, S. Laxminarayana, and L. Reden, Shape recovery algorithms using level sets in 2-D/3-D medical imagery: A state-of-the-art review, *IEEE Trans. Inform. Technol. Biomed.*, **6**: 8–28, 2002.
18. P. Sahoo, S. Soltani, and A. Wong, A survey of thresholding techniques, *Comput. Vision. Graph. Image Process.*, **42** (2): 233–260, 1988.
19. M. Sezgin and B. Sankur, Survey over image thresholding techniques and quantitative performance evaluation, *J. Electron. Imaging*, **13** (1): 146–168, 2004.
20. N. Otsu, A threshold selection method from gray level histograms, *IEEE Trans. Sys., Man Cybernet.*, **9**: pp. 62–66, 1979.
21. R. Adams and L. Bischof, Seeded region growing, *IEEE Trans. Pattern Anal. Mach. Intell.*, **16** (6): 641–647, 1994.
22. L. Ibanez, W. Schroeder, L. Ng, and J. Cates, *The ITK Software Guide*. Kitware Inc., 2003. Available: http://www.itk.org.
23. P. Gibbs, D. Buckley, S. Blackband, and A. Horsman, Tumour volume determination from MR images by morphological segmentation, *Phys. Med. Biol.*, **41**: 2437–2446, 1996.
24. S. Pohlman, K. Powell, N. Obuchowski, W. Chilcote, and S. Broniatowski, Quantitative classification of breast tumors in digitized mammograms, *Med. Phys.*, **23**: 1337–1345, 1996.
25. R. Ohlander, K. Price, and D. Reddy, Picture segmentation using recursive region splitting method, *Comput. Graph. Image Proc.*, **8**: 313–333, 1978.
26. S.-Y. Chen, W.-C. Lin, and C.-T. Chen, Split-and-merge image segmentation based on localized feature analysis and statistical tests, *CVGIP: Graph. Models Image Process.*, **53** (5): 457–475, 1991.
27. M. Sonka and J. M. Fitzpatrick, (eds.), *Handbook of Medical Imaging*, vol. 2, ser. Medical Image Processing and Analysis. Bellingham, WA: SPIE Press, 2000.
28. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ: Prentice-Hall, Inc., 1988.
29. S. Z. Selim and M. Ismail, K-means-type algorithms: A generalized convergence theorem and characterization of local optimality, *IEEE Trans. Pattern Anal. and Machine Intell.*, **6**: 81–87, 1984.
30. M. Singh, P. Patel, D. Khosla, and T. Kim, Segmentation of functional MRI by k-means clustering, *IEEE Trans. Nucl. Sci.*, **43**: 2030–2036, 1996.
31. A. P. Dhawan and L. Arata, Knowledge-based 3-D analysis from 2-D medical images, *IEEE Eng. Med. Biol. Mag.*, **10**: 30–37, 1991.
32. L. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum Press, 1981.
33. D. L. Pham and J. L. Prince, Adaptive fuzzy segmentation of magnetic resonance images, *IEEE Trans. Med. Imag.*, **18**: 193–199, 1999.
34. A. O. Boudraa, S. M. Dehak, Y. M. Zhu, C. Pachai, Y. G. Bao, and J. Grimaud, Automated segmentation of multiple sclerosis lesions in multispectral MR imaging using fuzzy clustering, *Comput. Biol. Med.*, **30**: 23–40, 2000.
35. M. N. Ahmed, S. M. Yamany, N. Mohamed, A. A. Farag, and T. Morianty, A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data, *IEEE Trans. Medical Imaging*, **21**: 193–199, 2002.
36. C. Zhu and T. Jiang, Multicontext fuzzy clustering for separation of brain tissues in magnetic resonance images, *Neuromimage*, **18**: 685–696, 2003.
37. P. Spyridonos, P. Ravazoula, D. Cavouras, K. Berberidis, and G. Nikiforidis, Computer-based grading of haematoxylin-eosin stained tissue sections of urinary bladder carcinomas, *Med. Inform. Internet Med.*, **26**: 179–190, 2001.
38. F. Chabat, G.-Z. Yang, and D. M. Hansell, Obstructive lung diseases: texture classification for differentiation at CT, *Radiology*, **228**: 871–877, 2003.
39. K. Jafari-Khouzani and H. Soltanian-Zadeh, Multiwavelet grading of pathological images of prostate, *IEEE Trans. Biomed. Eng.*, **50**: 697–704, 2003.
40. C. I. Christodoulou, C. S. Pattichis, M. Pantziaris, and A. Nicolaides, Texture-based classification of atherosclerotic arotid plaques, *IEEE Trans. Med. Imag.*, **22**: 902–912, 2003.

41. S. D. Pathak, P. D. Grimm, V. Chalana, and Y. Kim, Pubic arch detection in transrectal ultrasound guided prostate cancer therapy, *IEEE Trans. Med. Imag.*, **17**: 762–771, 1998.
42. T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham, Active shape models – their training and application, *Comput. Vision Image Understand.*, **61** (1): 38–59, 1995.
43. T. F. Cootes, C. Beeston, G. J. Edwards, and C. J. Taylor, A unified framework for atlas matching using active appearance models, *Proc. Int. Conf. on Image Processing in Medical Imaging.*, 1999, pp. 322–333.
44. T. F. Cootes and C. J. Taylor, Statistical models of appearance for medical image analysis and computer vision, *SPIE Med. Imag.*, San Diego, CA: 2001, 236–248.
45. J. Xie, Y. Jiang, and H. T. Tsui, Segmentation of kidney from ultrasound images based on texture and shape priors, *IEEE Trans. Med. Imag.*, **24** (1): 45–57, 2005.
46. P. Yan and A. A. Kassim, Medical image segmentation using minimal path deformable models with implicit shape priors, *IEEE Trans. Inform. Technol. Biomed.*, **10** (4): 677–684, 2006.
47. M. Kass, A. Witkin, and D. Terzopoulos, Snakes: Active contour models, *Int. J. Comp. Vision*, **1** (4): 321–331, 1987.
48. C. Xu and J. L. Prince, Snakes, shapes, and gradient vector flow, *IEEE Trans. Image Process.*, **7**: 359–369, 1998.
49. V. Caselles, R. Kimmel, and G. Sapiro, Geodesic active contours, *Int. J. Comp. Vision*, **22** (1): 61–79, 1997.
50. S. Kichenassamy, A. Kumar, P. J. Olver, A. Tannenbaum, and A. J. Yezzi, Gradient flows and geometric active contour models, in *IEEE Int. Conf. Computer Vision*, Cambridge, MA: 1995, pp. 810–815.
51. L. D. Cohen, On active contour models and balloons, *CVGIP: Image Understand.*, **53** (2): 211–218, 1991.
52. L. H. Staib and J. S. Duncan, Parametrically deformable contour models, *Proc. IEEE Conf. Computer Vision and Pattern Recognition.*, San Diego, CA, 1989, pp. 98–103.
53. J. W. Snell, M. B. Merickel, J. M. Ortega, J. C. Goble, J. R. Brookeman, and N. F. Kassell, Model-based boundary estimation of complex objects using hierarchical active surface templates, *Patt. Recog.*, **28** (10): 1599–1609, 1995.
54. I. Cohen, L. Cohen, and N. Ayache, Using deformable surfaces to segment 3D images and infer differential structures, *CVGIP: Image Understand.*, **56** (2): 242–263, 1992.
55. L. H. Staib and J. S. Duncan, Model-based deformable surface finding for medical images, *IEEE Trans. Med. Imag.*, **15** (5): 720–731, 1996.
56. S. Osher and J. A. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, *J. Computational Physics*, **79**: 12–49, 1988.
57. J. A. Sethian, *Level Set Methods and Fast Marching Methods*, 2nd ed. New York: Cambridge University Press, 1999.
58. V. Caselles, F. Catte, T. Coll, and F. Dibos, A geometric model for active contours, *Numerische Mathematik*, **66**: 1–31, 1993.
59. R. Malladi, J. A. Sethian, and B. C. Vermuri, Shape modeling with front propagation: A level set approach, *IEEE Trans. Pattern Anal. Mach. Intell.*, **17** (2): 158–174, 1995.
60. J. S. Suri, K. Liu, L. Reden, and S. Laxminarayan, A review on MR vascular image processing: skeleton versus nonskeleton approaches: part II, *IEEE Trans. Inform. Technol. Biomed.*, **6** (4): 338–350, 2002.
61. T. F. Chan and L. A. Vese, Active contours without edges, *IEEE Trans. Image Proc.*, **10** (2): 266–277, 2001.
62. P. Yan and A. A. Kassim, MRA image segmentation with capillary active contours, *Proc. Medical Image Computing and Computer-Assisted Intervention*, Palm Springs, CA, 2005, pp. 51–58.
63. P. Yan and A. A. Kassim, MRA image segmentation with capillary active contours, *Med. Image Anal.*, **10** (3): 317–329, 2006.
64. L. G. Brown, A survey of image registration techniques, *ACM Comput. Surveys*, **24** (4): 325–376, 1992.
65. J. B. Antoine Maintz and M. A. Viergever, A survey of medical image registration, *Med. Image Anal.*, **2** (1): 1–36, 1998.
66. D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, Medical image registration, *Phys. Med. Biol.*, **46**: 1–45, 2001.
67. W. M. Wells, W. E. L. Grimson, R. Kikinis, and F. A. Jolesz, Adaptive segmentation of MRI data, *IEEE Trans. Med. Imag.*, **15** (4): 429–442, 1996.
68. F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens, Multimodality image registration by maximization of mutual information, *IEEE Trans. Med. Imag.*, **16** (2): 187–198, 1997.
69. Y. Shimada, K. Uemura, B. A. Ardekani, T. Nagakota, K. Ishiwata, H. Toyama, K. Ono, M. Senda, Application of PET-MRI registration techniques to cat brain imaging, *J. Neurosci. Methods*, **101**: 1–7, 2000.
70. D. L. Hill, D. J. Hawkes, M. J. Gleason, T. C. Cox, A. J. Strang, and W. L. Wong, Accurate frameless registration of MR and CT images of the head: Applications in planning surgery and radiation theraphy, *Radiology*, **191**: 447–454, 1994.
71. B. Zitová and J. Flusser, Image registration methods: a survey, *Image Vis. Comput.*, **21**: 977–1000, 2003.
72. M. R. Kaus, S. K. Warfield, A. Nabavi, P. M. Black, F. A. Jolesz, and R. Kikinis, Automated segmentation of MR images of brain tumors, *Radiology*, **218**: 586–591, 2001.
73. [Online]. Available: http://www.loni.ucla.edu/ICBM/.
74. B. Fischl, D. H. Salat, E. Busa, M. Albert, M. Dieterich, C. Haselgrove, et al., Whole brain segmentation: Automated labeling of neuroanatomical structures in the human brain, *Neuron*, **33** (3): 341–355.
75. M. B. Cuadra, L. Cammoun, T. Butz, O. Cuisenaire, and J.-P. Thiran, Validation of tissue modelization and classification techniques in t1-weighted MR brain images. *IEEE Trans. Med. Imag.*, **24**: 1548–1565, 2005.
76. K. Wong, H. Liu, A. J. Sinusas, and P. Shi, Multiframe nonrigid motion analysis with anisotropic spatial constraints: Applications to cardiac image analysis, *Internat. Conf. Image Proc.,* **1**: 131–134, 2004.
77. A. Mohamed, D. Shen, and C. Davatzikos, Deformable registration of brain tumor images via a statistical model of tumor-induced deformation, *Proc. Medical Image Computing and Computer-Assisted Intervention*, **2**, 263–270, 2005.
78. H. Jiang, Y. Xu, N. Iftimia, L. Baron, and J. Eggert, Three-dimensional optical tomographic imaging of breast in a human subject, *IEEE Trans. Med. Imag.*, **20** (12): 1334–1340, 2001.
79. J. Jossinet, E. Marry, and A. Montalibet, Inverse impedance tomography: Imaging tissue from inside, *IEEE Trans. Med. Imag.*, **21** (6): 560–565, 2002.

Ashraf Kassim
Pingkun Yan
National University of Singapore
Singapore

RADIOMETRIC CAMERA CALIBRATION

Radiometric calibration is a method of capturing digital images and processing the digital image data so that the pixel values represent actual measurements of light. Even without radiometric calibration, it is usually the case that a higher-valued pixel in an image represents a greater light intensity, but one cannot say how much light is represented by each pixel value. If the purposes of digital images are merely to be viewed or analyzed for their structural content, then radiometric calibration is not usually necessary. However, when detailed information about brightness, color, or shading is important, radiometric calibration may be essential.

Radiometry is the measurement of electromagnetic radiation, including light (1). Devices called radiometers are designed specifically to measure radiation in units of energy or power. Digital cameras contain many small light-sensitive devices usually arranged in a rectangular grid. Each of these light-sensitive devices may potentially be used as a radiometer if the output (a pixel value) can be interpreted as a light measurement in units of energy or power. Radiometric camera calibration provides the link between raw pixel values obtained by a camera and calibrated pixel values that measure light energy. Radiometric camera calibration converts ordinary digital images into radiometric images.

TYPES OF CALIBRATION

Two types of radiometric calibration exist, absolute and relative (2). Absolute radiometric calibration means that the image pixels are calibrated with reference to physical units such as watts per steradian per square meter ($W.sr^{-1}.m^{-2}$) or photon counts. For absolute calibration, a known or measurable relationship exists between the pixel values and the corresponding measurements expressed in physical units. Relative radiometric calibration means that the image pixels are calibrated with reference to each other—the ratio of two image pixels is the same as the ratio of the corresponding physical measurements of light. Relative radiometric calibration implies that the image pixels have a fixed but unknown multiplicative relationship to the corresponding measurements expressed in physical units. A set of images has shared relative radiometric calibration if the pixels in different images are calibrated with reference to each other. Relative radiometric calibration can be converted to absolute radiometric calibration by including a target with known radiance in an image, then considering the relationship between the known radiance of the target and the corresponding relative radiometric pixel values.

A related calibration is reflective calibration in which image pixels are calibrated with reference to the reflective properties of objects in the scene such as albedo or color. Relative radiometric calibration can be converted to reflective calibration by including a target with known reflectance properties in an image, then using the pixel values corresponding to the target to calibrate the image.

Radiometric calibration has a wide range of applications in science and technology. Astronomy uses radiometric calibration to measure the brightness of stars, galaxies, and other astronomical objects. Remote sensing (2) uses radiometric imaging of the surface of the earth (such as aerial and satellite imaging) to map surface features (e.g., identification of ground cover, forest, specific crops, etc.). Remote sensing systems may use infrared and/or ultraviolet imaging in addition to visible-light imaging. Machine vision systems use radiometric imaging techniques to measure the reflective and transmissive properties of materials when illuminated with controlled light sources. Shape-from-shading and photometric stereo techniques estimate surface orientation from relative radiometric calibrated images. Image analysis techniques such as subpixel edge detection benefit from relative radiometric calibration. Techniques that combine multiple images, such as image stitching and high dynamic range (HDR) imaging, also benefit from radiometric calibration of the component images, although these particular applications do not strictly require radiometric calibration as will be explained below.

METHODS

Methods for radiometric calibration typically involve up to four steps. First, the camera response is corrected to be linear. Second, the camera's black response is subtracted. Third, corrections are applied for spatial variations in the effective aperture of the lens and the sensor sensitivity. This correction yields a relative radiometric calibration. A final step converts the relative calibration to absolute or reflective calibration by applying an appropriate scaling factor.

The first step is linear correction of the camera response function (CRF). Linear correction is required because electronic and digital cameras typically respond nonlinearly to light. A variety of techniques is available for estimating the nonlinear response curve. The inverted response curve is used to transform the pixel values, which yield linear corrected images.

The second step corrects the camera's black response—the pixel value produced by the camera when no light reaches the sensor. Once linear correction is applied, the black response can be estimated from sample images. Subtracting the black response yields an image in which pixel values of zero represent absolute black.

The third step corrects spatial variations in the camera's response to light. The response of individual sensor elements to light depends on their position in the sensor array. Optical effects in the camera's lens cause a radial variation in the intensity of light reaching the sensor elements—usually, elements further from the optical axis

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

receive less light. In addition, manufacturing tolerances mean that individual sensor elements may have different sensitivity to light. These effects can be corrected by applying an individual scaling factor to each pixel value, which yields a relative radiometric image. A set of such images will have shared relative radiometric calibration if the camera's exposure parameters and behavior are consistent for all the images.

The final step of converting the relative radiometric calibration to absolute or reflective calibration requires a scaling factor. If the camera's response is consistent (i.e., the same amount of light always produces the same pixel value), then the necessary scaling factor can be obtained by analyzing a sample image of a known radiometric standard (for absolute radiometric calibration) or of a known reflectance standard (for reflective calibration). However, if the camera's response changes from image to image (for example, because of an automatic iris in the lens), then a reference target should be included in each image to allow calibration between the images. For shared relative radiometric calibration of a set of images, an unknown constant reference target is sufficient, but for absolute radiometric calibration, a reference target with known radiance is required.

If reflective calibration is required for measurement of albedo or color, then a reference target with known reflectance properties is used instead of a known radiance target. The derived scaling factor automatically adjusts for temporal variations in source-illumination brightness.

The remainder of this article first considers the basis of radiometric calibration—the additive property of light and the formation of digital images. Techniques for radiometric calibration are then explained, focusing on the four steps of estimating the camera response function, measuring the black response, correcting for vignetting and sensitivity, and using calibration targets. Finally, applications of radiometric imaging are briefly discussed.

THE ADDITIVE PROPERTY OF LIGHT

Consider a scene illuminated by multiple light sources, each of which can be turned on or off. Assume that the light sources are independent so that the interaction of light from the different sources does not produce interference effects. Also assume that the scene disperses light by reflection and transmission, but it does not exhibit fluorescence. Under these assumptions, light behaves additively; that is, the scene radiance that results from the combined illumination of multiple light sources is the sum of the radiances that result from the individual light sources operating separately.

The additive property of light is useful for testing the linearity of a camera and for estimating the CRF. It was first used in 1893 by Elster and Giedel to demonstrate that the photoelectric current produced by a photocell is proportional to the irradiance (the quantity of light falling on the photocell), and is still used for testing and calibrating spectrophotometers (3,4) and photometers (5).

HOW DIGITAL IMAGES ARE FORMED

Radiometric calibration techniques are based on mathematical models of digital image formation. Digital image pixel values depend on the source illumination, the reflection of light by the scene, and the conversion of light to digital data by the camera. It is particularly important to mathematically model the camera so that image data can be related to the quantity of light arriving at the camera. However, it may also be necessary to model some aspects of the interplay of light with objects in the scene as part of the calibration process. Here, we will briefly describe that process and then present mathematical models that are suitable for understanding radiometric calibration techniques.

Images begin with light. One or more light sources illuminate objects in a scene that the camera is viewing. Reflection (both specular and diffuse), absorption, and transmission of light by the objects direct a portion of the light towards the camera. This light is measured with radiometric imaging.

The camera lens focuses the light onto an array of sensors. The lens usually includes an adjustable aperture that restricts the light reaching the sensors to prevent too much light falling on the sensors, because the sensors have a limited range of response. The aperture may be automatically adjusted under the control of a signal from the camera (auto iris), or it may be manually adjustable (manual iris).

The amount of light admitted by the lens depends on the angle between the light ray and the optical axis of the lens. As a result of optical geometry, less light is admitted for light rays that are farther from the optical axis, which modifies the effective aperture of the lens depending on the direction of the arriving light ray.

When the light strikes the sensors, photons of light are converted to electrical signals. In charge coupled device (CCD) and complementary metal oxide semiconductor (CMOS) sensors, photons dislodge individual electrons that are collected in a charge well. The exposure time of the sensors is controlled by the camera electronics. Before exposure commences, the charge wells are drained to remove all accumulated charge. During exposure, the charge well collects electrons dislodged by photons. The charge well also collects additional electrons dislodged by thermal motion, which produces a dark current—a base level of sensor response in the complete absence of light that is dependent on the sensor temperature and the exposure time. After the exposure time is complete, CCD arrays shift the collected charges to storage locations, one for each pixel. The charges are then read out by shifting them through the CCD array from one storage location to another until they reach an output amplifier. The amplifier produces an electrical signal in proportion to the charge it is reading at each point in time. Healey and Kondepudy present a more detailed model of CCD cameras for radiometric calibration, including consideration of noise characteristics (6). In contrast to CCD sensors, CMOS sensors amplify the collected charge at each pixel individually.

In digital cameras, the amplifier's signal is converted internally to digital pixel values that are transmitted to a computer for analysis. Some imaging systems use analog cameras that produce a video signal that is then converted to digital pixel values by a separate frame grabber card. The use of analog cameras is becoming less common as the availability, price, and quality of digital cameras has improved.

MATHEMATICAL MODEL OF IMAGE FORMATION

Mathematically, we can model the process of digital image formation for each individual pixel. First, we will model a typical digital camera including the lens.

Digital Camera Model

Let p designate a pixel. Let $L_p(\lambda)$ denote the radiance of light emanating toward the camera from the small portion of the scene that is imaged by pixel p. Here, the radiance is expressed as a function of wavelength λ, which represents the spectral characteristics (e.g., the color) of the light. Mathematically, $L_p(\lambda) = \mathrm{d}L_p/\mathrm{d}\lambda$ where L_p is the radiance as defined by the CIE (1).

Aperture. Let A_p denote the effective aperture for the pixel p. The effective aperture A_p has a multiplicative effect on the sensor response and is assumed to be wavelength independent. It includes the lens aperture and other effects such as vignetting and the $\cos^4$ effect. The effective aperture A_p is typically different for different pixel locations.

Sensor Sensitivity. Let $S_p(\lambda)$ denote the sensor sensitivity function of pixel p. $S_p(\lambda)$ is a function of the wave length λ of the incident light; for example, CCD sensors are typically more sensitive to red light than to blue light. Some cameras employ specific filters to modify the sensitivity function; an infrared blocking filter may be used to inhibit the response of a CCD sensor to infrared light. For the purposes of our equations, $S_p(\lambda)$ includes the effects of any such filters. The sensitivity function has a multiplicative effect on the light reaching the sensor for each wavelength λ. For a color or multispectral imaging system, a separate sensitivity function $S_{bp}(\lambda)$ exists for each spectral band b. Typically, we assume that $S_{bp}(\lambda) = s_{bp}S_b(\lambda)$ for all pixels p; that is, the spectral sensitivity curve is the same for all pixels for each spectral band b. Here, $S_b(\lambda)$ is the spectral sensitivity curve for spectral band b, and s_{bp} is the absolute sensitivity of the individual pixel p for spectral band b. Manufacturing tolerances can mean that individual pixel sensors have different dimensions and hence different sensitivity.

Exposure. Let t represent the exposure time. The exposure time has a multiplicative effect on the signal produced by light striking the sensor. The longer the exposure time, the greater the signal until the sensor reaches its operating limit and saturates.

Expressing the above concepts in mathematical notation, the following equation describes the sensor's exposure to light X_{bp}. X_{bp} is a linear equation of the scene radiance $L_p(\lambda)$.

$$X_{bp} = t\,A_p\,s_{bp} \int S_b(\lambda)\,L_p(\lambda)\,\mathrm{d}\lambda \tag{1}$$

Dark Current. In typical sensors, thermal motion dislodges some electrons even without light striking the sensor. These electrons are called dark current. Let D_{bp} denote the dark current of pixel p for spectral band b. The sensor's output signal is the sum of the dark current and the response to light striking the sensor. The dark current is part of the black response of the camera.

Camera Response Function. Let f_{bp} denote the CRF for pixel p and spectral band b. The CRF represents the nonlinearity of the camera. It combines the nonlinearity of the output amplifier and any subsequent electrical signal processing up to and including the conversion to a digital pixel value. Note that the electrical processing may also contribute to the black response of the camera; for example, the black response may be adjusted by modifying the video offset parameter that is often provided in video frame grabbers and digital cameras.

Camera Model. Let V_{bp} represent the digital value of pixel p for spectral band b. Then the following equation represents the formation of a multiband image as described above.

$$\begin{aligned} V_{bp} &= f_{bp}(X_{bp} + D_{bp}) \\ &= f_{bp}(t\,A_p\,s_{bp} \textstyle\int S_b(\lambda)\,L_p(\lambda)\,\mathrm{d}\lambda + D_{bp}) \end{aligned} \tag{2}$$

Ideally, we would like to estimate $L_p(\lambda)$ as a radiometric image with full spectral data for each pixel, but it is not possible with the limited spectral data from a camera. Instead, we estimate $C_{bp} = \int S_b(\lambda)\,L_p(\lambda)\,\mathrm{d}\lambda$ as a radiometric image for each spectral band b. If we know f_{bp}, A_p, s_{bp}, t and D_{bp}, then we can determine C_{bp} directly from V_{bp}. Under reasonable assumptions (such as monotonicity of f_{bp}), we obtain $C_{bp} = (\,f_p^{-1}(V_{bp}) - D_p)/(t\,A_p\,s_{bp})$, where f_p^{-1} is the inverse CRF. Unfortunately, it is unlikely that we have prior knowledge of all relevant parameters for a particular camera, so some method is required to estimate them. The estimation process produces radiometric calibration data for the camera.

Scene Illumination and Reflectance Model

Radiometric calibration techniques typically involve estimating f_{bp} and some or all other parameters. The estimation is based on images in which relevant parameters are known or controlled. These images may require some control over the scene radiance $L_p(\lambda)$. Because it is difficult to directly control the scene radiance, it is useful to model the formation of the scene radiance $L_p(\lambda)$ in terms of the illumination and reflection within the scene.

Let $E_{ip}(\lambda)$ denote the radiance of light from a particular light source i that falls on the small portion of the scene imaged by pixel p. Let $R_p(\lambda)$ denote the reflectance function of the small portion of the scene that is imaged by pixel p. $R_p(\lambda)$ denotes the proportion of incident illumination of wavelength λ that is reflected toward the camera, assuming no fluorescence.

Assuming that the conditions hold for the additive property of light, we can express the scene radiance in terms of the illumination and reflectance as $L_p(\lambda) = \sum_i E_{ip}(\lambda) R_p(\lambda)$. It follows that

$$X_{bp} = t A_p s_{bp} \sum_i \left(\int S_b(\lambda) E_{ip}(\lambda) R_p(\lambda) \, \mathrm{d}\lambda \right) \quad (3)$$

and thus

$$V_{bp} = f_{bp} \left(t A_p s_{bp} \left(\sum_i \int S_b(\lambda) E_{ip}(\lambda) R_p(\lambda) \, \mathrm{d}\lambda \right) + D_{bp} \right) \quad (4)$$

Some techniques for radiometric camera calibration can be understood primarily as methods for estimating f_{bp} by measuring or controlling some or all of the parameters and functions V_{bp}, t, A_p, s_{bp}, S_b, E_{ip}, R_p and D_p. If all these parameters are measured or controlled accurately in a sufficiently large and diverse set of sample images, then the camera response function f_{bp} can be accurately determined, and the inverse of this function can then be used to perform radiometric calibration. A later section considers techniques for radiometric calibration, including specific methods for estimating the CRF.

Modeling Effective Aperture

The lens aperture controls the light admitted by the lens. The aperture is an iris that may be controlled manually or automatically, depending on the lens design. Each aperture step (f-stop) typically represents a factor of approximately two in aperture area and hence a factor of two in light admitted by the lens. Larger f-stop numbers represent smaller aperture areas.

As a result of optical geometry, the amount of light admitted by the lens varies depending on the angle between the arriving light ray and the optical axis of the lens. This variation is captured in the effective aperture, which varies from one pixel to another. Two significant factors that affect the effective aperture are vignetting and the $\cos^4$ effect. The effective aperture is also affected by lens distortions such as pin-cushion and barrel distortions that change the density of the projected image.

Vignetting. Vignetting occurs for pixels that are far from the center of the image. It occurs because the light admitted to the lens does not fill the exit pupil of the lens—a portion of the exit pupil is in shadow (Fig. 1), so the effective aperture is reduced. Vignetting is most noticeable in the corners of an image and is strongest when the aperture area is large (i.e., for small f-stops). Vignetting results from the combination of lens design and use. A good lens design can reduce or eliminate vignetting entirely provided that the lens is used as intended. In particular, the sensor array physical dimensions should not exceed the dimensions assumed in the lens design. Asada et al. (8) mathematically model and correct vignetting in zoom lenses.

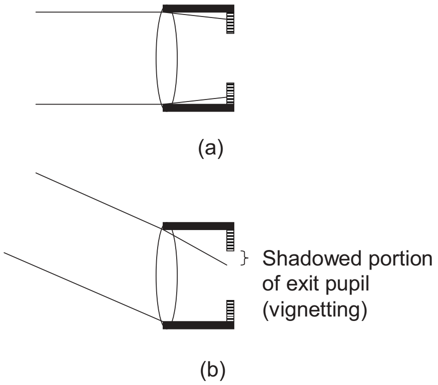


Figure 1. Vignetting in a lens. (a) Ray aligned with optical axis has no vignetting. (b) Ray produces vignetting due to angle from optical axis.

Cos^4 Effect

The $\cos^4$ effect applies to light arriving at the lens opening from directions that are not parallel to the optical axis. For most lenses, the effective illumination of the image sensor falls off as the fourth power of the cosine of the angle between the incident light ray and the optical axis (7). The $\cos^4$ effect is unavoidable, but some lens designs modify the $\cos^4$ effect through other factors (9). Graded neutral density filters can be used to optically correct the $\cos^4$ effect.

The combination of vignetting, the $\cos^4$ effect and lens distortion reduces the light that reaches an optical sensor by a factor that depends primarily on the angle between the incident light ray and the optical axis. For a fixed lens, the angle can be computed from the position of the pixel relative to the optical centre of the image. Note that the optical center of an image may not be the center of the image coordinates, because the sensor array may be off-center relative to the optics.

TECHNIQUES FOR RADIOMETRIC CALIBRATION

The first step in radiometric calibration is to estimate the CRF. The inverse of this function is used to correct the pixel data to be linear. The second step is to subtract the black response so that zero pixels represent absolute black. After this, the effective aperture and sensitivity corrections are applied, dividing each pixel by an individual scaling factor. These steps yield a relative radiometric image. For absolute radiometric calibration, a final step converts relative calibration to absolute based on a standard radiance.

Modeling the Camera Response Function

Direct modeling of the CRF assumes that the physical processes are sufficiently well understood, and a mathematical model of f_{bp} can be used. The simplest model is linear—the response is assumed to be a linear function of the incident light. For example, CCD sensors are known to be linear over a reasonable operating range. If the camera electronics are of sufficient quality, the sensor linearity may be adequately maintained during the conversion to digital pixels, which simplifies radiometric calibration. It is recommended, however, to check the linearity of the camera using one of the techniques described below.

Gamma Encoding. Many cameras deliberately introduce nonlinearity into the video signal, which is called gamma encoding. The original video signal V_0 is transformed by a power-law equation parameterized by gamma (γ), which yields a nonlinear video signal V_e.

$$V_e = V_0^{1/\gamma} \tag{5}$$

When this video signal is displayed on a cathode ray tube (CRT) monitor, the inverse gamma transformation is applied so that the light output from the monitor is a linear function of the light originally captured by the camera. Gamma encoding compresses the dynamic range of bright pixels relative to dark pixels in the transmitted video signal V_e, which provides improved transmission of detail in dark portions of the image. A gamma value of 2.2 is typical in television (10); the reciprocal (0.45) is used for simple gamma correction (11). Figure 2 shows ideal gamma response curves with $\gamma = 2.2$ and $\gamma = 1.4$, with a linear response curve for comparison.

Camera Response Function Models. Gamma encoding is often implemented in electronics and may not exactly follow the power-law curve. Errors in the model of the CRF limit the accuracy of the radiometric calibration, so more sophisticated functions are often used to model the CRF.

Grossberg and Nayar (12) model the CRF as the weighted sum of a set of basis functions. Principal components analysis is used to derive the basis functions from a database of 175 real-world response functions, including film brands, digital and video cameras, and a small selection of gamma curves. Three basis functions are sufficient to fit most CRFs with a RMSE of 0.7%.

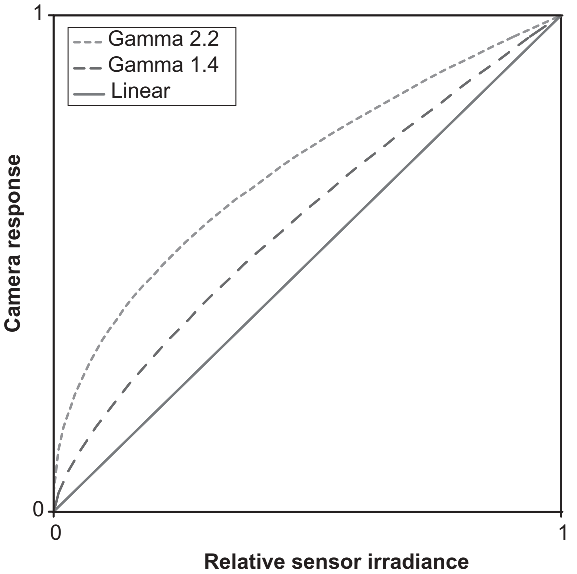


Figure 2. Examples of gamma response curves.

General-purpose modeling functions may also be used. Klinker, et al. (10) model the CRF using cubic splines. Hamey (13) uses a feed-forward artificial neural network. Mitsunaga and Nayar (14) use a polynomial model. Robertson et al. (15) represent the CRF as a mapping from 256 discrete camera response values to corresponding linear pixel values. They use Gauss-Seidel relaxation to solve this highly parameterized model.

Estimating the CRF

The CRF may be estimated from image data captured with the camera itself. To reduce the image data requirements, we may assume that the CRF is the same for all pixels (i.e., $f_{bp} = f_b$). With this assumption, we estimate a single CRF for each spectral (color) band of the camera, combining the data from multiple pixel locations. The validity of this assumption depends on the design and manufacture of the imaging system. For example, if all the pixels of a CCD array are read out by the same amplifier and analog-to-digital circuitry, we may reasonably assume that the CRF is the same for all pixels, provided that no pixel-specific nonlinearity exists in the camera design. Recent CCD cameras use two or more amplifiers (taps) to read the pixels out of the CCD array. The assumption of a common CRF may only be applicable to the regions of the images that share a single tap because manufacturing tolerances may produce different nonlinearity for each tap. CMOS sensors individually amplify each pixel, so manufacturing tolerances may produce different nonlinearity for each image pixel in CMOS cameras.

Typically, the CRF is estimated independently for each camera. Multiple cameras of the same make and model may have different CRFs because of manufacturing tolerances. Hamey (13) observed a difference of 1% between the CRFs for two cameras of the same make and model.

Estimating Gamma. If a gamma model is assumed, but the gamma parameter is unknown, then gamma may be estimated from image data. Farid (16) estimates gamma to an accuracy of 7.5% by analyzing the spatial frequency effects of the non-linear transformation introduced by gamma encoding. The nonlinear CRF introduces high spatial-frequency harmonics into the image. Using bispectral analysis, he recovers the gamma value that minimizes these harmonics in the corrected image.

Shafique and Shah (17) estimate gamma to an accuracy of 3% based on image invariants. In their technique, the same scene is imaged with different illumination conditions such as four separate light sources, one at a time. The technique uses cross-ratios of exposure terms that involve three color bands. These cross-ratios are invariant to changing illumination conditions under suitable models of scene reflection. The technique requires at least three spectral bands. Each spectral band may have a different gamma parameter.

Exposure Time-Based Methods. Controlling the exposure time t in Equation (2) gives direct control over the major part of the parameter of the function f_{bp}. A set of images is captured with different exposure times but with all other parameters remaining constant. For each pixel, the different images provide known relative exposures and corresponding camera response values that can be used to reconstruct the CRF.

Under the assumption of a shared CRF f_b, naturally occurring brightness variations in the scene increase the available information. For example, suppose n images are captured using exposure times t_1 through t_n. For each pixel location, these images yield a vector of camera response values (V_{1p}, V_{2p}, $\ldots$ V_{np}) corresponding to the exposures ($k_p t_1$, $k_p t_2$, $\ldots$, $k_p t_n$) for some unknown constant k_p that depends upon the scene radiance, effective aperture and other variables. The CRF is reconstructed by analyzing these data vectors.

The simplest method of recovering the CRF from such data is to collect many images with exposures densely covering the entire range of camera response for a single pixel. Plotting the pixel values against the corresponding exposure times for a single pixel reveals the CRF. More sophisticated analysis combines incomplete curves from different pixels to derive an overall curve from fewer images.

Techniques based on exposure time are limited by the accuracy of the camera's exposure time control. In addition, theoretical limitations exist. Grossberg and Nayar (18) show that a self-similar ambiguity exists when only two images are used or when the exposure ratios of multiple images are similar, such as doubling the exposure of each image compared with the previous image. The ambiguity can be resolved by using at least three images with exposure ratios $a = t_2/t_1$ and $b = t_3/t_2$ such that $\log_a b$ is irrational. Alternatively, the ambiguity may be resolved by imposing smoothness constraints on the CRF (15), or by assuming a model with a limited number of parameters such as basis functions (12) or a polynomial model (14). Radiometric calibration based on known exposure times is included in HALCON, a commercial machine vision library (19).

Chart-Based Methods. In this approach, the CRF is estimated from images that contain calibration targets with known reflective properties. For example, a Kodak grayscale contains 21 reflectance targets with different gray levels. In one method, each reflectance target is imaged in turn at the same physical location under time-constant illumination. In a slightly different approach, the entire grayscale is imaged under uniform illumination. Using either source of image data, the mean pixel value of each target is calculated and related to the known reflectance level of the corresponding target. The CRF is estimated by interpolating the data points. The accuracy of this technique is limited by the accuracy of the reflectance data provided with the reflectance targets and also by interpolation errors. When the entire gray scale is imaged at once, the accuracy of the technique is limited further by spatial variations in the effective aperture and sensitivity.

Klinker et al. (10) use the GretagMacbeth Color Checker (X-Rite Incorporated, Grand Rapids, MI) to estimate the CRF by fitting cubic splines to the camera response data from a single image. Grossberg and Nayar (12) also use the GretagMacbeth Color Checker to estimate the CRF by fitting a set of basis functions derived from a database of known CRFs.

Additive Illumination. Using the additive property of light, it is possible to solve for unknown illumination levels, scene reflectance values, and camera response function simultaneously. The techniques use multiple light sources. Images are taken with each light source individually and with combinations of the light sources. Because of the additive property of light, the exposure of images with multiple light sources is the sum of the exposures with the individual light sources.

Manders et al. (20) use a static real-world scene with two different illumination sources. Taking an image with each source independently and a third image with both sources, they use singular value decomposition (SVD) to solve for the inverse CRF using the equation

$$f^{-1}(V_{1p}) + f^{-1}(V_{2p}) = f^{-1}(V_{3p}). \qquad (6)$$

Here, V_{1p} and V_{2p} represent corresponding pixels from the images taken with individual light sources, whereas V_{3p} is the corresponding pixel with both light sources active. The technique assumes no ambient illumination. The inverse CRF is represented as a discrete function.

Hamey (13) uses four illumination sources and a collection of reflectance targets. Each reflectance target is imaged under all combinations of the four lights being gated on and off. The experiments are conducted in a dark room so that the only ambient illumination is scatter from the lights. At least 40 targets are used, which vary from black through white, yielding 640 images. A neural network model is fitted to estimate the CRF, the unknown illumination level of each light, and the unknown relative reflectance value of each target for each pixel location simultaneously.

The model simplifies Equation (4) above. Assuming that the spectral distribution of all light sources is the same, let $E_{ip}(\lambda) = e_{ip} E(\lambda)$ where e_{ip} is the relative brightness of lamp i illuminating the region of the target imaged by pixel p, and $E(\lambda)$ is the common spectral curve. Let T represent a reflectance target, and let $R_{pT}(\lambda)$ denote the reflectance curve of target T. Let $R_{bpT} = t A_p s_{bp} \int S_b(\lambda) E(\lambda) R_{pT}(\lambda)\, d\lambda$, which combines the spectral curves and the exposure parameters into a single relative reflectance value. Allowing the CRF to be an arbitrary monotonic function g_{bp}, subsume the dark current D_{bp} into the CRF. With some manipulation, we derive

$$V_{bp}(j) = g_{bp}\left(\left(\sum_T P_T(j)\, R_{bpT}\right)\left(\sum_i (L_i(j)\, e_{ip}\right)\right). \qquad (7)$$

Here, j denotes an image, $P_T(j) = 1$ when target T is present in image j and $L_i(j) = 1$ when lamp i is on in image j. Ambient illumination is assumed to be constant and is represented by $i = 0$ with $L_0(j) = 1$ for all j.

Fitting the above model as a neural network, Hamey (13) recovered CRFs with a root mean squared error (RMSE) of 0.15%, applying the results to the color inspection of baked goods.

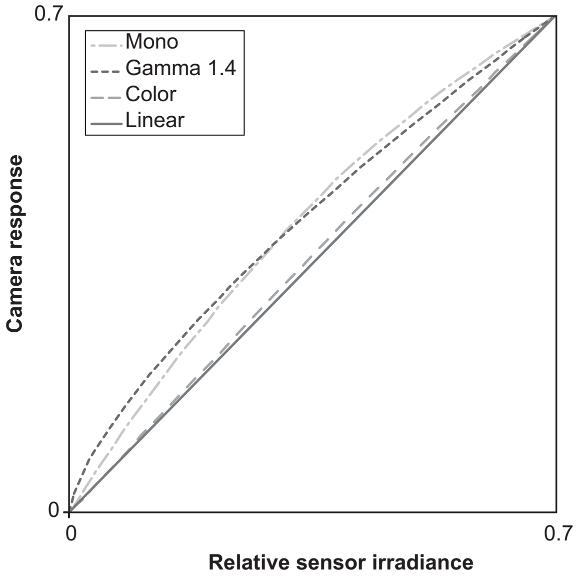


Figure 3. Examples of camera response functions for a monochome CCD camera and a color CCD camera, with a gamma curve and linear response for comparison.

Figure 3 shows measured camera response functions for a monochrome CCD camera and for one band of a color CCD camera. The camera response functions are shown over the range from absolute black to approximately 70% of the camera response range (pixel value 183) to avoid sensor saturation effects. For comparison, the figure includes a linear response curve and a gamma response curve with $\gamma = 1.4$. The monochrome camera's response curve was nominally a gamma curve, but it deviates significantly from the ideal gamma response curve. The color camera was operated with gamma encoding disabled, but it deviates from linear by up to 1.7%. In contrast, the monochrome camera deviates from a linear response by up to 12%.

Figure 4 shows the linear correction function for the monochrome camera. It is implemented as a lookup table that converts camera pixel values to the corresponding linear response values. Note that, for this camera, pixel value 26 corresponds to absolute black.

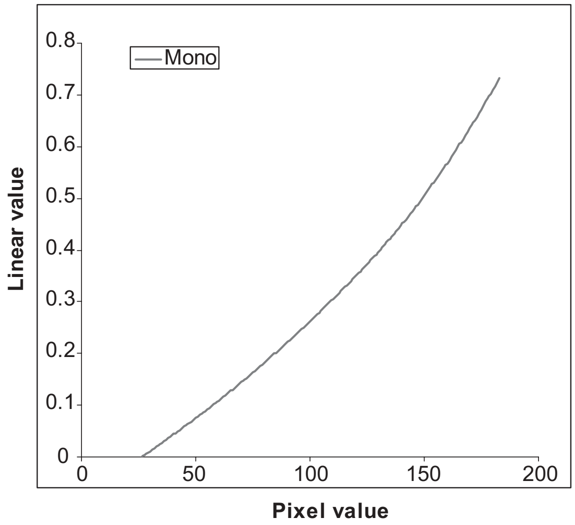


Figure 4. Linear correction function for a monochrome camera.

Comparametric Analysis. The comparametric approach involves solving equations of the form $h(f(X_1)) = f(kX_2)$ where f is the unknown CRF, k is the unknown exposure ratio between a pair of images, X_1 and X_2 are unknown exposure levels for corresponding pixels in the pair of images, and h is the comparametric function (also known as a mapping function or registration function) that maps pixel values in one image to the corresponding pixel values in another image (i.e., $h(V_1) = V_2$). Mann (21) shows that f is related to h. For example, if f is gamma encoding, then h is a straight line. Fitting h yields a corresponding solution for f, but this solution is exponentially ambiguous (18), because it yields an unknown gamma encoding unless the exposure ratio is known. An unknown gamma encoding is not suitable for radiometric calibration but remains useful for image stitching and HDR imaging applications as discussed below.

Measuring the Black Response

Radiometric calibration requires that the calibrated response value is zero when the camera views absolute black. However, a linear CRF may have a nonzero response value for black. A relative radiometric calibration can be achieved by measuring the black response and subtracting it from the linear pixels. The following are some techniques that may be used to measure the black response.

Lens Cap Technique. A simple technique for measuring the camera's black response is to cover the camera lens so that no light is admitted and then capture a black image. The black image is subtracted from other images to remove the black response.

This technique is widely used but may yield incorrect results depending on the camera and frame grabber. In particular, if the frame grabber dynamically adjusts the digitization range to the range of the video signal, then an all-black image may yield different pixel values than an equally dark object in an ordinary image. When using the lens cap technique, the all-black image should be compared with a black reference in a normal image to ensure that the imaging system has not adjusted inappropriately to the all-black image conditions.

Masked Pixels. Some machine vision cameras can measure the sensor dark current and automatically subtract it from the image data. The design uses masked pixels (not exposed to light) to measure the dark current.

Shadowed Black Reference. It is difficult to create a truly black object. Black materials still reflect some light, typically 2% or more. To achieve blackness that is close to 0% reflectance, the black material must be overshadowed to protect it from all sources of illumination.

A simple technique previously used by the author involves creating a black box (Fig. 5). The box is painted matte black on the inside and has a small hole in one face.

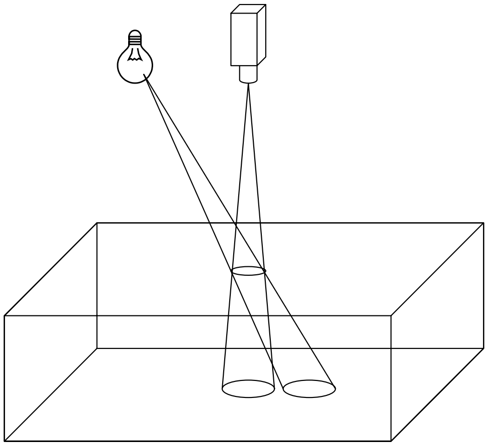


Figure 5. Black box black reference.

The camera views a portion of the far side of the box through the hole. The depth of the box is such that lamp light that passes through the hole does not directly illuminate the area viewed by the camera. Lamp light entering the hole is scattered inside the box and is absorbed by the black paint so that very little of it ever reaches the camera. A large box with a small hole produces a black reference region that is very close to 0% reflectance.

Calibrated Reflectance Targets. Using two targets with known reflectance values, the linear camera response can be extrapolated to estimate the absolute black response. For example, one may use targets with known reflectance values of $R_1 = 2\%$ and $R_2 = 99\%$. If the corresponding (average) linear pixel values over the targets are T_1 and T_2, then the estimated black response is $T_1 - R_1(T_2 - T_1)/(R_2 - R_1)$.

To eliminate effective aperture effects, the two reflectance targets should be placed in the same position in separate images (assuming constant illumination). If a single image is used, then the targets should be placed in comparable positions relative to the optical axis of the camera and the illumination should be uniform for both targets.

It is also important to ensure that the illumination and viewing conditions in which the calibrated target is used are consistent with the original calibration method. The reflectance of a surface depends on the angles of illumination and viewing, so if the original calibration used a different configuration, then the calibration data may be inaccurate for the particular application.

Measuring the Effective Aperture and Sensitivity

As discussed above, the effective aperture depends on the radial position of the sensor element relative to the optical axis. The effective aperture can be modeled as a combination of vignetting (8) and the $\cos^4$ effect (7). However, the model parameters depend on the lens design, and it is difficult to ensure their accuracy. An alternative approach is to measure the effective aperture and sensitivity as a combined per-pixel response field.

For measurement, a flat radiance field is required. A flat Lambertian surface under uniform illumination is suitable. After linear correction and black response subtraction, an image of the flat radiance field records the combined sensitivity and effective aperture for each pixel. To avoid the effects of small imperfections in the Lambertian surface, the surface or the camera may be moved between taking multiple images. Linear correct these images, subtract the black response, and combine them on a per-pixel basis using a robust technique such as a trimmed mean. The result is a response field image that may be normalized to have a maximum value of one. To calibrate subsequent data images, first apply linear correction and subtract the dark response, then divide corresponding pixels by the response field. This process yields relative radiometric calibrated images.

It may be difficult to obtain uniform illumination. If it is not possible to obtain a large, truly flat radiance field, then a small time-invariant flat radiance field can be used to measure the response field of small areas of the sensor array, one area at a time, by changing the camera view angle between sets of images.

Absolute and Reflective Radiometric Calibration

The final step for absolute or reflective radiometric calibration involves multiplying the relative radiometric image by a scale factor. To obtain the scale factor, capture and calibrate a reference image that contains an appropriate reference target. For absolute calibration, a radiance standard is appropriate, such as a calibrated lamp or the sun (22), whereas reflective calibration requires a reflectance standard such as a calibrated target. The required scale factor is the reciprocal of the average pixel value that corresponds to the standard in the reference image. The camera and lens settings must be the same for all images, including the reference image.

APPLICATIONS OF RADIOMETRIC IMAGING

Object Shape Estimation

Shape from shading estimates the local surface orientation of Lambertian objects. For such surfaces, the reflected light is proportional to the cosine of the angle between the local surface normal and the incident light ray. Assuming smoothness of the object surface, an under-constrained set of equations can be solved to estimate the local surface orientation over the object. Shape from shading requires relative radiometric calibration to relate the reflected light values to each other and to estimate the surface orientation.

Photometric stereo is similar to shape from shading but uses more than one image. Each image is taken with a different light source, which provides multiple light measurements at each point. With at least three light measurements, the local surface orientation can be calculated directly from the light measurements. Photometric stereo also requires relative radiometric calibration.

Remote Sensing

Remote sensing uses multispectral data to measure properties of the earth's surface such as identifying minerals, crops, or other land use. Each material to be identified has a characteristic spectral reflectance signature. To recognize particular spectral signatures, the image data must be radiometric. Both relative and absolute radiometric calibrations are used (2) with the ultimate goal of reflective calibration. Slater et al. (22) discuss absolute calibration using the sun as a reference light source.

Color Measurement

Color measurement uses reflected light to measure the color of objects. Color may be expressed in RGB color space or converted to another color space such as CIE $L^*a^*b^*$. The color space conversion assumes a linear (relative radiometric) RGB color measurement. The image is calibrated to produce color measurements using color standards for reflective calibration. Uneven illumination may also be corrected (23).

Subpixel Edge Detection

Subpixel edge detection locates edges in images to a fraction of a pixel position. The technique is based on the profile of the pixel values across the edge. Because of lens limitations and the dimensions of the sensor elements, pixel values close to an edge are illuminated by light from both sides of the edge. For a straight edge, the subpixel edge location is the position where the pixel receives equal portions of light from each side of the edge. If the camera response function is linear, then the corresponding pixel value is halfway between the pixel values that represent the regions on either side of the edge. However, if the camera response function is nonlinear, then the edge response curve is distorted, and the subpixel edge location may be incorrect. Relative radiometric calibration produces linear pixel values, which are suitable for accurately estimating the subpixel edge location.

Image Stitching and HDR Imaging

Image stitching (24) combines multiple overlapping images of a scene to produce a single, large image. It has applications that range from consumer digital photography, in which panoramas are created by stitching images together to scientific applications in space exploration. Images to be stitched together must be transformed to have the same exposure and CRF to eliminate brightness steps at the stitch lines.

HDR imaging (14, 15, 21) combines multiple images of the same scene to provide an increased dynamic range. For example, one image may be taken with a short exposure time to expose a sunlit scene of a hillside correctly while underexposing the interior of a tunnel in the hillside. A second image may be taken with a long exposure time to expose the interior of the tunnel correctly while overexposing the brightly lit hillside. Combining the two images would provide a HDR image that shows both the hillside and the interior of the tunnel.

Radiometric calibration is sufficient but not required for these applications. It is sufficient that all the images have the same arbitrary monotonic relationship between scene radiance and pixel values. In practice, images are transformed to a common brightness scale that may include an unknown gamma encoding. Comparametric analysis may be used to derive the transformation from a collection of spatially-registered images with different unknown exposures. These images may be the source images themselves (21, 25).

BIBLIOGRAPHY

1. International Commission on Illumination, *International Lighting Vocabulary*, 3rd ed., CIE publication no. 17 (E-1-1), Vienna, Austria: International Commission on Illumination (CIE), 1970.
2. M. Dinguirard and P. N. Slater, Calibration of space-multispectral imaging sensors: a review, *Remote Sens. Environ.*, **68**: 194–205, 1999.
3. K. D. Mielenz and K. L. Eckerle, Spectrophotometer linearity testing using the double-aperture method, *Appl. Opt.*, **11**: 2294–2303, 1972.
4. J. F. Clare, Correction for nonlinearity in the measurement of luminous flux and radiant power, *Measur. Sci. Technol.*, **13**: N38–N41, 2002.
5. Y. Ohno, *NIST Measurement Services: Photometric Calibrations*, Gaithersburg, MD: National Institute of Standards and Technology, NIST Special Publication 250–37, 1997.
6. G. E. Healey and R. Kondepudy, Radiometric CCD camera calibration and noise estimation, *IEEE Trans. Patt. Anal. Mach. Intell.*, **16**: 267–276, 1994.
7. B. K. P. Horn, *Robot Vision*, Cambridge, MA: MIT Press, 1986.
8. N. Asada, A. Amano, and M. Baba, Photometric calibration of zoom lens systems, *Proc. 13th International Conference on Pattern Recognition*, vol. **1**, 1996, pp. 186–190.
9. S. F. Ray, *Applied Photographic Optics: Imaging Systems for Photograph, Film and Video*, London: Focal Press, 1988.
10. G. J. Klinker, S. A. Shafer and T. Kanade, The measurement of highlights in color images, *Internat. J. Comp. Vis.*, **2**: 7–32, 1988.
11. Y. V. Haeghen, J. M. Naeyaert, I. Lemahieu and W. Philips, An imaging system with calibrated color image acquisition for use in dermatology, *IEEE Trans. Med. Imag.*, **19**: 722–730, 2000.
12. M. D. Grossberg and S. K. Nayar, Modeling the space of camera response functions, *IEEE Trans. Patt. Anal. Mach. Intelli.*, **26**: 1272–1282, 2004.
13. L. G. C. Hamey, Simultaneous estimation of camera response function, target reflectance and irradiance values, *Proc. of Digital Image Computing: Techniques and Applications*, 2005, pp. 51–58,
14. T. Mitsunaga and S. K. Nayar, Radiometric self calibration, *IEEE Conf. Comp. Vis. Patt. Recog.*, vol. **7**, 1999, pp. 374–380.
15. M. A. Robertson, S. Borman, and R. L. Stevenson, Dynamic range improvement through multiple exposures, *Internat. Conf. Image Proc.*, vol. **3**, 1999, pp. 159–163.
16. H. Farid, Blind inverse gamma correction, *IEEE Trans. Image Proc.*, **10**: 1428–1433, 2001.
17. K. Shafique and M. Shah, Estimation of the radiometric response functions of a color camera from differently illuminated images, *International Conference on Image Processing*, 2004, pp. 2339–2342.

18. M. D. Grossberg and S. K. Nayar, Determining the camera response from images: what is knowable? *IEEE Trans. Pattern Anal. Mach. Intell.*, **25**: 1455–1467, 2003.
19. radiometric_self_calibration. (2007, October 22), MVTec HALCON Documentation. Available: http://www.mvtec.com/download/documentation/reference-8.0/hdevelop/radiometric_self_calibration.html.
20. C. Manders, C. Aimone, and S. Mann, Camera response function recovery from different illuminations of identical subject matter, *International Conference on Image Processing*, vol. **5**, 2004, pp. 2965–2968.
21. S. Mann, Comparametric equations with practical applications in quantigraphic image processing, *IEEE Trans. Image Proc.*, **9**: 1389–1406, 2000.
22. P. N. Slater, S. F. Biggar, J. M. Palmer, and K. J. Thome, Unified approach to absolute radiometric calibration in the solar-reflective range, *Remote Sens. Environ.*, **77**: 293–303, 2001.
23. Y.-C. Chang and J. F. Reid, RGB calibration for color image analysis in machine vision, *IEEE Trans. Image Proc.*, **5**: 1414–1422, 1996.
24. A. Litvinov and Y. Y. Schechner, Radiometric framework for image mosaicking, *J. Optical Soc. Am. A*, **22**: 839–848, 2005.
25. F. M. Candocia, Simultaneous homographic and comparametric alignment of multiple exposure-adjusted pictures of the same scene, *IEEE Trans. Image Proc.*, **12**: 1485–1494, 2003.

Leonard G.C.Hamey
Macquarie University
Sydney, Australia

RAY TRACING ACCELERATION TECHNIQUES

INTRODUCTION

Today, ray-tracing techniques (RTT) are being used successfully in many computer applications (1–4). The most common applications are static and dynamic scene visualizations, movies, commercials, and video games, wherein near perfect realism can be achieved. Virtual reality is also an important field for the application of RTT, not only for recreational purposes, but also for scientific and engineering research. Another field in which RTT play a crucial role is in the study of acoustic and electromagnetic wave propagation in complex scenarios (5). The deployment of modern radio communication networks, particularly for communication or surveillance wireless systems, requires high-quality RTT that are well fitted to the particular needs of this field.

The aforementioned RTT applications (including many others that are not cited or that will be developed in the future) have a common need for computational efficiency in the vein of central processing unit time and memory usage. Often, very complex scenarios are visualized or analyzed in “real time” for many observation points or illumination sources.

This article is devoted to presenting a tutorial view of modern RTT. Although all the aforementioned techniques share the common need for computational efficiency, the geometry and morphology of scenarios, the nature of the lightsources, and the phenomenology of wave transmission and scattering can differ drastically. Each application requires a partially or completely specialized RTT.

In the next section, we consider the different kinds of geometries, morphologies, and qualities of illumination found in ray-tracing problems. Then, we introduce ray-tracing mechanisms to elucidate the different tasks involved in successfully applying RTT. The problems and corresponding complexity of these tasks in typical scenarios are covered in the section on problems and complexity of RTT, wherein the two main subproblems associated with RTT are discussed: flash-points searching, that is, searching for reflection, diffraction, and transmission points, and ray shooting query, which consists of determining whether a given segment is cut by any entity of the scene. The two main strategies for addressing the first subproblem are discussed in the sections on RTT strategies for the *flash-point searching* (FPS): the shooting and bouncing of rays and the solution of the inverse problem. Several efficient techniques for solving a ray shooting query are outlined in the section on Algorithms to reduce the computationalcost. The section on the angular Z-Buffer algorithm introduces the angular zeta buffer algorithm as a solution to the inverse problem strategy and also as a tool for flash-point searching. Finally, the last section compares the techniques presented in the previous two sections.

RTT SCENARIOS. DESCRIPTION OF THE GEOMETRY

The data that describe a scenario can be defined in a raster or a vector format (6). For the raster format, a scenario is divided into cells (i.e., pixels). For the vector format, the scene is defined by a set of geometrical entities (i.e., surfaces or volumes). Scenarios will be defined using the vector format for RTT applications. Several algorithms are available to transform raster formatted scenes into vector scenes (7).

Flat-facet models define the geometry of many visualization applications. Flat-facet models are also employed in many electromagnetic tools for analyzing radio propagation or for studying the scattering from complex structures. This method can approximate an object’s curved surface using interconnected sets of plane facets and straight edges. This approach has been used ubiquitously because of its simplicity. The high number of flat facets, which can range from the thousands to the millions, required to approximate a complex curved object can generate artificial edges (edges between two approximating facets) that are not present in the real curved object. Figure 1 depicts an example of a DC 10 aircraft modeled by plane facets. A total of 2317 facets are required to model the object accurately.

As a result of flat facet-induced error, some computer graphics applications, especially in the automobile and aerospace industries, and industries using computer-aided design software, require the usage of curved surfaces, which are often in the format of nonuniform rational b-spline surfaces (NURBS) (7,8). Using NURBS as an approximating surface set provides an accurate representation of a real object that employs significantly less information (on the order of a thousand curved surfaces), and mitigates the inherent problem of flat-facet approximation-induced artificial edges. Figure 2 shows the same aircraft from Fig. 1, but it is modeled by NURBS surfaces. Only 168 surfaces are necessary to represent the object accurately, and no artificial edges are generated. The primary disadvantage of the NURBS representation is that the computational cost of the ray-tracing is much greater than that of the flat facets model (5,9,10). The details of this will be discussed later.

RAY-TRACING MECHANISMS

A ray-tracing mechanism is defined as a set of coupling rays that link a source with an observation point (see Fig. 3). As illustrated in Fig. 3, these rays can be a direct ray from the source to the observation point, simple effect rays, double effect rays, and higher-order effect rays. Simple effect ray-tracing mechanisms are the reflected rays, the transmitted rays, and the diffracted rays. Reflected rays are defined as rays that leave a source, are reflected in a facet, and then arrive at the observation point. Rays that link the source and observation point by transmission through a facet are transmitted rays. Rays that link the source and observation point by scene edge diffraction are diffracted rays. Double or higher-order effects are combinations of simple effects.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

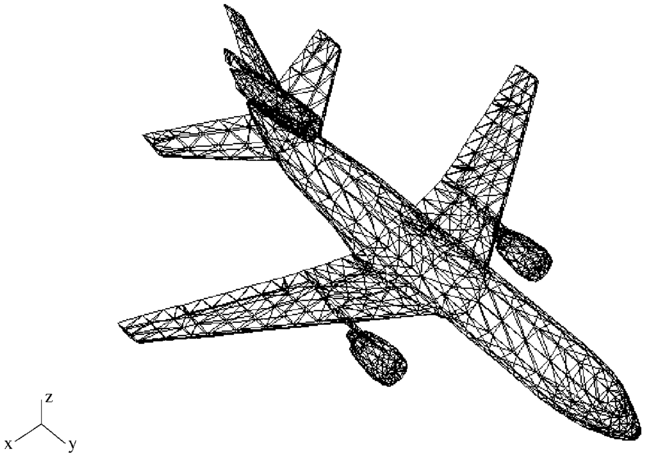


Figure 1. Geometrical model of a DC10 plane using 2317 flat facets.

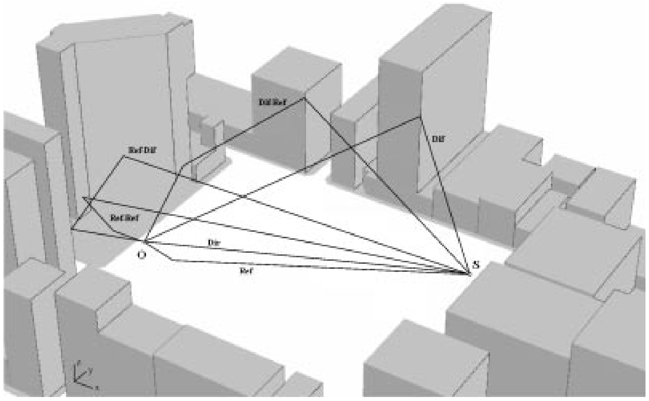


Figure 3. Example of ray-tracing mechanisms on an urban scene given the position of the source and the observation point. Only simple (direct, reflected, and diffracted rays) and double (double reflected, reflected-diffracted, and diffracted-reflected rays) effects are depicted in the figure for the sake of simplicity.

For example, a third-order ray can be a reflected-transmitted-reflected ray. The points involved in each ray are known as flash points. Shadowing is another relevant RTT mechanism; a ray is shadowed if any segments forming that ray are occluded by any of the scene facets. When occlusion occurs, the ray must then be discarded because it does not contribute to the illumination of the observation point. If the ray mechanism under analysis includes transmission, then the transmission facet should not be considered in the shadowing test, and the contribution of that ray should be considered for the illumination of the observation point.

Among the aforementioned mechanisms, the most common are the shadowing and diffuse reflection of rays that leave illuminating sources. The shadowing discard, as not visible from an observation point, includes many facets of a scene. The diffuse reflection permits a visualization of all points on every surface visible from the observation point. Transmission through surfaces (transparency), refractions in volumes, double, triple, and higher-order reflections on surfaces, may also be required for visualization. For RTT electromagnetic applications, the number of reflections, transmissions, and combinations of both is usually high, which require a different treatment. Reflections are treated as specular; however, if the surface is not smooth with respect to the wavelength, part of the energy is reflected in nonspecular directions. Models can be used to compute the energy fraction reflected along the specular direction for ray tracing, but energy reflected along nonspecular directions is not used in RTT, which is a significant limitation of this technique in electromagnetic applications. Furthermore, these applications require an accurate evaluation of the path length for phase computation, because illumination is achieved using coherent waves. Coherent waves can interfere destructively depending on their relative phase, whereas incoherent waves interfere constructively. Additionally, wedge-edge diffractions (11) play an important role in these applications because of the broad angular expansion of the waves after diffraction. This phenomenon has been demonstrated to supply the illumination in many areas of a scene.

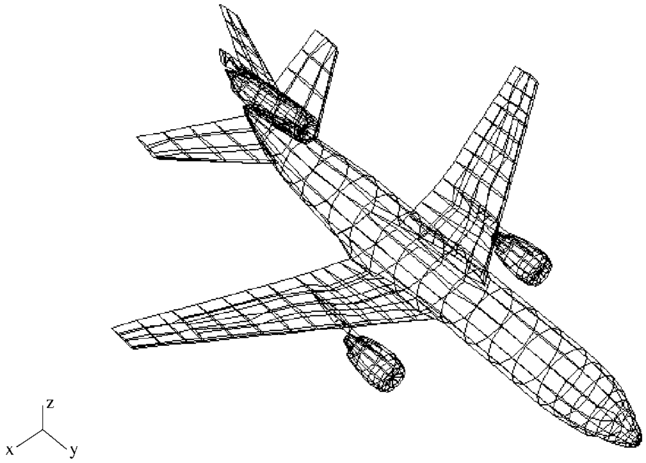


Figure 2. Geometrical model of a DC10 plane using 168 NURBS surfaces.

PROBLEMS AND COMPLEXITY OF RTT

The main problem inherent to RTT involves the fact that a complex scene is composed of many flat and/or curved facets illuminated by a source (see Fig. 4). We need to know all the coupling ray mechanisms that link the source with the observation point. These rays, as stated in the previous section, can be the direct ray from the source to the observation point, simple effect rays, double effect rays, and higher-order effect rays. Thus for a given problem, two subproblems must be solved:

1. We must establish the set of all facets that are involved in each ray mechanism (e.g., in a double reflection this set is formed by two facets). Furthermore, some additional information of the intersection point of the ray with its associated facets set must be obtained (e.g., intersection point coordinates, radii of

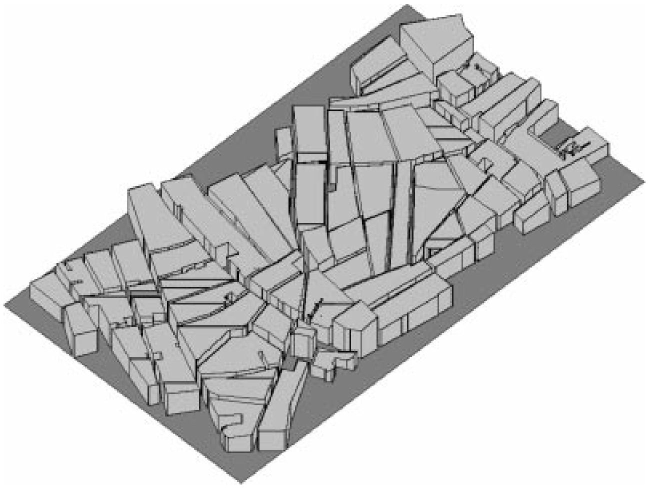


Figure 4. Example of an urban scene defined by several thousand facets. The fields radiated by a Base Station located at the top of a high building requires the evaluation of tens of thousands of observation points along the streets of the urban scene.

curvature, and surface roughness at those points). This part is known as *flash-point searching* (FPS) and is accomplished by using the strategies described in the next section.

2. We must conduct the shadowing analysis, as described in the previous section, to determine whether the rays contribute to the illumination of the observation point; if so, they must be discarded. To solve this problem, we must apply intersection tests to determine the facets that potentially can occlude a ray. This problem is known as the *ray-shooting query* (RSQ)

The RTT problem is usually solved for many observation points and for many source positions. Therefore, the level of complexity is very high.

In the vein of the first subproblem (the FPS), if we have a scene with N_f facets and N_e edges (see Fig. 4) for a given source and intersection point, then every facet can be involved with a reflected or transmitted ray, and every edge can be involved with diffracted rays. Therefore, they all must be investigated to determine whether these effects are present. The computational cost for reflection, transmission, and diffraction is on the order of $2N_f + N_e$. Once a simple effect is produced, a double effect can appear with any element in the scene except the one from which the simple effect originally took place. The computational cost to determine simple effects is on the order of $(2N_f+N_e)\cdot(2N_f+N_e-1)$. Following this thought process, the computational cost to solve the subproblem for an n-order effect is proportional to $(2N_f+N_e)\cdot(2N_f+N_e-1)\ldots.\ (2N_f+N_e-r+1)$, which can be greater than the computational capacity of a powerful computer running for several days (a typical value for N_f is 100,000, whereas 3 or 4 are common values for n in some electromagnetic applications).

With respect to the RSQ, the number of intersection tests that a naïve algorithm (an algorithm that exhaustively checks all facets) requires for determining whether a direct or simple effect ray is occluded by any facet of the scene is of the order of N_f. For a n-order effect ray, the number of intersection tests may become as high as $n \times N_f$. On the other hand, the number of facet sets that can produce an n-order effect is $N_f(N-1)\ldots.(N_f+1-n)\approx(N_f)^n$. Considering the previous typical value for N_f, a naïve procedure that exhaustively checks all the potential facets sets for a given ray will require unaffordable computer resources.

Therefore, for both subproblems, efficient algorithms should be used to reduce the computational cost of ray tracing. The following sections provide a description of the most efficient procedures developed therein.

RTT STRATEGIES FOR THE FPS

The complexity of the first subproblem can be reduced drastically by using modern RTT algorithms. Two main strategies are implemented in these algorithms:

(a) the shooting and bouncing of rays (SBR) strategy (11,12)

(b) the solution of the inverse problem (SIP) strategy (13)

The SBR strategy is illustrated in Fig. 5. This technique is based on shooting many rays in radial directions from the source. The path followed by each individual ray is obtained. This technique only considers the contribution from the reflections in the specular direction. Therefore, applying this technique, when a ray hits a surface, its path is redirected following Snell's law of reflection (14), and if the surface is penetrable, the law for transmission. The ray path is followed until it leaves the volume of the scene or when it has suffered a predetermined number of reflections. A ray is considered to contribute to the total illumination

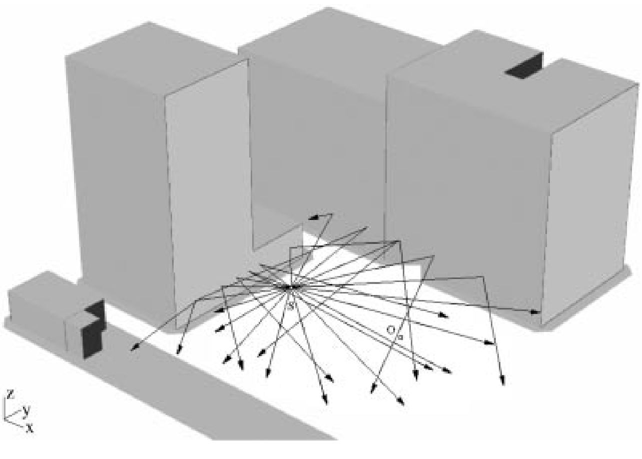


Figure 5. Example of SBR on a simple urban scene. Many rays are launched from the position of the source (S), and only the rays that pass close to the position of the observation point (O) are considered to illuminate that point.

strength at a given observation point when it passes near that point (5,15). The contribution of the ray is added to the total strength of the illumination at the observation point. This contribution is computed by considering the distance from the observation point to the ray and other ray parameters, such as the spread factor (which determines the variation of the amplitude of the ray because of the ray propagation and depends on the length of the ray path) and the number and nature of the reflections that the ray has suffered before approaching the observation point. This procedure estimates (or extrapolates) the parameters of the ray that has the same ray path history (e.g., reflected/transmitted on the same surfaces) and impacts the observation point. The estimation of the ray parameter is usually enough to obtain a good approximation of the contribution intensity of the ray to the total illumination at the observation point when the illumination is composed of incoherent waves. The SBR strategy works well for many applications of visualization of scenes, wherein the ray effect order is not greater than two or three (and often the simple reflection is enough to describe the problem). Also, in these applications, the illumination is incoherent light, which simplifies the evaluation of the ray contribution because no phase computation of the field is needed. The phase evaluation requires a very precise computation of the path length, which is difficult to obtain using the SBR strategy. In any case, the SBR needs to shoot an extremely dense beam of rays from the source point covering the entire spatial angle that illuminates the scene; typically more than 50,000 rays are shot.

The philosophy of the SIP strategy is different. Instead of shooting many rays from the source to find those that reach the observation point, the SIP attempts to solve the following inverse problems:

- Determine whether the direct ray connecting the source and observation points is occluded.
- Find all facets of the scene where reflected (or transmitted) rays can link the source and the observation points. After identifying these surfaces, the ray paths of the reflected (or transmitted) rays are obtained.
- Find all edges of the scene that can link the source and the observation point by diffraction. Determine the corresponding ray paths.
- Find all sets of facets that can join the source and the observation points by a second or higher order coupling mechanism and the corresponding ray paths.

The total strength of illumination is computed by adding the contribution of all ray paths found that link the source and the observation points. A rays path that connects the source and the observation points through reflections (or transmissions) can be computed in two different ways: using the image method for the case of a flat facets (5) or by minimizing the total length of the path (16) for curved facets.

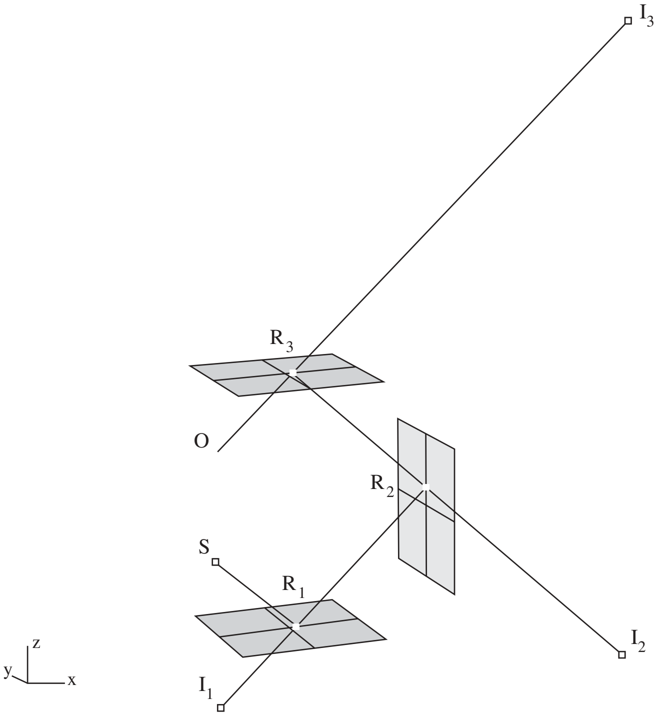


Figure 6. Example of the ray-tracing of a triple reflection using image method. Q1

The image method is based on image theory, which states that, given a source point and a flat facet, the reflected rays in the facet can be considered as rays radiated by the image source. The image source is a virtual source whose location is the specular image of the original source with respect to the plane that contains the flat facet. Once obtained, and given the observation point and image position, the position of the reflection point for that observer can be obtained easily as the intersection of the segment that links the image with the observation point and the facet. These procedures can be generalized to obtain multiple reflections, simply by obtaining the nth-order image of the source, and from that, obtaining the complete trajectory of the multiple-reflected ray. Figure 6 illustrates this procedure.

If curved facets are used to model the scenario, then the image method cannot be applied. The trajectory of the ray is obtained using the generalized Fermat's principle (14). This principle states that the length of the ray path that links two points is a minimum. A distance function can be defined whose value is the length of the ray as a function of the position of the points over the curved facet. The idea is to find the point whose distance function is a minimum, which determines this point as a reflection point. This procedure has two disadvantages: First, the computational cost associated to the minimization process is high, and second, the possibility of local minima in the distance function can provide an erroneous solution for the position of the reflection point. Therefore, this procedure should be implemented with these points in mind, because no alternative can be used if curved facets are present in the scenario.

The SIP strategy is more complex and difficult to implement than the SBR because it solves inverse problems. However, the SIP is superior to the SBR for problems that require the illumination phase, diffractions, and high-order coupling. These requirements occur in many applications of

RTT for radio-wave propagation and other electromagnetic analyses, wherein the illumination has a coherent nature and therefore the phase plays a fundamental role. The SIP strategy permits a precise computation of all ray parameters for rays that arrive at the observation point. The application of the SBR strategy is easy but less accurate than the SIP strategy.

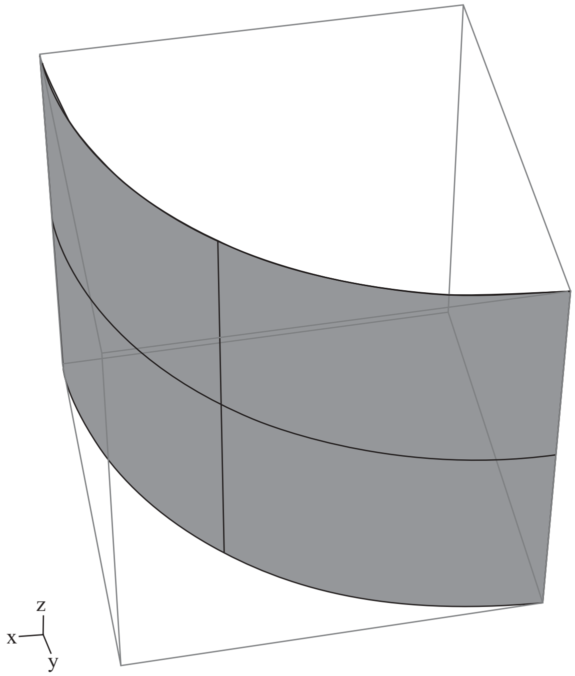


Figure 7. Example of the bounding box that encloses a curved surface.

ALGORITHMS TO REDUCE THE COMPUTATIONAL COST ASSOCIATED TO THE RSQ

The computational costs of RSQ have directed research to develop many powerful algorithms, some of which will be summarized here. It is important to note that the aim of these algorithms is to only reduce the cost associated to the second sub-problem described in the section on the Problems and complexity of RTT. These algorithms only address the shadowing problem (which appears in both SBR and SIP) and their application is the same, independent of the ray mechanism. The searching of the ray path associated to these mechanisms is not improved by these algorithmic techniques.

RSQ algorithms are usually applied in a preprocessing phase and in an execution phase (4). In the preprocessing phase, the input data structure that contains the information of the entities (usually facets) that define the geometry is reorganized. This reorganization groups close and/or visible geometrical entities together in the data structure, and/or relations of proximity or visibility are associated to the appropriate entities.

The simplest RSQ algorithm encloses completely each entity of the scene in a bounding box, preferably in a parallelepiped of minimum volume, with sides perpendicular to the Cartesian absolute system chosen to represent the geometry (see Fig. 7). The bounding box RSQ is compatible with other RSQ algorithms. It is performed as follows: Before checking the intersection of the ray with the curved facet, by means of a rigorous checking procedure, it is determined if a ray impacts the box. Obviously, the ray cannot be occluded by the surface if the check is negative. This surface is not considered in the shadowing test, which avoids more time-consuming procedures for checking the possible intersection of the ray with the facet.

Most RSQ algorithms are based on a volumetric spatial partition (VSP) of the scene (17). After this division, the scene is split into small subvolumes that contain groups of scene entities, with some subvolumes potentially empty. To determine whether a segment is occluded by one or more scene entities, we only need to check the entities contained in the subvolumes that the segment crosses. Figure 8 shows an example of the division of a simple two-dimensional (2-D) scene in voxels. An example of the ray tracing between a source (S) and an observation point (O) is depicted in the figure. It can be noted that only the entities contained in the voxels that the ray crosses (facets 3, 4, 9, 10, 11) are involved in the RSQ, which reduces the complexity of this task.

Many efficient RSQ algorithms are based on VSP. The major difference between these algorithms is the strategy used for space division, which can significantly impact final algorithm efficiency. One of those algorithms is binary space partitioning (BSP). BSP requires the space to be divided into subvolumes in successive steps such that each subvolume is split in two halves (binary partitioning). The BSP has two versions. In the polygon-aligned version, the space subdivision in two halves is made considering the regions above and below each flat facet of the scene (17). This version is not applicable when the scene has curved surfaces. In the axis-aligned version, a division into two halves of a subvolume is made by a plane perpendicular to

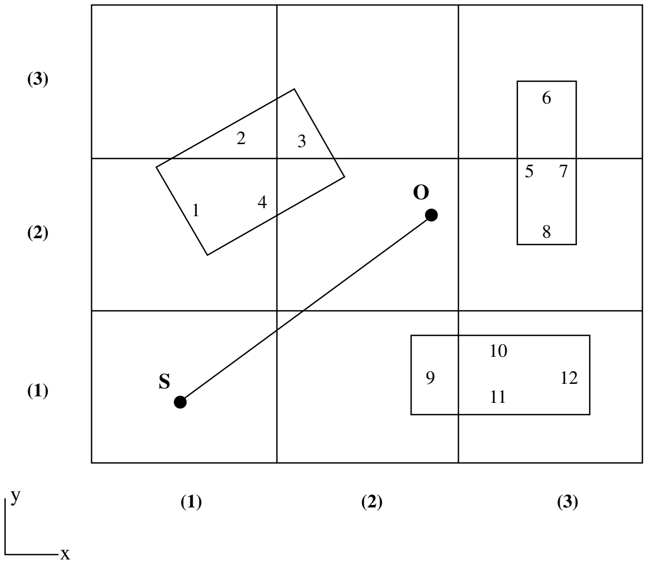


Figure 8. Example of the application of the SVP for a simple 2-D scene. The scene is divided in voxels.

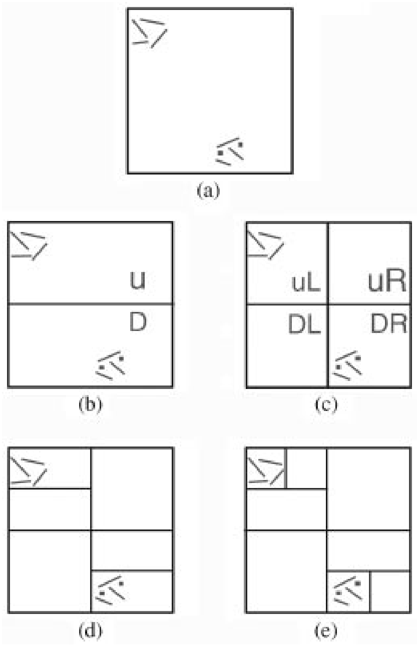


Figure 9. Example of spatial partition of and scene using the BSP axis-aligned technique. The scene has only two groups of entities, which are indicated by the small straight lines in the top and in the bottom of the scene rectangle.

the Cartesian coordinate system that defines the scene (18). Figure 9 shows the application of the axis-aligned version of BSP to a simple 2-D scene, and Fig. 10 presents the associated BSP tree. The subdivision in a level tree is perpendicular to a coordinate axis; the axes alternate in consecutive tree levels. Each node of the BSP tree corresponds to a division of a subvolume. When a subvolume is empty, it is not subdivided, and its tree node becomes a tree leaf. This tree is very helpful in the RSQ because, to determine the intersection of a ray segment with one of the entities of the scene, only the entities that belong to the low-level subvolumes of the BSP tree that the segment crosses. For example, in Fig. 10, if a segment joins two solid points in the subvolume of node DRDL when applying the BSP, only the entities in the subvolume DRLD must be checked, obviously reducing considerably the number of intersection tests to be performed.

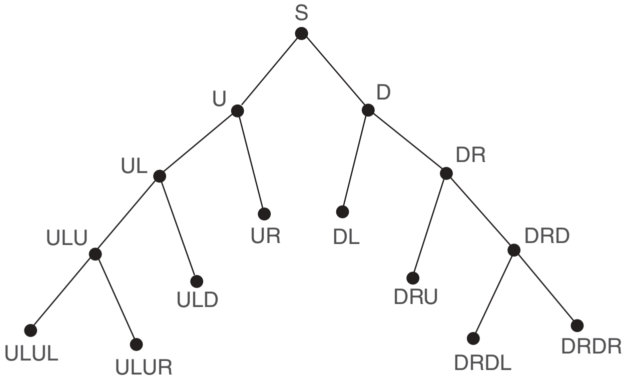


Figure 10. BSP tree for the case of Fig. 6.

Today, one of the most efficient algorithms for solving the RSQ is the *kd*-tree algorithm (19), which is similar to the axis-aligned version of the BSP. The difference between these algorithms is that, during the preprocessing phase of the *kd*-tree, the subvolumes in each binary division do not necessarily have the same size. The size of the subvolumes is fitted to give the bigger size to the empty or less populated subvolume.

Octree Space subdivision is another technique for solving the RSQ that is similar to the BSP, except that now each subvolume is split into eight small subvolumes (called octrees) by the three coordinate planes (20). Other variants of the VSP are the division of the scene volume into a uniform grid of equal-sized sub-volumes (21). Simultaneously, each subvolume can be split into a new grid of uniform sub-volumes. This process can be repeated several times to form hierarchical grids (22).

THE ANGULAR ZBUFFER ALGORITHM (AZB)

The AZB algorithm deserves special attention because it possesses features markedly different compared with the techniques presented in the previous section. The main advantage of this algorithm is that, not only can it be used to solve the RSQ problem (second subproblem of the RTT), but also it is especially applicable to the SIP strategy to solve the FPS (first subproblem) of the RTT. The AZB algorithm is based on the light buffer technique (23) used in computer graphic design for solving problems of hidden lines or surfaces to determine which areas of a given environment are visible from a point in space (the source).

The algorithm is based on the division of the space observed from the source point or from the facets in spherical sectors. These spherical sectors are called anxels, and they are defined by the spherical coordinates θ and ϕ with respect to the coordinate system defined by the source or by the facets (see Fig. 11) (17). The number of anxels depends of the angular increments Δθ and Δϕ) that define the size of the anxel:

$$\begin{aligned} N_\theta &= 180/\Delta\theta; N_\phi = 360/\Delta\phi \\ & N_{\text{anxels}} = N_\theta * N_\phi \end{aligned}$$

The coordinates θ and ϕ define a plane called the AZB plane that contains the objects located in each anxel, and it identifies the directions where the object is located, taking the source point or each facet as reference. The AZB plane of a point (like the source) is easier to obtain than the AZB plane of a facet.

The 2-D scene with planar facets depicted in Fig. 12 illustrates AZB of the source point. S marks the position of the source. If the space is divided into eight angular regions, each region will contain the facets given in Table 1 that contain the information of the so-called AZB matrix associated with the source point. For example, in anxel 2 (angular margin from 0° to 45°), there are only two facets. Then, to determine whether a direct ray reaches an observation

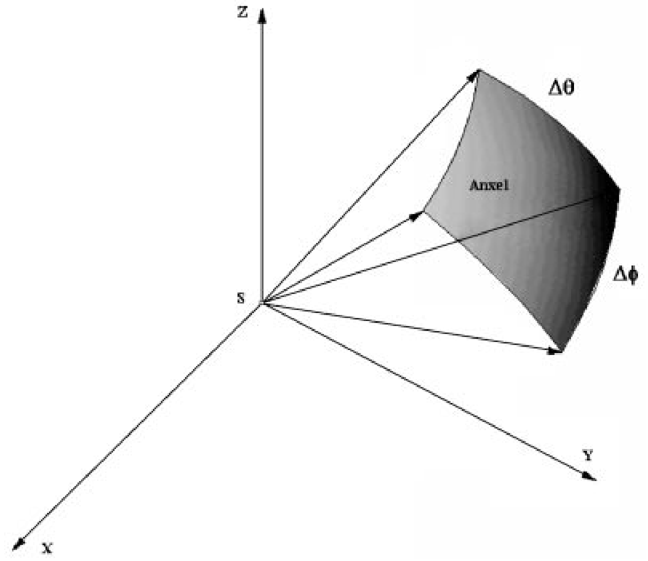


Figure 11. Anxel definition as a function of the angular steps Δθ and Δϕ.

Table 1. Facets Stored for Each Anxel of the Example in Fig. 12

Anxel	Facets
1	9, 10
2	10
3	1, 4, 6
4	6, 8
5	
6	
7	13
8	13, 14

point placed in anxel 1, the RSQ must only be performed for the two facets contained in that anxel, instead of for the 16 facets that appear in the scene. The facets that do not satisfy the criterion of the back face culling (5) cannot hide a ray traced from the source to a given observer, and they are not stored in the AZB matrix. For this reason, facets 11 and 12 have not been considered in the table for anxel 1.

This idea can be generalized easily to a three-dimensional (3-D) case. If a model composed of planar facets is considered, the spherical coordinates of the facet vertices are calculated, which define a window based on the minimum and maximum θ and ϕ coordinates ($\theta_{\min}$, $\theta_{\max}$, $\phi_{\min}$, $\phi_{\max}$) used to determine the anxel where the facet must be stored. If the scene contains curved surfaces, then the computation of this window becomes more complex. A good solution to this problem consists of enclosing the surface within a bounding box. Once the surface is enclosed, the eight vertices of the box can be considered for the creation of the AZB matrix by considering the spherical coordinates of the eight vertices as a whole, in a similar way to the vertices of a facet for the case of planar surfaces (see Fig. 13). All information that corresponds to the planar and the curved surfaces is stored in the AZB matrix, which can be considered as a grid in the coordinates θ and ϕ, as depicted in Fig. 14. Each vertex of an entity is placed in an element of the grid, and the entity will be contained in the set of elements of the grid defined by its vertices.

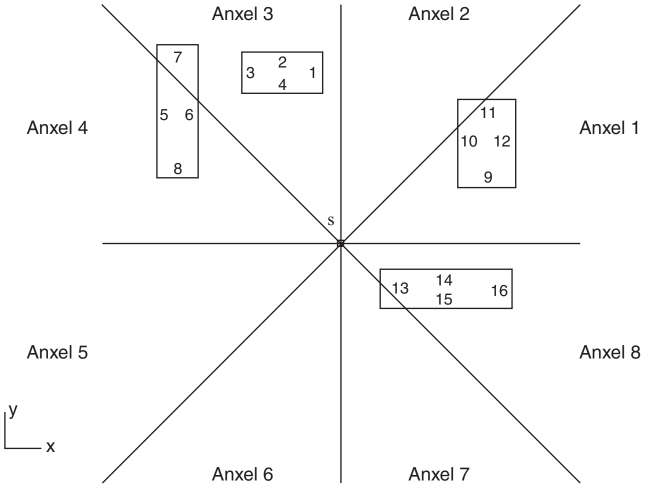


Figure 12. Example of 2-D scenario and additional division in angular sectors.

The application of the aforementioned algorithm is only valid for the analysis of direct ray shadowing because the source has been taken as a reference. Some modifications must be performed to apply this algorithm to reflected and diffracted rays. In reflection, the application of the algorithm is very easy for the planar entities of the scenario, and subsequently image method for reflection can be applied. Therefore, an AZB matrix can be built for every image in a given scenario. The only difference is that the space to be divided in this case, is only the region where the reflected ray can appear. The so-called reflection window defines this region (see Fig. 15).

The definition of the reflection window is more complicated for a curved surface. In this case, all possible directions of reflection have to be considered with respect to all points on the surface. To avoid this, an AZB matrix is

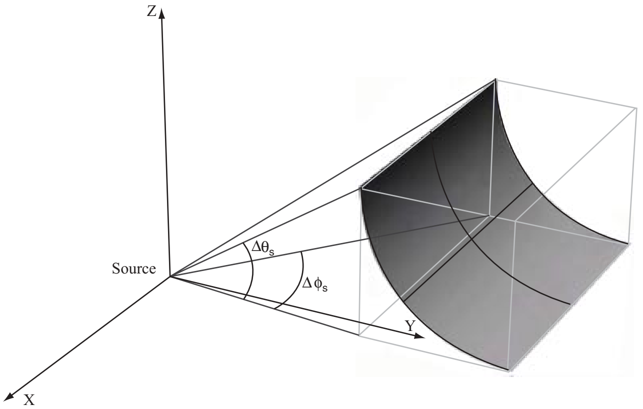


Figure 13. Angular margins of a surface considering the enclosing box.

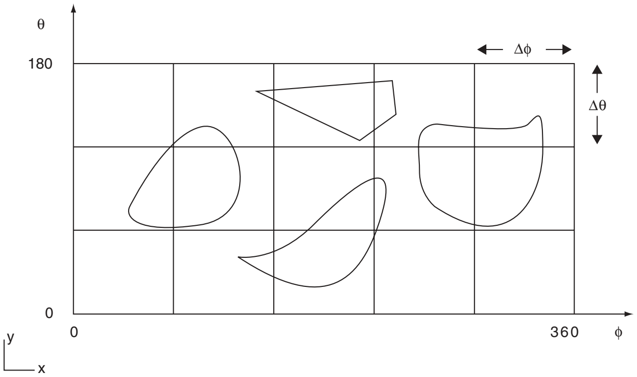


Figure 14. Example of the storage of different surfaces in a grid defined by the AZB matrix.

created for each of the eight vertices that define the box that encloses the surface. The angular margin for the total reflection window is obtained as the union of the regions defined by each of the vertices of the box (see Fig. 16). More details about the complete procedure to create this window can be found in Reference 24.

In the case of a diffracted ray, a diffraction window must be created from the properties of this effect. When a ray arrives at an edge, infinite rays are diffracted, forming a cone whose vertex is the diffraction point, the axis is the edge itself, and the angle is the same as that formed between the edge and the incident ray. This cone is called "Keller's cone" (14). The region contained within this cone defines the diffraction window. New angular coordinates must be considered to limit this diffraction window according to these properties. This coordinates are α and β, defined in Fig. 17. Therefore, analogous to the reflection case, there is a diffracted window for every illuminated edge that corresponds to an AZB diffraction matrix.

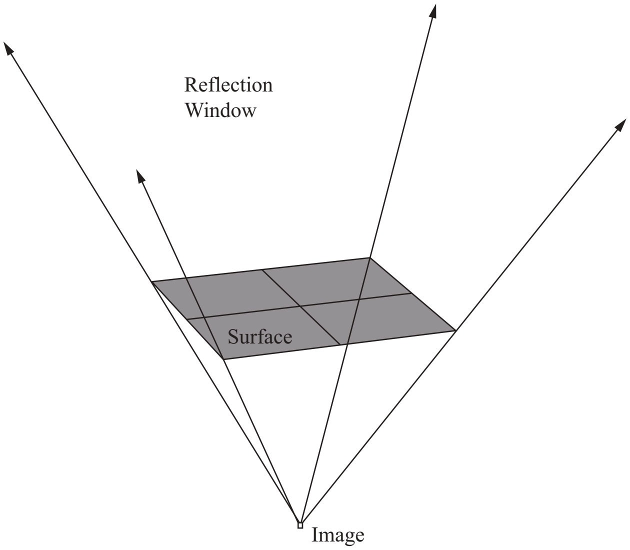


Figure 15. Reflection window for a planar surface.

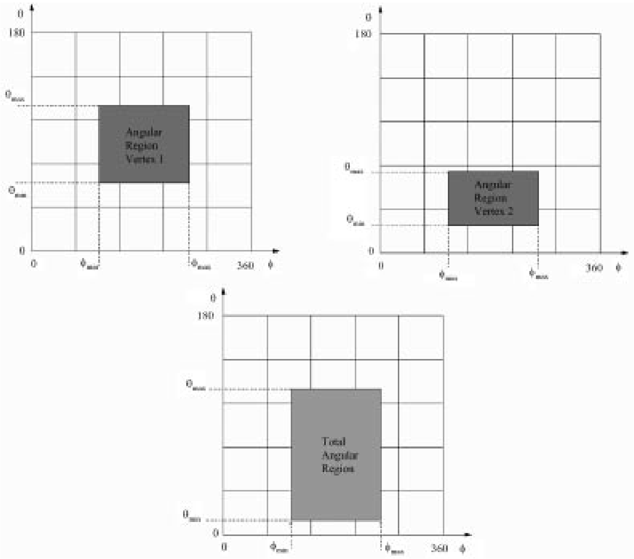


Figure 16. Generation of the total angular margin of a surface from the angular margin of two points. The procedure can be generalized to the eight vertices of the enclosing parallelepiped.

COMPARISON BETWEEN THE DIFFERENT ACCELERATION RTT

In this section, a comparison between the features of different RTT is outlined. The idea is to help the reader select the most suitable technique for their application. This comparison is performed considering only the features related to the RSQ, because only the AZB technique has been presented to accelerate the FPS.

First of all, it can be stated that the BSP technique is efficient for 2-D scenes and making the BSP tree is relatively easy. Unfortunately, for 3-D scenarios, this task is more complex because finding an optimum tree is difficult. Often, these trees are long and with many broken facets, and they are not useful for efficient analysis.

The VSP, when applied to large scenes, requires many voxels to load a low amount of facets per voxel. Furthermore, when the source is far away from the observation point, the number of voxels that cross the ray is high and,

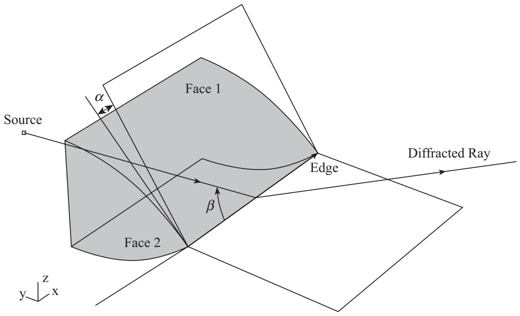


Figure 17. Coordinates α and β that define the AZB window for diffraction.

therefore, the amount of facets that are considered in the intersection test is enormous. Fortunately, VSP has the advantage of a low memory requirement, since it only needs a matrix that depends only on the scene. In contrast, the AZB technique generates a matrix depending on the source (antenna, image, edge, etc.). For higher-order effects, and when the number of sources is large and the number of observation points involved in the effect is low, it is not efficient to create the AZB matrices. In these cases, the SVP method can be combined with the AZB.

Finally, the AZB technique has the advantage, with respect to the other methods, of only loading the facets not shadowed by illumination or eclipse (those that satisfy the back-face culling criterion as explained in the previous section), because the space is divided taking the source as a reference. This method allows a reduction in the number of facets stored in each matrix.

CONCLUSIONS

A short summary of RTT has been presented. The complexity of the ray tracing has been discussed in the context of its varied applications to different kinds of scenarios. Because of this complexity, efficient techniques are necessary for implementing RTT, and most of this article has been dedicated to describing the most commonly used among these techniques. Finally, ray-tracing acceleration techniques have been presented, with special attention dedicated to AZB because of its features that permit the reduction of the computational cost associated with the two subproblems related to the RTT. Readers interested in additional investigation of this subject are encouraged to read the cited works as well as References 25–27.

BIBLIOGRAPHY

1. A. S. Glassner (ed.), *An Introduction to Ray Tracing*, San Diego, CA: Academic Press, 1989.
2. J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics. Principles and Practice*, 2nd ed., New York: Addison-Wesley, 1995.
3. A. Watt and M. Watt, *Advanced Animation and Rendering Techniques*, Reading, MA: ACM-PRESS, Addison-Wesley, 1992.
4. V. Havran, Heuristic Ray Shooting Algorithms, Dissertation Thesis, Prague, November 2000.
5. M. F. Catedra and J. P. Arriaga, *Cell Planning for Wireless Communications*, Norwood, MA: Artech House, 1999.
6. D. J. Maguire, M. F. Goodchild, and D. W. Rhind, *Geographical Information Systems*, Essex, UK: Longman Scientific & Technical, 1991.
7. L. Piegl and W. Tiller, *The NURBS Book, Monographs in Visual Communication*, 2nd ed., New York: Springer-Verlag, 1997.
8. C. de Boor, *A Practical Guide to Splines*, Berlin: Springer, 1978.
9. G. E. Farin, *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide*, 2nd edtion, London: Academic Press Inc., 1990.
10. A. Efremov, V. Havran, and H. P. Seidel, Robust and numerically stable Bezier clipping method for ray tracing NURBS surfaces, *Proc. of the 21st Spring Conference on Computer Graphics SCCG '05*, 2005, pp. 127–135.
11. S. W. Lee, H. Ling, and R. Chou, Ray-tube integration in shooting and bouncing ray method, *Microwave Opt. Technol. Lett.*, **1** (8): 286–289, 1988.
12. S. Y. Seidel and T. S. Rappaport, Site-specific propagation prediction for wireless in building personal communication system design. *IEEE Trans. Vehicu. Technolo.* **43** (4): 879–891, 1994.
13. J. W. McKown and R. L. Hamilton, Ray tracing as a design tool for radio networks, *IEEE Network Magazine*, November 1991, pp. 27–30.
14. C. A. Balanis, *Advanced Engineering Electgromagnetics*, New York: John Wiley & Sons, 1989.
15. W. Honcharenko, H. L. Bertoni, J. L. Dailing, J. Quian, and H. D. Yee, Mechanism governing UHF propagation on single floors in modern office buildings, *IEEE Trans. Vehic. Techno.* **41** (4): 496–504, 1992.
16. I. González, C. Delgado, F. Saéz de Adana, O. Gutiérrez, and M. F. Cátedra, A new 3D ray-tracing acceleration technique for the analysis of propagation and radiation in complex environments, *Applied Computat. Electromag. Soc. J.*, July 2007, pp. 201–206.
17. M. F. Cátedra, J. Pérez, F. Saez de Adana, and O. Gutiérrez, Efficient ray-tracing techniques for three-dimensional analyses of propagation in mobile communications: application to picocell and microcell scenarios, *IEEE Ante. Propag. Mag.*, **40** (2): 15–28, 1998.
18. M. Kaplan. Space-tracing: a constant time ray-tracer. Course notes from tutorial State of the art in image synthesis, *SIGGRAPH 85*, 1985, pp. 173–193.
19. J. D. MacDonald and K. S. Booth, Heuristics for ray tracing using space subdivision, *Vis. Comput.*, **6** (6): 153–65, 1990.
20. A. S. Glassner, Space subdivision for fast ray tracing, *IEEE Comp. Graph. Applicat.*, **4** (10): 15–22, 1984.
21. A. Fujimoto, C. G. Perrott, and K. Iwata, ARTS: accelerated ray-tracing system, *IEEE Comp. Graph. Applicat.* **6** (4): 16–26, 1986.
22. K. S. Klimaszewski and T. W. Sederberg, Faster ray tracing using adaptive grids, *IEEE Comp. Graph. Applicat.*, **17** (1): 42–51, 1997.
23. E. A. Hines and D. P. Greenberg, The light buffer: a shadow-testing accelerator, *IEEE Comput. Graph. Applicat.*, **6** (9): 6–16, 1986.
24. I. González, C. Delgado, F. Saez de Adana, O. Gutiérrez, and M. F. Cátedra. A New 3D Ray-Tracing Acceleration Technique for the Analysis of Propagation and Radiation in Complex Environments, *Appl. Computat. Electromag. Soc. J.*, **22** (2): 201–206, 2007.
25. Available: http://objects.povworld.org/3dlinks.html.
26. Available: http://www.siggraph.org.
27. Available: http://www.vision-systems.com.

M. Felipe Cátedra
Iván González
Francisco Saez de Adana
Oscar Gutiérrez
Lorena Lozano
Universidad de Alcalá
Alcalá de Henares, Spain

SCALE-SPACE

THE NEED FOR MULTI-SCALE REPRESENTATION OF IMAGE DATA

An inherent property of real-world objects is that only they exist as meaningful entities over certain ranges of scale. A simple example is the concept of a branch of a tree, which makes sense only at a scale from, say, a few centimeters to at most a few meters; it is meaningless to discuss the tree concept at the nanometer or kilometer level. At those scales, it is more relevant to talk about the molecules that form the leaves of the tree or the forest in which the tree grows. When observing such real-world objects with a camera or an eye, an addition scale problem exists because of perspective effects. A nearby object will seem larger in the image space than a distant object, although the two objects may have the same size in the world. These facts, that objects in the world appear in different ways depending on the scale of observation and in addition may undergo scale changes during an imaging process, have important implications if one aims to describe them. It shows that the notion of *scale* is fundamental to understand both natural and artificial perception.

In computer vision and image analysis, the notion of scale is essential to design methods for deriving information from images and multidimensional signals. To extract any information from image data, obviously one must interact with the data in some way, using some operator or measurement probe. The type of information that can be obtained is largely determined by the relationship between the size of the actual structures in the data and the size (resolution) of the operators (probes). Some very fundamental problems in computer vision and image processing concern *what* operators to use, *where* to apply them, and *how large* they should be. If these problems are not addressed appropriately, then the task of interpreting the operator response can be very hard. Notably, the scale information required to view the image data at an appropriate scale may in many cases be a priori unknown.

The idea behind a scale-space representation of image data is that in the absence of any prior information about what scales are appropriate for a given visual task, the only reasonable approach is to represent the data at multiple scales. Taken to the limit, a scale-space representation furthermore considers representations at *all* scales simultaneously. Thus, given any input image, this image is embedded into a one-parameter family of derived signals, in which fine-scale structures are progressively suppressed. When constructing such a multiscale representation, a crucial requirement is that the coarse-scale representations should constitute simplifications of corresponding structures at finer scales—they should not be accidental phenomena created by the smoothing method intended to suppress fine scale structures. This idea has been formalized in a variety of ways by different authors, and a noteworthy coincidence is that similar conclusions can be obtained from several different starting points. A fundamental result of scale-space theory is that if general conditions are imposed on the types of computations that are to be performed in the earliest stages of visual processing, then convolution by the Gaussian kernel and its derivatives provide a canonical class of image operators with unique properties. The requirements (scale-space axioms; see below) that specify the uniqueness essentially are linearity and spatial shift invariance, combined with different ways of formalizing the notion that "new structures should not be created" in the transformation from fine to coarse scales.

In summary, for any two-dimensional (2-D) signal $f : \mathbb{R}^2 \to \mathbb{R}$, its *scale-space representation* $L : \mathbb{R}^2 \times \mathbb{R}_+ \to \mathbb{R}$ is defined by (1–6)

$$L(x,y;\, t) = \int_{(\xi,\eta)\in \mathbb{R}^2} f(x-\xi, y-\eta)\, g(\xi,\eta;\, t)\, d\xi\, d\eta \qquad (1)$$

where $g : \mathbb{R}^2 \times \mathbb{R}_+ \to \mathbb{R}$ denotes the Gaussian kernel

$$g(x,y;\, t) = \frac{1}{2\pi t} e^{-(x^2+y^2)/2t} \qquad (2)$$

and the variance $t = \sigma^2$ of this kernel is referred to as the *scale parameter*. Equivalently, the scale-space family can be obtained as the solution of the (linear) diffusion equation

$$\partial_t L = \frac{1}{2}\nabla^2 L \qquad (3)$$

with initial condition $L(\cdot,\cdot;\, t) = f$. Then, based on this representation, *scale-space derivatives* at any scale t can be computed either by differentiating the scale-space directly or by convolving the original image with Gaussian derivative kernels:

$$L_{x^\alpha y^\beta}(\cdot,\cdot;\, t) = \partial_{x^\alpha y^\beta} L(\cdot,\cdot;\, t) = (\partial_{x^\alpha y^\beta} g(\cdot,\cdot;\, t)) * f(\cdot,\cdot) \qquad (4)$$

Because scale-space derivatives can also be computed by convolving the original image with Gaussian derivative operators $g_{x^\alpha y^\beta}(\cdot,\cdot;\, t)$ they are also referred to as *Gaussian derivatives*. This way of defining derivatives in scale-space makes the inherently ill-posed problem of computing image derivatives well-posed, with a close connection to generalized functions.

For simplicity, we shall here restrict ourselves to 2-D images. With appropriate generalizations or restrictions, however, most of these concepts apply in arbitrary dimensions.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

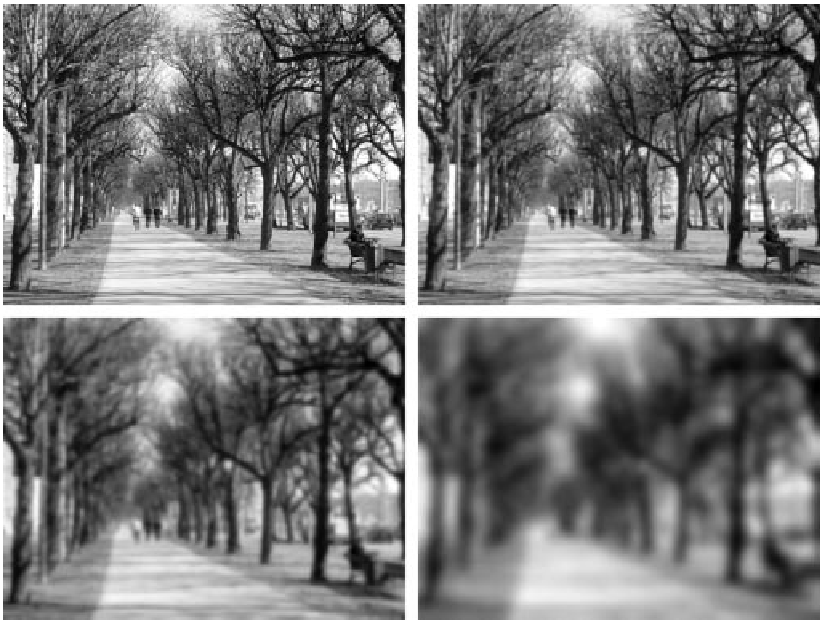


Figure 1. (top left) A gray-level image of size 560×420 pixels, (top right)−(bottom right) Scale-space representations computed at scale levels $t = 1$, 8, and 64 (in pixel units).

FEATURE DETECTION AT A GIVEN SCALE IN SCALE-SPACE

The set of scale-space derivatives up to order N at a given image point and a given scale is referred to as the *N-jet* (7,8) and corresponds to a truncated Taylor expansion of a locally smoothed image patch. Together, these derivatives constitute a basic type of feature within the scale-space framework and provide a compact characterization of the local image structure around the image point at that scale. For $N = 2$, the 2-jet at a single scale contains the partial derivatives

$$(L_x, L_y, L_{xx}, L_{xy}, L_{yy}) \tag{5}$$

and directional filters in any direction $(\cos\varphi, \sin\varphi)$ can be obtained from

$$\begin{aligned} \partial_\varphi L &= \cos\varphi\, L_x + \sin\varphi\, L_y \quad \text{and} \\ \partial_{\varphi\varphi} L &= \cos^2\varphi L_{xx} + 2\cos\varphi\sin\varphi\, L_{xy} + \sin^2\varphi\, L_{yy} \end{aligned} \tag{6}$$

From the five components in the 2-jet, four differential invariants can be constructed, which are invariant to local rotations: the gradient magnitude $|\nabla L|$, the Laplacian $\nabla^2 L$, the determinant of the Hessian det $\mathcal{H}L$, and the rescaled level curve curvature $\tilde{\kappa}(L)$:

$$\begin{cases} |\nabla L|^2 = L_x^2 + L_{yx}^2 \\ \nabla^2 L - L_{xx} + L_{yyl} \\ det\,\mathcal{H}L = L_{xx}L_{yy} - L_{xy}^2 \\ \tilde{\kappa}(L) = L_x^2 L_{yy} + L_y^2 L_{xx} - 2L_x L_y L_{xy} \end{cases} \tag{7}$$

A theoretically well-founded approach to feature detection is to use rotationally variant descriptors such as the N-jet, directional filter banks, or rotationally invariant differential invariants as primitives for expressing visual modules. For example, with v denoting the gradient direction $(L_x, L_y)^T$ a differential geometric formulation of *edge detection* at a given scale can be expressed from the image points for which the second-order directional derivative in the gradient direction L_{vv} is zero and the third-order directional derivative L_{vvv} is negative:

$$\begin{cases} \tilde{L}_{vv} = L_x^2\, L_{xx} + 2\, L_x\, L_y\, L_{xy} + L_y^2\, L_{yy} = 0, \\ \tilde{L}_{vvv} = L_x^3\, L_{xxx} + 3\, L_x^2\, L_y\, L_{xxy} + 3\, L_x\, L_y^2\, L_{xyy} + L_y^3\, L_{yyy} < 0 \end{cases} \tag{8}$$

A single-scale *blob detector* that responds to bright and dark blobs can be expressed from the minima and the maxima of the Laplacian response $\nabla^2 L$. An affine covariant blob detector that also responds to saddles can be expressed from the maxima and the minima of the determinant of the Hessian det $\mathcal{H}L$. A straight forward and affine covariant *corner detector* can be expressed from the maxima and minima of the rescaled level curve curvature $\tilde{\kappa}(L)$. With p denoting the main eigendirection of the Hessian matrix, which is parallel to the vector

$$(\cos\varphi, \sin\varphi)^{\sim} \left(\sqrt{1 + \frac{L_{xx} - L_{yy}}{\sqrt{(L_{xx} - L_{yy})^2 + 4L_{xy}^2}}}, \right. \\ \left. \operatorname{sign}(L_{xy}) \sqrt{1 - \frac{L_{xx} - L_{yy}}{\sqrt{(L_{xx} - L_{yy})^2 + 4L_{xy}^2}}} \right) \tag{9}$$

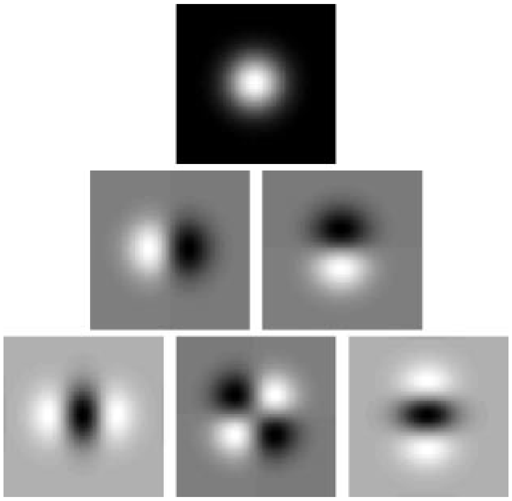


Figure 2. The Gaussian kernel and its derivatives up to order two in the 2-D case.

a differential geometric *ridge detector*at a fixed scale can be expressed from the zero-crossings of the first derivative L_p in this direction for which the second-order directional derivative L_{pp} is negative and in addition $|L_{pp}| \geq |L_{qq}|$. Similarly, valleys can be extracted from the zero-crossings of L_q that satisfy $L_{qq} \geq 0$ and $|L_{qq}| \geq |L_{pp}|$.

FEATURE-CLASSIFICATION AND IMAGE MATCHING FROM THE MULTISCALE N-IET

By combining N-jet representations at multiple scales, usually with the scale levels distributed by ratios of two when the scale parameter is measured in units of the standard deviation $\sigma = \sqrt{t}$ of the Gaussian, we obtain a *multiscale N-jet vector*. This descriptor is useful for a variety of different tasks. For example, the task of *texture classification* can be expressed as a classification and/or clustering problem on the multi-scale N-jet over regions in the image (9). Methods for *stereo matching* can be formulated in terms of comparisons of local N-jets (10), either in terms of explicit search or coarse-to-fine schemes based on differential corrections within the support region of the multi-scale N-jet. Moreover, straightforward (rotationally dependent) methods for *image-based object recognition* can expressed in terms of vectors or (global or regional) histograms of multi-scale N-jets (11–14). Methods for rotationally invariant image-based recognition can formulated in terms of histograms of multiscale vectors of differential invariants.

WINDOWED IMAGE DESCRIPTORS WITH TWO SCALE PARAMETERS

The image descriptors considered so far are all dependent on a single scale parameter t. For certain problems, it is useful to introduce image descriptors that depend on two scale parameters. One such descriptor is the *second-moment matrix* (structure tensor) defined as

$$\begin{aligned}\mu(x,y;\,t,s) &= \int_{(\xi,\eta)\in \boldsymbol{R}^2} \begin{pmatrix} L_x^2(\xi,\eta;\,t) & L_x(\xi,\eta;\,t)L_y(\xi,\eta;\,t) \\ L_x(\xi,\eta;\,t)L_y(\xi,\eta;\,t) & L_y^2(\xi,\eta;\,t)\end{pmatrix} \\ &\quad \times g(x-\xi,y-\eta;\,s)\,d\xi\,d\eta \end{aligned} \tag{10}$$

where t is a local scale parameter that describes the scale of differentiation, and s is an integration scale parameter that describes the extent over which local statistics of derivatives is accumulated. In principle, the formulation of this descriptor implies a two-parameter variation. In many practical applications, however, it is common practice to couple the two scale parameters by a constant factor C such that $s = Ct$ where $C > 1$.

One common application of this descriptor is for a multiscale version of the *Harris corner detector*(15), by detecting positive spatial maxima of the entity

$$H = det(\mu) - k\,\mathrm{trace}^2(\mu) \tag{11}$$

where $k \approx 0.04$ is a constant. Another common application is for affine normalization/shape adaptation and will be developed below. The eigenvalues $\lambda_{1,2} = \mu_{11} + \mu_{22} \pm \sqrt{(\mu_{11} - \mu_{22})^2 + 4\mu_{12}^2}$ and the orientation $\arg(\mu_{11} -$

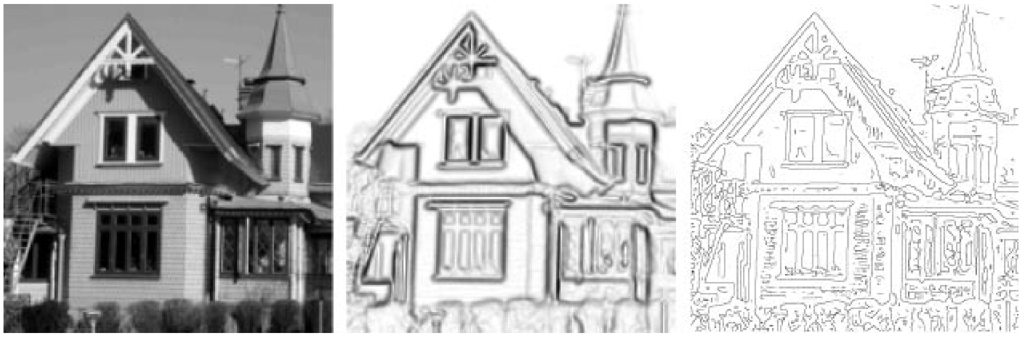


Figure 3. *Edge detection*: (left) A gray-level image of size 180 × 180 pixels, (middle) The negative value of the gradient magnitude $|\nabla L|$ computed at $t = 1$. (right) Differential geometric edges at $t = 1$ with a complementary low threshold $\sqrt{t}|\nabla L| \geq 1$ on the gradient magnitude.

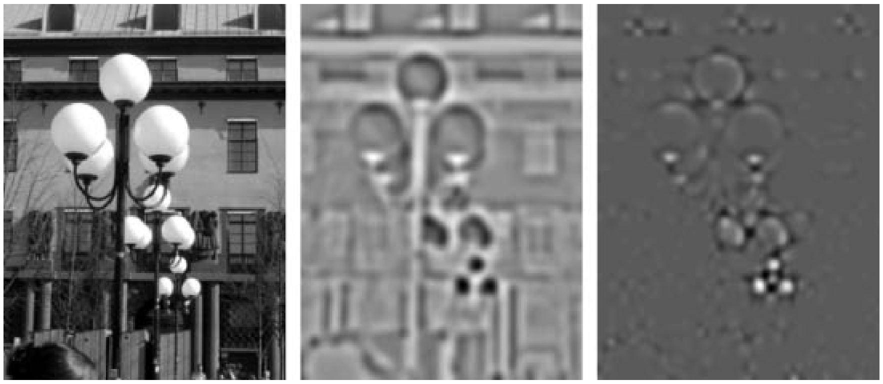


Figure 4. *Differential descriptors for blob detection / interest point detection*: (left) A gray level image of size 210 × 280 pixels, (middle) The Laplacian $\nabla^2 L$ computed at $t = 16$. (right) The determinant of the Hessian det $\mathcal{H}L$ computed at $t = 16$.

$\mu_{22}, 2\mu_{12}$) of μ are also useful for texture segmentation and for *texture classification*.

Other commonly used image descriptors that depend an additional integration scale parameter include regional histograms obtained using Gaussian window functions as weights (16).

SCALE-SPACE REPRESENTATION OF COLOR IMAGES

The input images *f* considered so far have all been assumed to be scalar gray-level images. For a vision system, however, color images are often available, and the use of color cues can increase the robustness and discriminatory power of image operators. For the purpose of scale-space representation of color images, an initial red green blue-yellow color opponent transformation is often advantageous (17). Although the combination of a scale-space representation of the RGB channels independently does not necessarily constitute the best way to define a color scale-space, by performing the following pretransformation prior to scale-space smoothing

$$\begin{cases} I = (R+G+B)/3 \\ U = R - G \\ V = B - (R+G)/2 \end{cases} \tag{12}$$

Gaussian scale-space smoothing, Gaussian derivatives, and differential invariants can then be defined from the IUV color channels separately. The luminance channel I will then reflect mainly, the interaction between reflectance, illuminance direction, and intensity, whereas the chromatic channels, $U$and V, will largely make the interaction between illumination color and surface pigmentation more explicit. This approach has been applied successfully to the tasks of image feature detection and image-based recognition based on the N-jet. For example, red-green and blue-yellow color opponent receptive fields of center-surround type can be obtained by applying the Laplacian $\nabla^2 L$ to the chromatic U and V channels.

An alternative approach to handling colors in scale-space is provided by the *Gaussian color model*, in which the spectral energy distribution $E(\lambda)$ over all wavelengths λ is approximated by the sum of a Gaussian function and first- and second-order derivatives of Gaussians (18). In this way a 3-D color space is obtained, where the channels $\hat{E}, \hat{E}_{\lambda}$ and $\hat{E}_{\lambda\lambda}$ correspond to a second-order Taylor expansion of a Gaussian-weighted spectral energy distribution around a specific wavelength λ_0 and smoothed to a fixed spectral scale t_{λ_0}. In practice, this model can be implemented in different ways. For example, with $\lambda_0 = 520$ nm and $t_{\lambda_0} = 55$ nm,

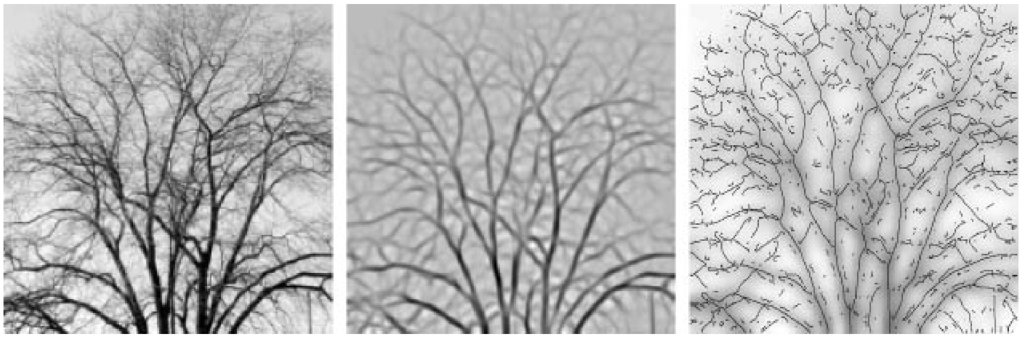


Figure 5. *Fixed scale valley detection*: (left) A gray-level image of size 180 × 180 pixels, (middle) The negative value of the valley strength measure L_{qq} computed at $t = 4$. (right) Differential geometric valleys detected at $t = 4$ using a complementary low threshold on $t|L_{qq}| \geq 1$ and then overlayed on a bright copy of the original gray-level image.

the Gaussian color model has been approximated by the following color space transformation (19):

$$\begin{pmatrix} \hat{E} \\ \hat{E}_\lambda \\ \hat{E}_{\lambda\lambda} \end{pmatrix} = \begin{pmatrix} -0.019 & 0.048 & 0.011 \\ 0.019 & 0 & -0.016 \\ 0.047 & -0.052 & 0 \end{pmatrix} \begin{pmatrix} 0.621 & 0.113 & 0.194 \\ 0.297 & 0.563 & 0.049 \\ -0.009 & 0.027 & 1.105 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad (13)$$

using the CIE 1931 XYZ color basis as an intermediate representation. In analogy with the IUV color space, spatio-spectral Gaussian derivatives and differential invariants can then be defined by applying Gaussian smoothing and Gaussian derivatives to the channels in this representation. This approach has been applied for constructing approximations of color invariants assuming specific models for illumination and/or surface properties (20).

AFFINE SCALE-SPACE, AFFINE IMAGE DEFORMATIONS, AND AFFINE COVARIANCE

The regular linear scale-space representation obtained by smoothing with the rotationally symmetric Gaussian kernel is closed under translations, rotations, and rescalings. This property means that image transformations within this group can be captured perfectly by regular scale-space operators. To obtain a scale-space representation that is closed under affine transformations, a natural generalization is to consider an *affine scale-space*(3) obtained by the convolution with Gaussian kernels with their shapes determined by positive definite covariance matrices Σ:

$$g(\bar{x};\, \Sigma_t) = \frac{1}{2\pi\sqrt{det\,\Sigma_t}} e^{-\bar{x}^T \Sigma_t^{-1} \bar{x}/2} \qquad (14)$$

where $\bar{x} = (x,y)^T$. This affine scale-space combined with directional derivatives can serve as a model for *oriented elongated filter banks*. Consider two input images f and f' that are related by an affine transformation $\bar{x}' = A\bar{x}$ such that $f'(A\bar{x}) = f(\bar{x})$. Then, the affine scale-space representations L and L' of f and f' are related according to $L'(x';\, \Sigma') = L(x;\, \Sigma)$ where $\Sigma' = A\Sigma A^T$. A second-moment matrix [Equation 10] defined from an affine scale-space with covariance matrices Σ_t and Σ_s transforms according to

$$\mu'(A\bar{x};\, A\Sigma_t A^T, A\Sigma_s A^T) = A^{-T}\mu(x;\, \Sigma_t, \Sigma_s)A^{-1} \qquad (15)$$

If we can determine covariance matrices Σ_t and Σ_s such that $\mu(x;\, \Sigma_t, \Sigma_s) = c_1\Sigma_t^{-1} = c_2\Sigma_s^{-1}$ for some constants c_1 and c_2, we obtain a fixed-point that is preserved under affine transformations. This property has been used to express *affine invariant interest point operators, affine invariant stereo matching*, as well as *affine invariant texture segmentation and texture recognition methods*. (21–25). In practice, affine invariants at a given image point can (up to an unknown scale factor and a free rotation angle) be accomplished by *shape adaptation*, that is by estimating the second-moment matrix μ using a rotationally symmetric scale-space and then iteratively by choosing the covariance matrices $\sum_t$ and $\sum_s$ proportional to μ^{-1} until the fixed-point has been reached or equivalently by warping the input image by linear transformations proportional to $A = \mu^{1/2}$ until the second-moment matrix is sufficiently close to a constant times the unit matrix.

AUTOMATIC SCALE SELECTION AND SCALE INVARIANT IMAGE DESCRIPTORS

Although the scale-space theory presented so far provides a well-founded framework for expressing visual operations at multiple scales, it does not address the problem of how to *select* locally appropriate scales for additional analysis. Whereas the problem of finding "the best scales" for handling a given data set may be regarded as intractable unless more information is available, in many situations a mechanism is required to generate hypotheses about interesting scales for additional analysis. Specifically, because the size of and the distance to objects may vary in real-life applications, a need exists to define scale invariant image descriptors. A general methodology (28) for generating hypotheses about interesting scale levels is by studying the evolution properties over scales of (possibly non-linear) combinations of γ-*normalized derivatives* defined by

$$\partial_\xi = t^{\gamma/2}\partial_x \quad \text{and} \quad \partial_\eta = t^{\gamma/2}\partial_y \qquad (16)$$

where γ is a free parameter to be determined for the task at hand. Specifically, *scale levels can be selected from the scales at which* γ-*normalized derivative expressions assume local extrema with respect to scale*. A general rationale for this statement is that under a scaling transformation $(x',y') = (sx, sy)$ for some scaling factor s with $f'(sx,sy) = f(x,y)$, if follows that for matching scale levels $t' = s^2t$ the m:th order normalized derivatives at corresponding scales $t' = s^2t$ in scale-space transform according to

$$L'_{\xi'^m}(x',y';\, t') = s^{m(\gamma-1)}L_{\xi^m}(x,y;\, t) \qquad (17)$$

Hence, for any differential expression that can be expressed as a homogeneous polynomial in terms of γ-normalized derivatives, it follows that local extrema over scales will be preserved under scaling transformations. In other words, if a γ-normalized differential entity $D_{norm}L$ assumes an extremum over scales at the point $(x_0, y_0;\, t_0)$ in scale-space, then under a rescaling transformation an extremum in the transformed differential invariant is assumed at $(sx_0, sy_0;\, s^2t_0)$, in the scale-space L' of the transformed image. This general property means that if we can find an expression based on γ-normalized Gaussian

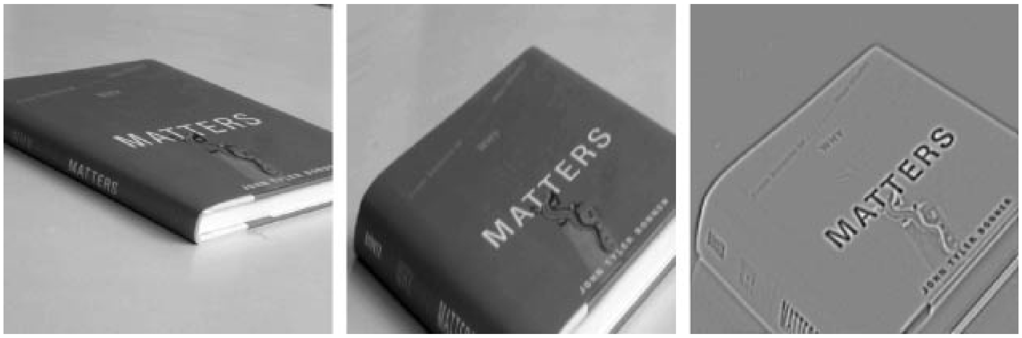


Figure 6. *Affine normalization by shape adaptation in affine scale-space*: (left) A gray-level image with an oblique view of a book cover. (middle) The result of effine normalization of a central image patch using iterative shape adaptation with affine transformations proportional to $A = \mu^{1/2}$ (right) An example of computing differential geometric descriptors, here the Laplacian $\nabla^2 L$ at scale $t = 2$, in the affinely normalized frame.

derivatives that assumes a local extremum over scales for a suitable set of image structures, then we can define a *scale invariant feature detector and / or image descriptor* by computing it at the scale at which the local extremum over scales is assumed.

The scale estimate $\hat{t}$ obtained from the scale selection step and can therefore be used for tuning or for guiding other early visual processes to be truly scale invariant (3,26) . With the incorporation of a mechanism for automatic scale selection, *scale-tuned visual modules* in a vision system can handle objects of different sizes as well as objects with different distances to the camera in the same manner. These requirements are essential for any vision system intended to function robustly in a complex dynamic world.

By studying the γ-normalized derivative response to a one-dimensional sine wave $f(x) = \sin(\omega x)$, for which the maximum over scales in the m:th order derivative is assumed at scale

$$\sigma_{max} = \sqrt{\frac{\gamma m \lambda}{2\pi}}$$

(measured in units of the standard deviation of the Gaussian) proportional to the wavelength $\lambda = 2\pi/\omega$ of the signal, one can see that a qualitative similarity exists between this construction and a peak in a local Fourier transform. However, also two major difference exist: *(1)* no window size is needed compute the Fourier transform and *(2)* this approach applies also to nonlinear differential expressions.

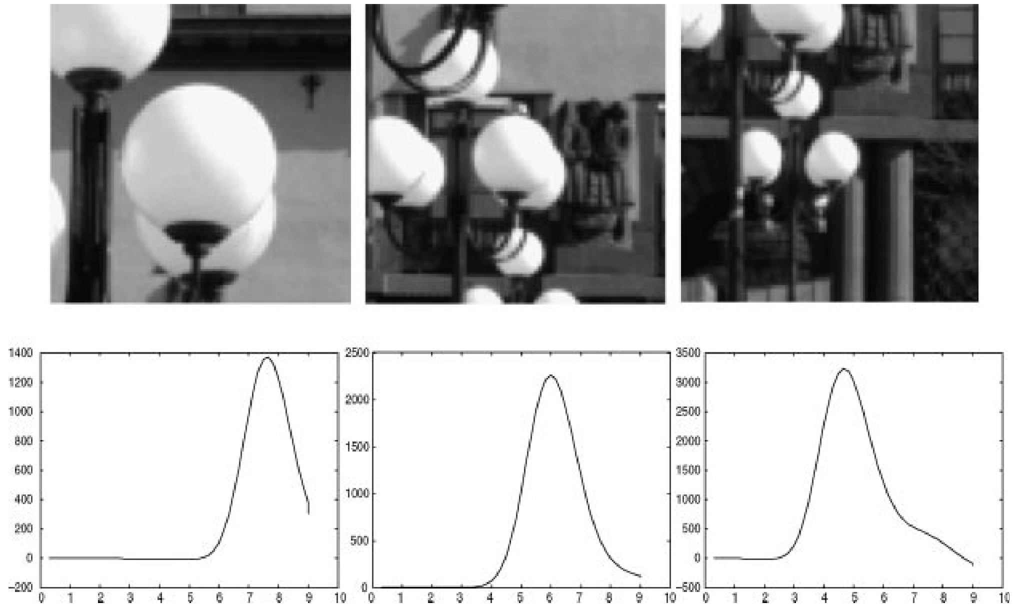


Figure 7. *Automatic scale selection from local extrema over scales of normalized derivatives*: (top row) Subwindows showing different details from Fig. 4 with image structures of different size, (bottom row) *Scale-space signatures* of the scale normalized determinant of the Hessian det $\mathcal{H}_{norm}L$ accumulated at the centre of each window. The essential property of the scale dependency of these scale normalized differential entities is that *the scale at which a local extremum over scales is assumed is proportional to the size of the corresponding image structure in the image domain*. The horizontal axis on these graphs represent effective scale, which corresponds roughly to the logarithm of the scale parameter: $\tau \approx \log_2 t$.

Specifically, if we choose γ=1, then under scaling transformations the magnitudes of normalized scale-space derivatives in Equation (17) are *equal* at corresponding points in scale-space. For $\gamma \neq 1$ they are related according to a (known) power of the scaling factor (see also Ref. 27 for earlier work on receptive field responses under similarity transformations).

SCALE INVARIANT FEATURE DETECTORS WITH INTERGRATED SCALE SELECTION MECHANISM

The most commonly used entity for automatic scale selection is the *scale normalized Laplacian* (3,26)

$$\nabla^2_{\text{norm}} L = t(L_{xx} + L_{yy}) \tag{18}$$

with $\gamma = 1$. A general motivation for the usefulness of this descriptor for general purpose scale selection can be obtained from the fact that the scale-space representation at any point can be decomposed into an integral of Laplacian responses over scales:

$$\begin{aligned} L(x,y;\, t_0) &= -(L(x,y;\, \infty) - L(x,y;\, t_0)) \\ &= -\int_{t=t_0}^{\infty} \partial_t L(x,y;\, t)\, dt \\ &= -\tfrac{1}{2}\textstyle\int_{t=t_0}^{\infty} \nabla^2 L(x,y;\, t)\, dt \end{aligned} \tag{19}$$

After a reparameterization of the scale parameter into *effective scale* $\tau = \log t$, we obtain:

$$\begin{aligned} L(x,y;\, t_0) &= -\tfrac{1}{2}\textstyle\int_{t=t_0}^{\infty} t\nabla^2 L(x,y;\, t)\, \frac{dt}{t} \\ &= -\textstyle\int_{\tau=\tau_0}^{\infty} \nabla^2_{norm} L(x,y;\, \tau)\, d\tau \end{aligned} \tag{20}$$

By detecting the scale at which the normalized Laplacian assumes its positive maximum or negative minimum over scales, we determine the scale for which the image in a scale-normalized bandpass sense contains the maximum amount of information.

Another motivation for using the scale-normalized Laplacian operator for early scale selection is that it serves as an excellent blob detector. For any (possibly nonlinear) γ-normalized differential expression $D_{norm}L$, let us first define a *scale-space maximum* (minimum) as a point for which the γ-normalized differential expression assumes a maximum (minimum) over *both* space and scale. Then, we obtain a straightforward *blob detector with automatic scale selection that responds to dark and bright blobs*, from the scale-space maxima and the scale-space minima of $\nabla^2_{norm}L$ Another *blob detector with automatic scale selection that also responds to saddles* can be defined from the scale-space maxima and minima of the scale-normalized determinant of the Hessian (26):

$$det\, \mathcal{H}_{\text{norm}} L = t^2 (L_{xx} L_{yy} - L_{xy}^2) \tag{21}$$

For both scale invariant blob detectors, *the selected scale reflects the size of the blob*. For the purpose of scale invariance, it is, however, not necessary to use the same entity to determine spatial interest points as to determine interesting scales for those points. An alternative approach to *scale interest point detection* is to use the Harris operator in Equation (11) to determine spatial interest points and the scale normalized Laplacian for determining the scales at these points (23). *Affine covariant interest points* can in turn be obtained by combining any of these three interest point operators with subsequent affine shape adaptation following Equation (15) combined with a determination of the remaining free rotation angle using, for example, the orientation of the image gradient.

The free parameter γ in the γ-normalized derivative concept can be related to the dimensionality of the type of image features to be detected or to a normalization of the Gaussian derivative kernels to constant L_{p^-}norm over scales. For blob-like image descriptors, such as the interest points described above, $\gamma = 1$ is a good choice and corresponds to L_1-normalization. For other types of image structures, such as thin edges, elongated ridges, or rounded corners, other values will be preferred. For example, for the problem of edge detection, $\gamma = 1/2$ is a useful choice to capture the width of diffuse edges, whereas for the purpose of ridge detection, $\gamma = 3/4$ is preferable for tuning the scale levels to the width of an elongated ridge (28–30).

SCALE-SPACE AXIOMS

Besides its practical use for computer vision problems, the scale-space representation satisfies several theoretic properties that define it as a unique form of multiscale image representation: The linear scale-space representation is obtained from *linear* and *shift-invariant* transformations. Moreover, the Gaussian kernels are *positive* and satisfy the *semi-group property*

$$g(\cdot,\cdot;\, t_1) * g(\cdot,\cdot;\, t_2) = g(\cdot,\cdot;\, t_1 + t_2) \tag{22}$$

which implies that any coarse-scale representation can be computed from any fine-scale representation using a similar transformation as in the transformation from the original image

$$L(\cdot,\cdot;\, t_2) = g(\cdot;\, t_2 - t_1) * L(\cdot,\cdot;\, t_1) \tag{23}$$

In one dimension, Gaussian smoothing implies that *new local extrema or new zero-crossings cannot be created with increasing scales* (1,3). In 2-D and higher dimensions scale-space representation obeys *nonenhancement of local extrema* (causality), which implies that the value at a local maximum is guaranteed not to increase while the vaiue at a local minimum is guaranteed to not decrease (2,3,31). The regular linear scale-space is *closed* under translations, rotations and scaling transformations. In fact, it can be shown that Gaussian smoothing arises *uniquely* for different

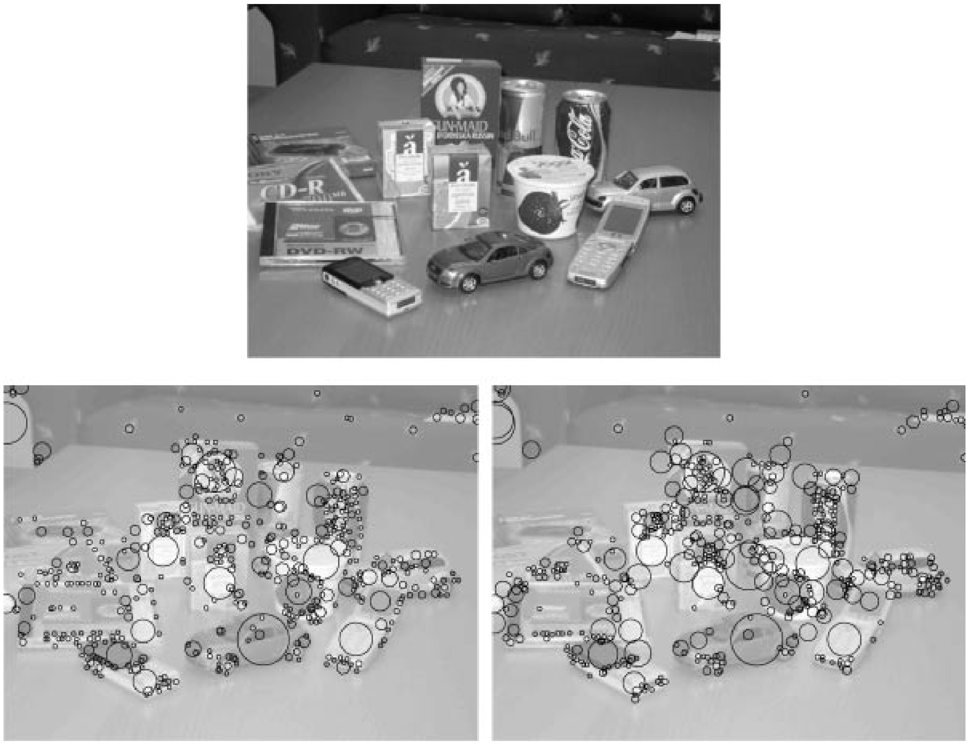


Figure 8. *Scale-invariant feature detection*: (top) Original gray-level image, (bottom left) The 1000 strongest scale-space extrema of the scale normalized Laplacian ∇^2_{norm}. (bottom right) The 1000 strongest scale-space extrema of the scale normalized determinant of the Hessian det $\mathcal{H}_{norm}L$. Each feature is displayed by a circle with its size proportional to the detection scale. In addition, the color of the circle indicates the type of image feature: red for dark features, blue for bright features, and green for saddle-like features.

subsets of combinations of these special and highly useful properties. For partial views of the history of scale-space axiomatics, please refer to Refs. 31 and 32 and the references therein.

Concerning the topic of automatic scale selection, it can be shown that, the notion of γ-normalized derivatives in Equation (16) devlops necessity from the requirement that local extrema over scales should be preserved under scaling transformations (26).

RELATIONS TO BIOLOGIC, VISION

Interestingly, the results of this computationally motivated analysis of early visual operations are in qualitative agreement with current knowledge about biologic vision. Neurophysiologic studies have shown that receptive field profiles exists in the mammalian retina and the visual cortex that can be well modeled by Gaussian derivative operators (33,34).

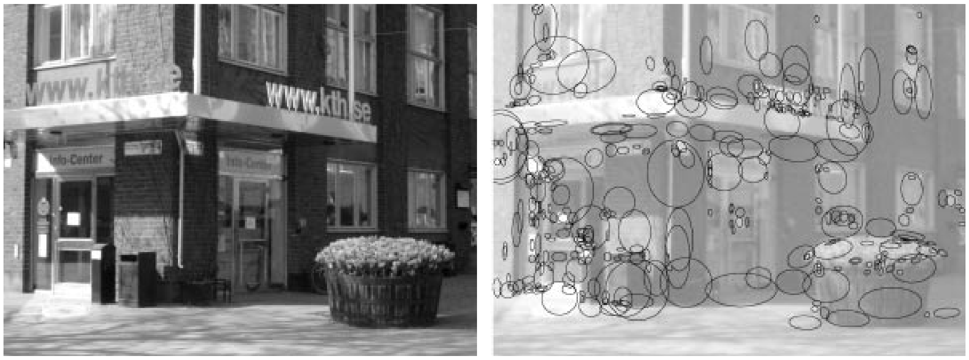


Figure 9. *Affine covariant image features*: (left) Original gray-level image. (right) The result of applying affine shape adaptation to the 500 strongest scale-space extrema of the scale normalized determinant of the Hessian det $\mathcal{H}_{norm}L$ (resulting in 384 features for which the iterative scheme converged). Each feature is displayed by an ellipse with its size proportional to the detection scale and the shape determined from a linear transformation A determined from a second-moment matrix μ. In addition, the color of the ellipse indicates the type of image feature: red for dark features, blue for bright features, and green for saddle-like features.

SUMMARY AND OUTLOOK

Scale-space theory provides a well-founded framework for modeling image structures at multiple scales, and the output from the scale-space representation can be used as input to a large variety of visual modules. Visual operations such as feature detection, feature classification, stereo matching, motion estimation, shape cues, and image-based recognition can be expressed directly in terms of (possibly nonlinear) combinations of Gaussian derivatives at multiple scales. In this sense, scale-space representation can serve as a basis for early vision.

The set of early uncommitted operations in a vision system, which perform scale-space smoothing, compute Gaussian derivatives at multiple scales, and combine these into differential invariants or other types of general purpose features to be used as input to later stage visual processes, is often referred to as a *visual front-end*.

Pyramid representation is a predecessor to scale-space representation, constructed by simultaneously smoothing and subsampling a given signal (35,36) In this way, computationally highly efficient algorithms can be obtained. A problem with pyramid representations, how-ever, is that it is algorithmically harder to relate structures at different scales, due to the discrete nature of the scale levels. In a scale-space representation, the existence of a continuous scale parameter makes it conceptually much easier to express this *deep structure*. For features defined as zero-crossings of differential invariants, the implicit function theorem defines trajectories directly across scales, and at those scales where bifurcations occur, the local behavior can be modeled by singularity theory. Nevertheless, pyramids are frequently used to express computationally more efficient approximations to different scale-space algorithms.

Extensions of linear scale-space theory concern the formulation of nonlinear scale-space concepts more committed to specific purposes (37,38). Strong relations exist between scale-space theory and *wavelets*, although these two notions of multiscale representations have been developed from somewhat different premises.

BIBLIOGRAPHY

1. A. P. Witkin, Scale-space filtering, *Proc. 8th Int. Joint Conf. Art. Intell.*, 1983, pp. 1019–1022.
2. J. J. Koenderink, The structure of images, *Biological Cybernetics* **50**: 363–370, 1984.
3. T. Lindeberg, *Scale-Space Theory in Computer Vision*, Dordrecht: Kluwer/Springer, 1994.
4. J. Sporring, M. Nielsen, L. Florack and P. Johansen, eds. *Gaussian Scale-Space Theory: Proc. PhD School on Scale-Space Theory*, Dordrecht: Kluwer/Springer, 1996.
5. L. M. J. Florack, *Image Structure*, Dorarecht: Kluwer/ Springer, 1997.
6. B. t. H. Romeny, *Front-End Vision and Multi-Scale Image Analysis*, Dordrecht: Kluwer/Springer, 2003.
7. J. J. Koenderink, and A. J. van Doorn, Representation of local geometry in the visual system, *Biological Cybernetics* **55**: 367–375, 1987.
8. J. J. Koenderink, and A. J. van Doorn, Generic neighborhood operations, *IEEE Trans. Pattern Anal. Machine Intell.* 14(6): 597–605, 1992.
9. T. Leung, and J. Malik, Representing and recognizing the visual appearance of materials using three-dimensional textons', *Int. J. of Computer Vision*, **43**(1): 29–44, 2001.
10. D. G. Jones, and J. Malik, A computational framework for determining stereo correspondences from a set of linear spatial filters, *Proc. Eur. Conf. Comp. Vis.*, 1992, pp. 395–410.
11. C. Schmid, and R. Mohr, Local grayvalue invariants for image retrieval, *IEEE Trans. Pattern Anal. Machine Intell.* **19**(5): 530–535, 1997.
12. B. Schiele, and J. Crowley, Recognition without correspondence using multidimensional receptive field histograms, *Int. J. of Computer Vision*, **36**(1): 31–50, 2000.
13. D. Lowe, Distinctive image features from scale-invariant keypoints, *Int. J. of Computer Vision*, **60**: (2), 91–110, 2004.
14. H. Bay, T. Tuytelaars, and Luc van Gool SURF: speeded up robust features, *Proc. European Conf. on Computer Vision*, Springer LNCS 3951, I: 404–417, 2006.
15. C. Harris, and M. Stevens, A combined corner and edge detector, *Proc. Alvey Workshop on Visual Motion*, 1988, pp. 156–162.
16. J. J. Koenderink and A. J. van Doorn, The structure of locally orderless images, *Int. J. of Computer Vision*, **31** (2): 159–168, 1999.
17. D. Hall, V. de Verdiere, and J. Crowley, Object recognition using coloured receptive fields, *Proc. European Conf. on Computer Vision*, Springer LNCS 1842, I: 164–177, 2000.
18. J. J. Koenderink, and A. Kappers, Colour space, unpublished lecture notes, Utrecht University, The Netherlands.
19. J. M. Geusebroek, R. van den Boomgaard, A. W. M. Smeulders, and A. Dev, Color and scale: The spatial structure of color images. *Proc. European Conf. on Computer Vision*, Springer LNCS 1842, I: 331–341, 2000.
20. J. M Geusebroek, R. van den Boomgaard, A. W. M. Smeulders, and H. Geerts, Color invariance, *IEEE Patt. Anal. Mach. Intell*, **23**: 12; 1338–1346, 2001.
21. T. Lindeberg, and J. Garding, Shape-adapted smoothing in estimation of 3-D depth cues from affine distortions of local 2-D structure, *Image and Vision Computing* **15**: 415–434, 1997.
22. A. Baumberg, Reliable feature matching across widely separated views *Proc. Comp. Vision Patt. Recogn.,* I: 1774–1781, 2000.
23. K. Mikolaiczyk, and C. Schmid, Scale and affine invariant interest point detectors', *Int. J. Comp. Vision*, **60**: 1, 63–86, 2004.
24. F. Schaffahtzky, and A. Zisserman, Viewpoint invariant texture matching and wide baseline stereo, *Proc. Int. Conf. Comp. Vis.*, 2: 636–644, 2001.
25. S. Lazebnik, C. Schmid, and J. Ponce, Affine-invariant local descriptors and neighbourhood statistics for texture recognition, *Proc. Int. Conf. Comp. Vis.*, I: 649–655, 2003.
26. T. Lindeberg, Feature detection with automatic scale selection, *Int. J. of Computer Vision* **30**(2): 77–116, 1998.
27. D. J. Field, Relations between the statistics of natural images and the response properties of cortical cells, *J. Opt. Soc. Am.* **4**: 2379–2394, 1987.
28. T. Lindeberg, Edge detection and ridge detection with automatic scale selection, *Int. J. of Computer Vision*, **30** (2): 117–154, 1998.

29. A. F. Frangi, W. J. Niessen, R. M. Hoogeveen, T. van Walsum, M. A. Viergever, Model-based detection of tubular structures in 3D images, *IEEE Trans. Med. Imaging*, **18**(10): 946–956, 2000.

30. K. Krissian, G. Malandain, N. Avache, R. Vaillant and Y. Trousset, Model-based detection of tubular structures in 3D images, *Comput. Vis. Image Underst.*, **80**(2): 130–171, 2000.

31. T. Lindeberg, On the axiomatic foundations of linear scale-space: Combining semi-group structure with causailty vs. scale invariance, In: J. Sporring et al. (eds.) *Gaussian Scale-Space Theory*, pp. Kluwer/Springer, 1997, 75–98.

32. J. Weickert, Linear scale space has first been proposed in Japan, *J. Math. Imaging and Vision*, **10**(3): 237–252, 1999.

33. R. A. Young, The Gaussian derivative model for spatial vision, *Spatial Vision* **2**: 273–293, 1987.

34. G. C. DeAngelis, I. Ohzawa and R. D. Freeman, Receptive field dynamics in the central visual pathways, Trends Neurosc. **18** (10): 451–457, 1995.

35. P. J. Burt, and E. H. Adelson, The Laplacian pyramid as a compact image code, *IEEE Trans. Comm.* **9:**4, 532–540, 1983.

36. J. Crowley, and A. C. Parker, A representation for shape based on peaks and ridges in the difference of low-pass transform, *IEEE Trans. Pattern Anal. Machine Intell.* **6**(2): 156–170, 1984.

37. B. t. H. Romeny, ed. *Geometry-Driven Diffusion in Computer Vision*, Dordrecht: Kluwer/Springer, 1997.

38. J. Weickert, *Anisotropic Diffusion in Image Processing*, Germany: Teubner-Verlag, 1998.

Tony Lindeberg
KTH (Royal Institute
of Technology)
Stockholm, Sweden

BEHAVIORAL SCIENCES AND COMPUTING

This article presents an overview of behavioral science research on human–computer interactions. The use of high-speed digital computers in homes, schools, and the workplace has been the impetus for thousands of research studies in the behavioral sciences since the 1950s. As computers have become an increasingly important part of daily life, more studies in the behavioral sciences have been directed at human–computer use. Research continues to proliferate, in part, because rapid technological advances continue to lead to the development of new products and applications from which emerge new forms of human–computer interactions. Examples include engaging in social interactions through electronic mail, chat, and discussion groups; using commercial websites for shopping and banking; using Internet resources and multimedia curriculum packages to learn in schools and at home; using handheld computers for work and personal life; collaborating in computer-supported shared workspaces; telecommuting via the Internet; engaging in one–many or many–many synchronous and asynchronous communications; and performing in "virtual" environments. Given the sheer quantity of empirical investigations in behavioral sciences computing research, the reader should appreciate the highly selective nature of this article. Even the reading list of current journals and books included at the end of this article is highly selective.

We present behavioral science computing research according to the following three categories: (1) antecedent-consequence effects, (2) model building, and (3) individual-social perspectives. The first category, antecedent-consequent effects, asks questions such as follows: How does variability in human abilities, traits, and prior performance affect computer use? How does use of computers affect variability in human abilities, traits, and subsequent performance? The second category, model building, consists of research on the nature of human abilities and performance using metaphors from computer science and related fields. Here, the behavioral scientist is primarily interested in understanding the nature of human beings but uses computer metaphors as a basis for describing and explaining human behavior. Model building can also start with assumptions about the nature of human beings, for example, limitations on human attention or types of motivation that serve as the basis for the development of new products and applications for human use. In this case, the behavioral scientist is mainly interested in product development but may investigate actual use. Such data may serve to modify the original assumptions about human performance, which in turn lead to refinements in the product. The third category, individual-social perspective, investigates the effects of increased access to and acceptance of computers in everyday life on human social relations. Questions addressed here are those such as follows: Do computers serve to isolate or connect persons to one another? What are the implications of lack of either access or acceptance of computers in modern cultures? These three categories of work in behavioral science computing are not mutually exclusive as the boundaries between any two of them are not fixed and firm.

ANTECEDENT-CONSEQUENCE RESEARCH

Personality

Research conducted since the 1970s has sought to identify what type of person is likely to use computers and related information technologies, succeed in learning about these technologies and pursue careers that deal with the development and testing of computer products. Most recently a great deal of attention has been given to human behavior on the Internet. Personality factors have been shown to be relevant in defining Internet behavior. Studies indicate that extroversion-introversion, shyness, anxiety, and neuroticism are related to computer use. Extroverts are outer directed, sociable, enjoy stimulation, and are generally regarded to be "people oriented" in contrast to introverts who are inner directed, reflective, quiet, and socially reserved. The degree of extroversion-introversion is related to many aspects of everyday life, including vocational choice, performance in work groups, and interpersonal functioning. Early studies suggested that heavy computer users tended to be introverts, and programming ability, in particular, was found to be associated with introversion. Recent studies reveal less relationship between introversion-extroversion and degree of computer use or related factors such as computer anxiety, positive attitudes toward computers, and programming aptitude or achievement. However, the decision to pursue a career in computer-related fields still shows some association with introversion.

Neuroticism is a tendency to worry, to be anxious and moody, and to evidence negative emotions and outlooks. Studies of undergraduate students and of individuals using computers in work settings have found that neuroticism is associated with anxiety about computers and negative attitudes toward computers. Neurotic individuals tend to be low users of computers, with the exception of the Internet, where there is a positive correlation between neuroticism and some online behavior. Neurotic people are more likely than others to engage in chat and discussion groups and to seek addresses of other people online. It is possible that this use of the Internet is mediated by loneliness, so that as neurotic people alienate others through their negative behaviors, they begin to feel lonely and then seek relationships online (1). Anxious people are less likely, however, to use the Internet for information searches and may find the many hyperlinks and obscure organization disturbing. Shyness, a specific type of anxiety related to social situations, makes it difficult for individuals to interact with others and to create social ties. Shy people

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

are more likely to engage in social discourse online than they are offline and can form online relationships more easily. There is a danger that as they engage in social discourse online, shy people may become even less likely to interact with people offline. There has been some indication, however, that shy people who engage in online relationships may become less shy in their offline relationships (2).

As people spend time on the Internet instead of in the real world, behavioral scientists are concerned that they will become more isolated and will lose crucial social support. Several studies support this view, finding that Internet use is associated with reduced social networks, loneliness, and difficulties in the family and at work. Caplan and others suggest that these consequences may result from existing psychosocial conditions such as depression, low self-efficacy, and negative self-appraisals, which make some people susceptible to feelings of loneliness, guilt, and other negative outcomes, so that significant time spent online becomes problematic for them but not for others (3). There is some evidence of positive effects of Internet use: Involvement in chat sessions can decrease loneliness and depression in individuals while increasing their self-esteem, sense of belonging, and perceived availability of people whom they could confide in or could provide material aid (4). However, Caplan cautions that the unnatural quality of online communication, with its increased anonymity and poor social cues, makes it a poor substitute for face-to-face relationships, and that this new context for communication is one that behavioral scientists are just beginning to understand.

As computer technologies become more integral to many aspects of life, it is increasingly important to be able to use them effectively. This is difficult for individuals who have anxiety about technology that makes them excessively cautious near computers. Exposure to computers and training in computer use can decrease this anxiety, particularly if the training takes place in a relaxed setting in small groups with a user-friendly interface, provides both demonstrations and written instructions, and includes important learning strategies that help to integrate new understanding with what is already known (5). However, some individuals evidence such a high degree of anxiety about computer use that they have been termed "computerphobics." Here the fear of computers is intense and irrational, and exposure to computers may cause distinct signs of agitation including trembling, facial expressions of distress, and physical or communicative withdrawal. In extreme cases, a generalized anxiety reaction to all forms of technology termed "technophobia" has been observed. Personality styles differ when individuals with such phobias are compared with those who are simply uncomfortable with computer use. Individuals with great anxiety about computers have personality characteristics of low problem-solving persistence and unwillingness to seek help from others (6). The training methods mentioned above are less likely to benefit individuals who evidence severe computerphobia or very high levels of neuroticism. Intensive intervention efforts are probably necessary because the anxiety about computers is related to a personality pattern marked by anxiety in general rather than an isolated fear of computers exacerbated by lack of experience with technology.

Gender

Studies over many years have found that gender is an important factor in human–computer interaction. Gender differences occur in virtually every area of computing including occupational tasks, games, online interaction, and programming, with computer use and expertise generally higher in males than in females, although recent studies indicate that the gender gap in use has closed and in expertise is narrowing. This change is especially noticeable in the schools and should become more apparent in the workforce over time. In the case of the Internet, males and females have a similar overall level of use, but they differ in their patterns of use. Males are found to more often engage in information gathering and entertainment tasks, and women spend more time in communication functions, seeking social interaction online (7). Females use e-mail more than males and enjoy it more, and they find the Internet more useful overall for social interaction. These differences emerge early. Girls and boys conceptualize computers differently, with boys more likely to view computers as toys, meant for recreation and fun, and to be interested in them as machines, whereas girls view computers as tools to accomplish something they want to do, especially in regard to social interaction (8). This may be due, in part, to differences in gender role identity, an aspect of personality that is related to, but not completely determined by, biological sex. Gender role identity is one's sense of self as masculine and/or feminine. Both men and women have traits that are stereotypically viewed as masculine (assertiveness, for example) and traits that are stereotypically viewed as feminine (nurturance, for example) and often see themselves as possessing both masculine and feminine traits. Computer use differs between people with a high masculine gender role identity and those with a high feminine gender role identity (9). Some narrowing of the gender gap in computer use may be due to changing views of gender roles.

Age

Mead et al. (10) reviewed several consequences of computer use for older adults, including increased social interaction and mental stimulation, increased self-esteem, and improvements in life satisfaction. They noted, however, that older adults are less likely to use computers than younger adults and are less likely to own computers; have greater difficulty learning how to use technology; and face particular challenges adapting to the computer-based technologies they encounter in situations that at one time involved personal interactions, such as automated check-out lines and ATM machines. They also make more mistakes and take more time to complete computer-based tasks. In view of these factors, Mead et al. suggested psychologists apply insights gained from studying the effects of age on cognition toward the development of appropriate training and computer interfaces for older adults. Such interfaces could provide more clear indications to the user of previously followed hyperlinks, for example, to compensate for failures in episodic memory, and employ cursors with larger activation areas to compensate for reduced motor control.

Aptitudes

Intelligence or aptitude factors are also predictors of computer use. In fact, spatial ability, mathematical problem-solving skills, and understanding of logic may be better than personality factors as predictors. A study of learning styles, visualization ability, and user preferences found that high visualizers performed better than low visualizers and thought computer systems were easier to use than did low visualizers (11). High visualization ability is often related to spatial and mathematical ability, which in turn has been related to computer use, positive attitudes about computers, and educational achievement in computer courses. Others have found that, like cognitive abilities, the amount of prior experience using computers for activities such as game playing or writing is a better predictor of attitudes about computers than personality characteristics. This may be because people who have more positive attitudes toward computers are therefore more likely to use them. However, training studies with people who have negative views of computers reveal that certain types of exposure to computers improve attitudes and lead to increased computer use. Several researchers have suggested that attitudes may play an intermediary role in computer use, facilitating experiences with computers, which in turn enhance knowledge and skills and the likelihood of increased use. Some have suggested that attitudes are especially important in relation to user applications that require little or no special computing skills, whereas cognitive abilities and practical skills may play a more important role in determining computer activities such as programming and design.

Attitudes

Attitudes about self-use of computers and attitudes about the impact of computers on society have been investigated. Research on attitudes about self-use and comfort level with computers presumes that cognitive, affective, and behavioral components of an attitude are each implicated in a person's reaction to computers. That is, the person may believe that computers will hinder or enhance performance on some task or job (a cognitive component), the person may enjoy computer use or may experience anxiety (affective components), and the individual may approach or avoid computer experiences (behavioral component). In each case, a person's attitude about him- or herself in interaction with computers is the focus of the analysis. Attitudes are an important mediator between personality factors and cognitive ability factors and actual computer use.

Individuals' attitudes with respect to the impact of computers on society vary. Some people believe that computers are dehumanizing, reduce human–human interaction, and pose a threat to society. Others view computers as liberating and enhancing the development of humans within society. These attitudes about computers and society can influence the individual's own behavior with computers, but they also have potential influence on individuals' views of computer use by others and their attitudes toward technological change in a range of settings.

Numerous studies have shown that anxiety about using computers is negatively related to amount of experience with computers and level of confidence in human–computer interaction. As discussed, people who show anxiety as a general personality trait evidence more computer use anxiety. In addition, anxiety about mathematics and a belief that computers have a negative influence on society are related to computer anxiety. Thus, both types of attitudes—attitudes about one's own computer use and attitudes about the impact of computers on society—contribute to computer anxieties (12).

With training, adult students' attitudes about computers become more positive. That is, attitudes about one's own interaction with computers and attitudes about the influence of computers on society at large generally become more positive as a result of instruction through computer courses in educational settings and of specific training in a variety of work settings. Figure 1 presents a general model of individual differences in computer use. The model indicates that attitudes are affected by personality and cognitive factors, and that they in turn can affect computer use either directly or by influencing values and expectations. The model also indicates that computer use can influence attitudes toward computers and personality factors such as loneliness.

Influence of Gender on Attitudes. Gender differences in attitudes toward computer use, although becoming less pronounced, appear relatively early, during the elementary school years, and persist into adulthood. Male students have more positive attitudes than female students, express greater interest in computers and greater confidence in their own abilities, and view computers as having greater utility in society than females at nearly every age level. One study revealed a moderate difference between males and females in their personal anxiety about using computers, with women displaying greater levels than men, and holding more negative views than men about the influence of computers on society. The findings of this study suggest that gender differences in computer-related behavior are due in part to differences in anxiety. When anxiety about computers was controlled, there were few differences between males and female's in computer behavior: It seems that anxiety mediates some gender differences in computer-related behavior (13). Other studies confirm that gender differences in computer behavior seem to be due to attitudinal and experiential factors. Compared with men, women report greater anxiety about computer use, lower confidence about their ability to use the computer, and lower levels of liking computer work. However, when investigators control the degree to which tasks are viewed as masculine or feminine and/or control differences in prior experiences with computers, gender differences in attitudes are no longer significant (14).

Middle-school students differ by gender in their reactions to multimedia learning interfaces and may have different sources for intrinsic satisfaction when using computers. Boys particularly enjoy control over computers and

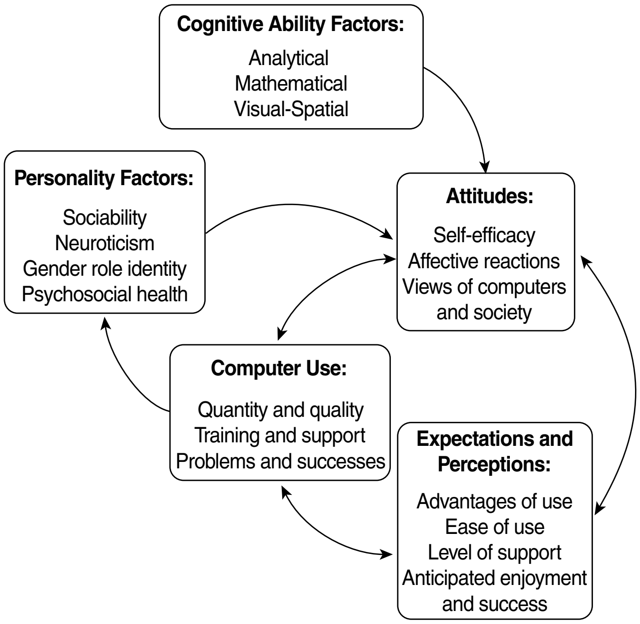


Figure 1. General model of individual differences in computer use.

look for navigational assistance within computer games, whereas girls prefer calmer games that include writing and ask for assistance rather than try to find navigational controls (15). This indicates that gender differences in attitudes about computers may be influenced in part through experience with gender-preferred computer interfaces, and that attitudes of girls toward computer use improve when gender-specific attention is paid to the design of the interface and the type of tasks presented. The differences by gender in patterns of Internet use by adults support this conclusion. When individuals are free to determine what type of tasks they do online, gender differences in overall use disappear, although males are more likely to gather information and seek entertainment and women are more likely to interact socially.

Other studies suggest that gender differences in attitudes toward computers may vary with the nature of the task. In one study, college students performed simple or more complex computer tasks. Men and women did not differ in attitudes following the simple tasks. However, the men reported a greater sense of self-efficacy (such as feelings of effective problem-solving and control) than the women after completing the complex tasks (16). Such findings suggest that, in addition to anxiety, a lack of confidence affects women more than men in the area of computer use. Training does not always reduce these differences: Although people generally become less anxious about computer use over the course of training, in some cases women become more anxious (17). This increase in anxiety may occur even though women report a concomitant increase in a sense of enjoyment with computers as training progressed. With training, both men and women have more positive social attitudes toward computers and perceive computers to be more like humans and less like machines.

To summarize the state of information on attitudes about computer use thus far, results suggest that attitudes about one's own computer use are related to personal anxiety about computer use as well as to math anxiety. These relationships are more likely to occur in women than in men. However, when women have more computer experiences, the relationship between anxiety and computer use is diminished and the gender difference is often not observed. Several attitudinal factors seem to be involved in computer use, including math anxiety, feelings of self-efficacy and confidence, personal enjoyment, and positive views of the usefulness of computers for society.

Workplace

Computers are used in a variety of ways in organizations, and computing attitudes and skills can affect both the daily tasks that must be performed in a routine manner and the ability of companies to remain efficient and competitive. The degree of success of computer systems in the workplace is often attributed to the attitudes of the employees who are end users of Inter- and intranet-based applications for communications and workflow, shared workspaces, financial applications, database management systems, data analysis software, and applications for producing Web resources and graphics. A study of factors influencing attitudes about computing technologies and acceptance of particular technologies showed that three factors were most important in determining user acceptance behaviors: perceived advantages to using the technology for improving job performance, perceived ease of use, and degree of enjoyment in using the technology. Anticipated organizational support, including technical support as well as higher management encouragement and resource allocation, had a significant effect on perceived advantage toward improving job performance, so is an additional factor in determining whether users make use of particular systems (18). The study also showed that the perceived potential for improving job performance was influenced by the extent to which the system was observed as consistent with existing needs, values, and past experiences of potential adopters. Overall, however, this and other research shows that the perception that systems will improve job performance is by far the strongest predictor of anticipated use.

Research on the attitudes of employees toward computers in the workplace reveals that, for the most part, computers are observed as having a positive effect on jobs, making jobs more interesting, and/or increasing employee effectiveness. Employees who report negative attitudes cite increased job complexity with the use of computers instead of increased effectiveness. They also report negative attitudes about the necessity for additional training and refer to a reduction in their feelings of competence. These mixed feelings may be related to their job satisfaction attitudes. When confusion and frustration about computer systems increase, job satisfaction decreases. The negative feelings about their own ability to use computers effectively lead employees to express greater dissatisfaction with the job as a whole (19).

Work-related computer problems can increase stress. Problems with computer systems (e.g., downtime, difficulties with access, lack of familiarity with software, etc.) often result in an increase of overall work time, a perception of increased workload and pressure, and less feeling of control over the job. In these situations, computers can be viewed as a detrimental force in the workplace even when users have a generally positive attitude toward them. There is some indication that individuals differ in their reactions to problems with computers, and that these differences play a role in views of the utility of computers on the job. Older staff who feel threatened by computers tend to complain more about time pressures and health-related issues related to computer use, whereas same-age peers who view computers more neutrally or positively report few problems (20).

Computer-supported collaborative work systems can facilitate group projects by allowing people to work together from different places and different times. Kunzer et al. (21) discuss guidelines for their effective design and use. For a system to be effective, it is important that its functions are transparent so that it is highly usable, and that it provides an information space structured for specific tasks for a particular group. Web-based portals that include these workspaces can greatly enhance collaborative activities. However, systems that are not highly usable and do not consider the unique requirements of the user are not well accepted and therefore not used. For example, inadequate consideration of user preferences regarding software applications can make it impossible for users to work with familiar sets of tools. Users then are likely to find other ways to collaborate either offline or in separate, more poorly coordinated applications. Successful shared workspace systems provide basic features such as a document component that allows various file formats with revision control and audit trails; calendars with search capabilities, schedule-conflict notification, and priority settings; threaded discussions with attachments, e-mail notification, and specification of read rights; and contact, project, and workflow management components. Finally, the most usable shared workspaces can be customized to different cooperation scenarios.

Differences in computer anxiety and negative attitudes about the social impact of computers are more likely to occur in some occupations than in others. Individuals in professional and managerial positions generally evidence more positive attitudes toward computers. Particular aspects of some jobs may influence individuals' attitudes and account for some of these differences. Medcof (22) found that the relative amounts of computing and noncomputing tasks, the job characteristics (such as skill variety, level of significance of assigned tasks, and autonomy), and the cognitive demand (e.g., task complexity) of the computing tasks interact with one another to influence attitudes toward computer use. When job characteristics are low and the computing components of the job also have low cognitive demand on the user (as in the case of data entry in a clerical job), attitudes toward computer use are negative, and the job is viewed as increasingly negative as the proportion of time spent on the low cognitive demand task increases. If a larger proportion of the work time is spent on a high cognitive demand task involving computer use, attitudes toward computer use and toward the job will be more positive.

Medcof's findings suggest that under some conditions job quality is reduced when computers are used to fulfill assigned tasks, although such job degradation can be minimized or avoided. Specifically, when jobs involve the use of computers for tasks that have low levels of cognitive challenge and require a narrow range of skills, little autonomy, and little opportunity for interaction with others, attitudes toward computer use, and toward the job, are negative. But varying types of noncomputing tasks within the job (increased autonomy or social interaction in noncomputing tasks, for example) reduces the negative impact; inclusion of more challenging cognitive tasks as part of the computing assignment of the job is especially effective in reducing negative views of computer use. The attitudes about computers in the workplace therefore depend on the relative degree of computer use in the entire job, the cognitive challenge involved in that use, and the type of noncomputing activities.

Older workers tend to use computers in the workplace less often than younger workers, and researchers have found that attitudes may be implicated in this difference. As older workers tend to have more negative attitudes toward computers than younger workers or those with less seniority, they use them less. Negative attitudes toward computer use and computer anxiety are better predictors of computer use than age alone (23).

MODEL BUILDING

Cognitive Processes

Modifications in theories of human behavior have been both the cause and the effect of research in behavioral science computing. A "cognitive" revolution in psychology occurred during the 1950s and 1960s, in which the human mind became the focus of study. A general approach called information processing, inspired by computer science, became dominant in the behavioral sciences during this time period. Attempts to model the flow of information from input-stimulation through output-behavior have included considerations of human attention, perception, cognition, memory, and, more recently, human emotional reactions and motivation. This general approach has become a standard model that is still in wide use.

Cognitive science's interest in computer technologies stems also from the potential to implement models and theories of human cognition as computer systems, such as Newell and Simon's General Problem Solver, Chomsky's transformational grammar, and Anderson's Atomic Components of Thought (ACT). ACT represents many components and activities of human cognition, including procedural knowledge, declarative knowledge, propositions, spreading activation, problem solving, and learning. One benefit to implementing models of human thought on computers is that the process of developing a computer model constrains theorists to be precise about their theories, making it easier to test and then refine them. As more has been learned about the human brain's ability to process many inputs and operations simultaneously, cognitive theorists have developed connectionist computer models made up of large networks of interconnected computer processors, each network comprising many interconnected nodes. The overall arrangement of interconnected nodes allows the system to organize concepts and relationships among them, simulating the human mental structure of knowledge in which single nodes may contain little meaning but meaning emerges in the pattern of connections. These and other implementations of psychological theories show how the interactions between computer scientists and behavioral scientists have informed understandings of human cognition.

Other recent theoretical developments include a focus on the social, contextual, and constructive aspects of human cognition and behavior. From this perspective, human cognition is viewed as socially situated, collaborative, and jointly constructed. Although these developments have coincided with shifts from stand-alone computers to networks and Internet-based systems that feature shared workspaces, it would be erroneous to attribute these changes in theoretical models and explanation solely to changes in available technology. Instead, many of today's behavioral scientists base their theories on approaches developed by early twentieth-century scholars such as Piaget and Vygotsky. Here the focus shifts from examining individual cognitive processing to evaluating how people work within a dynamic interplay of social factors, technological factors, and individual attitudes and experiences to solve problems and learn (24). This perspective encourages the development of systems that provide mechanisms for people to scaffold other learners with supports that can be strengthened or faded based on the learner's understanding. The shift in views of human learning from knowledge transfer to knowledge co-construction is evident in the evolution of products to support learning, from early computer-assisted instruction (CAI) systems, to intelligent tutoring systems (ITS), to learning from hypertext, to computer-supported collaborative learning (CSCL). An important principle in this evolution is that individuals need the motivation and capacity to be more actively in charge of their own learning.

Human Factors

Human factors is a branch of the behavioral sciences that attempts to optimize human performance in the context of a system that has been designed to achieve an objective or purpose. A general model of human performance includes the human, the activity being performed, and the context (25). In the area of human–computer interactions, human factors researchers investigate such matters as optimal workstation design (e.g., to minimize soft tissue and joint disorders); the perceptual and cognitive processes involved in using software interfaces; computer access for persons with disabilities such as visual impairments; and characteristics of textual displays that influence reading comprehension. An important principle in human factors research is that improvements to the system are limited if considered apart from interaction with actual users. This emphasis on contextual design is compatible with the ethnographic movement in psychology that focuses on very detailed observation of behavior in real situations (24). A human-factors analysis of human learning from hypermedia is presented next to illustrate this general approach.

Hypermedia is a method of creating and accessing nonlinear text, images, video, and audio resources. Information in hypermedia is organized as a network of electronic documents, each a self-contained segment of text or other interlinked media. Content is elaborated by providing bridges to various source collections and libraries. Links among resources can be based on a variety of relations, such as background information, examples, graphical representations, further explanations, and related topics. Hypermedia is intended to allow users to actively explore knowledge, selecting which portions of an electronic knowledge base to examine. However, following links through multiple resources can pose problems when users become disoriented and anxious, not knowing where they are and where they are going (26). Human factors research has been applied to the hypermedia environments of digital libraries, where users search and examine large-scale databases with hypermedia tools.

Rapp et al. (27) suggest that cognitive psychology's understanding of human cognition should be considered during the design of digital libraries. Hypermedia structures can be fashioned with an awareness of processes and limitations in human text comprehension, mental representations, spatial cognition, learning, memory, and other aspects of cognitive functioning. Digital libraries can in turn provide real-world environments to test and evaluate theories of human information processing. Understandings of both hypermedia and cognition can be informed through an iterative process of research and evaluation, where hypotheses about cognitive processes are developed and experiments within the hypermedia are conducted. Results are then evaluated, prompting revisions to hypermedia sources and interfaces, and generating implications for cognitive theory. Questions that arise during the process can be used to evaluate and improve the organization and interfaces in digital library collections. For example, how might the multiple sources of audio, video, and textual information in digital libraries be organized to promote more elaborated, integrated, and better encoded mental representations? Can the goal-directed, active exploration and search behaviors implicit in hypermedia generate the multiple cues and conceptual links that cognitive science has found best enhance memory formation and later retrieval?

The Superbook hypertext project at Bellcore was an early example of how the iterative process of human-factor analysis and system revision prompted modifications in original and subsequent designs before improvements over traditional text presentations were observed. Dillon (28) developed a framework of reader–document interaction that hypertext designers used to ensure usability from the learner's perspective. The framework, intended to be an approximate representation of cognition and behavior central to reading and information processing, consists of four interactive elements: (1) a task model that deals with the user's needs and uses for the material; (2) an information model that provides a model of the information space; (3) a set of manipulation skills and facilities that support physical use of the materials; and (4) a processor that represents the cognitive and perceptual processing involved in reading words and sentences. This model predicts that the users' acts of reading will vary with their needs and knowledge of the structure of the environment that contains textual information, in addition to their general ability to "read" (i.e., acquire a representation that approximates the author's intention via perceptual and cognitive processes). Research comparing learning from hypertext versus traditional linear text has not yielded a consistent pattern of results (29). User-oriented models such as Dillon's enable designers to increase the yield from hypertext versus traditional text environments.

Virtual environments provide a rich setting for human–computer interaction where input and output devices are adapted to the human senses. Individuals using virtual reality systems are immersed into a virtual world that provides authentic visual, acoustic, and tactile information. The systems employ interface devices such as data gloves that track movement and recognize gestures, stereoscopic visualizers that render scenes for each eye in real time, headphones that provide all characteristics of realistic sound, and head and eye tracking technologies. Users navigate the world by walking or even flying through it, and they can change scale so they effectively shrink to look at smaller structures in more detail. Krapichler et al. (30) present a virtual medical imaging system that allows physicians to interactively inspect all relevant internal and external areas of a structure such as a tumor from any angle. In these and similar applications, care is taken to ensure that both movement through the virtual environment and feedback from it are natural and intuitive.

The emerging field of affective computing applies human factors research to the emotional interaction between users and their computers. As people seek meaning and patterns in their interactions, they have a tendency to respond to computers as though they were people, perceiving that that they have human attributes and personalities, and experiencing appropriate emotions when flattered or ignored. For example, when a computer's voice is gendered, people respond according to gender-stereotypic roles, rating the female-voiced computer as more knowledgeable about love and the male voice as more knowledgeable about technical subjects, and conforming to the computer's suggestions if they fall within its gender-specific area of expertise (31). Picard and Klein lead a team of behavioral scientists who explore this willingness to ascribe personality to computers and to interact emotionally with them. They devise systems that can detect human emotions, better understand human intentions, and respond to signs of frustration and other negative emotions with expressions of comfort and support, so that users are better able to meet their needs and achieve their objectives (32). The development and implementation of products that make use of affective computing systems provide behavioral theorists a rich area for ongoing study.

INDIVIDUAL-SOCIAL PERSPECTIVE

In a previous section we presented an overview of research on gender differences, attitudes toward the impact of computers on society, and the use of computers in the workplace. Each of these issues relates to the effects of computers on human social relations. One prevalent perception is that as people spend time on the Internet instead of engaging with people face-to-face they will become more isolated, lonely, and depressed, and that as the kinds of social discourse available online to them may be less meaningful and fulfilling, they could lose social support. As we discussed, increased Internet use may at times be a result of loneliness, not a cause of it. The emerging research about this concern is inconclusive, making this an important area for further research. Kling (33) lists additional social controversies about the computerization of society: class divisions in society; human safety and critical computer systems; democratization; the structure of labor markets; health; education; military security; computer literacy; and privacy and encryption. These controversies have yet to be resolved and are still being studied by behavioral scientists.

Psychologists explore the influence of computers on relationships among people, not only in terms of their online behaviors and interactions, but also their perceptions of themselves and one another. Power among relationships is sometimes renegotiated because of differing attitudes and competencies with technology. One researcher proposed that relatively lower computer expertise among fathers in contrast to their sons, and a sense of dependence on their sons for technical support, can change the family dynamic and emasculate fathers, reducing perceptions of their strength and of their own sense of competence (34).

Computer-Mediated Communication

Much research on computer–human interactions is developed in the context of using computers to communicate with other people. Computer-mediated communication

(CMC) is a broad term that covers forms of communication including e-mail, listservs, discussion groups, chat, instant messaging, and videoconferencing. In comparison with face-to-face communication, CMC has fewer social and nonverbal cues but allows people to communicate easily from different places and different times. Computer-supported collaborative work systems we discussed in regard to computers in the workplace are examples of specialized CMC systems that facilitate collaborative group work. They include many forms of communication, allowing both synchronous and asynchronous interactions among multiple participants. In the cases of chat, instant messaging, and videoconferencing, people communicate at the same time but may be in different locations. E-mail, listservs, and discussion groups have the added benefits and difficulties of asynchronous communication. The reduction in social and nonverbal cues in these forms of communication can be problematic, as people misinterpret messages that are not carefully constructed and may respond negatively. It has been found that these problems diminish with experience. As users adapt to the medium and create means of improving communication, and become more adept using linguistic cues, differences between CMC and face-to-face communication may be lessened. Social norms and conventions within groups serve to reduce individual variability across formats rendering CMC similar to face-to-face communication, especially in established organizations. For example, messages from superiors receive more attention than messages from coworkers or from subordinates.

Research on learning in the workplace and in educational institutions has examined CMC's ability to support the transfer of knowledge (an "instructional" perspective) and the social, co-construction of knowledge (a "conversational" perspective) (35). Grabe and Grabe (36) discuss how CMC can change the role of the teacher, effectively decentering interactions so that students feel freer to communicate and to become more involved in their own learning. CMC results in more diverse participation and a greater number of interactions among students when compared with traditional classroom discussion characterized by longer periods of talking by the teacher and shorter, less complex individual responses from students. With CMC, instructors can focus on observing and facilitating student learning and can intervene to help with scaffolding or direct instruction as needed. Structure provided by the teacher during CMC learning is important to help create a social presence and to encourage participation. Grabe and Grabe suggest that teachers assume responsibility for managing the discussion by defining the overall purpose of the discussion, specifying roles of participants, establishing expectations, and responding to negative or passive behaviors. In a study of interaction in an online graduate course, increased structure led to more interaction and increased dialogue (37).

Another potential area for discussion, computer-supported collaborative learning, is somewhat beyond the scope of this article but is included in our reading list.

Access

Behavioral scientists are interested in what inequities exist in access to technology, how the inequities developed, and what can be done to reduce them. Until recently, the number of computers in schools was significantly lower for poorer communities. These data are changing, and the differences are diminishing. However, socioeconomic factors continue to be powerful predictors for whether people have computers or Internet access in their homes. The National Telecommunications and Information Administration (NTIA) (38) reported that, although Internet use is increasing dramatically for Americans of all incomes, education levels, ages, and races, many inequities remain. Individuals in high-income households are much more likely to be computer and Internet users than those in low-income households (over 80% for the highest income households; 25% for the lowest income households). Age and level of education are also powerful predictors: Computer and Internet use is higher among those who are younger and those who are more highly educated. In addition, people with mental or physical disabilities are less likely than others to use computers or the Internet.

Many inequities in access to computers are declining. According to the NTIA's report, rates of use are rising much more rapidly among poorer, less educated, and elderly people, the very groups who have been most disadvantaged. The report attributes this development to the lowering cost of computer technology, and to the expanding use of the Internet at schools, work, and libraries, which makes these resources available to people who do not own computers.

Journals

Applied Ergonomics

Behavior and Information Technology

Behavior Research Methods, Instruments and Computers

Computers in Human Behavior

CyberPsychology and Behavior

Ergonomics Abstracts

Hypertext and Cognition

Interacting with Computers

International Journal of Human Computer Interaction

International Journal of Human Computer Studies

Journal of Educational Computing Research

Journal of Educational Multimedia and Hypermedia

Journal of Occupational and Organizational Psychology

Books

E. Barrett (ed.), *Sociomedia, Multimedia, Hypermedia, and the Social Construction of Knowledge*. Cambridge, MA: The MIT Press, 1992.

C. Cook, *Computers and the Collaborative Experience of Learning*. London: Routledge, 1994.

S. J. Derry, M. Siegel, J. Stampen, and the STEP Research Group, The STEP system for collaborative case-based teacher education: Design, evaluation and future directions, in *Proceedings of Computer Support for Collaborative Learning (CSCL) 2002*. Mahwah, NJ: Lawrence Erlbaum Associates, 2002, pp. 209–216.

D. H. Jonassen, K. Peck, and B. G. Wilson, *Learning With Technology: A Constructivist Perspective*. Columbus, OH: Merrill/Prentice-Hall, 1999.

D. H. Jonassen (ed.), *Handbook of Research on Educational Communications and Technology*, 2nd ed. Mahwah, NJ: Lawrence Erlbaum Associates, 2004.

T. Koschmann (ed.), *CSCL: Theory and Practice of an Emerging Paradigm*. Mahwah, NJ: Lawrence Erlbaum Associates, 1996.

J. A. Oravec, *Virtual Individual, Virtual Groups: Human Dimensions of Groupware and Computer Networking*. Melbourne, Australia: Cambridge University Press, 1996.

D. Reinking, M. McKenna, L. Labbo and R. Kieffer (eds.), *Handbook of Literacy and Technology: Transformations in a Post-typographic World*. Mahwah, NJ: Lawrence Erlbaum Associates, 1998.

S. Vosniadou, E. D. Corte, R. Glaser and H. Mandl, *International Perspectives on the Design of Technology-Supported Learning Environments*. Mahwah, NJ: Lawrence Erlbaum Associates, 1996.

B. B. Wasson, S. Ludvigsen, and U. Hoppe (eds.), *Designing for Change in Networked Learning Environments: Proceedings of the International Conference on Computer Support for Collaborative Learning 2003*. Boston, MA: Kluwer Academic, 2003.

BIBLIOGRAPHY

1. Y. Amichai-Hamburger and E. Ben-Artzi, Loneliness and Internet use, *Comput. Hum. Behav.*, **19**(1): 71–80, 2000.
2. L. D. Roberts, L. M. Smith, and C. M. Polluck, “U r a lot bolder on the net”: Shyness and Internet use, in W. R. Crozier(ed.), *Shyness: Development, Consolidation and Change*. New York: Routledge Farmer: 121–138, 2000.
3. S. E. Caplan, Problematic Internet use and psychosocial well-being: Development of a theory-based cognitive-behavioral measurement instrument, *Comput. Hum. Behav.*, **18**(5): 553–575, 2002.
4. E. H. Shaw and L. M. Gant, Users divided? Exploring the gender gap in Internet use, *CyberPsychol. Behav.*, **5**(6): 517–527, 2002.
5. B. Wilson, Redressing the anxiety imbalance: Computerphobia and educators, *Behav. Inform. Technol.*, **18**(6): 445–453, 1999.
6. M. Weil,L. D. Rosen, and S. E. Wugalter, The etiology of computerphobia, *Comput. Hum. Behav.*, **6**(4): 361–379, 1990.
7. L.A. Jackson, K.S. Ervin, and P.D. Gardner, Gender and the Internet: Women communicating and men searching, *Sex Roles*, **44**(5/6): 363–379, 2001.
8. L. M. Miller, H. Schweingruber, and C. L. Brandenberg, Middle school students’ technology practices and preferences: Re-examining gender differences, *J. Educ. Multimedia Hypermedia*, **10**(2): 125–140, 2001.
9. A. M. Colley, M. T. Gale, and T. A. Harris, Effects of gender role identity and experience on computer attitude components, *J. Educ. Comput. Res.*, **10**(2): 129–137, 1994.
10. S. E. Mead, P. Batsakes, A. D. Fisk, and A. Mykityshyn, Application of cognitive theory to training and design solutions for age-related computer use, *Int. J. Behav. Develop.***23**(3): 553–573, 1999.
11. S. Davis and R. Bostrom, An experimental investigation of the roles of the computer interface and individual characteristics in the learning of computer systems, *Int. J. Hum. Comput. Interact.*, **4**(2): 143–172, 1992.
12. F. Farina et al., Predictors of anxiety towards computers, *Comput. Hum. Behav.*, **7**(4): 263–267, 1991.
13. B. E. Whitley, Gender differences in computer related attitudes: It depends on what you ask, *Comput. Hum. Behav.*, **12**(2): 275–289, 1996.
14. J. L. Dyck and J. A.-A. Smither, Age differences in computer anxiety: The role of computer experience, gender and education, *J. Educ. Comput. Res.*, **10**(3): 239–248, 1994.
15. D. Passig and H. Levin, Gender interest differences with multimedia learning interfaces, *Comput. Hum. Behav.*, **15**(2): 173–183, 1999.
16. T. Busch, Gender differences in self-efficacy and attitudes toward computers, *J. Educ. Comput. Res.*, **12**(2): 147–158, 1995.
17. L. J. Nelson, G. M. Wiese, and J. Cooper, Getting started with computers: Experience, anxiety and relational style, *Comput. Hum. Behav.*, **7**(3): 185–202, 1991.
18. S. Al-Gahtani and M. King, Attitudes, satisfaction and usage: Factors contributing to each in the acceptance of information technology, *Behav. Inform. Technol.*, **18**(4): 277–297, 1999.
19. A. J. Murrell and J. Sprinkle, The impact of negative attitudes towards computers on employee’s satisfaction and commitment within a small company, *Comput. Hum. Behav.*, **9**(1): 57–63, 1993.
20. M. Staufer, Technological change and the older employee: Implications for introduction and training, *Behav. Inform. Technol.*, **11**(1): 46–52, 1992.
21. A. Kunzer , K. Rose, L. Schmidt, and H. Luczak, SWOF—An open framework for shared workspaces to support different cooperation tasks, *Behav. Inform. Technol.*, **21**(5): 351–358, 2002.
22. J. W. Medcof, The job characteristics of computing and non-computing work activities, *J. Occupat. Organizat. Psychol.*, **69**(2): 199–212, 1996.
23. J. C. Marquie et al., Age influence on attitudes of office workers faced with new computerized technologies: A questionnaire analysis, *Appl. Ergon.*, **25**(3): 130–142, 1994.
24. J. M. Carrol, Human-computer interaction: Psychology as a science of design, *Annu. Rev. Psychol.*, **48**: 61–83, 1997.
25. R. W. Bailey, *Human Performance Engineering: Designing High Quality, Professional User Interfaces for Computer Products, Applications, and Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 1996.
26. P. A. Chalmers, The role of cognitive theory in human-computer interface, *Comput. Hum. Behav.*, **19**(5): 593–607, 2003.
27. D. N. Rapp, H. A. Taylor, and G. R. Crane, The impact of digital libraries on cognitive processes: Psychological issues of hypermedia, *Comput. Hum. Behav.*, **19**(5): 609–628, 2003.
28. A. Dillon, Myths, misconceptions, and an alternative perspective on information usage and the electronic medium, in J. F. Rouetet al. (eds.), *Hypertext and Cognition*. Mahwah, NJ: Lawrence Erlbaum Associates, 1996.
29. A. Dillon and R. Gabbard, Hypermedia as an educational technology: A review of the quantitative research literature on learner expectations, control and style, *Rev. Educ. Res.*, **68**(3): 322–349, 1998.
30. C. Krapichler, M. Haubner, A. Losch, D. Schumann, M. Seeman, K. Englmeier, Physicians in virtual environments—Multimodal human-computer interaction, *Interact. Comput.*, **11**(4): 427–452, 1999.

31. E. Lee, Effects of “gender” of the computer on informational social influence: The moderating role of the task type, *Int. J. Hum.-Comput. Studies*, **58**(4): 347–362, 2003.

32. R. W. Picard and J. Klein, Computers that recognise and respond to user emotion: Theoretical and practical implications, *Interact. Comput.*, **14**(2): 141–169, 2002.

33. R. Kling, Social controversies about computerization, in R. Ming (ed.), *Computerization and Controversy, Value Conflicts and Social Choices*, 2nd ed. New York: Academic Press, 1996.

34. R. Ribak, ‘Like immigrants’: Negotiating power in the face of the home computer, *New Media Soci.*, **3**(2), 220–238, 2001.

35. A. J. Romiszowski and R. Mason, Computer-mediated communication, in D. H. Jonassen (ed.), *Handbook of Research for Educational Communications and Technology*. New York: Simon & Schuster Macmillan, 1996.

36. M. Grabe and C. Grabe, *Integrating Technology for Meaningful Learning*. New York: Houghton Mifflin, 2004.

37. C. Vrasidas and M. S. McIsaac, Factors influencing interaction in an online course, *The Amer. J. Distance Educ.*, **13**(3): 22–36, 1999.

38. National Telecommunications and Information Administration. (February 2002). *A nation online: How Americans are expanding their use of the Internet.* [Online]. Available: http://www.ntia.doc.gov/ntiahome/dn/index.html, Accessed December 10, 2003.

FURTHER READING

Journal Articles

M. J. Alvarez-Torrez, P. Mishra, and Y. Zhao, Judging a book by its cover! Cultural stereotyping of interactive media and its effect on the recall of text information, *J. Educ. Media Hypermedia*, **10**(2): 161–183, 2001.

Y. Amichai-Hamburger, Internet and Personality, *Comput. Hum. Behav.*, **18**(1): 1–10, 2002.

B. Boneva, R. Kraut, and D. Frohlich, Using email for personal relationships, *Amer. Behav. Scientist*, **45**(3): 530–549, 2001.

J. M. Carrol, Human-computer interaction: Psychology as a science of design, *Annu. Rev. Psychol.*, **48**: 61–83, 1997.

E. C. Boling, The Transformation of Instruction through Technology: Promoting inclusive learning communities in teacher education courses, *Action Teacher Educ.*, **24**(4), 2003.

R.A. Davis, A cognitive-behavioral model of pathological Internet use, *Comput. Hum. Behav.*, **17**(2): 187–195, 2001.

C. E. Hmelo, A. Nagarajan, and R. S. Day, Effects of high and low prior knowledge on construction of a joint problem space, *J. Exper. Educ.*, **69**(1): 36–56, 2000.

R. Kraut, M. Patterson, and V. Ludmark, Internet paradox: A social technology that reduces social involvement and psychological well-being, *Amer. Psychol.*, **53**(9), 1017–1031, 1998.

K.Y.A. McKenna and J.A. Bargh, Plan 9 from cyberspace: The implications of the Internet form personality and social psychology, *Personality Social Psychol. Rev.*, **4**: 57–75, 2000.

A. G. Namlu, The effect of learning strategy on computer anxiety, *Comput. Hum. Behav.,* **19**(5): 565–578, 2003.

L. Shashaani, Gender differences in computer experiences and its influence on computer attitudes, *J. Educ. Comput. Res.*, **11**(4): 347–367, 1994.

J. F. Sigurdsson, Computer experience, attitudes towards computers and personality characteristics in psychology undergraduates, *Personality Individual Differences*, **12**(6): 617–624, 1991.

C. A. Steinkuehler, S. J. Derry, C. E. Hmelo-Silver, and M. DelMarcelle, Cracking the resource nut with distributed problem-based learning in secondary teacher education, *J. Distance Educ.*, **23**: 23–39, 2002.

Steven M. Barnhart
Richard De Lisi
Rutgers, The State University of New Jersey
New Brunswick, New Jersey

BIOLOGY COMPUTING

INTRODUCTION

The modern era of molecular biology began with the discovery of the double helical structure of DNA. Today, sequencing nucleic acids, the determination of genetic information at the most fundamental level, is a major tool of biological research (1). This revolution in biology has created a huge amount of data at great speed by directly reading DNA sequences. The growth rate of data volume is exponential. For instance, the volume of DNA and protein sequence data is currently doubling every 22 months (2). One important reason for this exceptional growth rate of biological data is the medical use of such information in the design of diagnostics and therapeutics (3,4). For example, identification of genetic markers in DNA sequences would provide important information regarding which portions of the DNA are significant and would allow the researchers to find many disease genes of interest (by recognizing them from the pattern of inheritance). Naturally, the large amount of available data poses a serious challenge in storing, retrieving, and analyzing biological information.

A rapidly developing area, *computational biology*, is emerging to meet the rapidly increasing computational need. It consists of many important areas such as information storage, sequence analysis, evolutionary tree construction, protein structure prediction, and so on (3,4). It is playing an important role in some biological research. For example, sequence comparison is one of the most important methodological issues and most active research areas in current *biological sequence analysis*. Without the help of computers, it is almost impossible to compare two or more biological sequences (typically, at least a few hundred characters long).

In this chapter, we survey recent results on evolutionary tree construction and comparison, computing syntenic distances between multi-chromosome genomes, and multiple sequence alignment problems.

Evolutionary trees model the evolutionary histories of input data such as a set of species or molecular sequences. Evolutionary trees are useful for a variety of reasons, for example, in homology modeling of (DNA and protein) sequences for diagnostic or therapeutic design, as an aid for devising classifications of organisms, and in evaluating alternative hypotheses of adaption and ancient geographical relationships (for example, see Refs. 5 and 6 for discussions on the last two applications). Quite a few methods are known to construct evolutionary trees from the large volume of input data. We will discuss some of these methods in this chapter. We will also discuss methods for comparing and contrasting evolutionary trees constructed by various methods to find their similarities or dissimilarities, which is of vital importance in computational biology.

Syntenic distance is a measure of distance between multi-chromosome genomes (where each chromosome is viewed as a set of genes). Applications of computing distances between genomes can be traced back to the well-known **Human Genome Project**, whose objective is to decode this entire DNA sequence and to find the location and ordering of genetic markers along the length of the chromosome. These genetic markers can be used, for example, to trace the inheritance of chromosomes in families and thereby to find the location of disease genes. Genetic markers can be found by finding DNA polymorphisms (i.e., locations where two DNA sequences "spell" differently). A key step in finding DNA polymorphisms is the calculation of the *genetic distance*, which is a measure of the correlation (or similarity) between two genomes.

Multiple sequence alignment is an important tool for sequence analysis. It can help extracting and finding biologically important commonalities from a set of sequences. Many versions have been proposed and a huge number of papers have been written on effective and efficient methods for constructing multiple sequence alignment. We will discuss some of the important versions such as *SP-alignment, star alignment, tree alignment, generalized tree alignment*, and *fixed topology alignment with recombination*. Recent results on those versions are given.

We assume that the reader has the basic knowledge of algorithms and computational complexity (such as NP, P, and MAX-SNP). Otherwise, please consult, for example, Refs. 7–9.

The rest of this chapter is organized as follows. In the next section, we discuss construction and comparison methods for evolutionary trees. Then we discuss briefly various distances for comparing sequences and explain in details the syntenic distance measure. We then discuss multiple sequence alignment problems. We conclude with a few open problems.

CONSTRUCTION AND COMPARISON OF EVOLUTIONARY TREES

The evolutionary history of organisms is often conveniently represented as trees, called *phylogenetic trees* or simply *phylogenies*. Such a tree has uniquely labeled leaves and unlabeled interior nodes, can be *unrooted* or *rooted* if the evolutionary origin is known, and usually has internal nodes of degree 3. Figure 1 shows an example of a phylogeny. A phylogeny may also have *weights* on its edges, where an edge weight (more popularly known as *branch length* in genetics) could represent the evolutionary distance along the edge. Many phylogeny reconstruction methods, including the distance and maximum likelihood methods, actually produce weighted phylogenies. Figure 1 also shows a weighted phylogeny (the weights are for illustrative purposes only).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Phylogenetic Construction Methods

Phylogenetic construction methods use the knowledge of evolution of molecules to infer the evolutionary history of the species. The knowledge of evolution is usually in the form of two kinds of data commonly used in phylogeny inference, namely, character matrices (where each position (i, j) is base j in sequence i) and distance matrices (where each position (i,j) contains the computed distance between sequence i and sequence j). Three major types of phylogenetic construction methods are the *parsimony and compatibility method*, the *distance method*, and the *maximum-likelihood method*, Below, we discuss each of these methods briefly. See the excellent survey in Refs. 10 and 11 for more details.

Parsimony methods construct phylogenetic trees for the given sequences such that, in some sense, the total number of changes (i.e., base substitutions) or some weighted sum of the changes is minimized. See Refs. 12–14 for some of the papers in this direction.

Distance methods (15)–(17) try to fit a tree to a matrix of pairwise distances between a set of n species. Entries in the distance matrices are assumed to represent evolutionary distance between species represented by the sequences in the tree (i.e., the total number of mutations in both lineages since divergence from the common ancestor). If no tree fits the distance matrix perfectly, then a measure of the discrepancy of the distances in the distance matrix and those in the tree is taken, and the tree with the minimum discrepancy is selected as the best tree. An example of the measure of the discrepancy, which has been used in the literature (15,16), is a weighted least-square measure, for example, of the form

$$\sum_{1\leq i,j\leq n} w_{ij}(D_{ij} - d_{ij})^2$$

where D_{ij} are the given distances and d_{ij} are the distances computed from the tree.

Maximum-likelihood methods (12,18,19) relies on the statistical method of choosing a tree that maximizes the likelihood (i.e., maximizes the probability) that the observed data would have occurred. Although this method is quite general and powerful, it is computationally intensive because of the complexity of the likelihood function.

All the above methods have been investigated by simulation and theoretical analysis. None of the methods work well under all evolutionary conditions, but each works well under particular situations. Hence, one must choose the appropriate phylogeny construction method carefully for the best results (6).

Comparing Evolutionary Trees

As discussed in the previous section, over the past few decades, many approaches for reconstructing evolutionary trees have been developed, including (not exhaustively) parsimony, compatibility, distance, and maximum-likelihood methods. As a result, in practice, they often lead to different trees on the same set of species (20). It is thus of interest to compare evolutionary trees produced by different methods, or by the same method on different data. Several distance models for evolutionary trees have been proposed in the literature. Among them, the best known is perhaps the *nearest-neighbor interchange* (nni) distance introduced independently in Refs. 21 and 22. Other distances include the *subtree-transfer* distance introduced in Refs. 23 and 24 and the the *linear-cost subtree-transfer distance*(25,26). Below, we discuss very briefly a few of these distances.

Nearest-Neighbor Interchange Distance

An *nni* operation swaps two subtrees that are separated by an internal edge (u, v), as shown in Fig. 2. The nni operation is said to *operate* on this internal edge. The nni distance, $D_{nni}(T_1, T_2)$, between two trees T_1 and T_2 is denned as the minimum number of nni operations required to transform one tree into the other. Culik II and Wood (27) [improved later by Tramp and Zhang (28)] proved that $n\log n + O(n)$ nni moves are sufficient to transform a tree of n leaves to any other tree with the same set of leaves. Sleator et al. (29) proved an $\Omega(n\log n)$ lower bound for most pairs of trees. Although the distance has been studied extensively in the literature (21,22,27–34), the computational complexity of computing it has puzzled the research community for nearly 25 years until recently, when the authors in Ref. 25 showed this problem to be NP-hard an erroneous proof of the NP-hardness of the nni distance between unlabeled trees was published in Ref. 34. As computing the nni distance is shown to be NP-hard, the next obvious question is: *Can we get a good approximation of the distance?* The authors in Ref. 28 show that the nni distance can be approximated in polynomial time within a factor of $\log n + O(1)$.

Subtree-transfer Distances

An nni operation can also be viewed as moving a subtree past a neighboring internal node. A more general operation is to transfer a subtree from one place to another arbitrary place. Figure 3 shows such a *subtree-transfer* operation. The subtree-transfer distance, $D_{st}(T_1, T_2)$, between two trees T_1 and T_2 is the minimum number of subtrees we need to move to transform T_1 into T_2(23–26,35).

It is sometimes appropriate in practice to discriminate among subtree-transfer operations as they occur with different frequencies. In this case, we can charge each subtree-transfer operation a cost equal to the distance (the number of nodes passed) that the subtree has moved in the current tree. The *linear-cost* subtree-transfer distance, $D_{lcst}(T_1, T_2)$, between two trees T_1 and T_2 is then the minimum total cost required to transform T_1 into T_2 by subtree-transfer operations (25,26). Clearly, both subtree-transfer and linear-cost subtree-transfer models can also be used as alternative measures for comparing evolutionary trees generated by different tree reconstruction methods. In fact, on unweighted phylogenies, the linear-cost subtree-transfer distance is identical to the nni distance (26).

The authors in Ref. (35) show that computing the subtree-transfer distance between two evolutionary trees is NP-hard and give an approximation algorithm for this distance with performance ratio 3.

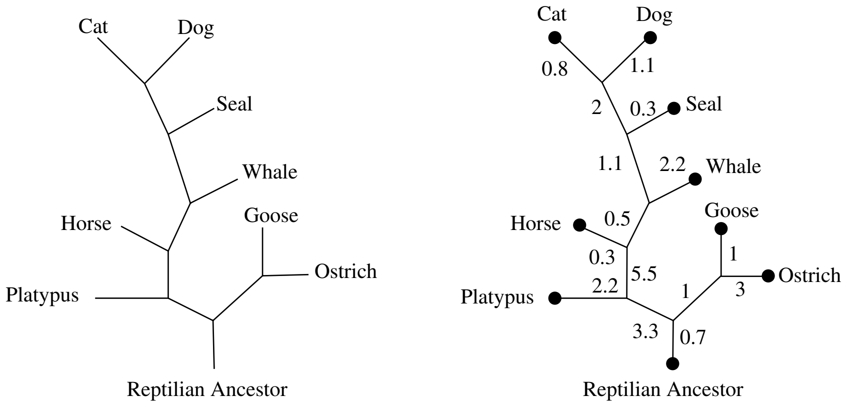


Figure 1. Examples of unweighted and weighted phylogenies.

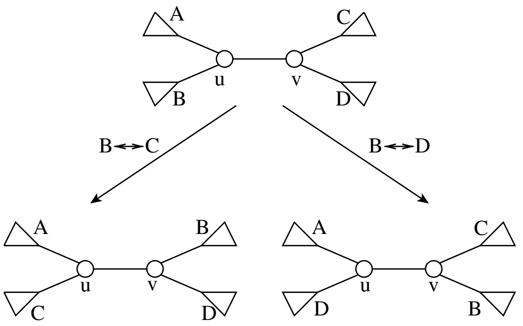


Figure 2. The two possible nni operations on an internal edge (u, v): exchange $B \leftrightarrow C$ or $B \leftrightarrow D$.

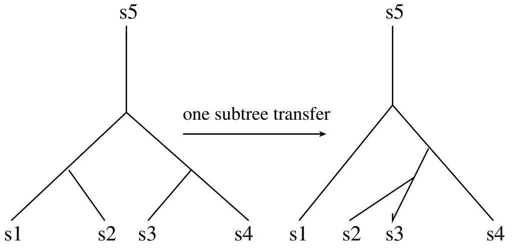


Figure 3. An example of a subtree-transfer operation on a tree.

Rotation Distance

Rotation distance is a variant of the nni distance for rooted, ordered trees. A *rotation* is an operation that changes one rooted binary tree into another with the same size. Figure 4 shows the general rotation rule. An easy approximation algorithm for computing distance with a performance ratio of 2 is given in Ref. 36. However, it is not known if computing this distance is NP-hard or not.

Distances on Weighted Phylogenies

Comparison of weighted evolutionary trees has recently been studied in Ref. 20. The distance measure adopted is based on the difference in the partitions of the leaves induced by the edges in both trees, and it has the drawback of being somewhat insensitive to the tree topologies. Both the linear-cost subtree-transfer and nni models can be naturally extended to weighted trees. The extension for nni is straightforward: An nni is simply charged a cost equal to the weight of the edge it operates on. In the case of linear-cost subtree-transfer, although the idea is immediate (i.e., a moving subtree should be charged for the weighted distance it travels), the formal definition needs some care and can be found in Ref. 26.

As computing the nni distance on unweighted phylogenies is NP-hard, it is obvious that computing this distance is NP-hard for weighted phylogenies also. The authors in Ref. 26 give an approximation algorithm for the linear-cost subtree-transfer distance on weighted phylogenies with performance ratio 2. In Ref. 25, the authors give an approximation algorithm for the nni distance on weighted phylogenies with performance ratio of $O(\log n)$. It is open whether the linear-cost subtree-transfer problem is NP-hard for weighted phylogenies. However, it has been shown that the problem is NP-hard for weighted trees with non-uniquely labeled leaves (26).

COMPUTING DISTANCES BETWEEN GENOMES

The definition and study of appropriate measures of distance between pairs of species is of great importance in

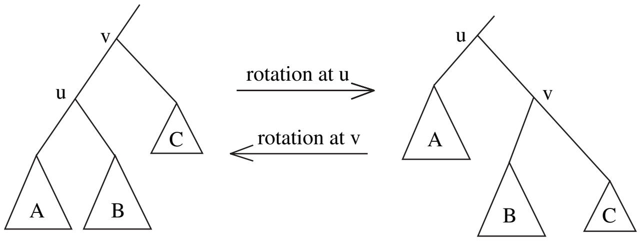


Figure 4. Left and right rotation operations on a rooted binary tree.

computational biology. Such measures of distance can be used, for example, in phylogeny construction and in taxonomic analysis.

As more and more molecular data become available, methods for denning distances between species have focused on such data. One of the most popular distance measures is the edit distance between homologous DNA or amino acid sequences obtained from different species. Such measures focus on point mutations and define the distance between two sequences as the minimum number of these moves required to transform one sequence into another. It has been recognized that the edit-distance may underestimate the distance between two sequences because of the possibility that multiple point mutations occurring at the same locus will be accounted for simply as one mutation. The problem is that the probability of a point mutation is not low enough to rule out this possibility.

Recently, there has been a spate of new definitions of distance that try to treat rarer, macro-level mutations as the basic moves. For example, if we know the order of genes on a chromosome for two different species, we can define the *reversal* distance between the two species to be the number of reversals of portions of the chromosome to transform the gene order in one species to the gene order in the other species. The question of finding the reversal distance was first explored in the computer science context by Kececioglu and Sankoff and by Bafna and Pevzner, and there has been significant progress made on this question by Bafna, Hannenhalli, Kececioglu, Pevzner, Ravi, Sankoff, and others (37–41). Other moves besides reversals have been considered as well. Breaking off a portion of the chromosome and inserting it elsewhere in the chromosome is referred to as a *transposition*, and one can similarly define the transposition distance (42). Similarly, allowing two chromosomes (viewed as strings of genes) to exchange suffixes (or sometimes a suffix with a prefix) is known as a *translocation*, and this move can also be used to define an appropriate measure of distance between two species for which much of the genome has been mapped (43).

Ferretti et al. (44) proposed a distance measure that is at an even higher level of abstraction. Here, even the order of genes on a particular chromosome of a species is ignored or presumed to be unknown. It is assumed that the genome of a species is given as a collection of sets. Each set in the collection corresponds to a set of genes that are on one chromosome and different sets in the collection correspond to different chromosomes (see Fig. 5). In this scenario, one can define a move to be an exchange of genes between two chromosomes, the fission of one chromosome into two, or the fusion of two chromosomes into one (see Fig. 6). The *syntenic distance* between two species has been defined by Ferretti et al. (44) to be the number of such moves required to transform the genome of one species to the genome of the other.

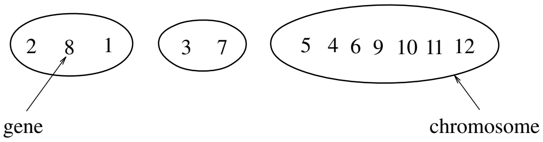


Figure 5. A genome with 12 genes and 3 chromosomes.

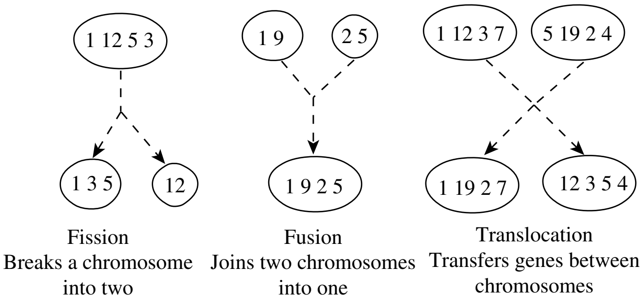

1 12 5 3
1 3 5
12
Fission
Breaks a chromosome into two
1 9
2 5
1 9 2 5
Fusion
Joins two chromosomes into one
1 12 3 7
5 19 2 4
1 19 2 7
12 3 5 4
Translocation
Transfers genes between chromosomes

Figure 6. Different mutation operations.

Notice that any recombination of two chromosomes is permissible in this model. By contrast, the set of legal translocations (in the translocation distance model) is severely limited by the order of genes on the chromosomes being translocated. Furthermore, the transformation of the first genome into the second genome does not have to produce a specified order of genes in the second genome. The underlying justification of this model is that the exchange of genes between chromosomes is a much rarer event than the movement of genes within a chromosome and, hence, a distance function should measure the minimum number of such exchanges needed.

In Ref. 45, the authors prove various results on the syntenic distance. For example, they show that computing the syntenic distance exactly is NP-hard, there is a simple polynomial time approximation algorithm for the synteny problem with performance ratio 2, and computing the syntenic distance is fixed parameter tractable.

The median problem develops in connection with the phylogenetic inference problem (44) and denned as follows: Given three genomes $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$, we are required to construct a genome $\mathcal{G}$ such that the *median distance* $\alpha_{\mathcal{G}} = \sum_{i=1}^{3} D(\mathcal{G}, \mathcal{G}_i)$ is minimized (where D is the syntenic distance). Without any additional constraints, this problem is trivial, as we can take $\mathcal{G}$ to be empty (and then $\alpha_{\mathcal{G}} = 0$). In the context of syntenic distance, any one of the following three constraints seem relevant: (cl) $\mathcal{G}$ must contain all genes present in *all the three* given genomes, (c2) $\mathcal{G}$ must contain all genes present in *at least two* of the three given genomes, and (c3) $\mathcal{G}$ must contain all genes present in *at least one* of the three given genomes. Then, computing the median genome is NP-hard with any one of the three constraints (cl), (c2), or (c3). Moreover, one can approximate the median problem in polynomial time [under any one of the constraints (cl), (c2), or (c3)] with a constant performance ratio. See Ref. 45 for details.

MULTIPLE SEQUENCE ALIGNMENT PROBLEMS

Multiple sequence alignment is the most critical cutting-edge tool for sequence analysis. It can help extracting, finding, and representing biologically important commonalities from a set of sequences. These commonalities could represent some highly conserved subregions, common functions, or common structures. Multiple sequence alignment is also very useful in inferring the evolutionary history of a family of sequences (46–49).

A *multiple alignment*$\mathcal{A}$ of $k \geq 2$ sequences is obtained as follows: Spaces are inserted into each sequence so that the resulting sequences $s'_i (i = 1, 2, \ldots k)$ have the same length l, and the sequences are arranged in k rows of l columns each.

The value of the multiple alignment $\mathcal{A}$ is denned as

$$\sum_{i=1}^{l} \mu(s'_1(i), s'_2(i), \ldots s'_k(i))$$

where $s'_l(i)$ denotes the ith letter in the resulting sequence s'_l, and $\mu(s'_1(i), s'_2(i), \ldots s'_k(i))$ denotes the score of the ith column. The multiple sequence alignment problem is to construct a multiple alignment minimizing its value.

Many versions have been proposed based on different objective functions. We will discuss some of the important ones.

SP-Alignment and Steiner Consensus String

For *SP-score* (Sum-of-the-Pairs), the score of each column is denned as:

$$\mu(s'_1(i), s'_2(i), \ldots s'_k(i)) = \sum_{1 \leq j < l \leq k} \mu(s'_j(i), s'_l(i))$$

where $\mu(s'_j(i), s'_l(i))$ is the score of the two opposing letters $s'_j(i)$ and $s'_l(i)$. The SP-score is sensible and has previously been studied extensively.

SP-alignment problem is to find an alignment with the smallest SP-score. It is first studied in Ref. 50 and subsequently used in Refs. 51–54. SP-alignment problem can be solved exactly by using dynamic programming. However, if there are k sequences and the length of sequences is n, it takes $O(n^k)$ time. Thus, it works for only small numbers of sequences. Some techniques to reduce the time and space have been developed in Refs. 51 and 55–57. With these techniques, it is possible to optimally align up to six sequences of 200 characters in practice.

In fact, SP-alignment problem is NP-hard (58). Thus, it is impossible to have a polynomial time algorithm for this problem. In the proof of NP-hardness, it is assumed that some pairs of identical characters have a non-zero score. An interesting open problem is if each pair of two identical characters is scored 0.

The first approximation algorithm was given by Gusfield (53). He introduced the *center star* algorithm. The center star algorithm is very simple and efficient. It selects a sequence (called *center string*) s_c in the set of k given sequences S such that $\sum_{i=1}^{k} dist(s_c, s_i)$ is minimized. It then optimally aligns the sequences in $S - \{s_c\}$ to s_c and gets $k - 1$ pairwise alignments. These $k - 1$ pairwise alignments lead to a multiple alignment for the k sequences in S. If the score scheme for pairs of characters satisfies the triangle inequality, the cost of the multiple alignment produced by the center star algorithm is at most twice the optimum (47,53). Some improved results were reported in Refs. 54 and 59.

Another score called *consensus* score is denned as follows:

$$\mu(s'_1(i), s'_2(i), \ldots s'_k(i)) = \min_{s \in \Sigma} \sum_{j=1}^{k} \mu(s'_j(i), s)$$

where $\sum$ is the set of characters that form the sequences. Here, we reconstruct a character for each column and thus obtain a string. This string is called a *Steiner consensus string* and can be used as a representative for the set of given sequences. The problem is called the *Steiner consensus string* problem.

The Steiner consensus string problem was proved to be NP-complete (60) and MAX SNP-hard (58). In the proof of MAX SNP-hardness, it is assumed that there is a "wild card," and thus the triangle inequality does not hold. Combining with the results in Ref. 61, it shows that there is no polynomial time approximation scheme for this problem. Interestingly, the same center star algorithm also has performance ratio 2 for this problem (47).

Diagonal Band Alignment. The restriction of aligning sequences within a constant diagonal band is often used in practical situations. Methods under this assumption have been extensively studied too. Sankoff and Kruskal discussed the problem under the rubric of "cutting corners" in Ref. 62. Alignment within a band is used in the final stage of the well-known FASTA program for rapid searching of protein and DNA sequence databases (63,64). Pearson showed that alignment within a band gives very good results for many protein superfamilies (65).

Other references on the subject can be found in Refs. 51 and 66–69. Spouge gives a survey on this topic in Ref. 70.

Let $\mathcal{S} = \{s_1, s_2, \ldots, s_k\}$ be a set of k sequences, each of length m (for simplicity), and $\mathcal{M}$ an alignment of the k sequences. Let the length of the alignment $\mathcal{M}$ be M. $\mathcal{M}$ is called a *c-diagonal alignment* if for any $p \leq m$ and $1 < i < j < k$, if the pth letter of s_i is in column q of $\mathcal{M}$ and the pth letter of s_j is in column r of $\mathcal{M}$, then $|q - r| \leq c$. In other words, the inserted spaces are "evenly" distributed among all sequences and the ith position of a sequence is about at most c positions away from the ith. position of any other sequence.

In Ref. 71 Li, et al. presented polynomial time approximation schemes of c-diagonal alignment for both SP-score and consensus score.

Tree Alignment

Tree score: To define the score $\mu(s'_1(i), s'_2(i), \ldots s'_k(i))$ of the ith. column, an *evolutionary* (or *phylogenetic*) tree $T = (V, E)$ with k leaves is assumed, each leaf j corresponding to a sequence S_j. (Here, V and E denote the sets of nodes and edges in T, respectively.) Let $k + 1, K + 2, \ldots, k + m$ be the internal nodes of T. For each internal node j, reconstruct a letter (possibly a space) $s'_j(i)$ such that $\sum_{(p,q) \in E} \mu(s'_p(i), s'_q(i))$ is minimized. The score

$\mu(s'_1(i), s'_2(i), \ldots s'_k(i))$ of the ith column is thus denned as

$$\mu(s'_1(i), s'_2(i), \ldots s'_k(i)) = \sum_{(p,q)\in E} \mu\left(s'_p(i), s'_q(i)\right)$$

This measure has been discussed in Refs. 14,48,51,59 and 72. Multiple sequence alignment with tree score is often referred to as *tree alignment* in the literature.

Note that a tree alignment induces a set of *reconstructed* sequences, each corresponding to an internal node. Thus, it is convenient to reformulate a tree alignment as follows: Given a set X of k sequences and an evolutionary tree T with k leaves, where each leaf is associated with a given sequence, reconstruct a sequence for each internal node to minimize the *cost* of T. Here, the cost of T is the sum of the edit distance of each pair of (given or reconstructed) sequences associated with an edge. Observe that once a sequence for each internal node has been reconstructed, a multiple alignment can be obtained by optimally aligning the pair of sequences associated with each edge of the tree. Moreover, the tree score of this induced multiple alignment equals the cost of T. In this sense, the two formulations of tree alignment are equivalent.

Sankoff gave an exact algorithm for tree alignment that runs in $O(n^k)$, where n is the length of the sequences and k is the number of given sequences. Tree alignment was proved to be NP-hard (58).

Therefore, it is unlikely to have a polynomial time algorithm for tree alignment. Some heuristic algorithms have also been considered in the past. Altschul and Lipman tried to cut down the computation volume required by dynamic programming (51). Sankoff, et al. gave an iterative improvement method to speed up the computation (48,72). Waterman and Perlwitz devised a heuristic method when the sequences are related by a binary tree (73). Hein proposed a heuristic method based on the concept of a *sequence graph* (74,75). Ravi and Kececioglu designed an approximation algorithm with performance ratio $\frac{deg+1}{deg-1}$ when the given tree is a *regular deg-ary* tree (i.e., each internal node has exactly *deg* children) (76).

The first approximation algorithm with a guaranteed performance ratio was devised by Wang, et al. (77). A ratio-2 algorithm was given. The algorithm was then extended to a polynomial time approximation scheme (PTAS) (i.e., the performance ratio could arbitrarily approach 1). The PTAS requires computing exact solutions for depth-t subtrees. For a fixed t, the performance ratio was proved to be $1+\frac{3}{t}$, and the running time was proved to be $O((k/deg^t)^{deg^{t-1}+2}M(2, t-1, n))$, where *deg* is the degree of the given tree, n is the length of the sequences, and $M(deg, t-1, n)$ is the time needed to optimally align a tree with $deg^{t-1}+1$ leaves, which is upper-bounded by $O(n^{deg^{t-1}+1})$. Based on the analysis, to obtain a performance ratio less than 2, exact solutions for depth-4 subtrees must be computed, and thus optimally aligning nine sequences at a time is required, which is impractical even for sequences of length 100.

An improved version was given in Ref. 78. They proposed a new PTAS for the case where the given tree is a regular deg-ary tree. The algorithm is much faster than the one in Ref. 77. The algorithm also must do local optimizations for depth-t subtrees. For a fixed t, the performance ratio of the new PTAS is $1+\frac{2}{t}-\frac{2}{t2^t}$ and the running time is $O(\min\{2^t, k\}kdM(deg, t-1, n))$, where d is the depth of the tree. Presently, there are efficient programs (72) to do local optimizations for three sequences ($t = 2$). In fact, we can expect to obtain optimal solutions for five sequences ($t = 3$) of length 200 in practice as there is such a program (55,56) for SP-score and similar techniques can be used to attack the tree alignment problem. Therefore, solutions with costs at most 1.583 times the optimum can be obtained in practice for strings of length 200.

For tree alignment, the given tree is typically a binary tree. Recently, Wang et al. (79) designed a PTAS for binary trees. The new approximation scheme adopts a more clever partitioning strategy and has a better time efficiency for the same performance ratio. For any fixed r, where $r = 2^{t-1}+1-q$ and $0 \le q \le 2^{t-2}-1$, the new PTAS runs in time $O(kdn^r)$ and achieves an approximation ratio of $1+\frac{2^{t-1}}{2^{t-2}(t+1)-q}$. Here, the parameter r represents the "size" of local optimization. In particular, when $r = 2^{t-1}+1$, its approximation ratio is simply $1+\frac{2}{t+1}$.

Generalized Tree Alignment

In practice, we often face a more difficult problem called *generalized tree alignment*. Suppose we are given a set of sequences. The problem is to construct an evolutionary tree as well as a set of sequences (called reconstructed sequences) such that each leaf of the evolutionary tree is assigned a given sequence, each internal node of the tree is assigned a reconstructed sequence, and the cost of the tree is minimized over all possible evolutionary trees and reconstructed sequences.

Intuitively, the problem is harder than tree alignment because the tree is not given and we have to compute the tree structure as well as the sequences assigned to internal nodes. In fact, the problem was proved to be MAX SNP-hard (58) and a simplified proof was given in Ref. 80. It implies that it is impossible to have a PTAS for generalized tree alignment unless P=NP (61), which confirms the observation from an approximation point of view.

Generalized tree alignment problem is, in fact, the Steiner tree problem in sequence spaces. One might use the approximation algorithms with guaranteed performance ratios (81) to graph Steiner trees, which, however, may lead to a tree structure where a given sequence is an internal node. Sometimes, it is unacceptable. Schwikowski and Vingron give a method that combines clustering algorithms and Hein's sequence graph method (82). The produced solutions contain biologically reasonable trees and keep the guaranteed performance ratio.

Fixed Topology History/Alignment with Recombination

Multigene families, viruses, and alleles from within populations experience recombinations (23,24,83,84). When recombination happens, the ancestral material on the present sequence s_1 is located on two sequences s_2 and s_3. s_2 and s_3 can be cut at k locations (break points) into $k+1$ pieces, where $s_2 = s_{2,1}s_2 \ldots s_{2,l+1}$ and $s_3 = s_{3,1}s_{3,2} \ldots s_{3,l+1}$. s_1 can be represented as $s_{\hat{2},1} = s_{\hat{3},2}s_{\hat{2},3} \ldots s_{\hat{2},i}s_{3,\hat{i}+1} \ldots$, where subsequences $s_{\hat{2},i}$ and $s_{3,\hat{i}+1}$ differ from the corresponding $s_{2,i}$ and $s_{3,i+1}$ by insertion, deletion, and substitution of letters. k, the number of times s_1 switches between s_2 and s_3, is called the number of *crossovers*. The cost of the recombination is

$$dist(s_{1,1}, s_{\hat{1},1}) + dist(s_{2,2}, s_{\hat{2},2}), \ldots dist(s_{1,i}, s_{\hat{1},i}) + dist \times (s_{2,i+1}, s_{2,\hat{i}+1}) + \cdots + k\chi$$

where $dist(s_{2,i+1}, s_{2,\hat{i}+1})$ is the edit distance between the two sequences $s_{2,i+1}$ and $s_{2,\hat{i}+1}$, k is the number of crossovers, and χ the crossover penalty. The *recombination* distance to produce s_1 from s_2 and s_3 is the cost of a recombination that has the smallest cost among all possible recombinations. We use $r\_dist(s_1, s_2, s_3)$ to denote the recombination distance. For more details, see Refs. 83 and 85.

When recombination occurs, the given topology is no longer a binary tree. Instead, some nodes, called *recombination nodes*, in the given topology may have two parents (23,24). In a more general case, as described in Ref. 83, the topology may have more than one root. The set of roots is called a *pro-toset*. The edges incident to recombination nodes are called *recombination* edges [see Fig. 7 (b)]. A node/edge is *normal* if it is not a recombination node/edge.

The cost of a pair of recombination edges is the recombination distance to produce the sequence on the recombination node from the two sequences on its parents. The cost of other normal edges is the edit distance between two sequences. A topology is *fully labeled* if every node in the topology is labeled. For a fully labeled topology, the cost of the topology is the total cost of edges in the topology. Each node in the topology with degree greater than 1 is an internal node. Each leaf/terminal (degree 1 node) in the topology is labeled with a given sequence. The goal here is to construct a sequence for each internal node such that the cost of the topology is minimized. We call this problem *fixed topology history with recombination* (FTHB).

Obviously, this problem is a generalization of tree alignment. The difference is that the given topology is no longer a binary tree. Instead, there are some recombination nodes that have two parents instead of one. Moreover, there may be more than one root in the topology.

A different version called *fixed topology alignment with recombination* (FTAR) is also dicsussed (86). From an approximation point of view, FTHR and FTAR are much harder than tree alignment. It is shown that FTHR and FTAR cannot be approximated within any constant performance ratio unless $P = NP$ (86).

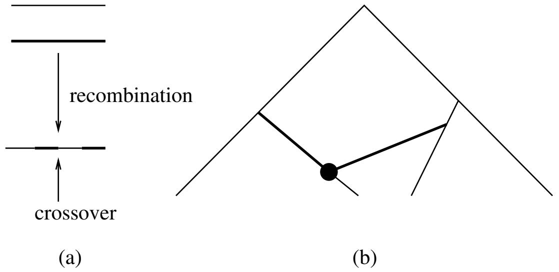


Figure 7. (a) Recombination operation, (b) The topology. The dark edges are recombination edges. The circled node is a recombination node.

A more restricted case, where each internal node has at most one recombination child and there are at most six parents of recombination nodes in any path from the root to a leaf in the given topology, is also considered. It is shown that the restricted version for both FTHR and FTAR is MAX-SNP-hard. That is, there is no polynomial time approximation scheme unless $P = NP$ (86).

The above hardness results are disappointing. However, recombination occurs infrequently. So, it is interesting to study some restricted cases. A *merge node* of recombination node v is the lowest common ancestor of v's two parents. The two different paths from a recombination node to its merge node are called *merge paths*. We then study the case, where

- (**C1**) each internal node has at most one recombination child and
- (**C2**) any two merge paths for different recombination nodes do not share any common node.

Using a method similar to the lifting method for tree alignment, one can get a ratio-3 approximation algorithm for both FTHR and HTAR when the given topology satisfies (**C1**) and (**C2**). The ratio-3 algorithm can be extended to a PTAS for FTAR with bounded number of crossovers (see Ref. 86).

Remarks: Hein might be the first to study the method to reconstruct the history of sequences subject to recombination (23,24). Hein observed that the evolution of a sequence with k recombinations could be described by k recombination points and $k + 1$ trees describing the evolution of the $k + 1$ intervals, where two neighboring trees were either identical or differed by one subtree transfer operation (23–26,35). A heuristic method was proposed to find the most parsimonious history of the sequences in terms of mutation and recombination operations.

Another strike was given by Kececioglu and Gusfield (83). They introduced two new problems, *recombination distance* and *bottleneck recombination history*. They tried to include higher-order evolutionary events such as block insertions and deletions (68) as well as tandem repeats (87,88).

CONCLUSION

In this article, we have discussed some important topics in the field of computational biology such as the phylogenetic construction and comparison methods, syntenic distance between genomes, and the multiple sequence alignment problems. Given the vast majority of topics in computational biology, these discussed topics constitute only a part of them. Some of the important topics that were *not* covered in this articls are as follows:

- protein structure prediction,
- DNA physical mapping problems,

- metabolic modeling, and
- string/database search problems.

We hope that this survey article will inspire the readers for further study and research of these and other related topics.

Papers on compuational molecular biology have started to appear in many different books, journals, and conferences. Below, we list some sources that could serve as excellent starting points for various problems that occur in computational biology:

Books: See Refs. 49,53,62 and 89–92.

Journals: Computer Applications in the Biosciences (recently renamed as Bioinformatics), Journal of Computational Biology, Bulletin of Mathematical Biology, Journal of Theoretical Biology, and so on.

Conferences: Annual Symposium on Combinatorial Pattern Matching (CPM), Pacific Symposium on Biocomputing (PSB), Annual International Conference on Computational Molecular Biology (RECOMB), Annual Conference on Intelligent Systems in Molecular Biology (ISMB), and so on.

Web pages: `http://www.cs.Washington.edu/education/courses/590bi`, `http://www.cse.ucsc.edu/research/compbio`, `http://www.cs.jhu.edu/salzberg/cs439. html`, and so on.

ACKNOWLEDGMENTS

We thank Prof. Tao Jiang for bringing the authors together. We also thank Dr. Todd Wareham for carefully reading the draft and giving valuable suggestions. Bhaskar DasGupta's work was partly supported by NSF grants CCR-0296041, CCR-0206795, and CCR-0208749. Lusheng Wang's work was fully supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 1070/02E).

BIBLIOGRAPHY

1. M. S. Waterman, Sequence alignments, in M. S. Waterman (ed.), *Mathematical Methods for DNA Sequences*, Boca Raton, FL: CRC Press, 1989, pp. 53–92.
2. W. Miller, S. Scbwartz, and R. C. Hardison, A point of contact between computer science and molecular biology, *IEEE Computat. Sci. Eng.*, 69–78, 1994.
3. K. A. Frenkel, The human genome project and informatics, *Commun. ACM*, **34**, (11): 41–51, 1991.
4. E. S. Lander, R. Langridge, and D. M. Saccocio, Mapping and interpreting biological information, *Commun. ACM*, **34**, (11): 33–39, 1991.
5. V. A. Funk, and D. R. Brooks, *Phylogenetic Systematics as the Basis of Comparative Biology*, Washington, DC: Smithsonian Institution Press, 1990.
6. D. M. Hillis, B. K. Mable, and C. Moritz, Applications of molecular systematics, in D. M. Hillis, C. Moritz, and B. K. Mable (eds.), *Molecular Systematics*, (2nd ed.) Sunderland, MA: Sinauer Associates, 1996, pp. 515–543.
7. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NF'-completeness*, New York: W. H. Freeman, 1979.
8. D. Hochbaum, *Approximation Algorithms for NP-hard Problems*, Boston, MA: PWS Publishers, 1996.
9. C. H. Papadimitriou, *Computational Complexity*, Reading, MA: Addison-Wesley, 1994.
10. J. Felsenstein, Phylogenies from molecular sequences: inferences and reliability, *Annu. Rev. Genet.*, **22**: 521–565, 1988.
11. D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M. Hillis, Phylogenetic inference, in D. M. Hillis, C. Moritz, and B. K. Mable (eds.), *Molecular Systematics*, (2nd ed.), Sunderland, MA: Sinauer Associates, 1996, pp. 407–514.
12. A. W. F. Edwards and L. L. Cavalli-Sforza, The reconstruction of evolution, *Ann. Hum. Genet*, **27**: 105, 1964, (also in *Heredity* **18**: 553, 1964).
13. W. M. Fitch, Toward denning the course of evolution: minimum change for a specified tree topology, *Syst Zool.*, **20**: 406–416, 1971.
14. D. Sankoff, Minimal mutation trees of sequences, *SIAM J. Appl. Math.*, **28**: 35–42, 1975.
15. L. L. Cavalli-Sforza and A. W. F. Edwards, Phylogenetic analysis: models and estimation procedures, *Evolution*, **32**: 550–570, 1967, (also published in *Am. J. Hum. Genet.*, **19**: 233–257, 1967.)
16. W. M. Fitch and E. Margoliash, Construction of phylogenetic trees, *Science*, **155**: 279–284, 1967.
17. N. Saitou and M. Nei, The neighbor-joining method: a new method for reconstructing phylogenetic trees, *Mol. Biol. Evol.*, **4**: 406–425, 1987.
18. D. Barry and J. A. Hartigan, Statistical analysis of hominoid molecular evolution, *Stat. Sci.*, **2**: 191–210, 1987.
19. J. Felsenstein, Evolutionary trees for DNA sequences: a maximum likelihood approach, *J. Mol. Evol.*, **17**: 368–376, 1981.
20. M. Kuhner and J. Felsenstein, A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Mol. Biol. Evol* **11** (3): 459–468, 1994.
21. D. F. Robinson, Comparison of labeled trees with valency three, *J. Combinatorial Theory, Series B*, **11**: 105–119, 1971.
22. G. W. Moore, M. Goodman, and J. Barnabas, An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets, *J. Theoret. Biol.*, **38**: 423–457, 1973.
23. J. Hein, Reconstructing evolution of sequences subject to recombination using parsimony, *Math. Biosci.*, **98**: 185–200, 1990.
24. J. Hein, A heuristic method to reconstruct the history of sequences subject to recombination, *J. Mol. Evol.*, **36**: 396–405, 1993.
25. B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang, On distances between phylogenetic trees, *Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997, pp. 427–436.
26. B. DasGupta, X. He, T. Jiang, M. Li, and J. Tromp, On the linear-cost subtree-transfer distance, to appear in the special issue in *Algorithmica* on computational biology, 1998.
27. K. Culik II and D. Wood, A note on some tree similarity measures, *Information Proc. Lett.*, **15**: 39–42, 1982.
28. M. Li, J. Tromp, and L. X. Zhang, On the nearest neighbor interchange distance between evolutionary trees, *J. Theoret. Biol.*, **182**: 463–467, 1996.

29. D. Sleator, R. Tarjan, W. Thurston, Short encodings of evolving structures, *SIAM J. Discr. Math.*, **5**: 428–450, 1992.

30. M. S. Waterman and T. F. Smith, On the similarity of dendrograms, *J. Theoret. Biol.*, **73**: 789–800, 1978.

31. W. H. E. Day, Properties of the nearest neighbor interchange metric for trees of small size, *J. Theoretical Biol.*, **101**: 275–288, 1983.

32. J. P. Jarvis, J. K. Luedeman, and D. R. Shier, Counterexamples in measuring the distance between binary trees, *Math. Social Sci.*, **4**: 271–274, 1983.

33. J. P. Jarvis, J. K. Luedeman, and D. R. Shier, Comments on computing the similarity of binary trees, *J. Theoret. Biol.*, **100**: 427–433, 1983.

34. M. Křivánek, Computing the nearest neighbor interchange metric for unlabeled binary trees is NP-complete, *J. Classification*, **3**: 55–60, 1986.

35. J. Hein, T. Jiang, L. Wang, and K. Zhang, On the complexity of comparing evolutionary trees, *Discrete Appl. Math.*, **7**: 153–169, 1996.

36. D. Sleator, R. Tarjan, and W. Thurston, Rotation distance, triangulations, and hyperbolic geometry, *J. Amer. Math. Soc.*, **1**: 647–681, 1988.

37. V. Bafna and P. Pevzner, Genome rearrangements and sorting by reversals, *34th IEEE Symp. on Foundations of Computer Science*, 1993, pp. 148–157.

38. V. Bafna, and P. Pevzner, Sorting by reversals: genome rearrangements in plant organelles and evolutionary history of X chromosome, *Mol. Biol. Evol*, **12**: 239–246, 1995.

39. S. Hannenhalli and P. Pevzner, Transforming Cabbage into Turnip (polynomial algorithm for sorting signed permutations by reversals), *Proc. of 27th Ann. ACM Symp. on Theory of Computing*, 1995, pp. 178–189.

40. J. Kececioglu and D. Sankoff, Exact and approximation algorithms for the inversion distance between two permutations, *Proc. of 4th Ann. Symp. on Combinatorial Pattern Matching*, Lecture Notes in *Comp. Sci.*, **684**: 87–105, 1993.

41. J. Kececioglu, and D. Sankoff, Efficient bounds for oriented chromosome inversion distance, *Proc. of 5th Ann. Symp. on Combinatorial Pattern Matching*, Lecture Notes in *Comp. Sci.*, **807**: 307–325, 1994.

42. V. Bafna, and P. Pevzner, Sorting by transpositions, *Proc. of 6th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1995, pp. 614–623.

43. J. Kececioglu and R. Ravi, Of mice and men: evolutionary distances between genomes under translocation, *Proc. of 6th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1995, pp. 604–613.

44. V. Ferretti, J. H. Nadeau, and D. Sankoff, Original synteny, in *Proc. of 7th Ann. Symp. on Combinatorial Pattern Matching*, 1996, pp. 159–167.

45. B. DasGupta, T. Jiang, S. Kannan, M. Li, and E. Sweedyk, On the complexity and approximation of syntenic distance, *1st Annual International Conference On Computational Molecular Biology*, 1997, pp. 99–108 (journal version to appear in *Discrete and Applied Mathematics*).

46. S. C. Chan, A. K. C. Wong, and D. K. T. Chiu, A survey of multiple sequence comparison methods, *Bull. Math. Biol.*, **54**, (4): 563–598, 1992.

47. D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge, UK: Cambridge University Press, 1997.

48. D. Sankoff, and R. Cedergren, Simultaneous comparisons of three or more sequences related by a tree, in D. Sankoff, and J. Kruskal (eds.), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, Reading, MA: Addison Wesley, 1983, pp. 253–264.

49. M. S. Waterman, *Introduction to Computational Biology: Maps, Sequences, and Genomes*, London: Chapman and Hall, 1995.

50. H. Carrillo and D. Lipman, The multiple sequence alignment problem in biology, *SIAM J. Appl. Math.*, **48**: 1073–1082, 1988.

51. S. Altschul, and D. Lipman, Trees, stars, and multiple sequence alignment, *SIAM J. Applied Math.*, **49**: 197–209, 1989.

52. D. Baconn, and W. Anderson, Multiple sequence alignment, *J. Molec. Biol.*, **191**: 153–161, 1986.

53. D. Gusfield, Efficient methods for multiple sequence alignment with guaranteed error bounds, *Bull. Math. Biol.*, **55**: 141–154, 1993.

54. P. Pevzner, Multiple alignment, communication cost, and graph matching, *SIAM J. Appl. Math.*, **56** (6): 1763–1779, 1992.

55. S. Gupta, J. Kececioglu, and A. Schaffer, Making the shortest-paths approach to sum-of-pairs multiple sequence alignment more space efficient in practice, *Proc. 6th Symposium on Combinatorial Pattern Matching, Springer LNCS937*, 1995, pp. 128–143.

56. J. Lipman, S. F. Altschul, and J. D. Kececioglu, A tool for multiple sequence alignment, *Proc. Nat. Acid Sci. U.S.A.*, **86**: 4412–4415, 1989.

57. G. D. Schuler, S. F. Altschul, and D. J. Lipman, A workbench for multiple alignment construction and analysis, in *Proteins: Structure, function and Genetics*, in press.

58. L. Wang and T. Jiang, On the complexity of multiple sequence alignment, *J. Computat. Biol.*, **1**: 337–348, 1994.

59. V. Bafna, E. Lawer, and P. Pevzner, Approximate methods for multiple sequence alignment, *Proc. 5th Symp. on Combinatorial Pattern Matching. Springer LNCS 807*, 1994, pp. 43–53.

60. E. Sweedyk, and T. Warnow, The tree alignment problem is NP-complete, *Manuscript*.

61. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, On the intractability of approximation problems, *33rd IEEE Symposium on Foundations of Computer Science*, 1992, pp. 14–23.

62. D. Sankoff and J. Kruskal (eds.). *Time Warps, String Edits, and Macro-Molecules: The Theory and Practice of Sequence Comparison*, Reading, MA: Addison Wesley, 1983.

63. W. R. Pearson and D. Lipman, Improved tools for biological sequence comparison, *Proc. Natl. Acad. Sci. USA*, **85**: 2444–2448, 1988.

64. W. R. Pearson, Rapid and sensitive comparison with FASTP and FASTA, *Methods Enzymol.*, **183**: 63–98, 1990.

65. W. R. Pearson, Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms, *Genomics*, **11**: 635–650, 1991.

66. K. Chao, W. R. Pearson, and W. Miller, Aligning two sequences within a specified diagonal band, *CABIOS*, **8**: 481–487, 1992.

67. J. W. Fickett, Fast optimal alignment, *Nucleic Acids Res.*, **12**: 175–180, 1984.

68. Z. Galil and R. Ciancarlo, Speeding up dynamic programming with applications to molecular biology, *Theoret. Comp. Sci.*, **64**: 107–118, 1989.

69. E. Ukkonen, Algorithms for approximate string matching, *Inform. Control*, **64**: 100–118, 1985.

70. J. L. Spouge, Fast optimal alignment, *CABIOS*, **7**: 1–7, 1991.

71. M. Li, B. Ma, and L. Wang, Near optimal multiple alignment within a band in polynomial time, *32th ACM Symp. on Theory of Computing*, 2000, pp. 425–434.

72. D. Sankoff, R. J. Cedergren, and G. Lapalme, Frequency of insertion-deletion, transversion, and transition in the evolution of 5S ribosomal RNA, *J. Mol. Evol.*, **7**: 133–149, 1976.

73. M. S. Waterman and M. D. Perlwitz, Line geometries for sequence comparisons, *Bull. Math. Biol*, **46**: 567–577, 1984.

74. J. Hein, A tree reconstruction method that is economical in the number of pairwise comparisons used, *Mol. Biol. Evol.*, **6**, (6): 669–684, 1989.

75. J. Hein, A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given, *Mol. Biol. Evol.*, **6**: 649–668, 1989.

76. R. Ravi and J. Kececioglu, Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree, *5th Annual Symposium on Combinatorial Pattern Matching*, 1995, pp. 330–339.

77. L. Wang, T. Jiang, and E. L. Lawler, Approximation algorithms for tree alignment with a given phylogeny, *Algorithmica*, **16**: 302–315, 1996.

78. L. Wang and D. Gusfield, Improved approximation algorithms for tree alignment, *J. Algorithms*, **25**: 255–173, 1997.

79. L. Wang, T. Jiang, and D. Gusfield, A more efficient approximation scheme for tree alignment, *SIAM J. Comput.*, **30**: 283–299, 2000.

80. H. T. Wareham, A simplified proof of the NP-hardness and MAX SNP-hardness of multiplel sequence tree alignment, *J. Computat. Biol.*, **2**: 509–514, 1995.

81. A. Z. Zelikovsky, The 11/6 approximation algorithm for the Steiner problem on networks, *Algorithmica*, **9**: 463–470, 1993.

82. B. Schwikowski and M. Vingron, The deferred path heuristic for the generalized tree alignment problem, *1st Annual International Conference On Computational Molecular Biology*, 1997, pp. 257–266.

83. J. Kececioglu and D. Gusfield, Reconstructing a history of recombinations from a set of sequences, *5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 471–480, 1994.

84. F. W. Stahl, *Genetic Recombination*, New York: Scientific American, 1987, pp. 90–101.

85. J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner, *Molecular Biology of the Gene*, 4th ed.Menlo Park, CA: Benjamin-Cummings, 1987.

86. B. Ma, L. Wang, and M. Li, Fixed topology alignment with recombination, *CPM98*, to appear.

87. S. Kannan and E. W. Myers, An algorithm for locating nonoverlapping regions of maximum alignment score, *3rd Annual Symposium on Combinatorial Pattern Matching*, 1993, pp. 74–86.

88. G. M. Landau and J. P. Schmidt, An algorithm for approximate tandem repeats, *3rd Annual Symposium on Combinatorial Pattern Matching*, 1993, pp. 120–133.

89. J. Collado-Vides, B. Magasanik, and T. F. Smith (eds.), *Integrative Approaches to Molecular Biology*, Cambridge, MA: MIT Press, 1996.

90. L. Hunter (ed.), *Artificial Intelligence in Molecular Biology*, Cambridge, MA: MIT Press, 1993.

91. J. Meidanis and J. C. Setubal, *Introduction to Computational Molecular Biology*, Boston, MA: PWS Publishing Company, 1997.

92. G. A. Stephens, *String Searching Algorithms*, Singapore: World Scientific Publishers, 1994.

Bhaskar DasGupta
University of Illinois at Chicago
Chicago, Illinois
Lusheng Wang
City University of Hong Kong
Kowloon, Hong Kong

COMPUTATIONAL INTELLIGENCE

INTRODUCTION

Several interpretations of the notion of computational intelligence (CI) exist (1–9). Computationally intelligent systems have been characterized by Bezdek (1,2) relative to adaptivity, fault-tolerance, speed, and error rates. In its original conception, many technologies were identified to constitute the backbone of computational intelligence, namely, neural networks (10,11), genetic algorithms (10,11), fuzzy sets and fuzzy systems (10,11), evolutionary programming (10,11), and artificial life (12). More recently, rough set theory (13–33) has been considered in the context of computationally intelligent systems (3, 6–11, 13–16, 29–31, 34, 35) that naturally led to a generalization in the context of granular computing (32). Overall, CI can be regarded as a field of intelligent system design and analysis that dwells on a well-defined and clearly manifested synergy of genetic, granular, and neural computing. A detailed introduction to the different facets of such a synergy along with a discussion of various realizations of such synergistic links between CI technologies is given in Refs. 3, 4, 10, 11, 19, 36, and 37.

GENETIC ALGORITHMS

Genetic algorithms were proposed by Holland as a search mechanism in artificially adaptive populations (38). A *genetic algorithm* (GA) is a problem-solving method that simulates Darwinian evolutionary processes and naturally occurring genetic operations on chromosomes (39). In nature, a *chromosome* is a thread-like linear strand of DNA and associated proteins in the nucleus of animal and plant cells. A chromosome carries genes and serves as a vehicle in transmitting hereditary information. A *gene* is a hereditary unit that occupies a specific location on a chromosome and that determines a particular trait in an organism. Genes can undergo mutation (alteration or structural change). A consequence of the mutation of genes is the creation of a new trait in an organism. In genetic algorithms, the traits of artificial life forms are stored in bit strings that mimic chromosome strings found in nature. The traits of individuals in a population are represented by a set of evolving chromosomes. A GA transforms a set of chromosomes to obtain the next generation of an evolving population. Such transformations are the result of applying operations, such as reproduction based on survival of the fittest, and genetic operations, such as sexual recombination (also called crossover) and mutation.

Each artificial chromosome has an associated fitness, which is measured with a fitness function. The simplest form of fitness function is known as raw fitness, which is some form of performance score (e.g., number of pieces of food found, amount of energy consumed, or number of other life forms found). Each chromosome is assigned a probability of reproduction that is proportional to its fitness. In a Darwinian system, natural selection controls evolution (10). Consider, for example, a collection of artificial life forms with behaviors resembling ants. Fitness will be quantified relative to the total number of pieces of food found and eaten (partially eaten food is counted). Reproduction consists in selecting the fittest individual x and the weakest individual y in a population and replacing y with a copy of x. After reproduction, a population will then have two copies of the fittest individual. A crossover operation consists in exchanging genetic coding (bit values of one or more genes) in two different chromosomes. The steps in a crossover operation are as follows: (*1*) Randomly select a location (also called the interstitial location) between two bits in a chromosome string to form two fragments, (*2*) select two parents (chromosomes to be crossed), and (*3*) interchange the chromosome fragments. Because of the complexity of traits represented by a gene, substrings of bits in a chromosome are used to represent a trait (41). The evolution of a population resulting from the application of genetic operations results in changing the fitness of individual population members. A principal goal of GAs is to derive a population with optimal fitness.

The pioneering works of Holland (38) and Fogel et al. (42) gave birth to the new paradigm of population-driven computing (evolutionary computation) resulting in structural and parametric optimization. Evolutionary programming was introduced by Fogel in the 1960s (43). The evolution of competing algorithms defines *evolutionary programming*. Each algorithm operates on a sequence of symbols to produce an output symbol that is likely to maximize an algorithm's performance relative to a well-defined payoff function. Evolutionary programming is the precursor of genetic programming (39). In *genetic programming*, large populations of computer programs are bred genetically. One may also refer to biologically inspired optimization, such as particle swarm optimization (PSO), ant colonies, and others.

FUZZY SETS AND SYSTEMS

Fuzzy systems (models) are immediate constructs that result from a description of real-world systems (say, social, economic, ecological, engineering, or biological) in terms of information granules, fuzzy sets, and the relationships between them (44). The concept of a fuzzy set introduced by Zadeh in 1965 (45,46) becomes of paramount relevance when formalizing a notion of partial membership of an element. Fuzzy sets are distinguished from the fundamental notion of a set (also called a crisp set) by the fact that their boundaries are formed by elements whose degree of belonging is allowed to assume numeric values in the interval [0, 1]. Let us recall that the characteristic function for a set X returns a Boolean value {0, 1} indicating whether an element x is in X or is excluded from it. A fuzzy set is noncrisp inasmuch as the characteristic function for a fuzzy

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

set returns a value in [0, 1]. Let $U, X, \tilde{A}$, and x be a universe of objects, subset of U, fuzzy set in U, and an individual object x in X, respectively. For a set X, $\mu\tilde{A} : X \rightarrow [0,\ 1]$ is a function that determines the degree of membership of an object x in X. A fuzzy set $\tilde{A}$ is then defined to be a set of ordered pairs, where $\tilde{A} = \{(\mathrm{x}, \mu\tilde{A}\,(\mathrm{x}))|\mathrm{x} \in \mathrm{X}\}$. The counterparts of intersection and union (crisp sets) are the t-norm and s-norm operators in fuzzy set theory. For the intersection of fuzzy sets, the min operator was suggested by Zadeh (29), and it belongs to a class of intersection operators (min, product, and bold intersection) known as triangular or t-norms. A t-norm is a mapping $t : [0, 1]^2 \rightarrow [0,\ 1]$. The s-norm (t-conorm) is a mapping $s : [0, 1]^2 \rightarrow [0,\ 1]$ (also a triangular conorm) that is commonly used for the union of fuzzy sets. The properties of triangular norms are presented in Ref. 84.

Fuzzy sets exploit imprecision in conventional systems in an attempt to make system complexity manageable. It has been observed that fuzzy set theory offers a new model of vagueness (13–16). Many examples of fuzzy systems are given in Pedrycz (47) and in Kruse et al. (48)

NEURAL COMPUTING

Neural networks offer a powerful and distributed computing architecture equipped with significant learning abilities (predominantly as far as parametric learning is concerned). They help represent highly nonlinear and multivariable relationships between system variables. Starting from the pioneering research of McCulloch and Pitts (49), Rosenblatt (50) as well as Minsky and Pappert (51) neural networks have undergone a significant metamorphosis and have become an important reservoir of various learning methods (52) as well as an extension of conventional techniques in statistical pattern recognition (53). Artificial neural networks (ANNs) were introduced to model features of the human nervous system (49). An *artificial neural network* is a collection of highly interconnected processing elements called neurons. In ANNs, a neuron is a threshold device, which aggregates ("sums") its weighted inputs and applies an activation function to each aggregation to produce a response. The summing part of a neuron in an ANN is called an adaptive linear combiner (ALC) in Refs. 54 and 55. For instance, a McCulloch–Pitts neuron n_i is a binary threshold unit with an ALC that computes a weighted sum net, where $\mathrm{net} = \sum_{j=0}^{n} w_j x_j$. A weight w_i associated with x_i represents the strength of connection of the input to a neuron. Input x_0 represents a bias, which can be thought of as an input with weight 1. The response of a neuron can be computed in several ways. For example, the response of neuron n_i can be computed using sgn(net), where sgn(net) = 1 for $net > 0$, sgn(net) = 0 for net = 0, and sgn(net) = −1, if $net < 0$. A neuron comes with adaptive capabilities that could be exploited fully assuming that an effective procedure is introduced to modify the strengths of connections so that a correct response is obtained for a given input. A good discussion of learning algorithms for various forms of neural networks can be found in Freeman and Skapura (56) and in Bishop (53). Various forms of neural networks have been used successfully in system modeling, pattern recognition, robotics, and process control applications (10,11,35,57,58).

ROUGH SETS

Zdzislaw Pawlak (13–16,59,60) introduced rough sets in 1981 (24,25). The rough set approach to approximation and classification was then elaborated and refined in Refs. 13–21, 24–31, 33, 61, and 62. Rough set theory offers an approach to CI by drawing attention to the importance of set approximation in knowledge discovery and information granulation (32).

In particular, rough set methods provide a means of approximating a set by other sets (17,18). For computational reasons, a syntactic representation of knowledge is provided by rough sets in the form of data tables. In general, an information system (IS) is represented by a pair (U, F), where U is a nonempty set of objects and F is a nonempty, countable set of probe functions that are a source of measurements associated with object features (63). For example, a feature of an image may be color with probe functions that measure tristimulus values received from three primary color sensors, brightness (luminous flux), hue (dominant wavelength in a mixture of light waves), and saturation (amount of white light mixed with a hue). Each $f \in F$ maps an object to some value in a set V_f. In effect, we have $f : U \rightarrow V_f$ for every $f \in F$.

The notions of equivalence and equivalence class are fundamental in rough sets theory. A binary relation $R \subseteq X \times X$ is an equivalence relation if it is reflexive, symmetric, and transitive. A relation R is reflexive if every object $x \in X$ has relation R to itself. That is, we can assert $x R x$. The symmetric property holds for relation R if xRy implies yRx for every $x, y \in X$. The relation R is transitive for every $x, y, z \in X$; then xRy and yRz imply xRz. The equivalence class of an object $x \in X$ consists of all objects $y \in X$ so that xRy. For each set of functions $B \subseteq F$, an equivalence relation $\sim B = \{(x, x')|\,\forall a \in B.\, a(x) = a(x')\}$ (indiscernibility relation)is associat with it. If $(x, x') \in {}^{\sim}\mathrm{B}$, we say that objects x and x' are indiscernible from each other relative to attributes from B. This concept is fundamental to rough sets. The notation $[x]_B$ is a commonly used shorthand that denotes the equivalence class defined by x relative to a feature set B. In effect, $[x]_B = \{y \in U | x \sim B\, y\}$. Furthermore, $Ul \sim B$ denotes the partition of U defined by relation $\sim B$. Equivalence classes $[x]_B$ represent B-granules of an elementary portion of knowledge that we can perceive as relative to available data. Such a view of knowledge has led to the study of concept approximation (64) and pattern extraction (65). For $X \subseteq U$, the set X can be approximated only from information contained in B by constructing a B-lower approximation $B \times X = \cup\{[x]_B | [x]_B \in Ul \sim B \text{ and } [x]_B \subseteq X\}$ and a B-upper approximation $B * X = \cup\{[x]_B | [x]_B \in Ul \sim B \text{ and } [x]_B \cap X \neq \varnothing\}$, respectively. In other words, a lower approximation $B \times X$ of a set X is a collection of objects that can be classified with full certainty as members of X using the knowledge represented by B. By contrast, an upper approximation $B * X$ of a set X is a collection of objects representing both certain knowledge (i.e., classes entirely contained in X) and possible uncertain knowledge (i.e., possible classes partially contained in X). In the case in which $B * X$ is a proper subset of $B \times X$, then the objects in X cannot be classified with certainty and the set X is rough. It has recently been observed by Pawlak (13–16) that this is exactly the idea of vagueness proposed by Frege (65). That is, the vagueness of an approximation of a set stems from its borderline region.

The size of the difference between lower and upper approximations of a set (i.e., boundary region) provides a basis for the "roughness" of an approximation, which is important because vagueness is allocated to some regions of what is known as the universe of discourse (space) rather than to the whole space as encountered in fuzzy sets. The study of what it means *to be a part of* provides a basis for what is known as mereology, which was introduced by Lesniewski in 1927 (66). More recently, the study of what it means to be a part of *to a degree* has led to a calculus of granules (23, 67–70). In effect, granular computing allows us to quantify uncertainty and to take advantage of uncertainty rather than to discard it blindly.

Approximation spaces introduced by Pawlak (24) which were elaborated by Refs. 17–19, 22, 23, 29, 30, 61, 62, 70, 71, and applied in Refs. 6–8, 35, 64, 72, and 73 serve as a formal counterpart of our perception ability or observation (61,62), and they provide a framework for perceptual reasoning at the level of classes (63). In its simplest form, an approximation space is denoted by $(U, F, \sim B)$, where U is a nonempty set of objects (called a universe of discourse), F is a set of functions representing object features, $B \subseteq F$, and $\sim B$ is an equivalence relation that defines a partition of U. Equivalence classes belonging to a partition $Ul \sim B$ are called elementary sets (information granules). Given an approximation space $S = (U, F, \sim B)$, a subset X of U is definable if it can be represented as the union of some elementary sets. Not all subsets of U are definable in space S(61,62). Given a nondefinable subset X in U, our observation restricted by $\sim B$ causes X to be perceived relative to classes in the partition $Ul \sim B$. An upper approximation $B^{*}X$ is the set of all classes in $Ul \sim B$ that have elements in common with X, and the lower approximation $B \times X$ is the set of all classes in $Ul \sim B$ that are proper subsets of X.

Fuzzy set theory and rough set theory taken singly and in combination pave the way for a variety of approximate reasoning systems and applications representing a synergy of technologies from computational intelligence. This synergy can be found, for example, in recent work on the relation among fuzzy sets and rough sets (13–16,35,37, 74,75), rough mereology (19,37,67–69), rough control (76,77), fuzzy–rough–evolutionary control (36), machine learning (18,57,72,78), fuzzy neurocomputing (3), rough neurocomputing (35), data mining (6,7,13–16,31), diagnostic systems (18,79), multiagent systems (8,9,80), real-time decision making (34,81), robotics and unmanned vehicles (57,82,83), intelligent systems (8,13,29,57), signal analysis (84), perception and classification of perceptual objects (29, 30,61–63), software engineering (4,84–87), a dominance-based rough set approach to multicriteria decision making and data mining (31), VPRS (33), and shadowed sets (75).

BIBLIOGRAPHY

1. J. C. Bezdek, On the relationship between neural networks, pattern recognition and intelligence, *Int. J. Approx. Reasoning*, **6**: 85–107. 1992.
2. J. C. Bezdek, What is computational intelligence? in J. Zurada, R. Marks, C. Robinson (eds.), *Computational Intelligence: Imitating Life*, Piscataway, NJ: IEEE Press, 1994, pp. 1–12.
3. W. Pedrycz, *Computational Intelligence: An Introduction*, Boca Raton, FL: CRC Press, 1998.
4. W. Pedrycz, J. F. Peters (eds.), Computational intelligence in software engineering, *Advances in Fuzzy Systems—Applications and Theory*, vol. **16**. Singapore: World Scientific, 1998.
5. D. Poole, A. Mackworth, R. Goebel, *Computational Intelligence: A Logical Approach*. Oxford: Oxford University Press, 1998.
6. N. Cercone, A. Skowron, N. Zhong (eds.), Rough sets, fuzzy sets, data mining, and granular-soft computing special issue, *Comput. Intelli.: An Internat. J.*, **17**(3): 399–603, 2001.
7. A. Skowron, S. K. Pal (eds.), Rough sets, pattern recognition and data mining special issue, *Pattern Recog. Let.*, **24**(6): 829–933, 2003.
8. A. Skowron, Toward intelligent systems: calculi of information granules, in: T. Terano, T. Nishida, A. Namatane, S. Tsumoto, Y. Ohsawa, T. Washio (eds.), *New Frontiers in Artificial Intelligence*, Lecture Notes in Artificial Intelligence 2253. Berlin: Springer-Verlag, 2001, pp. 28–39.
9. J. F. Peters, A. Skowron, J. Stepaniuk, S. Ramanna, Towards an ontology of approximate reason, *Fundamenta Informaticae*, **51**(1-2): 157–173, 2002.
10. *IEEE World Congress on Computational Intelligence*, Vancouver B.C., Canada, 2006.
11. M. H. Hamaza, (ed.), *Proceedings of the IASTED Int. Conf. on Computational Intelligence*. Calgary, AB, Canada, 2005.
12. R. Marks, Intelligence: computational versus artificial, *IEEE Trans. on Neural Networks*, **4**: 737–739, 1993.
13. Z. Pawlak and A. Skowron, Rudiments of rough sets, *Information Sciences*, **177**(1): 3–27, 2007.
14. J. F. Peters and A. Skowron, Zdzislaw Pawlak life and work (1926–2006), *Information Sciences*, **177**(1): 1–2, 2007.
15. Z. Pawlak and A. Skowron, Rough sets: Some extensions, *Information Sciences*, **177**(1): 28–40, 2007.
16. Z. Pawlak, and A. Skowron, Rough sets and Boolean reasoning, *Information Sciences*, **177**(1): 41–73, 2007.
17. Z. Pawlak, Rough sets, *Int. J. of Informat. Comput. Sciences*, **11**(5): 341–356, 1982.
18. Z. Pawlak, *Rough Sets. Theoretical Aspects of Reasoning about Data*, Dordrecht: Kluwer Academic Publishers, 1991.
19. L. Polkowski, Rough sets, *Mathematical Foundations. Advances in Soft Computing*, Heidelberg: Physica-Verlag, 2002.
20. Z. Pawlak, Some issues on rough sets, *Transactions on Rough Sets* I, LNCS 3100, 2004, pp. 1–58.
21. Z. Pawlak, A treatise on rough sets, *Transactions on Rough Sets* IV, LNCS 3700, 2005, pp. 1–17.
22. A. Skowron, and J. Stepaniuk, Generalized approximation spaces, in: T. Y. Lin, A. M. Wildberger, (eds.), *Soft Computing*, San Diego, CA: Simulation Councils, 1995, pp. 18–21.
23. A. Skowron, J. Stepaniuk, J. F. Peters and R. Swiniarski, Calculi of approximation spaces, *Fundamenta Informaticae*, **72**(1–3): 363–378, 2006.

24. Z. Pawlak, Classification of Objects by Means of Attributes. *Institute for Computer Science*, Polish Academy of Sciences, Report **429**: 1981.
25. Z. Pawlak, Rough Sets. *Institute for Computer Science*, Polish Academy of Sciences, Report **431**: 1981.
26. Z. Pawlak, Rough classification, *Int. J. of Man–Machine Studies*, **20**(5): 127–134, 1984.
27. Z. Pawlak, Rough sets and intelligent data analysis, *Information Sciences: An Internat, J.*, **147**(1–4): 1–12, 2002.
28. Z. Pawlak, Rough sets, decision algorithms and Bayes' theorem, European *J. Operat. Res.*, **136**: 181–189, 2002.
29. M. Kryszkiewicz, J. F. Peters, H. Rybinski and A. Skowron (eds.), rough sets and intelligent systems paradigms, *Lecture Notes in Artificial Intelligence* 4585 Berlin: Springer, 2007.
30. J. F. Peters and A. Skowron, Transactions on Rough Sets, volumes I-VII, Berlin: Springer, 2004–2007. Avaiilable: http://www.springer.com/west/home/computer/lncs?SGWID=4–164–6–99627–0.
31. R. Slowinski, S. Greco and B. Matarazzo, Dominance-based rough set approach to reasoning about ordinal data, in: M. Kryszkiewicz, J. F. Peters, H. Rybinski and A. Skowron, eds., *Rough Sets and Intelligent Systems Paradigms*, Lecture Notes in Artificial Intelligence, Berlin: Springer, 2007, pp. 5–11.
32. L. Zadeh, Granular computing and rough set theory, in: M. Kryszkiewicz, J. F. Peters, H. Rybinski, and A. Skowron, (eds.), *Rough Sets and Intelligent Systems Paradigms*, Lecture Notes in Artificial Intelligence, Berlin: Springer, 2007, pp. 1–4.
33. W. Ziarko, Variable precision rough set model, *J. Comp. Sys. Sciences*, **46**(1): 39–59, 1993.
34. J. F. Peters, Time and clock information systems: concepts and roughly fuzzy petri net models, in J. Kacprzyk (ed.), Knowledge Discovery and Rough Sets. Berlin: Physica Verlag, 1998.
35. S. K. Pal, L. Polkowski and A. Skowron (eds.), *Rough-Neuro Computing: Techniques for Computing with Words*. Berlin: Springer-Verlag, 2003.
36. T. Y. Lin, Fuzzy controllers: An integrated approach based on fuzzy logic, rough sets, and evolutionary computing, in T. Y. Lin and N. Cercone (eds.), *Rough Sets and Data Mining: Analysis for Imprecise Data*. Norwell, MA: Kluwer Academic Publishers, 1997, pp. 109–122.
37. L. Polkowski, Rough mereology as a link between rough and fuzzy set theories: a survey, *Trans. Rough Sets* II, LNCS 3135, 2004, pp. 253–277.
38. J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press, 1975.
39. J. R. Koza, *Genetic Programming: On the Progamming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1993.
40. C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London: John Murray, 1959.
41. L. Chambers, *Practical Handbook of Genetic Algorithms*, vol. 1. Boca Raton, FL: CRC Press, 1995.
42. L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, Chichester: J. Wiley, 1966.
43. L. J. Fogel, *On the organization of the intellect*. Ph. D. Dissentation, Los Angeles: University of California Los Angeles, 1964.
44. R. R. Yager and D. P. Filev, *Essentials of Fuzzy Modeling and Control*. New York, John Wiley & Sons, Inc., 1994.
45. L. A. Zadeh, Fuzzy sets, *Information and Control*, **8**: 338–353, 1965.
46. L. A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Trans. on Systems, Man, and Cybernetics*, **2**: 28–44, 1973.
47. W. Pedrycz, *Fuzzy Control and Fuzzy Systems*, New York: John Wiley & Sons, Inc., 1993.
48. R. Kruse, J. Gebhardt and F. Klawonn, *Foundations of Fuzzy Systems*. New Yark: John Wiley & Sons, Inc., 1994.
49. W. S. McCulloch and W. Pitts, A logical calculus of ideas immanent in nervous activity, *Bulletin of Mathemat. Biophy.*, **5**: 115–133, 1943.
50. F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Washington, D.C: Spartan Press, 1961.
51. M. Minsky and S. Pappert, *Perceptrons: An Introduction to Computational Geometry*, Cambridge: MIT Press, 1969.
52. E. Fiesler and R. Beale (eds.), *Handbook on Neural Computation*. oxford: Institute of Physics Publishing and Oxford University Press, 1997.
53. C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 1995.
54. B. Widrow and M. E. Hoff, Adaptive switching circuits, *Proc. IRE WESCON Convention Record, Part 4*, 1960, pp. 96–104.
55. B. Widrow, Generalization and information storage in networks of adaline "neurons". in M. C. Yovits, G. T. Jacobi, and G. D. Goldstein (eds.), *Self-Organizing Systems*. Washington, D.C.: Spartan, 1962.
56. J. A. Freeman and D. M. Skapura, *Neural Networks: Algorithms, Applications and Programming Techniques*. Reading, MA: Addison-Wesley, 1991.
57. D. Lockery, and J. F. Peters, Robotic target tracking with approximation space-based feedback during reinforcement learning, Springer best paper award, *Proceedings of Eleventh International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing* (RSFDGrC 2007), Joint Rough Set Symposium (JRS 2007), Lecture Notes in Artificial Intelligence, vol. 4482, 2007, pp. 483–490.
58. J. F. Peters, L. Han, and S. Ramanna, Rough neural computing in signal analysis, *Computat. Intelli.*, **1**(3): 493–513, 2001.
59. E. Orlowska, J. F. Peters, G. Rozenberg and A. Skowron, *New Frontiers in Scientific Discovery. Commemorating the Life and Work of Zdzislaw Pawlak*, Amsterdam: IOS Press, 2007.
60. J. F. Peters, and A. Skowron, Zdzislaw Pawlak: Life and Work. 1926–2006, *Transactions on Rough Sets*, V, LNCS 4100, Berlin: Springer, 2006, pp. 1–24.
61. E. Orlowska, *Semantics of Vague Concepts. Applications of Rough Sets*. Institute for Computer Science, Polish Academy of Sciences, Report **469**: 1981.
62. E. Orlowska, Semantics of vague concepts, in: G. Dorn, and P. Weingartner, (eds.), *Foundations of Logic and Linguistics, Problems and Solutions*, London: Plenum Press, 1985, pp. 465–482.
63. J. F. Peters, Classification of perceptual objects by means of features, Internat. *J. Informat. Technol. Intell. Comput.*, 2007. in Press.
64. H. S. Bazan, Nguyen, A. Skowron, and M. Szczuka, A view on rough set concept approximation, in: G. Wang, Q. Liu, Y. Y. Yao, A. Skowron, *Proceedings of the Ninth International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing RSFDGrC'2003)*, Chongqing, China, 2003, pp. 181–188.

65. J. Bazan, H. S. Nguyen, J. F. Peters, A. Skowron and M. Szczuka, Rough set approach to pattern extraction from classifiers, *Proceedings of the Workshop on Rough Sets in Knowledge Discovery and Soft Computing at ETAPS'2003*, pp. 2–3.

66. S. Lesniewski, O podstawach matematyki (in Polish), Przeglad Filozoficzny, **30**: 164–206, **31**: 261–291, **32**: 60–101, **33**: 142–170, 1927.

67. L. Polkowski and A. Skowron, Implementing fuzzy containment via rough inclusions: Rough mereological approach to distributed problem solving, *Proc. Fifth IEEE Int. Conf. on Fuzzy Systems*, vol. 2, New Orleans, 1996, pp. 1147–1153.

68. L. Polkowski and A. Skowron, Rough mereology: A new paradigm for approximate reasoning, *Internat. J. Approx. Reasoning*, **15**(4): 333–365, 1996.

69. L. Polkowski and A. Skowron, Rough mereological calculi of granules: A rough set approach to computation, *Computat. Intelli. An Internat. J.*, **17**(3): 472–492, 2001.

70. A. Skowron, R. Swiniarski, and P. Synak, Approximation spaces and information granulation, *Trans. Rough Sets* III, LNCS 3400, 2005, pp. 175–189.

71. A. Skowron and J. Stepaniuk, Tolerance approximation spaces, *Fundamenta Informaticae*, **27**(2–3): 245–253. 1996.

72. J. F. Peters and C. Henry, Reinforcement learning with approximation spaces. *Fundamenta Informaticae*, **71**(2–3): 323–349, 2006.

73. J. F. Peters, Rough ethology: towards a biologically-inspired study of collective behavior in intelligent systems with approximation spaces, *Transactions on Rough Sets* III, LNCS 3400, 2005, pp. 153–174.

74. W. Pedrycz, Shadowed sets: Representing and processing fuzzy sets, *IEEE Trans. on Systems, Man, and Cybernetics, Part B*: Cybernetics, **28**: 103–108, 1998.

75. W. Pedrycz, Granular computing with shadowed sets, in: D. Slezak, G. Wang, M. Szczuka, I. Duntsch, and Y. Yao (eds.), *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, LNAI 3641. Berlin: Springer, 2005, pp. 23–31.

76. T. Munakata and Z. Pawlak, Rough control: Application of rough set theory to control, *Proc. Fur. Congr. Fuzzy Intell. Techool. EUFIT'96*, 1996, pp. 209–218,

77. J. F. Peters, A. Skowron, Z. Suraj, An application of rough set methods to automatic concurrent control design, *Fundamenta Informaticae*, **43**(1–4): 269–290, 2000.

78. J. Grzymala-Busse, S. Y. Sedelow, and W. A. Sedelow, Machine learning & knowledge acquisition, rough sets, and the English semantic code, in T. Y. Lin and N. Cercone (eds.), *Rough Sets and Data Mining: Analysis for Imprecise Data*. Norwell, MA: Kluwer Academic Publishers, 1997, pp. 91–108.

79. R. Hashemi, B. Pearce, R. Arani, W. Hinson, and M. Paule, A fusion of rough sets, modified rough sets, and genetic algorithms for hybrid diagnostic systems, in T. Y. Lin, N. Cercone (eds.), *Rough Sets and Data Mining: Analysis for Imprecise Data*. Norwell, MA: Kluwer Academic Publishers, 1997, pp. 149–176.

80. R. Ras, Resolving queries through cooperation in multi-agent systems, in T. Y. Lin, N. Cercone (eds.), *Rough Sets and Data Mining: Analysis for Imprecise Data*. Norwell, MA: Kluwer Academic Publishers, 1997, pp. 239–258.

81. A. Skowron and Z. Suraj, A parallel algorithm for real-time decision making: a rough set approach. *J. Intelligent Systems*, **7**: 5–28, 1996.

82. M. S. Szczuka and N. H. Son, Analysis of image sequences for unmanned aerial vehicles, in: M. Inuiguchi, S. Hirano, S. Tsumoto (eds.), *Rough Set Theory and Granular Computing*. Berlin: Springer-Verlag, 2003, pp. 291–300.

83. H. S. Son, A. Skowron, and M. Szczuka, Situation identification by unmanned aerial vehicle, *Proc. of CS&P 2000, Informatik Berichte, Humboldt-Universitat zu Berlin*, 2000, pp. 177–188.

84. J. F. Peters and S. Ramanna, Towards a software change classification system: A rough set approach, *Software Quality J.*, **11**(2): 87–120, 2003.

85. M. Reformat, W. Pedrycz, and N. J. Pizzi, Software quality analysis with the use of computational intelligence, *Informat. Software Technol.*, **45**: 405–417, 2003.

86. J. F. Peters and S. Ramanna, A rough sets approach to assessing software quality: concepts and rough Petri net models, in: S. K. Pal and A. Skowron, (eds.), *Rough-Fuzzy Hybridization: New Trends in Decision Making*. Berlin: Springer-Verlag, 1999, pp. 349–380.

87. W. Pedrycz, L. Han, J. F. Peters, S. Ramanna and R. Zhai, Calibration of software quality: fuzzy neural and rough neural approaches. *Neurocomputing*, **36**: 149–170, 2001.

FURTHER READING

G. Frege, *Grundlagen der Arithmetik, 2*, Verlag von Herman Pohle, Jena, 1893.

W. Pedryz, Granular computing in knowledge intgration and reuse, in: D. Zhang, T. M. Khoshgoftaar and M. -L. Shyu, (eds.), *IEEE Int. conf. on Information Reuse and Intergration*. Las Vegas, NV: 2005, pp.15–17.

W. Pedrycz and G. Succi, Genetic granular classifiers in modeling software quality, *J. Syste. Software*, **76**(3): 277–285, 2005.

W. Pedrycz and M. Reformat, Genetically optimized logic models, *Fuzzy Sets & Systems*, **150**(2): 351–371, 2005.

A. Skowron and J. F. Peters, Rough sets: trends and challenges, in G. Wang, Q., Liu, Y. Yao, and A. Skowron, (eds.), *Proceedings 9th Int. Conf. on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing* (RSFDGrC2003), LNAI 2639, Berlin: Springer-Verlag, 2003, pp. 25–34.

J.F. Peters, Near sets: General theory about nearness of objects, *Appl. Math. Sci.*, **1**(53): 2609–2629, 2007.

J.F. Peters, A. Skowron, and J. Stepanuik, Nearness of objects: Extension of approximation space model, *Fundamenta Informaticae*, **79**: 497–512, 2007.

James F. Peters
University of Manitoba
Winnipeg, Manitoba, Canada
Witold Pedrycz
University of Alberta
Edmonton, Alberta, Canada

COMPUTER ENGINEERING EDUCATION

Computer engineering education (CEE) is concerned with preparing qualified computer engineers who can take the lead and innovation in the rapid growth of technology and computer engineering fields that dominate all aspects in our life. The issues of computer engineering education rely on a broad mathematical and scientific knowledge base. Before discussing issues related to CEE, a short description of computer engineering is helpful.

Developments in technologies such as telecommunications, remote sensing and detection, electronic components and robotics, as well as computers have fundamentally altered human life. The field of computer engineering is at the epicenter of this development. Computer engineering is the design, construction, implementation, and maintenance of computers and computer-controlled equipment for the benefit of humankind. A computer engineer encompasses a blend of both electrical engineering and computer science. Computer engineers are involved in many aspects of computing, from the design of individual microprocessors, personal computers, and supercomputers, to circuit design. Those in computer engineering careers engage in activities that advance technology and provide new concepts. They then put those new concepts into physical form.

Usual tasks involving computer engineers include writing software and firmware for embedded microcontrollers, designing Very Large-Scale Integration (VLSI) chips, designing analog sensors, designing mixed signal circuit boards, and designing operating systems. Computer engineers are also suited for robotics research, which relies heavily on using digital systems to control and monitor electrical systems like motors, communications, and sensors. They also solve technical problems, develop new products from initial idea conception through completion, and install computer systems.

The computer engineering field provides an opportunity to work in the continually changing information technology sector. With the development of faster hardware components, new communication systems and software, there is a need for computer engineers. The computer engineer will work as part of a team helping to solve technical problems and pass that information on to software engineers who would do the programming or network engineers who can install and operate network and communication infrastructure.

All computer engineers may do any or all of the following tasks (1):

- Analyze information to determine client needs
- Conduct training and presentations
- Collaborate with clients, project managers, and team members to organize and plan projects
- Determine whether the project will meet the desired budget
- Research, develop, integrate, and distribute security tools and associated documentation
- Design systems
- Develop and direct testing procedures
- Provide technical support
- Coordinate installation of computer hardware or software
- Document and evaluate project progress
- Manage assigned accounts
- Specify project requirements
- Provide analysis and recommendations for overall system architecture
- Maintain equipment

To be able to achieve these tasks, computer engineers rely on a broad mathematical and scientific knowledge base, as well as on a modern (up-to-date) computer engineering education.

In the sequel of this work, we first will briefly introduce the history of computer engineering curriculum evolution. Then we will explore the computer engineering field as an academic discipline. A brief summary of the IEEE-CS/ACM joint task force report on computer engineering curriculum guidelines (CE2004) will be given. Next, we will discuss modern computer engineering education practices. By "modern" we mean applying advanced technology in the computer engineering learning process (such as multimedia, Web technologies, e-learning, simulations, etc.) to produce teaching/learning materials that meet the computer engineering students' learning preferences. Learning technologies such as learning design and learning management systems will then be discussed. Finally, a vision on the future of computer engineering education will be introduced followed by concluding remarks.

HISTORY AND EVOLUTION OF COMPUTER ENGINEERING EDUCATION

In the 1940s and 1950s, several traditional subjects and courses in engineering and science started to shift toward the then new born computer engineering courses. In the 1970s, most colleges and universities worldwide began offering majors in computer science and some in computer engineering.

The Association for Computing Machinery (ACM) began publishing curriculum recommendations for computer science in 1968 (a preliminary version appeared in 1965) and for information systems in 1972. In a parallel effort, The Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) began providing curriculum recommendations in 1977. The IEEE Computer Society published its first computer engineering curriculum around 1983 (2). Prior to the 1990s, each society produced its own curriculum recommendations. Over time, the advantages of cooperative work among them

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

became obvious. Today, the societies cooperate in creating curriculum standards and, in this way, unify the efforts and send a single message to the computing and engineering communities.

IEEE-CS and ACM joined forces in the late 1980s to create a joint curriculum report for computing. Published in 1991 and known as *Computing Curricula 1991*, or CC'91, it provided guidelines for curricula for four-year bachelor's degree programs in computer science. CC'91 defined computer science in terms of 3 processes, 9 fundamental subject areas, 12 recurring concepts, and a social and professional context.

According to CC'91, the three processes of computer science are defined as follows:

- Theory (mathematical roots)
- Abstraction (scientific roots)
- Design (engineering roots)

The curriculum also included nine fundamental subject areas:

1. Algorithms and data structures
2. Architecture
3. Artificial intelligence and robotics
4. Database and information retrieval
5. Human–computer communication
6. Numerical and symbolic computations
7. Operating systems
8. Programming languages
9. Software methodology and engineering

The 12 recurring concepts are listed as follows:

1. Binding
2. Complexity of large problems
3. Conceptual and formal models
4. Consistency and completeness
5. Efficiency
6. Evolution
7. Levels of abstraction
8. Ordering in space
9. Ordering in time
10. Reuse
11. Security
12. Trade-offs and consequences

Throughout the 1990s, various efforts were made to produce curricula guidelines for other programs in computing education. By 1993, ACM had produced five reports for two-year associate-degree programs, one report each for *computer science*, *computer engineering technology*, *information systems*, *computer support services*, and *computing for other disciplines*. By the end of the 1990s, it was becoming clear that the field of computing had not only grown rapidly but had also grown in many dimensions. This rapid growth in computing results in a growing number of kinds of computing degree programs. IEEE–CS and ACM again joined forces in the late 1990s to produce an up-to-date curriculum report to replace CC'91, and to tackle the problem of the growing number of kinds of computing degree programs. IEEE–CS and ACM created a joint task force with the goal of producing Computing Curricula 2001 (CC2001), a single report that would provide curriculum guidelines for degree programs for the various computing disciplines. However, the members of the task force soon recognized the new reality: Computing had grown in so many dimensions that no single view of the field seemed adequate. The days when the field of computing consisted of only *computer science, computer engineering*, and *information systems* were over, and the richness and breadth provided by the various computing disciplines called for a new way of defining what computing curricula should be. The work of this task force, known as *Computing Curricula 2001* (CC2001), was published in December 2001. The CC2001 Report contains detailed curricula guidelines for undergraduate degree programs in computer science (3).

In response to the CC2001 model, work soon began on other discipline-specific volumes:

- The *information systems* community published its updated *IS2002* report in 2002.
- The *software engineering* community published its first report, *SE2004*, in 2004.
- The *computer engineering* community published its *CE 2004* report in 2004.
- The *information technology* community published its (draft) *IT2005* report in 2005.

The post-1990s world presents meaningful choices: Computer science, software engineering, and computer engineering each include their own perspective on software development. These three choices imply real differences: For computer engineering, software attention is focused on hardware devices; for software engineering, the emphasis is on creating software that satisfies robust real-world requirements; and for computer science, software is the currency in which ideas are expressed and a wide range of computing problems and applications are explored (see Fig. 1) (3).

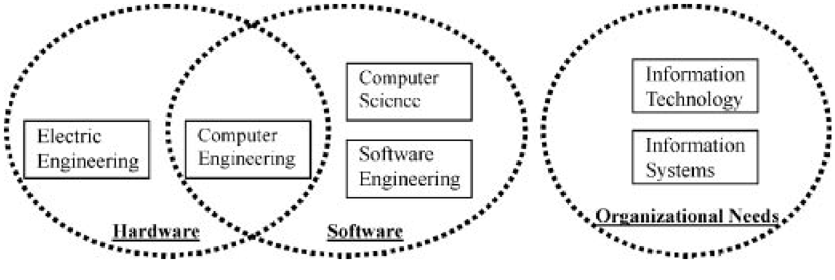


Figure 1. IEEE-CS/ACM joint task force classification for computing disciplines post-1990s (3).

COMPUTER ENGINEERING AS AN ACADEMIC DISCIPLINE AND A BRIEF OVERVIEW OF THE IEEE–CS/ACM CE 2004 REPORT

Computer engineering as an academic field encompasses the broad areas of computer science and electrical engineering.

According to the IEEE-CS/ACM joint task force classification for computing related subjects, there are three areas: hardware, software, and organizational needs. Hardware includes electric engineering and computer engineering. Software includes computer engineering, computer science, and software engineering. Organizational needs include information technology and information systems. According to this classification, computer engineering comes in the intersection between hardware and software areas. Figure 1 shows this classification.

Most universities offer computer engineering as either a degree program of its own or as a subdiscipline of electrical engineering. With the widespread use and integration of computers into our everyday lives, it is hard to separate what an electrical engineer needs to know and what a computer engineer needs to know. Because of this, several universities offer a dual degree in both electrical and computer engineering.

The field of computer science and engineering has been in a rapid growth phase for the last 50 years. This state of rapid growth and change has placed challenges on universities and colleges to define and administrate modern academic programs to prepare students adequately to enter this profession.

Due to increasing job requirements for engineers, who can design and manage all forms of computer systems used in industry, some tertiary institutions around the world offer a bachelor's degree generally called "computer engineering." Both computer engineering and electronic engineering programs include analog and digital circuit design in their curricula. As with most engineering disciplines, having a sound knowledge of mathematics and sciences is necessary for computer engineers.

A computer engineering program should contain sufficient coursework at the introductory, intermediate, and advanced levels based on a sound body of knowledge of computer engineering. Programs should be augmented by a judicious selection of elective courses that build on that foundation. Breadth and depth in science and mathematics are important to this discipline. The curriculum should also emphasize professional practice, legal and ethical issues, and the social context in which graduates implement engineering designs. A sound graduation project is also necessary. Problem-solving and critical thinking skills, oral and written communication skills, teamwork, and a variety of laboratory experiences are essential to the study of computer engineering.

The joint IEEE-CS/ACM *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* CE'2004 defines the core knowledge areas (that is, the topics the field should cover) of computer engineering as follows (4, pp. 12):

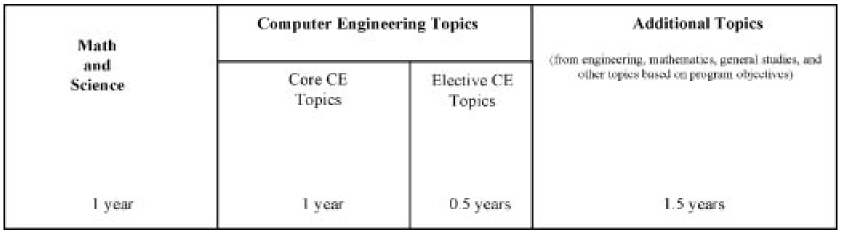
- Algorithms
- Computer architecture and organization
- Computer systems engineering
- Circuits and signals
- Database systems
- Digital logic
- Digital signal processing
- Electronics
- Embedded systems
- Human–computer interaction
- Computer networks
- Operating systems
- Programming fundamentals
- Social and professional issues
- Software engineering
- VLSI design and fabrication
- Descrite structures
- Probability and statistics

The last two areas cover related mathematical topics. The breadth of disciplines studied in computer engineering is not limited to the above subjects but can include any subject found in engineering. For example, the concepts of automata theory, which can model several hardware and software applications, are in the core of many computer science and computer engineering curriculum.

In addition to the core knowledge areas, the CE'2004 report introduced a set of possible elective courses that involves both hardware and software topics at an advanced level. Table 1 shows some examples of the proposed elective courses (4, pp. 35).

Table 1. Examples of Elective Courses

Fault-tolerant computer systems	Performance evaluation	Advanced computer architecture
Digital video processing	System-level integration	Audio signal processing
Parallel processing	High-performance computer systems	Mobile computer systems
Reconfigurable computing	Hardware software codesign	Multimedia signal processing
Intelligent systems	Computer security	Security in wireless systems
Safety critical systems	Tool development	Computer-based devices
Pervasive computing	Multimedia systems and algorithms	Novel computer architectures
Advanced graphical systems	Genetic algorithms	Distributed information systems
Computer-based medical systems	Entertainment systems	Virtual devices
Virtual environment	Robotics	Multivalued logic systems
Quantum computing	DNA computing	Nanocomputing



Math and Science	Computer Engineering Topics		Additional Topics (from engineering, mathematics, general studies, and other topics based on program objectives)
	Core CE Topics	Elective CE Topics	
1 year	1 year	0.5 years	1.5 years

Figure 2. Organization of a four-year computer engineering curriculum.

The CE'2004 report introduced a four-year model computer engineering program. The model includes one year of mathematics and science courses, one year of computer engineering core courses, one-half year of computer engineering electives, one-half year of additional engineering studies, and one year of general studies. Figure 2 illustrates this model program (4, pp. 19).

Program Evaluation and Accreditation

Professional evaluation and accreditation systems for engineering programs are critical to ensure that graduates have the proper preparation expected by society. Such accreditation is done by widely accepted professional institutions such as ABET in the United States, the *British Computer Society* (BCS) in the United Kingdom, and the *Japan Accreditation Board for Engineering Education* (JABEE) in Japan. The first accredited computer engineering degree program in the United States was established at Case Western Reserve University in 1971; as of October 1, 2007, there were 196 ABET-accredited computer engineering programs in the United States.

The evaluation criteria for accreditation of engineering programs could be summarized in the following general points:

1. Learning and educational objectives: The program must have a set of clear learning and educational objectives that reflect that the students will gain proper knowledge to have a positive impact on the society both from scientific and ethical points of view.
2. Students evaluation: The program must have a measure for evaluating the students' performances and achievements against the learning and educational objectives.
3. Curriculum requirements: The program must specify minimum curriculum requirements. For example, minimum hours for each core knowledge subject, contact office hours, and number of credit units required for graduation.
4. Educational environment: The program must ensure a proper educational environment such as facilities and equipments for labs and classes, and financial resources for operation and maintenance of the equipments, etc.
5. Learning and educational improvement: The program must have a system for continuing improvement in the learning process, such as student support system, proper use of students' feedback, and faculty development system, and so on.

These are some of the general criteria for the accreditation of engineering programs. More specific details vary from country to country. Many countries have established their own processes for evaluation and/or accreditation through governmental or professional societies.

TECHNOLOGY APPLICATIONS IN COMPUTER ENGINEERING EDUCATION

Applications of technology can provide course content with multimedia systems, active learning opportunities, and instructional technology to facilitate education in the area of computer engineering to a broad range of learners.

Multimedia in Computer Engineering Education

Multimedia is an exciting area that spans many disciplines within computer engineering—it is a computer-based communications system that integrates and delivers a complete package of audio, video, animations, graphics, and text to end users.

Throughout the 1980s and 1990s, the concept of multimedia took on a new meaning, as the capabilities of satellites, computers, audio, and video converged to create new media with enormous potential. Combined with the advances in hardware and software, these technologies were able to provide an enhanced learning facility with attention to the specific needs of individual users.

Besides being a powerful tool for making presentations, multimedia offers unique advantages in the field of education. For instance, text alone simply does not allow students to get a feel of how operating systems work. In teaching computer architecture, an instructor cannot make a simulation of a processor in a classroom. Multimedia enables us to provide a way by which learners can experience their subject in a vicarious manner. The key to providing this experience is having simultaneous graphics, video and audio, and computer simulation rather than in a text manner. The appeal of multimedia learning is best illustrated by the popularity of the video games currently available in the market. These are multimedia programs combining text, audio, video, and animated graphics in an easy-to-use fashion (5).

The benefits of using multimedia in education in general and in computer engineering in particular include, but are not limited to:

- Allowing students to act as designers by using simulation tools for analyzing the world.

- Encouraging deep reflective thinking by using more brain sensors.
- Creating personally meaningful learning opportunities.

Multimedia use has become a common practice in many computer engineering courses. For example, in Ref. 6, a simulation of electrooptic and acoustooptic theory and devices for computer engineering students was introduced. In this multimedia simulation, graphical outputs, hypertexts, and animations are widely used. There are hundreds of examples of using multimedia in computer engineering education. A few recent examples can be found in Refs. 7–13.

Web-based Technology in Computer Engineering Education

The Internet is a powerful new means of communication. It is global, it is fast, it is cheap, and it is growing rapidly. It has transformed information at nearly real-time speed. The World Wide Web is bringing rapid and radical change into all aspects of our lives. For education, the Internet is making it possible for more individuals than ever to access knowledge and to learn in new and different ways. The Internet enables bringing learning to students instead of bringing students to learning. It is allowing for the creation of learning communities that overcome the constraints of time, distance, and boundaries.

Web-based education is currently a hot research and development area. The benefits of Web-based education are clear: Learners from all over the world can enroll in learning activities, communicate with other students or teachers, as well as discuss and control their learning progress.

The Internet has provided the option of pursuing courses in computer engineering online as a part of the e-learning systems. The Web was first used as a telecontrol medium in 1994. Since then it has been applied more and more in the educational context.

The modern university needs to extend lifelong learning opportunities to its students at anytime and at anyplace to be successful in the global educational marketplace. Online Web-based learning is made possible by advancements in network infrastructure and the development of video/voice/multimedia protocols for seamless transport of information. However, the developer of a Web-based e-learning system, in addition to considering knowledge-domain requirements, must ensure good pedagogy and learning practices given the technical constraints with regard to bandwidth, quality of service, real-time interactions, multiple users, and security (14). Despite these challenges, Web-based education has been offered by universities in undergraduate computer engineering courses since 1996, with the number and sophistication of these efforts growing each year (15).

Web-based education is just beginning, with something of far greater promise emerging in the middle distance. For example, Web-based intelligent tutoring systems will soon be able to recognize a remote user's affective state and respond with an appropriate intervention. Web-based grading systems are being developed by which students can be automatically graded. All these environments seen to supplement greatly the active learning classroom/laboratory (16).

There are thousands of examples of using Web-based technology in computer engineering education. Here we refer to a few recent ones (11,12,17–19).

Active and Collaborative Learning in Computer Engineering Education

Active and collaborative learning provides a powerful mechanism to enhance depth of learning, increase material retention, and get students involved with the material instead of passively listening to a lecture. Active learning is a learning with students involved in the learning process as active partners: meaning they are "doing," "observing," and "communicating" instead of just "listening" as in the traditional (lecture-driven) learning style.

In traditional lecture-driven education, material to be learned is often transmitted to students by teachers. That is, learning is passive. In active learning, students are much more actively engaged in their own learning while educators take a more guiding role. This approach is thought to promote processing of skills/knowledge to a much deeper level than passive learning (20).

Rossati (21) and Hamada (11) showed that engineering students have strong active learning preferences. Such results suggest that active teaching materials are recommended for computer engineering students. In designing learning tools, computer engineering educators need to consider the active construction learning model (22,23), which has several basic design principles, including the following:

1. Teachers act as facilitators, not as knowledge transmitters. This means knowledge must be actively constructed by learners, not passively transmitted by teachers.
2. To motivate learners and get them actively involved in knowledge construction, learning activities should focus around a set of motivating problems and examples that have applications in the real world.
3. Learning should take place in a collaborative environment.
4. Assessment procedures should be embedded in the learning process and should consider learners' individual orientations.

As an example of recent application of active and collaborative teaching materials in computer engineering, we refer to Hamada (11).

LEARNING THEORIES IN COMPUTER ENGINEERING EDUCATION

Learning science research indicates that engineering students tend to have active and sensing learning preferences, and engineering-related educators are recognizing the need for more active and collaborative learning pedagogy (24). So far, several learning models have been developed (e.g., Refs. 25–28) for the realization of the learning preferences of learners. Among these models, Felder-Silverman (25) is simpler and easier to implement through a Web-based quiz system, as in Felder-Soloman (29). The model classifies engineering learners into four axes: active versus reflective, sensing versus intuitive, visual versus verbal, and sequential versus global. Active learners gain information through a learning-by-doing style, whereas reflective learners gain information by thinking about it. Sensing learners tend to learn facts through their senses, whereas intuitive learners prefer discovering possibilities and relationships. Visual learners prefer images, diagrams, tables, movies, and demos, whereas verbal learners prefer written and spoken words. Sequential learners gain understanding from details and logical sequential steps, whereas global learners tend to learn a whole concept in large jumps.

In Rosati (21), a study of this model was carried out to classify the learning style axes of engineering learners. The study showed that engineering learners tend to have strong active, sensing, visual, and sequential learning preferences. A similar result was obtained by Hamada (11) for computer engineering learners.

Learning Design

The central ideas behind learning design represent new possibilities for increasing the quality and variety of teaching and learning within an (e)-learning context (30, 31):

- The first general idea behind learning design is that people learn better when actively involved in doing something (i.e., are engaged in a *learning activity*).
- The second idea is that learning activities may be sequenced or otherwise structured carefully and deliberately in a *learning workflow* to promote more effective learning.
- The third idea is that it would be useful to be able to record "learning designs" for sharing and reuse in the future.

It is a recommended practice to follow the guidelines of learning design, such as the IMS-LD learning design specifications (31), when preparing teaching materials for computer engineering courses. For a recent example of using IMS-LD specifications in computer engineering, we refer to Ref. 32.

Learning Management Systems

A learning management system (LMS) is a software application or Web-based technology used to plan, implement, and assess a specific learning process. Typically, a learning management system provides an instructor with a way to create and deliver content, monitor student participation, and assess student performance. A learning management system may also provide students with the ability to use interactive features such as threaded discussions, video conferencing, and discussion forums. The Advanced Distance Learning group (33), sponsored by the United States Department of Defense, has created a set of specifications called Shareable Content Object Reference Model (SCORM) to encourage the standardization of learning management systems. Another well-known model of learning management systems is the Modular Object Oriented Dynamic Learning Environment (MOODLE) (34). MOODLE supports a range of different resource types that allow almost any kind of digital content into courses. The resource may already exist in electronic form so it can be linked to an uploaded file or external website, or simply display the complete contents of a directory in the course files and then users can pick the file themselves.

Several computer engineering courses are now integrated with such learning management systems to get benefits from its features in enhancing the learning process. See, for example, Refs. 29 and 35.

A VISION FOR THE FUTURE

Technology is growing rapidly. The following three laws are generally accepted as governing the spread of technology:

- *Moore's Law for computer performance.* The computing power is growing rapidly. According to Moore's law, which is widely believed will remain valid until at least 2020, the processing power of a chip doubles every 18 months. Moore's law suggests that performances are expected to rise to 10^{15} instructions/second (i.e., 1 petaflops/second) by 2010 and 10^{16} instructions/second by 2013.
- *Gilder's Law for communications systems.* Networks that operate at the rate of 100 GB/second exist (GB = Giga Byte, 1 GB = 1024 Megabyte). According to Gilder's law, the total bandwidth of communication systems triples every 12 months. Thus, by 2013, the capacity of communications systems will have moved to about 100 TB/second (TB = Tera Byte, 1 TB = 1024 GB).
- *Metcalfe's Law for value of a network.* Robert Metcalfe, originator of Ethernet and founder of 3COM, stated that: "the value of a network is proportional to the square of the number of nodes; so, as a network grows, the value of being connected to it grows at a polynomial rate, while the cost per user remains the same or even reduces."

In addition to this rapid growth in technology, several technical areas seem to be emerging and point to future developments in computing and computer engineering in particular. A few examples are as follows.

- *Quantum computing*: Quantum computing relies on quantum physics by taking advantage of certain quantum physics properties of atoms or nuclei that allow them to work together as quantum bits, or qubits, to be the computer's processor and memory. By interacting with each other while being isolated from the external environment, qubits can perform certain calculations exponentially faster than conventional computers. Currently, quantum computers and quantum information technology remain in the pioneering stage.

- *Nanotechnology*: Nanotechnology is the engineering of functional systems at the molecular scale. It refers to the projected ability to construct items from the bottom up to complete high-performance products. This process needs development of new techniques and tools.
- *Semantic web*: Associated with information is its semantics or meaning. If computer systems can address questions of semantics, the route becomes open for systems to engage in interesting exchanges and to carry out deduction. Currently, these developments are some way off.
- *Ubiquitous computing*: This is a paradigm shift where technology becomes virtually invisible in our lives. Instead of having a desktop or laptop machine, the technology we use will be embedded in our environment.

These technology developments suggest a new focus and possible new titles for new degrees in the future. There is no doubt that the undergraduate curriculum and the quality of its delivery are at the heart of shaping the professional life of an engineer and the degree to which he or she will be successful. The content must be relevant to the engineering community, and the pedagogy must ensure that students have a solid understanding of engineering principles and the ability to think. Furthermore, the curriculum must stimulate the motivation and development of both students and faculty.

When transforming the traditional lecture-driven educational paradigm in computer engineering into a new model that will promote learning activities that are independent of time and place, it is necessary to consider the infrastructure that allows such a model to function. The future of interactive media in a new computer engineering educational model is that of ubiquitous communication tools. This may be enabled by new technologies such as WebTVs, NetPCs, or others.

The potential for computer engineering students to author, as part of a creative educational program that is based on achievement of goals and competencies rather than on time served, will assist educators to shift from teacher to facilitator and mentor. Interactive communication tools will transform our capability to embrace an educational paradigm that deals with computer engineering learning as a vital, fulfilling, and continuing part of life at home and in the workplace as well as within educational institutions.

Because of the rapid pace of change in the computing field, computer engineers must be life-long learners to maintain their knowledge and skills within their chosen discipline.

CONCLUSION

At many universities, computer engineering emerged from electrical engineering during the late 1970s and the 1980s, but it was not until the 1990s that computer chips became basic components of most kinds of electrical devices and many kinds of mechanical devices. (For example, modern automobiles contain numerous computers that perform tasks that are transparent to the driver.) Computer engineers design and program the chips that permit digital control of many kinds of devices. The dramatic expansion in the kinds of devices that rely on chip-based digital logic helped computer engineering solidify its status as a strong field, and during the 1990s, unprecedented numbers of students applied to computer engineering programs. Computer engineering is a difficult major but it is a major that is in demand. Software engineering companies, telecommunications firms, designers of digital hardware, health-care industry, transportation, academics, financial institutions, and service-oriented businesses, as well as many other business enterprises, hire computer engineering majors right out of college (36).

The rapid advances in technology and the increasing demand for qualified computer engineers add new challenges for computer engineering educators. These challenges have caused the traditional lecture-driven classroom to give way to a new and more active environment, where students have access to a variety of multimedia and interactive course materials. Such interactive course materials have already been introduced for several topics in computer engineering courses; see, for example, Refs. 10, 13, 19, 37, and 38.

BIBLIOGRAPHY

1. Available: www.unix1.com/dir/maths_and_engineering/computer/computerengineering_jobs.
2. Educational Activities Board, The 1983 model program in computer science and engineering. Technical Report 932, Computer Society of the IEEE, December 1983.
3. IEEE-CS, ACM, and AIS Joint Task Force, Computing Curricula 2005: The Overview report, 2005.
4. IEEE Computer Society, ACM, Computer Engineering: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering, 2004.
5. Available: www.cemca.org/EMHandbook/Section2.pdf.
6. P. Jimenez et. al., Tutorial and simulation electrooptic and acoustooptic software as innovation methodology to improve the quality of electronic and computer engineering formation, *IEEE Trans. Educ.*, **49**(2): 302–308, 2006.
7. J. Boluda et. al., An active methodology for teaching electronic systems design, *IEEE Trans. Educ.*, **49**(3): 355–359, 2006.
8. W. Chen and Yu C. Cheng, Teaching object-oriented programming laboratory with computer game programming, *IEEE Trans. Educ.*, **50**(3): 197–203, 2007.
9. J. Haffner et. al., Computer-assisted evaluation of undergraduate courses in frequency-domain techniques for system control, *IEEE Trans. Educ.*, **49**(2): 224–235, 2006.
10. M. Hamada, Visual tools and examples to support active E-learning and motivation with performance evaluation, Lecture Notes in Computer Science, Vol. 3942, pp. 147–155, 2006.
11. M. Hamada, An integrated virtual environment for active e-learning in theory of computation, Lecture Notes in Computer Science,Vol. 4469, pp. 422–432, 2007.
12. M. Hamada, Web-based tools for active learning in information theory, *SIGCSE Bull.*, **39**(1): 60–64, 2007.
13. S. Li and R. Challoo, Restructuring an electric machinery course with integrative approach and computer-assisted teach methodology, *IEEE Trans. Educ.*, **49**(1): 16–28, 2006.

14. S. Sivakimar et. al., A Web-based remote interactive laboratory for internetworking education, *IEEE Trans. Educ.*, **48**(4): 586–598, 2005.
15. E. Lindsay and M. Good, Effects of laboratory access modes upon learning outcomes, *IEEE Trans. Educ.*, **48**(4): 619–631, 2005.
16. R. Reilly, *Web-based instruction: Doing things better and doing better things, IEEE Trans. Educ.,* **48**(4): 565–566, 2005.
17. M. Grigoriadou, E. Kanidis, and A. Gogoulou, A web-based educational environment for teaching the computer cache memory, *IEEE Trans. Educ.*, **49**(1): 147–156, 2006.
18. H. C. Lin, An Internet-based graphical programming tool for teaching power system harmonic measurement, *IEEE Trans. Educ.*, **49**(3): 404–414, 2006.
19. R. Nelson and A. Shariful Islam, MES—A Web-based design tool for microwave engineering, *IEEE Trans. Educ.*, **49**(1): 67–73, 2006.
20. Adult Learning. Available: http://www.nald.ca/adultlearningcourse/glossary.htm.
21. P. Rosati, The learning preferences of engineering students from two perspectives, *Proc. Frontiers in Education*, Tempe, AZ, 1998, pp. 29–32.
22. S. Hadjerrouit, Toward a constructivist approach to e-learning in software engineering, *Proc. E-Learn-World Conf. E-Learning Corporate, Government, Healthcare, Higher Education*, Phoenix, AZ, 2003, pp. 507–514.
23. G. Wilson (ed.), *Constructivist Learning Environments: Case Studies in Instructional Design*. Englewood Cliffs, NJ: Educational Technology, 1998.
24. Transforming undergraduate education in science, mathematics, engineering, and technology, in National Research Council (ed.), *Committee on Undergraduate Science Education, Center for Science, Mathematics, and Engineering Education*. Washington, DC: National Academy Press, 1999.
25. R. Felder and L. Silverman, Learning and teaching styles in engineering education, Eng. Educ., **78**(7): 674–681, 1988.
26. N. Herrmann, *The Creative Brain*. Lake Lure, NC: Brain Books, 1990.
27. D. Kolb, *Experiential Learning: Experience as the Source of Learning and Development*. Englewood Cliffs, NJ: Prentice-Hall, 1984.
28. I. Myers, *Gifts Differing*. Palo Alto, CA: Consulting Psychologists Press, 1980.
29. B. Soloman and R. Felder, Index of Learning Style Questionnaire. Available: http://www.engr.ncsu.edu/learningstyle/ilsweb.html.
30. S. Britain, A Review of Learning Design: Concepts, Specifications and Tools, A report for JISC E-learning Pedagogy Programme, May 2004.
31. IMS-LD learning design specifications, Available: http://www.imsglobal.org, 2007.
32. M. Hamada, An integrated virtual environment for active and collaborative e-learning in theory of computation, Simul. Gaming, to be published.
33. The Advanced Distance Learning group. Available: http://www.adlnet.gov 2007.
34. Modular Object Oriented Dynamic Learning Environment (MOODLE). Available: http://moodle.org, 2007.
35. D. D. Corso, E. Ovcin, and G. Morrone, A teacher friendly environment to foster learner-centered customization in the development of interactive educational packages, *IEEE Trans. Educ.*, **48**(4): 574–579, 2005.
36. Available:www.princetonreview.com/college/research/majors/majorBasics.asp?majorID=70.
37. S. Hadjerrouit, Learner-centered Web-based instruction in software engineering, *IEEE Trans. Educ.*, **48**(1): 99–104, 2005.
38. J. Masters and T. Madhyastha, Educational applets for active learning in properties of electronic materials, *IEEE Trans. Educ.*, **48**(1): 2005.

MOHAMED HAMADA
University of Aizu
Aizu-Wakamatsu, Japan

COMPUTER-SUPPORTED ASYNCHRONOUS LEARNING: THE RELEVANCE OF COMMUNICATION AND FACILITATION

INTRODUCTION

An increasing number of developments in the field of electronic support for asynchronous learning can be assigned to computer-supported collaborative learning (CSCL) (1). During collaborative learning, participants learn from each other by actively co-constructing knowledge (2). This active co-construction of knowledge bases on the theory of constructivism is described as the "motivating theory in CSCL literature" (3). The active involvement is the most important advantage of collaborative learning since participants get a deeper understanding of the learning object.

Computer-supported collaborative learning, which is often distributed locally and temporally, mainly focuses on communication since direct experience of a situation and learning by observation are mostly inapplicable. The more communication occurs, the better the situation is ranked by tutors, as well as by researchers (4). With respect to learning it was found that the communication has a crucial relevance for learning and the development of thinking (5). Thus, the support of communication is viewed as a precondition for computer-supported collaborative learning (6). These findings result in a high relevance of an appropriate communication support in CSCL systems.

This article deals on the one hand with the design of communication support for asynchronous communication and collaborative learning (see the next section). As an underlying approach for the design of communication support in CSCL systems, a communication theory is presented. In a second step, the typical characteristics of systems that follow the requirements based on this communication theory are described. This view of the theory gives the reader a chance to understand and rate the support for asynchronous learning, and this view of the characteristics of the system informs readers about actual developments in this field.

If the construction of knowledge is a joint process of several students, it is the main role of the teacher to support and facilitate the communication processes for constructing knowledge. Therefore, this article deals on the other hand with the facilitation of asynchronous discussions in CSCL processes. Based on related work, the main activities of facilitation and computer support are presented. From the presentation of communication and facilitation support, general guidelines for the design of communication and facilitation support for asynchronous collaborative learning are derived.

THE DESIGN OF ASYNCHRONOUS COMMUNICATION SUPPORT

Context-Oriented Communication Theory

In computer science, models of communication are generally characterized by the sender–recipient model by Shannon and Weaver (7). In this model, the transmission of a sender's message to a recipient by a channel is assumed, where the message is encoded before and decoded after the transmission. However, even if human communication acts are transmitted by a CSCL system, they are more than only the technical transport of a coded message from A to B and a following decoding. Psychologically oriented research shows that both communication partners have to contribute if the mutual understanding and construction of ideas is to succeed in dialogues (8). Communication is a process that is influenced by several selections. A communicator selects from the universe of his/her beliefs what he/she wants to say, and the recipient selects, with respect to his/her universe of beliefs, what he/she wants to understand. In the course of social interaction, these selection processes cannot be determined in advance but can only be influenced by the communication.

Another basic characteristic of communication is the relevance of context. Communication can only succeed, if the communicator's expressions are completed by the context that can be perceived by the communicator and the recipients (9). Context is understood as the physical and social setting in which the communication takes place (10). The context of the communicators is represented by what they perceive during communication and by what they have perceived before the moment of the communication act. Since context can refer to the past, an expression of the moment can become part of another expression's context in the future. The starting point or the boundaries of the context of a communication act cannot be defined deterministically. It belongs to the task of the communicators to encircle the scope of context that can support their communication.

By referring to the available context, two essential advantages are achieved. On the one hand, the explicitness of the conveyed content does not need to be maximal, because only these pieces of information have to be given that are required to complete the context in such a way that the message can be reconstructed and understood by the recipient. For example "Where is the car?" can be answered with "behind the red house" if there is only one red house that is part of the perceptible context. The communicator has to anticipate the scope of context that is available for the addressee. This anticipation can be supported by knowledge about the communication partner. Eventually, the need for explicit communication can be reduced ("where is the car?"—"same place as yesterday"). On the other hand,

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

the available context assists in finding out whether the communication partners understand each other. Depending on how a situation evolves, there are either indicators for the success of a communication task or an identifiable necessity to recheck the comprehension of the message or simply to improve the communication ("let's get up immediately" — "why are you not getting up right now" — don't you understand me?").

To emphasize the role of the context, this theory is called context-oriented communication support (11). It can and should be used for the design of CSCL systems, especially its communication support.

Computer Support for Context-Oriented Communication

In CSCL systems, the provided material can and has to be used to serve as context. For the design of CSCL systems, a design rationale for computer-supported communication is suggested that emphasizes the difference between communicative contributions and context as well as the tight interweavement of both. Hmelo-Silver, for example, mentions this requirement with respect to computer-supported, problem-based learning: "There needs to be a mechanism for the facilitator and other students to negotiate and discuss the contents of the whiteboards in an integrated fashion" (12).

These requirements are fulfilled in systems that use annotations for the support of communication. The annotations (= communicative contributions) are related to learning material (= context). Discussions occur by annotating annotations. The design of communicative contributions in the form of annotations in CSCL systems is inspired by systems for joint creating and editing of text like CoNote (13), CaMILE (14) or WebAnn (15). All these systems focus on functionalities enabling annotations but do not support the linkage of fine-grained material (CoNote and CaMILE) or material that is added by the learners (WebAnn). Therefore, the material cannot be used flexibly as context. Two CSCL systems that follow the context-oriented communication theory are anchored discussions (16) and KOLUMBUS (17).

In the following discussion, the main concepts for the support of context-oriented communication and facilitation are explained by the example KOLUMBUS. KOLUMBUS[1] is a Web-based CSCL system and should be viewed as one example for the class of CSCL systems that support asynchronous learning. The name is an acronym for a German term that stands for "Collaborative Learning Environment for Universities and Schools." The concentration on only one system in this article offers the possibility to show interrelations between different functionalities and to add experiences with the described functionalities.

The crucial feature of KOLUMBUS is to support the segmentation of content into small units. This segmentation allow the learners a highly flexible intertwining of content as context with communicative contributions in the form of annotations. KOLUMBUS provides two different views of content. In the tree view, each item is represented as a node in a hierarchical tree-structure. To focus on relevant content, parts of the tree or the whole tree can be expanded or minimized. Furthermore, newly inserted items are indicated as new. Although the structure of a set of interrelated annotations represents a dialogue-oriented discussion thread, the hierarchical structure of the material depends of the logical relationships between its content.

By contrast to the tree view, the article view shows content in a visually more attractive and readable way. Here, different types of presentations are combined to form a single document. Within the paper view, KOLUMBUS supports the perception of meaningful structures built up on a didactical basis. The teacher is responsible for preparing material, arranging how the (initial) material is displayed in the paper view. Didactically prepared material of the teacher can be used by the students as a starting point for their own research. In the paper view, it is also possible to expand or reduce the scope of displayed items and, therefore, to achieve an adaptable extent of context as being necessary to facilitate communicative understanding. Figure 1 shows the menu that can be activated at every single item. It allows users, for example, to add communicative contributions (in the form of annotations) or material. All these functions of KOLUMBUS are available in both types of representation (paper or tree view).

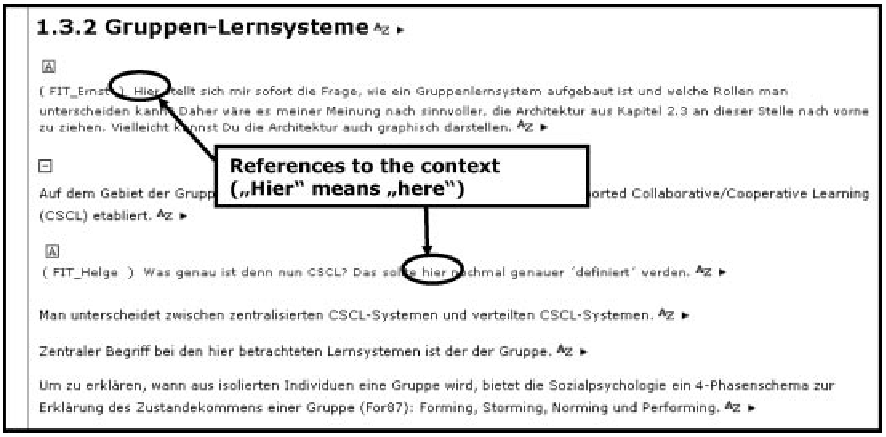
Both the tree view and the paper view can be used to work with an individual's own material as well as with that of others. To differentiate between annotations and material, the tree view uses different icons, whereas the paper view employs different colors. In the paper view, the communicative character of annotations is increased by prefixing the annotation with the author's name, which is similar to the convention with newsgroups. After switching from tree to paper view, annotations are hidden behind a nonintrusive symbol to offer the chance to assimilate the material first.

An advantage of the concept of the fine-grained item-structure is that communicative contributions can be directly linked to that part of the content to which they refer, and which therefore provides the relevant context. From this point of view, it becomes obvious that the definition of context depends on the communication act itself. Context is everything to which an annotation refers.

Figure 1 shows the paper view from a seminar in computer science: a title and some sections of material and two annotations (communicative contributions). Annotations are signed with an "A" and the name of the author in front. Because the communicative contributions are placed in the direct context, the author does not need to include hints for additional context. This leads to relatively short contributions and to the usage of direct references (in both annotations in Fig. 1, the word "hier" (German for "here") is used to reference the context).

Asynchronous learning by communication is supported by the possibility of discussion threads that can be developed by annotating other participants' annotations. These threads can be handled in the same manner as, for example, in newsgroups. Threads can occur in parallel; they can be expanded or minimized (as all items in KOLUMBUS). Figure 2 shows an example in the tree view. Items are signed with the pencil and post-it icon. Since the tree view

[1] See http://www.imtm-iaw.rub.de/projekte/k2/index.html for details and examples.



1.3.2 Gruppen-Lernsysteme

(FIT_Ernst) Hier stellt sich mir sofort die Frage, wie ein Gruppenlernsystem aufgebaut ist und welche Rollen man unterscheiden kann. Daher wäre es meiner Meinung nach sinnvoller, die Architektur aus Kapitel 2.3 an dieser Stelle nach vorne zu ziehen. Vielleicht kannst Du die Architektur auch graphisch darstellen.

Auf dem Gebiet der Grupp … orted Collaborative/Cooperative Learning (CSCL) etabliert.

(FIT_Helge) Was genau ist denn nun CSCL? Das sollte hier nochmal genauer 'definiert' verden.

Man unterscheidet zwischen zentralisierten CSCL-Systemen und verteilten CSCL-Systemen.

Zentraler Begriff bei den hier betrachteten Lernsystemen ist der der Gruppe.

Um zu erklären, wann aus isolierten Individuen eine Gruppe wird, bietet die Sozialpsychologie ein 4-Phasenschema zur Erklärung des Zustandekommens einer Gruppe (For87): Forming, Storming, Norming und Performing.

Figure 1. Communicative contributions in context.

should only give an overview, just the beginning of the annotations (as well as text-based material) is presented in one row. The whole content can be read in a tool tip that appears with the mouse-over.

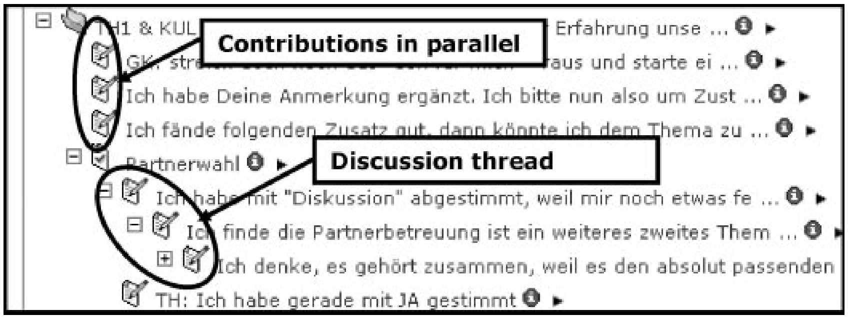
Experiences with KOLUMBUS (see Ref. 17 for details) revealed on the one hand that the integration of communication and work on material is an appropriate concept to support context-oriented communication and collaborative learning. The studies support the requirements derived from the context-oriented model of communication. Learning material serves as context and supports the communication. The tight integration of communicative contributions in the form of annotations and segmented learning material helps in general the communicator to select the appropriate pieces of context information and the recipient to understand better the utterance of the communicator. However, problems with the detection of new communicative contributions occur when the content structure is growing very fast — this lead to the necessity for concepts like the annotation window. This problem is also related to the question of an appropriate granularity. A fine granularity helps a communicator to relate his expression exactly to the context but results in a fast-growing content tree. A coarse granularity, on the other hand, leads to a manageable content structure but does not offer the possibility to relate the annotation to exact context information. The granularity of paragraphs seems to be appropriate for the joint development of texts.

On the other hand, they emphasize improvements concerning the handling of communicative contributions in the form of annotations and their interweavement with material as context. These improvements deal with the following topics:

Differentiation between Organizational and Content-Related Annotations. Two requested types of annotations are available. A new annotation has the property "content-related" by default, but it can be labeled as "organizational" by the author. The different labels correspond to different colors in all views that help the reader to differentiate the annotations at a first glance. These categories are context information that helps the recipient to estimate the aim of the communicative contribution.

The second round of studies revealed many incorrect typed contributions with the default entry "content-related" although they include only organizational issues. This finding shows that the participants often did not reflect the type of their contributions and the recipients had the major burden to reconstruct the real aim of the annotation. The existence of a default entry is misleading because the entry suggests information that is not given by the communicator.

On the other hand, the recategorization showed that both types are relevant for collaborative learning. This is especially true in long period scenarios that do not include weekly face-to-face meetings because all organizational



TH1 & KUL … Erfahrung unse …
GK: stre… aus und starte ei …
Ich habe Deine Anmerkung ergänzt. Ich bitte nun also um Zust …
Ich fände folgenden Zusatz gut, dann könnte ich dem Thema zu …
Partnerwahl
Ich habe mit "Diskussion" abgestimmt, weil mir noch etwas fe …
Ich finde die Partnerbetreuung ist ein weiteres zweites Them …
Ich denke, es gehört zusammen, weil es den absolut passenden
TH: Ich habe gerade mit JA gestimmt

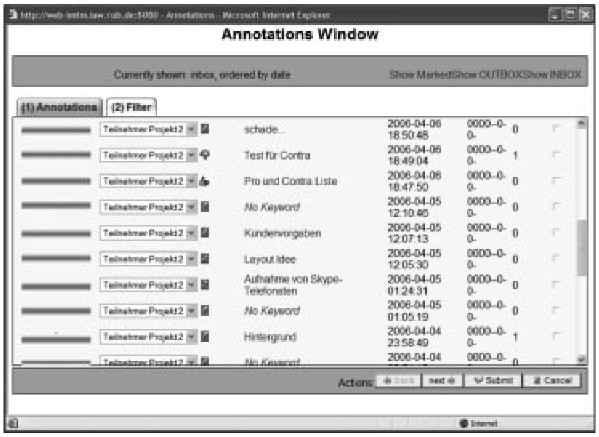
Figure 2. Discussions by using annotations.

issues are discussed with the help of the CSCL system, and this requires organizational contributions. The studies showed that in short period scenarios the organizational effort is not that high, and in a setting with weekly face-to-face meetings, a lot of organizational issues were discussed in the meeting. To keep all these arguments in mind, the usage of the two categories content-related and organizational without a default entry are proposed. Thus, a context is not suggested that is not also given.

Usage of Keywords as Context Information. Keywords are a summary of the communicative contribution. The communicator labels the contribution with words that are important for him and that help the recipient to estimate the content. A keyword for the annotation can be added similarly to the subject field of an e-mail. This keyword summarizes the annotation and helps the reader to recognize the content of the annotation at a glance. The keyword as well as the author and the date are prefixed to the annotation itself.

The results of the studies revealed that keywords are more often used when the previous annotation was written a long time before. In timely nearby contributions, the communicator seems to suppose that the recipient can conclude the context in the form of the appropriate discussion thread. We conclude that keywords are a helpful kind of context information that has to be included in a CSCL system—especially in long period asynchronous discussions. A reply-entry (like in e-mail applications) could support a communicator in automatic filling the keyword when contributing to an already existing discussion thread.

Chronological View of Communicative Contributions. Another technical improvement contains an annotation window as an additional view (see Fig. 3). This window is comparable to an e-mail inbox that gives an overview of all annotations in the chosen content area. The entries in the list are links that guide the user to the annotation's position in the integrated view. The list can be sorted by different metadata (e.g., author, date, and subject) and filtered (e.g., only content-related annotations). This window helps to perceive the annotations in chronological order and to be aware of new annotations.



Currently shown: inbox, ordered by date   Show Marked Show OUTBOX Show INBOX

[redacted]	Teilnehmer Projekt2	schade...	2006-04-06 18:50:48	0000-0-0-	0	☐
[redacted]	Teilnehmer Projekt2	Test für Contra	2006-04-06 18:49:04	0000-0-0-	1	☐
[redacted]	Teilnehmer Projekt2	Pro und Contra Liste	2006-04-06 18:47:50	0000-0-0-	0	☐
[redacted]	Teilnehmer Projekt2	*No Keyword*	2006-04-05 12:10:46	0000-0-0-	0	☐
[redacted]	Teilnehmer Projekt2	Kundenvorgaben	2006-04-05 12:07:13	0000-0-0-	0	☐
[redacted]	Teilnehmer Projekt2	Layout Idee	2006-04-05 12:05:30	0000-0-0-	0	☐
[redacted]	Teilnehmer Projekt2	Aufnahme von Skype-Telefonaten	2006-04-05 01:24:31	0000-0-0-	0	☐
[redacted]	Teilnehmer Projekt2	*No Keyword*	2006-04-05 01:05:19	0000-0-0-	0	☐
[redacted]	Teilnehmer Projekt2	Hintergrund	2006-04-04 23:58:49	0000-0-0-	1	☐
[redacted]	Teilnehmer Projekt2	*No Keyword*	2006-04-04	0000-0-	0	☐

Figure 3. Annotation window as an overview of new contributions (names are hidden due to provacy reasons).

Studies revealed that the usefulness of the annotation window depends on the underlying learning scenario. In scenarios with the joint creation of material by the group of learners, the content structure is growing very fast and the detection of new annotations "somewhere" in this structure becomes difficult. For these scenarios, an annotation window comparable to an e-mail in- and outbox gives an overview of communicative contributions and serves as a helpful awareness feature.

FACILITATING ASYNCHRONOUS DISCUSSIONS

Activities of a Facilitator

Most of the present literature on facilitation in computer-supported settings addresses the practitioners in the field (18–20). These publications deal with the tasks and responsibilities of a facilitator in computer-supported situations that are on the one hand similar to activities well known from face-to-face situations (19). Generally speaking, these tasks are the initiation of discussions, the guidance of the discussion process (includes asking appropriate questions to push students to think deeply on the learning content), and the stimulation of summary generation by the learners (12).

On the other hand, it is stated that it is necessary to develop new strategies when facilitating computer-supported communication processes (18). One of the lesser known studies concerning these strategies was done by Friedrich et al. (21). They compared two different methods to initiate an asynchronous discussion. One method relied on a neutral opening statement, whereas the other made use of problem-centric, curiosity-arousing wording when initially characterizing the discussions' objective. They confirmed the assumption that the latter type of discussion initiation results in an increased number of contributions from discussion participants. Furthermore it was found that the fewer statements facilitators contribute to the discussion, the greater the number of participant statements.

This background leads to the necessity for research on two levels. (1) What are the main facilitation activities and (2) how are they supporter with technical functionalities. The following results from a study are presented that address both levels. Details of the study can be found in Ref. 22. In this study, a group of students had the task to discuss and write their final project documentation. To gain experience with the task of a facilitator, a professionally trained facilitator planned interventions in cooperation with the researcher. These interventions, resp. facilitation strategies, vary in the degree of content-related responsibility of the facilitator.

Open Questions without any Instructions. At the beginning of the study, the facilitator asked open questions as it is similarly the case with traditional facilitation in face-to-face groups. This implies in particular that the students had to decide by themselves which functionalities they used and when they answered the question. It leads to low participation. In the first group interview, the students were asked to explain their low participation, and they reported their uncertainty about when and in which form the answers were required. Furthermore, they described obscurities concerning the (subjective) cognition of the progress in a discussion thread, especially whether a discussion was finished. Following these answers, explicit deadlines were demanded. Furthermore, it was unclear for the students how the future development of a discussion thread will look like, especially how and when it will come to an end. With respect to this open-endedness, the students' preference for explicit deadlines became apparent in their answers.

Instruction, Deadline, and Finalizing Conclusion (One Step More Responsibility for the Facilitator). In a second step, the facilitator used more instructional contributions that included deadlines. This strategy led to higher participation levels in the discussion. The analysis reveals for the first time that students worked in a rhythm similar to that given by the facilitator. On deadline days, more contributions were added.

As known from traditional facilitation methods, the facilitator gave a summary after the deadline and asked for additional comments. Reaction to this query was reduced. In the following group, interview students complained that with such questions the discussion was not terminated. This is the first difference compared with face-to-face settings, in which closing queries are a widely accepted technique. Students believed that every discussion participant in computer-supported asynchronous settings has the opportunity to contribute because of the longer period of time. Therefore, no additional comments should be requested in the asynchronous settings compared with the face-to-face situations.

Reviewing the content and results of the discussion, it must be stated that the initial aim of generating a collaboratively developed table of contents accepted for the joint documentation of group work by all group members was not achieved in the computer-supported discussion. Students reported different problems in the group interview. The first issue was the starting point of students' participation in the discussion; the temptation to wait to see what others were going to add was great. A second problem concerned concurrency and easy negotiation of opinions, which proved to be unmanageable in the system; a simple agreement like head nodding in face-to-face situations seemed to be impossible in the computer-supported asynchronous setting.

To conclude, it was clear that although participation was high discussions were not terminated in the computer-supported discussion. Students felt termination or finalizing should be done by the facilitator.

Conclusions with Decisions by the Facilitator (Full Responsibility of the Facilitator). In a third step, the facilitator intervened more than during previous steps. She did not only formulate more instructions that included deadlines but terminated discussions. If some topics did not come to an end by the deadline, the facilitator decided to stop and proposed a solution. This is a second aspect in computer-supported asynchronous facilitation that differs from that in a face-to-face situation where the facilitator is mostly not responsible for the content of the discussion or of the group result.

A high level of participation was recorded in this step; once again the participation was highest on deadline days and more detailed some minutes before the deadline. Apparently, the students followed the rhythm set by the facilitator. The discussion in this step led to the aim of coordinating tasks for writing the group's documentation. In the group interview, the students confirmed that the progress of the process was achieved by the facilitators' intervention.

In the summarizing subsection of this chapter, these findings regarding the activities of a facilitator are related to the findings regarding the computer support of a facilitator.

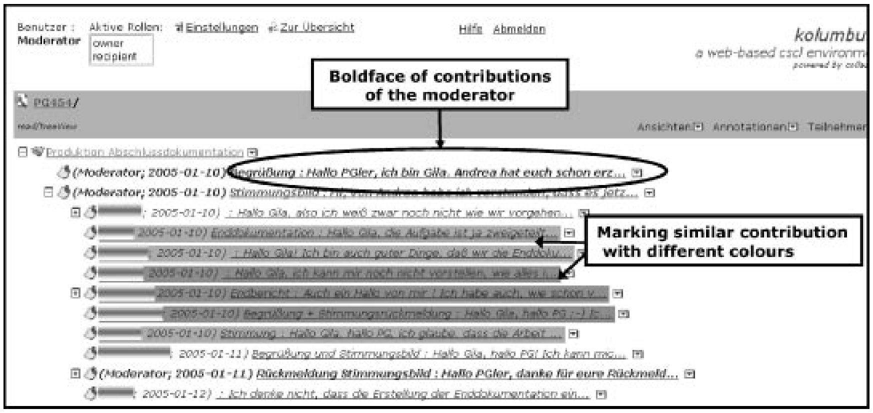
Computer Support of a Facilitator

With respect to the computer support for the tasks of a facilitator, so far only a few approaches exist. Prominent examples concerning asynchronous settings are the facilitation of online forums (23) or electronic mailing lists (24). In these approaches, all communicative contributions are sent to the facilitator, which filters them and distributes them to others. For collaborative learning scenarios, these approaches seem to be too restrictive to support an open discussion of the learners because the facilitator is more seen as an editor than as a guide; discussions where the learners are the decision makers and lead their own discussion are repressed (25). Concerning the domain of asynchronous learning only, requirements for the support of a facilitator are published (12).

Figure 4 gives an overview of the proposed technical functionalities for asynchronous facilitation support realized in KOLUMBUS 2 (22). In contrast to the above-described approaches, the users can contribute unfiltered to the different discussions. In the discussion threads, **moderator contributions are highlighted** with bold type, directing attention of the discussion's participants to the facilitator's inputs. This bold type of the facilitator's statements does also visually structure the discussion and reduce the necessity to reconstruct the course of a debate when working asynchronously. By this structuring, the initiation, resp. leading over to the next phase, as well as the summarization are supported.

Students confirmed that emphasizing a facilitator's statements by using bold fonts proved to be helpful in following the course of a discussion. Since the contributions of a facilitator often brought up a new topic and thus resulted in a new discussion thread, emphasizing them pointed out the structure of an extensive discussion more clearly. For instance, if two facilitator statements were displayed one below the other, topics thus far not discussed became rapidly apparent.

To promote contributions to an ongoing discussion, two functionalities are offered. Emphasis can be placed on single contributions to a discussion by using **a highlighting functionality**. To label an element of a discussion thread, the facilitator can choose from a variety of background colors. Marking contributions in this way can be


Benutzer : Moderator
Aktive Rollen: owner recipient
Einstellungen
Zur Übersicht
Hilfe
Abmelden
kolumbu
a web-based cscl environm
PG454/
Ansichten
Annotationen
Teilnehmer
Boldface of contributions of the moderator
Marking similar contribution with different colours

Produktion Abschlussdokumentation

(Moderator; 2005-01-10) Begrüßung : Hallo PGler, ich bin Gila. Andrea hat euch schon erz...

(Moderator; 2005-01-10) Stimmungsbild : Hi, von Andrea habe ich verstanden, dass es jetz...

[redacted]; 2005-01-10) : Hallo Gila, also ich weiß zwar noch nicht wie wir vorgehen...

[redacted] 2005-01-10) Enddokumentation : Hallo Gila, die Aufgabe ist ja zweigeteilt...

[redacted]; 2005-01-10) : Hallo Gila! Ich bin auch guter Dinge, daß wir die Enddoku...

[redacted] 2005-01-10) : Hallo Gila, ich kann mir noch nicht vorstellen, wie alles i...

[redacted] 2005-01-10) Endbericht : Auch ein Hallo von mir ! Ich habe auch, wie schon v...

[redacted]; 2005-01-10) Begrüßung + Stimmungsrückmeldung : Hallo Gila, hallo PG ;-) Ic...

[redacted] 2005-01-10) Stimmung : Hallo Gila, hallo PG, ich glaube, dass die Arbeit ...

[redacted]; 2005-01-11) Begrüßung und Stimmungsbild : Hallo Gila, hallo PG! Ich kann mic...

(Moderator; 2005-01-11) Rückmeldung Stimmungsbild : Hallo PGler, danke für eure Rückmeld...

[redacted]; 2005-01-12) : Ich denke nicht, dass die Erstellung der Enddokumentation ein...

Figure 4. The support of a facilitator.

used, for example, to group similar contributions or to accentuate important arguments or to stress (intermediary) the results of a discussion. There is no predefined meaning to the usage of different colors. It was intended that a user group develops the corresponding conventions without a predefined meaning. The discussion of the meaning assigned to the applied colors fosters the development of shared understanding of the applied functionalities.

Although it would have been useful to label similar or agreeing proposals when students were collecting ideas on outline and content of the documentation they had to compose, in the case study, the facilitator did not apply the highlighter functionality to draw attention to single contributions. Investigating possible reasons for this behavior, it turned out that she considered the design of the highlighting mechanism as too coarse-grained as only the entire contribution could be highlighted. In this context, the facilitator referred to a technique known from the facilitation of face-to-face meetings whereby crucial points are committed in writing to cards that can be arranged on a pin board. In these situations, one does not put down complete statements but confines oneself to recording only the most important keywords. According to this, calling attention to a whole contribution in a discussion thread by highlighting it proves to be an inappropriate means if one only intends to underscore essential propositions. An initial suggestion for improvement can be derived from these findings. Instead of being restricted to the level of items, subsequent versions of the highlighter mechanism described here should be applicable in a more fine-grained manner (i.e., facilitating the selection of single words) in order to allow for a precise accentuation.

Furthermore, **system-internal links** can be established if contributions that are semantically related to each other have to be interconnected. Establishing a relation between elements in such a way is especially reasonable if they deal with similar aspects of a topic but are distributed over several discussion threads and not directly connected to each other.

Since there was no situation where similar aspects of a topic were addressed in various discussion threads, there was no necessity to connect semantically related contributions using links between different discussion threads. Thus, the corresponding functionality remained unused. I assume that this is a consequence of the carefully planned interventions of the facilitator. The discussion was well guided—the participants added their contributions to the appropriate contribution of the facilitator. Because these well-prepared interventions from the facilitator cannot be expected in every case, I still propose that system-internal links may be a benefit when facilitating asynchronous discussions of group of learners.

During the study, the facilitator made proposals for additional functionalities aimed at improved support for activities typical to the facilitation of both face-to-face and computer-supported discussions. First, a facilitator should be able to "assign questions and work orders individually" by means of a collaboratively shared task list. Supporting the assignment and handling of tasks is closely related to functionalities fostering the participant's awareness of the current state of the collaborative process in which they are involved. Furthermore, the facilitator asked for a means to support synchronous voting in order to speed up the process by which participants reach a group decision. To achieve results for asynchronous facilitation, synchronous communication was not offered. However, it confirms the need for synchronous facilitation as described in the following section as well as a fluent integration of both synchronous and asynchronous modes (for integration, see Ref. 17).

GENERAL GUIDELINES

Based on these findings, the general design guidelines for the technical support of communication as well as for the activities of a facilitator and their support can be derived.

For the design of the support of communication in asynchronous learning scenarios, be aware of the following guidelines:

1. Support context-orientation by integrating of communication as annotations into segmented learning material as context. The granularity of paragraphs is appropriate for the scenario of the joint development of material.

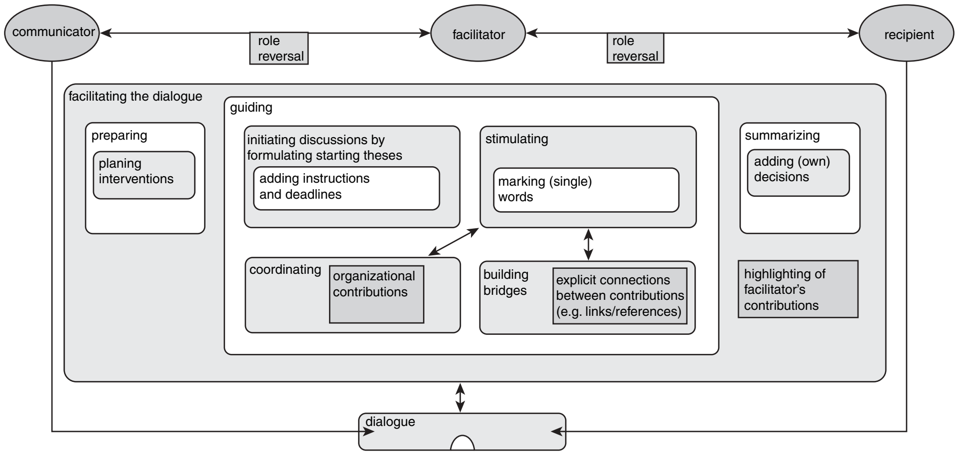

communicator
role reversal
facilitator
role reversal
recipient
facilitating the dialogue
preparing
planing interventions
guiding
initiating discussions by formulating starting theses
adding instructions and deadlines
stimulating
marking (single) words
summarizing
adding (own) decisions
coordinating
organizational contributions
building bridges
explicit connections between contributions (e.g. links/references)
highlighting of facilitator's contributions
dialogue

Figure 5. The facilitation of asynchronous learning: activities and supporting functionalities.

2. Support detailed information about the requested context information like categories and keywords. They are especially helpful in long-period settings and overlapping discourses because these situations request more explicit context information. Keep in mind that users have to be aware that they participate in communication acts and explain the benefit of using this context information.
3. Provide a special view that helps to overview the course of annotations. This is especially useful in scenarios of discourses that start with a divergent phase where annotations are connected to different items of material that are widely spread over the content.

For the support of a facilitator in asynchronous learning scenarios, a framework is derived that includes the activities of a facilitator of asynchronous learning scenarios and their support. Figure 5 combines in the middle the tasks of the facilitator that build the facilitation strategy (elements with rounded corners) and functionalities of the technical system (rectangles). When taking the role of the facilitator, the participant should be aware of the activities. For the design of technical systems that support asynchronous learning, it is recommended to add tips for fulfilling activities, for example, as help/tool tip for facilitators.

In general facilitation in the framework is divided into the activities of preparing, guiding, and summarizing that is inspired by related work as well as by the findings of the study. For asynchronous learning, a facilitator should prepare the discussions by carefully planning her/his interventions. This reduced, for example, parallel discussions with similar content that might result in misunderstandings.

During the discussion, a facilitator initiates it. In the asynchronous mode, a facilitator's contributions, especially starting a thesis, should contain instructions for the expected reactions on the starting thesis and deadlines.

For ongoing discussions, the facilitator has the task of stimulating the discussion. Here content should be emphasized (e.g., marking words). It is often accompanied by activities of building bridges and coordination acts. For these activities, technical functionalities are recommended based on the findings from the study. For the coordination, organizational contributions should be offered, for building bridges explicit connections by links.

At the end, discussions should be summarized in both modes. To increase the perception of these summaries, facilitators should ensure that participants be aware of these summaries. From a technical point of view, summaries should be placed in a prominent manner to emphasize their relevance. For asynchronous settings, it is important that the facilitator should include decisions made during the discussion. If necessary the facilitator makes decisions on his own—in contrast to face-to-face settings where the facilitator is not responsible for the content of decisions.

BIBLIOGRAPHY

1. T. Koschmann (ed.) *CSCL: Theory and Practice*, Hillsdale, NJ: Lawrence Erlbaum Associates, 1996.
2. G. Stahl, Contributions to a theoretical framework on CSCL, *Proc. of CSCL 2002*. 2002, 62–71.
3. D. D. Suthers, Technology affordances for intersubjective meaning making: A research agenda for CSCL, *Int. J. Comput. Support. Collab. Learning*, **1** (3): 2006.
4. F. Henri, Distance learning and computer-mediated communication: Interactive, quasi-interactive or monologue, in: C. O'Malley (ed.), *Computer Supported Collaborative Learning*, Berlin, Springer, Germany, 146–161, 1995.
5. H.-C. Arnseth and S. Ludvigsen, Approaching institutional contexts: systemic versus dialogic research in CSCL, *Int. J. Compu. Sup. Collab. Learning*, **1**, (2): 167–185, 2006.
6. R. D. Pea, Seeing what we build together: distributed multimedia learning environments for transformative communications, in T. Koschmann (ed.), *CSCL: Theory and Practice*, Hillsdale, NJ: Lawrence Erlbaum Associates, 1996.
7. C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, Urbana, IL: The University of Illinois, 1949.

8. H. H. Clark and S. E. Brennan, Grounding in communication, In: L. B. Resnick, J. M. Levine, and S. D. Teasley (eds.), *Perspectives on Socially Shared Cognition,* Washington, DC: American Psychological Association, 1991.
9. G. Ungeheuer, Vor-urteile über sprechen, mitteilen, verstehen, In *Kommunikationstheoretische Schriften 1*, Ungeheuer (ed.), Aachen, Germany, Rader, 1982.
10. O. Ducrot and T. Todorov, *Encyclopedic Dictionary of the Sciences of Language*, Baltimore, MD: Johns Hopkins University Press, 1987.
11. Th. Herrmann and A. Kienle, Context-oriented communication and the design of computer supported discoursive learning, *Int. J. Comput. Suppor. Collab. Learning*, In-Press.
12. C. E. Hmelo-Silver, Collaborative ways of knowing: Issues in facilitation, in: G. Stahl, (ed.), *Computer Support for Collaborative Learning, Foundations for a CSCL community*, Mahwah, NJ: LEA, 2002.
13. J. Davis and D. Huttenlocher, Shared annotation for cooperative learning, in: *Proc. of CSCL* 1995, pp. 84–88.
14. M. Guzdial and J. Turns, Effective discussion through a computer-mediated anchored forum. *J. Learning Sci.*, **9** (4): 437–470, 2000.
15. A. J. Bernheim Brush, D. Bargeron, J. Grudin, A. Borning, and A. Gupta, Supporting interactions outside of class: Anchored discussions vs. discussion boards. in G. Stahl (ed.), *Computer Support for Collaborative Learning, Foundations for a CSCL Community*. Mahwah, NJ: LEA, 2002.
16. J. van der Pol, W. Admiraal, and P. R. J. Simons, The affordance of anchored discussion for the collaborative processing of academic texts, *J. Comput. Support. Collab. Learning*, **1** (3), 2006.
17. A. Kienle, Integration of knowledge management and collaborative learning by technical supported communication processes, *Educat. Info. Tech.*, **11** (2): 161–185, 2006.
18. G. Collison, B. Elbaum, S. Haavind, and R. Tinker, Facilitating Online Learning. Effective Strategies for Moderators, Madison, WI: Adwood Publishing, 2000.
19. G. Salmon, E-Moderating. The key to teaching and learning online, London, UK: Kogan Page, 2000.
20. G. Salmon, E-tivities. The key to active online learning, London, UK: Kogan Page, 2002.
21. H. F. Friedrich, F. W. Hesse, S. Ferber, and J. Heins, Partizipation im virtuellen Seminar in Abhängigkeit von der Moderationsmethode – eine empirische Untersuchung, in: *Die virtuelle Konferenz: Neue Möglichkeiten für die politische Kommunikation*, C. Bremer and M. Fechter (eds.), (Essen, Germany: Klartext), 1999.
22. A. Kienle and C. Ritterkamp, Facilitating asynchronous discussions in learning communities — The impact of moderation strategies, *Int. J. Behaviour Info. Tech.*, **26** (1): 73–80, 2007.
23. M. Hammond, Issues associated with participation in on line forums – the case of the communicative learner, *Educat. Inform. Tech.*, **4**, 353–367, 1999.
24. Z. L. Berge and M. P. Collins, Perceptions of E-moderators about their roles and functions in moderating electronic mailing lists, *Distance Ed. Int. J.* **21**, 81–100, 2000.
25. M. O. Thirunarayanan and A. Perez-Prado, Structured chat, 2007, Available: http://www.acm.org/ubiquity/views/m_thirunarayanan_2.html.

FURTHER READING

C. M. Hoadley and N. Enyeda, Between information and communication: Middle spaces in computer media for learning. *Proc. of the CSCL*1999, pp. 242–251.

L. Kimball, Managing distance learning – new challenges for faculty. In: R. Hazemi, S. Hailes, and S. Wilbur, (eds.), *The Digital University*, London, UK: Springer, 1998.

T. Koschmann, Dewey's contribution to a standard of problem-based learning, in P. Dillenbourg, A. Eurelings, K. Hakkarainen, (eds.), European Perspectives on Computer-Supported Collaborative Learning, *Maasticht*, The Netherland McLuhan Institute, 2001.

G. Stahl, A model of collaborative knowledge-building, *Proc. of the International Conference on the Learning Science* (ICLS), 2000.

Andrea Kienle
Fraunhofer IPSI
Darmstadt, Germany

COMPUTING ACCREDITATION: EVOLUTION AND TRENDS ASSOCIATED WITH U.S. ACCREDITING AGENCIES

INTRODUCTION

Accreditation is the primary quality assurance mechanism for institutions and programs of higher education, which helps a program prove that it is of a quality acceptable to constituents (1). Its original emergence in the United States was to ensure that federal student loans were awarded in support of quality programs. Since then, the need for accreditation has strengthened considerably, as the need for quality assurance continues to be in the forefront of educational concerns. For example, a recent report from the Commission on Higher Education (2) identified several problems with the education system in the United States, a central one being the overall quality of higher education.

The continuing interest in accreditation emerges of three factors. First, recognition exists that the public is not always in a position to judge quality, certainly not for programs or institutions for higher education. Therefore, a need exists for an independently issued stamp of approval for programs and institutions of higher education.

Second, a lack of oversight exists in higher education. Unlike the public school system, colleges and universities have considerable freedom when it comes to curriculum design, hiring practices, and student expectations (1). This lack of oversight requires a different mechanism to ensure quality, and higher education has opted to use accreditation for this purpose. Finally, many organizations have recognized the importance of continuous quality improvement, and higher education is no exception. As explained in this article, recent developments in accreditation reflect this trend. The objective of accreditation is not only to ensure that educational institutions strive for excellence but also to make certain that the process for ensuring high quality is apposite.

Two major types of accreditation are available in the United States: (*1*) institutional accreditation (also called regional accreditation in the United States) and (*2*) specialized accreditation. Specialized accreditation includes both program and school-specific accreditation, with the former applied to a unique program and the latter applied to an administrative unit within an institution. In the United States, institutional accreditation is generally the responsibility of a regional accreditation body such as the Southern Association of Colleges and Schools (SACS) (www.sacs.org) or the Middle States Commission on Higher Education (www.msche.org).

This article concentrates on accreditation in the computing discipline, focusing on the primary accreditation bodies accrediting U.S. programs, namely ABET, Inc. (www.abet.org) and the Association to Advance Collegiate Schools of Business (AACSB) (www.aacsb.edu). ABET, Inc. is the primary body responsible for specialized program-level accreditation in computing. AACSB, on the other hand, provides specialized accreditation at the unit level and accredits business schools only. The latter implies that all the programs offered within the unit are accredited, including any computing programs that it may offer.

Both accrediting bodies have concentrated primarily on the accreditation of programs housed within U.S. institutions. ABET has evaluated programs outside the United States for "substantial equivalency," which means the program is comparable in educational outcomes with a U.S.-accredited program. "Substantial equivalency" has been phased out, and international accreditation pilot visits are now being employed for visits within the United States. In 2003, AACSB members approved the international visits as relevant and applicable to all business programs and have accredited several programs outside of the United States.

The purpose of this article is to (*1*) review key concepts associated with accreditation, (*2*) discuss the state of computing accreditation by U.S. accrediting organizations and how accreditation has evolved through recent years, (*3*) describe the criteria for accreditation put forth by the primary agencies that review computing programs, and (*4*) review the typical process of accreditation. Although many concepts are applicable to any accreditation process, we refer to AACSB and ABET, Inc., which well are recognized agencies that ensure quality in educational units that include technology or specialized computing programs. ABET's Computer Accreditation Commission (CAC), as the primary agency targeting computing programs, is emphasized.

KEY CONCEPTS OF ACCREDITATION

For many years, accrediting organizations established focused and specific guidelines to which a program had to adhere to receive its stamp of approval. For instance, a fixed number of credits, required in a specific area, had been the norm, and any program that wished to be granted accreditation had to offer the required number of credits in relevant areas. Quality was measured through a checklist of attributes that were expected to be met by the various inputs into learning processes, such as curriculum, teaching faculty, laboratory, and other facilities and resources. The definition of quality, implicit in this approach to accreditation, was that of meeting specific standards, to be followed by every institution.

Some proof exists that in computer science education, this approach is successful. In a study of accredited and nonaccredited programs, Rozanski (3) reports that although similarities and differences exist between these programs, accredited programs have more potential to increase specific quality indicators.

In the past it was straightforward to determine whether a program or institution met the accreditation criteria. A

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

major drawback of this approach was that it forced uniformity among institutions, preventing innovation and the consideration of specialized needs of a program's or school's constituencies. Other controversies centered on the expense and time necessary to navigate successfully the process. Smaller schools especially felt the guidelines were targeted toward the larger institution (3).

Partly in response to concerns and to the danger of a lack of innovation and, at least in the United States, partly under pressure from the federal government, accreditation agencies have moved to an outcomes-based approach, and accreditation criteria now embody a definition of quality more in line with that adopted by many quality improvement approaches, namely "fitness for purpose" (4). The basis for this approach is the premise that quality is multifaceted.

From the overall mission of an institution, units or programs are expected to establish long-term educational objectives or goals, which describe achievements of graduates a few years after graduation, and to derive a set of learning outcomes, which are statements defined as the aforementioned skills, knowledge, and behaviors that students are expected to acquire in their matriculation through the program (5). Additionally, an institution or program is expected to establish an assessment process to determine how well its graduates are achieving its objectives and outcomes and to establish a quality enhancement program that uses the data collected through this assessment process to improve the program. Assessment processes foster program improvement by enabling visit teams to make judgments about program effectiveness in preparing graduates for entry into a field.

Some differences exist between the various accreditation agencies concerning the range of decisions that they can make. Clearly, each will have the option of whether to accredit or not. However, different options are available should it be determined that a program or institution does not meet all criteria. For example, some accreditation agencies may reduce the period for which the program or institution is accredited. Alternatively, they may provisionally accredit but make a final decision contingent on an interim report by the program or institution in which it makes clear how it has addressed any weaknesses or concerns identified during the team visit.

Programs continue to be reviewed cyclically. Again, differences exist between the various agencies relative to the maximum length for which a program or institution can be accredited. The ABET CAC maximally accredits a program for 6 years, whereas the AACSB has operated on a 5- or 10-year cycle and is moving more toward the use of maintenance reports.

Clearly, given the importance of the agency in the accreditation process, the question develops concerning the basis used to determine whether an organization can become an accreditation agency. The answer differs from country to country. In many countries, accreditation agencies are governmental or quasi-governmental organizations established through an act or parliament. In the United States, most accreditation agencies are essentially private organizations. However, under the Higher Education Act, the U.S. Secretary of Education is required to recognize an accreditation agency before students enrolled in programs or institutions accredited by it can receive federal funding.

Also, several professional organizations recognize accreditation agencies, chief among then are the International Network for Quality Assurance Agencies in Higher Education (INQAAHE) and the Council of Higher Education Accreditation (CHEA) in the United States.

ACCREDITATION AGENCIES RELEVANT TO COMPUTING

The AACSB

The AACSB International accredits both undergraduate and graduate education for business and accounting. Founded in 1916, the first standards were adopted in 1919 (6). The AACSB accredits computing programs, typically in Management Information Systems (MIS) or Information Systems (IS), only as part of an evaluation of a business program, and it does not review any one specific program. By accrediting the unit that offers the various business-related programs, it accredits indirectly any specific program offered within the unit. The one exception is accounting, which can receive a program-specific evaluation.

Both undergraduate and graduate programs must be evaluated in making an accreditation decision. The AACSB puts forth standards for continuous quality improvement. Important in the process is the publication of a mission statement, academic and financial considerations, and student support.

AACSB International members approved mission-linked accreditation standards and the peer review process in 1991, and in 2003, members approved a revised set of worldwide standards. The application of the AACSB's accreditation standards is based on the stated mission of each institution, and the standards thus provide enough flexibility so that they can be applied to a wide variety of business schools with different missions. This flexibility offers the opportunity for many institutions that offer online and other distance learning programs, as well as the more conventional on-campus programs, to be accredited.

ABET Inc.

In the early 1980s, computer science accreditation was initiated by groups from the Association for Computing Machinery, Inc. (ACM) and the Institute of Electrical and Electronics Engineers, Inc. Computer Society (IEEE-CS). Criteria for accreditation of computer science were established, and visits started in 1984. The initial visits were made by the Computer Science Accreditation Commission (CSAC), which in 1985 established the Computer Science Accreditation Board (CSAB) with the explicit purpose "to advance the development and practices of computing disciplines in the public interest through the enhancement of quality educational degree programs in computing" (http://www.csab.org).

Eventually CSAC was incorporated into ABET with CSAB remaining as the lead society within ABET for accreditation of programs in computer science, information systems, information technology, and software engineering. It should be noted that the ABET Engineering Accreditation Commission (EAC) is responsible for software engineering accreditation visits. In this capacity, the CSAB is responsible for recommending changes to the accreditation criteria and for the recruitment, selection, and training of program evaluators (PEVs). All other accreditation activities, which were conducted previously by the CSAC, are now conducted by the ABET CAC.

The CSAB is governed by a Board of Directors whose members are appointed by the member societies. The current member societies of the CSAB, which include the ACM and the IEEE-CS, as well as its newest member, the Association for Information Systems (AIS), are the three largest technical, educational, and scientific societies in the computer and computer-related fields.

Since the incorporation of the CSAC into the ABET, computing programs have been accredited by the ABET CAC. The first programs accredited by the ABET CAC were in computer science (CS). In 2004, IS criteria were completed and partly funded by a National Science Foundation grant. With the addition of IS criteria, the scope of computing program accreditation was enlarged. This addition has been followed by the addition of information technology (IT), with criteria currently being piloted.

The CAC recognized a need to address the growing number of programs in emerging computing areas, which has resulted in even more revision of accreditation criteria, allowing such programs to apply for accreditation under computing general criteria. Thus, these areas can benefit from accreditation as well. We discuss these revisions in the next section.

THE AACSB AND ABET CAC ACCREDITATION CRITERIA

The AACSB Criteria

Although the AACSB does not accredit IS programs by themselves, the standards used support the concept of continuous quality improvement and require the use of a systematic process for curriculum management. Normally, the curriculum management process will result in an undergraduate degree program that includes learning experiences in such general knowledge and skill areas as follows (7):

- Communication abilities
- Ethical understanding and reasoning abilities
- Analytic skills
- Use of information technology
- Multicultural and diversity understanding
- Reflective thinking skills

Recent changes required of schools seeking accreditation include (8):

- Assessment activities are focused toward degree programs rather than toward the majors within a degree program (AACSB, 2006). In other words, recent criteria focus on learning goals applied to each degree rather than an separate majors.
- The requirements of direct measures measuring knowledge or skills that students will expected to attain by the time they graduate.
- The development of learning goals for the overall degree program.
- Involvement of faculty members to a far greater extent than under prior standards.
- The involvement of faculty to effect improvement.

The ABET Criteria

For several reasons, the ABET CAC revised significantly its accreditation criteria, a copy of which is available for inspection and comment from the ABET website (www.abet.org). The ABET is currently piloting the proposed criteria, which are expected to be officially in place for the 2008–2009 accreditation cycle.

Two significant revisions were made to the criteria. First, following the lead of, in particular, the ABET's Engineering Accreditation Commission (EAC), the criteria have been reorganized into a set of general criteria that apply to all programs in computing, and program-specific criteria for programs in CS, IS, and IT. For any program to be accredited in one of these specific disciplines, it must meet both the general and the associated program-specific criteria. However, programs in emerging areas of computing that are not strictly CS, IS or IT, such as programs in computer game design, or telecommunications, will be accredited under the ABET CAC's general criteria. The revision thus broadened the range of computing programs that can benefit from the ABET CAC accreditation.

Second, although criteria have required programs to establish program educational objectives and outcomes, and to set up an assessment and quality improvement process, it was not emphasized sufficiently. The revised criteria place greater emphasis on the need to set up a continuous improvement process.

The proposed CAC criteria for all computing programs are divided into nine major categories (9):

1. Students
2. Objectives
3. Outcomes
4. Continuous improvement
5. Curriculum
6. Faculty
7. Facilities
8. Support
9. Program criteria

The criteria are outcomes based, and it is expected that program outcomes are to be based on the needs of the program's constituencies. However, the criteria also will specify a minimum set of skills and knowledge that students must achieve by graduation. The general criteria specify that students must be able to demonstrate minimally the following (9):

(a) An ability to apply knowledge of computing and mathematics appropriate to the discipline

(b) An ability to analyze a problem and to identify and define the computing requirements appropriate to its solution

(c) An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs

(d) An ability to function effectively on teams to accomplish a common goal

(e) An understanding of professional, ethical, legal, security, and social issues and responsibilities

(f) An ability to communicate effectively with a range of audiences

(g) An ability to analyze the local and global impact of computing on individuals, organizations, and society

(h) Recognition of the need for, and an ability to engage in, continuing professional development

(i) An ability to use current techniques, skills, and tools necessary for computing practice

To this criteria, computer science adds the following:

(j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices

(k) An ability to apply design and development principles in the construction of software systems of varying complexity

Information systems adds:

(j) An understanding of processes that support the delivery and management of information systems within a specific application environment

Whereas information technology adds:

(j) An ability to use and apply current technical concepts and practices in the core information technologies

(k) An ability to identify and analyze user needs and take them into account in the selection, creation, evaluation, and administration of computer-based systems

(l) An ability to effectively integrate IT-based solutions into the user environment

(m) An understanding of best practices and standards and their application

(n) An ability to assist in the creation of an effective project plan.

Similarities and Differences

Many similarities exist between accreditation criteria formulated by the ABET CAC and the AACSB. Both organizations stress the need for explicit learning outcomes for graduating students and for explicitly documented assessment and quality improvement processes. Both organizations also recognize the need for graduates to be well rounded with qualities beyond skills needed to understand specialized areas. Both accrediting bodies, as do many others, now require more than perceptions of constituents to determine the level of program accomplishments. A need for more direct assessment of knowledge and skills is required.

It should also be noted that IS programs offered through institutions accredited by the AACSB may also be accredited by the ABET. Indeed, a handful of ABET-accredited programs in IS are offered in AACSB-accredited business schools. Programs that are accredited by both organizations offer constituencies the added benefit of knowing that IS is offered within a high-quality business program and that it has a quality technology component integrated into the program.

Both agencies include in their accreditation criteria similar sets of attributes that they expect graduating students to achieve. For instance, the AACSB includes a management of curricula criterion that requires an undergraduate degree program to include learning experiences in specific general knowledge and skill areas as depicted above.

The difference in level of detail between the minimal learning outcomes in the ABET CAC accreditation criteria and those in the AACSB accreditation can be explained by the fact that the ABET CAC criteria are more focused. Both sets of evaluative rules promote continuous quality improvement. Rather than being designed for a class of programs, as the AACSB criteria are, the ABET CAC criteria focus on a single type of program. They can, therefore, be more specific about the expectations of graduates, especially in the technology area. Note, however, that both the ABET CAC and the AASCB merely formulate a minimal set of guidelines. Specific programs, whether they apply for accreditation under the ABET CAC criteria or under the AACSB criteria, are expected to formulate their own sets of objectives and learning goals (AACSB terminology) or outcomes (ABET terminology). Moreover, both insist that the specific objectives adopted be based on the needs of their specific constituencies rather than on the whims of faculty or other involved parties.

Although the ABET CAC is more specific in the specification of minimal outcomes, the AACSB is generally more specific when it comes to some other criteria. For example, the ABET CAC provides relatively general requirements for faculty. The faculty responsible for the program must have the required skills to deliver the program and to modify it, and some of them must also possess terminal degrees. The AACSB, on the other hand, provides a set of detailed guidelines that spell out the qualifications that faculty must have and what percentage of courses within a program must typically be covered by qualified faculty. However, such differences should not detract from the

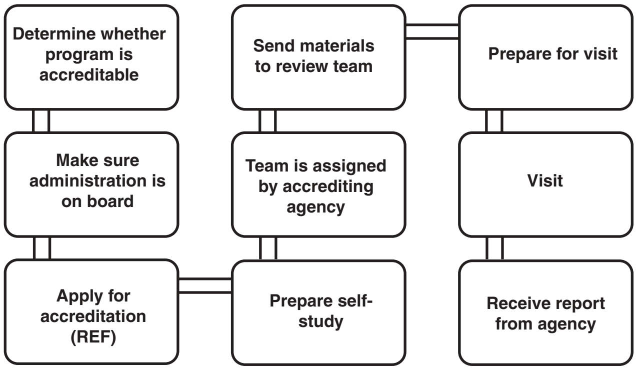

Determine whether program is accreditable
Make sure administration is on board
Apply for accreditation (REF)
Prepare self-study
Team is assigned by accrediting agency
Send materials to review team
Prepare for visit
Visit
Receive report from agency

Figure 1. The process of accreditation.

fact that in both cases, continuous quality improvement is of central importance.

THE TYPICAL ACCREDITATION PROCESS

Program or Unit Being Accredited

The process of obtaining accreditation typically begins with the institution or program completing a self-analysis to determine whether the program meets the demands of the accrediting agency. Gorgone et al. (10) recommend that the accreditation process needs to begin at least a year before the time of an anticipated visit. Given that this step goes well, and the administration is supportive, a Request for Evaluation (RFE) begins the formal process. Figure 1 illustrates these and subsequent steps that include the preparation of a self-study, collection of materials for a review team, and the culminating accreditation visit. Most accreditation agencies prefer a program or institution to have completed its self-study before it applies for accreditation (and pays its fees).

Once the request for accreditation has been received by the accreditation agency, it appoints several program evaluators, typically in consultation with the program or institution seeking accreditation. Program evaluators are peers, often drawn from academia but not infrequently drawn from industry. The accreditation agency also appoints a team chair, again typically in consultation with the program or institution.

The remainder of the process is driven by the self-study. The visiting team will visit the program or institution. The primary purpose of the site visit is to verify the accuracy of the self-study and to make observations regarding issues that are hard to gauge from a self-study, such as faculty and staff morale and the students' view of the institution or program. The team writes a report of its findings, which is submitted to the accreditation agency. Generally, the institution or program is allowed to make comments on early drafts of the report, which may lead the visiting team to revise its report. Eventually, the accreditation agency will make its decision.

The Accrediting Agency

The accrediting agency has an enormous amount of responsibility. Typically a professional staff supports the accreditation process, and academic and industry volunteers in the areas being evaluated create the criteria. Associated tasks are summarized as (*1*) the identification of team chairs, (*2*) the selection and identification of institutions to visit, (*3*) the assignment and training of team chairs and a review team, and (*4*) the preparation of sample forms and reports (8).

CONCLUSION

Quality assurance in higher education has become an important issue for many. This article describes the primary quality assurance mechanism for higher education, namely accreditation. It has emphasized accreditation for programs in computing.

Programs or institutions that have voluntarily assented to the accreditation process and have achieved accreditation meet the quality that reasonable external stakeholders can expect them to have. Moreover, the emphasis on outcome-based criteria and the concomitant requirements that programs or institutions put in place for accreditation provides constituents with the assurance of quality and continuous quality improvement. Although we do dispute the fact that accreditation is the only way to assure quality in higher education, we do believe that accreditation is an excellent method for doing so and that every program or institution that has been accredited by a reputable accreditation agency is of high quality.

BIBLIOGRAPHY

1. D. K. Lidtke and G. J. Yaverbaum, Developing accreditation for information system education, *IT Pro*, **5**(1): 41–45, 2003.
2. U.S. Department of Education, [Online] *A test of leadership: charting the future of U.S. Higher Education*. Washington, D.C., U. S. Department of Education, 2006. Available: http://www.ed.gov/about/bdscomm/list/hiedfuture/reports/pre-pub-report.pdf.
3. E. P. Rozanski, Accreditation: does it enhance quality, *ACM SIGCSE Bulletin, Proceedings of the Twenty-fifth SIGCSE Symposium on Computer Science Education*, **26**(1): 145–149, 1994.
4. D. Garvin, What does product quality really mean? *Sloan Management Review*, **26**(1): 25–43, 1984.
5. ABET, Inc., Accreditation policy and procedure manual, 2007. Available: http://www.abet.org/forms.shtml.

6. AACSB International. Accreditation standards, 2007. Available: http://www.aacsb.edu/accreditation/standards.asp.
7. AACSB International, Eligibility procedures and accreditation standards for business accreditation, 2007. Available: http://www.aacsb.edu/accreditation/process/documents/AACSB_STANDARDS_Revised_Jan07.pdf.
8. C. Pringle, and M. Mitri, Assessment practices in AACSB business schools, *J. Educ. Business*, **4**(82), 202–212, 2007.
9. ABET, Inc. Criteria for accrediting programs, 2007. Available: http://www.abet.org/forms.shtml#For_Computing_Programs_Only.
10. J. Gorgone, D. Lidtke and D. Feinstein, Status of information systems accreditation, *ACM SIGCSE Bulletin*, **33**(1): 421–422, 2001.

FURTHER READING

D. Crouch and L. Schwartzman, Computer science accreditation, the advantages of being different, *ACM SIGCSE Bulletin*, **35**(1): 36–40, 2003.

J. Impagliazzo, J. Gorgone, Professional accreditation of information systems programs, *Communications of the AIS*, **9**: 2002.

J. Gorgone, D. Feinstein, and D. Lidtke, Accreditation criteria format IS/IT programs, *Informa. Syst.*, **1**: 166–170, 2000.

L. G. Jones and A. L. Price, Changes in computer science accreditation, *Communicat. ACM*, **45**: 99–103, 2002.

W. King, J. Gorgone, and J. Henderson, Study feasibility of accreditation of programs in computer information science/systems/technology. NSF Grant, 1999–2001.

D. K. Lidtke, K. Martin, L. Saperstein, and D. Bonnette, What's new with ABET/CSAB integration, *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, ACM Press, 2001, p. 413.

K. You, Effective course-based learning outcome assessment for ABET accreditation of computing programs, *Consortium for Computing Sciences in Colleges, South Central Conference*, 2007.

GAYLE J. YAVERBAUM
Penn State Harrisburg
Harrisburg, Pennsylvania

HAN REICHGELT
Southern Polytechnic State
University
Marietta, Georgia

C

CYBERNETICS

The word cybernetics was coined by Norbert Wiener (1) and denotes a subject area that he indicated by the subtitle of his 1948 book: "control and communication in the animal and the machine." It is derived from the Greek word for "steersman," and the word "governor" comes from the same root. The word had been used in a related but more restricted sense by Ampère in the nineteenth century to denote a science of government, but Wiener initially was not aware of this. It also was used much earlier by Plato with a similar meaning.

The linking of "animal" and "machine" implies that these have properties in common that allow description in similar terms. At a simple level this view was not new, because nerves were identified as communication pathways by Descartes in the seventeenth century; however, later developments, especially during World War II and including the emergence of analog and digital electronic computers, allowed a deeper and more fruitful unified approach. It also was intended that "animal" should be understood to include organizations and societies, and later work increasingly has focused on them. A revised definition that has been suggested is: "communication and control within and between man, organizations and society."

HISTORY OF CYBERNETICS

The publication of the book by Wiener gave name and status to the subject area, but earlier origins can be traced. At the time of the publication, the ideas also were promoted vigorously by Warren McCulloch (2), and the emergence of cybernetics has been attributed to the meeting and collaboration of Wiener with McCulloch and Walter Pitts (3). McCulloch was a neurophysiologist who epitomized his own lifetime quest as the attempt to answer the question: "What is a number, that a man may know it, and a man, that he may know a number?" This question led him to study medicine and, particularly, neurophysiology. An international center for studies of what became known as cybernetics was planned by him in the 1920s but had to be abandoned for financial reasons in the 1930s.

Before the meeting with McCulloch, (2) a number of influences guided Wiener toward the initiative, a major one being his wartime work on the possibility of a predictor to extrapolate the curving path of an enemy aircraft so as to direct anti-aircraft gunfire more effectively. He had exposure to biological studies in a number of contexts, including introduction to electroencephalography by W. Grey Walter in the United Kingdom (4) and work on heart muscle with the Mexican physiologist Rosenblueth; he also had learned about biological homeostasis in discussions with Walter Cannon. He also had been involved with analog and digital computing.

The topic area, and its designation by name, were advanced greatly by a series of discussion meetings held in New York and sponsored by the Josiah Macy, Jr. Foundation between 1946 and 1953. Ten meetings took place, chaired by Warren McCulloch (2) and with the participation of scientists from different specializations, especially bringing together biological and nonbiological sciences. The first five meetings, which were not recorded in print, had various titles that referred to circular mechanisms and teleology. From the sixth meeting onward, the proceedings were edited by Heinz von Foerster and published. Wiener's book had appeared in the meantime, and in honor of his contribution, the reports on the last five meetings were entitled: *Cybernetics: Circular, Causal and Feedback Mechanisms in Biological and Social Systems*.

Relevant developments were not confined, however, to the United States. In Britain, an informal group called the Ratio Club was founded in 1949 and developed many of the basic ideas of cybernetics at first independently of the American work, although links were formed later. It also is noteworthy that the book by Wiener was published in French before the appearance of the English version and gave rise to a Circle of Cybernetic Studies in Paris. The Ratio Club was considered to have served its purpose, and the transatlantic separation ended by a conference in the National Physical Laboratory in Teddington, United Kingdom in 1958.

FEEDBACK AND SERVOMECHANISMS

What has been termed circular causation is a central characteristic of living systems as well as of many modern artifacts. The flow of effect, or causation, is not linear from input to output but has loops or feedbacks. The system is sensitive to what it, itself, influences, and it regulates its influence so as to achieve a desired result. Conscious muscular movement is an obvious example, where the actual movement is monitored by proprioceptors in joints and muscles and perhaps also visually and the force exerted by the muscles is regulated using this feedback so as to keep the movement close to what is wanted despite unknown weight to be lifted, friction, and inertia. The feedback that produces stability is negative feedback, which is to say that an excessive movement in the desired direction must cause diminution of muscular effort and conversely for insufficient movement.

Servomechanisms are artificial devices that similarly use negative feedback. They featured strongly in wartime applications, such as control of aircraft gun turrets, but were known much earlier, examples being the steering engines of ships and Watt's governor to regulate the speed of a steam engine. The use of the term "governor" here encouraged the later appellation of Wiener. Negative feedback also appears in a crude but effective form in the familiar ball-cock valve of the toilet cistern, which regulates inflow according to the sensed water level, and in thermostats in rooms, ovens, and refrigerators. Regulation of

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

temperature and many other variables by feedback from sensors also is a vital feature of biological systems.

Servomechanisms are required to respond rapidly to disturbances and, at the same time, to give stable control without large overshoots or oscillation. The achievement of this stability where the environment includes inertia and elasticity and viscous friction depends on the mathematical methods of Wiener and others, including Bode and Nyquist. One connection with biology noted by Wiener is that the tremors of a patient with Parkinson's disease are similar to the operation of a maladjusted servo. Negative feedback also is applied in electronic amplification and allows high-fidelity audio reproduction despite the nonlinear characteristics of the active components, whether vacuum tubes or transistors; important mathematical theory has been developed in this context.

NEUROPHYSIOLOGY AND DIGITAL COMPUTING

The nervous system is the most obvious and complex form of communication "in the animal," although the endocrine system also operates in a broadcast or "to whom it may concern" fashion and other forms are playing a part. McCulloch looked to neurophysiology for answers to the question he posed, and 5 years before the appearance of the book by Wiener, he and Walter Pitts published a mathematical proof of the computational capabilities of networks of model neurons that have some correspondence to the real kind. Much speculation was aroused by the apparent correspondence between the all-or-nothing response of neurons and the use of binary arithmetic and two-valued logic in digital computers.

In this and other ways, interest in cybernetics arose from advances in electronics, especially those stimulated by World War II. Young scientists who had worked on military projects wanted to turn their skills to something to help humanity, such as biological research. Ways were found of making micro electrodes that allowed recording from single neurons and stimulating them, and it looked as though the nervous system could be analyzed like an electronic device.

A great deal has been learned about the nervous system using microelectrodes, not least by a group around Warren McCulloch in the Massachusetts Institute of Technology in the 1950s and 1960s. Further insight came from theoretical treatments, including the idealization of neural nets as cellular automata by John von Neumann and the mathematical treatment of morphogenesis (the development of pattern or structure in living systems) pioneered by Alan Turing. Nevertheless, much of the working of the central nervous system remains mysterious, and in recent decades, the main focus of cybernetics has shifted to a higher-level view of thought processes and to examination of inter personal communication and organizations.

A recurring theme is that of self-organization, with several conferences in the 1950s and 1960s nominally devoted to discussion of self-organizing systems. The aim was to study learning in something like a neural net, associated with the spontaneous emergence of structure or at least a major structural change. Von Foerster treated the topic in terms of thermodynamics and pointed out that spontaneous emergence in an isolated system would be contrary to the second law. More recently, self-organization has been discussed rather differently with emphasis on the emergence of structure as such, independent of learning or goal-seeking.

Ilya Prigogine (5) and his followers have resolved the contradiction between the apparent implications of the second law of thermodynamics, that entropy or disorder can increase only, and the observed increase of order in biological evolution. The contradiction is resolved by observing that the second law applies to systems close to equilibrium, and living systems only exist far from equilibrium and can be termed "dissipative structures" as they receive energy and pass it to their environments. It has been shown that spontaneous emergence of structure is natural in such conditions. Implications for psychology and social systems, as well as cosmology, are drawn.

INFORMATION THEORY

An important part of cybernetics commonly is called information theory, although arguably more appropriately termed communication theory. It depends on the fact that information, in a certain sense of the word, can be measured and expressed in units. The amount of such information in a message is a measure of the difficulty of transmitting it from place to place or of storing it, not a measure of its significance. The unit of information is the "bit," the word being derived from "binary digit." It is the amount of information required to indicate a choice between two possibilities that previously were equally probable. The capacity of a communication channel can be expressed in bits per second. The principal author of the modern theory is Claude Shannon (6), although a similar measure of information was introduced by R. V. L. Hartley as early as 1928. The later work greatly extends the theory, particularly in taking account of noise.

In its simple form, with reference to discrete choices, the theory accounts nicely for some biological phenomena, for instance, the reaction times of subjects in multiple-choice experiments. It is extended to apply to continuous signals and to take account of corruption by random disturbances or "noise." The mathematical expressions then correspond, with a reversal of sign, to those expressions for the evaluation of entropy in thermodynamics. The theory applies to the detection of signals in noise and, therefore, to perception generally, and one notable treatment deals with its application to optimal recovery and detection of radar echoes in noise.

The effects of noise often can be overcome by exploiting redundancy, which is information (in the special quantitative sense) additional to that needed to convey the message in the absence of noise. Communication in natural language, whether spoken or written, has considerable redundancy, and meaning usually can be guessed with a fair degree of confidence when a substantial number of letters, syllables, or words effectively are lost because of noise, interference, and distortion. Much attention has been given to error-detecting and error-correcting coding that allow the introduction of redundancy in particularly effective ways. One theorem of information theory refers to the necessary capacity of an auxiliary channel to allow the correction of a corrupted message and corresponds to Ashby's (7) principle of requisite variety, which has found important application in management.

ARTIFICIAL INTELLIGENCE

In the attempt to understand the working of the brain in mechanistic terms, many attempts were made to model some aspect of its working, usually that of learning to perform a particular task. Often the task was a form of pattern classification, such as recognition of hand-blocked characters. An early assumption was that an intelligent artifact should model the nervous system and should consist of many relatively simple interacting units. Variations on such a scheme, indicated by the term "perceptron" devised by Frank Rosenblatt, could learn pattern classification but only of a simple kind without significant learned generalization. The outcomes of these early attempts to achieve "artificial intelligence" were not impressive, and at a conference in 1956 the term "Artificial Intelligence" (with capitals), or AI, was given a rather different meaning.

The aim of the new AI was to use the full power of computers, without restriction to a neural net or other prescribed architecture, to model human capability in areas that are accepted readily as demonstrating "intelligence." The main areas that have received attention are as follows:

Theorem Proving. The automatic proving of mathematical theorems has received much attention, and search methods developed have been applied in other areas, such as path planning for robots. They also are the basis of ways of programming computers declaratively, notably using the language PROLOG, rather than procedurally. In declarative programming, the required task is presented effectively as a mathematical theorem to be proved and, in some application areas, allows much faster program development than is possible by specifying procedures manually.

Game Playing. Chess has been seen as a classical challenge, and computers now can compete at an extremely high level, such that a computer beat the highest-scoring human chess player in history, Gary Kasparov. Important pioneering work was done using the game of checkers (or "draughts").

Pattern Recognition. Pattern recognition can refer to visual or auditory patterns; or patterns in other or mixed modalities; or in no particular modality, as when used to look for patterns in medical or weather data. Attention also has been given to the analysis of complete visual scenes, which presents special difficulty because, among other reasons, objects can have various orientations and can obscure each other partially. Scene analysis is necessary for advanced developments in robotics.

Use of Natural Language. Question-answering systems and mechanical translation have received attention, and practical systems for both have been implemented but leave much to be desired. Early optimistic predictions of computer performance in this area have not materialized fully. This lack is largely because the "understanding" of text depends on semantic as well as syntactical features and, therefore, on the huge amount of knowledge of the world that is accumulated by a person.

Robotics. Robotics has great practical importance in, for example, space research, undersea exploration, bomb disposal, and manufacturing. Many of its challenges are associated with processing sensory data, including video images, so as to navigate, recognize, and manipulate objects with dexterity and energy efficiency. Apart from their immediate use, these developments can be expected to throw light on corresponding biological mechanisms. Bipedal locomotion has been achieved only with great difficulty, which shows the complexity of the biological control of posture and balance. For practical mobile robots, wheeled or tracked locomotion is used instead. A topic area associated with advanced robotics projects is that of virtual reality, where a person is given sensory input and interactions that simulate a nonexistent environment. Flight simulators for pilot training were an early example, and computer games implement the effect to varying degrees.

Expert Systems. This term has been used to refer to systems that explicitly model the responses of a human "domain expert," either by questioning the expert about his/her methods or deriving rules from examples set to him/her. A favourite application area has been the medical diagnosis in various specializations, both for direct use and for training students. The general method has been applied to a very wide range of tasks in which human judgement is superior to any known analytic approach. Under the general heading of diagnosis, this range of topics includes fault finding in computers and other complex machinery or in organizations. Other applications are made to business decisions and military strategy.

A great deal has been achieved under the heading of AI. It has underlined the importance of heuristics, or rules that do not always "work" (in the sense of leading directly to a solution of a problem). Heuristics indicate where it may be useful to look for solutions and are certainly a feature of human, as well as machine, problem-solving. In this and other ways, studies of AI have contributed to the understanding of intelligence, not least by recognizing the complexity of many of the tasks studied. Apart from this, the influence of AI studies on computer programming practice has been profound; for example, the use of "list-processing," which has sometimes been seen as peculiar to AI programs, is used widely in compilers and operating systems.

Nevertheless, progress in AI is widely felt to have been disappointing. Mathematical theorem-proving and chess playing are forms of intellectual activity that people find difficult, and AI studies have produced machines proficient in them but unable to perform ordinary tasks like going round a house and emptying the ashtrays. Recognizing chairs, tables, ashtrays, and so forth in their almost infinite variety of shapes and colors is hard because it is hard to define these objects in a way that is "understandable" to a robot and because more problems arise in manipulation, trajectory planning, and balance. If the evolution of machine intelligence is to have correspondence to that of natural intelligence, then what are seen as low-level manifestations should appear first. The ultimate possibilities for machine intelligence were discussed comprehensively by Turing and more recently and sceptically using the parable of Searle's "Chinese Room" in which an operator manipulates symbols without understanding.

In relatively recent decades a revival of interest has grown in artificial neural nets (ANNs). This revival of interest is attributable partly to advances in computer technology that make feasible the representation and manipulation of large nets, but a more significant factor is the invention of useful ways of implementing learning in ANNs. The most powerful of these ways is "backpropagation," which depends on information pathways in the net that are additional to those serving its primary function, conducting in the opposite direction. Some applications are of a "control" or continuous-variable kind where the net provides a means of learning the continuous relation between a number of continuous variables, one of them a desired output that the net learns to compute from the others. Other application areas have an entirely different nature and include linguistics.

These relatively recent studies have been driven mainly by practical considerations, and the correspondence to biological processing often is controversial. The "backpropagation of error" algorithm, the basis of the majority of applications, has been argued to be unlikely to operate in biological processing. However, other forms of backpropagation probably do play a part, and biological considerations are invoked frequently in arguing the merits of schemes using ANNs.

CYBERNETIC MACHINES

Because a main aim of many cyberneticians is to understand biological learning, various demonstrations have involved "learning machines" realized either as computer programs or as special-purpose hardware. The various schemes for artificial neural nets are examples, and an earlier one was the "Homeostat" of Ross Ashby (7), which sought a stable equilibrium despite disturbances that could include alteration of its physical structure. A number of workers, starting with Grey Walter (4), made mobile robots or "tortoises" (land turtles) that showed remarkably lifelike behavior from simple internal control arrangements. They could avoid obstacles and would seek "food" (electric power) at charging stations when "hungry." The "Machina speculatrix" by Grey Walter did not learn, actually, but later developments implemented learning in various forms.

A task that has been used in a number of studies is pole-balancing, where the pole is an inverted pendulum constrained to pivot about a single axis and mounted on a trolley. The task is to control the trolley so that the pole does not fall and the trolley remains within a certain length of track. The input data to the learning controller are indications of the position of the trolley on the track and of the angle of the pendulum, and its output is a signal to drive the trolley. In one study, the controller was made to copy the responses of a human performing the task; in others, it developed its own control policy by trial.

Learning, unless purely imitative, requires feedback of success or failure, referred to as reinforcement. The term "reinforcement learning," however, has been given special significance as indicating methods that respond not only to an immediate return from actions but also to a potential return associated with the change of state of the environment. A means of estimating an ultimate expected return, or value, for any state has to exist. The most favorable action is chosen to maximize the sum of the immediate return and the change in expected subsequent return. The means of evaluating states is subject, itself to modification by learning.

This extension of the meaning of "reinforcement learning," having some correspondence to the "dynamic programming" of Richard Bellman, has led to powerful learning algorithms and has been applied successfully to the pole-balancing problem as well as to writing a program that learned to play a very powerful game of backgammon.

CYBERSPACE

Interactions using the Internet and other channels of ready computer communication are said to occur in, and to define, cyberspace. The new environment and resulting feeling of community are real and amenable to sociological examination. The prefix "cyber-" is applied rather indiscriminately to any entity strongly involving computer communication, so that a café offering its customers Internet access is termed a "cybercafé", the provision of bomb-making instructions on the Internet is described as "cyberterrorism," and so on. In science fiction, such terms as "cybermen" have been used to refer to humans who are subject to computer control. These uses of the prefix must be deprecated as supporting an erroneous interpretation of cybernetics.

SECOND-ORDER CYBERNETICS

The idea of the circular causality implicit in cybernetics has been extended, originally by Heinz von Foerster, to include the circle comprising an observed system and the observer. This extension is a departure from traditional Newtonian science and from earlier views of cybernetics where the observer is assumed to have autonomy that puts him or her outside any causal loop. The earlier version of cybernetics is termed "first-order"; the extension is called "second-order" or "cybernetics of cybernetics."

The extension is most clearly relevant to the observation of social systems and, hence also, to teaching and management. In these contexts, an observer must either be part of the observed system or have involvement with it. In other contexts, such as those of the so-called exact sciences, the involvement of the observer may be less obvious, but still, complete objectivity is impossible.

The impossibility of access to anything to be called reality also has been recognized under the heading of constructivism, a philosophical viewpoint that predates cybernetics. Clearly the construction formed by an individual has to allow effective interaction with the environment if he or she is to operate effectively and, indeed, to survive, but no precise image or model is implied by this construction.

MANAGEMENT

The application of cybernetics to management was pioneered by Stafford Beer (8,9) and is a major field of interest. In an early paper, he listed characteristics of a cybernetic, or viable, system that include internal complexity and the capability of self-organization, along with a means of interacting appropriately with its environment. He indicated points of similarity between the communications and control within a firm and a human or animal central nervous system. For example, he likened the exclusive pursuit of short-term profit by a firm to the behavior of a "spinal" dog deprived of cerebral function.

The view of human organizations as viable is supported by the observation that groups of people in contact spontaneously form a social structure. The viability of organizations has been described as "social autopoiesis," as part of sociocybernetics where autopoiesis is a principle originally used with reference to biological systems to indicate self-production.

The Beer "Viable System Model" (8,9) has found wide application in management studies and depends on his listing of a set of components that are essential for viability. Failures or inadequacies of management performance may be attributed to absence or weakness of one or more of these components. It is asserted that viable systems have recursive character in that they have other viable systems embedded in them and are themselves components of larger ones. The cells of the body are viable systems that are embedded in people, who in turn form organizations, and so on. Connections to higher and lower viable systems are part of the model.

An aspect emphasized by Beer (8,9) is the need for a rapid response to disturbances, achieved in the nervous system by local reflexes, which respond automatically but also are subject to higher-level control. His work also makes extensive use of the Ashby (7) principle of Requisite Variety, which corresponds to a theorem of the Shannon Information Theory and states that a disturbance of a system only can be corrected by a control action whose variety, or information content, is at least equal to that of the disturbance. This view has been epitomized as: "only variety can absorb variety." Beer (8,9) analyzed many management situations in terms of variety, alternatively termed complexity, and claimed to be practising "complexity engineering."

SYSTEMS SCIENCE

Cybernetics is concerned essentially with systems, and valuable discussions of the meaning of "system" appear in works of Ashby (7), Pask (10,11), and Beer (8,9). No firm distinction exists between cybernetics and systems science except for a difference of emphasis because of the essentially biological focus of cybernetics. However, the seminal work of Ludwig von Bertalanffy (12)on systems theory has very substantial biological content.

SOCIOCYBERNETICS

Cybernetics has social implications under two distinct headings. One is the social effect of the introduction of automation, including, with computers and advanced robots, the automation of tasks of intellectual and skilled nature. Norbert Wiener, in his book *The Human Use of Human Beings*(13), expressed his grave concern over these aspects.

The topic that has come to be termed sociocybernetics is not concerned primarily with these aspects but with the examination of social systems in terms of their control and informational features and with the use of concepts from cybernetics and systems theory to describe and model them. The need for second-order cybernetics became particularly clear in this context, and its use has allowed valuable analyses of, for example, international and inter faith tensions and aspects of the "war on terror."

The theory of autopoiesis, or self-production, developed originally by Humberto Maturana (14) and Francisco Varela with reference to living cells, has been applied to the self-maintenance of organizations in society as "social autopoiesis." This term is applied by Niklas Luhmann (15) to various entities, including the legal system. The autopoietic character is indicated by the reference to "organizational (or operative) closure." In management and social studies, the correspondence of a social entity to a living organism is emphasized frequently by reference to pathology and diagnosis. The theoretical treatment of self-organization due to Prigogine (5), mentioned earlier, has been applied in social studies.

A range of topics bearing on psychology and sociology were treated in cybernetic terms by Gregory Bateson (16) and his wife the anthropologist Margaret Mead, both participants in the Macy conference series. His insights were based on experience of anthropological fieldwork in Bali and New Guinea, as well as psychiatry among schizophrenics and alcoholics and the study of communication behavior in octopi and dophins. Another study that bears particularly on education is the "conversation theory" of Gordon Pask (10,11), with the aim of exteriorizing thought processes in managed conversations.

GAIA

In 1969 James Lovelock (17) advanced the suggestion that the totality of living things in the biosphere acts like one large animal to regulate environmental variables. This hypothesis has been termed the Gaia hypothesis, where the name Gaia was one given to the Greek earth goddess. It was assumed previously that environmental conditions on the earth (temperature, ocean salinity, oxygen concentration of the atmosphere, and so on) just happened to be compatible with life. Lovelock points out that these variables have remained remarkably steady despite disturbances, including a large change in the strength of solar radiation.

The environment is influenced by biological activity to a greater extent than usually is realized; for example, without life almost no atmosphere would exist. This influence makes regulation feasible, and the Lovelock theory (17) has given accurate predictions, including a gloomy view of the likely consequences of the effects of human activity on Gaia, especially with regard to carbon emissions and other factors that contribute to global warming. It is widely accepted, in fact, that the greatest threat to humanity is that the feedback mechanisms that regulate the temperature of the planet may be overwhelmed by the ignorant, selfish, and short-sighted behavior of humans and that a greater understanding of these issues urgently is required.

Lovelock (17) has suggested how regulation could have come into being, with the help of a parable called Daisyworld. Daisyworld is a simple model of a planet on which two species of plant ("daisies") grow, one species black and the other white. The dependence of growth rate on temperature is the same for both species. The spread of black daisies causes the planet to become warmer, and the spread of white daisies affects the planet conversely.

It has been shown that with reasonable assumptions about heat conductivity of the planet, such that black daisies are a little warmer than the planet generally and white ones a little cooler, effective regulation of temperature can result. Apart from its environmental significance, the mechanism is interesting as an example of control without an obvious set-point.

OTHER TOPICS

Other topics, among many, that impinge on cybernetics include the use of "fuzzy" methods to allow operation under uncertainty, based on the introduction of the fuzzy set theory by Lotfi Zadeh (18), as well as methods for the study of complex systems under headings of chaos and fractals and artificial life.

BIBLIOGRAPHY

1. N. Wiener, *Cybernetics or Control and Communication in the Animal and the Machine*, New York: Wiley, 1948.
2. W. S. McCulloch, What is a number, that a man may know it, and a man, that he may know a number?, *General Semantics Bulletin*, **26 & 27**: 7–18; reprinted in W. S. McCulloch, *Embodiments of Mind*. Cambridge, MA: MIT Press, 1965, pp. 1–18.
3. W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophyics,* **5**: 115–133, 1943.
4. W. G. Walter, *The Living Brain*, London: Duckworth, 1953.
5. I. Prigogine and I. Stengers, *Order Out of Chaos: Man's New Dialogue with Nature*. London: Flamingo, 1985.
6. C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. Urbana: University of Illinois Press, 1949.
7. W. R. Ashby, *An Introduction to Cybernetics.*New York: Wiley, 1956.
8. S. Beer, *Cybernetics and Management.*London: English Universities Press, 1959.
9. S. Beer, Towards the cybernetic factory, in H. vonFoerster and G. W. Zopf (eds.), *Principles of Self-Organization.*Oxford: Pergamon, 1962, pp. 25–89.
10. G. Pask, *An Approach to Cybernetics*. London: Hutchinson, 1961.
11. G. Pask, *Conversation Theory: Applications in Education and Epistemology*. Amsterdam: Elsevier, 1976.
12. L. von Bertalanffy, *General Systems Theory.* Harmondsworth: Penguin, 1973. (First published in the United States in 1968).
13. N. Wiener, *The Human Use of Human Beings: Cybernetics and Society*, Boston MA: Houghton Mifflin, 1954.
14. H. R. Maturana and B. Poerksen, *From Being to Doing: The Origins of the Biology of Cognition.*, Carl-Auer Heidelberg, 2002.
15. N. Luhmann, *Law as a Social System.* Oxford University Press, 2004.
16. G. Bateson, *Steps to an Ecology of Mind*, London: Paladin, 1973.
17. J. E. Lovelock, *Gaia: A New Look at Life on Earth.*Oxford University Press, 1979.
18. L. A. Zadeh, Fuzzy sets, *Information and Control*, **8**: 338–353, 1965.

FURTHER READING

C. Adami, *Introduction to Artificial Life*, New York: Springer, 1998.

A. M. Andrew, F. Conway, and J. Siegelman, Appendix to review of *Dark Hero of the Information Age: In Search of Norbert Wiener, the Father of Cybernetics*, (2005). Reviewed in: *Kybernetes*, **34**(7/8): 1284–1289, 2005.

M. A. Arbib (ed.), *The Handbook of Brain Theory and Neural Networks.* Cambridge, MA: MIT Press, 1998.

S. Beer, *Brain of the Firm*, 2nd ed. Chichester: Wiley, 1981.

F. Conway and J. Siegelman, *Dark Hero of the Information Age: In Search of Norbert Wiener, the Father of Cybernetics*, New York: Basic Books, 2005.

B. J. Copeland (ed.), *The Essential Turing: The Ideas that Gave Birth to the Computer Age*. Oxford University Press, 2004.

E. A. Feigenbaum and J. Feldman, (eds.), *Computers and Thought.* New York: McGraw-Hill, 1963.

F. Geyer and J. van derZouwen, Cybernetics and social science: theories and research in sociocybernetics, *Kybernetes*, **20**(6), 81–92, 1991.

O. Holland and P. Husbands, The origins of british cybernetics: The ratio club, *Kybernetes*, 2008. In Press.

J. Lovelock, *The Revenge of Gaia: Why the Earth is Fighting Back—and How We Can Still Save Humanity.* London: Allen Lane, 2006.

P. R. Masani, *Norbert Wiener, 1894–1964*, Birkhäuser, Basel, 1990.

H.-O. Peitgen, H. Jürgens and D. Saupe, *Chaos and Fractals: New Frontiers of Science*, New York: Springer, 1992.

J. Preston and J. M. Bishop, (eds.), *Views into the Chinese Room: New Essays on Searle and Artificial Intelligence*. Oxford University Press, 2002.

H. Rheingold, *Virtual Reality*, London: Seeker and Warburg, 1991.

D. E. Rumelhart, J. L. McClelland and the PDP [Parallel Distributed Processing] Research Group (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 2 Vols. Cambridge, MA: MIT Press, 1986.

R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: MIT Press, 1998.

H. Von Foerster and B. Poerksen, *Understanding Systems: Conversations on Epistemology and Ethics*. New York: Kluwer/Plenum, 2002.

A. J. Watson and J. E. Lovelock, Biological homeostasis of the global environment: The parable of Daisyworld, *Tellus***35B**: 284–289, 1983.

N. Wiener, *I am a Mathematician*. London: Gollancz, 1956.

P. M. Woodward, *Probability and Information Theory with Applications to Radar*, 2nd ed. Oxford: Pergamon, 1964.

M. Zeleny (ed.), *Autopoiesis: A Theory of Living Organization*, New York: North Holland, 1981.

ALEX M. ANDREW
University of Reading
Berkshire, United Kingdom

EDUCATION AND TRAINING IN SOFTWARE ENGINEERING

INTRODUCTION

Although the term “software engineering” has been used widely since the late 1960s, the question of whether to treat it as a distinct academic discipline has only been addressed during the past 15 years. The fact that a steadily increasing number of software engineering degree programs in colleges and universities exists throughout the world indicates a greater (although still far from unanimous) acceptance of software engineering as a separate discipline of study.

These software engineering degree programs emerged first at the master’s level, and then more recently at the bachelor’s and doctoral levels, in a process paralleling the development of computer science programs in the 1960s and 1970s. In both cases, the process began with the introduction of specialized elective courses in an existing curriculum. With computer science, as the body of knowledge grew, more courses were introduced, relationships among topics were better understood, textbooks were written, and better teaching methods were developed. Eventually, the fundamentals of computer science were codified in an undergraduate curriculum that provided the necessary skills to meet the increasing demand for computing practitioners, whereas the growth of computer science research and the demand for new faculty in the discipline led to doctoral programs in the field. Currently, software engineering is following this same pattern.

In addition, the evolution of software engineering has meant that industry and government both need to retrain workers from other fields as software engineer and to provide additional software engineering skills to current computing practitioners. Therefore, a variety of software engineering training courses and techniques has also been developed over the years.

This article contains:

- A history of software engineering in academia,
- The role of accreditation in various countries,
- Various curriculum models that have been proposed,
- The gap between academic education and professional knowledge required,
- An overview of software engineering training issues and academic certificate programs,
- University/industry collaborations in software engineering education,
- Distance learning and web-based education in software engineering,
- The role of professional issues in software engineering education, and
- Information on software engineering education conferences and publications.

HISTORY

In the spring of 1968, Douglas T. Ross taught a “special topics” graduate course on the topic of software engineering at the Massachusetts Institute of Technology (1). Ross claims that it is the first academic course with a title that used the term “software engineering”, James Tomayko, in his excellent article on the history of software engineering education (2), agrees that no apparent evidence exists to the contrary. This was several months before the now-famous first Conference on Software Engineering sponsored the North Atlantic Treaty Organization (NATO)!

The idea fostered by that first NATO conference of applying engineering concepts such as design, reliability, performance, and maintenance to software was revolutionary. As early as 1969, the software community began to recognize that the graduates of the fledgling computer science programs in universities were not prepared for industrial software development. In response, the industry started to ask for software engineering education rather than computer science education, including separate degree programs (3). Sketches of model curricula for software engineering programs also began to appear in the literature (4,5).

Early Software Engineering Courses

The most common form of software engineering education, even today, is a one-semester survey course, usually with a toy project built by groups of three students. In the extended decade between the first appearance of these courses and the establishment for the first software engineering degree programs, they became the mechanism for academia to experiment with teaching software engineering concepts.

Throughout the 1970s, industry continued to struggle to build larger and more complex software systems, and educators continued to create and teach the new discipline of computer science. Recognizing the problem of the divergence of these two communities, in 1976 Peter Freeman of the University of California, Irvine, and Anthony I. Wasserman of the University of California, San Francisco, organized the first U.S. workshop on software engineering education. The 40 participants represented academia, industry, and government, including companies involved in building large software systems. The workshop focused on the kind of work a professional software engineer actually did and on what the academic preparation for such a profession should be. The proceedings of the workshop (6) were published and distributed widely, and it still influences software engineering education today.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

ACM Curriculum 68 and 78 Recommendations

ACM Curriculum 68, the first major undergraduate computer science curriculum modeling effort (7), said little about learning how to develop complex software, except through work opportunities (e.g., through summer employment) or special individual project courses. (The latter became the basis for the "senior project" courses that are commonplace today of the undergraduate level, and the "software studio" courses in some graduate curricula.)

Later, ACM Curriculum 78 (8) listed a course in "software design and development"; however, it was not required in this undergraduate model. Therefore, although some growth occurred in software engineering undergraduate education, it remained limited to one or two semesters for the next 10 years. Meanwhile, software engineering as a graduate discipline would grow at a faster rate.

First Master's Degree Programs

Of course, one or two courses do not make a full curriculum. So, by the late 1970s, a few schools had started Master's degree programs in software engineering, primarily because of the pressure from local industry. To date, most graduate software engineering degree programs have used the model of a professional degree such as an MBA. (The degree title "Master of Software Engineering," or MSE, will be used here, although variations of this name are used.) A professional master's degree is a terminal degree, with graduates going into industry rather than academia. In particular, a software engineering master's degree has been a professional degree where the students in the program are already programmers or software developers in the workplace, with either a bachelor's degree in another computing discipline, or having completed sufficient undergraduate leveling to do the graduate coursework.

The first three U.S. MSE-type programs were developed at Seattle University, Texas Christian University, and the now-defunct Wang Institute of Graduate Studies in the late 1970s. Texas Christian University, located in Fort Worth, established a graduate degree program in software engineering in 1978 (9). The original curriculum was influenced greatly by the 1976 workshop. Because of external pressure prompted by the absence of an engineering college at the university, the program name was changed in 1980 to *Master of Software Design and Development*, and Texas Christian later discontinued the program.

In 1977, Seattle University initiated a series of discussions with representatives from local business and industry, during which software engineering emerged as a critical area of need for specialized educational programs. Leading software professionals were invited to assist in the development of an MSE program, which was initiated in 1979.

The Wang Institute of Graduate Studies was founded in 1979 by An Wang, founder and chairman of Wang Laboratories. It offered an MSE degree beginning in 1978. The Institute continued to depend heavily on Wang for financial support, because of the relatively low tuition income from the small student body (typically 20–30 students per year). Business declines at Wang Laboratories in the early 1980s reduced the ability to continue that support, and the institute closed in the summer of 1987. Its facilities in Tyngsboro, Massachusetts, were donated to Boston University, and its last few students were permitted to complete their degrees at that school. During its existence, the Wang program was considered to be the premier program of its kind.

According to Tomayko (2), by the mid-1980s, these three programs had similar curricula. The core courses at these institutions focused on various stages of the life cycle such as analysis, design, implementation, and testing, whereas each of the programs had a capstone project course lasting one or more semesters. These curricula were to have a large impact not only for future graduate programs, but for undergraduate curricula as well.

Academic Certificate Programs

Many higher education institutions around the world offer graduate-level academic certificates in software engineering. Although such certificate programs are targeted primarily toward those that received their undergraduate degrees in a non computing field, they differ from training programs in that they typically consist of several software engineering courses offered for graduate course credit at that institution. Typically, such graduate certificate programs typically the completion of a series of semester-long courses, constituting some portion of what might be required for a Bachelor's degree or Master's degree program. These academic certificate programs should not be confused with professional certification in software engineering, which will be addressed later in this article.

Development of Undergraduate Degree Programs

Undergraduate degree programs in software engineering have been slow to develop in most countries. One exception is the United Kingdom, where the British Computer Society (BCS) and the Institution of Electrical Engineers (IEE) have worked together to promote software engineering as a discipline. One of the oldest undergraduate software engineering degree programs in the world is at the University of Sheffield (10), which started in 1988.

Several Australian universities have also created bachelor's degree programs since the 1990 creation of an undergraduate program at the University of Melbourne.

The first undergraduate software engineering program in the United States started at the Rochester Institute of Technology in the fall of 1996, and now over 30 such programs exist (11).

Several software engineering undergraduate programs have been implemented in Canada. The program at McMaster University is probably the closest to a traditional engineering program of all those in software engineering, including requirements for courses in materials, thermodynamics, dynamics, and engineering economics, using a model outlined by its chair, David Lorge Parnas (12).

SOFTWARE ENGINEERING ACCREDITATION

Usually, accreditation of educational degree programs in a particular country is performed either by organizations in conjunction with professional societies or directly by the societies themselves. The mechanisms for software engineering accreditation in several different countries are provided below. More details are available in Ref. 13.

United States

Accreditation of engineering programs in the United States is conducted by the Engineering Accreditation Commission (EAC) of the Accreditation Board of Engineering and Technology (ABET) and until recently, accreditation of computer science programs had been conducted by a commission of the Computer Science Accreditation Board (CSAB).

In the late 1990s, the Institute of Electrical and Electronics Engineers (IEEE) developed criteria for the accreditation of software engineering programs by ABET/EAC (14). ABET and CSAB subsequently merged, with CSAB reinventing itself as a "participating society" of ABET. (As part of this change, ABET is not considered an acronym, and the organization bills itself as the accreditation body for applied science, computing, engineering, and technology education.) CSAB has lead responsibility for the accreditation of software engineering programs by ABET, which accredited its first programs during the 2002–2003 academic year, and as of October 2006 it has 13 schools accredited.

Canada

Canada had a legal dispute over the use of the term "engineering" by software engineers and in universities (15). The Association of Professional Engineers and Geoscientists of Newfoundland (APEGN) and the Canadian Council of Professional Engineers (CCPE) filed a Statement of Claim alleging trademark violation by Memorial University of Newfoundland (MUN) for using the name "software engineering" for a baccalaureate program. The APEGN and the CCPE claimed the program was not accreditable as an engineering program.

Subsequently, the parties came to an agreement: MUN dropped the use of the title "software engineering" for its program, APEGN and CCPE discontinued their lawsuit (with a five-year moratorium placed on new legal action), and the three organizations agreed to work together to define the appropriate uses of the term "software engineering" within Canadian universities (16). As a result of this action, the Canadian Engineering Accreditation Board (CEAB) began to develop criteria for accreditation of software engineering undergraduate degree programs. CEAB first accreditations were of three software engineering programs during the 2000–2001 academic year; nine programs are accredited as of 2007.

In part because of the five-year moratorium, the Computer Science Accreditation Council (CSAC), an accrediting body for computing programs in Canada sponsored by the Canadian Information Processing Society (CIPS), also started accrediting software engineering programs in 2000–2001, and also now has nine accredited schools (some schools are accredited by both CSAC and CEAB).

McCalla (17) claims that

> The difference between the CEAB and CSAC programs is substantial. The CEAB programs are offered as part of a standard engineering degree, with students highly constrained in their options and with the focus heavily on specific software engineering topics. The CSAC programs are offered as variations of standard computer science programs, with more flexibility for student choice and with the focus on a wider range of applied computer science topics than just software engineering.

Despite the end of the five-year moratorium in July 2005, both CEAB and CSAC continue to accredit software engineering degree programs.

United Kingdom

The BCS and the Institution of Engineering and Technology (IET) have accredited software engineering programs in the United Kingdom for several years. As of 2007, 64 schools with degree courses (programs) that have the words "software engineering" in their titles are accredited by BCS, according to their website at www.bcs.org.

Japan

In 2000, the Japan Board for Engineering Education (JABEE) applied a trial software engineering accreditation program (18). The Osaka Institute of Technology (OIT) was chosen for this trial, and was visited by a JABEE examination team in December 2000. The criteria used for examining the OIT program included the J97 curriculum model for Japanese computer science and software engineering programs and the IEEE-ACM Education Task Force recommended accreditation guidelines (both are discussed in the "Curriculum Models" section below).

Australia

The Institution of Engineers, Australia (IEAust) has been accrediting software engineering programs since 1993. A discussion of the IEAust accreditation process, and how the University of Melbourne developed an undergraduate software engineering degree program that was accredited in 1996, in described in Ref. 19.

CURRICULUM MODELS

Curriculum Development Issues

Below are some primary issues frequently addressed when developing or evaluating a software engineering curriculum model.

- Software engineering content
- Computer science content
- The role of calculus, laboratory sciences, and engineering sciences
- Application domain-specific topics
- Capstone experience
- Flexibility

The curriculum models below include some of the earliest, the most recent, and the most widely distributed in the field of software engineering.

Software Engineering Institute MSE Model

The mission of the Software Engineering Institute (SEI) at Carnegie Mellon University, in Pittsburgh, Pennsylvania is to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. Recognizing the importance of education in the preparation of software professionals, the institute's charter required it to "influence software engineering curricula throughout the education community." Thus, the SEI Education Program began in 1985, only one year after the Institute's founding. This program emerged at the right time to play a key role in the development of software engineering education in the United States. The program was organized with a permanent staff of educators along with a rotating set of visiting professors.

Under the direction of Norman Gibbs (1985–1990) and Nancy Mead (1991–1994), the SEI Education Program accomplished a wide variety of tasks, including developing a detailed Graduate Curriculum Model, several curriculum modules on various topics, an outline of a undergraduate curriculum model; compiling a list of U.S. graduate software engineering degree programs; creating a directory of software engineering courses offered in U.S. institutions; developing educational videotape series for both academia and industry; and creating and initial sponsoring of the Conference on Software Engineering Education. Although the Education Program was phased out at SEI in 1994, its work is still influential today.

With regard to curriculum models, the SEI concentrated initially on master's level programs for two reasons. First, it is substantially easier within a university to develop and initiate a one-year master's program than a four-year bachelor's program. Second, the primary client population at the time was software professionals, nearly all of whom already have a bachelor's degree in some discipline. In 1987, 1989, and 1991, the SEI published model curricula for university MSE programs (described below) (20–22).

Because the goal of an MSE degree program is to produce a software engineer who can assume rapidly a position of substantial responsibility within an organization, SEI proposed a curriculum designed to give the student a body of knowledge that includes balanced coverage of the software engineering process activities, their aspects, and the products produced, along with sufficient experience to bridge the gap between undergraduate programming and professional software engineering. Basically, the program was broken into four parts: undergraduate prerequisites, core curriculum, the project experience component, and electives. The minimal undergraduate prerequisites were discrete mathematics, programming, data structures, assembly language, algorithm analysis, communication skills, and some calculus. Laboratory sciences were not required. The six core curriculum courses were:

Specification of Software Systems
Software Verification and Validation
Software Generation and Maintenance
Principles and Applications of Software Design
Software Systems Engineering
Software Project Management

The topics and content of these courses were in many ways an outgrowth of the common courses identified in the first MSE programs. These courses would likely be taken in the first year of a two-year MSE sequence. The bulk of the report consists of detailed syllabi and lecture suggestions for these six courses.

The project experience component took the form of the capstone project, and might require different prerequisites according to the project. The electives could be additional software engineering subjects, related computer science topics, system engineering courses, application domain courses, or engineering management topics.

Although the final version of the SEI model for an MSE was released almost 15 years ago, it remains today the most influential software engineering curriculum model for Master's degree programs.

BCS/IEE Undergraduate Model

As stated previously, the BCS and the IET have cooperated for many years in the development and accreditation of software engineering degree programs, dating back to the IEE, one of the two societies that merged to form the IET. Therefore, it is not surprising that the first major effort to develop a curriculum model for the discipline was a joint effort by these two societies (23).

The curriculum content defined in the report was "deliberately designed to be non-prescriptive and to encourage variety" (p. 27). Three types of skills are defined: central software engineering skills, supporting fundamental skills (technical communication, discrete math, and various computer science areas), and advanced skills.

At the end of the section on curriculum content, the authors state "the topics alone do not define a curriculum, since it is also necessary to define the depth to which they may be taught. The same topic may be taught in different disciplines with a different target result..." (p. 31). This reinforces the comments concerning the variety and non prescriptive nature of the recommendations at the beginning of that section.

It is interesting to note that the curriculum content does *not* include a need for skills in areas traditionally taught to engineers (e.g., calculus, differential equations, chemistry, physics, and most engineering sciences). It is remarkable that the computer scientists and electrical engineers on the working party that produced this report were able to agree on a curriculum that focused primarily on engineering process and computer science skills, and did not tie itself to traditional engineering courses.

SEI Undergraduate Model

In making undergraduate curriculum recommendations, Gary Ford (24) of SEI wanted a curriculum that would be compatible with the general requirements for ABET and CSAB and the core computing curriculum in the IEEE-CS/ ACM Curricula 91 (which was then in draft form). Also, although he was complimentary of the BCS/IEE recommendations, Ford was more precise in defining his model curriculum. The breakdown (for a standard 120 semester hour curriculum) was as follows:

Mathematics and Basic Sciences	27 semester hours
Software Engineering Sciences and Design	45 semester hours
Humanities and Social Sciences	30 semester hours
Electives	18 semester hours

The humanities & social sciences and electives were included to allow for maximum flexibility in implementing the curriculum. Mathematics and basic sciences consisted of two semesters of both discrete mathematics and calculus, and one semester of probability and statistics, numerical methods, physics, chemistry, and biology.

In designing the software engineering science and design component, Ford argues that the engineering sciences for software are primarily computer science, rather than sciences such as statics and thermodynamics (although for particular application domains, such knowledge may be useful). Four different software engineering areas are defined: software analysis, software architectures, computer systems, and software process.

Ford goes on to define 14 courses (totaling 42 semester hours; one elective is allowed), with 3 or 4 courses in each of these four areas, and places them (as well as the other aspects of the curriculum) in the context of a standard four-year curriculum. The four software process courses were placed in the last two years of the program, which allowed for the possibility of a senior project-type experience.

Thus, several similarities existed between the BCS/IEE and SEI recommended undergraduate models: they focused on similar software engineering skills, did not require standard engineering sciences, and (interestingly) did not require a capstone experience.

IEEE-ACM Education Task Force

The Education Task Force of an IEEE-ACM Joint Steering Committee for the Establishment of Software Engineering developed some recommended accreditation criteria for undergraduate programs in software engineering (25). Although no accreditation board has yet adopted these precise guidelines, it has influenced several accreditation and curriculum initiatives.

According to their accreditation guidelines, four areas exist (software engineering, computer science, and engineering, supporting areas suchc technical communication and mathematics, and advanced work in one or more area) that are each about three-sixteenths of the curriculum, which amounts to 21–24 semester hours for each area in a 120-hour degree plan. (The remaining hours were left open, to be used by, for instance, general education requirements in the United States). As in the SEI graduate guidelines, a capstone project is addressed explicitly.

Guidelines for Software Engineering Education

This project was created by a team within the Working Group for Software Engineering Education and Training (WGSEET). The Working Group was a “think tank” of about 30 members of the software engineering and training communities, who come together twice a year to work on major projects related to the discipline. WGSEET was established in 1995 in part to fill a void left by the demise of the SEI Education Program.

At the November 1997 Working Group meeting, work began on what was originally called the “Guidelines for Software Education.” A major difference in goals of the *Guidelines* with the previously discussed projects is that the latter developed computer science, computer engineering, and information systems curricula with software engineering concentrations in addition to making recommendations for an undergraduate software engineering curriculum. (This article will focus only on the software engineering model.)

Here are the recommended number of hours and courses for each topic area in the software engineering model, from Ref. 26.

Software Engineering – Required (24 of 120 Semester Hours)

Software Engineering Seminar (One hour course for first-semester students)
Introduction to Software Engineering
Formal Methods
Software Quality
Software Analysis and Design I and II
Professional Ethics
Senior Design Project (capstone experience)

Computer Science – Required (21 Semester Hours)

Introduction to Computer Science for Software Engineers 1 and 2
(Similar to first year computer science, but oriented to software engineering)
Data Structures and Algorithms
Computer Organization
Programming Languages
Software Systems 1 and 2
(Covers operating systems, databases, and other application areas)

Computer Science / Software Engineering – Electives (9 Semester Hours)

Various topics

Lab Sciences (12 Semester Hours)

Chemistry 1
Physics 1 and 2

Engineering Sciences (9 Semester Hours)

Engineering Economics
Engineering Science 1 and 2
(Provides an overview of the major engineering sciences)

Mathematics (24 Semester Hours)

Discrete Mathematics
Calculus 1, 2 and 3
Probability and Statistics
Linear Algebra
Differential Equations

Communication/Humanities/Social Sciences (18 Semester Hours)

Communications 1 and 2 (first year writing courses)
Technical Writing
Humanities/Social Sciences electives

Open Electives (3 Semester Hours)

Any course

This model is intended to be consistent with the IEEE-ACM Education Task Force recommendations and criteria for all engineering programs accredited by ABET. The nine hours of engineering sciences is less than for traditional engineering disciplines and that contained in the McMaster University degree program, but more than that is required in the other software engineering curriculum models.

Not much flexibility exists in this model, because only one technical elective outside of Computer Science/Software Engineering can be taken (using the open elective). However, note that the model is for a minimal number of semester hours (120), so additional hours could provide that flexibility.

Information Processing Society of Japan (IPSJ)

IPSJ has developed a core curriculum model for computer science and software engineering degree programs commonly called J97 (18). Unlike the other curriculum models discussed above, J97 defines a common core curriculum for every undergraduate computer science and software engineering program. This core curriculum includes the following learning units:

Computer Literacy Courses

Computer Science Fundamentals
Programming Fundamentals

Mathematics Courses

Discrete Mathematics
Computing Algorithms
Probability and Information Theory
Basic Logic

Computing Courses

Digital Logic
Formal Languages and Automata Theory
Data Structures
Computer Architecture
Programming Languages
Operating Systems
Compilers
Databases
Software Engineering
The Human-Computer Interface

Note that, unlike most computing curriculum models, basic programming skills are considered part of basic computer literacy, whereas algorithm analysis (computing algorithms) is treated as part of the mathematics component. The units listed above provide for a broad background in computer science as well as an education in basic computer engineering and software engineering fundamentals.

Computing Curricula – Software Engineering

Over the years, the ACM/IEEE-CS joint Education Task Force went through several changes, eventually aligning itself with similar projects being jointly pursued by the two societies. The Software Engineering 2004 (SE 2004) project (as it was eventually named) was created to provide detailed undergraduate software engineering curriculum guidelines which could serve as a model for higher education institutions across the world. The result of this project was an extensive and comprehensive document which has indeed become the leading model for software engineering undergraduate curricula (27). SE 2004 was defined using the following principles:

1. Computing is a broad field that extends well beyond the boundaries of any one computing discipline.
2. Software Engineering draws its foundations from a wide variety of disciplines.
3. The rapid evolution and the professional nature of software engineering require an ongoing review of the corresponding curriculum. The professional associations in this discipline.
4. Development of a software engineering curriculum must be sensitive to changes in technologies, practices, and applications, new developments in pedagogy, and the importance of lifelong learning. In a field that evolves as rapidly as software engineering, educational institutions must adopt explicit strategies for responding to change.
5. SE 2004 must go beyond knowledge elements to offer significant guidance in terms of individual curriculum components.

6. SE 2004 must support the identification of the fundamental skills and knowledge that all software engineering graduates must possess.
7. Guidance on software engineering curricula must be based on an appropriate definition of software engineering knowledge.
8. SE 2004 must strive to be international in scope.
9. The development of SE2004 must be broadly based.
10. SE 2004 must include exposure to aspects of professional practice as an integral component of the undergraduate curriculum.
11. SE 2004 must include discussions of strategies and tactics for implementation, along with high-level recommendations.

The SE 2004 document also defines student outcomes; that is, what is expected of graduates of a software engineering program using the curriculum guidelines contained within:

1. Show mastery of the software engineering knowledge and skills, and professional issues necessary to begin practice as a software engineer.
2. Work as an individual and as part of a team to develop and deliver quality software artifacts.
3. Reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems, and organizations.
4. Design appropriate solutions in one or more application domains using software engineering approaches that integrate ethical, social, legal, and economic concerns.
5. Demonstrate an understanding of and apply current theories, models, and techniques that provide a basis for problem identification and analysis, software design, development, implementation, verification, and documentation.
6. Demonstrate an understanding and appreciation for the importance of negotiation, effective work habits, leadership, and good communication with stakeholders in a typical software development environment.
7. Learn new models, techniques, and technologies as they emerge and appreciate the necessity of such continuing professional development.

The next step was to define Software Engineering Education Knowledge (SEEK), a collection of topics considered important in the education of software engineering students. SEEK was created and reviewed by volunteers in the software engineering education community. The SEEK body is a three-level hierarchy, initially divided into knowledge areas (KAs) as follows:

Computing essentials
Mathematical and engineering fundamentals
Professional practice
Software modeling and analysis
Software design
Software verification & validation
Software evolution
Software process
Software quality
Software management

Those KAs are then divided even more into units, and finally, those units are divided into *topics*. For example, DES.con.7 is a specific topic (Design Tradeoffs) in the Design Concepts unit of the Software Design knowledge area. Each topic in *SEEK* is also categorized for its importance: Essential, Desired, or Optional.

The SE 2004 document also contains recommendations for possible curricula and courses, tailored towards models for specific countries and specific kinds of institutions of higher-level education. Software engineering topics consisted of at least 20% of the overall curriculum in each case. Most of the models recommended for use in the United States included some type of introduction to software engineering during the first two years of study, followed by six SE courses, with a capstone project occurring throughout the fourth and final (senior) year of study.

THE GAP BETWEEN EDUCATION AND INDUSTRY

Tim Lethbridge of the University of Ottawa has done a considerable amount of work in attempting to determine what industry thinks is important knowledge for a software professional to receive through academic and other educational venues through a series of surveys (28). The results of these surveys show that a wide gap still exists between what is taught in education versus what is needed in industry. For instance, among the topics required of professionals that had to be learned on the job include negotiation skills, human-computer interaction methods, real-time system design methods, management and leadership skills, cost estimation methods, software metrics, software reliability and fault tolerance, ethics and professionalism practice guidelines, and requirements gathering and analysis skills. Among the topics taught in most educational programs but not considered important to industry included digital logic, analog systems, formal languages, automata theory, linear algebra, physics and chemistry. Industry and academia agreed that a few things were important, including data structures, algorithm design, and operating systems.

The survey results also demonstrate that it is essential for industry and academia to work together to create future software engineering curricula, for both groups to use their resources more wisely and effectively in the development of software professionals in the future.

Another excellent article outlining the industrial view was by Tockey (29). Besides stressing the need for software practitioners to be well-versed in computer science and discrete mathematics, he identified software engineering economics as an important “missing link” that current software professionals do not have when entering the workforce.

TRACKING SOFTWARE ENGINEERING DEGREE PROGRAMS

Over the years, several surveys have been attempted to determine what software engineering programs exist in a particular country or for a particular type of degree. By 1994, 25 graduate MSE-type programs existed in the United States, and 20 other programs with software engineering options were in their graduate catalogs, according to an SEI survey (30) that was revised slightly for final release in early 1996.

Thompson and Edwards' excellent article on software engineering education in the United Kingdom (31) provides an excellent list of 43 Bachelor's degree and 11 Master's programs in software engineering in the UK.

Doug Grant (32) of the Swinburne University of Technology reported that Bachelor's degree programs insoftware engineering were offered by at least 9 of Australia's 39 universities, with more being planned.

In June 2002, Fred Otto of CCPE provided the author with a list of 10 Canadian schools with undergraduate software engineering degree programs.

Bagert has in recent years published a list of 32 undergraduate (11) and 43 Master's level (33) SE programs in the United States, along with some information concerning those programs.

Currently, few doctoral programs in software engineering exist. In the late 1990s, the first Doctor of Philosophy (Ph.D.) programs in software engineering in the United States were approved at the Naval Postgraduate School (34) and at the Carnegie Mellon University (35). Also, in 1999, Auburn University changed its doctoral degree in Computer Science to "Computer Science and Software Engineering."

INDUSTRY/UNIVERSITY COLLABORATIONS

Collaborations between industry and higher education are common in engineering disciplines, where companies can give students experience with real-world problems through project courses. However, formal collaborations between university and industry have become more frequent in the last decade. Beckman et al. (36) discussed the benefits to industry and academia of such collaborations:

- Benefits to industry:
  - ◦Cost-effective, customized education and training
  - ◦Influence on academic programs
  - ◦Access to university software engineering research
  - ◦New revenue sources
- Benefits to academia:
  - ◦Placement of students
  - ◦Insight into corporate issues at the applied, practical level
  - ◦Research and continuing education opportunities for faculty
  - ◦Special funding from a corporate partner to the cooperating university

This paper also provided details on three successful formal university/industry partnerships in software engineering. Through evaluation of these and other evaluations, Beckman et al. (36), developed a model for a successful collaboration which included:

- Formation by the university of an advisory board made up of representatives of its industrial partners.
- A clear definition of the goals and expectations of the collaboration.
- Developing and executing a multiphase process for the collaboration project.
- Defining, measuring, and evaluating effectively metrics for project success.

WGSEET has documented several innovative formal industry-university collaborations in software engineering over the years; (37) Ref. 37 is the seventh and latest edition of their directory of such collaborations, providing details of 23 such formal partnerships, including the three described above. The initial results of an even more recent survey by this group can be found in Ref. 38.

TRAINING IN SOFTWARE ENGINEERING

To this point, this discussion has concentrated on academic education, as opposed to education within industry (training). The term *software engineering education and training* is used commonly to encompass both academic education and industrial training issues.

Starting over 20 years ago, several large companies involved in software development began to embrace the concept of software engineering. Faced with both a software development workforce mostly untrained in software engineering skills and paucity of academic coursework in software engineering available, many companies began developing an array of in-house courses to meet the need. Among the first of these companies was the IBM Software Engineering Education Program, which was started in the late 1970s. This program was influenced greatly by software engineering pioneer Harlan Mills, who worked for IBM from 1964 to 1987.

Also among the oldest software engineering training programs is that of the Texas Instruments Defense Systems and Electronics Group (DSEG), which is now part of Raytheon. Moore and Purvis (39) discussed the DSEG software training curriculum as it existed then. First, those engineers assigned to develop software would take a three-day "Software Engineering Workshop" for engineers, which would introduce the workers to DSEG software practices and standards, as well as DoD life cycle requirements. This workshop could be followed with courses such as software quality assurance, software configuration management, introduction to real-time systems, structured analysis, software design, software testing and software management.

Motorola is another example of a company that has invested considerably in the software engineering training of its employees. Sanders and Smith (40) estimated that its software engineering population at the time required 160,000 person-hours of training per year, which it provided both through its own Motorola University, as well as through collaborations with various universities (such as the one with Florida Atlantic University discussed in the previous section).

Over the years, several companies have offered a wide variety of software engineering training courses to both companies and individuals . Construx Software lists on its web page (41) a wide variety of training seminars. In addition, Construx has a significant professional development program for its own employees, employing readings, classes, discussion groups, mentoring, and so on.

Typically software engineering training courses offered by companies are of length anywhere from a half-day course associated with a meeting such as the International Conference on Software Engineering to a one or two-week stand-alone course. Generally, such courses cost $500–1000 U.S. per day for each registrant.

Software process improvement training has increased significantly over the past 10 years. For instance, several of companies offer training services to corporations that want their software divisions to obtain ISO 9000 registration or a certain level of the Capability Maturity Model Integration (registered in the U.S. Patent and Trademark office). In addition, also many training courses exist in both the Personal Software Process and the related Team Software Process (e.g., Ref. 42) (Personal Software Process and Team Software Process are service marks of Cargegie Mellon University, Pittsburgh, PA).

DISTANCE LEARNING AND WEB-BASED EDUCATION

Both software engineering academic and training courses have been available through various distance means over the years. A early example was the Software Engineering Institute's Video Dissemination Project begun the early 1990s, which provided two series of courses: one to support academic instruction and the other to continue education.

With the advent of the Internet and advanced multimedia technology, software engineering distance education has increased significantly. Synchronous and asynchronous distance education and training in software engineering allows for increased schedule flexibility for the participant, and it also helps to satisfy the demand for such courses despite the limited number of instructors available.

Carnegie Mellon offers its entire MSE degree online through the use various media. The first group of Carnegie Mellon students to graduate with an MSE without ever setting foot on campus was in August 2000 (43).

A survey of Master's level software engineering programs in the United States by Bagert and Mu (33) found that of the 43 schools that had "software engineering" as the name of the Master's degree, 24 schools deliver the programs only face-to-face, three only online, and 16 schools provide for both or have a curriculum that has both face-to-face and online courses.

THE ROLE OF PROFESSIONAL ISSUES

As software engineering has begun to evolve as a distinct discipline, various efforts have been related to professional issues in software engineering, including accreditation of degree programs, the identification of the software engineering body of knowledge, the licensing and certification of software engineers, and the development of a code of ethics and professional practices for software engineering. The software engineering education and training community, recognizing the impact of professional issues on their curricula, has begun to address such matters in education and training conferences and publications.

SOFTWARE ENGINEERING EDUCATION CONFERENCES AND PUBLICATIONS

Since 1987, the Conference on Software Engineering Education (CSEE&T) has become tremendously influential to the software engineering education and training community worldwide. Originally created and run by the SEI, the conference has in recent years been sponsored by the IEEE Computer Society. As the conference evolved, it grew to include training (hence the name change) and professional issues, and for a time was colocated with the ACM SIGCSE (Special Interest Group on Computer Science Education) Symposium on Computer Science Education, giving educators in computer science and software engineering an opportunity to meet together and discuss issues of common concern. Today, CSEE&T remains the world's premiere conference dedicated to software engineering education, training, and professional issues.

FASE (Forum for Advancing Software engineering Education) was started in 1991 by members of the software engineering education community to have an electronic forum for the dissemination and discussion of events related to software engineering education. The original acronym for *FASE* was Forum for Academic Software Engineering, but it was subsequently changed so that it was more inclusive to industrial and government training issues. In the last few years, FASE has also covered a wide variety of professional issues. During the last few years, FASE has limited its coverage mostly to announcements of upcoming events and faculty openings. An archive of all issues through February 2004 is available at http://cstl-csm.semo.edu/dbagert/fase.

Although currently no refereed journals are devoted exclusively to software engineering education, several publications have devoted special issues to the subject over the years, including *IEEE Software, IEEE Transactions on Software Engineering*, *Journal of Systems and Software*, *Information and Software Technology*, and *Computer Science Education*.

CONCLUSIONS AND FUTURE DIRECTIONS

Only a few years ago, the number of software engineering degree programs were consistently increasing, giving great hope for the future. However, the rate of increase of such programs has slowed [e.g., its status in the United States was discussed by Bagert and Chenoweth (11)]. A unique opportunity in software engineering education is before the computing and engineering disciplines, one that has the potential to open both to tremendous possibilities. However, this can be done only by a joint effort of the computing and engineering communities, just as BCS and IEE did in the United Kingdom almost 20 years ago. In other countries, the results of such attempted collaborations in other countries have been at best mixed.

Industrial education, both through training courses and from collaborations with academic institutions, will continue to expand as the demand for software engineers also continues to increase. This requires the continuing education of software professionals as well as the retraining of workers with backgrounds from related disciplines. The need for more collaboration between industry/university is especially important in the face of surveys that demonstrate a distinct gap between academic educational outcomes and the industrial knowledge required for software professionals.

It is likely that distance education will be impacting all academic and professional disciplines in the near future. In addition, distance learning will be especially important for the software engineering community as long as instructor shortages remain.

Finally, perhaps the most critical step required for the future of software engineering education and training is the need for a true internationalization of major initiatives. Many projects discussed in this article were successful efforts that were developed within a single country. The SE 2004 project was notable in part for the fact that it went to great lengths to be a document truly international in development and scope. Not enough communication of the successes of a particular country to the international community exists; for instance, the accomplishments of BCS and IEE regarding software engineering education in the United Kingdom over past dozen years is largely unknown in the United States even today. The challenge is to use the Internet, the World Wide Web, and other technological innovations (which were, after all, developed in large part by software professionals!) to advance the discipline itself even more by creating an effective and truly global software engineering education and training community.

ACKNOWLEDGMENTS

This article builds on the excellent “Education and Curricula in Software Engineering” by Gary A. Ford and James E. Tomayko, which appeared in the first edition (1994) of the *Encyclopedia of Software Engineering,* also published by John Wiley and Sons.

BIBLIOGRAPHY

1. D. T. Ross, The NATO conferences from the perspective of an active software engineer, *Annals Hist. Comput.*, **11**(2): 133–141, 1989.
2. J. E. Tomayko, Forging a discipline: an outline history of software engineering education, *Annals Soft. Engineer*. **6**: 3–18, 1998.
3. F. F. Kuo, Let's make our best people into software engineers and not computer scientists, *Computer Decisions***1**(2): 94, 1969.
4. R. E. Fairley, Toward model curricula in software engineering, *SIGCSE Bulletin*, **10**(3): 77–79, 1978.
5. R. W. Jensen and C. C. Tonies, *Software Engineering*, Englewood Cliffs, NJ: Prentice Hall, 1979.
6. A. I. Wasserman and P. Freeman, eds., *Software Engineering Education: Needs and Objectives*, New York: Springer-Verlag, 1976.
7. ACM Curriculum Committee on Computer Science, Curriculum 68: Recommendations for the undergraduate program in computer science, *Communicat. ACM*, **11**(3): 151–197, 1968.
8. ACM Curriculum Committee on Computer Science, Curriculum 78: Recommendations for the undergraduate program in computer science, *Communicat. ACM*, **22**(3): 147–166, 1979.
9. J. R. Comer and D. J. Rodjak, Adapting to changing needs: a new perspective on software engineering education at Texas Christian University, in N. E. Gibbs and R. E. Fairley, eds., *Software Engineering Education: The Educational Needs of the Software Community*, New York: Springer-Verlag, 1987, pp. 149–171.
10. A. J. Cowling, The first decade of an undergraduate degree programme in software engineering, *Annals Software Engineer.*, **6**: 61–90, 1999.
11. D. J. Bagert and S. V. Chenoweth, Future growth of software engineering baccalaureate programs in the United States, *Proc. of the ASEE Annual Conference*, Portland, Oregon, 2005, CD-ROM, 8 pp.
12. D. L. Parnas, Software engineering programmes are not computer science programmes, *Annals Soft. Enginee.***6**: 19–37, 1998.
13. D. J. Bagert and N. R. Mead, Software engineering as a professional discipline, *Comp. Sci. Educ.*, **11**(1): 2001 73–87, 2001.
14. G. Gillespie, ABET asks the IEEE to look at software engineering accreditation, *IEEE—The Institute* **21** (7): 1, 1997.
15. D. K. Peters, Update on lawsuit about use of the term 'Software Engineering', *Forum for Advancing Software engineering Education (FASE)*, **9**(3): 1999.
16. Association of Universities and Colleges of Canada and Council of Professional Engineers, Software Engineering Lawsuit Discontinued, communiqué reprinted under the heading “Canadian Lawsuit Discontinued” in *Forum for Advancing Software engineering Education (FASE)*, **9**(10): 1999.
17. G. McCalla, Canadian news and views, *Comput. Res. News*, **16**(4): 2004.
18. Y. Matsumoto, Y. Akiyama, O. Dairiki, and T. Tamai, A case of software engineering accreditation, *Proc. of the 14th Conference on Software Engineering Education and Training*, Charlotte, NC, 2001, pp. 201–209.
19. P. Dart, L. Johnston, C. Schmidt, and L. Sonenberg, Developing an accredited software engineering program, *IEEE Software*, **14**(6): 66–71, 1997.
20. G. A. Ford, N. E. Gibbs, and J. E. Tomayko, *Software Engineering Education: An Interim Report from the Software Engineering Institute*, CMU/SEI-TR-87-109, Pittsburgh, PA: Carnegie Mellon University, 1987.
21. M. A. Ardis and G. A. Ford, *1989 SEI Report on Graduate Software Engineering Education*, CMU/SEI-89-TR-2, Pittsburgh, PA: Carnegie Mellon University, 1989.

22. G. A. Ford, *1991 SEI Report on Graduate Software Engineering Education*, CMU/SEI-91-TR-2, Pittsburgh, PA: Carnegie Mellon University, 1991.

23. British Computer Society and The Institution of Electrical Engineers, *A Report on Undergraduate Curricula for Software Engineering*, Institution of Electrical Engineers, 1989.

24. G. A. Ford, 1990 SEl Report on Undergraduate Software Engineering Education, CMU/SEI-90-TR-3, Pittsburgh, PA: Carnegie Mellon University, 1990.

25. G. L. Engel, Program criteria For software engineering accreditation programs, *IEEE Software*, **16**(6): 31–34, 1999.

26. D. J. Bagert, T. B. Hilburn, G. Hislop, M. Lutz, M. McCracken, and S. Mengel, *Guidelines for Software Engineering Education Version 1.0. CMU/SEI-99-TR-032, Pittsburgh, PA: Carnegie Mellon University, 1999.*

27. Joint Task Force on Computing Curricula, *Software Engineering* 2004, Piscataway, NJ: IEEE Computer Society and the Association for Computing Machinery.

28. T. Lethbridge, What knowledge is important to a software professional?, *IEEE Computer*, **33**(5): 44–50, 2000.

29. S. Tockey, A missing link in software engineering, *IEEE Software*, **14**(6): 31–36, 1997.

30. G. A. Ford, *A Progress Report on Undergraduate Software Engineering Education, CMU/SEI-94-TR-11, Pittsburgh, PA: Carnegie Mellon University, 1994.*

31. J. B. Thompson and H. M. Edwards, Software engineering in the UK 2001, *Forum for Advancing Software engineering Education (FASE)*, **11**(11): 2001.

32. D. Grant, Undergraduate software engineering degrees in Australia, *Proc. of the 13th Conference on Software Engineering Education and Training*, Austin, Te, 2000, pp. 308–309.

33. D. J. Bagert and X. Mu, Current state of software engineering Master's X. degree programs in the United States, *Proc. of the Frontiers in Education Conference*, Indianapolis, Indiana, 2005, F1G1–F1G6.

34. Luqi, Naval Postgraduate School Offers First USA PhD in Software Engineering, *Forum for Advancing Software engineering Education (FASE)*, **9**(7): 1999.

35. D. Garlan, P. Koopman, W. Scherlis, and M. Shaw, PhD in Software Engineering: A New Degree Program at Carnegie Mellon University, *Forum for Advancing Software engineering Education (FASE)*, **10**(2): 2000.

36. K. Beckman, N. Coulter, S. Khajenoori and N. Mead, Industry/university collaborations: closing the gap between industry and academia, *IEEE Software*, **14**(6): 49–57, 1997.

37. K. Beckman, *Directory of Industry and University Collaborations with a Focus on Software Engineering Education and Training, Version 7*, CMU/SEI-99-SR-001, Pittsburgh, PA: Carnegie Mellon University, 1999.

38. S. Ellis, N. R. Mead and D. Ramsey, Summary of the initial results of the university/industry survey performed by the university/industry subgroup of the working group on software engineering education and training, *Forum for Advancing Software Engineering Education (FASE)*, **11**(1): 2001.

39. F. L. Moore and P. R. Purvis, Meeting the training needs of practicing software engineers at Texas Instruments, *Proc. of the Second Conference on Software Engineering Education*, Fairfax, VA, 1988, pp. 32–44.

40. G. Sanders and G. Smith, Establishing Motorola-university relationships: a software engineering training perspective, *Proc. of the Fourth Conference on Software Engineering Education*, Pittsburgh, PA, 1990, pp. 2–12.

41. Construx Software Construx Public Seminars. [Online]. Construx Software, Bellvue, Washington. 2001. Available: http://www.construx.com/Page.aspx?nid=12.

42. T. Hilburn, Personal Software Process^{SM} and Team Software Process^{SM} 2001 summer faculty workshops, *Forum for Advancing Software Engineering Education (FASE)*, **11**(3): 2001.

43. J. E. Tomayko, Master of Software Engineering, Carnegie Mellon University, *Forum for Advancing Software engineering Education (FASE)*, **10**: 2000.

DONALD J. BAGERT
Southeast Missouri State University
Cape Girardeau, Missouri

ETHICS AND PROFESSIONAL RESPONSIBILITY IN COMPUTING

INTRODUCTION

Computing professionals perform a variety of tasks: They write specifications for new computer systems, they design instruction pipelines for superscalar processors, they diagnose timing anomalies in embedded systems, they test and validate software systems, they restructure the back-end databases of inventory systems, they analyze packet traffic in local area networks, and they recommend security policies for medical information systems. Computing professionals are obligated to perform these tasks conscientiously because their decisions affect the performance and functionality of computer systems, which in turn affect the welfare of the systems' users directly and that of other people less directly. For example, the software that controls the automatic transmission of an automobile should minimize gasoline consumption and, more important, ensure the safety of the driver, any passengers, other drivers, and pedestrians.

The obligations of computing professionals are similar to the obligations of other technical professionals, such as civil engineers. Taken together, these professional obligations are called *professional ethics*. Ethical obligations have been studied by philosophers and have been articulated by religious leaders for many years. Within the discipline of philosophy, ethics encompasses the study of the actions that a responsible individual should choose, the values that an honorable individual should espouse, and the character that a virtuous individual should have. For example, everyone should be honest, fair, kind, civil, respectful, and trustworthy. Besides these general obligations that everyone shares, professionals have additional obligations that originate from the responsibilities of their professional work and their relationships with clients, employers, other professionals, and the public.

The ethical obligations of computing professionals go beyond complying with laws or regulations; laws often lag behind advances in technology. For example, before the passage of the Electronic Communications Privacy Act of 1986 in the United States, government officials did not require a search warrant to collect personal information transmitted over computer communication networks. Nevertheless, even in the absence of a privacy law before 1986, computing professionals should have been aware of the obligation to protect the privacy of personal information.

WHAT IS A PROFESSION?

Computing professionals include hardware designers, software engineers, database administrators, system analysts, and computer scientists. In what ways do these occupations resemble recognized professions such as medicine, law, engineering, counseling, and accounting? In what ways do computing professions resemble occupations that are not thought of traditionally as professions, such as plumbers, fashion models, and sales clerks?

Professions that exhibit certain characteristics are called *strongty differentiated* professions (1). These professions include physicians and lawyers, who have special rights and responsibilities. The defining characteristics of a strongly differentiated profession are specialized knowledge and skills, systematic research, professional autonomy, a robust professional association, and a well-defined social good associated with the profession.

Members of a strongly differentiated profession have specialized knowledge and skills, often called a "body of knowledge," gained through formal education and practical experience. Although plumbers also have special knowledge and skills, education in the trades such as plumbing emphasizes apprenticeship training rather than formal education. An educational program in a professional school teaches students the theoretical basis of a profession, which is difficult to learn without formal education. A professional school also socializes students to the values and practices of the profession. Engineering schools teach students to value efficiency and to reject shoddy work. Medical schools teach students to become physicians, and law schools teach future attorneys. Because professional work has a significant intellectual component, entry into a profession often requires a post-baccalaureate degree such as the M.S.W. (Master of Social Work) or the Psy.D. (Doctor of Psychology).

Professionals value the expansion of knowledge through systematic research; they do not rely exclusively on the transmission of craft traditions from one generation to the next. Research in a profession is conducted by academic members of the professioin and sometimes by practitioner members too- Academic physicians, for example, conduct medical research. Because professionals understand that professional knowledge always advances, professionals should also engage in continuing education by reading publications and attending conferences. Professionals should share general knowledge of their fields, rather than keeping secrets of a guild. Professionals are obligated, however, to keep specific information about clients confidential.

Professionals tend to have *clients*, not *customers*. Whereas a sales clerk should try to satisfy the customer's *desires*, the professional should try to meet the client's *needs* (consistent with the welfare of the client and the public). For example, a physician should not give a patient a prescription for barbiturates just because the patient wants the drugs but only if the patient's medical condition warrants the prescription.

Because professionals have specialized knowledge, clients cannot fully evaluate the quality of services provided by professionals. Only other members of a profession, the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

professional's peers, can sufficiently determine the quality of professional work. The principle of peer review underlies accreditation and licensing activities: Members of a profession evaluate the quality of an educational program for accreditation, and they set the requirements for the licensing of individuals. For example, in the United States, a lawyer must pass a state's bar examination to be licensed to practice in that state. (Most states have reciprocity arrangements—a professional license granted by one state is recognized by other states.) The license gives professionals legal authority and privileges that are not available to unlicensed individuals. For example, a licensed physician may legitimately prescribe medications and perform surgery, which are activities that should not be performed by people who are not medical professionals.

Through accreditation and licensing, the public cedes control over a profession to members of the profession. In return for this autonomy, the profession promises to serve the public good. Medicine is devoted to advancing human health, law to the pursuit of justice, and engineering to the economical construction of safe and useful objects. As an example of promoting the public good over the pursuit of self-interest, professionals are expected to provide services to some indigent clients without charge. For instance, physicians volunteer at free clinics, and they serve in humanitarian missions to developing countries. Physicians and nurses are expected to render assistance in cases of medical emergency—for instance, when a train passenger suffers a heart attack. In sum, medical professionals have special obligations that those who are not medical professionals do not have.

The purposes and values of a profession, including its commitment to a public good, are expressed by its code of ethics. A fortiori, the creation of a code of ethics is one mark of the transformation of an occupation into a profession.

A profession's code of ethics is developed and updated by a national or international professional association. This association publishes periodicals and hosts conferences to enable professionals to continue their learning and to network with other members of the profession. The association typically organizes the accreditation of educational programs and the licensing of individual professionals.

Do computing professions measure up to these criteria for a strongly differentiated profession? To become a computing professional, an individual must acquire specialized knowledge about discrete algorithms and relational database theory and specialized skills such as software development techniques and digital system design. Computing professionals usually learn this knowledge and acquire these skills by earning a baccalaureate degree in computer science, computer engineering, information systems, or a related field. As in engineering, a bachelor's degree currently suffices for entry into the computing professions. The knowledge base for computing expands through research in computer science conducted in universities and in industrial and government laboratories.

Like electrical engineers, most computing professionals work for employers, who might not be the professionals' clients. For example, a software engineer might develop application software that controls a kitchen appliance; the engineer's employer might be different from the appliance manufacturer. Furthermore, the software engineer should prevent harm to the ultimate users of the appliance and to others who might be affected by the appliance. Thus, the computing professional's relationship with a client and with the public might be indirect.

The obligations of computing professionals to clients, employers, and the public are expressed in several codes of ethics. The later section on codes of ethics reviews two codes that apply to computing professionals.

Although the computing professions meet many criteria of other professions, they are deficient in significant ways. Unlike academic programs in engineering, relatively few academic programs in computing are accredited. Furthermore, in the United States, computing professionals cannot be licensed, except that software engineers can be licensed in Texas. As of this writing, the Association for Computing Machinery (ACM) has reaffirmed its opposition to state-sponsored licensing of individuals (2). Computing professionals may earn proprietary certifications offered by corporations such as Cisco, Novell, Sun, and Microsoft. In the United States, the American Medical Association dominates the medical profession, and the American Bar Association dominates the legal profession, but no single organization defines the computing profession. Instead, multiple distinct organizations exist, including the ACM, the Institute of Electrical and Electronics Engineers Computer Society (IEEE-CS), and the Association of Information Technology Professionals (AIPT). Although these organizations cooperate on some projects, they remain largely distinct, with separate publications and codes of ethics.

Regardless of whether computing professions are strongly differentiated, computing professionals have important ethical obligations, as explained in the remainder of this article.

WHAT IS MORAL RESPONSIBILITY IN COMPUTING?

In the early 1980s Atomic Energy of Canada Limited (AECL) manufactured and sold a cancer radiation treatment machine called the Therac 25, which relied on computer software to control its operation. Between 1985 and 1987, the Therac-25 caused the deaths of three patients and serious injuries to three others (3). Who was responsible for the accidents? The operator who administered the massive radiation overdoses, which produced severe burns? The software developers who wrote and tested the control software, which contained several serious errors? The system engineers who neglected to install the backup hardware safety mechanisms that had been used in previous versions of the machine? The manufacturer, AECL? Government agencies? We can use the Therac-25 case to distinguish among four different kinds of responsibility (4,5).

Causal Responsibility

Responsibility can be attributed to causes: For example, "the tornado was responsible for damaging the house." In the Therac-25 case, the proximate cause of each accident was the operator, who started the radiation treatment. But just as the weather cannot be blamed for a moral failing, the

Therac-25 operators cannot be blamed because they followed standard procedures, and the information displayed on the computer monitors was cryptic and misleading.

Role Responsibility

An individual who is assigned a task or function is considered the responsible person for that role. In this sense, a foreman in a chemical plant may be responsible for disposing of drums of toxic waste, even if a forklift operator actually transfers the drums from the plant to the truck. In the Therac-25 case, the software developers and system engineers were assigned the responsibility of designing the software and hardware of the machine. Insofar as their designs were deficient, they were responsible for those deficiencies because of their roles. Even if they had completed their assigned tasks, however, their role responsibility may not encompass the full extent of their professional responsibilities.

Legal Responsibility

An individual or an organization can be legally responsible, or liable, for a problem. That is, the individual could be charged with a crime, or the organization could be liable for damages in a civil lawsuit. Similarly, a physician can be sued for malpractice. In the Therac-25 case, AECL could have been sued. One kind of legal responsibility is *strict liability*: If a product injures someone, then the manufacturer of the product can be found liable for damages in a lawsuit, even if the product met all applicable safety standards and the manufacturer did nothing wrong. The principle of strict liability encourages manufacturers to be careful, and it provides a way to compensate victims of accidents.

Moral Responsibility

Causal, role, and legal responsibilities tend to be exclusive: If one individual is responsible, then another is not. In contrast, moral responsibility tends to be shared: many engineers are responsible for the safety of the products that they design, not just a designated safety engineer. Furthermore, rather than assign blame for a past event, moral responsibility focuses on what individuals should do in the future. In the moral sense, responsibility is a virtue: A "responsible person" is careful, considerate, and trustworthy; an "irresponsible person" is reckless, inconsiderate, and untrustworthy.

Responsibility is shared whenever multiple individuals collaborate as a group, such as a software development team. When moral responsibility is shared, responsibility is *not* atomized to the point at which no one in the group is responsible. Rather, each member of the group is accountable to the other members of the group and to those whom the group's work might affect, both for the individual's own actions and for the effects of their collective effort. For example, suppose a computer network monitoring team has made mistakes in a complicated statistical analysis of network traffic data, and that these mistakes have changed the interpretation of the reported results. If the team members do not reanalyze the data themselves, they have an obligation to seek the assistance of a statistician who can analyze the data correctly. Different team members might work with the statistician in different ways, but they should hold each other accountable for their individual roles in correcting the mistakes. Finally, the team has a collective moral responsibility to inform readers of the team's initial report about the mistakes and the correction.

Moral responsibility for recklessness and negligence is not mitigated by the presence of good intentions or by the absence of bad consequences. Suppose a software tester neglects to sufficiently test a new module for a telephone switching system, and the module fails. Although the subsequent telephone service outages are not intended, the software tester is morally responsible for the harms caused by the outages. Suppose a hacker installs a keystroke logging program in a deliberate attempt to steal passwords at a public computer. Even if the program fails to work, the hacker is still morally responsible for attempting to invade the privacy of users.

An individual can be held morally responsible both for acting and for failing to act. For example, a hardware engineer might notice a design flaw that could result in a severe electrical shock to someone who opens a personal computer system unit to replace a memory chip. Even if the engineer is not specifically assigned to check the electrical safety of the system unit, the engineer is morally responsible for calling attention to the design flaw, and the engineer can be held accountable for failing to act.

Computing systems often obscure accountability (5). In particular, in an embedded system such as the Therac-25, the computer that controls the device is hidden. Computer users seem resigned to accepting defects in computers and software that cause intermittent crashes and losses of data. Errors in code are called "bugs," regardless of whether they are minor deficiencies or major mistakes that could cause fatalities. In addition, because computers seem to act autonomously, people tend to blame the computers themselves for failing, instead of the professionals who designed, programmed, and deployed the computers.

WHAT ARE THE RESPONSIBILITIES OF COMPUTING PROFESSIONALS?

Responsibilities to Clients and Users

Whether a computing professional works as a consultant to an individual or as an employee in a large organization, the professional is obligated to perform assigned tasks competently, according to professional standards. These professional standards include not only attention to technical excellence but also concern for the social effects of computers on operators, users, and the public. When assessing the capabilities and risks of computer systems, the professional must be candid: The professional must report all relevant findings honestly and accurately. When designing a new computer system, the professional must consider not only the specifications of the client but also how the system might affect the quality of life of users and others. For example, a computing professional who designs an information system for a hospital and should allow speedy access by physicians and nurses and yet protect patients' medical records from unauthorized access; the technical requirement to provide fast access may conflict with the social obligation to ensure patients' privacy.

Computing professionals enjoy considerable freedom in deciding how to meet the specifications of a computer system. Provided that they meet the minimum performance requirements for speed, reliability, and functionality, within an overall budget, they may choose to invest resources to decrease the response time rather than to enhance a graphical user interface, or vice versa. Because choices involve tradeoffs between competing values, computing professionals should identify potential biases in their design choices (6). For example, the designer of a search engine for an online retailer might choose to display the most expensive items first. This choice might favor the interest of the retailer, to maximize profit, over the interest of the customer, to minimize cost.

Even moderately large software artifacts (computer programs) are inherently complex and error-prone. Furthermore, software is generally becoming more complex. It is therefore reasonable to assume that all software artifacts have errors. Even if a particular artifact does not contain errors, it is extremely difficult to prove its correctness. Faced with these realities, how can a responsible software engineer release software that is likely to fail sometime in the future? Other engineers confront the same problem, because all engineering artifacts eventually fail. Whereas most engineering artifacts fail because physical objects wear out, software artifacts are most likely to fail because of faults *designed into* the original artifact. The intrinsically faulty nature of software distinguishes it from light bulbs and I-beams, for example, whose failures are easier to predict statistically.

To acknowledge responsibilities for the failure of software artifacts, software developers should exercise due diligence in creating software, and they should be as candid as possible about both known and unknown faults in the software—particularly software for *safety-critical systems*, in which a failure can threaten the lives of people. Candor by software developers would give software consumers a better chance to make reasonable decisions about software before they buy it (7). Following an established tradition in medicine, Miller (8) advocates "software informed consent" as a way to formalize an ethical principle that requires openness from software developers. Software informed consent requires software developers to reveal, using explanations that are understandable to their customers, the risks of their software, including the likelihoods of known faults and the probabilities that undiscovered faults still exist. The idea of software informed consent motivates candor and requires continuing research into methods of discovering software faults and measuring risk.

Responsibilities to Employers

Most computing professionals work for employers. The employment relationship is contractual: The professional promises to work for the employer in return for a salary and benefits. Professionals often have access to the employer's proprietary information such as trade secrets, and the professional must keep this information confidential. Besides trade secrets, the professional must also honor other forms of *intellectual property* owned by the employer: The professional does not have the right to profit from independent sale or use of this intellectual property, including software developed with the employer's resources.

Every employee is expected to work loyally on behalf of the employer. In particular, professionals should be aware of potential conflicts of interest, in which loyalty might be owed to other parties besides the employer. A *conflict of interest* occurs when a professional is asked to render a judgment, but the professional has personal or financial interests that may interfere with the exercise of that judgment. For instance, a computing professional may be responsible for ordering computing equipment, and an equipment vendor owned by the professional's spouse might submit a bid. In this case, others would perceive that the marriage relationship might bias the professional's judgment. Even if the spouse's equipment would be the best choice, the professional's judgment would not be trustworthy. In a typical conflict of interest situation, the professional should *recuse* herself: that is, the professional should remove herself and ask another qualified person to make the decision.

Many computing professionals have managerial duties, and some are solely managers. Managerial roles complicate the responsibilities of computing professionals because managers have administrative responsibilities and interests within their organizations in addition to their professional responsibilities to clients and the public.

Responsibilities to Other Professionals

Although everyone deserves respect from everyone else, when professionals interact with each other, they should demonstrate a kind of respect called *collegiality*. For example, when one professional uses the ideas of a second professional, the first should credit the second. In a research article, an author gives credit by properly citing the sources of ideas from other authors in previously published articles. Using these ideas without attribution constitutes plagiarism. Academics consider plagiarism unethical because it represents the theft of ideas and the misrepresentation of those ideas as the plagiarist's own.

Because clients cannot adequately evaluate the quality of professional service, individual professionals know that their work must be evaluated by other members of the same profession. This evaluation, called *peer review* occurs in both practice and research. Research in computing is presented at conferences and is published in scholarly journals. Before a manuscript that reports a research project can be accepted for a conference or published in a journal, the manuscript must be reviewed by peer researchers who are experts in the subject of the manuscript.

Because computing professionals work together, they must observe *professional standards*. These standards of practice are created by members of the profession or within organizations. For example, in software development, one standard of practice is a convention for names of variables in code. By following coding standards, a software developer can facilitate the work of a software maintainer who subsequently modifies the code. For many important issues for which standards would be appropriate theoretically, however, “standards” in software engineering are controversial, informal, or nonexistent. An example of this problem is the difficulties encountered when the IEEE and the ACM attempted to standardize a body of knowledge for software engineering to enable the licensing of software engineers.

Senior professionals have an obligation to *mentor* junior professionals in the same field. Although professionals are highly educated, junior members of a profession require additional learning and experience to develop professional judgment. This learning is best accomplished under the tutelage of a senior professional. In engineering, to earn a P.E. license, a junior engineer must work under the supervision of a licensed engineer for at least four years. More generally, professionals should assist each other in continuing education and professional development, which are generally required for maintaining licensure.

Professionals can fulfill their obligations to contribute to the profession *volunteering*. The peer review of research publications depends heavily on volunteer reviewers and editors, and the activities of professional associations are conducted by committees of volunteers.

Responsibilities to the Public

According to engineering codes of ethics, the engineer’s most important obligation is to ensure the safety, health, and welfare of the public. Although everyone must avoid endangering others, engineers have a special obligation to ensure the safety of the objects that they produce. Computing professionals share this special obligation to guarantee the safety of the public and to improve the quality of life of those who use computers and information systems.

As part of this obligation, computing professionals should enhance the public’s understanding of computing. The responsibility to educate the public is a collective responsibility of the computing profession as a whole; individual professionals might fulfill this responsibility in their own ways. Examples of such public service include advising a church on the purchase of computing equipment and writing a letter to the editor of a newspaper about technical issues related to proposed legislation to regulate the Internet.

It is particularly important for computing professionals to contribute their technical knowledge to discussions about public policies regarding computing. Many communities are considering controversial measures such as the installation of Web filtering software on public access computers in libraries. Computing professionals can participate in communities’ decisions by providing technical facts. Technological controversies involving the social impacts of computers are covered in a separate article of this encyclopedia.

When a technical professional’s obligation of loyalty to the employer conflicts with the obligation to ensure the safety of the public, the professional may consider *whistlebhwing,* that is, alerting people outside the employer’s organization to a serious, imminent threat to public safety. Computer engineers blew the whistle during the development of the Bay Area Rapid Transit (BART) system near San Francisco (9). In the early 1970s, three BART engineers became alarmed by deficiencies in the design of the electronics and software for the automatic train control system, deficiencies that could have endangered passengers on BART trains. The engineers raised their concerns within the BART organization without success. Finally, they contacted a member of the BART board of directors, who passed their concerns to Bay Area newspapers. The three engineers were immediately fired for disloyalty. They were never reinstated, even when an accident proved their concerns were valid. When the engineers sued the BART managers, the IEEE filed an *amicus curiae* brief on the engineers’ behalf, stating that engineering codes of ethics required the three engineers to act to protect the safety of the public. The next section describes codes of ethics for computing professionals.

CODES OF ETHICS

For each profession, the professional’s obligations to clients, employers, other professionals, and the public are stated explicitly in the profession’s code of ethics or code of professional conduct. For computing professionals, such codes have been developed by ACM, the British Computer Society (BCS), the, IEEE-CS, the AITP, the Hong Kong Computer Society, the Systems Administrators Special Interest Group of USENEX (SAGE), and other associations. Two of these codes will be described briefly here: the ACM code and the Software Engineering Code jointly approved by the IEEE-CS and the ACM.

The ACM is one of the largest nonprofit scientific and educational organization devoted to computing. In 1966 and 1972, the ACM published codes of ethics for computing professionals. In 1992, the ACM adopted the current Code of Ethics and Professional Conduct (10), which appears in Appendix 1. Each statement of the code is accompanied by interpretive guidelines. For example, the guideline for statement 1.8, *Honor confidentiality*, indicates that other ethical imperatives such as complying with a law may take precedence. Unlike ethics codes for other professions, one section of the ACM code states the ethical obligations of “organizational leaders,” who are typically technical managers.

The ACM collaborated with the IEEE-CS to produce the Software Engineering Code of Ethics and Professional Practice (11). Like the ACM code, the Software Engineering Code also includes the obligations of technical managers. This code is notable in part because it was the first code to focus exclusively on software engineers, not on other computing professionals. This code is broken into a short version is composed of and a long version. The short version is composed of a preamble and eight short principles; this version appears in Appendix 2. The long version expands on the eight principles with multiple clauses that apply the principles to specific issues and situations.

Any code of ethics is necessarily incomplete—no document can address every possible situation. In addition, a code must be written in general language; each statement in a code requires interpretation to be applied in specific circumstances. Nevertheless, a code of ethics can serve multiple purposes (12,13). A code can inspire members of a profession to strive for the profession's ideals. A code can educate new members about their professional obligations, and tell nonmembers what they may expect members to do, A code can set standards of conduct for professionals and provide a basis for expelling members who violate these standards. Finally, a code may support individuals in making difficult decisions. For example, because all engineering codes of ethics prioritize the safety and welfare of the public, an engineer can object to unsafe practices not merely as a matter of individual conscience but also with the full support of the consensus of the profession. The application of a code of ethics for making decisions is highlighted in the next section.

ETHICAL DECISION MAKING FOR COMPUTING PROFESSIONALS

Every user of e-mail has received unsolicited bulk commercial e-mail messages, known in a general way as *spam*. (A precise definition of "spam" has proven elusive and is controversial; most people know spam when they see it, but legally and ethically a universally accepted definition has not yet emerged.) A single spam broadcast can initiate millions of messages. Senders of spam claim that they are exercising their free speech rights, and few laws have been attempted to restrict it. In the United States, no federal law prohibited spamming before the CAN-SPAM Act of 2003. Even now, the CAN-SPAM law does not apply to spam messages that originate in other countries. Although some prosecutions have occurred using the CAN-SPAM Act, most people still receive many e-mail messages that they consider spam.

Some spam messages are designed to be deceptive and include intentionally inaccurate information, but others include only accurate information. Although most spamming is not illegal, even honest spamming is considered unethical by many people, for the following reasons. First, spamming has bad consequences: It wastes the time of recipients who must delete junk e-mail messages, and these messages waste space on computers; in addition, spamming reduces users' trust in e-mail. Second, spamming is not reversible: Senders of spam do not want to receive spam. Third, spamming could not be allowed as a general practice: If everyone attempted to broadcast spam messages to wide audiences, computer networks would become clogged with unwanted e-mail messages, and no one could communicate via e-mail at all.

The three reasons advanced against spam correspond to three ways in which the morality of an action can be evaluated: first, whether on balance the action results in more good consequences than bad consequences; second, whether the actor would be willing to trade places with someone affected by the action: and third, whether everyone (in a similar situation) could choose the same action as a general rule. These three kinds of moral reasons correspond to three of the many traditions in philosophical ethics: consequentialism, the Golden Rule, and duty-based ethics.

Ethical issues in the use of computers can also be evaluated through the use of analogies to more familiar situations. For example, a hacker may try to justify gaining unauthorized access to unsecured data by reasoning that because the data are not protected, anyone should be able to read it. But by analogy, someone who finds the front door of a house unlocked is not justified in entering the house and snooping around. Entering an unlocked house is trespassing, and trespassing violates the privacy of the house's occupants.

When making ethical decisions, computing professionals can rely not only on general moral reasoning but also on specific guidance from codes of ethics, such as the ACM Code of Ethics (10). Here is a fictional example of that approach.

> ***Scenario:*** XYZ Corporation plans to monitor secretly the Web pages visited by its employees, using a data mining program to analyze the access records. Chris, an engineer at XYZ, recommends that XYZ purchase a data mining program from Robin, an independent contractor, without mentioning that Robin is Chris's domestic partner. Robin had developed this program while previously employed at UVW Corporation, without the awareness of anyone at UVW.
> ***Analysis:*** First, the monitoring of Web accesses intrudes on employees' privacy; it is analogous to eavesdropping on telephone calls. Professionals should respect the privacy of individuals (ACM Code 1.7, *Respect the privacy of others*, and 3.5, *Articulate and support policies that protect the dignity of users and others affected by a computing system).* Second, Chris has a conflict of interest because the sale would benefit Chris's domestic partner. By failing to mention this relationship, Chris was disingenuous (ACM Code 1.3, *Be honest and trustworthy).* Third, because Robin developed the program while working at UVW, some and perhaps all of the property rights belong to UVW. Robin probably signed an agreement that software developed while employed at UVW belongs to UVW. Professionals should honor property rights and contacts (ACM Code 1.5, *Honor property rights including copyrights and patent*, and 2.6, *Honor contracts, agreements, and assigned responsibilities).*

Applying a code of ethics might not yield a clear solution of an ethical problem because different principles in a code might conflict. For instance, the principles of honesty and confidentiality conflict when a professional who is questioned about the technical details of the employer's forthcoming product must choose between answering the question completely and keeping the information secret. Consequently, more sophisticated methods have been developed for solving ethical problems. Maner (14) has studied and collected what he calls "procedural ethics, step-by-step ethical reasoning procedures ... that may prove useful to computing professionals engaged in ethical decision-making." Maner's list includes a method specialized for business ethics (15), a paramedic method (16), and a procedure from the U.S. Department of Defense (17). These procedures appeal to the problem-solving ethos of engineering, and they help professionals avoid specific traps that might otherwise impair a professional's ethical judgment. No procedural ethics method should be interpreted as allowing complete objectivity or providing a mechanical algorithm for reaching a conclusion about an ethical problem, however, because all professional ethics issues of any complexity require subtle and subjective judgments.

COMPUTING AND THE STUDY OF ETHICS: THE ETHICAL CHALLENGES OF ARTIFICIAL INTELLIGENCE AND AUTONOMOUS AGENTS

Many ethical issues, such as conflict of interest, are common to different professions. In computing and engineering, however, unique ethical issues develop from the creation of machines whose outward behaviors resemble human behaviors that we consider "intelligent." As machines become more versatile and sophisticated, and as they increasingly take on tasks that were once assigned only to humans, computing professionals and engineers must rethink their relationship to the artifacts they design, develop, and then deploy.

For many years, ethical challenges have been part of discussions of artificial intelligence. Indeed, two classic references in the field are by Norbert Wiener in 1965 (18) and by Joseph Weizenbaum in 1976 (19). Since the 1990s, the emergence of sophisticated "autonomous agents," including Web "bots" and physical robots, has intensified the ethical debate. Two fundamental issues are of immediate concern: the responsibility of computing professionals who create these sophisticated machines, and the notion that the machines themselves will, if they have not already done so, become sufficiently sophisticated so that they will be considered themselves moral agents, capable of ethical praise or blame independent of the engineers and scientists who developed them. This area of ethics is controversial and actively researched. A full discussion of even some of the nuances is beyond the scope of this article. Recent essays by Floridi and Sanders (20) and Himma (21) are two examples of influential ideas in the area.

APPENDIX 1. ACM CODE OF ETHICS AND PROFESSIONAL CONDUCT

http://www.acm.org/about/code-of-ethics.

PREAMBLE

Commitment to ethical professional conduct is expected of every member (voting members, associate members, and student members) of the Association for Computing Machinery (ACM).

This Code, consisting of 24 imperatives formulated as statements of personal responsibility, identifies the elements of such a commitment. It contains many, but not all, issues professionals are likely to face. Section 1 outlines fundamental ethical considerations, while Section 2 addresses additional, more specific considerations of professional conduct. Statements in Section 3 pertain more specifically to individuals who have a leadership role, whether in the workplace or in a volunteer capacity such as with organizations like ACM. Principles involving compliance with this Code are given in Section 4.

The Code shall be supplemented by a set of Guidelines, which provide explanation to assist members in dealing with the various issues contained in the Code. It is expected that the Guidelines will be changed more frequently than the Code.

The Code and its supplemented Guidelines are intended to serve as a basis for ethical decision making in the conduct of professional work. Secondarily, they may serve as a basis for judging the merit of a formal complaint pertaining to violation of professional ethical standards.

It should be noted that although computing is not mentioned in the imperatives of Section 1, the Code is concerned with how these fundamental imperatives apply to one's conduct as a computing professional. These imperatives are expressed in a general form to emphasize that ethical principles which apply to computer ethics are derived from more general ethical principles.

It is understood that some words and phrases in a code of ethics are subject to varying interpretations, and that any ethical principle may conflict with other ethical principles in specific situations. Questions related to ethical conflicts can best be answered by thoughtful consideration of fundamental principles, rather than reliance on detailed regulations.

1. GENERAL MORAL IMPERATIVES

As an ACM member I will . . . .

1.1 Contribute to society and human well-being

This principle concerning the quality of life of all people affirms an obligation to protect fundamental human rights and to respect the diversity of all cultures. An essential aim of computing professionals is to minimize negative consequences of computing systems, including threats to health and safety. When designing or implementing systems, computing professionals must attempt to ensure that the products of their efforts will be used in socially responsible ways, will meet social needs, and will avoid harmful effects to health and welfare.

In addition to a safe social environment, human well-being includes a safe natural environment. Therefore, computing professionals who design and develop systems must be alert to, and make others aware of, any potential damage to the local or global environment.

1.2 Avoid harm to others

"Harm" means injury or negative consequences, such as undesirable loss of information, loss of property, property damage, or unwanted environmental impacts. This principle prohibits use of computing technology in ways that result in harm to any of the following: users, the general public, employees, employers. Harmful actions include intentional destruction or modification of files and programs leading to serious loss of resources or unnecessary expenditure of human resources such as the time and effort required to purge systems of "computer viruses."

Well-intended actions, including those that accomplish assigned duties, may lead to harm unexpectedly. In such an event the responsible person or persons are obligated to undo or mitigate the negative consequences as much as possible. One way to avoid unintentional harm is to carefully consider potential impacts on all those affected by decisions made during design and implementation.

To minimize the possibility of indirectly harming others, computing professionals must minimize malfunctions by following generally accepted standards for system design and testing. Furthermore, it is often necessary to assess the social consequences of systems to project the likelihood of any serious harm to others. If system features are misrepresented to users, coworkers, or supervisors, the individual computing professional is responsible for any resulting injury.

In the work environment the computing professional has the additional obligation to report any signs of system dangers that might result in serious personal or social damage. If one's superiors do not act to curtail or mitigate such dangers, it may be necessary to "blow the whistle" to help correct the problem or reduce the risk. However, capricious or misguided reporting of violations can, itself, be harmful. Before reporting violations, all relevant aspects of the incident must be thoroughly assessed. In particular, the assessment of risk and responsibility must be credible. It is suggested that advice be sought from other computing professionals. See principle 2.5 regarding thorough evaluations.

1.3 Be honest and trustworthy

Honesty is an essential component of trust. Without trust an organization cannot function effectively. The honest computing professional will not make deliberately false or deceptive claims about a system or system design, but will instead provide full disclosure of all pertinent system limitations and problems.

A computer professional has a duty to be honest about his or her own qualifications, and about any circumstances that might lead to conflicts of interest.

Membership in volunteer organizations such as ACM may at times place individuals in situations where their statements or actions could be interpreted as carrying the "weight" of a larger group of professionals. An ACM member will exercise care to not misrepresent ACM or positions and policies of ACM or any ACM units.

1.4 Be fair and take action not to discriminate

The values of equality, tolerance, respect for others, and the principles of equal justice govern this imperative. Discrimination on the basis of race, sex, religion, age, disability, national origin, or other such factors is an explicit violation of ACM policy and will not be tolerated.

Inequities between different groups of people may result from the use or misuse of information and technology. In a fair society, all individuals would have equal opportunity to participate in, or benefit from, the use of computer resources regardless of race, sex, religion, age, disability, national origin or other such similar factors. However, these ideals do not justify unauthorized use of computer resources nor do they provide an adequate basis for violation of any other ethical imperatives of this code.

1.5 Honor property rights including copyrights and patent

Violation of copyrights, patents, trade secrets and the terms of license agreements is prohibited by law in most circumstances. Even when software is not so protected, such violations are contrary to professional behavior. Copies of software should be made only with proper authorization. Unauthorized duplication of materials must not be condoned

1.6 Give proper credit for intellectual property

Computing professionals are obligated to protect the integrity of intellectual property. Specifically, one must not take credit for other's ideas or work, even in cases where the work has not been explicitly protected by copyright, patent, etc.

1.7 Respect the privacy of others

Computing and communication technology enables the collection and exchange of personal information on a scale unprecedented in the history of civilization. Thus there is increased potential for violating the privacy of individuals and groups. It is the responsibility of professionals to maintain the privacy and integrity of data describing individuals. This includes taking precautions to ensure the accuracy of data, as well as protecting it from unauthorized access or accidental disclosure to inappropriate individuals. Furthermore, procedures must be established to allow individuals to review their records and correct inaccuracies.

This imperative implies that only the necessary amount of personal information be collected in a system, that retention and disposal periods for that information be clearly defined and enforced, and that personal information gathered for a specific purpose not be used for other purposes without consent of the individual(s). These principles apply to electronic communications, including electronic mail, and prohibit procedures that capture or monitor electronic user data, including messages, without the permission of users or bona fide authorization related to system operation and maintenance. User data observed during the normal duties of system operation and maintenance must be treated with strictest confidentiality, except in cases where it is evidence for the violation of law, organizational regulations, or this Code. In these cases, the nature or contents of that information must be disclosed only to proper authorities.

1.8 Honor confidentiality

The principle of honesty extends to issues of confidentiality of information whenever one has made an explicit promise to honor confidentiality or, implicitly, when private information not directly related to the performance of one's duties becomes available. The ethical concern is to respect all obligations of confidentiality to employers, clients, and users unless discharged from such obligations by requirements of the law or other principles of this Code.

2 MORE SPECIFIC PROFESSIONAL RESPONSIBILITIES

As an ACM computing professional I will . . ..

2.1 Strive to achieve the highest quality, effectiveness and dignity in both the process and products of professional work

Excellence is perhaps the most important obligation of a professional. The computing professional must strive to achieve quality and to be cognizant of the serious negative consequences that may result from poor quality in a system.

2.2 Acquire and maintain professional competence

Excellence depends on individuals who take responsibility for acquiring and maintaining professional competence. A professional must participate in setting standards for appropriate levels of competence, and strive to achieve those standards. Upgrading technical knowledge and competence can be achieved in several ways: doing independent study; attending seminars, conferences, or courses; and being involved in professional organizations.

2.3 Know and respect existing laws pertaining to professional work

ACM members must obey existing local, state, province, national, and international laws unless there is a compelling ethical basis not to do so. Policies and procedures of the organizations in which one participates must also be obeyed. But compliance must be balanced with the recognition that sometimes existing laws and rules may be immoral or inappropriate and, therefore, must be challenged. Violation of a law or regulation may be ethical when that law or rule has inadequate moral basis or when it conflicts with another law judged to be more important. If one decides to violate a law or rule because it is viewed as unethical, or for any other reason, one must fully accept responsibility for one's actions and for the consequences.

2.4 Accept and provide appropriate professional review

Quality professional work, especially in the computing profession, depends on professional reviewing and critiquing. Whenever appropriate, individual members should seek and utilize peer review as well as provide critical review of the work of others.

2.5 Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks

Computer professionals must strive to be perceptive, thorough, and objective when evaluating, recommending, and presenting system descriptions and alternatives. Computer professionals are in a position of special trust, and therefore have a special responsibility to provide objective, credible evaluations to employers, clients, users, and the public. When providing evaluations the professional must also identify any relevant conflicts of interest, as stated in imperative 1.3.

As noted in the discussion of principle 1.2 on avoiding harm, any signs of danger from systems must be reported to those who have opportunity and/or responsibility to resolve them. See the guidelines for imperative 1.2 for more details concerning harm, including the reporting of professional violations.

2.6 Honor contracts, agreements, and assigned responsibilities

Honoring one's commitments is a matter of integrity and honesty. For the computer professional this includes ensuring that system elements perform as intended. Also, when one contracts for work with another party, one has an obligation to keep that party properly informed about progress toward completing that work.

A computing professional has a responsibility to request a change in any assignment that he or she feels cannot be completed as defined. Only after serious consideration and with full disclosure of risks and concerns to the employer or client, should one accept the assignment. The major underlying principle here is the obligation to accept personal accountability for professional work. On some occasions other ethical principles may take greater priority.

A judgment that a specific assignment should not be performed may not be accepted. Having clearly identified one's concerns and reasons for that judgment, but failing to procure a change in that assignment, one may yet be obligated, by contract or by law, to proceed as directed. The computing professional's ethical judgment should be the final guide in deciding whether or not to proceed. Regardless of the decision, one must accept the responsibility for the consequences.

However, performing assignments "against one's own judgment" does not relieve the professional of responsibility for any negative consequences.

2.7 Improve public understanding of computing and its consequences

Computing professionals have a responsibility to share technical knowledge with the public by encouraging understanding of computing, including the impacts of computer systems and their limitations. This imperative implies an obligation to counter any false views related to computing.

2.8 Access computing and communication resources only when authorized to do so

Theft or destruction of tangible and electronic property is prohibited by imperative 1.2 - "Avoid harm to others." Trespassing and unauthorized use of a computer or communication system is addressed by this imperative. Trespassing includes accessing communication networks and computer systems, or accounts and/or files associated with those systems, without explicit authorization to do so. Individuals and organizations have the right to restrict access to their systems so long as they do not violate the discrimination principle (see 1.4), No one should enter or use another's computer system, software, or data files without permission. One must always have appropriate approval before using system resources, including communication ports, file space, other system peripherals, and computer time.

3. ORGANIZATIONAL LEADERSHIP IMPERATIVES

As an ACM member and an organizational leader, I will . . . .

BACKGROUND NOTE: This section draws extensively from the draft IFIP Code of Ethics, especially its sections on organizational ethics and international concerns. The ethical obligations of organizations tend to be neglected in most codes of professional conduct, perhaps because these codes are written from the perspective of the individual member. This dilemma is addressed by stating these imperatives from the perspective of the organizational leader. In this context "leader" is viewed as any organizational member who has leadership or educational responsibilities. These imperatives generally may apply to organizations as well as their leaders. In this context "organizations" are corporations, government agencies, and other "employers," as well as volunteer professional organizations.

3.1 Articulate social responsibilities of members of an organizational unit and encourage full acceptance of those responsibilities

Because organizations of all kinds have impacts on the public, they must accept responsibilities to society. Organizational procedures and attitudes oriented toward quality and the welfare of society will reduce harm to members of the public, thereby serving public interest and fulfilling social responsibility. Therefore, organizational leaders must encourage full participation in meeting social responsibilities as well as quality performance.

3.2 Manage personnel and resources to design and build information systems that enhance the quality of working life

Organizational leaders are responsible for ensuring that computer systems enhance, not degrade, the quality of working life. When implementing a computer system, organizations must consider the personal and professional development, physical safety, and human dignity of all workers. Appropriate human-computer ergonomic standards should be considered in system design and in the workplace.

3.3 Acknowledge and support proper and authorized uses of an organization's computing and communication resources

Because computer systems can become tools to harm as well as to benefit an organization, the leadership has the responsibility to clearly define appropriate and inappropriate uses of organizational computing resources. While the number and scope of such rules should be minimal, they should be fully enforced when established.

3.4 Ensure that users and those who will be affected by a system have their needs clearly articulated during the assessment and design of requirements; later the system must be validated to meet requirements

Current system users, potential users and other persons whose lives may be affected by a system must have their needs assessed and incorporated in the statement of requirements. System validation should ensure compliance with those requirements.

3.5 Articulate and support policies that protect the dignity of users and others affected by a computing system

Designing or implementing systems that deliberately or inadvertently demean individuals or groups is ethically unacceptable. Computer professionals who are in decision making positions should verify that systems are designed and implemented to protect personal privacy and enhance personal dignity.

3.6 Create opportunities for members of the organization to learn the principles and limitations of computer systems

This complements the imperative on public understanding (2.7). Educational opportunities are essential to facilitate optimal participation of all organizational members. Opportunities must be available to all members to help them improve their knowledge and skills in computing, including courses that familiarize them with the consequences and limitations of particular types of systems. In particular, professionals must be made aware of the dangers of building systems around oversimplified models, the improbability of anticipating and designing for every possible operating condition, and other issues related to the complexity of this profession.

4. COMPLIANCE WITH THE CODE

As an ACM member I will . . . .

4.1 Uphold and promote the principles of this code

The future of the computing profession depends on both technical and ethical excellence. Not only is it important for ACM computing professionals to adhere to the principles expressed in this Code, each member should encourage and support adherence by other members.

4.2 Treat violations of this code as inconsistent with membership in the ACM

Adherence of professionals to a code of ethics is largely a voluntary matter. However, if a member does not follow this code by engaging in gross misconduct, membership in ACM may be terminated.

APPENDIX 2: SOFTWARE ENGINEERING CODE OF ETHICS AND PROFESSIONAL PRACTICE (SHORT VERSION)

http://www.acm.org/about/se-code/
Short Version

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

BIBLIOGRAPHY

1. A. Goldman. *The Moral Foundation of Professional Ethics*, Totowa, NJ: Rowman & Littlefield, 1980.
2. J. White and B. Simons, ACM's position on the licensing of software engineers, *Communications ACM*, **45**(11): 91, 2002.
3. N. G. Leveson and C. S. Turner, An investigation of the Therac-25 accidents, *Computer*, **26**(7): 18–41, 1993.
4. J. Ladd, Collective and individual moral responsibility in engineering: some questions, *IEEE Technology and Society Magazine*, **1**(2): 3–10, 1982.
5. H. Nissenbaum, Computing and accountability, *Communications of the ACM*, **37**(1): 73–80, 1994.
6. B. Friedman and H. Nissenbaum, Bias in computer systems, *ACM Transa. Informa. Sys.*, **14**(3): 330–347, 1996.
7. C. Kaner, Blog: Software customer bill of right. Available: from http://www.satisfice.com/kaner/.
8. K. Miller, Software informed consent: docete emptorem, not caveat emptor, *Science Engineer. Ethics*, **4**(3): 357–362, 1998.
9. G. D. Friedlander, The case of the three engineers vs. BART, *IEEE Spectrum*, **11**(10): 69–76, 1974.
10. R. Anderson, D. G. Johnson, D. Gotterbarn, and J. Perrolle, Using the new ACM code of ethics in decision making, *Communications ACM*, **36**(2): 98–107, 1993.
11. D. Gotterbarn, K. Miller, and S. Rogerson, Software engineering code of ethics is approved, *Communications ACM*, **42**(10): 102–107, 1999.
12. M. Davis, Thinking like an engineer: the place of a code of ethics in the practice of a profession, *Philosophy and Public Affairs*, **20**(2): 150–167, 1991.
13. D. Gotterbarn, Computing professionals and your responsibilities: virtual information and the software engineering code of ethics, in D. Langford (ed.), *Internet Ethics*, New York: St. Martin's Press, 2000, pp. 200–219.
14. W. Maner, Heuristic methods for computer ethics, *Metaphilosophy*, **33**(3): 339–365, 2002.
15. D. L. Mathison, Teaching an ethical decision model that skips the philosophers and works for real business students, *Proceedings, National Academy of Management*, New Orleans: 1987, pp. 1–9.
16. W. R. Collins and K. Miller, A paramedic method for computing professionals, *J. Sys. Software*, **17**(1): 47–84, 1992.
17. "United States Department of Defense. Joint ethics regulation DoD 5500.7-R." 1999. Available: http://www.defenselink.mil/dodg/defense_ethics/ethics_regulation/jerl-4.doc.
18. N. Wiener, *Cybernetics: or the Control and Communication in the Animal and the Machine*, Cambridge, MA: MIT Press, 1965.

19. J. Weizenbaum, *Computer Power and Human Reason: From Judgment to Calcution*, New York: WH Freeman & Co., 1976.

20. L. Floridi and J. Sanders On the morality of artificial agents, *Minds and Machines*, **14**(3): 349–379, 2004.

21. K. Himma, There's something about Mary: the moral value of things *qua* information objects, *Ethics Informat. Technol.*, **6**(3): 145–159, 2004.

FURTHER READING

D. G. Johnson, Professional ethics, in *Computer Ethics*, 3rd ed., Upper Saddle River, NJ: Prentice Hall, 2001, pp. 54–80.

M. J. Quinn, Professional ethics, in *Ethics for the Information Age*, Boston, MA: Pearson/Addison-Wesley, 2005, pp. 365–403.

H. Tavani, Professional ethics, codes of conduct, and moral responsibility, in *Ethics and Technology: Ethical Issues in an Age of Information and Communication Technology*, New York: Wiley, 2004, pp. 87–116.

MICHAEL C. LOUI
University of Illinois at
Urbana-Champaign
Urbana, Illinois
KEITH W. MILLER
University of Illinois at
Springfield
Springfield, Illinois

FIXED-POINT COMPUTER ARITHMETIC

INTRODUCTION

This article begins with a brief discussion of the two's complement number system in the section on "Number Systems." The section on "Arithmetic Algorithms" provides examples of implementations of the four basic arithmetic operations (i.e., add, subtract, multiply, and divide).

Regarding notation, capital letters represent digital numbers (i.e., n-bit words), whereas subscripted lowercase letters represent bits of the corresponding word. The subscripts range from n – 1 to 0 to indicate the bit position within the word (x_{n-1} is the most significant bit of X, x_0 is the least significant bit of X, etc.). The logic designs presented in this chapter are based on positive logic with AND, OR, and INVERT operations. Depending on the technology used for implementation, different logical operations (such as NAND and NOR) or direct transistor realizations may be used, but the basic concepts do not change significantly.

NUMBER SYSTEMS

At the current time, fixed-point binary numbers are represented generally using the two's complement number system. This choice has prevailed over the sign magnitude and one's complement number systems, because the frequently performed operations of addition and subtraction are easiest to perform on two's complement numbers. Sign magnitude numbers are more efficient for multiplication, but the lower frequency of multiplication and the development of the efficient Booth's two's complement multiplication algorithm have resulted in the nearly universal selection of the two's complement number system for most implementations. The algorithms presented in this article assume the use of two's complement numbers.

Fixed-point number systems represent numbers, for example A, by n bits: a sign bit, and $n-1$ "data" bits. By convention, the most significant bit, a_{n-1}, is the sign bit, which is generally a ONE for negative numbers and a ZERO for positive numbers. The $n-1$ data bits are a_{n-2}, a_{n-3},. . ., a_1, a_0. In the two's complement fractional number system, the value of a number is the sum of $n-1$ positive binary fractional bits and a sign bit which has a weight of -1:

$$A = -a_{n-1} + \sum_{i=0}^{n-2} a_i 2^{i-n+1} \qquad (1)$$

Examples of 4-bit fractional two's complement fractions are shown in Table 1. Two points are evident from the table: First, only a single representation of zero exists (specifically 0000) and second, the system is not symmetric because a negative number exists, −1, (1000), for which no positive equivalent exists. The latter property means that taking the absolute value of, negating or squaring a valid number (−1) can produce a result that cannot be represented.

Table 1. 4-Bit fractional two's complement numbers

Decimal Fraction	Binary Representation
+7/8	0111
+3/4	0110
+5/8	0101
+1/2	0100
+3/8	0011
+1/4	0010
+1/8	0001
+0	0000
−1/8	1111
−1/4	1110
−3/8	1101
−1/2	1100
−5/8	1011
−3/4	1010
−7/8	1001
−1	1000

Two's complement numbers are negated by inverting all bits and adding a ONE to the least significant bit position. For example, to form −3/8:

```
          +3/8              = 0011

          invert all bits   = 1100

          add 1               0001
                              ----
                              1101     = –3/8


Check:    invert all bits   = 0010

          add 1               0001
                              ----
                              0011     = 3/8
```

Truncation of two's complement numbers never increases the value of the number. The truncated numbers have values that are either unchanged or shifted toward negative infinity. This shift may be seen from Equation (1), in which any truncated digits come from the least significant end of the word and have positive weight. Thus, to remove them will either leave the value of the number unchanged (if the bits that are removed are ZEROs) or reduce the value of the number. On average, a shift toward $-\infty$ of one-half the value of the least significant bit exists. Summing many truncated numbers (which may occur in scientific, matrix, and signal processing applications) can cause a significant accumulated error.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

3/8	= 0011	
+ 1/2	= $\underline{0100}$	
	0111	= 7/8
– 3/8	= 1101	
– 1/2	= $\underline{1100}$	
Ignore Carry Out	(1) 1001	= –7/8
3/8	= 0011	
– 1/2	= $\underline{1100}$	
	1111	= –1/8
– 3/8	= 1101	
+ 1/2	= $\underline{0100}$	
Ignore Carry Out	(1) 0001	= 1/8

5/8	= 0101	
+ 1/2	= $\underline{0100}$	
MSB $C_{in} \neq C_{out}$	1001	= –7/8
– 5/8	= 1011	
– 1/2	= $\underline{1100}$	
MSB $C_{in} \neq C_{out}$	0111	= 7/8

ARITHMETIC ALGORITHMS

This section presents a reasonable assortment of typical fixed-point algorithms for addition, subtraction, multiplication, and division.

Addition

Addition is performed by summing the corresponding bits of the two n-bit numbers, which includes the sign bit. Subtraction is performed by summing the corresponding bits of the minuend and the two's complement of the subtrahend.

Overflow is detected in a two's complement adder by comparing the carry signals into and out of the most significant adder stage (i.e., the stage which computes the sign bit). If the carries differ, the addition has overflowed and the result is invalid. Alternatively, if the sign of the sum differs from the signs of the two operands, the addition has overflowed.

ADDITION IMPLEMENTATION

A wide range of implementations exist for fixed point addition, which include ripple carry adders, carry lookahead adders, and carry select adders. All start with a full adder as the basic building block.

Full Adder

The full adder is the fundamental building block of most arithmetic circuits. The operation of a full adder is defined by the truth table shown in Table 2. The sum and carry outputs are described by the following equations:

$$\mathrm{s_k} = \mathrm{a_k} \oplus \mathrm{b_k} \oplus \mathrm{c_k} \tag{2}$$

$$\mathrm{c_{k+1}} = \mathrm{a_k b_k} + \mathrm{a_k c_k} + \mathrm{b_k c_k} \tag{3}$$

where $\mathrm{a_k}$, $\mathrm{b_k}$, and $\mathrm{c_k}$ are the inputs to the k-th full adder stage, and s_k and c_{k+1} are the sum and carry outputs, respectively, and $\oplus$ denotes the Exclusive-OR logic operation.

In evaluating the relative complexity of implementations, often it is convenient to assume a nine-gate realization of the full adder, as shown in Fig. 1. For this implementation, the delay from either $\mathrm{a_k}$ or $\mathrm{b_k}$ to $\mathrm{s_k}$ is six gate delays and the delay from $\mathrm{c_k}$ to $\mathrm{c_{k+1}}$ is two gate delays. Some technologies, such as CMOS, form inverting gates (e.g., NAND and NOR gates) more efficiently than the noninverting gates that are assumed in this article. Circuits with equivalent speed and complexity can be constructed with inverting gates.

Ripple Carry Adder

A ripple carry adder for n-bit numbers is implemented by concatenating n full adders, as shown in Fig. 2. At the k-th bit position, bits a_k and b_k of operands A and B and the carry signal from the preceding adder stage, c_k, are used to generate the kth bit of the sum, s_k, and the carry, c_{k+1}, to

Table 2. Full adder truth table

Inputs			Outputs	
$\mathrm{a_k}$	$\mathrm{b_k}$	$\mathrm{c_k}$	$\mathrm{c_{k+1}}$	$\mathrm{s_k}$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

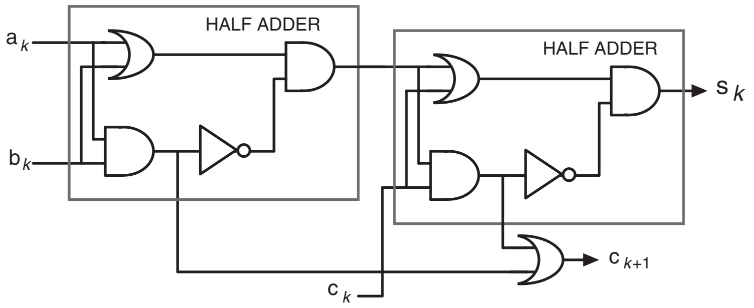


Figure 1. Nine gate full adder

.

the next adder stage. This adder is called a ripple carry adder, because the carry signals "ripple" from the least significant bit position to the most significant.

If the ripple carry adder is implemented by concatenating n of the nine gate full adders, which were shown in Fig. 1, an n-bit ripple carry adder requires $2n + 4$ gate delays to produce the most significant sum bit and $2n + 3$ gate delays to produce the carry output. A total of $9n$ logic gates are required to implement the n-bit ripple carry adder. In comparing the delay and complexity of adders, the delay from data input to most significant sum output denoted by DELAY and the gate count denoted by GATES will be used. These DELAY and GATES are subscripted by RCA to indicate ripple carry adder. Although these simple metrics are suitable for first-order comparisons, more accurate comparisons require more exact modeling because the implementations may be realized with transistor networks (as opposed to gates), which will have different delay and complexity characteristics.

$$\text{DELAY}_{\text{RCA}} = 2n + 4 \tag{4}$$

$$\text{GATES}_{\text{RCA}} = 9\text{n} \tag{5}$$

Carry Lookahead Adder

Another popular adder approach is the carry lookahead adder (1,2). Here, specialized logic computes the carries in parallel. The carry lookahead adder uses eight gate modified full adders that do not form a carry output for each bit position and lookahead modules, which form the carry outputs. The carry lookahead concept is best understood by rewriting Equation (3) with $\text{g}_k = \text{a}_k\text{b}_k$ and $\text{p}_k = \text{a}_k + \text{b}_k$.

$$\text{c}_{k+1} = \text{g}_k + \text{p}_k\text{c}_k \tag{6}$$

This equation helps to explain the concept of carry "generation" and "propagation": A bit position "generates" a carry regardless of whether there is a carry into that bit position if g_k is true (i.e., both a_k and b_k are ONEs), and a stage "propagates" an incoming carry to its output if p_k is true (i.e., either a_k or b_k is a ONE). The eight-gate modified full adder is based on the nine-gate full adder shown on Fig. 1. It has AND and OR gates that produce g_k and p_k with no additional complexity.

Extending Equation (6) to a second stage:

$$\begin{aligned} \text{c}_{k+2} &= \text{g}_{k+1} + \text{p}_{k+1}\text{c}_{k+1} \\ &= \text{g}_{k+1} + \text{p}_{k+1}(\text{g}_k + \text{p}_k\text{c}_k) \\ &= \text{g}_{k+1} + \text{p}_{k+1}\text{g}_k + \text{p}_{k+1}\text{p}_k\text{c}_k \end{aligned} \tag{7}$$

Equation (7) results from evaluating Equation (6) for the $(k + 1)$th stage and substituting c_{k+1} from Equation (6). Carry c_{k+2} exits from stage k+1 if: (*1*) a carry is generated there, (*2*) a carry is generated in stage k and propagates across stage k+1, or (*3*) a carry enters stage k and propagates across both stages k and k+1, etc. Extending to a third stage:

$$\begin{aligned} \text{c}_{k+3} &= \text{g}_{k+2} + \text{p}_{k+2}\text{c}_{k+2} \\ &= \text{g}_{k+2} + \text{p}_{k+2}(\text{g}_{k+1} + \text{p}_{k+1}\text{g}_k + \text{p}_{k+1}\text{p}_k\text{c}_k) \\ &= \text{g}_{k+2} + \text{p}_{k+2}\text{g}_{k+1} + \text{p}_{k+2}\text{p}_{k+1}\text{g}_k + \text{p}_{k+2}\text{p}_{k+1}\text{p}_k\text{c}_k \end{aligned} \tag{8}$$

Although it would be possible to continue this process indefinitely, each additional stage increases the fan-in (i.e., the number of inputs) of the logic gates. Four inputs [as required to implement Equation (8)] frequently are the maximum number of inputs per gate for current technologies. To continue the process, block generate and block propagate signals are defined over 4-bit blocks

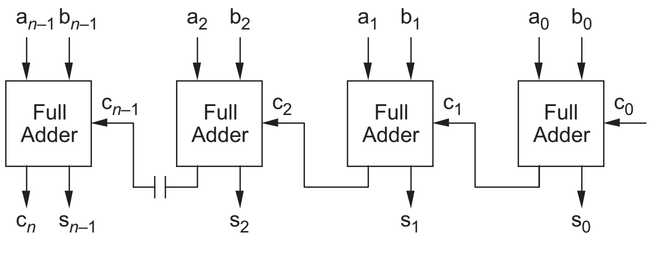


Figure 2. Ripple carry adder.

(stages k to k + 3), $g_{k:k+3}$ and $p_{k:k+3}$, respectively:

$$g_{k:k+3} = g_{k+3} + p_{k+3}g_{k+2} + p_{k+3}p_{k+2}g_{k+1} + p_{k+3}p_{k+2}p_{k+1}g_k \tag{9}$$

and

$$p_{k:k+3} = p_{k+3}p_{k+2}p_{k+1}p_k \tag{10}$$

Equation (6) can be expressed in terms of the 4-bit block generate and propagate signals:

$$c_{k+4} = g_{k:k+3} + p_{k:k+3}c_k \tag{11}$$

Thus, the carry out from a 4-bit wide block can be computed in only four gate delays [the first to compute p_i and g_i for i = k through k + 3, the second to evaluate $p_{k:k+3}$, the second and third to evaluate $g_{k:k+3}$, and the third and fourth to evaluate c_{k+4} using Equation (11)].

An n-bit carry lookahead adder requires $\lceil (n-1)/(r-1) \rceil$ lookahead logic blocks, where r is the width of the block. A 4-bit lookahead logic block is a direct implementation of Equations (6)–(10), with 14 logic gates. In general, an r-bit lookahead logic block requires $\frac{1}{2}(3r + r^2)$ logic gates. The Manchester carry chain (3) is an alternative switch-based technique to implement a lookahead logic block.

Figure 3 shows the interconnection of 16 adders and five lookahead logic blocks to realize a 16-bit carry lookahead adder. The events that occur during an add operation are:

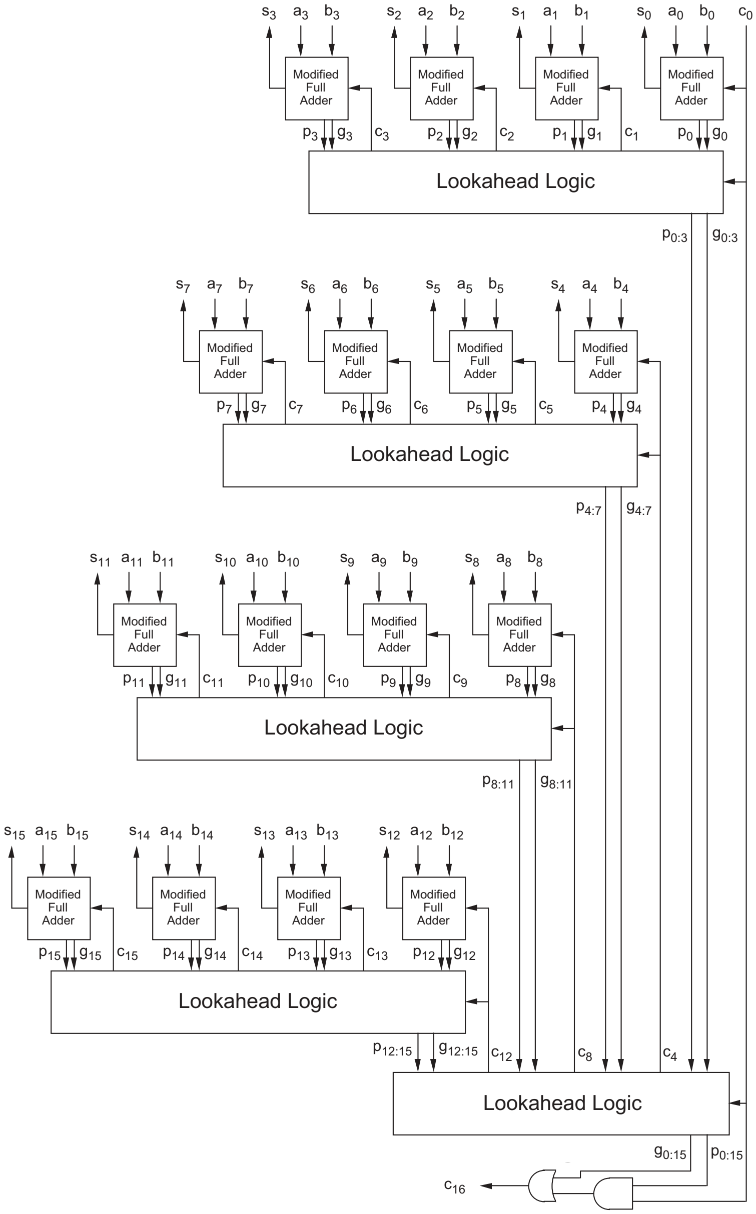


Figure 3. 16-bit carry lookahead adder.

(*1*) apply A, B, and carry in signals at time 0, (*2*) each modified full adder computes P and G, at time 1, (*3*) first level lookahead logic computes the 4-bit block propagate at time 2 and block generate signals by time 3, (*4*) second level lookahead logic computes c_4, c_8, and c_{12}, at time 5, (*5*) first level lookahead logic computes the individual carries at time 7, and (*6*) each modified full adder computes the sum outputs at time 10. This process may be extended to larger adders by subdividing the large adder into 16-bit blocks and by using additional levels of carry lookahead (e.g., a 64-bit adder requires three levels).

The delay of carry lookahead adders is evaluated by recognizing that an adder with a single level of carry lookahead (for r-bit words) has six gate delays, and that each additional level of lookahead increases the maximum word size by a factor of r and adds four gate delays. More generally, the number of lookahead levels for an n-bit adder is $\lceil \mathrm{Log}_r n \rceil$ where r is the "width" of the lookahead logic block (generally equal to the maximum number of inputs per logic gate). Because an r-bit carry lookahead adder has six gate delays and four additional gate delays exist per carry lookahead level after the first,

$$\mathrm{DELAY}_{\mathrm{CLA}} = 2 + 4\lceil \mathrm{Log}_r n \rceil \tag{12}$$

The complexity of an n-bit carry lookahead adder implemented with r-bit lookahead logic blocks is n modified full adders (each of which requires eight gates) and $\lceil (n-1)/(r-1) \rceil$ lookahead logic blocks (each of which requires $\frac{1}{2}(3r + r^2)$ gates). In addition, two gates are used to calculate the carry out from the adder, c_n, from $p_{0:n-1}$ and $g_{0:n-1}$.

$$\mathrm{GATES}_{\mathrm{CLA}} = 8\mathrm{n} + \tfrac{1}{2}(3\mathrm{r} + \mathrm{r}^2)\lceil (\mathrm{n}-1)/(\mathrm{r}-1) \rceil + 2 \tag{13}$$

If $r = 4$

$$\mathrm{GATES}_{\mathrm{CLA}} \approx 12\tfrac{2}{3}n - 2\tfrac{2}{3} \tag{14}$$

The carry lookahead approach reduces the delay of adders from increasing in proportion to the word size (as is the case for ripple carry adders) to increasing in proportion to the logarithm of the word size. As with ripple carry adders, the carry lookahead adder complexity grows linearly with the word size (for $r = 4$, the complexity of a carry lookahead adder is about 40% greater than the complexity of a ripple carry adder). It is important to realize that most carry lookahead adders require gates with up to 4 inputs, whereas ripple carry adders use only inverters and two input gates.

Carry Select Adder

The carry select adder divides the words to be added into blocks and forms two sums for each block in parallel (one with a carry in of ZERO and the other with a carry in of ONE). As shown for a 16-bit carry select adder in Fig. 4, the carry out from the previous block controls a multiplexer that selects the appropriate sum. The carry out is computed using Equation (11), because the block propagate signal is the carry out of an adder with a carry input of ONE, and the block generate signal is the carry out of an adder with a carry input of ZERO.

If a constant block width of k is used, $\lceil n/k \rceil$ blocks will exist and the delay to generate the sum is $2k + 3$ gate delays to form the carry out of the first block, two gate delays for each of the $\lceil n/k \rceil - 2$ intermediate blocks, and three gate delays (for the multiplexer) in the final block. To simplify the analysis, the ceiling function in the count of intermediate blocks is ignored. The total delay is thus

$$\mathrm{DELAY}_{\mathrm{C-SEL}} = 2k + 2n/k + 2 \tag{15}$$

where $\mathrm{DELAY}_{\mathrm{C-SEL}}$ is the total delay. The optimum block size is determined by taking the derivative of $\mathrm{DELAY}_{\mathrm{C-SEL}}$ with respect to k, setting it to zero, and solving for k. The result is

$$\mathrm{k} = \mathrm{n}^{.5} \tag{16}$$

$$\mathrm{DELAY}_{\mathrm{C-SEL}} = 2 + 4\mathrm{n}^{.5} \tag{17}$$

The complexity of the carry select adder is $2n - k$ ripple carry adder stages, the intermediate carry logic and $(\lceil n/k \rceil - 1)$ k-bit wide 2:1 multiplexers for the sum bits and one 1-bit wide multiplexer for the most significant carry output.

$$\mathrm{GATES}_{\mathrm{C-SEL}} = 21n - 12k + 3\lceil n/k \rceil - 2 \tag{18}$$

This result is somewhat more than twice the complexity of a ripple carry adder.

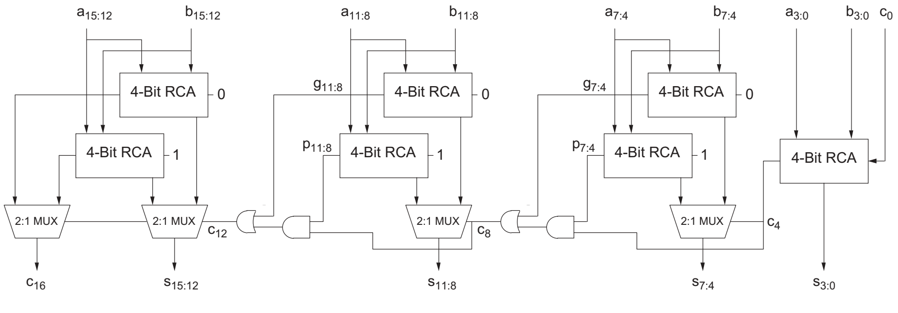


Figure 4. 16-bit carry select adder.

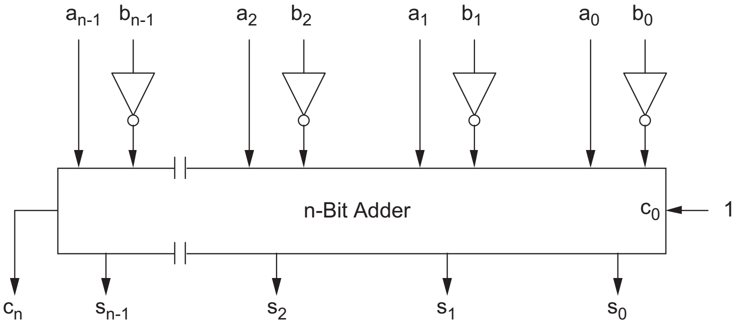


Figure 5. Two's complement subtracter.

Slightly better results can be obtained by varying the width of the blocks. The optimum is to make the two least significant blocks the same size and make each successively more significant block one bit larger than its predecessor. With four blocks, this gives an adder that is 3 bits wider than the conventional carry select adder.

SUBTRACTION

As noted previously, subtraction of two's complement numbers is accomplished by adding the minuend to the inverted bits of the subtrahend and adding a one at the least significant position. Figure 5 shows a two's complement subtracter that computes A – B. The inverters complement the bits of B; the formation of the two's complement is completed by setting the carry into the least significant adder stage to a ONE.

MULTIPLICATION

The bit product matrix of a 5-bit by 5-bit multiplier for unsigned operands is shown on Fig. 5. The two operands, A and B, are shown at the top, followed by n rows (each consisting of n bit products) that compose the bit product matrix. Finally, the product (2*n* bits wide) is at the bottom.

Several ways exist to implement a multiplier. One of the oldest techniques is to use an *n* bit wide adder to sum the rows of the bit product matrix in a row by row fashion. This process can be slow because $n - 1$ cycles (each long enough to complete an n bit addition) are required. If a ripple carry adder is used, the time to multiply two *n*-bit numbers is proportional to n^2. If a fast adder such as a carry lookahead adder is used, the time is proportional to $n \text{ Log}_2 (n)$.

Booth Multiplier

The Booth multiplier (4) and the modified Booth multiplier are attractive for two's complement multiplication, because they accept two's complement operands, produce a two's complement product, directly, and are easy to implement. The sequential Booth multiplier requires *n* cycles to form the product of a pair of *n*-bit numbers, where each cycle consists of an *n*-bit addition and a shift, an *n*-bit subtraction and a shift, or a shift without any other arithmetic operation. The radix-4 modified Booth multiplier (2) takes half as many cycles as the "standard" Booth multiplier, although the operations performed during each cycle are slightly more complex (because it is necessary to select one of five possible addends, namely, ±2B, ±B, or 0 instead of one of three).

Modified Booth Multiplier

To multiply A B, the radix-4 modified Booth multiplier [as described by MacSorley (2)] uses *n*/2 cycles where each cycle examines three adjacent bits of A, adds or subtracts 0, B, or 2B to the partial product and shifts the partial product two bits to the right. Figure 7 shows a flowchart for a radix-4 modified Booth multiplier. After an initialization step, there are *n*/2 passes through a loop where three bits of A are tested and the partial product P is modified. This algorithm takes half the number of cycles of the "standard" Booth multiplier (4), although the operations performed during a cycle are slightly more complex (because it is necessary to select one of four possible addends instead of one of two). Extensions to higher radices that examine more than three bits per cycle (5) are possible, but generally not attractive because the addition/subtraction operations involve nonpower of two

					b_4	b_3	b_2	b_1	b_0
					a_4	a_3	a_2	a_1	a_0
					$a_0 b_4$	$a_0 b_3$	$a_0 b_2$	$a_0 b_1$	$a_0 b_0$
				$a_1 b_4$	$a_1 b_3$	$a_1 b_2$	$a_1 b_1$	$a_1 b_0$	
			$a_2 b_4$	$a_2 b_3$	$a_2 b_2$	$a_2 b_1$	$a_2 b_0$		
		$a_3 b_4$	$a_3 b_3$	$a_3 b_2$	$a_3 b_1$	$a_3 b_0$			
	$a_4 b_4$	$a_4 b_3$	$a_4 b_2$	$a_4 b_1$	$a_4 b_0$				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Figure 6. 5-bit by 5-bit multiplier for unsigned operands.

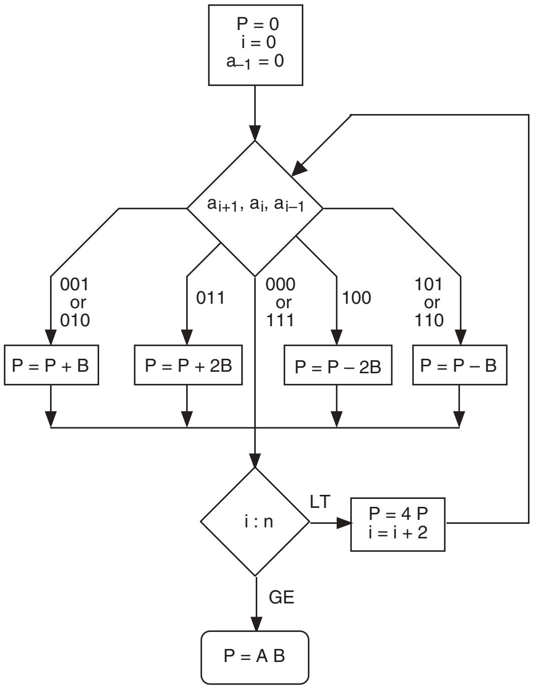


Figure 7. Flowchart of radix-4 modified booth multiplication.

multiples of B (such as 3B, 5B, etc.), which raises the complexity.

The delay of the radix-4 modified Booth multiplier is relatively high because an n-bit by n-bit multiplication requires $n/2$ cycles in which each cycle is long enough to perform an n-bit addition. It is low in complexity because it requires only a few registers, an n-bit 2:1 multiplexer, an n-bit adder/subtracter, and some simple control logic for its implementation.

Array Multipliers

A more hardware-intensive approach to multiplication involves the combinational generation of all bit products and their summation with an array of full adders. The bit product matrix of a 5-bit by 5-bit array multiplier for two's complement numbers (based on Ref. 6, p. 179) is shown in Fig. 8. It consists of a 5 by 5 array of bit product terms where most terms are of the form a_i AND b_j. The terms along the left edge and the bottom row are the complement of the normal terms (i.e., a_4 NAND b_0) as indicated by the over bar. The most significant term a_4b_4 is not complemented. Finally, ONEs are added at the sixth and tenth columns. In practice, the ONE at the tenth column is usually omitted.

Figure 9 shows an array multiplier that implements a 6-bit by 6-bit array multiplier realizing the algorithm shown on Fig. 8. It uses a 6 column by 6 row array of cells to form the bit products and do most of the summation and five adders (at the bottom of the array) to complete the evaluation of the product. Five types of cells are used in the square array: AND gate cells (marked G in Fig. 9) that form x_iy_j, NAND gate cells (marked NG) that form x_5 NAND y_j, half adder cells (marked HA) that sum the second input to the cell with x_iy_j, full adder cells (marked FA) that sum the second and third inputs to the cell with x_iy_j, and special full adder cells (marked NFA) that sum the second and third inputs to the cell with x_i NAND y_5. A special half adder, *HA, (that forms the sum of its two inputs and 1) and

$$
\begin{array}{cccccccccc}
 & & & & & b_4\ . & b_3 & b_2 & b_1 & b_0 \\
 & & & & & a_4\ . & a_3 & a_2 & a_1 & a_0 \\
\hline
 & & & & 1 & \overline{a_0 b_4} & a_0 b_3 & a_0 b_2 & a_0 b_1 & a_0 b_0 \\
 & & & & \overline{a_1 b_4} & a_1 b_3 & a_1 b_2 & a_1 b_1 & a_1 b_0 & \\
 & & & \overline{a_2 b_4} & a_2 b_3 & a_2 b_2 & a_2 b_1 & a_2 b_0 & & \\
 & & \overline{a_3 b_4} & a_3 b_3 & a_3 b_2 & a_3 b_1 & a_3 b_0 & & & \\
1 & a_4 b_4 & \overline{a_4 b_3} & \overline{a_4 b_2} & \overline{a_4 b_1} & \overline{a_4 b_0} & & & & \\
\hline
p_9 & p_8\ . & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

Figure 8. 5-bit by 5-bit multiplier for two's complement operands

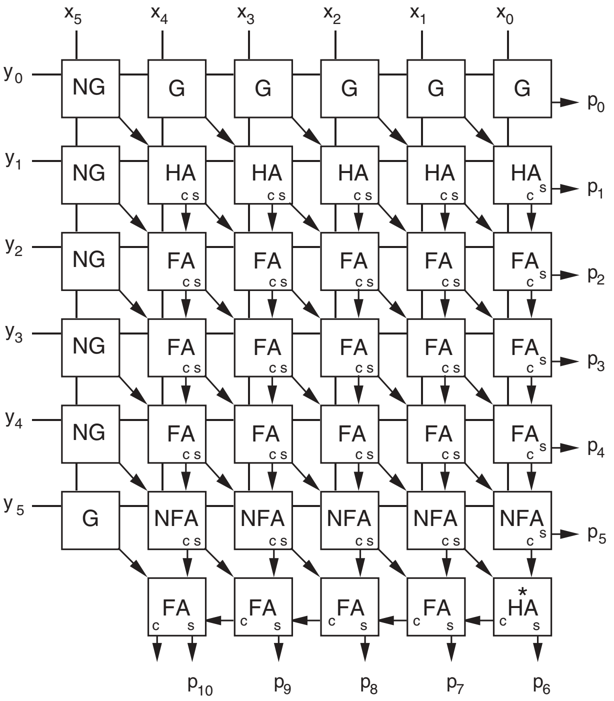


Figure 9. 6-bit by 6-bit two's complement array multiplier.

standard full adders are used in the five-cell strip at the bottom. The special half adder takes care of the extra 1 in the bit product matrix.

The delay of the array multiplier is evaluated by following the pathways from the inputs to the outputs. The longest path starts at the upper left corner, progresses to the lower right corner, and then progresses across the bottom to the lower left corner. If it is assumed that the delay from any adder input (for either half or full adders) to any adder output is k gate delays, then the total delay of an n-bit by n-bit array multiplier is:

$$\text{DELAY}_{\text{ARRAY MPY}} = k(2n - 2) + 1 \qquad (19)$$

The complexity of the array multiplier is n^2 AND and NAND gates, n half adders (one of which is a *half adder), and $n^2 - 2n$ full adders. If a half adder is realized with four gates and a full adder with nine gates, the total complexity of an n-bit by n-bit array multiplier is

$$\text{GATES}_{\text{ARRAY MPY}} = 10n^2 - 14n \qquad (20)$$

Array multipliers are laid out easily in a cellular fashion which makes them attractive for VLSI implementation, where minimizing the design effort may be more important than maximizing the speed.

Wallace/Dadda Fast Multiplier

A method for fast multiplication was developed by Wallace (7) and was refined by Dadda (8). With this method, a three-step process is used to multiply two numbers: (*1*) The bit products are formed, (*2*) the bit product matrix is "reduced" to a two row matrix whose sum equals the sum of the bit products, and (*3*) the two numbers are summed with a fast adder to produce the product. Although this may seem to be a complex process, it yields multipliers with delay proportional to the logarithm of the operand word size, which is "faster" than the array multiplier, which has delay proportional to the word size.

The second step in the fast multiplication process is shown for a 6-bit by 6-bit Dadda multiplier on Fig. 10. An input 6 by 6 matrix of dots (each dot represents a bit product) is shown as matrix 0. "Regular dots" are formed with an AND gate, and dots with an over bar are formed with a NAND gate. Columns that have more than four dots (or that will grow to more than four dots because of carries) are reduced by the use of half adders (each half adder takes in two dots and outputs one in the same column and one in the next more significant column) and full adders (each full adder takes in three dots from a column and outputs one in the same column and one in the next more significant column) so that no column in matrix 1 will have more than four dots. Half adders are shown by two dots connected by a "crossed" line in the succeeding matrix and full adders are shown by two dots connected by a line in the succeeding matrix. In each case, the right-most dot of the pair that are connected by a line is in the column from which the inputs were taken in the preceding matrix for the adder. A special half adder (that forms the sum of its two inputs and 1) is shown with a doubly crossed line. In the succeeding steps reduction to matrix 2, with no more than three dots per column, and finally matrix 3, with no more than two dots per column, is performed. The reduction shown on Fig. 10 (which requires three full adder delays) is followed by an 10-bit carry propagating adder. Traditionally, the carry propagating adder is realized with a carry lookahead adder.

The height of the matrices is determined by working back from the final (two row) matrix and limiting the height of each matrix to the largest integer that is no more than 1.5

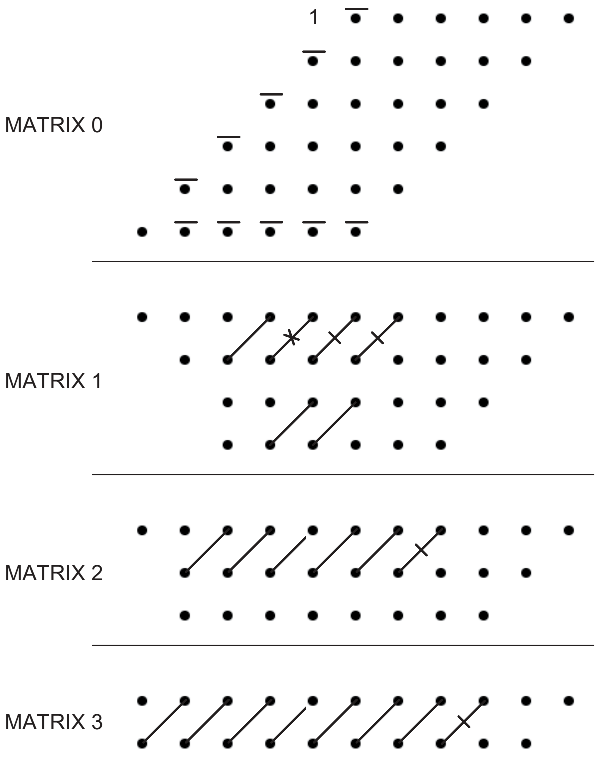


Figure 10. 6-bit by 6-bit two's complement Dadda multiplier.

times the height of its successor. Each matrix is produced from its predecessor in one adder delay. Because the number of matrices is related logarithmically to the number of rows in matrix 0, which is equal to the number of bits in the words to be multiplied, the delay of the matrix reduction process is proportional to log n. Because the adder that reduces the final two row matrix can be implemented as a carry lookahead adder (which also has logarithmic delay), the total delay for this multiplier is proportional to the logarithm of the word size.

The delay of a Dadda multiplier is evaluated by following the pathways from the inputs to the outputs. The longest path starts at the center column of bit products (which require one gate delay to be formed), progresses through the successive reduction matrices (which requires approximately $\text{Log}_{1.44}(n)$ full adder delays) and finally through the $2n-2$-bit carry propagate adder. If the delay from any adder input (for either half or full adders) to any adder output is k gate delays, and if the carry propagate adder is realized with a carry lookahead adder implemented with 4-bit lookahead logic blocks (with delay given by Equation (12), the total delay (in gate delays) of an n-bit by n-bit Dadda multiplier is:

$$\text{DELAY}_{\text{DADDA MPY}} = 1 + k\,\text{LOG}_{1.44}(n) + 2 + 4\lceil \text{Log}_r(2n-2)\rceil \qquad (21)$$

The complexity of a Dadda multiplier is determined by evaluating the complexity of its parts. n^2 gates ($2n-2$ are NAND gates, the rest are AND gates) to form the bit product matrix exist, $(\text{n}-2)^2$ full adders, $n-1$ half adders and one special half adder for the matrix reduction and a $2n-2$-bit carry propagate adder for the addition of the final two row matrix. If the carry propagate adder is realized with a carry lookahead adder (implemented with 4-bit lookahead logic blocks), and if the complexity of a full adder is nine gates and the complexity of a half adder (either regular or special) is four gates, then the total complexity is:

$$\text{GATES}_{\text{DADDA MPY}} = 10n^2 - 6\,\frac{2}{3}n - 26 \qquad (22)$$

The Wallace tree multiplier is very similar to the Dadda multiplier, except that it does more reduction in the first stages of the reduction process, it uses more half adders, and it uses a slightly smaller carry propagating adder. A dot diagram for a 6-bit by 6-bit Wallace tree multiplier for two's complement operands is shown on Fig. 11. This reduction (which requires three full adder delays) is followed by an 8-bit carry propagating adder. The total complexity of the Wallace tree multiplier is a bit greater than the total complexity of the Dadda multiplier. In most cases, the Wallace and Dadda multipliers have about the same delay.

DIVISION

Two types of division algorithms are in common use: digit recurrence and convergence methods. The digit recurrence approach computes the quotient on a digit-by-digit basis,

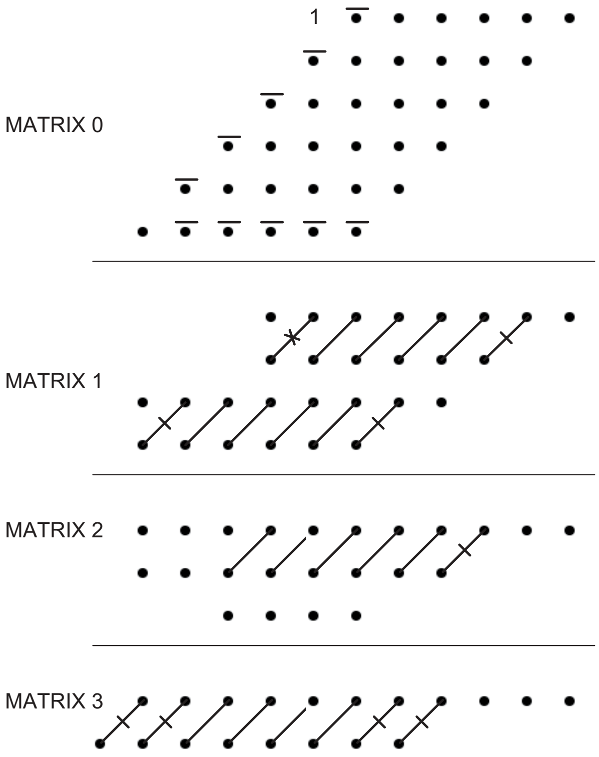


Figure 11. 6-bit by 6-bit Two's Complement Wallace Multiplier.

hence they have a delay proportional to the precision of the quotient. In contrast, the convergence methods compute an approximation that converges to the value of the quotient. For the common algorithms the convergence is quadratic, which means that the number of accurate bits approximately doubles on each iteration.

The digit recurrence methods that use a sequence of shift, add or subtract, and compare operations are relatively simple to implement. On the other hand, the convergence methods use multiplication on each cycle. This fact means higher hardware complexity, but if a fast multiplier is available, potentially a higher speed may result.

Digit Recurrent Division

The digit recurrent algorithms (9) are based on selecting digits of the quotient Q (where Q = N/D) to satisfy the following equation:

$$P_{k+1} = rP_k - q_{n-k-1}D \text{ for } k = 1, 2, \ldots, n-1 \qquad (23)$$

where P_k is the partial remainder after the selection of the kth quotient digit, $P_0 = N$ (subject to the constraint $|P_0| < |D|$), r is the radix, q_{n-k-1} is the kth quotient digit to the right of the binary point, and D is the divisor. In this subsection, it is assumed that both N and D are positive, see Ref. 10 for details on handling the general case.

Binary SRT Divider

The binary SRT division process (also known as radix-2 SRT division) selects the quotient from three candidate quotient digits {±1, 0}. The divisor is restricted to $.5 \leq D < 1$. A flowchart of the basic binary SRT scheme is shown in Fig. 12. Block 1 initializes the algorithm. In step 3, 2 P_k and the divisor are used to select the quotient digit. In step 4, $P_{k+1} = 2\ P_k - q\ D$. Step 5 tests whether all bits of the quotient have been formed and goes to step 2 if more need to be computed. Each pass through steps 2–5 forms one digit of the quotient. The result upon exiting from step 5 is a collection of n signed binary digits.

Step 6 converts the n digit signed digit number into an n-bit two's complement number by subtracting, N, which has a 1 for each bit position where $q_i = -1$ and 0 elsewhere from, P, which has a 1 for each bit position where $q_i = 1$ and 0 elsewhere. For example:

Q = 0 . 1 1 –1 0 1 = 21/32

P = 0 . 1 1 0 0 1

N = 0 . 0 0 1 0 0

Q = 0 . 1 1 0 0 1 – 0 . 0 0 1 0 0

Q = 0 . 1 1 0 0 1 + 1 . 1 1 1 0 0

Q = 0 . 1 0 1 0 1 = 21/32

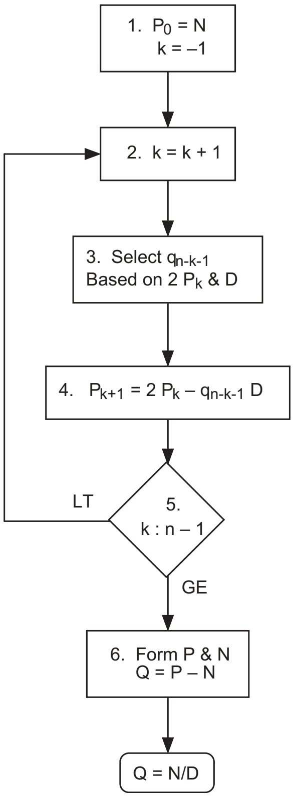


Figure 12. Flowchart of binary SRT division.

The selection of the quotient digit can be visualized with a P-D Plot such as the one shown in Fig. 13. The plot shows the divisor along the x axis and the shifted partial remainder (in this case 2 $\mathrm{P_k}$) along the y-axis. In the area where 0.5 $\le$ D $<$ 1, values of the quotient digit are shown as a function of the value of the shifted partial remainder. In this case, the relations are especially simple. The digit selection and resulting partial remainder are given for the k-th iteration by the following relations:

$$\text{If } \mathrm{P}_k > .5, \quad \mathrm{q}_{n-k-1} = 1 \quad \text{and } \mathrm{P}_{k+1} = 2\mathrm{P}_k - \mathrm{D} \tag{24}$$

$$\text{If } -.5 < P_k < .5, \quad q_{n-k-1} = 0 \quad \text{and } \mathrm{P}_{k+1} = 2P_k \tag{25}$$

$$\text{If } P_k \le -.5, \quad q_{n-k-1} = -1 \text{ and } P_{k+1} = 2P_k + \mathrm{D} \tag{26}$$

Computing an n-bit quotient will involve selecting n quotient digits and up to $n + 1$ additions.

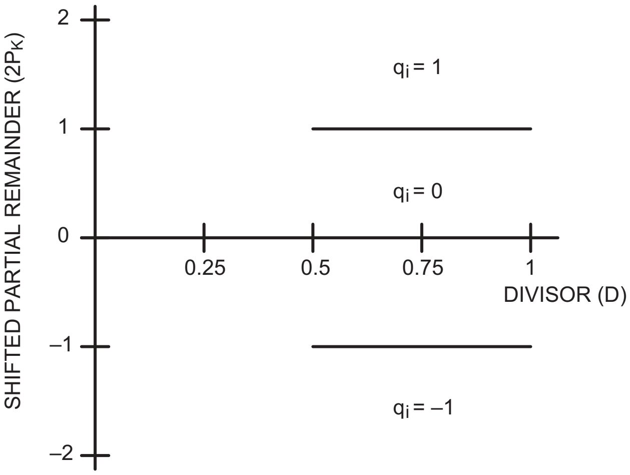


Figure 13. P-D plot for binary SRT division.

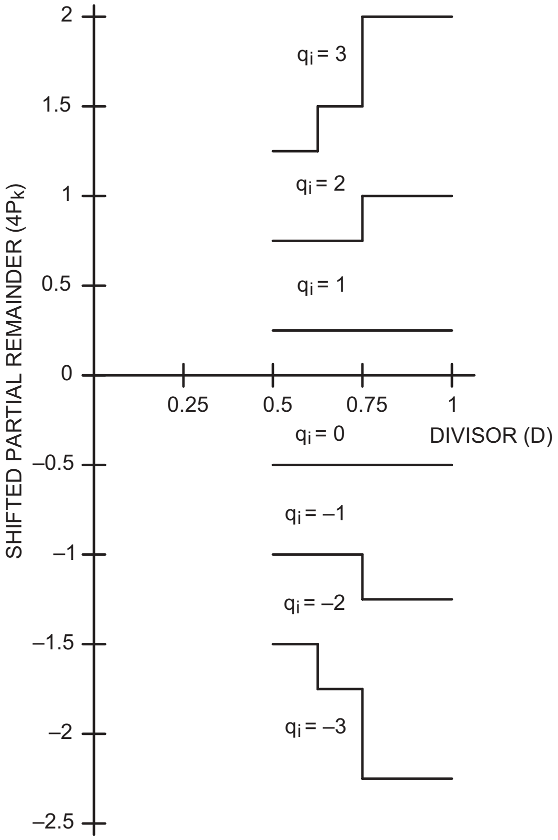


Figure 14. P-D Plot for radix-4 maximally redundant SRT division.

Radix-4 SRT Divider

The higher radix SRT division process is similar to the binary SRT algorithms. Radix 4 is the most common higher radix SRT division algorithm with either a minimally redundant digit set of {±2, ±1, 0} or the maximally redundant digit set of {±3, ±2, ±1, 0}. The operation of the algorithm is similar to the binary SRT algorithm shown on Fig. 12, except that in step 3, 4 P_k, and D are used to determine the quotient digit. A P-D Plot is shown on Fig. 14 for the maximum redundancy version of radix-4 SRT division. Seven values are possible for the quotient digit at each stage. The test for completion in step 5 becomes $k : \frac{n}{2} - 1$. Also the conversion to two's complement in step 6 is modified slightly because each quotient digit provides two bits of the P and N numbers that are used to form the two's complement number.

Newton–Raphson Divider

The second category of division techniques uses a multiplication-based iteration to compute a quadratically convergent approximation to the quotient. In systems that include a fast multiplier, this process may be faster than the digit recurrent methods. One popular approach is the Newton-Raphson algorithm that computes an approximation to the reciprocal of the divisor that is then multiplied by the dividend to produce the quotient. The process to compute Q = N/D consists of three steps:

1. Calculate a starting estimate of the reciprocal of the divisor, $R_{(0)}$. If the divisor, D, is normalized (i.e., $\frac{1}{2} \leq D < 1$), then $R_{(0)} = 3 - 2D$ exactly computes 1/ D at D = .5 and D = 1 and exhibits maximum error (of approximately 0.17) at $D = \frac{1}{2}^{.5}$. Adjusting $R_{(0)}$ downward to by half the maximum error gives

$$R_{(0)} = 2.915 - 2D \tag{27}$$

   This produces an initial estimate, that is within about 0.087 of the correct value for all points in the interval $\frac{1}{2} \leq D < 1$.

2. Compute successively more accurate estimates of the reciprocal by the following iterative procedure:

$$R_{(i+1)} = R_{(i)}(2 - D\,R_{(i)}) \text{ for } i = 0, 1, \ldots, k \tag{28}$$

3. Compute the quotient by multiplying the dividend times the reciprocal of the divisor.

$$Q = N\,R_{(k)} \tag{29}$$

where i is the iteration count and N is the numerator. Figure 15 illustrates the operation of the Newton–Raphson algorithm. For this example, three iterations (which involve a total of four subtractions and

$A = .625$
$B = .75$

$R_{(0)} = 2.915 - 2 \cdot B$ 1 Subtract
$= 2.915 - 2 \cdot .75$

$R_{(0)} = 1.415$

$R_{(1)} = R_{(0)}\,(2 - B \cdot R_{(0)})$ 2 Multiplies, 1 Subtract
$= 1.415\,(2 - .75 \cdot 1.415)$
$= 1.415 \cdot .95875$
$R_{(1)} = 1.32833125$

$R_{(2)} = R_{(1)}\,(2 - B \cdot R_{(1)})$ 2 Multiplies, 1 Subtract
$= 1.32833125\,(2 - .75 \cdot 1.32833125)$
$= 1.32833125 \cdot 1.00375156$
$R_{(2)} = 1.3333145677$

$R_{(3)} = R_{(2)}\,(2 - B \cdot R_{(2)})$ 2 Multiplies, 1 Subtract
$= 1.3333145677\,(2 - .75 \cdot 1.3333145677)$
$= 1.3333145677 \cdot 1.00001407$
$R_{(3)} = 1.3333333331$

$Q = A \cdot R_{(3)}$ 1 Multiply
$= .625 \cdot 1.3333333331$
$Q = .83333333319$

Figure 15. Example of Newton–Raphson division.

seven multiplications) produces an answer accurate to nine decimal digits (approximately 30 bits).

With this algorithm, the error decreases quadratically so that the number of correct bits in each approximation is roughly twice the number of correct bits on the previous iteration. Thus, from a 3.5-bit initial approximation, two iterations produce a reciprocal estimate accurate to 14-bits, four iterations produce a reciprocal estimate accurate to 56-bits, and so on.

The efficiency of this process is dependent on the availability of a fast multiplier, because each iteration of Equation (28) requires two multiplications and a subtraction. The complete process for the initial estimate, three iterations, and the final quotient determination requires four subtraction operations and seven multiplication operations to produce a 16-bit quotient. This process is faster than a conventional nonrestoring divider if multiplication is roughly as fast as addition, which is a condition that is satisfied for some systems that include a hardware multiplier.

CONCLUSIONS

This article has presented an overview of the two's complement number system and algorithms for the basic fixed-point arithmetic operations of addition, subtraction, multiplication, and division. When implementing arithmetic units, often an opportunity exists to optimize the performance and the complexity to match the requirements of the specific application. In general, faster algorithms require more area and power; often it is desirable to use the fastest algorithm that will fit the available area and power bugdets.

BIBLIOGRAPHY

1. A. Weinberger and J. L. Smith, A logic for high-speed addition, *National Bureau of Standards Circular*, **591**: 3–12, 1958.
2. O. L. MacSorley, High-speed arithmetic in binary computers, *Proceedings of the IRE*, **49**: 67–91, 1961.
3. T. Kilburn, D. B. G. Edwards, and D. Aspinall, A parallel arithmetic unit using a saturated transistor fast-carry circuit, *Proceedings of the IEEE*, Part B, **107**: 573–584, 1960.
4. A. D. Booth, A signed binary multiplication technique, *Quarterly J. Mechanics Appl. Mathemat.*, **4**: 236–240, 1951.
5. H. Sam and A. Gupta, A generalized multibit recoding of two's complement binary numbers and its proof with application in multiplier implementations, *IEEE Trans. Comput.*, **39**: 1006–1015, 1990.
6. B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, New York: Oxford University Press, 2000.
7. C. S. Wallace, A suggestion for a fast multiplier, *IEEE Trans. Electron. Comput*, **13**: 14–17, 1964.
8. L. Dadda, Some schemes for parallel multipliers, *Alta Frequenza*, **34**: 349–356, 1965.

9. J. E. Robertson, A new class of digital division methods, *IEEE Trans. Electr. Comput*, **7**: 218–222, 1958.
10. M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Their Implementations*, Boston, MA: Kluwer Academic Publishers, 1994.
11. *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE Std 754–1985, Reaffirmed 1990.
12. M. D. Ercegovac and T. Lang, *Digital Arithmetic*, San Francisco, CA: Morgan Kaufmann Publishers, 2004.
13. I. Koren, *Computer Arithmetic Algorithms*, 2nd Edition, Natick, MA: A. K. Peters, 2002.

EARL E. SWARTZLANDER, JR.
University of Texas at Austin
Austin, Texas

FLOATING-POINT COMPUTER ARITHMETIC

INTRODUCTION

The field of floating-point computer arithmetic is a subsection of computer engineering that concentrates on the development and execution of arithmetic in microprocessors. The floating-point sections of processor chips and coprocessors are the units in a microprocessor that perform most of the arithmetic operations for applications such as 3D graphics, multimedia, signal processing, Fourier transforms, etc.

The floating-point format is a form of scientific notation. A floating-point number may represent a vast range of real numbers, with values ranging from nearly infinitely large to nearly infinitely small, all in a finite bit-width. Although many floating-point formats have been used (1), this chapter is based on the IEEE-754 Standard (2) that is widely used at the current time.

This article begins with a brief review of the IEEE-754 floating-point format in the section on floating-point number systems. The next section provides implementation examples of the fundamental floating-point execution units: the floating-point adder and the floating-point multiplier. Other floating-point mathematical operations are derived from these two instructions, so most of the same hardware is used. Finally, the section on other IEEE-754 specifications covers miscellaneous other aspects of the IEEE-754 specifications, such as convert instructions, denormalized numbers, and the representation of special "numbers" infinity and zero.

FLOATING-POINT NUMBER SYSTEMS

The IEEE-754 standard sets down specific rules and formats for binary floating-point arithmetic. A processor that follows every rule in the specification is considered "IEEE-754 compliant." Although some specialized machines [such as DirectX graphics cards (3), CELL processors (4), etc.] are not fully compliant with all parts of the standard, the fundamentals of the specification are essentially universal and should be understood by any engineer or scientist developing a floating-point application.

The IEEE-754 standard defines a format that consists of three parts: a sign bit, a significand, and a biased exponent. Four standard formats are defined: single precision (stored as a 32-bit number), single extended (often realized as double precision), double precision (stored as a 64-bit number), and double extended (stored as greater than 78 bits). The significand, which is a mixed number with a leading 1 for its integer part, takes the place of the fractional mantissa or the integer coefficient of previous floating-point formats. The significand is a fixed point magnitude in the range $1 \leq \text{significand} < 2$. If the sign = 1, then the number is negative; if the sign = 0, then the number is positive. For single-precision numbers, the significand is a 24-bit magnitude, for double-precision numbers it is a 53-bit magnitude.

The biased exponent is an excess-127 integer for single precision and excess-1023 for double precision. The use of the word "biased" means that the stored integer value in the exponent bits actually represents that integer number minus the bias. The rationale behind biased exponents is to allow an equal range of positive and negative exponents (i.e., '127' is the midpoint of the linear range in an 8-bit unsigned binary field. Single-precision stored numbers above this bias represent positive exponents and those below are negative exponents).

With all bits combined, the value of a number, A, is given by:

$$A = (-1)^{\text{sign A}} \times \text{significand}_{\text{A}} \times 2^{\text{exponent A}-\text{bias}} \tag{1}$$

Numbers stored by the processor are "packed" by first normalizing the significand, which is done by removing the integer 1. The remaining fraction bits, the sign bit, and the biased exponent bits are then packed together into a single entity that is stored. This use of normalized significand packing and the removal of the "hidden one" for storage provide one extra (free) bit of precision.

Following the standard, "single precision" numbers that consist of 23 fraction bits, 8 biased exponent bits, and 1 sign bit are stored in a 32-bit register or memory location. The implicit "1" from the significand is not stored. Double-precision numbers that consist of 52 fraction bits, 11 biased exponent bits, and 1 sign bit are stored in a 64-bit register or memory location. Figure 1 shows the bit partitioning of the *stored* binary words.

To clarify the formats, consider the following example: The decimal number 7.25 is to be saved in memory as a single-precision number. As a fixed point number, it would be represented in binary as:

$$7.25 = 111;01$$

The ";" is used throughout this chapter to represent the binary point (i.e., the numeric split between the integers and fractions) in binary numbers. Meanwhile, the same decimal number stored as a single-precision floating-point number is:

S	Exp								Fraction																						
0	1	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Sign = 0 (sign = 0 means the number is positive)

Biased Exponent = 1000_0001 = 129

Fraction = 1101_0000_0000_0000_0000_000 (as stored in memory)

Significand = 1;1101_0000_0000_0000_0000_000 (as used in the processor)

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

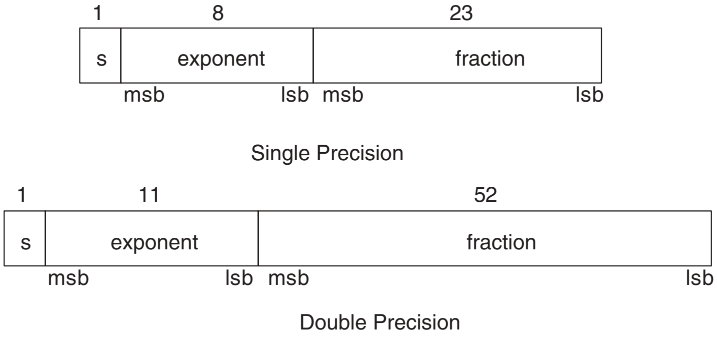


Figure 1. The IEEE-754 single and double precision floating-point data types (2).

Therefore

$$7.25 = (-1)^0 \times 2^{129-127} \times (1;1101) = 1;1101 \times 2^2$$
$$= 111;01 \times 2^0 = 111;01$$

Notice that the position of the binary point is a function of the biased exponent value, hence the term "floating-point." It should be noted that negative numbers in the IEEE floating-point format are identical to positive numbers except for the sign bit of '1.'

Arithmetic in Floating-Point Format

The IEEE-754 standard enumerates the required arithmetic functions that must be supported in a floating-point machine if it is to be identified as compliant to the standard. The following operations are required: add; subtract; multiply; divide; square root; remainder; round to integer in floating-point format; convert between floating-point formats; convert between floating-point and integer; convert binary to decimal; and compare. The operations may be implemented in hardware, software, or a combination of the two (2).

In practice, the key to floating-point arithmetic is simply keeping track of the various binary-point positions. For instance, floating-point addition requires that the binary points of each input number are normalized to the same exponential value before addition may occur. This task effectively aligns the binary points. Determining which direction and to what extent a fractional value must shift for this normalization is performed by a simple difference of biased exponents. In multiplications, the biased exponents must additively combine to produce the correct binary-point position. Each case is shown in examples below.

Floating-Point Addition Example

As an example, consider the addition of two single-precision floating-point numbers. Each number is stored in a 32-bit word in memory. The addition of the decimal numbers 7.25 and 1.75 is to be performed. The two numbers are stored in memory as shown in Fig. 2.

As the numbers are brought into the processor, they are "unpacked" revealing the hidden '1' bits:

$$1;1101\_0000\_0000\_0000\_0000\_000 \times 2^{129-127} = 1;1101 \times 2^2$$
$$1;1100\_0000\_0000\_0000\_0000\_000 \times 2^{127-127} = 1;1100 \times 2^0$$

Currently, the numbers cannot be directly added, as their exponents differ. The significand of the number with the smaller exponent needs to shifted down by two bit positions so that both numbers to have the same exponent. After alignment, the numbers may be correctly added:

$$\begin{array}{r} 1;1101\_0000\_0000\_0000\_0000\_000 \times 2^{129-127} = \ 1;1101 \times 2^2 \\ +\,0;0111\_0000\_0000\_0000\_0000\_000 \times 2^{129-127} = \ 0;0111 \times 2^2 \\ \hline 10;0100\_0000\_0000\_0000\_0000\_000 \times 2^{129-127} = 10;0100 \times 2^2 \end{array}$$

S			Exp																Fraction												
0	1	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a) Representation of 7.25

S			Exp																Fraction												
0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) Representation of 1.75

Figure 2. The addends as stored in the memory.

S			Exp																Fraction												
0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3. The sum (= 9) as stored in the memory.

Before this result is stored back in the memory, the number must be "packed," by shifting the significand so that it is a mixed number with the integer portion being a 1. In this example, the addition caused an overflow, so the exponent must be incremented and the significand normalized:

$$10;0100\_0000\_0000\_0000\_0000\_000 \times 2^{129-127}$$
$$= 10;0100 \times 2^{2}$$

is normalized to

$$1;0010\_0000\_0000\_0000\_0000\_000 \times 2^{130-127}$$
$$= 1;0010 \times 2^{3}$$

The result is stored in the memory as observed in Fig. 3.

Floating-Point Multiplication Example

The only difference between fixed-point and floating-point multiplication is that the exponent values need to be combined. The significands of the floating-point numbers are multiplied as described in the chapter on fixed-point computer arithmetic.

Assume the same numerical values that were used for the adder example. The multiplication of the decimal numbers 7.25 and 1.75 is to be performed. The significands of each operand are multiplied:

$$\begin{array}{r} 1;1101\_0000\_0000\_0000\_0000\_000 \\ \times\ 1;1100\_0000\_0000\_0000\_0000\_000 \\ \hline 11;0010\_1100\_0000\_0000\_0000\_0000\_ \\ 0000\_0000\_0000\_0000\_0000\_00 \end{array}$$

The exponent values are summed and the bias value is adjusted. For the numerical example:

$$\begin{array}{r} 2^{129-127} = 2^{2} \\ \times\ 2^{127-127} = 2^{0} \\ \hline 2^{129+127-2(\text{BIAS})} = 2^{2} \end{array}$$

At this point, both the exponent and the significand have been multiplied. Because the significand requires no rounding, all that remains is to normalize the result to the correct format before passing it to memory.

$$11;0010\_1100\_0000\_0000\_0000\_000$$
$$\times 2^{\,129-127} = 11;0010\_1100 \times 2^{2}$$

is normalized to

$$1;1001\_0110\_0000\_0000\_0000\_000$$
$$\times 2^{130-127} \quad = \quad 1;1001\_0110 \times 2^{3}$$

The result is stored as shown in Fig. 4.

Floating-Point Rounding

If the significand of a calculation has too many bits for the given format (i.e., single precision, single extended precision, or double precision) the result must be rounded to the appropriate number of bits. The IEEE-754 standard requires the support of four rounding modes. These rounding modes provide numerical precision flexibility for application programmers.

In hardware designs that conform to this requirement, rounding is performed at the end of an execution block. IEEE-754 compliance in rounding is generally considered to be one of the more difficult and expensive requirements of the floating-point standard. This requirement is so hardware intensive that it is commonly the first rule ignored in applications that do not conform entirely to the IEEE-754 standard.

For instance, DirectX (3) compliant graphics cards allow a greater margin of error in their rounding, specifying that results must be within 1 least significant bit of an IEEE-754 compliant CPU. Although this change seems small, the hardware reductions are significant. Additionally, it is common in noncompliant implementations to ignore rounding in the case of massive cancellation (a special case in subtraction where two operands are nearly identical), as the hardware cost of maintaining full precision is too expensive.

The subject of rounding is too large to explain fully in a short article. This section only covers the basics and provides a simple example. The subject of floating-point rounding methods is quite important, and a more thorough explanation of corner cases, requirements, and methods may be found in Refs. 5-8.

The Four IEEE-754 Rounding Modes. The four IEEE-754 rounding modes are: round to nearest-even (RNE), round toward positive infinity (RPI), round toward negative infinity (RNI), and round toward zero (RTZ). The mathematical behavior required from each of the four rounding modes may be observed in Figs. 5–8. The figures show the real numerical number range centered around zero. Each tick of the graph represents a numerical value that may be represented by the result precision range.

S			Exp																Fraction												
0	1	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4. The product (= 12.6875) as stored in the memory.

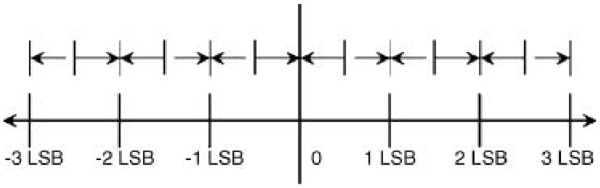


Figure 5. Round to nearest-even (RNE).

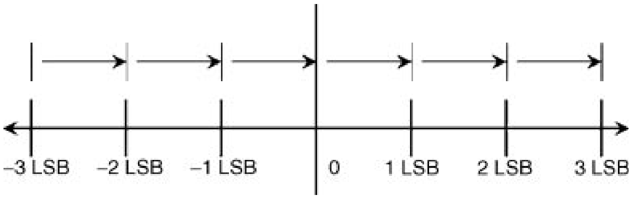


Figure 6. Round toward positive infinity (RPI).

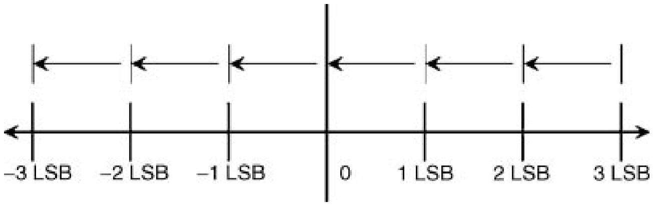


Figure 7. Round toward negative infinity (RNI).

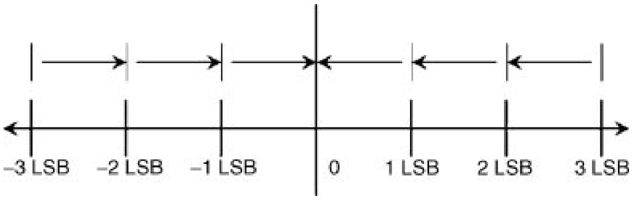


Figure 8. Round toward zero (RTZ).

The arrows represent the direction that all numbers between the possible number representations round to in the different modes. Finally, rounding methods and precisions generally refer to actions relative to the binary unit in the last place (ULP) also known as the least significant bit (LSB).

The most complex of the rounding schemes is round to nearest-even. The rules of RNE are as follows:

1. If the sum of the bits beyond the LSB is less than LSB/2, then truncate the bits.
2. If the sum of the bits beyond the LSB is more than LSB/2, then increment the LSB.
3. If the sum of the bits beyond the LSB is exactly LSB/2, then round to the nearest even number.

The round to infinity schemes, RPI and RNI, each either require an increment of the result to round-up or require a truncation should data exist beyond the desired precision. The decision to round-up or to truncate is based on which direction to infinity is specified and the sign of the number. For RPI, positive data that exceeds the desired precision require an increment of the least significant bit, whereas negative data only require truncation. For RNI, positive data require truncation whereas negative data require an increment of the least significant bit.

The simplest of the schemes, round to zero, is achieved by truncating all data beyond the range of the output result. Once a floating-point result is calculated and normalized back into the correct format, any bits that exceed the precision specified by the application are simply discarded. Positive results see the equivalent of a round-down, and negative results see the equivalent of a round-up after truncation.

The Rounding Bits. To perform the rounding required by the IEEE-754 standard, a series of bits called the "rounding bits" are commonly used. These bits include the "guard bit" (G), the "round bit" (R), the "carry bit" (C), and the "sticky bit" (S). Because these bits are not explicitly identified in the IEEE-754 specification, the naming conventions vary, but the concepts remain the same.

For floating-point addition and related instructions, the guard, round, and sticky bits are used for rounding. These three bits are typically included in order at the end of an intermediate result. Figure 9 shows the guard bit, G, is in the bit position directly following the LSB of the intermediate result. The round bit, R, is adjacent to the guard bit, and the sticky bit, S, completes the set.

The first bit to understand is the sticky bit. This bit is a logical OR of all bits in its position and smaller. Figure 10 shows formation of the sticky bit. If the sticky bit is a 0, then the intermediate result up to the round bit is exact.

The guard and round bits are primarily included to help determine what action to perform in the RNE mode. Specifically, in a normal addition, if G = 1, then the result has

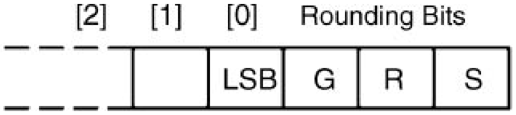


Figure 9. Rounding bits for floating-point addition.

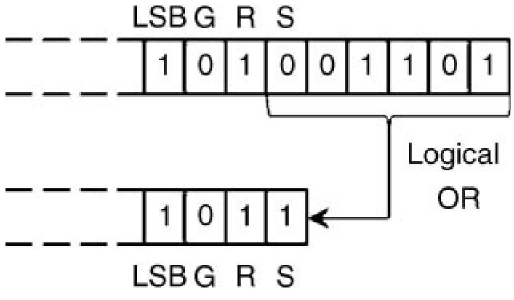


Figure 10. Creation of the sticky bit.

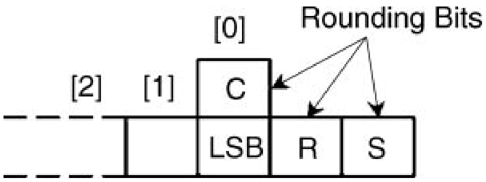


Figure 11. Floating-point multiplication rounding bits.

the value of LSB/2 exceeding the output range. In this case, if the R = 0 and S = 0, then the final result should round to nearest even. If G = 1 and either R = 1 or S = 1, then the result needs to be rounded up. Finally, if G = 0, then the R and S bits serve no function, and the result is rounded down.

In floating-point subtractions, several cases can occur in which the final result needs to be shifted one position to the left (i.e., when the result after the subtraction is in the form 0;1XXX...). In these cases, the result is shifted and the guard bit becomes the LSB, which leave the R and S bits to perform rounding. If the round bit were not present then the sticky bit would be all that is left to determine rounding, and an incorrect rounding decision might be made.

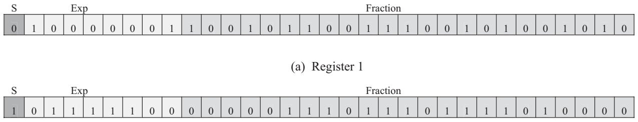
Floating-point multiplication shares a similar rounding scheme for RNE modes, but with slightly different (and more difficult to calculate) rounding bits. Figure 11 shows that floating-point multiplication uses a, C, a R, and an S. No guard bit is needed, as no subtractions are required in multiplication.

The carry bit is a carry-in from the partial product combination of all bits below the range of the output result. Calculating the sticky bit of all out-of-range bits is not enough for a correctly rounded multiplication, as a partial product combination that incorrectly kills a carry can cause a rounding error greater than LSB/2. Because every multiplication results in an intermediate result twice as wide as the original operands, the carry bit requires an adder structure large enough to propagate a carry all the way to the upper-half of the result. Because the carry may cause a carry out of the original MSB of the product, the carry must be propagated upward before the round and sticky bit are calculated to determine the final rounded result.

Rounding Example. Assume a 32-bit single-precision subtraction in RNE mode between the following two data in memory, as shown in Fig. 12.

The numerical representation of these two numbers is as follows:

$$\begin{array}{rl} & 1;1001\_0110\_0111\_0010\_1001\_010 \times 2^{129-127=2} \\ (-) & 1;0000\_0111\_0111\_0111\_1010\_000 \times 2^{124-127=-3} \end{array}$$



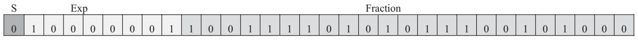
S			Exp																Fraction												
0	1	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	1	0

(a) Register 1

S			Exp																Fraction												
1	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	1	1	1	1	0	1	0	0	0	0

(b) Register 2

Figure 12. The operands as stored in the memory.



S			Exp																Fraction												
0	1	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0	1	0	1	0	1	1	1	0	0	1	0	1	0	0	0

Figure 13. The RNE rounded result.

The smaller fraction is aligned so that the exponent values of each operand match and the subtraction may occur. However, observing that the sign bit of the smaller operand is a 1, the second operand is a negative number. Therefore, the subtraction of a negative number is a "true addition," which results in the following:

$$\begin{array}{rl} & 1;1001\_0110\_0111\_0010\_1001\_010\_\overset{\text{GRS}}{000} \times 2^2 \\ -\quad(-) & 0;0000\_1000\_0011\_1011\_1011\_110\_100 \times 2^2 \\ \hline & 1;1001\_1110\_1010\_1110\_0101\_000\_\overset{\text{GRS}}{100} \times 2^2 \end{array}$$

At this point, because the instruction is in RNE mode, the G, R, and S bits are observed. G = 1, R = 0, and S = 0. This finding means the intermediate result's rounding bits sum to exactly LSB/2. This specific case requires a round to nearest even. Because the closest even result is a round-down, the final result truncates the rounding bits and is stored in memory as:

$$1;1001\_1110\_1010\_1110\_0101\_000 \times 2^{129-127=2}$$

Or more formally as is shown in Fig. 13.

FLOATING-POINT HARDWARE

The hardware required to support floating-point operations can vary widely. As stated in the IEEE-754 specification, compliant floating-point systems may support the requirements in hardware, software, or a combination of both. Therefore, the actual implementation of the floating-point arithmetic is at the discretion of the designer.

However, high-performance IEEE-754 compliant machines typically are implemented mostly in hardware for performance reasons. Floating-point units (FPUs) are usually built as co-processors. The first of these by Intel were off-chip devices that communicated to the central processing unit (CPU) with an external bus. Because the original coprocessor and its derivatives were named as some version of the 8087, the suffix of "x87" to describe floating-point logic stuck. Modern FPU coprocessors embedded on the same silicon as the CPU are still referred to as "x87" devices.

A modern x87 FPU typically comprises a scheduler, register file, load and store units, convert units, MMX units and floating-point execution units. Although all the pieces of the x87 coprocessor are necessary and important, the most critical are the execution units: the floating-point adder (FPA) and the floating-point multiplier (FPM).

FPA Hardware

The FPA is one of the most fundamental units used in x87 coprocessors. It performs floating-point additions in the way described by the addition examples in the previous sections. The unit takes two input floating-point addends and performs an addition, subtraction, or any derivative of the fundamental addition instruction. The floating-point result is rounded according to the IEEE-754 specifications and passed out of the block. In some designs, more common in the x86 market, the floating-point adder also produces an unrounded result so that the control units may detect special cases, denormals, exceptions, and various other data for trap handling.

A modern floating-point adder is nearly always designed using the Farmwald dual-path architecture (9). This general scheme, which has also been selected as the floating-point adder example shown here, splits the addition datapath into two separate parallel cases. The reason for this is that the hardware required to handle data for operands with large exponent differences is very different than that for exponents that are equal or nearly equal. Floating-point adders now build two paths to handle these different data ranges, which are commonly known as the "far path" and the "close path."

It is important to first identify the two general addition/subtraction cases possible in floating-point addition. First, in cases with large exponent differences (i.e., the far path case), the significands must be aligned via a large shifter before addition or subtraction may occur. This behavior has already been demonstrated in the earlier floating-point addition examples: Given two addends with different exponents, the significand of the addend with the smaller exponent must be shifted until the exponents are equal before the significands are added.

However the second close path case was not covered in the examples. In certain subtraction cases with equal exponents, subtraction of nearly equal significands may result in what is called "massive cancellation," in which the final significand may be much smaller than the original and must be normalized via a large shift. Such cases use specialized hardware in the close path.

For example, two floating-point operands are to be subtracted. The two numbers themselves also happen to be nearly identical to one another, say decimal 1.000000 and 1.000001. When the two are subtracted, the result before normalization is decimal 0.000001. Normalization requires a large left-shift of the significand and a significant reduction in the value of the exponent. Cases like these use a completely different set of hardware than normal additions and subtractions. Older floating-point adder units used to handle all of these cases in a single serial path, but modern x87 units gain performance by splitting and parallelizing these mutually exclusive cases in a "dual-path" FPA.

Figure 14 shows the top-view architecture of a dual-path floating-point adder. The design is for the IEEE-754 double-precision format with 64-bit buses for inputs and outputs. The example architecture uses the Farmwald split, which employs a far path and close path in parallel. When the correct path is chosen by the exponent logic, the selected path sends its data to a combined add/round stage (5,6), where performance is gained by combining addition and rounding logic, and both an unrounded and rounded result are produced.

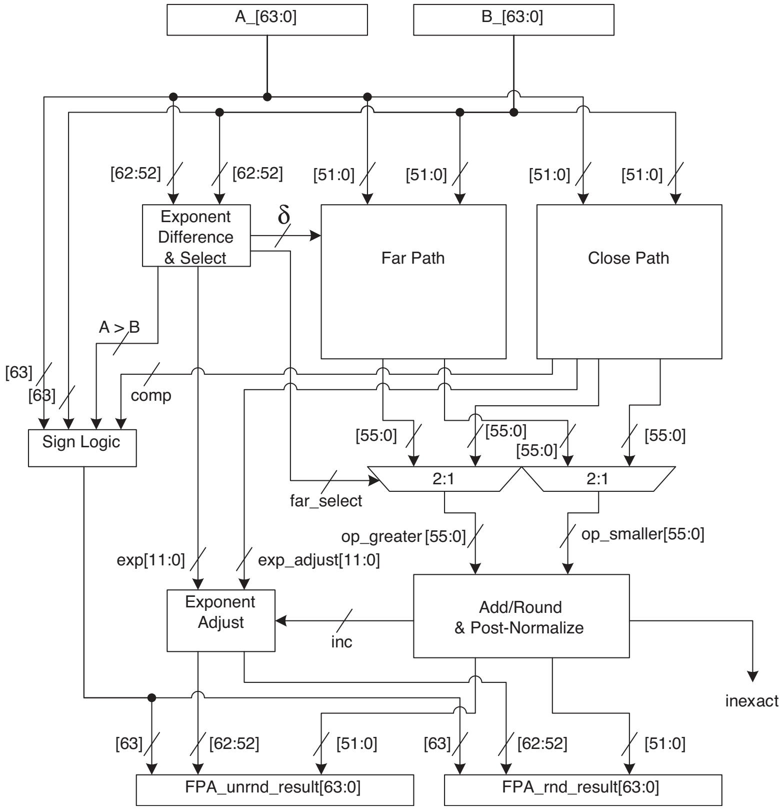


Figure 14. Top view of double-precision floating-point adder.

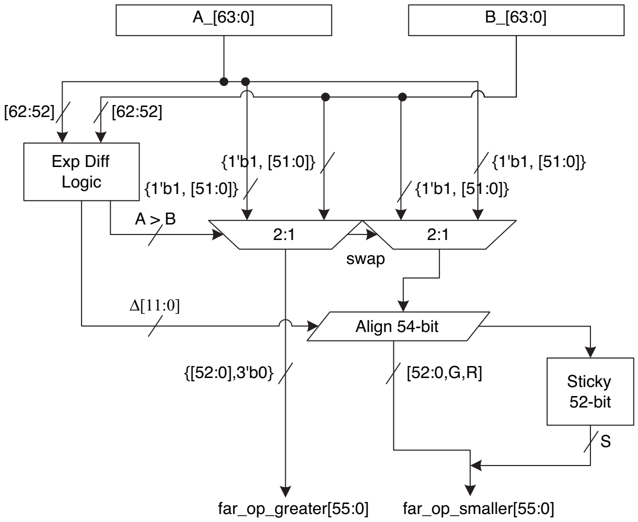


Figure 15. Far path of floating-point adder.

The FPA Far Path. A floating-point adder far path is the significand datapath for all additions and for all subtractions when exponents differ by more than two. The exponent difference of two is selected based on mathematical analysis of floating-point subtraction cases (10). This far path, shown in Fig. 15, is set up to determine how far apart the operand exponents are and to align the significands so that correct floating-point addition/subtraction occurs. Additionally, if the smaller operand is out of the range of what these designs refer to as the "anchor," or the larger operand, which has a static position, then the smaller operand's significand is collected in a sticky bit for rounding.

The implementation of the far path scheme uses a comparator in the exponent logic to determine which operand is larger. When the large operand is identified, the significands of the inputs pass through the "swap" multiplexer stage, which chooses which input is the anchor and which is the one for alignment. In double precision, the smaller operand enters a 54-bit aligner and passes any bits that exceed 54-bits to the sticky tree. The stage is complete when the smaller operand is aligned to the anchor, and the far path results are passed out of the block.

The FPA Close Path. The floating-point adder close path is the significand data path for all subtractions with operand exponents within the difference range of {−1,0,1}. In this data range, a subtraction may cause massive cancellation, which requires a large normalization before the result may be correctly rounded. Massive cancellation cases, like the floating-point adder design presented here, are commonly handled by leading zero anticipator blocks (LZAs).

The LZA is a specialized piece of hardware designed to detect the number of ZEROs above the first ONE in the significand of the result. The LZA logical block runs in parallel with the subtraction itself (and sometimes ahead of it, as seen in this design), calculating only the number of leading ZEROs, which indicates how many bits of normalization are required. This parallel execution allows for an immediate normalization after (or before) the subtraction occurs. The mathematics involved in the construction of an LZA are complex (11).

The close-path architecture is shown in Fig. 16. The input significands are passed to a swap block, a comparator, and three leading-ONEs predictors (LOPs, part of the LZA algorithm). The block begins by determining which exponent, if any, is greater. Once the exponent difference is determined, the operands are swapped, putting the greater

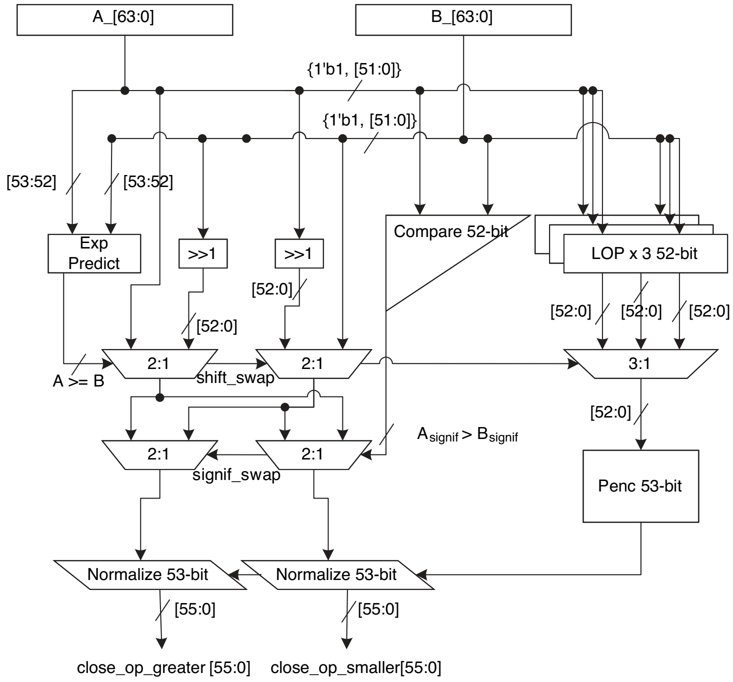


Figure 16. Close path of floating-point adder.

exponent in the "greater operand" path. Three LOPs are used: one for A > B, one for A = B, and one for A < B. The exponent control selects the correct LOP at the same time as the shift swap. The operands are sorted and the LOP result is sent to a priority encoder (the second half of a LZA).

In the case when the exponents are equal, the greater operand is still unknown. To resolve this problem, a second swap stage uses a significand compare select to determine the greater operand. When the swap stages are over, the LZA priority encoder prenormalizes both operands, and the results are passed to the round stage. Prenormalization (or normalization before actual subtraction occurs) is valid because cases of massive cancellation will wipe out any leading ONE bits when the numbers are subtracted, so shifting them out early saves a stage after the actual subtraction occurs.

The FPA Add/Round Stage. Following the parallel processing of the floating-point adder far and close path, the greater and smaller operands from each adder merge paths in two parallel multiplexers. Exponent control has by this point determined the correct numerical path, and the operands from the selection are passed to the combined addition and rounding stage.

The add/round stage architectures used in the floating-point adder are shown in Fig. 17. Two adders are used in parallel, one unbiased and one with a constant, to compute both a rounded and an unrounded result similar to the suggestions by Quach and colleagues (5,6), and the implementation of the SPARC-64 (12). The scheme uses the concept that a correct IEEE-754 rounded result may be obtained by operating on the LSB in an adder sum or an adder sum + 2.

In this implementation, the two input operands are passed to dual 59-bit adders. One of the adders precombines the two operands with a constant: a +2 constant for additions and a +1 constant for subtractions. The MSBs and LSBs of both results are sent to a rounding table, where the correct rounding decision is made based on rounding control. The final rounded significand is selected by the final multiplexer, and both the rounded and un-rounded results are passed out of the block.

FPM Hardware

The FPM is generally the largest block in a floating-point unit, which is built to take two input operands and provide a multiplied and rounded result. The unit itself, when compared with a floating-point adder, has a simpler overall architecture, but it contains components that use a lot of area and power.

Adding to the floating-point multiplier's size and latency is an array of complex arithmetic functions beyond simple multiplication. A common floating-point unit uses the multiplier to process transcendental, divide, and square root algorithms that use ROM tables and multiplicative iterations. Without this additional burden, the floating-point multiplier has a very fast performance with low latency. However, with the additional instructions that must be handled by the unit, the latency increases and the delay becomes similar to that of a floating-point adder.

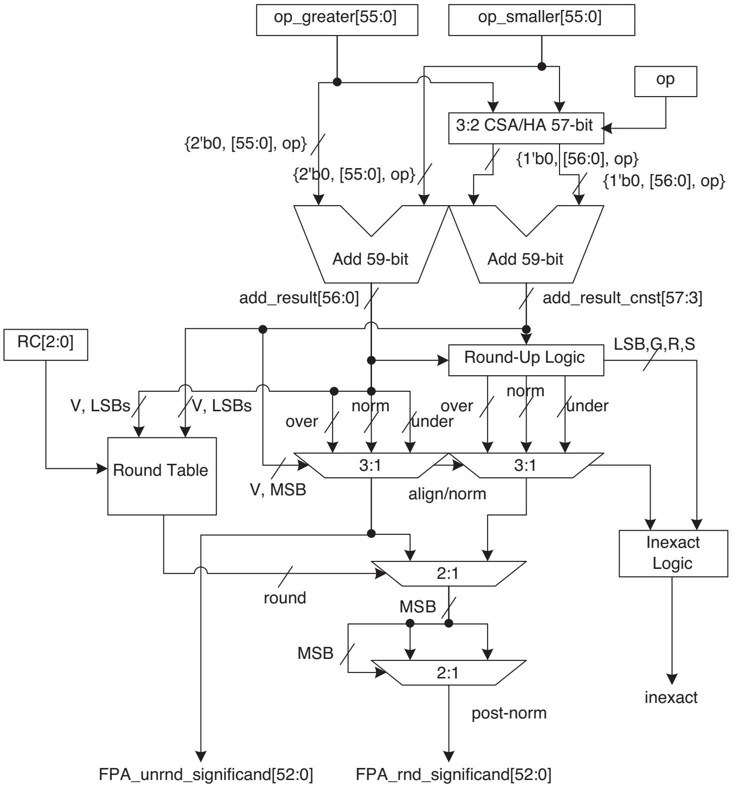


Figure 17. Floating-point adder add/round stage.

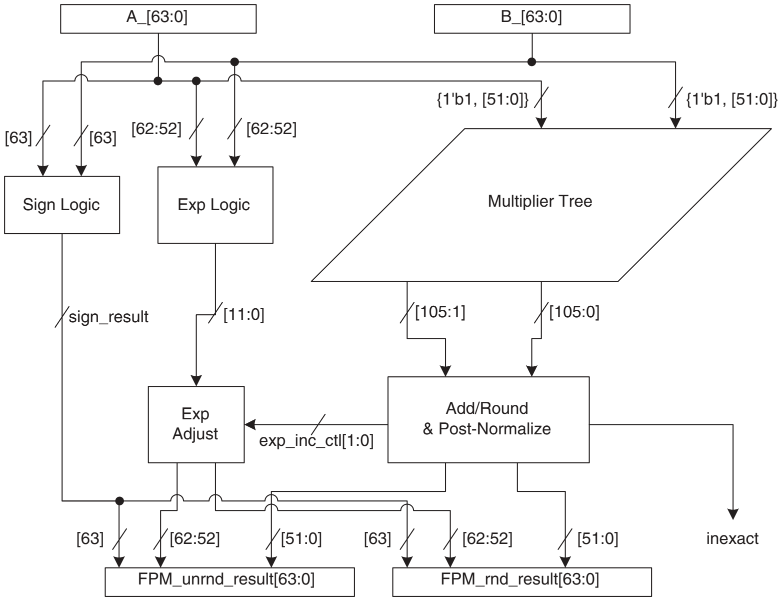


Figure 18. Top view of the floating-point multiplier.

The floating-point multiplier described here has been designed without the transcendental, square root, or division algorithms. Although these algorithms are necessary in a floating-point unit that complies with the IEEE-754 standard, the design is intended to provide the implementation details of a pure floating-point multiplication instruction.

The floating-point multiplier implemented architecture is shown in Fig. 18. The multiplier tree product result passes to a combined add/round stage, where the carry/save product is combined and rounded. The stage outputs both an unrounded and rounded result, and the floating-point multiplication is complete. Both the sign logic and the exponent datapath run in parallel to the significand processing.

The FPM Add/Round Stage. The addition and rounding stage in a floating-point multiplier requires a rounding stage more complex than that of a floating-point adder. As briefly described in earlier sections, the calculation of the rounding bits for FPM units is not simple. Specifically, this multiplier needs unique hardware to produce a rounded result that is the same size as its input operands, all while correctly propagating carries from the lower half of the double precision product.

The floating-point multiplier add/round stage, shown in Fig. 19, uses an architecture similar to those suggested by the authors of Refs. 7 and 8. The upper half of the input carry/save product is passed to two half-adder (HA) stages, where the LSB from each stage is stripped off and sent to a constant 2-bit adder. The remaining upper half enters a compound adder, where the sum and augmented sum (sum + 1) are calculated.

The lower half of the add/round stage input is sent to a carry and sticky tree, where the LSB, C, R, and S bits are produced. These bits combine with rounding control and select the correct increment of the final result's lower 2-bits. Depending on the bit sequence selection of the lower 2-bit constant adder output, the upper half of the result will either be ready for post-normalization or will require the augmented selection. Both the lower 2-bits and the selected compound adder output are postnormalized, and the product is complete.

OTHER IEEE-754 SPECIFICATIONS

The IEEE-754 standard includes far more than precision specifications and descriptions of addition, multiplication, and rounding. The standard identifies several other required mathematical functions, such as divide, square root, reciprocal, as well as convert instructions to pass data between fixed point and floating point. Also included are specifications on reserved values for infinity, not-a-number (NaN), zero, and denormalized numbers, which are frequently referred to as denormals. Each of these items is briefly touched on in this section.

The other functions required by the standard are handled by different x87 machines in different ways. Performance-intensive applications commonly map these other mathematical operations to the floating-point adder and floating-point multiplier hardware. For instance, division can be mapped to either the FPA or FPM, because digit-recurrence (13), Goldschmidt, and Newton-Raphson division require either adders, multipliers, or both in their algorithms (see the chapter on fixed-point computer arithmetic). Reciprocal and square-root estimates may be mapped to a FPM that employs iterative multiplication based on bipartite ROM initial estimations (14). Transcendentals and CORDIC algorithms may be realized with combinations of FPA and FPM hardware. Convert instructions that involve float-to-float, fixed-to-float, and float-to-fixed translations are special

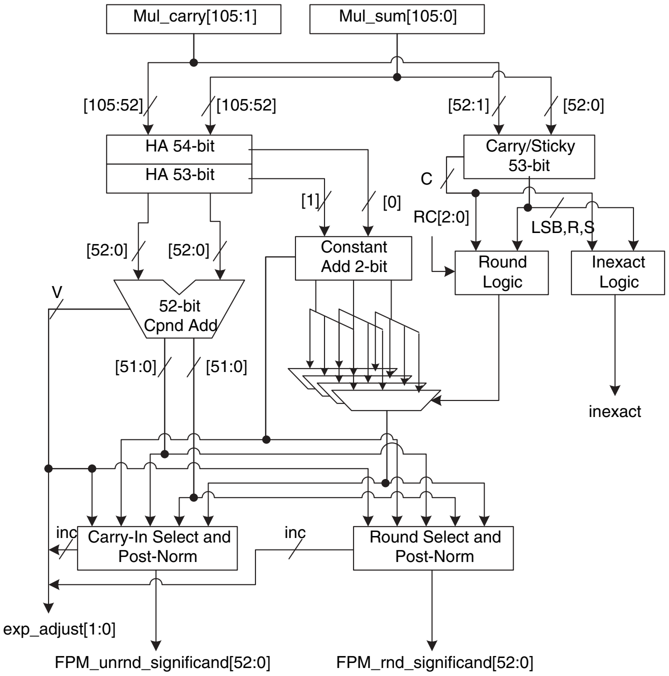


Figure 19. Floating-point multiplier add/round stage.

cases of floating-point addition in which one of the operands is a constant. Depending on the application, convert instructions may be mapped to the FPA or implemented with a separate store-convert unit, which is in essence just a simplified version of the FPA. In any case, most IEEE-754 mathematical operations are subsets of the FPA and FPM operations.

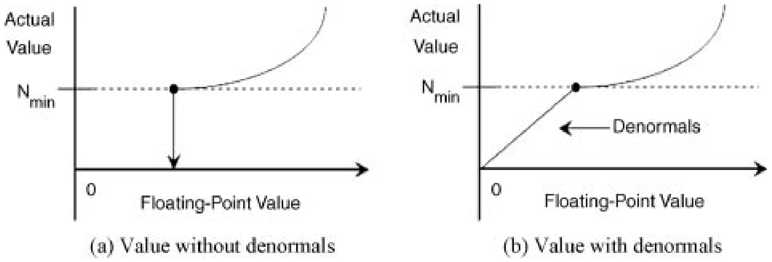
The IEEE-754 specification reserves two exponent fields for special numbers, specifically the largest and the smallest exponent values. For example, in single-precision, the exponent value of 1111_1111 is reserved for ± infinity and NaN, whereas the exponent value of 0000_0000 is reserved for ± zero and denormals.

When the exponent is the lowest value (i.e., 0000_0000 for single precision), if the significand is all 0s, then the number is interpreted as either positive or negative zero,If the significand is non-zero, then the result is a valid number, called a "denormal."

Denormals are special numbers that fill the space between N_{min} (i.e., exp = 0000_0001 and a minimum significand of 1.0000.... for single-precision) and zero. In the normalized floating-point format, no values exist between N_{min} and zero, but rather, the lowest floating-point value would simply jump from N_{min} directly to zero. Denormals solve this problem by holding a numerical value and indicating to the machine that underflow has occurred. Figure 20 shows graphical representations of the range of floating-point values possible with and without denormal numbers, respectively.

Denormals in the IEEE-754 specification are used to express values between N_{min} and zero. If a number is less than N_{min}, then without denormals the number would be rounded to either to zero or to N_{min}. Instead, denormals allow the x87 machine to save the denormalized result. In this fashion, no data are lost.

Actual denormal values are represented in a slightly different format than normal floating-point numbers. Specifically, when the exponent value of a number is all 0s, the significand does NOT have an implicit integer 1. Instead, the significand is assumed to begin with a '0' (i.e., $0;XXX\ldots \times 2^{N_{min}}$). For a better understanding of denormals and how they are handled in x87 machines, refer to Ref. 15.



(a) Value without denormals

(b) Value with denormals

Figure 20. Representation of small numbers.

CONCLUSIONS

This article presents an overview of the IEEE-754 specification for floating-point numbers and how they are used in arithmetic operations, such as addition and multiplication. It then follows with a brief look at floating-point addition and floating-point multiplication via double-precision hardware implementations that convey the fundamentals of an x87 execution unit design. Finally, the article briefly discusses the various special "numbers" required by the standard such as denormals, infinites, zeros, and conversions.

Although many modern processors do not necessarily comply entirely with all elements of the IEEE-754 floating-point standard, the fundamentals of all floating-point systems are similar. This brief introduction should provide a useful starting point. More details are available in Refs. 16-19 .

BIBLIOGRAPHY

1. J. B. Gosling, Design of large high-speed floating-point-arithmetic units, *IEE Proceedings*, vol. **118**, 1971, pp. 493–498.
2. IEEE std 754-1985. IEEE Standard for binary floating-point arithmetic,
3. *Direct3D 10.0 Functional Specification*, Redmond, WA, The Microsoft Corporation, Version 1.06, May 24, 2007.
4. H.-J. Oh, S. M. Mueller, C. Jacobi, K. D. Tran, S. R. Cottier, B. W. Michael, H. Nishikawa, T. Namatame, N. Yano, T. Machida, and S. H. Dhong, A fully pipelined single-precision floating-point unit in the synergistic processor element of a CELL processor, *IEEE J. Solid-State Circ.*, **41**; 759–771, 2006.
5. N. Quach, N. Takagi, and M. Flynn, *On Fast IEEE Rounding*, Technical Report CSL-TR-91-459, Computer Systems Laboratory, Palo Alto, CA: Stanford University, 1991.
6. N. Quach and M. J. Flynn, *An Improved Algorithm for High-Speed Floating Point Addition*, Technical Report CSL-TR-90-442, Computer Systems Laboratory, Palo Alto, CA: Stanford University, 1990.
7. G. Even and P. M. Seidel, A Comparison of three rounding algorithms for IEEE floating-point multiplication, *IEEE Trans. Comput.*, **49**: 638–650, 2000.
8. R. K. Yu and G. B. Zyner, 167 MHz Radix-4 floating-point multiplier, *Proc. of the 12th IEEE Symposium on Computer Arithmetic*, 1995, pp. 149–154.
9. M. P. Farmwald, *On the Design of High Performance Digital Arithmetic Units*, Ph.D. Thesis, Palo Alto, CA: Stanford University, 1981.
10. P.-M. Seidel and G. Even, How many logic levels does floating-point addition require? *Proc. of the International Conference on Computer Design: VLSI in Computers and Processors*, 1998, pp. 142–149.
11. M. S. Schmookler and K. J. Nowka, Leading zero anticipation and detection – a comparison of methods, *Proc. of the 15th IEEE Symposium on Computer Arithmetic*, 2001, pp. 7–12.
12. A. Naini, A. Dhablania, W. James, and D. Das Sarma, 1 GHz HAL Sparc64 dual floating point unit with RAS features, *Proc. of the 15th IEEE Symposium on Computer Arithmetic*, 2001, pp. 173–183.
13. M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Boston, MA: Kluwer Academic Publishers, 1994.
14. M. J. Schulte and J. E. Stine, Symmetric bipartite tables for accurate function approximation, *Proc. of the 13th IEEE Symposium on Computer Arithmetic*, 1997, pp. 175–183.
15. E. M. Schwarz, M. Schmookler, and S. D. Trong, FPU implementation with denormalized numbers, *IEEE Trans. Comput.*, **54**: 825–836, 2006.
16. B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, New York: Oxford University Press, 2000.
17. I. Koren, *Computer Arithmetic Algorithms*, 2nd ed., Wellesley, MA: A K Peters, Ltd., 2001.
18. M. D. Ercegovac and T. Lang, *Digital Arithmetic*, San Francisco, CA: Morgan Kaufmann Publishers, 2004.
19. M. L. Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, Philadelphia, PA: Society for Industrial and Applied Mathematics, 2001.

ERIC QUINNELL
Advanced Micro Devices
Austin Texas

EARL E. SWARTZLANDER, JR.
University of Texas at Austin
Austin, Texas

FLUENCY WITH INFORMATION TECHNOLOGY

Fluency with Information Technology specifies a degree of competency with computers and information in which users possess the skills, concepts, and capabilities needed to apply Information Technology (IT) confidently and effectively and can acquire new knowledge independently. Fluency with Information Technology transcends computer literacy and prepares students for lifelong learning of IT.

The concept of *Fluency with Information Technology* derives from a National Research Council (NRC) study initiated in 1997 and funded by the National Science Foundation (NSF). The study focused on defining "what everyone should know about Information Technology." For the purposes of the study, "everyone" meant the population at large, and IT was defined broadly to include computers, networking, software and applications, as well as information resources—virtually anything one would encounter using a network-connected personal computer.

The NSF motivation for requesting the study was driven by the belief that much of the United States population is already "computer literate," but that literacy is not enough. With more knowledge, people would make greater use of IT, and doing so would generally be beneficial. Specifically, the NSF noted these points:

- Most users have had no formal training in the use of IT because of the relatively brief period during which it has entered our society; more complete knowledge could be useful.
- Many users seem to have only a limited understanding of the applications they use and (probably correctly) assume they are underutilizing them.
- Many users are not confident nor do they feel in control when confronted with Information Technology opportunities or problems.
- Extravagant claims have been made about the potential benefits of IT, but most citizens do not enjoy them; they want to apply IT to personally relevant goals.
- Informed participation in certain contemporary social and political discussions—strong encryption, copyright, spam, privacy, and so forth—requires a sound understanding of IT.

What knowledge would address these concerns?

The NRC, under the auspices of the Computer Science and Telecommunication board, appointed seven experts to the ad hoc Committee on Information Technology Literacy:

Lawrence Snyder, University of Washington, Chair
Alfred V. Aho, Lucent Technologies
Marcia Linn, University of California at Berkeley
Arnold Packer, The Johns Hopkins University
Allen Tucker, Bowdoin College
Jeffrey Ullman, Stanford University
Andries Van Dam, Brown University

Herbert Lin of the NRC staff assisted the committee.

Meeting over a two-year period, the committee broadly solicited information, using invited testimony from various stakeholders, electronic queries to the community at large, and a public forum. The broad range of views expressed indicated that *computer literacy*, which currently is understood to teach students how to use specific applications, does not have the "staying power" to prepare people for the continual change that is so familiar in IT. Users must be trained to be more adaptive and ready and willing to change. The committee decided that a deeper, more foundational understanding was needed that would allow people to respond to change through a process of lifelong learning. The term *fluency*, suggested by Yasmin Kafai of UCLA, became the moniker for that deeper, foundational knowledge.

The committee issued its report, *Being Fluent with Information Technology*, in June 1999, published by the National Academy Press (1). The report attracted considerable interest, and some schools immediately began offering college-level classes. By July 2002 Addison Wesley published the first textbook *Fluency with Information Technology*. And in the fall of 2003, an online course to teach the content was launched with NSF funding.

CONTENT OF FLUENCY WITH INFORMATION TECHNOLOGY

To provide the necessary foundation to support lifelong learning in IT, the committee recommended a tripartite body of knowledge that covers contemporary skills, fundamental concepts, and intellectual capabilities:

Skills—the ability to use contemporary computer applications. Skills knowledge makes computers and information resources useful immediately; it provides valuable employment training and supports the other aspects of *fluency* education. Examples include word processing, web searching, and so forth.

Concepts—the fundamental information about IT drawn from its supporting fields. Concepts knowledge includes both "general science" knowledge, such as how a computer works, that educated citizens should know and directly applicable knowledge, such as algorithmic thinking, that underpins future applications of IT by users.

Capabilities—the higher-level thinking skills needed for everything from finding new ways to exploit IT to

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

recovering from errors. Capabilities—reasoning, debugging, problem solving, and so forth—apply in everyday life as well as in IT. However, because they occur often and intensively in IT, it is essential that users be accomplished with them.

Notice that *skills* generally align with traditional computer literacy, so *fluency* includes literacy as a component.

The committee, charged with developing the content, chose not to enumerate every conceivable skill, concept or capability but rather to identify the top ten most important items in each category. Their recommendation (1, p. 4) is shown in the companion table. NRC Recommended Fluency Topics

Skills –

- Set up a personal computer
- Use basic operating system facilities
- Use a word processor
- Use a graphics/artwork/presentation tool
- Connect a PC to a network
- Search the Internet to locate information
- Send and receive e-mail
- Use a spreadsheet
- Query a database
- Use online help and tutorial facilities

Concepts –

- Principles of computer operation
- Enterprise information systems
- Networking
- Digital representation of information
- Information structure and assessment
- Modeling the world with computers
- Algorithmic thinking and programming
- Universality of computers
- Limitations of computers
- Information and society

Capabilities –

- Engage in sustained reasoning
- Manage complexity
- Test a solution
- Locate bugs in an IT solution
- Organize and navigate information structures
- Collaborate using IT
- Communicate an IT solution to others
- Expect the unexpected
- Anticipate technological change
- Think technologically

An important aspect of the recommended topics is that the *skills*, built around contemporary software and systems, can be expected to change over time. The *concepts* and *capabilities*, focused on fundamental ideas and intellectual processes, are generally time-invariant.

PROJECTS AS AN INTEGRATING MECHANISM

The committee asserted that *skills*, *concepts*, and *capabilities* are separate but interdependent types of knowledge. Furthermore, because most nontrivial applications of information technology rely on applying all three kinds of knowledge seamlessly, it is best not to teach them in isolation. Rather, the material should be integrated using projects. Projects—multi week activities that produce a specific IT "product" such as a web page or database—can be designed to use several components from each list to give students an experience that approximates realistic IT situations. Through the projects, *fluency* becomes a coherent body of knowledge rather than a set of 30 disparate topics.

An example of a project might be to build a database to support patient processing at a storefront medical clinic. Students would set up the database tables, define the queries, and design the user interfaces to track patients through processing at the clinic. In addition to a record of the visit, documents would be generated for follow-up actions such as referral letters, labels for specimens for laboratory tests, prescriptions, (possibly) payment invoices and receipts, and so foth.

The medical database example combines the use of several *skills*, *concepts*, and *capabilities*. Among the *skills* applied might be Web searching to find out privacy requirements, database querying, using basic operating system facilities, using online help facilities, and probably others. Among the *concepts* applied might be fundamentals of database design, information representation, information structure and organization, social implications of computing, and probably others. Among the *capabilities* applied might be sustained logical reasoning, debugging, solution testing, communicating a technological solution, and probably others. When working on a project, a student is focused on solving the problem, not on which of the curricular components he or she is applying at the moment. As a result, projects produce an integrated application of IT that conforms closely to how it is used under non academic circumstances. Furthermore, successful completion of such substantial efforts can give students a greater degree of confidence as computer users than can exercises specialized to a particular topic.

Appendix A of the report (1, pp. 67–77) gives a listing of other sample projects.

ALGORITHMIC THINKING AND PROGRAMMING

Since the introduction of the first computer courses for non specialists in the 1970s, the question has been debated whether programming should be a component of the computing knowledge of the general population. The subject is complex and often has been divisive. Fair-minded commentators have offered thoughtful and substantive arguments on both sides, but no clear resolution has emerged. Curricula have been developed taking each position.

The NRC committee was confronted with the question, too: Should programming be listed among the *skills*, *concepts*, and *capabilities*? The matter took on a crisp form in the testimony presented to the committee. Most contributors recommended that *algorithmic thinking* be a part of "what everyone should know about Information Technology." They asserted that educated people should be able to formulate procedural solutions to problems with sufficient precision that someone (or something) else could implement them. It also was common for the committee to be advised not to include *programming* as a requirement for the general population. Programming embodies professional knowledge that is too advanced and too detail-oriented to be of value to users. That is, the committee was told that algorithmic thinking was essential and programming was not.

Although algorithmic thinking and programming are distinct, they overlap substantially. The committee wrestled with how to interpret these largely contradictory inputs. Was the committee being told to prescribe those aspects of algorithmic thinking that do not involve programming in any way? Or was it being told to prescribe algorithmic thinking in full but to include programming only to support it, not a full, professional-level programming course? In addition to the algorithmic thinking/programming question, related issues existed: What were those individuals who recommended programming including in their recommendation, passing familiarity to programming or deep knowledge? Also, because programming is a challenging engineering discipline, how much aptitude for learning programming does the general population possess?

The committee finally resolved these difficult issues by including algorithmic thinking and programming as item # 7 among the *concepts*. In explaining the recommendation (1, pp. 41–48), the committee stated that programming is included to the extent necessary to support a thorough development of algorithmic thinking. It did not recommend "majors" -level programming knowledge. Specifically, the committee identified a small list of programming concepts that it regarded as sufficiently fundamental and generally accessible to be considered part of that programming content. These concepts include name and value, assignment, conditional execution, and repeated execution. It supported the choice by presenting examples where the ideas occur both within and beyond IT. Since the publication of the report, its wide distribution has exposed the approach to considerable scrutiny. Because the *fluency* concept has been embraced widely and its algorithmic thinking/programming resolution has engendered almost no comment of any kind, it must be concluded that the committee recommendation settles—yes, but only a little—the longstanding programming question.

IMPLEMENTING FLUENCY WITH INFORMATION TECHNOLOGY

The charge of the NRC committee was to specify only the content that "everyone should know about Information Technology." The implementation of FITness—the term used by the committee to identify those who are fluent with information technology—education would be left until later. But the committee was composed of academics, so, inevitably, teaching *fluency* at the college level was addressed.

The committee began by noting the desirability of FITness as a post condition of college, that is, knowledge with which students leave college. Eventually, the report stated, FITness should be a pre condition of college, like basic knowledge of science, math, and foreign languages. Although appropriate for pre-college, the challenges are so great for implementing fluency instruction in K–12 that it will be years before FITness can be an entrance requirement. Therefore, teaching *fluency* in college is essential in the short run. Also, compared with K–12 public instruction, post secondary institutions are far more flexible in terms of their ability to develop curricula and infrastructure for new pedagogical endeavors.

The NRC committee did not define a curriculum, only the content. Curriculum development began in the spring term of 1999 with the offering of the first FITness class, CSE100 *Fluency with Information Technology* at the University of Washington. The goal of this class is to teach the recommended *skills*, *concepts* and *capabilities* to freshmen in one 10-week quarter. The class has three lectures and two labs per week, each of 50 minutes. *Skills* are taught primarily in the labs, *capabilities* are presented primarily as lecture demonstrations, and *concepts* are learned primarily through reading. [Course notes from early offerings of this program became the textbook, *Fluency with Information Technology*, published by Addison Wesley (2).] Three projects integrate the material. Other universities, colleges, and community colleges have developed FITness curricula since then.

Because the *skills*, *concepts*, and *capabilities* are such different kinds of knowledge, teaching *fluency* requires a varied strategy.

Skills material—word processing, browsing, processing e-mail, and so on—is best taught in a lab with a video projector connected to the computer of the instructor. In the lab, students "learn through doing," and, because an important part of learning an application is familiarity with the GUI, the video display facilitates demonstrations. Furthermore, the detailed "click here, click there" instruction should give way quickly to more generic instruction that describes general properties of PC applications. This process allows students to learn how to learn an application, which makes them more independent.

Concepts material—computer operation, database principles, network protocols, and so forth—is effectively science and can be learned most efficiently through a combination of textbook reading and lectures that amplify and illustrate the ideas. Because computers are so fast and the common applications are built with millions of lines of software, students will not be able to recognize the instruction interpretation cycle of a computer or TCP/IP in their direct experience. So, the goal is simply to explain, as is done in physics or biology, how basic processes work.

Capabilities material—logical reasoning, complexity management, debugging, and so forth—is higher-level thinking that often is learned through life experience. Because *capabilities* generally are non algorithmic, they are somewhat more challenging to teach. Lecture demonstrations are effective because the class can, say, debug a problem together, which illustrates the process and provides a context for commentary on alternative techniques. The *capabilities*, being instances of thinking, are not learned entirely in a FITness course; students will continue to hone their knowledge throughout life.

As noted, projects provide the opportunity to apply and integrate these three kinds of knowledge.

The committee recommended that the ideal case would be for *fluency* instruction to be incorporated into discipline-specific IT instruction when possible. That is, as architecture students, business majors, and pharmacists learn the technology that supports their specialties, they also learn the recommended *skills*, *concepts*, and *capabilities*. Projects could be specialized to their area of study, and problem solving could incorporate discipline-specific methodologies. Although it has advantages, the recommendation implies that students learn FITness relatively late in their college career, that is, after choosing a major. Because the material applies across the curriculum, it is advantageous to learn it much earlier, for example, freshman year or before college, so that it can support the whole academic program. A compromise would be to offer generic FITness, as described above, as early as possible, and then to give further instruction on the *capabilities* in a research methods or career tools class once students have specialized in a major.

NEXT STEPS FOR FLUENCY

Universities, colleges, and community colleges are adopting *fluency* courses at a rapid rate, and schools outside the United States are beginning to promote FITness. These courses typically replace traditional computer literacy classes both because *fluency* provides a more useful body of knowledge and because a majority of students are computer literate when they enter post secondary schools. So, *college-age* students are becoming FIT, but what about the remainder of the population?

The original NSF question asked what the population at large should know about IT. Enrolled college students are only a small part of that, which raises the question of how the adult population not in school is to become *fluent*. This problem remains unsolved. One small contribution to the effort is a free, online, self-study version of the University of Washington Fluency with Information Technology course, called BeneFIT100. Any person with a computer and Internet connection who speaks basic English, is computer literate enough to browse to the University of Washington Website, is disciplined enough to take an online class, and is motivated to become *fluent* can use BeneFIT100 to do so. Although many people doubtless meet those five qualifications, still they are a minority. Expanding access to *fluency* instruction likely will remain a difficult problem and the main challenge for the foreseeable future.

BIBLIOGRAPHY

1. National Research Council, *Being Fluent with Information Technology*, Washington, D. C.: National Academy Press, 1999.
2. L. Snyder, *Fluency with Information Technology: Skills, Concepts, and Capabilites*, 3rd ed., Reading, MA: Addison Wesley, 2007.

LAWRENCE SNYDER
University of Washington—Seattle
Seattle, Washington

INFORMATION TECHNOLOGY

This article provides a brief account of information technology. The purpose of information technology is discussed and a brief account is given of its historical evolution. Then the many profound influences that information technology has had, is having, and will have on organizations, and the way in which this effect will strongly affect the engineering of information technology products and services will be discussed. A presentation of 10 major challenges for information technology in the 21st century concludes the article.

Fundamentally, information technology (IT) is concerned with improvements in a variety of human problem-solving endeavors through the design, development, and use of technologically based systems and processes that enhance the efficiency and effectiveness of information and associated knowledge in a variety of strategic, tactical, and operational situations. Ideally, this is accomplished through critical attention to the information needs of humans in problem-solving tasks and in the provision of technological aids, including computer-based systems of hardware and software and associated processes, that assist in these tasks. Information technology activities complement and enhance, as well as transcend, the boundaries of traditional engineering through emphasis on the information basis for engineering as contrasted with the physical science basis for traditional engineering endeavors.

Information technology is composed of hardware and software that enable the acquisition, representation, storage, transmission, and use of information. Success in information technology is dependent upon being able to cope with the overall architecture of systems, their interfaces with humans and organizations, and their relations with external environments. It is also very critically dependent upon the ability to successfully convert information into knowledge.

The initial efforts at provision of information technology based systems concerned implementation and use of new technologies to support office functions. These have evolved from electric typewriters and electronic accounting systems to include very advanced technological hardware, such as facsimile machines and personal computers to perform such important functions as electronic file processing, accounting, and word processing. Now networking is a major facet of information technology.

It is neither possible nor desirable to provide a detailed discussion of the present IT based devices and products, or the many possible future developments in this exciting area. There are a plethora of relevant articles in this encyclopedia that discuss these in some detail.

HISTORICAL EVOLUTION OF INFORMATION TECHNOLOGY

In the early days of human civilization, development was made possible primarily through the use of human effort, or labor. Human ability to use natural resources led to the ability to develop based not only on labor, but also on the availability of natural resources. Natural resources are usually denoted land as the classic economic term that implies natural physical resources. At that time, most organizations were composed of small proprietorships. The availability of financial capital during the Industrial Revolution led to this being a third fundamental economic resource, and also to the development of large, hierarchical corporations. This period is generally associated with centralization, mass production, and standardization.

In the latter part of the industrial revolution, electricity was discovered, and later the semiconductor. This has led to the information age, or the information technology age. Among the many potentially critical information technology based tools are: data base machines, E-mail, artificial intelligence tools, facsimile transmission (fax) devices, fourth-generation programming languages, local area networks (LAN), integrated service digital networks (ISDN), optical disk storage (CD-ROM) devices, personal computers, parallel processing algorithms, word processing software, computer-aided software engineering packages, word processing and accounting software, and a variety of algorithmically based software packages. Virtually anything that supports information acquisition, representation, transmission, and use can be called an information technology product.

Easy availability of technologies for information capture, storage, and processing has led to information, as well as the product of information in context or knowledge, as a fourth fundamental economic resource for development. This is the era of total quality management, mass customization of products and services, reengineering at the level of product and process, and decentralization and horizontalization of organizations, and systems management. While information technology has enabled these changes, much more than just information technology is needed to bring them about satisfactorily. In this article, attention will be primarily focused upon information technology and its use by individuals and organizations to improve productivity and the human condition.

Commentators such as Bell (1), Toffler (2,3), and Zuboff (4) have long predicted the coming of the information age. The characteristics of the information age are described in a number of contemporary writings as well (5,6). Alvin Toffler writes of three waves: the agriculture, industrial, and

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

information or knowledge ages. Within these ages are numerous subdivisions. For example, the information age could be partitioned into the era of vertically integrated and stand-alone systems, process reengineering, total quality management, and knowledge and enterprise integration. Information and knowledge are now fundamental resources that augment the traditional economic resources: land or natural resources, human labor, and financial capital. Critical success factors for success in the third wave, or information age, have been identified (7) and include: strategy, customer value, knowledge management, business organization, market focus, management accounting, measurement and control, shareholder value, productivity, and transformation to the third-wave model for success.

There are numerous other methods of partition of these "ages." The information age could be partitioned into the age of mainframe computers, minicomputers, microcomputers, networked and client–server computers, and the age of knowledge management.

Major growth in the power of systems for computing and communicating, and associated networking is quite fundamental and has changed relationships among people, organizations, and technology. These capabilities allow us to study much more complex issues than was formerly possible. They provide a foundation for dramatic increases in learning and in both individual and organizational effectiveness. In large part, this is due to the networking capability that enables enhanced coordination and communications among humans in organizations. It is also due to the vastly increased potential availability of knowledge to support individuals and organizations in their efforts. However, information technologies need to be appropriately integrated within organizational frameworks if they are to be broadly useful. This poses a transdisciplinary challenge of unprecedented magnitude if we are to move from high-performance information technologies to high-performance organizations.

In years past, broadly available capabilities never seemed to match the visions offered, especially in terms of the time frame of their availability. Consequently, despite compelling predictions, traditional methods of information access and utilization continued their dominance. As a result, comments that go something like "computers are everywhere except in productivity statistics" have often been made ((8)). In just the past few years, the pace has quickened quite substantially and the need for integration of information technology issues with organizational issues has led to the creation of a field of study sometimes called "organizational informatics" or, more recently, "knowledge management," the objectives of which generally include:

- Capturing human information and knowledge needs in the form of system requirements and specifications
- Developing and deploying systems that satisfy these requirements
- Supporting the role of cross-functional teams in work
- Overcoming behavioral and social impediments to the introduction of information technology systems in organizations
- Enhancing human communication and coordination for effective and efficient workflow through knowledge management

Networks in general and the internet and World Wide Web in particular (9,10) have become ubiquitous in supporting these endeavors. Almost everyone is growing up digital (5), searching for designs for living in the information age, and going on line to communicate with one another and to use digital libraries. It also seems that hardware and software become obsolete as soon as they are unboxed. Indeed, with current trends, boxes themselves may become quaint collectibles. However, organizational productivity is not necessarily enhanced unless attention is paid to the human side of developing and managing technological innovation (11) to ensure that systems are designed for human interaction.

Because of the importance of information and knowledge to an organization, two related areas of study have arisen. The first of these is concerned with technologies associated with the effective and efficient acquisition, transmission, and use of information, or information technology. When associated with organizational use, this is sometimes called organizational intelligence, or organizational informatics. The second area, known as knowledge management, refers to an organization's capacity to gather information, generate knowledge, and act effectively and in an innovative manner on the basis of that knowledge. This provides the capacity for success in the rapidly changing or highly competitive environments of knowledge organizations. Developing and leveraging organizational knowledge is a key competency and, as noted, it requires information technology and much more than information technology capabilities. Thus, it can be seen that information technology has led to organizational informatics, and in this has led to knowledge management.

It would be very difficult to capture the state of information technology in an encyclopedia article without the article being hopelessly out of date even before the encyclopedia emerged from the publisher. Consequently, the goal of this article is to consider several broad trends, which we feel will persist regardless of the specific technologies that enable them. In particular, we focus on effects of information technology and on the organizational implications of these effects.

INFORMATION TECHNOLOGY AND INFORMATION TECHNOLOGY TRENDS

There are several ways in which one can define information technology. The US Bureau of Economic Analysis appears to define it in terms of office, computing, and accounting machinery. Others consider information technology as equivalent to information-processing equipment, which includes communications equipment, computers, software, and related office automation equipment. Still others speak about the technologies of the information revolution (12) and identify such technologies as advanced semiconductors, advanced computers, fiber optics, cellular technology, satellite technology, advanced networking, improved human–computer interaction, and digital transmission and digital compression. We would not quarrel with the content in this list, although we would certainly add software and middleware technology. It could be argued that software is intimately associated with advanced computers and communications. This is doubtlessly correct; however, there is still software associated with the integration of these various technologies of hardware and software to comprise the many information technology based systems in evidence today and which will be in use in the future.

Several trends transcend debates about technology alternatives—currently alternatives such as PCs versus Net PCs or ISDN versus cable. These overriding trends concern directions of computer and communications technologies, and the impacts of these directions on knowledge management and organizations.

The information revolution is driven by technology and market considerations and by market demand and pull for tools to support transaction processing, information warehousing, and knowledge formation. Market pull has been shown to exert a much stronger effect on the success of an emerging technology than technology push. There is hardly any conclusion that can be drawn other than that society shapes technology (13) or, perhaps more accurately stated, technology and the modern world shape each other in that only those technologies which are appropriate for society will ultimately survive.

The potential result of this mutual shaping of information technology and society is knowledge capital, and this creates needs for knowledge management. The costs of the information technology needed to provide a given level of functionality have declined dramatically over the past decade—especially within the last very few years—due to the use of such technologies as broadband fiber optics, spectrum management, and data compression. A transatlantic communication link today costs one tenth of the price that it did a decade ago, and may well decline by another order of magnitude within the next three or four years. The power of computers continues to increase and the cost of computing has declined by a factor of 10,000 or so over the past 25 years. Large central mainframe computers have been augmented, and in many cases replaced, by smaller, more powerful, and much more user-friendly personal computers. There has, in effect, been a merger of the computer and telecommunications industries into the information technology industry and it now is possible to store, manipulate, process, and transmit voice, digitized data, and images at very little cost.

Current industrial and management efforts are strongly dependent on access to information and associated knowledge. The world economy is in a process of globalization and it is possible to detect several important changes. The contemporary and evolving world is much more service oriented, especially in the more developed nations. The service economy is much more information- and knowledge-dependent and much more competitive. Further, the necessary mix of job skills for high-level employment is changing. The geographic distance between manufacturers and consumers, and between buyers and sellers, is often of little concern today. Consequently, organizations from diverse locations compete in efforts to provide products and services. Consumers potentially benefit as economies become more transnational.

The information technology revolution is associated with an explosive increase of data and information, with the potential for equally explosive growth of knowledge. Information technology and communication technology have the capacity to change radically the production and distribution of products and services and, thereby to bring about fundamental socioeconomic changes. In part, this potential for change is due to progressively lowered costs of computer hardware. This is associated with reduction in the size of the hardware and, therefore, to dematerialization of systems. This results in the ability to use these systems in locations and under conditions that would have been impossible just a few years ago. Software developments are similarly astonishing. The capabilities of software increase steadily, the costs of production decrease, reliability increases, functional capabilities can be established and changed rapidly, and the resulting systems are ideally and often user friendly through systems integration and design for user interaction. The potential for change is also brought about due to the use of information technology systems as virtual machines, and the almost unlimited potential for decentralization and global networking due to simultaneous progress in optical fiber and communication satellite technology (10).

The life cycle of information technology development is quite short and the technology transfer time in the new "postindustrial," or knowledge-based, society brought about by the information revolution is usually much less than in the Industrial Revolution. Information technology is used to aid problem-solving endeavors by using technologically based systems and processes and effective systems management. Ideally, this is accomplished through:

- Critical attention to the information needs of humans in problem-solving and decision-making tasks
- Provision of technological aids, including computer-based systems of hardware and software and associated processes, to assist in these tasks

Success in information technology and engineering-based efforts depends on a broad understanding of the interactions and interrelations that occur among the components of large systems of humans and machines. Moreover, a successful information technology strategy also seeks to meaningfully evolve the overall architecture of systems, the systems' interfaces with humans and organizations, and their relations with external environments.

As just discussed, the most dominant recent trend in information technology has been more and more computer power in less and less space. Gordon Moore, a founder of Intel, noted that since the 1950s the density of transistors on processing chips has doubled every 18 to 24 months. This observation is often called "Moore's Law." He projected that doubling would continue at this rate. Put differently, Moore's Law projects a doubling of computer performance every 18 months within the same physical volume. Schaller (14) discusses the basis for Moore's Law and suggests that this relationship should hold through at least 2010. The implication is that computers will provide increasingly impressive processing power. The key question, of course, is what we will be able to accomplish with this power.

Advances in computer technology have been paralleled by trends in communications technology. The Advanced Research Projects Agency Network, or ARPAnet as it is commonly called, emerged in the 1960s, led to the Internet Protocol in the 1970s, and the Internet in the 1980s. Connectivity, or the ability to connect to almost any destination, is now on most desktops, E-mail has become a "must have" business capability, and the World Wide Web is on the verge of becoming a thriving business channel. The result is an emerging networking market. Business publications are investing heavily to attract readers—or browsers—of their on-line publications (15). Telecommunications companies are trying to both avoid the obsolescence that this technology portends and to figure out how to generate revenues and profits from this channel. The result has been a flurry of mergers and acquisitions in this industry.

These strong trends present mixed blessings for end-users. The dramatic increases of processing power on desktops is quickly consumed by operating systems, browsers, multimedia demos, and so on. The escalating availability of data and information often does not provide the knowledge that users of this information need in order to answer questions, solve problems, and make decisions. The notion of "data smog" has become very real to many users (16). Not surprisingly, data smog possibilities encourage opportunities for technologies for coping with data smog. For example, "push" technology enables information to find users (17,18). Another possibility is "intelligent agent" technology whereby users can instruct autonomous programs to search for relevant data and information (19,20). While these types of capabilities are far from mature, they appear quite promising. The specifics of the above trends will surely change—probably by the time this article appears in print. However, the overall phenomenon of greater and greater processing power, connectivity, and information will be an underlying constant.

That the price of computing has dropped in half approximately every two years since the early 1980s is nothing short of astounding. Had the rest of the economy matched this decline in prices, the price of an automobile would be in the vicinity of $10. Organizational investments in information technologies have increased dramatically and now account for approximately 10% of new capital equipment investments by US organizations. Roughly half of the labor force is employed in information related activities. On the other hand, productivity growth seems to have continually declined since the early 1970s, especially in the service sector that comprises about 80% of information technology investments (8). This situation implies the need to effectively measure how IT contributes to productivity, to identify optimal investment strategies in IT, and to enhance IT effectiveness through knowledge management for enhanced productivity.

On the basis of a definitive study of IT and productivity, Brynjolfsson and Yang (8) draw a number of useful conclusions. They suggest that the simple relationship between decreases of productivity growth in the US economy and the rapid growth of computer capital is too general to draw meaningful conclusions. In particular, poor input and output data quality are responsible for many difficulties. Many of the studies they review suggest that the US economy would not be enjoying the boom that it is enjoying in the latter half of the 1990s without information technology contributions. Their study suggests improvements in accounting and statistical record keeping to enable better determination of costs and benefits due to IT. They support the often expressed notion that information technology helps us perform familiar tasks better and more productively, but that the greatest improvement is in enabling us to perform entirely new activities that would not be possible without these new information technologies. One of these, for example, is the ability to make major new product introductions into the world economy with no macro-level inventory changes being needed. They indicate some of the new economic and accounting approaches that are needed to provide improved measures for performance evaluation, such as activity based costing and activity based management (21,22).

Although information technology does indeed potentially support improvement of the designs of existing organizations and systems, it is also enables the creation of fundamentally new ones, such as virtual corporations and major expansions of organizational intelligence and knowledge. It does so not only by allowing for interactivity in working with clients to satisfy present needs, but also through proactivity in planning and plan execution. An ideal organizational knowledge strategy accounts for future technological, organizational, and human concerns, to support the graceful evolution of products and services that aid clients. Today, we realize that human and organizational considerations are vital to the success of information technology.

This is clearly the network age of information and knowledge. One of the major challenges we face today is that of capturing value in the network age (23). Associated with these changes are a wide range of new organizational models. Distributed collaboration across organizations and time zones is becoming increasingly common. The motivation for such collaboration is the desire to access sources of knowledge and skills not usually available in one place. The result of such new developments in information technology as network computing, open systems architectures, and major new software advances has been a paradigm shift that has prompted the reengineering of organizations; the development of high-performance business teams, integrated organizations, and extended virtual enterprises (24); the emergence of loosely-structured organizations (25) and has enabled the United States to regain the productive edge (26) that it once had but lost in the 1980s and early 1990s.

INFORMATION TECHNOLOGY CHALLENGES

There are a number of substantial challenges associated with use of information technology in terms of enhancing the productive efforts of an individual, a group, and an organization. Many of these are challenges that affect managing both people and knowledge. The command-and-control model of leadership is a poor fit for managing organizations where the participants are not bound to the organization by traditional incentive and reward systems (27). A collaborative effort has to continue to make sense and to provide value to participants for it to be sustained. Otherwise, knowledge and skills are quite portable and the loss of knowledgeable workers is a major potential downside risk for organizations today.

There are three keys to organizations prospering in this type of environment: (1) speed, (2) flexibility, and (3) discretion (25). Speed means understanding a situation (e.g., a market opportunity), formulating a plan for pursuing this opportunity (e.g., a joint venture for a new product), and executing this plan (e.g., product available in stores) quickly, in this case all within a few weeks at most. The need for flexibility is obvious; otherwise, there would not be any advantage to speed. However, flexibility is also crucial for reconfiguring and redesigning organizations, and consequently reallocating resources. Functional walls must be quite portable, if they exist at all.

Discretion is what transforms flexibility into speed. Distributed organizations must be free to act. While they may have to conform to established protocols, or play by the rules of the game—at least the rules of the moment—they need to be able to avoid having to wait unnecessarily for permission to proceed. Similarly, they need to be able to alter courses of action, or pull the plug, when things are not working. In this way, resources are deployed quickly and results are monitored just as quickly. Resource investments that are not paying off in the expected time frame can be quickly redeployed elsewhere.

A major determinant of these organizational abilities is the extent to which an organization possesses intellectual capital, or knowledge capital, such that it can create and use innovative ideas to produce productive results (28–30) and the ability to manage in a time of great change (31). We would add communications, collaboration, and courage to the formulation of Ulrick (32) representing intellectual capital to yield:

$$\begin{aligned}\text{Intellectual capital} = {} & \text{Competence} \times \text{Commitment} \\ & \times \text{Communications} \times \text{Collaboration} \\ & \times \text{Courage}\end{aligned}$$

What matters very much today is the ability to make sense of market and technology trends, to quickly decide how to take advantage of these trends, and to act faster than competitors, or other players. Sustaining competitive advantage requires redefining market-driven value propositions and quickly leading in providing value in appropriate new ways. Accomplishing this in an increasingly information-rich environment is a major challenge, both for organizations experiencing these environments and for those who devise and provide systems engineering and management methods and tools for supporting these new ways of doing business. Thus we see that the network age of information and knowledge is an integrated age in which the mutually reinforcing influences of information technology, organizations, and people have led to major challenges and major opportunities in which the new integrated system takes on global proportions.

A key issue now is how systems engineering and management can support business, government, and academia to address these major issues associated with productive use of information technology innovations in a successful manner. The remainder of this article discusses ten challenges, discussed in more depth in Ref. (33), that this integrative discipline must pursue and resolve if we are to support continued progress through information technology. Addressing these challenges will require much continued effort.

1. *Systems Modeling*
   Our methods and tools for modeling, optimization, and control depend heavily on exploiting problem structure. However, for loosely structured systems, behavior does not emerge from fixed structures. Instead, structure emerges from collective behaviors of agents. The distributed, collaborative, and virtual organizations that information technology enables are such that the system elements are quite fluid. Distinctions between what is inside and outside the system depend on time-varying behaviors and consequences. Satisfying this need will significantly challenge typical modeling methods and tools.
2. *Emergent and Complex Phenomena*
   Meeting this modeling challenge is complicated by the fact that not all critical phenomena cannot be fully understood, or even anticipated, based on analysis of the decomposed elements of the overall system. Complexity not only arises from there being many elements of the system, but also from the possibility of collective behaviors that even the participants in the system could not have anticipated (34). An excellent example is the process of technological innovation. Despite the focused intentions and immense efforts of the plethora of inventors and investors attracted by new technologies, the ultimate market success of these technologies almost always is other than what these people expect (35) and new technologies often cause great firms to fail (36). In other words, many critical phenomena can only be studied once they emerge and the only way to identify such phenomena is to let them happen, or to create ways to recognize the emergence of unanticipated phenomena through modeling and simulation. An important emergent phenomena is that of path dependence (37). The essence of this phenomenon begins with a supposedly minor advantage or inconsequential head start in the marketplace for some technology, product, or

standard. This minor advantage can have important and irreversible influences on the ultimate market allocation of resources even if market participants make voluntary decisions and attempt to maximize their individual benefits. One of the potential characteristics of information technology products, especially software, is that of increasing returns to scale. Also, there may be a network effect, or "network externality," which occurs because the value of a product for an individual consumer may increase with increased adoption of that product by other consumers and this, in turn, raises the potential value for additional users. An example is the telephone, which is only useful if at least one other person has one, and for which the utility of the product becomes increasingly higher as the number of potential users increases. These are major issues concerning information technology today and are at the heart of the concern in the late 1990s with respect to the Windows operating system and Internet Explorer, each from Microsoft.

3. *Uncertainties and Control*
   Information technology enables systems in which the interactions of many loosely structured elements can produce unpredictable and uncertain responses that may be difficult to control. The challenge is to understand such systems at a higher level and success in doing this may depend much more on efficient experimentation and simulation than on mathematical optimization due to the inherent complexities that are involved.
4. *Access and Utilization of Information and Knowledge*
   Information access and utilization, as well as management of the knowledge resulting from this process, are complicated in a world with high levels of connectivity and a wealth of data, information, and knowledge. Ubiquitous networks and data warehouses are touted as the information technology based means to taking advantage of this situation. However, providers of such "IT solutions" seldom address the basic issue of what information users really need, how this information should be processed and presented, and how it should be subsumed into knowledge that reflects context and experiential awareness of related issues. The result is an effort to obtain massive amounts of data and information, and associated large investments in information technology with negligible improvements of productivity (38). One of the major needs in this regard is for organizations to develop the capacity to become learning organizations (39,40) and to support bilateral transformations between tacit and explicit knowledge (41). Addressing these dilemmas should begin with the recognition that information is only a means to gaining knowledge and that information must be associated with the contingency task structure to become knowledge. This knowledge is the source of desired advantages in the marketplace or versus adversaries. Thus understanding and supporting the transformations from information to knowledge to advantage are central challenges to enhancing information access and utilization in organizations. This requires the contingency task structure of experiential familiarity with previous situations and understanding the environmental context for the present situation.
5. *Information and Knowledge Requirements*
   Beyond adopting a knowledge management perspective, one must deal with the tremendous challenge of specifying information requirements in a highly information-rich environment. Users can have access to virtually any information they want through modern information technology, regardless of whether they know what to do with it or how to utilize the information as knowledge. While information technology has evolved quite rapidly in recent decades, human information processing abilities have not evolved significantly (8). These limited abilities become bottlenecks as users attempt to digest the wealth of information they have requested. The result is sluggish and hesitant decision making, in a sense due to being overinformed.
6. *Information and Knowledge Support Systems*
   Information technology based support systems, including decision support and expert systems, can potentially provide the means to help users cope with information-rich environments. However, these support systems are difficult to define, develop, and deploy in today's highly networked and loosely structured organizations due to inherently less well-defined contexts and associated tasks and decisions. Effective definition, development, deployment, and use of these systems are also complicated by continually evolving information sources and organizational needs to respond to new opportunities and threats. In part, these difficulties can be overcome by adopting a human-centered approach to information and knowledge support system design (42,43). This approach begins with understanding the goals, needs, and preferences of system users and other stakeholders—for example, system maintainers. This approach focuses on stakeholders' abilities, limitations, and preferences, and attempts to synthesize solutions that enhance abilities, overcome limitations, and foster acceptance. From this perspective, information technology and alternative sources of information and knowledge are enablers rather than ends in themselves.
7. *Inductive Reasoning*
   Prior to the development of agent-based simulation models and complexity theory, most studies involved use of linear models and assumed time-invariant processes (i.e., ergodicity). Most studies also assumed that humans use deductive reasoning and techno-economic rationality to reach conclusions. But information imperfections and limits on available time often suggest that rationality must be bounded. Other forms of rationality and inductive reasoning are necessary. We interpret knowledge in terms of context and experience by sensing situations and

recognizing patterns. We recognize features similar to previously recognized situations. We simplify the problem by using these to construct internal models, hypotheses, or schemata to use on a temporary basis. We attempt simplified deductions based on these hypotheses and act accordingly. Feedback of results from these interactions enables us to learn more about the environment and the nature of the task at hand. We revise our hypotheses, reinforcing appropriate ones and discarding poor ones. This use of simplified models is a central part of inductive behavior (44).

8. *Learning Organizations*
Realizing the full value of the information obtained from an IT system, and the ability to interpret this as knowledge is strongly related to an organization's abilities to learn and its ability to become a learning organization. Learning involves the use of observations of the relationships between activities and outcomes, often obtained in an experiential manner, to improve behavior through the incorporation of appropriate changes in processes and products. Two types of organizational learning are defined by Argyris and Schon (40). Single-loop learning is learning which does not question the fundamental objectives or actions of an organization. It enables the use of present policies to achieve present objectives. The organization may well improve but this will be with respect to the current way of doing things. Organizational purpose, and perhaps even process, are seldom questioned. Often, they need to be. Double-loop learning involves identification of potential changes in organizational goals and approaches to inquiry that allow confrontation with and resolution of conflicts, rather than continued pursuit of incompatible objectives which usually leads to increased conflict. Double-loop learning is the result of organizational inquiry which resolves incompatible organizational objectives through the setting of new priorities and objectives. New understanding is developed which results in updated cognitive maps and scripts of organizational behavior. Studies show that poorly performing organizations learn primarily on the basis of single-loop learning and rarely engage in double-loop learning in which the underlying organizational purposes and objectives are questioned. Peter Senge (39) has discussed extensively the nature of learning organizations. He describes learning organizations as "organizations where people continually expand their capacity to create the results they truly desire, where new and expansive patterns of thinking are nurtured, where collective aspiration is set free, and where people are continually learning how to learn together." Five component technologies, or disciplines, enable this type of learning: (1) systems thinking; (2) personal mastery through proficiency and commitment to lifelong learning; (3) shared mental models of the organization markets, and competitors; (4) shared vision for the future of the organization; and (5) team learning. Systems thinking is denoted as the "fifth discipline." It is the catalyst and cornerstone of the learning organization that enables success through the other four dimensions. Lack of organizational capacity in any of these disciplines is called a learning disability. It is important to emphasize that the extended discussion of learning organizations in this section is central to understanding how to create organizations that can gain full benefits of information technology and knowledge management. This is crucial if we are to transform data to information to insights to meaningful and effective programs of action.

9. *Planning*
Dealing successfully with the above challenges requires that approaches to planning and its effect on the other systems life cycles be reconsidered. Traditionally, planning is an activity that occurs before engineering production and before systems are placed into operation. However, for loosely structured systems, systems planning must be transformed to something done in an interactive and integrative manner that considers each of the other life-cycles.

10. *Measurement and Evaluation*
Successfully addressing and resolving the many issues associated with the information technology challenges described in this article requires that a variety of measurement challenges be understood and resolved. Systems associated with access and utilization of information, and knowledge management, present particular measurement difficulties because the ways in which information and knowledge affect behaviors are often rather indirect. For this and a variety of related reasons, it can be quite difficult to evaluate the impact of information technology and knowledge management. Numerous studies have failed to identify measurable productivity improvements as the result of investments in these technologies. The difficulty is that the impact of information and knowledge is not usually directly related to numbers of products sold, manufactured, or shipped. Successful measurement requires understanding the often extended causal chain from information to knowledge to actions and results. Transformations from information to knowledge also present measurement problems. Information about the physical properties of a phenomena are usually constant across applications. In contrast, knowledge about the context-specific implications of these properties depends on human intentions relative to these implications. Consequently, the ways in which information is best transformed to knowledge depends on the intentions of the humans involved. The overall measurement problem involves inferring—or otherwise determining—the intentions of users of information technology based systems, both products and processes.

CONCLUSION

Ongoing trends in information technology and knowledge management pose substantial challenges for electrical and electronics engineering as implementation technologies enable enhanced physical system capabilities. Addressing the ten key challenges elaborated here requires utilizing many of concepts, principles, methods, and tools discussed elsewhere in this encyclopedia, especially those associated with systems engineering and management. In addition, it requires a new, broader perspective on the nature of information access and utilization, as well as knowledge management. Despite these challenges, there are many potential applications of information technology. Such studies as that on integrated manufacturing (45) and the role of information technology in conflict situations (46) are continually appearing. That this subject is a most relevant one for electrical engineering today can hardly be overemphasized.

BIBLIOGRAPHY

1. D. Bell, *The Coming of Post Industrial Society*, New York: Basic Books, 1973.
2. A. Toffler, *The Third Wave*, New York: Morrow, Bantam Books, 1980, 1991.
3. A. Toffler and H. Toffler, *Creating a New Civilization*, Atlanta: Andrews and McNeel, 1995.
4. S. Zuboff, *In the Age of the Smart Machine: The Future of Work and Power*, New York: Basic Books, 1988.
5. D. Tapscott, *Growing up Digital: The Rise of the Net Generation*, New York: McGraw-Hill, 1998.
6. E. Dyson, *Release 2.0: A Design for Living in the Digital Age*, New York: Broadway Books, 1997.
7. J. Hope and T. Hope, *Competing in the Third Wave: The Ten Key Management Issues of the Information Age*, Boston: Harvard Bus. School Press, 1997.
8. E. Brynjolfsson and S. Yang, Information Technology and Productivity: A Review of the Literature, in M. Yovitz, ed., *Advances in Computers*, New York: Academic Press, 1996, pp. 179–214.
9. National Research Council, *More than Screen Deep: Toward Every-Citizen Interfaces to the Nation's Information Infrastructure*, Washington, DC: Nat. Academy Press, 1997.
10. M. Dertouzos, *What Will Be: How the New World of Information Will Change Our Lives*, New York: HarperCollins, 1997.
11. R. Katz (ed.), *The Human Side of Managing Technological Innovation*, London: Oxford Univ. Press, 1997.
12. D. S. Alberts and D. S. Papp (eds.), *Information Age Anthology: Information and Communication Revolution*, Washington, DC: National Defense Univ. Press, 1997.
13. R. Pool, *Beyond Engineering: How Society Shapes Technology*, London: Oxford Univ. Press, 1997.
14. R. R. Schaller, Moore's law: Past, present, and future, *IEEE Spectrum*, **34** (6): 52–59, 1997.
15. R. Grover and L. Himelstein, All the news that's fit to browse, *Business Week*, June 16, 1997, pp. 133–134.
16. D. Shenk, *Data smog: Surviving the info glut*, San Francisco, CA: Harper, 1998.
17. K. Kelly and G. Wolf, PUSH! Kiss Your Browser Goodbye: The Radical Future of Media Beyond the Web, *Wired*, **5** (3), March 1997.
18. W. Andrews, Agent makers expand into push in effort to deliver increasingly relevant information to users, *WebWeek*, **3** (14): 1, 1997.
19. P. Maes, Agents that reduce work and information overload, *Comm. ACM*, **37** (7): 30–40, 146, 1994.
20. A. Caglayan and C. Harrison, *Agent Sourcebook: A Complete Guide to Desktop, Internet, and Intranet Agents*, New York: Wiley, 1997.
21. R. S. Kaplan and R. Cooper, *Cost and Effect: Using Integrated Cost Systems to Drive Profitability and Performance*, Boston: Harvard Bus. School Press, 1997.
22. R. Cooper and R. S. Kaplan, *The Design of Cost Management Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1991.
23. S. P. Bradley and R. L. Nolan, *Sense and Respond: Capturing Value in the Network Age*, Boston: Harvard Bus. School Press, 1998.
24. D. Tapscott and A. Caston, *Paradigm Shift: The New Promise of Information Technology*, New York: McGraw-Hill, 1993.
25. W. B. Rouse, Connectivity, Creativity, and Chaos: Challenges of Loosely-Structured Organizations, *Proc. 1997 Int. Conf. on Syst. Man, Cybern.*, Orlando, FL, October 1997.
26. R. K. Lester, *The Productive Edge: How U.S. Industries Are Pointing the Way to a New Era of Economic Growth*, New York: Norton, 1998.
27. P. F. Drucker, Toward the new organization, *Leader to Leader*, **3**: 6–8, 1997.
28. A. Brooking, *Intellectual Capital: Core Asset for the Third Millennium Enterprise*, London: Int. Bus. Press, 1996.
29. L. Edvinsson and M. S. Malone, *Intellectual Capital: Realizing Your Company's True Value by Finding Its Hidden Brainpower*, New York: HarperCollins, 1997.
30. D. A. Klein, *The Strategic Management of Intellectual Capital*, Boston: Butterworth-Heineman, 1998.
31. P. F. Drucker, *Managing in a Time of Great Change*, New York: Penguin, 1995.
32. D. Ulrich, Intellectual capital = Competence × commitment, *Sloan Manag. Rev.*, **39** (2): 15–26, 1998.
33. W. B. Rouse and A. P. Sage, Information Technology and Knowledge Management, in A. P. Sage and W. B. Rouse (eds.), *Handbook of Systems Engineering and Management*, New York: Wiley, 1999.
34. J. L. Casti, *Would-Be Worlds: How Simulation Is Changing the Frontiers of Science*, New York: Wiley, 1997.
35. J. Burke, *The Pinball Effect: How Renaissance Water Gardens Made the Carburetor Possible and Other Journeys Through Knowledge*, Boston: Little, Brown, 1996.
36. C. M. Christensen, *The Innovator's Dilemma: When New Technologies Cause Great Films to Fail*, Boston: Harvard Bus. School Press, 1997.
37. W. B. Arthur, *Increasing Returns and Path Dependence in the Economy*, Ann Arbor, MI: Univ. Michigan Press, 1994.
38. D. H. Harris (ed.), *Organizational Linkages: Understanding the Productivity Paradox*, Washington, DC: National Academy Press, 1994.
39. P. M. Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization*, New York: Doubleday, 1990.
40. C. Argyris and A. Schon, *Organizational Learning II: Theory, Method, and Practice*, Reading, MA: Addison-Wesley, 1996.

41. I. Nonaka and H. Takeuchi, *The Knowledge Creating Company*, London: Oxford Univ. Press, 1995.

42. W. B. Rouse, *Design for Success: A Human-Centered Approach to Designing Successful Products and Systems*, New York: Wiley, 1991.

43. W. B. Rouse, Human-Centered Design of Information Systems, in J. Wesley-Tanaskovic, J. Tocatlian, and K. H. Roberts (eds.), *Expanding Access to Science and Technology: The Role of Information Technology*, Tokyo: United Nations Univ. Press, 1994, pp. 214–223.

44. J. H. Holland et al., *Induction: Processes of Inference, Learning, and Discovery*, Cambridge, MA: MIT Press, 1986.

45. *Information Technology for Manufacturing: A Research Agenda*, Natl. Res. Council, Natl. Acad. Press, Washington, DC, 1995.

46. J. Arquilla and D. Rondfeldt, *In Athena's Camp: Preparing for Conflict in the Information Age*, Santa Monica, CA: RAND, 1997.

Andrew P. Sage
George Mason University
Fairfax, Virginia
William B. Rouse
Georgia Institute of Technology
Atlanta, Georgia

KERNEL

The kernel is the subset of an operating system's functions that must be executed in privileged or supervisor mode, so that they might properly protect users and allocate resources among them. The privileged mode gives those programs unrestricted memory access and the right to executive sensitive instructions. Kernel functions typically include:

1. **Basic Input/Output System (BIOS).** Routines giving a process access to the essential I/O devices connected to the system—the display, keyboard, mouse, hard disks, and network. BIOS starts those devices and responds to their interrupts when they are done.
2. **Interrupt Management.** Routines for transferring the processor to a high priority task shortly after a signal indicates the need for the task. The names of interrupt handling routines are in an "interrupt vector." The actual routines are elsewhere; for example, the page fault handler is part of the memory manager (see below), and the network packet received handler is part of the interprocess communication manager (see below).
3. **Process Management.** Routines for switching processors among processes; using the clock interrupt for round-robin scheduling; sending semaphore signals among processes; creating and removing processes; and controlling entry to supervisor state.
4. **Memory Management.** Routines for mapping virtual to real addresses; transferring pages between main memory and secondary memory; controlling multiprogramming load; and handling page fault and protection violations.
5. **Interprocess Communication.** Routines for opening and closing connections between processes; transferring data over open connections; handling network protocols and routing; and processing name service.
6. **Security.** Routines for enforcing the access and information-flow control policies of the system, changing protection domains, and encapsulating programs.

The kernels of early operating systems were constrained by memory limitations to be small. MIT's Compatible Time Sharing System (1960) had a total of 256 kilobytes of main memory, of which half was allocated to the kernel. Early Unix systems (1972) also had small kernels. By the early 1980s, system such as Berkeley Unix grew their kernels to take advantage of larger main memory. However, large kernels were more error prone, less secure, and slow.

Operating systems designers introduced the microkernel as a way to minimize the operating system code that had to be main-memory resident and operate in supervisor mode. Only the bare essentials were included; all other routines were executed outside the kernel in the user's memory space. For example, a device driver can be executed in user mode; only the instructions that invoke it and receive its interrupts are privileged. Also, file access control lists are stored in the file system; the microkernel security manager simply verifies that a requested access is permitted by the access list. This strategy has led to very efficient, fast, and compact microkernels for many operating systems.

Peter J. Denning
Naval Postgraduate School
Monterey, California

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

M

MONITOR

The term *monitor* denotes a control program that oversees the allocation of resources among a set of user programs. Along with *supervisor* and *executive*, it was an early synonym for operating system. An old example was the Fortran Monitor System (FMS), which appeared on the IBM 709 series beginning in the late 1950s to provide run-time support for Fortran programs. A later example was the Conversational Monitor System (CMS), a single-user interactive system that ran on a virtual machine within the IBM VM/370. Beginning in the early 1970s, the term *monitor* was slowly dissociated from operating systems and associated with object-class managers. The first such use was in secure operating systems: The *reference monitor* was an object-class manager that enforced an access policy among the users of those objects (1).

In 1974, Hoare (2) proposed programming language structures that simplified operating systems by providing a separate scheduler for each class of resources. The monitor maintains records of queues and resource states, and it gives external processes access through a high-level interface consisting of monitor procedures (today called methods). As with abstract data types, the monitor's methods restrict the caller to a few well-defined operations on the monitor's resources, hiding most details. Unlike abstract data types, monitors have internal locks that permit only one process to execute monitor instructions at a time, thereby permitting concurrent operations on the objects. Other processes must wait in a queue to enter the monitor. If a process in the monitor stops to wait for a resource to become available, the monitor must be unlocked so that another process (notably one that will release the desired resource) can gain access.

An example of a resource monitor is given below in Java, a modern language that supports the concept. With this monitor, a process can request that it be allocated a unit from a pool of resources; it can use that unit exclusively for a period of time, and then release it back to the pool. To set this up, a programmer includes these statements to declare a monitor for printers and to initialize it with a vector containing the names of all available printers:

```
ResourceMonitor printers;
printers = new ResourceMonitor(...);
```

Thereafter, the programmer can cause a thread (process) to acquire, use, and release a printer with this pattern:

```
myPrinter = printers.acquire();
  ...
  (use of printer)
  ...
  printers.release(myPrinter);
```

The Java code for the resource monitor is as follows:

```
class ResourceMonitor{
   Vector pool;
   /* initialization */
    ResourceMonitor(Vector initialResources){
        pool = initialResources;
    }
   /* acquire an available resource; wait if none
     available */
    public synchronized Object acquire(){
        Object out;
        while (pool.isEmpty()) wait();
        out = pool.firstElement();
        pool.removeElement(out);
        return out;
    }
   /* return a resource to the pool */
    public synchronized void release(Object in){
        pool.addElement(in);
        notify();
    }
   }
```

The methods `acquire()` and `release()` are declared as *public*, meaning that any process can call them, and *synchronized*, meaning that they are executed atomically. This implements the central feature of a monitor, which is that the operations are mutually exclusive and that their component steps cannot be interleaved. This feature is implemented by locks inserted by the compiler.

The Java operation `wait()` suspends the calling thread (process), releases the mutual exclusion lock on the methods (acquire, release), and records the process identifier in a waiting queue; `notify()` signals one of those. The signaled thread is removed from the queue and resumes its operation after relocking the monitor. Note that in this example, the call to `wait()` appears in a loop; it is possible that the thread may find the pool empty by the time it reenters the monitor; in which case, it will call `wait()` again. This can occur if another thread overtakes the signaled one.

Monitors are also useful in structuring distributed systems. The monitor operations can be called via remote procedure calls (RPCs). Most RPC mechanisms have built-in error controls to guard against lost messages; a lost message could cause a deadlock when a thread called the release method but the monitor did not receive the release message.

A monitor callable via RPC must be organized to avoid deadlock when network connections fail. When a remote call message arrives at the monitor, the method to be executed is given to a "proxy thread" running on the server housing the monitor. Meanwhile, the client application thread that issued the RPC is suspended on its host computer until

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

the proxy thread completes and returns the result. Even if the connection or the caller's host crashes, the proxy thread will complete and unlock the monitor, allowing other threads on other computers to call the monitor.

The programmer of the application that called the remote monitor sees some complications. They originate from the dynamic bindings between the programmer's computer and the server housing the monitor. The complications show up as new exceptions from the RPC system—e.g., unknown host, failed or hung connection, parameter marshaling error, no such remote procedure, no such object, and no response within the time-out period. The responses to these exceptions will depend on the application.

BIBLIOGRAPHY

1. P. Denning and G. S. Graham. Protection: principles and practice, *Proc. AFIPS Spring Joint Comput. Conf.*, **40**, 417–429, 1972.
2. C. A. R. Hoare, Monitors: An operating system structuring concept, *Commun. ACM*, **17**(10), 1974.

FURTHER READING

P. Brinch Hansen, *The Architecture of Concurrent Programs*. Englewood Cliffs, NJ: Prentice-Hall, 1977.

K. Arnold and J. Gosling, *The Java Programming Language*. Reading, MA: Addison-Wesley, 1996.

Peter J. Denning
Naval Postgraduate School
Monterey, California

Walter F. Tichy
University of Karlsruhe
Karlsruhe, Germany

OVERHEAD

Overhead in computer systems is like overhead in organizations—shared functions that benefit everyone but that cannot be conveniently associated with any one activity. In organizations, rent, furnishings, electricity, telephones, utilities, supplies, auditing, accounting, general clerical support, and management are counted as overhead. In computer systems, allocation of resources, scheduling, conflict resolution, error correction, exceptional conditions, protection, security, performance monitoring, auditing, accounting, input–output control, caching, distributed functions, and network protocols are all counted as overhead. As in organizations, excessive overhead diminishes capacity and increases cost without increasing productivity. Overhead in computer systems manifests as slower processing, less memory, less network bandwidth, or bigger latencies than expected.

Overhead is not always easy to measure. The time an operating system spends in supervisor state is not pure overhead because many important operations requested by user tasks are implemented as system functions that run in supervisor state, for example, input–output, file operations, and message-passing. A measurement that an operating system spends 80% of its time in supervisor state does not mean that the system spends only 20% of its time doing useful work: We need to know what portion of the 80% is spent responding to requests from user tasks. Moreover, many operating systems use special coprocessors to perform important overhead tasks such as authentication, virtual memory control, external communications, or peripheral device management; these coprocessors do not diminish processor capacity, memory capacity, or bandwidth available to user tasks. When coprocessors perform system functions, a measurement that a processor spends 90% of the time running user tasks does not mean that overhead is low.

Listed below are the main functions that usually count as overhead. Each one has a cost in processing time, memory space, network bandwidth, and latency (response time).

Allocation of Resources. Many resources such as CPU cycles, disk sectors, main-memory page frames, local-network packet slots, and shared files can be used by only one task at a time. To prevent conflicts and deadlocks, operating systems implement schedulers for these resources. The time spent running a scheduler and the memory occupied by a scheduler's queues count as overhead.

Error Correction. Data are stored and transmitted with redundant bits that permit detecting and correcting errors. These bits consume some space and bandwidth.

Exceptional Conditions. Most system functions have normal and error returns; the instructions that test for and respond to errors consume some space and processing time. Examples are arithmetic contingencies, data transmission failures, addressing snags, and illegal actions.

Protection and Security. Monitors, firewalls, authenticators, backup systems, virus detectors, and other means of securing systems against unauthorized use, denial of service, and intruders are necessary but often expensive.

Performance Monitoring, Auditing and Accounting. Recording key actions and events, logging each task's usage of resources, figuring costs and billings to users of the system, and generating statistics on resource usage and performance cost.

Input–Output Control. Many I/O operations are easy to specify at the user level—for example, open or read a file. But the device spoolers and drivers can be complex because they must queue up requests from multiple tasks, translate each request into the low-level instruction sets of the devices, automatically work around known problems such as bad disk sectors, and handle interrupt conditions from their devices.

Caching. The speed of operations on secondary storage devices or remote servers can often be significantly improved by keeping a copy of the data in a local memory. Microcomputer register-windows, virtual memories, disk drivers, open-file managers, network browsers, and Internet edge servers are among the many prominent examples of caching. Caching consumes memory and processing time to locate and load copies of items into the cache and to maintain consistency with the originals.

Distributed Functions. Modern operating systems distribute their functions transparently over a collection of servers and workstations connected by a high-speed local network. Examples are file servers, printing servers, compute servers, authentication servers, and workstations. Maintaining the appearance that files, printers, processors, and login-sites are location independent significantly improves usability but is not cheap.

Network Protocols. Protocols for opening connections, transferring data, obtaining encryption keys, routing, and authenticating access all cost processing time, memory, and bandwidth.

Mainframe operating systems often charge users for processor, memory, and disk usage. In these systems, overhead will be charged back to users as percentage increases in each of these components. In personal computer networks and time-shared research computing systems, users are not charged for resource usage or overhead.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Overhead detracts from system performance only to the extent that the overhead functions do not add to the productivity of user tasks. Many services are provided by the system to relieve programmers from having to provide these functions themselves or to prevent expensive breakdowns. As long as the system can provide these functions more efficiently than its users, the resulting increases in overhead are offset by better service, improved performance, and lower overall costs.

PETER J. DENNING
Naval Postgraduate School
Monterey, California

QUALITY IN COMPUTER SCIENCE AND COMPUTER ENGINEERING EDUCATION

INTRODUCTION

Any attempt to define the concept of quality, as it exists in higher education, tends to reveal the fact that in some sense it is elusive; on the other hand, "you know it when you see it". In his article, Peter Knight (1) articulates some of the dilemmas such as efficiency or effectiveness, emphasis on measurement or emphasis on process, and well-specified procedures or well-rehearsed goals. In some sense, one must achieve a proper balance. Of necessity, students themselves must be active, engaged, with and committed to the educational process. Given the current recognition of the phenomenon of the globalization of the workforce, it is very important for students to be competitive in the international workplace by the time they graduate.

The approaches to quality are many and varied, with different mechanisms and approaches. In turn these approaches produce different emphases, and inevitably, these tend to condition the behavior of those involved. Invariably, the mechanisms should place an emphasis on improvement or enhancement, although judgements inevitably have some role when it comes to assessing quality either through accreditation or through some other process.

The purpose of this article is to focus on the quality issues within computer science and computer engineering degree programs. In some countries, program quality is fostered through an accreditation process, whereas in others, slightly different approaches are employed. Here we attempt to span that spectrum.

THE CONCEPT OF QUALITY

Many definitions of the term *quality* exist as it applies in the context of higher education. Even in the context of individual programs of study, Different ways exist to define in the concept. In the United Kingdom, for instance, many different interpretations exist. Earlier definitions proved excessively complex to manage and to operate. A relatively recent document outlines certain principles that auditors should use during the review of programs of study (see Ref. 2). In this document, only seven aspects are most important, which include the following:

1. Aims and outcomes. What are the intended learning outcomes of the program and how were these obtained (e.g., from guidance from benchmarking standards, professional body requirements, local needs)? How do these out comes relate to the overall aims of the provision? Are staff and students familiar with these outcomes?
2. Curricula. Does the design and content of the curriculum encourage achievement of the full range of learning outcomes? Do modern approaches to learning and teaching by current developments and scholarship influence the discipline?
3. Assessment. Does the assessment process enable students to demonstrate acquisition of the full range of intended learning objectives? Does criteria exist to define different levels of achievement? Are full security and integrity associated with the assessment processes? Does the program require formative as well as summative assessment? Does the program meet benchmarking standards?
4. Enhancement. Does the program use an activity that seeks to improve standards regularly, (e.g., via internal or external reviews, appropriate communication with the external examiner(s), accreditation activity) and how deep and thorough is that activity? Are data analyzed regularly and are appropriate actions taken?
5. Teaching and learning. Are the breadth, depth, and challenge of the teaching of the full range of skills as well as the pace of teaching appropriate? Does the program implement a suitable variety of appropriate methods? If so, do these methods truly engage and motivate the students? Are the learning materials of high quality? Are the students participating in learning?
6. Student progression. Does the program have an appropriate strategy for academic support? Is admissions information clear and does it reflect the course of study faithfully? Are supervision arrangements in place? Do students receive appropriate induction throughout their course?
7. Learning resources. Are the teaching staff appropriately qualified to teach the given program of study and do they have the opportunity to keep teaching materials as well as their competences up-to-date? is effective support provided in laboratories and for practical activity? How does the institution use resources for the purposes of learning? Is the student working space attractive and is the general atmosphere in the department conducive to learning?

PARTICULAR APPROACHES

The U.S. Situation—ABET and CC2001

Within the United States, ABET, Inc. (formerly known as the Accreditation Board for Engineering and Technology, established in 1932) undertakes accreditation in the field of applied sciences, computing, engineering, and technology. In this capacity, it undertakes the accreditation of individual programs of study. It recognizes accreditation that is a process whereby programs are scrutinized to ascertain whether they meet quality standards estab-

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

lished by the professions. The view is that accreditation benefits:

- Students who choose programs of study with careers in mind
- Employers to enable them to recognize graduates who will meet these standards
- Institutions that are provided with feedback and even guidance on their courses.

Within ABET, CSAB, Inc. (formerly known as the Computing Sciences Accreditation Board) is a *participating body* and it is the lead society to represent programs in computer science, information systems, software engineering, and information technology. It is also a cooperating society (with the IEEE as the lead society) that represents programs in computer engineering. However, within ABET, its computing accrediting commission (CAC) is responsible to conduct the accreditation process for programs in computer science, information systems, and information technology. engineering accreditation commission (EAC) is responsible for the accreditation of programs in software engineering and in computer engineering.

Criteria for Accrediting Computer Science Programs. The ABET criteria for computer science programs may be found in Ref. 3. What follows is based directly on that publication.

Each program of study will possess some published overall set of objectives that aim to capture the philosophy of the program. Typically, these objective might address now students are prepare for a variety of possible careers in computer science or possible advanced study in that field.

A program of study, as defined by the individual modules, must be consistent with these objectives. Beyond this, the program requires additional elements phrased in terms of a general component, a technical computer science component, a mathematics and science component, and a component that addresses additional areas of study. The requirements themselves stipulate minimum times to devote to particular areas and issues that rate to content.

To quantify the time element, a typical academic year consists of two semester terms (though three quarter terms are also possible). A semester consists of 15 weeks and one semester hour is equivalent to meeting a class for 50 minutes one time a week for 15 weeks; that is, it is equivalent to 750 minutes of class time during a semester. Students take courses minimally worth 30 semester hours per year for four years; this is equivalent to 120 semester hours for a baccalaureate degree.

General Component. The general component of the accreditation criteria stipulates that a degree in computer science must contain at least 40 semester hours of appropriate and up-to-date topics in computer science. In addition, the degree program must contain 30 semester hours of mathematics and science, 30 hours of study in the humanities, social sciences, and the arts to provide a broadening education.

Computer Science Component. Specifically, the study of computer science must include a broad-based core of computer science topics and should have at least 16 semester hours; the program should have at least 16 hours of advanced material that builds on that core. Core material must cover algorithms, data structures, software design, concepts in programming languages, and computer architecture and organization. The program must stress theoretical considerations and an ethos of problem solving and design must exist. In addition, students require exposure to a variety of programming languages and systems and they must become proficient in at least one high-level programming language.

Mathematics and Science Component. The program must contain at least 15 semester hours of mathematics; the mathematical content must include discrete mathematics, differential and integral calculus, as well as probability and statistics. In addition, the program must contain at least 12 semester hours of science to include an appropriate two-semester sequence of a science course.

Additional Areas of Study. Oral and written communication skills must be well developed and be applied throughout the program. Additionally, students must be knowledgeable about legal, ethical, and professional issues. Beyond these matters of a technical nature, the program is also required to have laboratories and general facilities of a high standard; moreover, the institution must have sufficient resources (interpreted in the broadest sense) to provide a good environment that is aligned with the objectives of the program.

Criteria for Accrediting Computer Engineering Programs. The ABET criteria for computer engineering programs can be found in Ref. 4. What follows is based directly on that publication.

The criteria here take a different form from those outlined above for computer science; they are phrased in terms of general criteria that are applicable to all engineering programs followed by criteria that are specific to computer engineering. Thus, the general criteria mention students, program educational objectives, associated outcomes and assessment, and a professional component. We now describe these four areas.

Students. The quality and performance of students is an important measure of the health of the program. Of course, appropriate advice and guidance must be available to enhance that quality. Mechanisms must be in place to ensure that all students meet the stated learning objectives of the program. Firther more, it is imperative that institutions not only have policies to accept students from elsewhere (and recognizing credit gained) but also to validate courses taken in other institutions.

Program Educational Objectives. The criteria indicate that detailed published objectives (validated and reviewed at regular intervals) must exist for each program of study and these objectives must be consistent with the mission of the institution. The educational programs must provide classes whose successful completion implies that the broad program meets its aims and objectives.

Program Outcomes and Assessment. This aspect of the criteria relates to the expected profile of a student when program of study is completed successfully. Thus, successful completion of the individual classes should produce a graduate who possesses a range of skill or attributes. These attributes include appropriate knowledge and skills, the ability to design and conduct experiments, the ability to work in multi-disciplinary teams, the ability to communicate effectively, the understanding of professional and ethical issues, and the ability to apply their skills to undertake effective engineering practice.

Professional Component. This aspect relates to the inclusion of one year of a combination of mathematics and basic science, one and a half years of relevant engineering topics, and a general education program that complements the technical aspects of the program.

Additional Criteria. Beyond the previous criteria, additional requirements exist concerning the setting in which the program is based. Thus, the teaching staff must have an appropriate background, the facilities must be of high quality, and the institution must have the general resources to provide an environment in which successful and fruitful study can take place.

The specific computer engineering criteria include the requirement that the program should exhibit both the breadth and depth across a range of engineering topics in the area of the program of study. This range must include knowledge of probability and statistics, mathematics, including the integral and differential calculus, computer science, and the relevant engineering sciences needed to design and develop complex devices.

Computing Curricula 2001. Within the ABET criteria, core, advanced topics, and so on are mentioned. To provide a meaning for these terms, several professional societies, such as the ACM and the IEEE Computer Society, have worked together to produce curriculum guidance that includes mention of these terms.

We refer to this recent guidance as *Computing Curricula 2001* (or CC 2001 for short). Earlier documents were published as single volumes and they tended to focus on computer science programs. The expansion in the field of computing resulted in an effort to culminate the publishing of five volumes that cover computer science, information systems, computer engineering, software engineering, as well as information technology (see Refs. 5–8). A sixth volume is intended to provide an overview of the entire field (see Ref. 9). These references should provide up-to-date guidance on curriculum development and extends to detailed outlines for particular programs of study and classes within these programs. The intent is that these works will receive updates at regular intervals (e.g., perhaps every five years) to provide the community with up-to-date information and advice on these matters.

The System in the United Kingdom

In the United Kingdom, the *benchmarking standards* capture important aspects of the quality of programs of study in individual disciplines. Representative groups from the individual subject communities have developed these standards. In the context of computing, the relevant document (included in Ref. 10) the Quality Assurance Agency (the government body with general responsibility for quality in higher education is published in the united kingdom by). This document also forms the basis for the accreditation of computing programs in the United Kingdom, an activity to ensure that institutions provide within their courses the basic elements or foundations for professionalism.

The benchmarking document in computing is a very general outline in the sense that it must accommodate a wide range of existing degree programs; at the time of its development, an important consideration was that it had not to stifle, but rather facilitate, the development of new degree programs. The approach adopted was not to be dogmatic about content but to place a duty on institutions to explain how they met certain requirements.

The benchmarking document contains a set of requirements that must be met by all honors degrees in computing offered by U.K. institutions of higher education. This document defines minimal criteria for the award of an honors degree, but it also addresses the criteria expected from an average honors student; in addition, it indicates that criteria should exist to challenge the higher achieving students (e.g., the top 10%) and it provides guidance on possible programs.

The benchmarking document addresses knowledge and understanding, cognitive skills, and practical and transferable skills; it views professional, legal, and ethical issues as important. It recognizes these aspects as interrelated delicately but intimately. Several aspects of the requirements contained within the benchmarking document require mentioning these because indicate requirements fundamental thinking.

First, all degree programs should include some theory that acts as underpinning and institutions are required to defend their position on this matter. The theory need not be mathematical (e.g., it might be based on psychology), but it will serve to identify the fundamentals on which the program of study is based. This requirement should guarantee some level of permanence to benefit the educational provision.

Furthermore, all degree programs must take students to the frontiers of the subject. Institutions must identify themes developed throughout the course of study, from the basics through to the frontiers of current knowledge. In addition, all students should undertake, usually in their final year, an activity that demonstrates an ability to tackle a challenging problem within their sphere of study and to solve this problem using the disciplines of the subject. The essence of this activity is to demonstrate an ability to take ideas from several classes or modules and explore their integration by solving a major problem.

Finally, from the point of view of the accreditation carried out by the British Computer Society (BCS), the standards impose some additional requirements beyond those of the benchmarking standard to allow for some specific interpretations. For instance, the major final year activity must be a practical problem-solving activity that involves implementation.

The Australian Approach

The Australian Universities Quality Agency, AUQA for short, has a key overarching role in relation to the quality in Australian Universities (and beyond). Its objectives that relate primarily to course accreditation appear in Ref. 11.

The objectives specify two main points. First, the need to arrange and to manage a system of periodic audits of the quality assurance arrangements of the activities of Australian universities, other self-accrediting institutions, and state and territory higher education accreditation bodies is specified. In addition, the domument sitpulates that institutions must monitor, review, analyze, and provide public reports on quality assurance arrangements in self-accrediting institutions, on processes and procedures of state and territory accreditation authorities, and on the impact of these processes on the quality of programs.

The AUQA places a great emphasis on the self-review carried out by each institution. This self review will use a set of internal quality processes, procedures, and practices that must be robust and effective, with an emphasis on improvement. These internal mechanisms should accommodate accreditation and other such activities carried out by professional bodies and must imply peer judgement.

The AUQA itself seeks to encourage improvement and enhancement through the publication of good practice, as well as providing advice, and where desirable, consultation. The possibility of inter-institutional discussion also exists.

The Swedish System

In 1995, the National Agency for Higher Education in Sweden initiated a set of quality audits of institutions of higher education. In this audit, institutions were required to produce a self-evaluation of their provision; to sustain a team visit of auditors; and to conduct meetings and discussions; After the visit, auditors would discuss their findings amongst themselves. Although this process had value, most have deemed it to be less than satisfactory because it failed to drill down to departmental activity (see Ref. 12).

In December 1999, the Swedish government passed a bill that placed the student at the very heart of quality and at the heart of developments in higher education. Since 2002, quality assessment activities would be conducted by the agency on a six-year cycle. The process would involve self-assessment, visits from peer review teams, and the publication of reports. Four main purposes in the new system (see Ref. 12) are as follows:

- Ensure that students are offered equivalent education of good quality, regardless of their home institution
- Ensure that assessments place a premium on improvement
- Provide both students and potential students with information regarding the quality of degree programs
- Allow stakeholders to make comparisons on the aspect of provision, which should include international comparisons

The new activity has improved the level of education greatly in Sweden.

ADDITIONAL OBSERVATIONS

Bloom's Taxonomy and Educational Objectives

The seminal work of Bloom et al. (13) has conditioned heavily current thinking on educational objectives. Since its publication in 1956, this area has attracted extensive interest. More recently, the preoccupation with standards of achievement and more general quality concerns have resulted in attempts to clarify and to reformulate the underlying principles. One such reference is Ref. 14, which forms the basis of the discussion here, see Table 1.

Table 1. The cognitive process dimension (from Ref. 14 with permission)

Process Categories	Definitions	Illustrations of Cognitive Processes
1. Remember	Retrieve knowledge from memory	Recognize, recall, define, describe, repeat
2. Understand	Construct meaning from	Interpret, give examples, place in already defined categories, summarize, infer, compare, explain, discuss, indicate, interpret, extrapolate
3. Apply	Perform a known procedure in a given situation	Execute, implement, illustrate, solve, demonstrate, measure
4. Analyze	Break material into constituent parts and determine the relationships between these parts and how they contribute to the whole	Discriminate, distinguish, organize, attribute, criticize, compare, contrast, examine, question, test
5. Evaluate	Make judgments based on criteria and on standards	Check, compare and contrast, critique, defend, estimate, judge, predict, select, assess, argue
6. Create	Put elements together to form a coherent and/or functional entity	Design, compose, construct, produce, plan, generate, innovate; introduce new classifications; introduce new procedures

The typical form of a learning objective is a verb (which implies an activity that provides evidence of learning and therefore enhanced behavior) applied to some area of knowledge. Txpically, The verbs capture what students must be able to do. The objective must imply the acquisition of cognitive skills such as the ability to carry out some activity, or the acquisition of some knowledge.

In its original formulation, Bloom's taxonomy paid little attention to the underlying knowledge. Yet, we can identify different types of knowledge. Within any discipline, of course, the community will classify knowledge as elementary and as advanced.

The Knowledge Dimension. We now recognize that different kinds of knowledge exist. In Ref. 14, four different kinds of knowledge are identified, and these are captured within Table 1.

The Cognitive Dimension. We base this on a set of levels that claim to identify key skills in an increasing order of difficulty. These skills have been refined over the years but again a recent version taken from Ref. 14 appears in Table 2.

Of course, one can apply learning objectives at different levels. For example, learning objectives can exist at the level of:

- A program of study—hence, program objectives
- A module or a class—hence, module or class objectives
- Some knowledge unit—hence, instructional objectives

These objectives range from the general to the particular. Their role and function are different at these levels. However, within a particular program of study, the attainment of the lower-level objectives should be consistent with, and indeed contribute to, the attainment of the higher-level objectives. A question is generated about consistency; that is, to determine whether the attainment of the lower-level learning objectives implies attainment of the higher-level objectives.

It is now widely recognized in higher education circles that these levels from Bloom's taxonomy are most effective when combined with levels of difficulty and the currency of the knowledge within the discipline.

Keeping Up-To-Date

Especially for the computing profession, keeping up-to-date is essentially an attitude of mind. It involves regular update of technical developments to ensure the maintenance of knowledge as well as skill levels at the forefront of developments. Doing this requires discipline and support. Where changes in attitude and/or changes of a fundamental kind are required, a period of intense study must take place.

Technological change can herald opportunities for creativity and innovation. This change can lead to advances that result in improvements such as greater efficiency, greater functionality, and greater reliability with enhanced products or even new products that emerge as a result. Novel devices and new ways of thinking are especially cherished.

Higher education must to prepare students for challenges and opportunities. We can achieve this through imaginative approaches to teaching as well as through imaginative assignments and projects. Promoting innovation and creativity is important in computing courses. The concept of a final-year project in the united kingdom—similar to a capstone project in U.S. parlance—is an important vehicle from this perspective. More generally, it provides the opportunity for students to demonstrate their ability to apply the disciplines and techniques of a program of study by solving a substantial problem. Such exercises should open up opportunities for the demonstration of novelty.

We can identify a number of basic strategies to ensure a stimulating and an up-to-date provision. First, the curriculum itself must be up-to-date, the equipment has to be up-to-date, and faculty need to engage in relevant scholarship. Teachers can highlight relevant references (textbooks, software, websites, case studies, illustrations, etc.) with the by

Table 2. The knowledge dimension (taken from Ref. 14 with permission)

Knowledge Types	Definitions	Knowledge Subtypes
Factual knowledge	Basic terminology and elements of the discipline	a) Terminology b) Specific details and elements
Conceptual knowledge	Inter-relationships among different elements of the discipline, structures and classifications, theories, and principles	a) Classifications and categories b) Principles and generalizations c) Theories, models, and structures
Procedural knowledge	Algorithms, methods, methodologies,techniques, skills, as well as methods of enquiry and processes	a) Subject-specific skills and algorithms b) Techniques and methods c) Criteria for selection of approach
Meta-cognitive knowledge	Knowledge of cognition in general and self-awareness in this regard	a) Strategic knowledge b) Knowledge about cognitive tasks, includinginfluences of context and constraints c) Self-knowledge

Table 3. (See Ref. 15)

Stage	Student	Instructor	Instructional Example
1	Dependent	Authority/coach	Lecturing, coaching
2	Interested	Motivator/guide	Inspirational lecture, discussion group
3	Involved	Facilitator	Discussion lead by instructor Who participates as equal
4	Self-directed	Consultant	Internships, dissertations, self-directed study group

identifying sources of up-to-date and interesting information. However, more fundamental considerations must be addressed.

We already mentioned that keeping up-to-date is essentially an attitude of mind. Institutions can foster this attitude by implementing new approaches to teaching and learning that continually question and challenge and by highlighting opportunities for advances. Teachers can challenge students by developing assessments and exercises that seek to explore new avenues. It is also essential to see learning as an aspect that merits attention throughout the curriculum. For instance, four stages are identified in Table 3 (15).

Basic Requirements of Assessment and Confidence Building

Certain basic requirements of assessment exist. Assessment must be fair and reliable, of course, and it should address the true learning objectives of the classes. In addition, however, appropriate assessment should be enjoyable and rewarding, but all too often institutions do not follow these geidelines.

Overall, numerous and different skills require assessment such as transferable skills, technical skills, and cognitive skills. An over-emphasis on assessment, however, can create an avalanche of anxiety and the normal trade-offs that are often "fun" become sacrificed, which makes the situation highly undesirable. Imaginative ways of assessing are needed and should be the subject of continual change and exploration whereby assessors can discover better and more enjoyable approaches.

Of course, self-assessment should be encouraged. This can take (at least) one of two forms, which depend on whether the student supplies the questions. However, where students formulate their own assessments, these tend to be useful at one level, but they tend not to challenge usefully or extend the horizons of individuals.

One of the important goals of higher education is to extend the horizons and the capabilities of students and to build up their confidence. That confidence should to be well-founded and based on what students have managed to accomplish and to achieve throughout their study. Accordingly, finding imaginative ways of assessing is important. Assessment should

- Seek to address a range of issues in terms of skills
- Build on the work of the class in a reasonable and meaningful manner
- Challenge students whose confidence is enhanced by successful completion of the work
- Extend the horizons of individuals
- Encourage attention to interesting and stimulating applications
- Address international issues of interest and relevance to the students and their future careers
- Encourage excellence through appropriate assessment guidance

As part of the preparation for such activity, students should understand what teachers expect of them and how they can achieve the higher, even the highest, grades.

CONCLUDING REMARKS

Ultimately, it is very important not to lose sight of the very real benefits gained when the teaching staff interacts with students. Thus, quality is what happens between students and the teachers of an institution. Often, aspects of bureaucracy can tend to mask this important observation, and metrics and heavy paperwork take over. Fundamentally, the quality of education is about:

- Teachers who have an interest in students have an interest in teaching well and in motivating their students so that they can be proud of their achievements, and they have an interest in their subject and its applications. This often means teachers will adapt to particular audiences without compromising important standards.
- Students who feel welcomed and integrated into the ways of their department, who enjoy learning, and whose ability to learn is developing so that the process becomes ever more efficient and effective find their studies stimulating and interesting, find the material of the program of study relevant and meeting their needs, and feel themselves developing and their confidence increasing.
- The provision of induction to give guidance on course options as well as guidance of what is expected in terms of achieving the highest standards in particular areas.
- An environment that is supportive with good levels of up-to-date equipment and resources, with case studies, and lecture material all being readily available to inform, stimulate, and motivate.

REFERENCES

1. P. T. Knight, The Achilles' Heel of Quality: the assessment of student learning, *Quality in Higher Educ.*, **8**(1): 107–115, 2002.

2. Quality Assurance Agency for Higher Education. *Handbook for Academic Review*, Gloucester, England: Quality Assurance Agency for Higher Education.
3. Computing Accreditation Commission, *Criteria for Accrediting Computing Programs*, Baltimore, MD: ABET, 2007.
4. Engineering Accreditation Commission, *Criteria for Accrediting Computing Programs*, Baltimore, MD: ABET, 2007.
5. E. Roberts and G. Engel (eds.), *Computing Curricula 2001: Computer Science*, Report of The ACM and IEEE-Computer Society Joint Task Force on Computing Curricula, Final Report, 2001
6. J. T. Gorgone, G. B. Davis, J. S. Valacich, H. Topi, D. L. Feinstein, and H. E. Longenecker Jr., *IS 2002: Model Curriculum for Undergraduate Degree Programs in Information Systems*, New York: ACM, 2002.
7. R. Le Blanc and A. Sobel et al., *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*, Piscataway, N.J, IEEE Computer Society, 2006
8. D. Soldan, et al., *Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, Piscataway, NJ: IEEE Computer Society, 2006
9. R. Shackelford, et al., *Computing Curricula 2005: The Overview Report*, New York: Association for Computing Machinery, 2006.
10. Quality Assurance Agency for Higher Education. *Academic Standards – Computing*, Gloucester, England: The Quality Assurance Agency for Higher Education, 2000.
11. Australian universities Quality Agency, AUQA The Audit Manual, Melbourne: Australian Universities Quality Agency, version 1, May 2002.
12. [Franke, 2002] S. Franke, From audit to Assessment: a national perspective on an international issue, *Quality in Higher Educ.*, **8**(1): 23–28, 2002.
13. B.S. Taxonomy of Educational Objectives: The Classification of Educational goals, Bloom, Handbook 1: Cognitive Domain, New York: Longmans, 1956.
14. L. W. Anderson, D. R. Kraathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayeer, P. R. Pintrich, J. Raths, M. C. Wittrock (eds.), *A Taxonomy for Learning, Teaching, and Assessing – A Revision of Bloom's Taxonomy of Educational Objectives*, Reading, MA: Addison Wesley Longman, Inc., 2001.
15. S. Fellows, R. Culver, P. Ruggieri, W. Benson, *Instructional Tools for Promoting Self-directed Skills in Freshmen*, FIE 2002, Boston, November, 2002.

FURTHER READING

D. R. Krathwohl, B. S: Bloom, and B. B. Masia, *Taxonomy of Educational Objectives: the Classification of Educational Goals*, London: Longmans Green and Co., 1956.

[Denning, 2003] P. J. Denning, Great principles of computing, *Comm.* ACM, **46**(11): 15–20, 2003.

K. McKeown, L. Clarke, and J. Stankovic, CRA Workshop on Research Related to National Security: Report and Recommendations, *Computing Res. Review News*, **15**, 2003.

National Science Foundation Blue-Ribbon Advisory Panel, *Revolutionizing Science and Engineering Through Cyber-infrastructure*, Arlington, UA: National Science Foundation, 2003.

Quality Assurance Agency for Higher Education. *National Qualification Frameworks*, published by Gloucester, England: The Quality Assurance Agency for Higher Education.

ANDREW MCGETTRICK
Glasgow, Scotland

Q

QUEUEING NETWORKS*

A major airline has set up a computerized transaction system used by its ticket agents to sell seats on its aircraft. The airline has authorized 1000 agents around the country to make reservations from their workstations. A "disk farm"—a large collection of magnetic-disk storage devices—in New York contains all the records of flights, routes, and reservations. On average, each agent issues a transaction against this database once every 60 seconds. One disk contains a directory that is consulted during every transaction to locate other disks that contain the actual data; on average, each transaction accesses the directory disk 10 times. The directory disk takes an average of five milliseconds to service each request, and it is busy 80% of the time.

On the basis of this information, can we calculate the throughput and response time of this system? Can we find the bottlenecks? Can we say what happens to the response time if we reduce directory disk visits by half or doubled the number of agents?

These performance questions are typical. The answers help analysts decide whether a system can perform well under the load offered, and where to add capacity if it is too slow. Most people think that no meaningful answers can be given without detailed knowledge of the system structure—the locations and types of the agents' workstations, the communication bandwidth between each workstation and the disk farm, the number and types of disks in the farm, access patterns for the disks, local processors and random-access memory within the farm, the type of operating system, the types of transactions, and more. It may come as a surprise, therefore, that the first two questions—concerning throughput, response time, and bottlenecks—can be answered precisely from the information given. The second two questions—concerning effects of configuration changes—can be answered with reasonable estimates made from the information given and a few plausible assumptions.

These questions illustrate the two important types of performance questions: calculation and prediction. Calculation questions seek metrics in the same observation period where parameters were measured. Prediction questions provide metrics in a different (future) observation period from when parameters were measured.

OPERATIONAL LAWS

A computer network is composed of interconnected servers. Servers include workstations, disks, processors, databases, printers, displays, and any other devices that can carry out computational tasks. Each server receives and queues up messages from other servers that specify tasks for the server to carry out; a typical message might ask a server to run a computationally intensive program, to perform an input-output transaction, or to access a database. A transaction is a specified sequence of tasks submitted to the network; when a server completes a particular task, it deletes the request from its queue and sends a request to another server to perform the next task in the same transaction.

Measurements of servers are always made during a definite observation period. Basic measures typically include event counters and timers. These and other measures derived from them are called operational quantities. Invariant relations among operations quantities that hold in every observation period are called operational laws.

By counting outgoing messages and by measuring the time that a server's queue is nonempty, it is easy to measure the output rate X, the mean service time S, and the utilization U of a server. These three empirical quantities satisfy the relation $U = SX$, known as the utilization law (Fig. 1). Similarly, by measuring the "space-time" accumulated by queued tasks, it is easy to determine the mean queue length Q and the mean response time R: These quantities satisfy the relation $Q = RX$, which is known as Little's Law (Fig. 2).

The utilization law and Little's law are counterparts of well-known limit theorems for stochastic queueing systems in a steady state. These theorems will usually be verified in actual measurements, not because a steady state has been attained, but because the measured quantities obey the operational laws (1, 2).

The tasks that make up a transaction can be regarded as a sequence of visits by the transaction to the servers of the network. The average number of visits per transaction to a particular server i is called the visit ratio V_i for that server; the server's output rate X_i and the system's output rate X_0 satisfy the relation $X_i = V_i\mathrm{X}_0$, which is known as the forced-flow law (Fig. 3). This remarkable law shows that knowledge of the visit ratios and the output rate of any one server is sufficient to determine the output rates of every other server and of the system itself. Moreover, any two networks with the same visit ratios have the same flows, no matter what is the interconnection structure among their servers.

In a network, a server's output is a portion of another server's input or of the system's output. It simplifies an analysis to assume that the input and output flows of a server are identical—a condition known as flow balance. Strictly speaking, "throughput" refers to the rate of a flow-balanced server. The definitions do not imply flow balance. In most real systems, a bound is placed on the number of tasks that can be in the system at once; as long as the number of completions at every server is large

*This article is adapted from two articles published by the author in *American Scientist* magazine in 1991: (a) Queueing in Networks of Computers, May-June, 206-209, and (b) In the Queue: Mean Values, September-October, 402-403. Copyright held by the author.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.
Copyright © 2008 John Wiley & Sons, Inc.

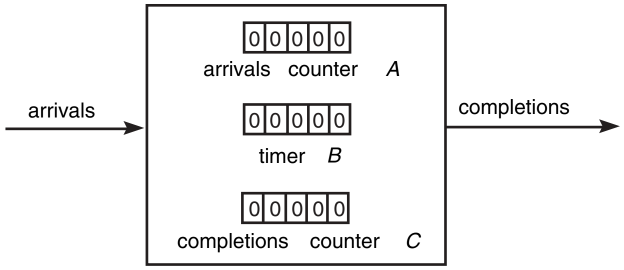


Figure 1. The task-processing server is the basic element of a network of computers. Over an observation period of length T, the counter A records the number of tasks that arrive at the server, the counter C records the number of tasks completed, and the timer B measures the total busy time (time when tasks are present). The utilization of the server is $U = B / T$, the output rate is $X = C / T$, and the mean service time per completed task is $S = B / C$. The identity $B/T = (C/T)(B/C)$ translates to the utilization law: $U = XS$. Because utilization cannot exceed 1, the output rate cannot exceed the saturation rate of $1/S$.

compared with this bound, the error introduced by assuming flow balance will be negligible. For this reason, flow balance does not generally introduce much error into the models.

When a network of servers receives all of its requests from a finite population of N users who each delay an average of Z seconds until submitting a new transaction, the response time for a request in the network satisfies the response-time formula $R = N/X_0 - Z$ (Fig. 4). This formula is exact for flow balance.

These formulas are sufficient to answer the throughput and response-time calculation questions posed earlier for the airline reservation network. We are given that each transaction generates an average of 10 directory-disk requests, and so $V_i = 10$ for the server represented by the directory disk. The mean service time at the directory disk is five milliseconds, so that $S_i = 0.005$ second. The directory disk's utilization is 80%: $U_i = 0.8$. Combining the forced-flow law and the utilization law, we have for total system throughput:

$$X_0 = U_i/V_iS_i = 0.8/(10^*0.005) = 16 \text{ transactions per second}$$

Thus, the entire airline reservation system is processing 57,600 transactions per hour. The response time experienced by any one of the 1000 agents is:

$$R = N/X_0 - Z = (1000/16) - 60 = 2.5 \text{ seconds.}$$

BOTTLENECKS

Every network has a bottleneck: A server that is slower than all the others and limits the overall throughput. Speeding up a bottleneck can yield a significant improvement in system throughput. Speeding up a nonbottleneck may hardly affect throughput. Our performance predictions will depend our knowledge of the bottlenecks.

Finding the bottleneck is easy. Suppose that the visit ratios and mean service times do not vary with N. Each server generates a potential bottleneck that would limit the system throughput to $1/V_iS_i$ and would give a lower bound to the response time of $NV_iS_i - Z$. Obviously, the server with the largest value of V_iS_i gives the tightest limit and is the real bottleneck. The products V_iS_i are sufficient to determine lower bounds on the response time as a function of N (Fig. 5).

The operational laws coupled with bottleneck analysis offer a simple but powerful method for performance analysis. For systems whose visit ratios and service times do not vary with overall load, the products V_iS_i—the total service time requirement for each server—are sufficient to answer these questions.

CHANGING THE CONFIGURATION

Consider the two prediction questions we asked at the beginning. They ask about the future effect of reducing directory-disk visit ratio or doubling the number of agents. The operational laws, which deal only with relations among quantities observed in a single measurement period, are

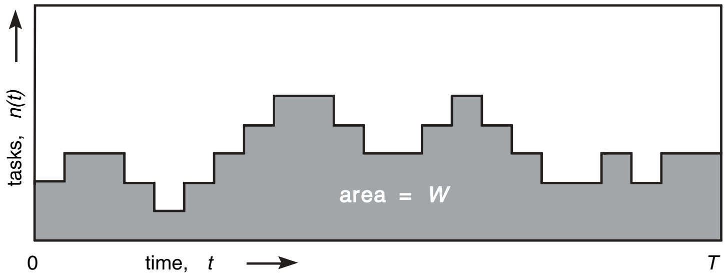


Figure 2. The average response time of a server can be calculated from just a few measurements. Let (nt) denote the number of tasks in the server at time t. Let W denote the area under the graph of $n(t)$ in the interval from time 0 to time T; W is the number of task-seconds of accumulated waiting. The mean number of tasks at the server is $Q = W / T$, and the mean response time per completed task is $R = W / C$. The identity $W / T = (C / T)(W / C)$ translates to Little's law: $Q = XR$. The mean service time S and the mean response time R are not the same; R includes queueing delay as well as service time.

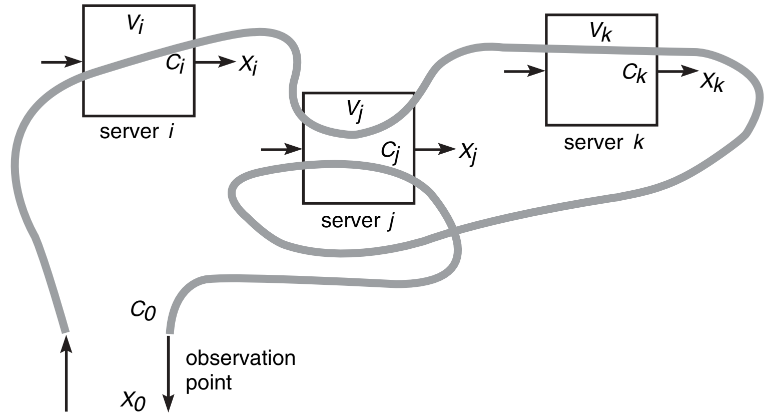


Figure 3. Flow of transactions through a network of servers can be calculated from a few selected measurements. Over an observation period T, the system completes C_0 transactions. The average number of tasks per transaction for server i is $V_i = C_i / C_0$; V_i is called the visit ratio because each task is regarded as a "visit" by the transaction to the server. Here the transaction visits servers i and k once and server j twice. The identity $C/T = (C_i/C_0)(C_0/T)$ translates to the forced-flow law: $X_i = V_iX_0$. This law says that the task flow at one point in the system determines the task flows everywhere. This law holds regardless of the interconnections among the servers; any two networks with the same visit ratios will have the same flows. The local throughput constraint $X_{\mathrm{i}} \leq 1 / S_i$ translates to a system throughput constraint $X_0 \leq 1/V_iS_i$.

not sufficient for making predictions. We must introduce additional forecasting assumptions that extrapolate measured parameter values from the past observation period into the future observation period; the laws can then be used to calculate the response time expected in that future period.

The most common type of forecasting assumption is that, unless otherwise specified, the visit ratios V_i, mean service times S_i, think time Z, and overall load N will be the same. When we change the workload or the strategy for processing data, or the electrical, mechanical, parallelism of a server, we alter only the affected parameters.

Our first prediction question asks what happens when the demand for the directory disk is cut in half (its speed is

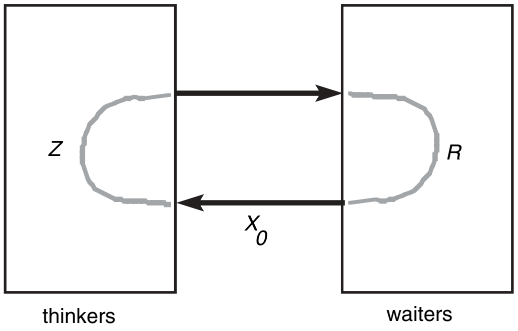


Figure 4. Users of a transaction system alternate between periods of "thinking" and "waiting" when using the system. The total number of thinkers and waiters is the number of users N. The average (waiting) response time per transaction is R and the average thinking time is Z. Little's law says that the mean number of active users in an entire system is equal to the mean cycle time of the system multiplied by the flow through the system. These three quantities are, respectively, N, $R+Z$, and X_0. Solving $N = (R+Z)X_0$ for the response time, we obtain the response-time formula: $R = N/X_0 - Z$. In the extreme case of response time almost zero—because all the servers are ultra-fast—the system throughput would be $X_0 = N/Z$ because transactions are flowing at the rate individual users complete their thinking intervals.

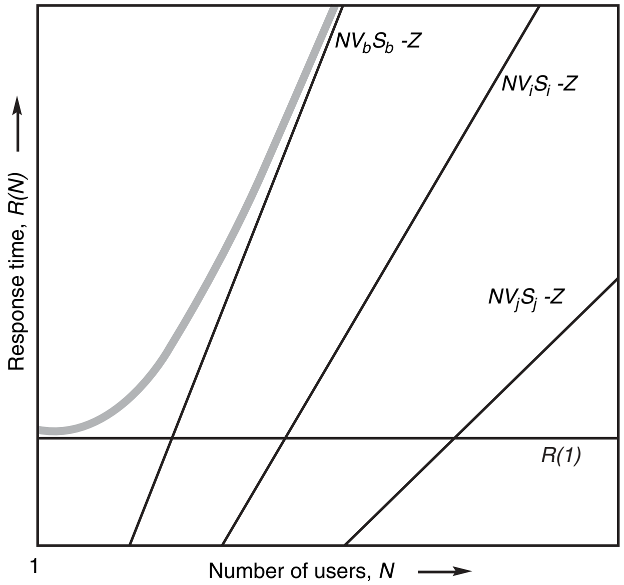


Figure 5. Bottleneck analysis shows how the response time changes as a function of N. When $N = 1$, the single user's transactions encounter no queueing delays from other transactions, whence $R(1) = V_1S_1 + \cdots + V_KS_K$, where K is the number of servers. Combining the utilization and forced-flow laws, $X_0 = X_i/V_i = U_i/V_iS_i < 1/V_iS_i$ because $U_i < 1$. Thus $R(N) > NV_iS_i - Z$ for all i. Each of the lines defined by these relations is a potential asymptote for $R(N)$ with large N. The actual asymptote is determined by the largest of the potential asymptotes. Taking server b (for bottleneck) to be the one with the largest V_iS_i, we have $R(N) > NV_bS_b - Z$. The bottleneck analysis assumes that the products V_iS_i do not vary with N.

doubled). The answer depends on whether the directory disk is the bottleneck:

1. If another server is the bottleneck, then speeding up the directory disk will not change the limit on system throughput.
2. If the directory disk is initially the bottleneck, but is no longer the bottleneck after its speed is doubled, then the throughput improves to the limit imposed by the second-slowest server.
3. If the directory disk is still the bottleneck after its speed is increased, then the system throughput improves to the new limit imposed by the directory disk.

The limits of cases 2 and 3 may be superseded by a think-time-imposed limit. No matter how fast the servers are, the maximum rate at which users submit transactions is N/Z; therefore, N/Z is also a limit on the system throughput. Cases 2 and 3 cannot improve beyond that limit.

The operational laws can yield nonsense if the bottleneck effects are ignored. For example, if we assume that doubling the directory disk speed will also double the system throughput, we would change the throughput from 16 to 32 in the response time formula and calculate an absurdity:

$$R = N/X_0 - Z = (1000/32) - 60 = -28.75\,\text{seconds},$$

The think-time constraint on throughput is $N/Z = 1000/60 = 16.7$ transactions per second. If we hypothesize that throughput X_0 can be larger, the response time law tells us that $R = N/X_0 - Z$ is less than zero. That is impossible.

All we can say with the given information and the given forecasting assumptions is that halving the demand for the directory disk will reduce the response time from 2.5 seconds to some small but still nonzero value. If the 2.5-second response time is acceptable, then this proposed change in directory search strategy would not be cost effective.

Consider the second configuration question: What happens to the response time if the number of agents is doubled? Again, we are limited by the lack of knowledge of the other disks. If the directory disk is the bottleneck, then doubling the number of agents is likely to increase its utilization to 100 %, giving a saturation value of throughput:

$$X_0 = 1/V_i S_i = 1/(10^* 0.005) = 20\,\text{transaction per second}$$

with corresponding response time,

$$R = N/X_0 - Z = (2000/20) - 60 = 40\,\text{second}$$

If the directory disk is not the bottleneck, then some other server will have a smaller saturation throughput, which forces response time to be longer than 40 seconds. Thus, doubling the number of agents will produce a response time that is likely to be unacceptably high.

COMPUTATIONAL ALGORITHMS

The simple methods described above cannot answer the more complex question of how throughput and response vary with the load N on the system. These questions can be answered with very simple algorithms that can be programmed easily on a spreadsheet or hand-held calculator.

The networks-of-queue model was first proposed by Jackson in 1957 (3), and mathematical expressions for its steady-state probabilities were presented by Gordon and Newell in 1967 (4). Their expressions were, unfortunately, exceedingly complex. To calculate a simple quantity such as central processing unit (CPU) usage required summations over enormous state spaces whose size grew exponentially with N. The computational algorithms for these models thus seemed to be intractable. Consequently, these models were not taken seriously, even though a few sporadic experimental studies showed they worked well. In 1973, Jeffrey Buzen presented a breakthrough: an algorithm that computed performance metrics from the Gordon-Newell model in time $O(N^2)$(5). Buzen's algorithm led to many studies that validated the models and extended them to many new cases and systems. Performance evaluation became the focus of a large and flourishing industry. One of the important extensions was Mean Value Analysis from Martin Reiser and Steve Lavenberg in 1980 (6), which eventually became the industry standard.

The Mean Value Algorithm is so named because it computes a set of means for a closed network (fixed load N)—the response times $R_i(N)$, queue lengths $Q_i(N)$, throughputs $X_i(N)$, and the system throughput $X(N)$ and response time $R(N)$. (Note that we have dropped the subscript "0" from the system throughput notation.) It does this iteratively for $N = 1, 2, 3, \ldots$ starting with the observation that the queue lengths are all 0 when $N = 0$.

The box below summaries the equations that the algorithm uses to obtain the mean values for load N once the mean values for load $N-1$ have been calculated. Following is an explanation for each of the equations.

Mean Value Equations

(1)	$R_i(N) = S_i(1 + Q_i(N-1))$	for all i
(2)	$R(N) = \sum_{i=1}^{K} V_i R_i(N)$	
(3)	$X(N) = \dfrac{N}{R(N) + Z}$	
(4)	$Q_i(N) = X(N) V_i R_i(N)$	for all i

1. When a job arrives at server i, it waits in the queue. That queue's length just before the arrival is approximated as the overall mean queue length

when the arriving job is not present (load $N-1$). Just after the arrival, that queue's length is one larger. The arriving job's response time is one service time S_i for each job in the queue just after its arrival. (We will discuss the accuracy of this approximation shortly.)

2. The overall response time is the sum of all per-visit server response times over all visits. This sum is actually an operational law. Multiply both sides by $X(N)$, apply Little's law to reduce the left side to N, and apply both the forced-flow law and Little's law to convert each term $X(N)V_iR_i(N)$ to $X_i(N)R_i(N)$ and then to the mean queue $Q_i(N)$. The result is the identity that N is the sum of the queue lengths of all the servers.
3. This equation is the response-time law solved for $X(N)$.
4. This equation is Little's law applied at each server.

Figure 6 is a very simple version of the airline reservation system. Figure 7 illustrates what the Mean Value Algorithm yields when applied to this model. It is easy to answer the two prediction questions simply by altering the parameters to their new values and applying the algorithm.

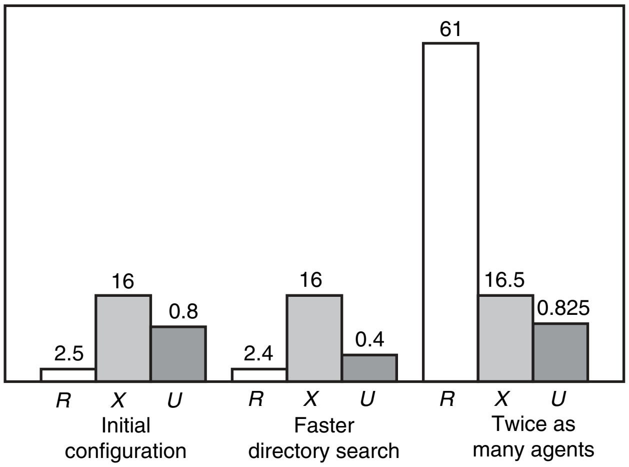


Figure 7. The bar graphs show the results of three network analyses. The original system has an average response time (R) of 2.5 seconds and a throughput (X) of 16 transactions per second; the directory, which seems likely to be the bottleneck, has a utilization (U) of 0.8. If the directory accesses are halved, the utilization of the directory disk falls to 0.4, but the improvement in response time is imperceptible. If the number of agents is doubled, then the response time jumps to 61 seconds.

THE RANDOM ARRIVAL ASSUMPTION

Our explanation for the Mean Value equations shows that only the first equation contains an approximation; the other three equations are operational laws. Let us consider the nature of this approximation. Whatever errors exist between values calculated from the model and values measured in a system develop from this approximation.

The approximation has two parts: (*1*) the arriving job observes the same queue length as a random outside observer would with one less job in the system, and (*2*) all jobs in the queue require the average service time to complete. These assumptions are not necessarily good assumptions. Here's why.

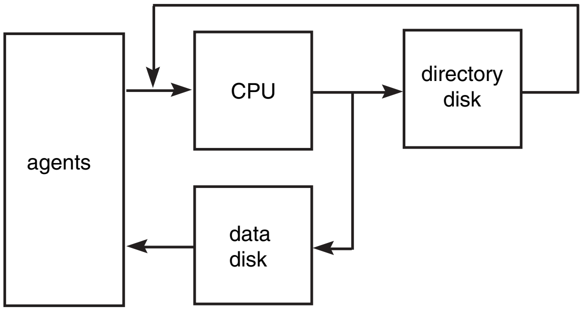


Figure 6. The hypothetical airline reservation system serves as an example of a computer network subject to mathematical performance analysis. In the initial configuration, 1000 agents access a database at the airline's central computing facility. Each agent thinks an average of 60 seconds between transactions. A typical transaction requires 10 lookups on the directory disk to locate the information requested, and then one lookup on the data disk to deliver the result. Each of these 11 disk accesses also requires service from the CPU. The total CPU time of a transaction averages 50 milliseconds. The directory disk service time is 5 milliseconds, and the data disk service time is 60.7 milliseconds.

In violation of part 1, the system may have a scheduling algorithm that admits new jobs to active status only when another job leaves. It synchronizes job arrivals with departures. In this case, the arrivals do not act as random outside observers. The arriving job may observe a queue that is shorter than what the random outside observer sees.

In violation of part 2, when a job arrives at a busy server, there is already a job in progress when it arrives. We know from queueing theory that the expected time until the job-in-progress completes is equal to S_i only if the distribution of service times is exponential. If the service times have a long-tail distribution (not unusual), then the expected time until the job-in-progress completes may be considerably larger than S_i. Thus, the assumption that every job in queue needs S_i time to complete may not hold for servers with long-tailed distributions. The arriving job may observe a response time that is longer than the approximation implies.

In both cases, Mean Value Equation (1) will underestimate the true response time. The underestimate may be considerable for long-tailed service distributions.

Few real systems exactly satisfy the assumption behind Equation (1). Yet extensive experiment studies have demonstrated that the models based on it are nonetheless robust: It is almost always possible to construct a model whose estimates of throughput and utilization are within 5% of the true values and whose estimates of response time are within 25% of the true values.

This is quite a remarkable track record for such a simple algorithm.

EXTENDING THE MEAN VALUE EQUATIONS

The Mean Value Equation [Equation (1)] is not the only way to approximate the response time. Yon Bard introduced another approximation in consultation with Paul Schweitzer in 1979 (7). The idea was to approximate the mean queue observed by the arriving job as a simple downward proration of the current mean queue length:

$$Q_i(N-1) = \frac{N-1}{N} Q_i(N)$$

After this subsitution, all the mean values mentioned in the equations are functions only of the load N, which can be dropped as an explicit parameter. The result is the simplified equations in the box below.

Bard-Schwetizer Equations for load N

(5)	$R_i = S_i\left(1 + \frac{N-1}{N} Q_i\right)$	for all i
(6)	$R = \sum_{i=1}^{K} V_i R_i$	
(7)	$X = \frac{N}{R+Z}$	
(8)	$Q_i = X V_i R_i$	for all i

For a given N, these equations are solved iteratively. The algorithm starts with any guess for the mean queue lengths, for example all 1. It then cycles through the Equations (5)–(8), which produces successive new guesses for mean queue lengths. The sequence of guesses converges to a set of values that solve the equations. Those mean values are the estimates of response time, throughput, and queue lengths for the system at load N.

Experimental validations have confirmed that the Bard-Schweitzer equations give good approximations in practice, usually with the same errors as the original Mean Value equations.

Another approximation addresses one problem mentioned earlier, that the actual response time is much larger than the Mean Value Equation [Equation (1)] assumes for long-tailed service distributions. It borrows from queueing theory the Pollaczek-Khintchine formula, which says

$$R_i = S_i\left(1 + \frac{U_i}{1-U_i}\,\frac{1+C_i^2}{2}\right)$$

where C_i is the coefficient of variation, namely the ratio of service time standard deviation to mean. This formula can replace Equation (5) in the Bard-Schweizer equations, with $U_i = X_i S_i$. Exponential service time distributions have $C_i = 1$, simplifying the formula to $R_i = S_i/(1-U_i)$ for those servers.

OPERATIONAL ANALYSIS

The discussion above is couched in operational terms: All parameters are taken directly from measured data, and all computed metrics represent measured metrics. For this reason, the analytic approach outlined above is called operational analysis. It begins with the laws and relationships among quantities observable in a system over a time period. It uses these laws to determine the limits bottlenecks impose on throughput and response time. With the additional assumptions of flow balance and random arrivals, it leads to the Mean Value Equations for calculating the throughput, response times, and mean queues. Operational analysis allows the measurement of errors caused by assumptions such as flow balance or random arrivals.

Traditional queueing theory assumes that stochastic (random) processes govern the performance quantities. Its more powerful methods allow calculation of metrics based on entire service distributions, not just the means. Many steady-state limit theorems of queueing theory turn into operational laws or formulas that hold for flow-balanced networks.

Operational analysis is more intuitive and easier to understand in the most common performance evaluation cases than traditional queueing theory. It gives more credibility to performance models because its assumptions are verifiable. It is commonly used in performance evaluation textbooks to introduce the basic ideas of queueing theory for computing systems and networks (8). Operational analysis, however, is not a replacement for traditional queueing theory.

The genesis of the operational interpretation was in the mid-1970s, when performance analysts were discovering that the formulas of Markovian queueing systems worked very well to predict utilizations, throughputs, and response times in real networks of computers, even though the Markovian assumptions themselves were grossly violated. Jeffrey Buzen proposed the operational hypothesis: Many traditional steady-state queueing formulas are also relations among observable quantities under simple, general conditions (9). This hypothesis has been substantiated in practice and has underpinned many computer programs that accurately calculate performance measures for network-of-server models of computer systems, computer networks, and manufacturing lines.

REFERENCES

1. P. J. Denning and J. P. Buzen, Operational analysis of queueing networks, ACM *Comput. Surv.* **10**(3): 225–261, 1978.
2. E. D. Lazowksa, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance*, Upper Saddle River, NJ: Prentice-Hall, 1984.
3. J. R. Jackson, Networks of waiting lines *Operations Res.* **5**: 518–521, 1957.
4. W. J. Gordon and G. F. Newell, Closed queueing systems with exponential servers, *Operations Res.* **15**: 254–256, 1967.
5. J. P. Buzen, Computational algorithms for closed queueing networks with exponential servers, ACM *Commun.* **15**(9): 527–531, 1973.

6. M. Reiser and S. Lavenberg. Mean value analysis of closed multichain queueing networks. *J. ACM* **27**: 313–322, 1980.
7. Y. Bard. Some extensions to multiclass queueing network analysis. *Proc. of the Fourth International Symposium on Computer Performance Modeling, Measurement, and Evaluation* (H. Beilner and E. Gelenbe, eds.), Amsterdam: North-Holland, 1979.
8. D. Menascé, V. Almeida, and L. Dowdy, *Capacity Planning and Performance Modeling*, Upper Saddle River, NJ: Prentice-Hall, 1994.
9. J. P. Buzen. Operational analysis: The key to the new generation of performance prediction tools. *Proc. IEEE COMPCON* **76**, Washington, DC: 166–171, 1976.

FURTHER READING

K. C. Sevcik and I. Mitrani, The distribution of queueing network states at input and output instants. *J. ACM* **28**: 358–371, 1981.

PETER J. DENNING
Naval Postgraduate School
Monterey, California

S

SWAPPING

Swapping is a general term for exchanging blocks of program code or data between main and secondary memory. An executing program encounters swapping in three ways:

- Swapping in: Moving the entire program into main memory at the start of execution.
- Swapping out: Moving the entire program out of main memory at the completion of execution.
- Page swapping: Moving individual pages of the program in or out of main memory during execution of a program.

The time to complete a swap is typically 10,000 to 100,000 times the basic instruction time. Since swapping is so expensive compared with instruction execution, system designers have always sought ways to minimize and mask it.

The term originated in the time-sharing systems of the early 1960s. These systems had insufficient memory to allow more than one program to be loaded (swapped in) for execution at a time. At the end of a time slice or stop for input/output, their operating systems swapped out the executing program and then swapped in another program. A single program could be swapped many times during its execution. Swapping was a perfect description of the main work of time-sharing—switching the CPU from one program to another.

In those early systems, swapping and CPU execution were disjoint activities. The operating system controlled the swapping overhead by setting time slices to be multiples of the average swapping time. MIT's CTSS was able to guarantee that the CPU would be executing programs about 50% of the time with this strategy (1).

To improve CPU efficiency to near 100%, operating systems of the late 1960s incorporated multiprogramming. Swapping was limited to swapping in and swapping out. The CPU could be switched among loaded programs without further swapping. Swaps were masked by performing them in parallel with CPU execution, without interrupting or slowing the CPU.

Multiprogramming is often combined with virtual memory. In that case, the operating system may allocate fewer pages of memory to a program than the size of its address space. There will be page swapping during execution. The operating system maintains a complete copy of the address space in a swap file on the disk; each page fault swaps a page from that file with a page from main memory. The paging algorithm attempts to minimize page swapping.

Modern operating systems use swapping in all these forms. Windows XP and Vista, Mac OS 10, and Linux all combine multiprogramming and virtual memory. They allow users the option of turning off virtual memory; in which case, the operating system will swap in a program's full address space at the beginning and then execute without page swapping.

BIBLIOGRAPHY

1. F. J. Corbato, M. Merwin-Daggett, and R. C. Daley, An experimental time sharing system, *Proceedings of the Spring Joint Computer Conference 21*. In: S. Rosen (ed.), *Programming Languages and Systems*. New York, McGraw-Hill, 1967, pp. 335–344.

FURTHER READING

A. Tanenbaum, *Modern Operating Systems*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 2007.

Peter J. Denning
Naval Postgraduate School
Monterey, California

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

THRASHING

Thrashing is an unstable collapse of throughput of a system as the load is increased. It violates the intuition that throughput should increase smoothly toward a saturation level as load increases. Thrashing occurs when contention for a critical resource prevents most jobs from actually using the resource.

Thrashing was first observed in the first-generation multiprogrammed virtual memory computing systems of the 1960s. Designers expected system throughput to increase toward a saturation level as the load (level of multiprogramming) increased. To their great surprise, throughput dropped suddenly after a critical load (see Fig. 1). Moreover, throughput would not return to its former high until the load was reduced below the critical value that triggered the thrashing — a form of hysteresis. The system did not crash, but it did slow to an imperceptible crawl. The engineers called it "paging to death" because all the jobs were constantly queued up waiting for the paging disk to satisfy their page faults.

Thrashing was a serious problem. It made the new generation of operating systems look like multi-million-dollar liabilities. Who would purchase such a system?

Thrashing was explained in 1968 (1). Increasing load in a fixed size memory causes the average partition to decrease, forcing an increase in the rate of page faults. The load at which the mean CPU time between page faults equals the disk service time is a tipping point: At higher loads, most jobs are queued at the paging disk, which becomes the system bottleneck.

As an example, consider a job that requires 1 second of CPU time on a system with a page fault service time of 10 milliseconds. At a small load, the job gets a large partition and generates (say) 20 page faults, which require a total of 0.2 seconds of disk time; this job's CPU efficiency is 1/(1 + 0.2) = 0.83. At an intermediate load, the job's partition has been squeezed a little and it generates (say) 100 page faults; the CPU efficiency drops to 0.5. At a large system load, the job's partition has been squeezed a lot and it generates (say) 1000 page faults; its efficiency drops to 0.09. At the small loads, the job is CPU-bound; at the large loads, it is disk-bound. At the tipping point (efficiency 0.5), the average CPU time between page faults equals the disk service time.

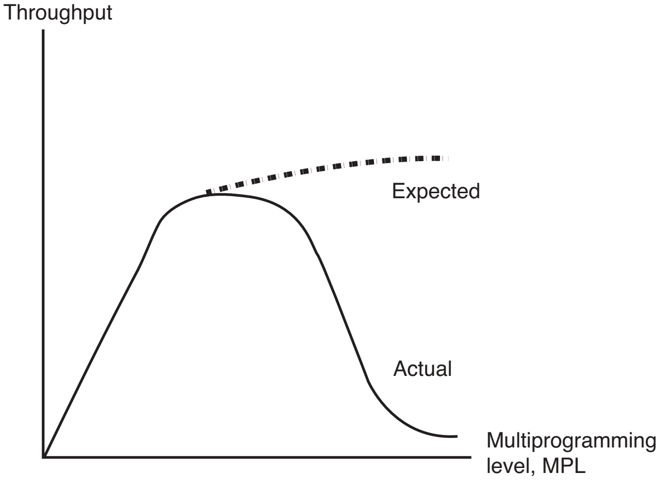


Figure 1. Thrashing in a multiprogrammed computer system.

The solution to thrashing is a load controller that allows all jobs enough memory space to keep their CPU efficiencies at 0.5 or higher. To accomplish this, a load controller separates submitted jobs into two sets: active and waiting. The active jobs are allowed to hold space in main memory. All other jobs are waiting. The working-set criterion is the ideal for deciding when to activate a waiting job. A job's working set is the set of pages required to allow the job to have a CPU efficiency above 0.5. If all active jobs have efficiencies above 0.5, the system will be below its thrashing tipping point. As long as every job gets enough space for its working set, it is impossible to activate enough jobs to push the system past its thrashing tipping point (2). The working-set criterion was found empirically not only to be optimal but to be more robust than other load-control criteria (3).

Within a decade after these early analyses, queueing network models were routinely used to quantify the relationship between throughput and the total demands for devices, and to help design load controllers to prevent thrashing (4).

Thrashing has been observed in other systems as well. Early packet networks provided examples. In unregulated networks, many servers vie for bandwidth in a common medium such as a satellite channel or Ethernet cable. When a server has a packet to transmit, it transmits into the medium. If another server jams it before completing, it stops transmitting, and tries again. This simple protocol works well for low loads, and system throughput increases with load. However, when the load gets high enough, there is a good chance that two or more servers will get into a loop where they start to transmit, detect a jam, stop, and repeat. Thereafter no one gets to transmit and throughput suddenly drops. This behavior was first observed in the ALOHA satellite communication network in the late 1960s (5).

Again, a properly designed load controller prevents the thrashing. In this case, the control is on the time interval from when a server drops out (it detected a jam) until it retries. It picks a random number for the first wait period; if that gets jammed, it waits twice as long; if that gets jammed, it waits twice as long again; and so on. This increasing-backoff protocol is used in Ethernets and cell phone networks.

Another instance of thrashing occurs from lock contention in database systems. A transaction requests and locks

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

all records it needs, but if a requested record is already locked by another transaction, it releases all its locks and tries again. Again, the doubling of successive backoff intervals prevents thrashing, a condition in which lock contention prevents anyone from locking anything.

The common features of these systems are as follows: (*1*) A shared resource for which there can be many requests, and (*2*) a contention resolution protocol whose delay increases with the number of contenders. As the number of contenders increases, more and more time is spent on contention resolution and few jobs get through to the resource.

BIBLIOGRAPHY

1. P. J. Denning, Thrashing: Its causes and prevention, *Proc. AFIPS Fall Joint Comput. Conf.* 1968; **32**: 915–922.
2. P. J. Denning, The working set model for program behavior, *ACM Commun.* 1968; **11** (May): 323–333.
3. P. J. Denning, Working sets past and present, *IEEE Trans. Softw. Eng.* 1980; **SE-6** (January): 64–84.
4. P. J. Courtois, *Decomposability*. Reading, MA: Academic Press, 1977.
5. L. Kleinrock, *Queueing Theory*, vol. 2. New York: Wiley, 1976, pp. 360–407.

FURTHER READING

A. Tanenbaum, *Modern Operating Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

PETER J. DENNING
Naval Postgraduate School
Monterey, California

T

THROUGHPUT

The throughput of a system is the number of jobs, tasks, or transactions the system completes per unit time. If during an observation period of length T the system completes C jobs, throughput X is measured as C/T. Throughput is often used as a figure of merit, with higher values indicating better performance. System engineers use analytic or simulation models to help determine a system configuration and server capacities that enable the system to meet throughput and response-time targets.

The term "throughput" suggests a stream of jobs flowing *through* a system. A through job would arrive and depart during the observation period. Completing jobs that were already in the system at the start of the observation period would not be counted. Therefore, counting all completions in the observation period overestimates throughput. However, in most systems, there is an upper limit N on the number of jobs in the system, reflecting the system capacity or the size of the user population. Therefore, the number of completions C is within N of the number of through jobs. For a long observation period, C is large compared with N, and throughput measured as $X = C/T$ is negligibly different from "true" throughput.

Throughput considered alone is often a deceptive measure of performance. One common deception is a server with a long queue: Its throughput is at saturation. Although the saturation throughput may be acceptable, the server's response time would be unacceptably high. Acceptable throughput does not guarantee acceptable response time.

Another deception can occur in a multiserver system. At a high load, the system's bottleneck saturates and produces its maximum possible throughput. In turn that limits the throughput at every other server in the system. If a nonbottleneck is used as the throughput reference point, the unacceptable throughput there will be due to the bottleneck, not to that server. Increasing the speed of the nonbottleneck server will not increase saturation throughput.

The subtle relationships among throughput, response time, and bottlenecks can be understood with the help of four fundamental laws of networked systems (1). The parameters are as follows:

Vi = number of visits per job to server i
Si = service time per visit at server i
N = number of users in the system
Z = think time of a user before making a new request of the system

The laws are as follows:

- *Utilization Law*: A server's utilization (fraction of time busy) is the product of its service time and throughput: $Ui = Si \times Xi$.
- *Little's Law*: A server's mean queue length is the product of its response time per visit and throughput: $Qi = Ri \times Xi$.
- *Forced Flow Law*: A server's throughput is the product of the system throughput and visit ratio: $Xi = X \times Vi$.
- *Response Time Law*: System response time per job plus think time is the number of jobs divided by the system throughput: $R + Z = N/X$.

A system bottleneck analysis follows immediately from these laws.

- Define demand $Di = Vi \times Si$ as the total expected job service required over all visits to a server.
- Since utilization cannot exceed 1, the utilization law implies $Xi \leq Vi/Di$.
- Combining with the forced flow law, the system throughput $X \leq 1/Di$ for all servers.
- Therefore, the server with largest Di limits the system throughput and is the bottleneck.
- Combining with the response time law, the mean response time cannot be smaller than $Di \times N - Z$.

An example will help illustrate these concepts. Consider a two-server system in which a job's total CPU demand is 2 seconds, a typical job visits a DISK server 100 times for 50 milliseconds each, and there are 10 users with an average think time of 20 seconds. The CPU demand is $D1 = 2$ seconds, and the DISK demand is $D2 = 5$ seconds. The DISK is the bottleneck, and system throughput is limited to 1/5 job per second. The CPU utilization cannot be higher than $D1/D2 = 2/5$. A faster CPU will have a smaller D1—speeding it up will only *reduce* its utilization. Speeding up the DISK will increase system throughput and CPU utilization. The system response time is at least $5\times10-20 = 30$ seconds. A response time of less than 30 seconds for 10 users is impossible.

The bottleneck analysis sketched above is for systems in which the parameters Vi and Si are independent of the load N in the system. If one Vi depends on N, then the upper bound on throughput may depend on N. An example of this occurs in multiprogrammed virtual memories: As load N

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

increases, job memory allocations are squeezed and paging traffic (Vi for the paging disk) increases. As load N increases, the total demand for the paging disk eventually exceeds all others and the paging disk becomes the bottleneck, forcing throughput down from a peak. (See "Thrashing" article.)

These simple relationships can be extended to systems with multiple job classes and variable rate servers (2).

BIBLIOGRAPHY

1. P. Denning and J. Buzen, Operational Analysis of queueing network models, *ACM Comput. Surv.,* **25** (September): 225–261, 1978.
2. D. Menascé, et al. *Capacity Planning*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

Peter J. Denning
Naval Postgraduate School
Monterey, California

V

VIRTUAL MEMORY

Virtual memory is the simulation of a storage space so large that users do not need to recompile their works when the capacity of a local memory or the configuration of a network changes. The name, borrowed from optics, recalls the virtual images formed in mirrors and lenses—images that are not there but behave as if they are. The designers of the Atlas Computer at the University of Manchester invented paged virtual memory in the 1950s to eliminate two looming problems: planning and scheduling data transfers between main and secondary memory and recompiling programs for each change of size of main memory.

For the first decade after its invention, virtual memory was the subject of intense controversies (1). It significantly improved programming productivity and ease of use, but its performance was unpredictable and it thrashed under multiprogramming. These problems were solved by the 1980s (2). Virtual memory is now so ordinary that few people think much about it. It is one of the engineering triumphs of the computer age.

One of the early lines of virtual memory accommodated objects of various sizes, stored in distinct storage segments. This line produced the first proposal for an object-oriented operating system (3), which led to a class of machines called capability machines (4,5), and even to a computer architecture for general object programming (6). The development of RISC produced such speeds that the special hardware in capability machines and their successors was not needed. However, all the methods used in these systems for mapping objects to their locations, protecting objects, and partitioning memory are at the heart of modern object-oriented runtime systems. We will therefore discuss virtual memory from an object point of view.

Virtual memory is ubiquitous in networked systems, which have many things to hide—on-chip caches, separate RAM chips, local disk storage, network file servers, many separately compiled program modules, multiple computers on the network, and the Internet.

MAPPING

The heart of virtual memory is a mapping between an address space and the real memory. The address space is a set of addresses sufficient to name all components of a program independent of their locations in the memory hierarchy. Virtual addresses do not change as objects are moved dynamically to various real addresses within the memory system. Programmers and users see only the virtual address space; the details of the real memory system are hidden.

Most early virtual memories were based on paging. A page is a fixed-size block or program code or data. Main and secondary memory are divided in slots of the same fixed size. Pages can then be moved from any memory slot to any other. Paging yields the simplest form of mapping but wastes storage in the last page assigned to the object.

Some early virtual memories were based on segmentation. A segment is a set of contiguous storage locations of any length. Segments could be sized as exact matches to objects such as procedures and arrays, but they are more difficult than pages to place. Segmentation has become common with object-oriented programming.

The method of mapping virtual addresses to real addresses is basically the same for both fixed and variable sized objects (paging and segmentation). The associations between virtual and real addresses are held in mapping tables; the hardware looks up the current real address for any virtual address generated by the processor. A schematic diagram is shown in Fig. 1.

Figure 1 shows the processor on the left and the memory system on the right. The processor generates virtual addresses from the address space of the process it is running. Virtual addresses are of the form (s,b), meaning byte b within segment s. Objects are stored as contiguous segments in the main and secondary memories. Figure 1 shows a segment of k bytes stored at real address c.

Between the processor and the memory is a device called a mapper. Its job is to translate virtual addresses into their current real addresses. For objects already loaded into main memory, translation consists of table lookups that yield the real address. For objects not loaded, the mapper first issues an "up" command to move it from its secondary memory file to an unused segment of main memory; and then it performs the translation. If main memory is full, the mapper will also issue "down" commands as needed to copy loaded objects back to their files and free up their space.

The mapper employs two types of tables, the Descriptor Table (DT) and the Object Tables (OT). Consider first the Descriptor Table. It has one entry for each object. Each object has a unique, system-wide name x. The entry for object x in this table consists of four parts:

- *Presence Bit*: P=1 means the object is loaded in main memory; P=0 means not.
- *Usage Bit*: U=1 means the object has been modified since being loaded; U=0 means not. Modified objects need to be copied back to their secondary files before their space can be released.
- *Base*: The base address of the segment in main memory.
- *Length*: The length of the segment in main memory.

The descriptor table is the only table in the system containing information about the physical locations of objects. When an object is moved, only the descriptor table must be updated.

The second table used by the mapper is an Object Table. There are actually multiple object tables, one for each process. A process's address space is called its "domain," and each address space has a unique domain identifier d. A

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

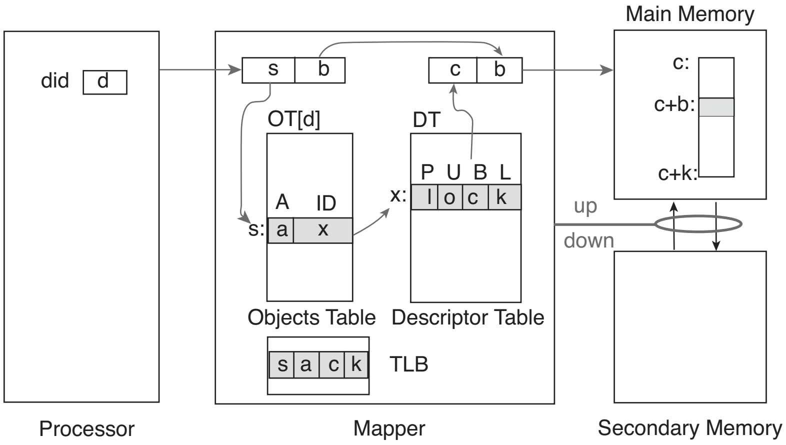


Figure 1. Object-oriented virtual memory.

register in the processor displays the domain of the current process. When the processor switches to a different process, its domain identifier register (did) is automatically changed. Thus, the processor always directs its addresses only to the objects of the currently executing process.

An object table has an entry for each segment s of its address space. That entry contains a handle of two parts:

- *Access Code*: A designates the allowable types of access, for example, read or write.
- *Identifier*: ID contains the unique system identifier x for the object.

The translation of a virtual address (s,b) to the real address containing the byte is straightforward:

1. From OT[d], get the handle for s and extract its identifier x.
2. From DT, get the base c from the descriptor for x.
3. Pass the real address c+b to the memory. (If b $\leq$ k, stop with an error.)

The object and descriptor tables are actually stored in a reserved area of main memory belonging to the mapper. Therefore, the table lookups require extra memory accesses. Those extra accesses could slow the system down a fraction of the speed without the mapper.

To bypass the table lookups whenever possible, the mapper contains a device called the Translation Lookaside Buffer (TLB) or address cache. It is a small very high-speed associative memory that holds the most recently used mapping paths. If segment s has been addressed recently, there will be an entry (s,A,B,L) in the TLB. It is built from the A-field of OT[d] plus the B- and L-fields of DT. The two table lookups are bypassed and are replaced with one ultrafast TLB lookup.

The mapper's basic cycle is as follows:

```
processor places (s,b) in address register
if ((a,c,k)=LOOKUP_TLB(s) undefined)
  then
       (a,x):=OT[d,s]
       (p,c,k):=DT[x]
       if p=0 then MISSING FAULT
       DT[x] .U:=1
       LOAD_TLB(s,a,c,k)
endif
if (b•k) then BOUNDS FAULT
if (request not allowed by a) then PROTECTION
FAULT place c+b in memory address register
```

The operation LOOKUP_TLB(s) scans all the TLB cells in parallel and returns the contents of the cell whose key matches s. The operation LOAD_TLB replaces the least-recently-used cell of TLB with (s,a,c,k). The mapper sets the usage bit U to 1 whenever the entry is accessed.

If the TLB already contains the path requested, the mapper bypasses the lookups in the object and descriptor tables. In practice, small TLBs (e.g., 64 or 128 cells) give high enough hit ratios that address-translation efficiency goals are easy to meet (7). The TLB is a powerful and cost-effective accelerator.

FAULTS

A fault is a condition that prevents additional processing. The mapper can generate three faults: missing object, out of bounds, and protection violation. Those three fault signals trigger the operating system to execute corresponding fault-handler routines that take corrective action.

The bounds fault and protection fault are fatal. References outside a segment are prohibited because they might read or write memory allocated to other objects. Unauthorized references of the wrong kind are also prohibited—for example attempting to write into a read-only object. The fault handlers for these two faults generally abort the running process.

The missing object fault occurs when the mapper encounters a not-present bit (P=0). The operating system interrupts with a missing object fault routine that

1. Locates the needed object in the secondary memory.
2. Selects a region of main memory to put that object in.
3. Empties that region by copying its contents to the secondary memory.
4. Copies the needed object into that region.
5. Updates the descriptor table.
6. Restarts the interrupted program, allowing it to complete its reference.

PERFORMANCE

The replacement policy is invoked by the missing object handler at step 2. The performance of virtual memory depends critically on the success of the replacement policy. Each missing object fault carries a huge cost: Accessing the object in main memory might take 10 nanoseconds while retrieving it from secondary memory might take 10 milliseconds—a speed differential of 100,000. It does not take very many missing object faults to seriously slow a process running in a virtual memory.

The ultimate objective of the replacement policy is to minimize the number of missing object faults. To do this, it seeks to minimize "mistakes"—replacements that are quickly undone when the process refers to those objects again. This objective is met ideally when the object selected for replacement will not be used again for the longest time among all the loaded objects. Unfortunately, the ideal cannot be realized because we have no way to look ahead into the future. A variety of non-lookahead replacement policies have been studied extensively to see how close they come to this ideal in practice. When the memory space allocated to a process is fixed in size, this usually is LRU (least recently used); when space can vary, it is WS (working set) (2).

The operating system can adjust the size of the main memory region allocated to a process so that the rate of missing object faults stays within acceptable limits. System throughput will be near-optimal when the virtual memory guarantees each active process just enough space to hold its working set (2).

PROTECTION

This structure provides the memory partitioning needed for multiprogramming. A process can refer *only* to the objects listed in its object table. It is impossible for a process to accidentally (or intentionally) read or write objects in another address space.

This structure also allows the operating system to restrict every process to a domain of least privilege. Only the objects listed in a domain's object table can be accessed by a process in that domain, and only then in accord with the access codes stored in the object's handle. In effect, the operating system walls each process off, giving it no chance to read or write the private objects of any other process. This has important benefits for system reliability. Should a process run amok, it can damage only its own objects: A program crash does not imply a system crash. This benefit is so important that many systems use virtual memory even if they allocate enough main memory to hold a process's entire address space.

THE WWW: VIRTUALIZING THE INTERNET

The World Wide Web extends virtual memory to the Internet. The Web allows an author to embed, anywhere in a document, a "universal resource locator" (URL), which is an Internet address of a file. By clicking the mouse on a URL string, the user triggers the operating system to map the URL to the file and then bring a copy of that file from the remote server to the local workstation for viewing. The URLs thus act as virtual addresses, and the combination of a server's IP address and a local file path name is the real address.

A URL is invalidated when the object's owner moves or renames the object. This can present operational problems to people who link to that object and depend on its presence. To overcome this problem, Kahn and Wilensky proposed a scheme that refers to mobile objects by location-independent "handles" and, with special servers, tracks the correspondence between handles and object locations (8). Their method is equivalent to that described earlier in Fig. 1: First it maps a URL to a handle, and then it maps the handle to the Internet location of the object.

CONCLUSION

Virtual memory is one of the great engineering triumphs of the computing age. Virtual memory systems meet one or more of the following needs:

Automatic Storage Allocation: Solving the overlay problem that originates when a program exceeds the size of the main memory available to it. Also solves the relocation and partitioning problems that develop with multiprogramming.

Protection: Each process is given access to a limited set of objects—its protection domain. The operating system enforces the rights granted in a protection domain by restricting references to the memory regions in which objects are stored and by permitting only the types of reference stated for each object (e.g., read or write). These constraints are easily checked by the hardware in parallel with the main computation. The same principles are used for efficient implementations of object-oriented programs.

Modular Programs: Programmers prepare codes as separately compiled, reusable, and sharable components into programs; their internal structure is hidden behind a public interface. Linker programs combine separate modules into a single address space.

Object-Oriented Programs: Programmers should be able to define managers of classes of objects and be assured that only the manager can access and modify the internal structures of objects (6). Objects should be freely sharable and reusable throughout a distributed system (9,10). Virtual memory mappings are designed for these objectives.

Data-Centered Programming: Computations in the World Wide Web tend to consist of many processes navigating through a space of shared, mobile objects. Objects can be bound to a computation only on demand.

Parallel Computations on Multicomputers: Scaleable algorithms that can be configured at run time for any number of processors are essential to mastery of highly parallel computations on clusters of computers. Virtual memory can join the memories of the component computers into a single address space and can reduce communication costs by eliminating some of the copying inherent in message-passing. This is known as distributed virtual memory (10).

BIBLIOGRAPHY

1. P. J. Denning, Virtual memory, *Comput. Surv.*, **2**(3): 153–189, 1970.
2. P. J. Denning, Working sets past and present, *IEEE Trans. Softw. Eng.*, **SE-6**(1): 64–84, 1980.
3. J. B. Dennis and E. C. Van Horn, Programming semantics for multiprogrammed computations, *ACM Commun.*, **9**(March): 143–155, 1966.
4. R. Fabry, Capability based addressing, *ACM Commun.*, **17**(July): 403–412, 1974.
5. M. V. Wilkes and R. Needham, *The Cambridge CAP Computer and Its Operating System.* Amsterdam, The Netherlands: North-Holland, 1979.
6. G. J. Myers, *Advances in Computer Architecture*, 2nd ed. New York: Wiley, 1982.
7. J. Hennessey and D. Patterson, *Computer Architecture: A Quantitative Approach*. New York: Morgan-Kaufmann, 1990.
8. R. Kahn and R. Wilensky, A framework for distributed digital object services. Technical Note 95-01, Corporation for National Research Initiatives. Available: http://www.cnri.reston.va.us, 1995.
9. J. S. Chase, H. M. Levy, M. J. Feeley, and E. D. Lazowska, Sharing and protection in a single-address-space operating system, *ACM TOCS*, **12**(4): 271–307, 1994.
10. A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Englewood Cliffs, NJ: Prentice-Hall, 2006.

FURTHER READING

P. J. Denning, Virtual memory, *Comput. Surv.*, **28**(4): 213–216, 1996.

PETER J. DENNING
Naval Postgraduate School
Monterey, California

WORKING SET

The working set is a dynamic subset of a process's address space that must be loaded in main memory to ensure acceptable processing efficiency. In the early days of computing, this intuitive idea enabled programmers to plan their memory usage over time in a constrained main memory. Later it turned into a formal definition of the ideal subset of address space to be loaded in main memory. Eventually the formal definition became the reference standard for all virtual memory replacement policies.

In early computing systems, programmers built overlay strategies for their programs. They made maps showing computational phases on the time axis and instruction or data block on the vertical axis. They checked off the blocks that needed to be loaded in main memory for each phase. They adjusted computational strategies and block contents until the blocks needed for each phase would fit. Then at the phase transitions, they programmed "down" and "up" commands. Down commands moved blocks from main to secondary memory when they were no longer needed in the next phase. Up commands moved blocks from secondary and main memory in time for the next phase. The set of blocks checked in the map was called the "working set" of a phase. The memory usage of the program could be characterized as a sequence

$$(\mathrm{L1}, \mathrm{T1})(\mathrm{L2}, \mathrm{T2})(\mathrm{L3}, \mathrm{T3})\ldots$$

in which each Li is the set of blocks loaded in phase i and Ti is the duration of the phase.

System designers were concerned that this process, already tedious and time consuming for small programs, would become unmanageable for large programs. In the late 1950s, the designers of the Atlas Computer at the University of Manchester invented virtual memory to automate this process. Their system broke program code and data into fixed size pages. It issued "up" commands at the moments that the program attempted to access a page not loaded. They invented a replacement algorithm that decided which loaded page to move "down" to make way for incoming "up" pages. Their algorithm assumed that each page was in a cycle of use and nonuse; by measuring the most recent use and nonuse periods, they predicted when each page would be used again. They selected for replacement the page not needed for the longest time.

Many people were attracted to the idea of virtual memory because of its big boost for programming productivity. But they were put off by the unpredictability of the replacement algorithms, which worked well for some programs and poorly for others. There were numerous contradictory experimental studies, but no one found a replacement algorithm that worked consistently well for all programs. In 1966 Les Belady (1) published an extensive study of replacement algorithms in which he demonstrated that replacement policies with usage bits performed better than those without. He suggested that this is due to "program locality," a tendency of programs to cluster references into subsets of their pages. He suggested that under multiprogramming a program should be given enough space to hold its "parachor," which was roughly the space at which the replacement algorithm's mean time between page faults equaled the page fault service time from the secondary memory.

In 1967, Denning (2) offered a precise definition of a working set. He defined it as the set of pages referenced in a virtual time window looking backwards for time T into the past. The working set dynamically varied as more or fewer pages appeared in the window. It was important to measure in virtual time—that is, not counting any interruptions—so as to get an intrinsic measure of the program's favored pages. He was able to show the somewhat surprising property that the paging rate and mean working set size could be computed easily from a histogram of the times between repeated references to the same page. It was then a simple matter to choose the window size T so that the mean time between page faults would always be larger than the mean page fault service time—that is, the CPU efficiency of the program would be at least 0.5.

Denning also showed that a multiprogrammed memory managed by a working set policy could not thrash. In later experiments with students and others, he established that the working set policy would produce system throughput within 5% to 10% of optimal, where optimal was defined in terms of perfect knowledge of the future (3). Thus, the working set policy became an ideal for other memory management policies.

The true working set would require measurements in a sliding window looking backwards from the current time. Although it worked perfectly (4), the cost of the mechanism was high. Many simpler software approximations were tried and tested, the most successful being the "WS Clock" (5).

Denning interpreted this definition of working set as a measure of the program's intrinsic memory demand. He hypothesized that programs have inherent tendencies to cluster their references into small subsets for extended periods, an idea he called "locality" after Belady. In numerous experiments with students and others, he concluded that the dynamic locality processes of programs consisted of phases and transitions; phases were periods of stability, with all references concentrated in a "locality set," and transitions were short periods of instability. In other words, every program has a natural sequence of locality sets and phases,

$$(\mathrm{L1}, \mathrm{T1})(\mathrm{L2}, \mathrm{T2})(\mathrm{L3}, \mathrm{T3})\ldots$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

A memory policy that loads exactly the locality set for each phase will achieve optimal paging behavior. As long as most phases are longer than the working set window T, the working set will be a very close measurement of these actual locality sets of varying sizes. Locality is the reason working sets work.

The locality behavior so painstakingly planned by early programmers is a natural property of programs anyway! It arises from the way that the human brain solves problems and pays attention.

In some systems, working sets can be deduced from a program's structure rather than by measurement of usage bits. For example, on machines using block-structured programming languages such as Ada, the working set can be defined as the current procedure segment, the stack, and all other data structures accessible from activated procedures.

In paging systems, it can be advantageous to "restructure" a program by clustering small, logical segments of the same locality on large pages. By preserving in the page references the locality originally present in the segment references, this strategy can yield the small working sets and efficient performance in systems with large page size. Restructuring is less important in systems with smaller page sizes.

BIBLIOGRAPHY

1. L. A. Belady, A study of replacement algorithms for virtual storage computers. *IBM Systems J.,* **5**(2): 78–101, 1966.
2. P. J. Denning,The working set model for program behavior. *Commun. ACM*,**11** 5 (May):323–333, 1968. First published in Proc. ACM Symp. on Operating Systems Principles, Gatlinburg, TN, 1967.
3. P. J. Denning, Working sets past and present. *IEEE Trans. Software Eng.,* **SE-6** 1 (January): 64–84, 1980.
4. J. Rodriguez-Rosell and J. P. Dupuy. The designimplementation, and evaluation of a working set dispatcher. *Commun. ACM*, **16** 4 (April), 1973.
5. R. Carr and J. Hennessey, WSCLOCK—a simple and effective algorithm for virtual memory management. *ACM SIGOPS Review*, **18** (December): 87–95, 1981.

FURTHER READING

A. Tanenbaum, *Modern Operating Systems*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 2007.

Peter J. Denning
Naval Postgraduate School
Monterey, California

A

ACTIVE DATABASE SYSTEMS

INTRODUCTION AND MOTIVATION

Traditionally, the database management system (DBMS) has been viewed as a passive repository of large quantities of data that are of interest for a particular application, which can be accessed/retrieved in an efficient manner. The typical commands for insertion of data records, deletion of existing ones, and updating of the particular attribute values for a selected set of items are executed by the DBMS upon the user's request and are specified via a proper interface or by an application program. Basic units of changes in the DBMS are the *transactions,* which correspond to executions of user programs/requests, and are perceived as a sequence of *reads* and/or *writes* to the database objects, which either *commit* successfully in their entirety or *abort*. One of the main benefits of the DBMS is the ability to optimize the processing of various queries while ensuring the consistency of the database and enabling a concurrent processing of multiple users transactions.

However, many applications that require management of large data volumes also have some behavioral aspects as part of their problem domain which, in turn, may require an ability to *react* to particular stimuli. Traditional exemplary settings, which were used as motivational scenarios for the early research works on this type of behavior in the DBMS, were focusing on monitoring and enforcing the integrity constraints in databases (1–4). Subsequently, it was recognized that this functionality is useful for a wider range of applications of DBMS. For example, a database that manages business portfolios may need to react to updates from a particular stock market to purchase or sell particular stocks (5), a database that stores users preferences/profiles may need to react to a location-update detected by some type of a sensor to deliver the right information content to a user that is in the proximity of a location of interest (e.g., deliver e-coupons when within 1 mile from a particular store) (6).

An active database system (ADBS) (1,7) extends the traditional database with the capability to *react* to various events, which can be either internal—generated by the DBMS (e.g., an insertion of a new tuple as part of a given transaction), or external—generated by an outside DBMS source (e.g., a RFID-like location sensor). Originally, the research to develop the reactive capabilities of the active databases was motivated by problems related to the maintenance of various declarative constraints (views, integrity constraints) (2,3). However, with the evolution of the DBMS technologies, novel application domains for data management, such as data streams (8), continuous queries processing (9), sensor data management, location-based services, and event notification systems (ENS) (10), have emerged, in which the efficient management of the reactive behavior is a paramount. The typical executional paradigm adopted by the ADBS is the so-called *event-condition-action* (ECA) (1,7) which describes the behavior of the form:

```
ON Event Detection
IF Condition Holds
THEN Execute Action
```

The basic tools to specify this type of behavior in commercially available DBMS are *triggers*—statements that the database automatically executes upon certain modifications. The *event* commonly specifies the occurrence of (an instance of) a phenomenon of interest. The *condition*, on the other hand, is a query posed to the database. Observe that both the detection of the event and the evaluation of the condition may require access not only to the current instance of the database but also to its history. The *action* part of the trigger specifies the activities that the DBMS needs to execute—either a (sequence of) SQL statement(s) or stored procedure calls. As a motivational example to illustrate the ECA paradigm, consider a scenario in which a particular enterprise would like to enforce the constraint that the average salary is maintained below 65K. The undesired modifications to the average salary value can occur upon: (1) an insertion of a new employee with above-average salary, (2) an update that increases the salaries of a set of employees, and (3) a deletion of employees with below-average salary. Hence, one may set up triggers that will react to these types of modifications (event) and, when necessary (condition satisfied), will perform corrective actions. In particular, let us assume that we have a relation whose schema is Employee(Name, ID, Department, Job-Title, Salary) and that, if an insertion of a new employee causes the average salary-cap to be exceeded, then the corrective action is to decrease everyone's salary by 5%. The specification of the respective trigger[1] in a typical DBMS, using syntax similar to the one proposed by the SQL-standard (11), would be:

```
CREATE TRIGGER New-Employee-Salary-Check
ON INSERT TO Employee
IF (SELECT AVG Employee.Salary) > 65,000
UPDATE Set Employee.
Salary = 0.95*Employee.Salary
```

This seemingly straightforward paradigm has generated a large body of research, both academic and industrial, which resulted in several prototype systems as well as its acceptance as a part of the SQL99 (11) standard that, in turn, has made triggers part of the commercially available DBMS. In the rest of this article, we will present some of the important aspects of the management of reactive behavior in ADBS and discuss their distinct features. In particular, in the section on formalizing and reasoning, we

[1] Observe that to fully capture the behavior described in this scenario, other triggers are needed—ones that would react to the UPDATE and DELETE of tuples in the Employee relation.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

motivate the need to formalize the active database behavior. In the section on semantic dimensions, we discuss the various parameters and the impact of the choice of their possible values, as they have been identified in the literature. In the overview section we present the main features of some prototype ADBS briefly, along with the discussion of some commercially available DBMS that provide the triggers capability. Finally, in the last section, we outline some of the current research trends related to the reactive behavior in novel application domains for data management, such as workflows (12), data streams (8,9), moving objects databases (13,14), and sensor networks (6).

FORMALIZING AND REASONING ABOUT THE ACTIVE BEHAVIOR

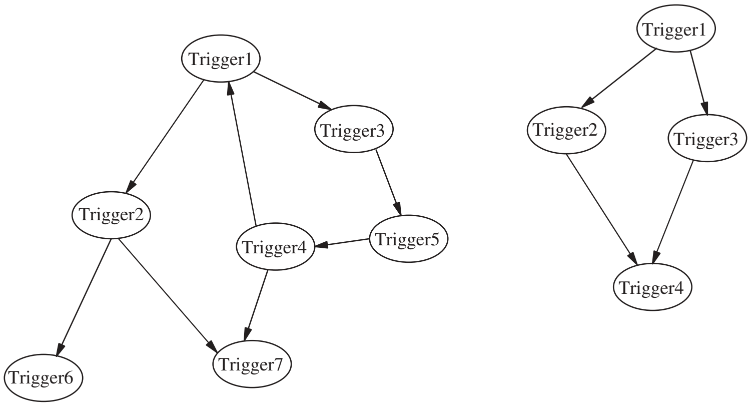
Historically, the reactive behavior expressed as a set of *condition* → *action* rules (IF *condition* holds, THEN execute *action*) was introduced in the Expert Systems literature [e.g., OPS5 (15)]. Basically, the inference engine of the system would "cycle" through the set of such rules and, whenever a left-hand side of a rule is encountered that matches the current status of the knowledge base (KB), the action of the right-hand side of that rule would be executed. From the perspective of the ECA paradigm of ADBS, this system can be viewed as one extreme point: CA rules, without an explicit event. Clearly, some kind of implicit event, along with a corresponding formalism, is needed so that the "C"-part (condition) can reflect properly and monitor/evaluate the desired behavior along the evolution of the database. Observe that, in general, the very concept of an *evolution* of the database must be defined clearly for example, the *state* of the data in a given instance together with the *activities log* (e.g., an SQL query will not change the data; however, the administrator may need to know which user queried which dataset). A particular approach to specify such conditions in database triggers, assuming that the "clock-tick" is the elementary implicit event, was presented by Sistla and Wolfson (16) and is based on temporal logic as an underlying mechanism to evaluate and to detect the satisfiability of the condition.

As another extreme, one may consider the EA type of rules, with a missing condition part. In this case, the detection of events must be empowered with the evaluation of a particular set of facts in a given state of the database [i.e., the evaluation of the "C"-part must be embedded within the detection of the events (5)]. A noteworthy observation is that even outside the context of the ADBS, the event management has spurred a large amount of research. An example is the field known as event notification systems in which various users can, in a sense, "subscribe" for notifications that, in turn, are generated by entities that have a role of "publishers"—all in distributed settings (10). Researchers have proposed various algebras to specify a set of *composite* events, based on the *operators* that are applied to the basic/primitive events (5,17). For example, the expression $E = E1 \wedge E2$ specifies that an instance of the event E should be detected in a state of the ADBS in which both *E1* and *E2* are present. On the other hand, $E = E1;E2$ specifies that an instance of the event E should be detected in a state in which the prior detection of *E1* is followed by a detection of *E2* (in that order). Clearly, one also needs an underlying detection mechanism for the expressions, for example, Petri Nets (17) or tree-like structures (5). Philosophically, the reason to incorporate both "E" and "C" parts of the ECA rules in ADBS is twofold: (1) It is intuitive to state that certain conditions should not always be checked but only upon the detection of certain events and (2) it is more cost-effective in actual implementations, as opposed to constant cycling through the set of rules.[2] Incorporating both events and conditions in the triggers has generated a plethora of different problems, such as the management of database state(s) during the execution of the triggers (18) and the binding of the detected events with the state(s) of the ADBS for the purpose of condition evaluation (19).

The need for formal characterization of the active rules (triggers) was recognized by the research community in the early 1990s. One motivation was caused by the observation that in different prototype systems [e.g., Postgres (4) vs. Starburst (2)], triggers with very similar syntactic structure would yield different executional behavior. Along with this was the need to perform some type of *reasoning* about the evolution of an active database system and to predict (certain aspects of) their behavior. As a simple example, given a set of triggers and a particular state of the DBMS, a database/application designer may wish to know whether a certain fact will hold in the database after a sequence of modifications (e.g., insertions, deletions, updates) have been performed. In the context of our example, one may be interested in the query *"will the average salary of the employees in the 'Shipping' department exceed 55K in any valid state which results via salary updates*." A translation of the active database specification into a logic program was proposed as a foundation for this type of reasoning in Ref. (20).

Two global properties that have been identified as desirable for any application of an ADBS are the *termination* and the *confluence* of a given set of triggers (21,22). The termination property ensures that for a given set of triggers in any initial state of the database and for any initial modification, the firing of the triggers cannot proceed indefinitely. On the other hand, the confluence property ensures that for a given set of triggers, in any initial state of the database and for any initial modification, the final state of the database is the same, regardless of the order of executing the (enabled) triggers. The main question is, given the specifications of a set of triggers, can one *statically*, (i.e., by applying some algorithmic techniques only to the triggers' specification) determine whether the properties of termination and/or confluence hold? To give a simple motivation, in many systems, the number of cascaded/recursive invocations of the triggers is bounded by a predefined constant to avoid infinite sequences of firing the triggers because of a particular event. Clearly, this behavior is undesirable, if the termination could have been achieved in a few more recursive executions of the triggers. Although run-time

[2] A noteworthy observation here is that the *occurrence* of a particular event is, strictly speaking, different from its *detection*, which is associated with a run-time processing cost.



(a) Non terminating cascading triggers

(b) Confluent triggers

Figure 1. Triggering graphs for termination and confluence.

termination analysis is a possible option, it is preferable to have static tools. In the earlier draft of the SQL3 standard, compile-time syntactic restrictions were placed on the triggers specifications to ensure termination/confluence. However, it was observed that these specifications may put excessive limitations on the expressive power on the triggers language, which is undesirable for many applications, and they were removed from the subsequent SQL99 draft.

For the most part, the techniques to analyze the termination and the confluence properties are based on labeled graph-based techniques, such as the triggering hyper graphs (22). For a simplified example, Fig. 1a illustrates a triggering graph in which the nodes denote the particular triggers, and the edge between two nodes indicates that the modifications generated by the action part of a given trigger node may generate the event that enables the trigger represented by the other node. If the graph contains a cycle, then it is possible for the set of triggers along that cycle to enable each other indefinitely through a cascaded sequence of invocations. In the example, the cycle is formed among Trigger1, Trigger3, Trigger4, and Trigger5. Hence, should Trigger1 ever become enabled because of the occurrence of its event, these four triggers could loop perpetually in a sequence of cascading firings. On the other hand, figure 1b illustrates a simple example of a confluent behavior of a set of triggers. When Trigger1 executes its action, both Trigger2 and Trigger3 are enabled. However, regardless of which one is selected for an execution, Trigger4 will be the next one that is enabled. Algorithms for static analysis of the ECA rules are presented in Ref. (21), which addresses their application to the triggers that conform to the SQL99 standard.

SEMANTIC DIMENSIONS OF ACTIVE DATABASES

Many of the distinctions among the various systems stem from the differences in the values chosen for a particular parameter (23). In some cases, the choice of that value is an integral part of the implementation (“hard wired”), whereas in other cases the ADBS provide a declarative syntax for the users to select a desired value. To better understand this concept, recall again our average salary maintenance scenario from the introduction. One of the possible sources that can cause the database to arrive at an undesired state is an update that increases the salary of a set of employees. We already illustrated the case of an insertion, now assume that the trigger that would correspond to the second case is specified as follows:

```
CREATE TRIGGER Update-Salary-Check
ON UPDATE OF Employee.Salary
IF (SELECT AVG Employee.Salary) > 65,000
UPDATE Employee
SET Employee.Salary = 0.95*Employee.Salary
```

Assume that it was decided to increase the salary of every employee in the “Maintenance” department by 10%, which would correspond to the following SQL statement:

```
UPDATE Employee
SET Employee.Salary = 1.10*Employee.Salary
WHERE Employee.Department = ‘Maintenance’
```

For the sake of illustration, assume that three employees are in the “Maintenance” department, Bob, Sam, and Tom, whose salaries need to be updated. Strictly speaking, an update is essentially a sequence of a deletion of an old tuple followed by an insertion of that tuple with the updated values for the respective attribute(s); however, for the purpose of this illustration, we can assume that the updates execute atomically. Now, some obvious behavioral options for this simple scenario are:

- An individual instance of the trigger Update-Salary-Check may be fired immediately, for every single update of a particular employee, as shown in Fig. 2a.
- The DBMS may wait until all the updates are completed, and then execute the Update-Salary-Check, as illustrated in Fig. 2b.

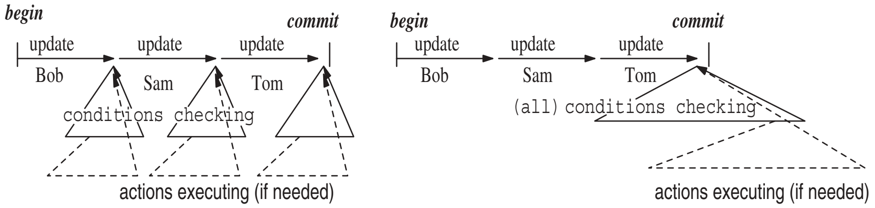


Figure 2. Different options for triggers execution.

- The DBMS waits for the completion of all the updates and the evaluation of the condition. If satisfied, the execution of the action for the instances of the Update-Salary-Check trigger may be performed either within the same transaction as the UPDATE Employee statement or in a different/subsequent transaction.

These issues illustrate some aspects that have motivated the researchers to identify various *semantic dimensions*, a term used to collectively identify the various parameters whose values may influence the executional behavior of ADBS. Strictly speaking, the model of the triggers' processing is coupled closely with the properties of the underlying DBMS, such as the data model, the transaction manager, and the query optimizer. However, some identifiable stages exist for every underlying DBMS:

- *Detection of events:* Events can be either *internal*, caused by a DBMS-invoked modification, transaction command, or, more generally, by any server-based action (e.g., clicking a mouse button on the display); or *external*, which report an occurrence of something outside the DBMS server (e.g., a humidity reading of a particular sensor, a location of a particular user detected by an RFID-based sensor, etc). Recall that (c.f. formalizing and reasoning), the events can be *primitive* or *composite*, defined in terms of the primitive events.
- *Detection of affected triggers:* This stage identifies the subset of the specified triggers, whose enabling events are among the detected events. Typically, this stage is also called the *instantiation* of the triggers.
- *Conditions evaluation:* Given the set of instantiated triggers, the DBMS evaluates their respective conditions and decides which ones are eligible for execution. Observe that the evaluation of the conditions may sometimes require a comparison between the values in the OLD (e.g., pretransaction state, or the state just before the occurence of the instantiating event) with the NEW (or current) state of the database.
- *Scheduling and execution:* Given the set of instantiated triggers, whose condition part is satisfied, this stage carries out their respective action-parts. In some systems [e.g., Starburst (2)] the users are allowed to specify a priority-based ordering among the triggers explicitly for this purpose. However, in the SQL standard (11), and in many commercially available DBMSs, the ordering is based on the time stamp of the creation of the trigger, and even this may not be enforced strictly at run-time. Recall (c.f. formalizing and reasoning) that the execution of the actions of the triggers may generate events that enable some other triggers, which causes a cascaded firing of triggers.

In the rest of this section, we present a detailed discussion of some of the semantic dimensions of the triggers.

Granularity of the Modifications

In relational database systems, a particular modification (insertion, deletion, update) may be applied to a single tuple or to a set of tuples. Similarly, in an object-oriented database system, the modifications may be applied to a single object or to a collection of objects—instances of a given class. Based on this distinction, the active rules can be made to react in a *tuple/instance* manner or in a *set-oriented* manner. An important observation is that this type of granularity is applicable in two different places in the active rules: (1) the events to which a particular rule reacts (is "awoken" by) and (2) the modifications executed by the action part of the rules. Typically, in the DBMS that complies with the SQL standard, this distinction is specified by using FOR EACH ROW (tuple-oriented) or FOR EACH STATEMENT (set-oriented) specification in the respective triggers. In our motivational scenario, if one would like the trigger Update-Salary to react to the modifications of the individual tuples, which correspond to the behavior illustrated in Fig. 2a, its specification should be:

```
CREATE TRIGGER Update-Salary-Check
ON UPDATE OF Employee.Salary
FOR EACH ROW
IF SELECT(...)
...
```

Coupling Among Trigger's Components

Because each trigger that conforms to the ECA paradigm has three distinct parts—the Event, Condition, and Action—one of the important questions is how they are synchronized. This synchronization is often called the *coupling* among the triggers components.

- *E-C coupling:* This dimension describes the temporal relationship among the events that enable certain triggers and the time of evaluating their conditions parts, with respect to the transaction in which the events were generated. With *immediate coupling*, the conditions are evaluated as soon as the basic modification that produced the events is completed. Under the *delayed coupling* mode, the evaluation of the conditions is delayed until a specific point (e.g., a special "event" takes place such as an explicit rule-processing point in the transaction). Specifically, if this special event is the attempt to *commit* the transaction, then the coupling is also called *deferred*.
- *C-A coupling:* Similarly, for a particular trigger, a temporal relationship exists between the evaluation of its condition and (if satisfied) the instant its action is executed. The options are the same as for the *E-C* coupling: *immediate*, in which case the action is executed as soon as the condition-evaluation is completed (in case it evaluates to true); *delayed,* which executes the action at some special point/event; and *deferred*, which is the case when the actions are executed at the end (just before *commit*) of the transaction in which the condition is evaluated.

A noteworthy observation at this point is that the semantic dimensions should not be understood as isolated completely from each other but, to the contrary, their values may be correlated. Among the other reasons, this correlation exists because the triggers manager cannot be implemented in isolation from the query optimizer and the transaction manager. In particular, the coupling modes discussed above are not independent from the transaction processing model and its relationship with the individual parts of the triggers. As another semantic dimension in this context, one may consider whether the conditions evaluation and the actions executions should be executed in the same transaction in which the triggering events have occurred (note that the particular transaction may be aborted because of the effects of the triggers processing). In a typical DBMS setting, in which the ACID (atomicity, consistency, isolation, and durability) properties of the transactions must be ensured, one would like to maintain the conditions evaluations and actions executions within the same transaction in which the triggering event originated. However, if a more sophisticated transaction management is available [e.g., nested transactions (24)] they may be processed in a separate subtransaction(s), in which the failure of a subtransaction may cause the failure of a parent transaction in which the events originated, or in two different transactions. This transaction is known commonly as a *detached* coupling mode.

Events Consumption and Composition

These dimensions describe how a particular event is treated when processing a particular trigger that is enabled due to its occurrence, as well as how the impact of the net effect of a set of events is considered.

One of the differences in the behavior of a particular ADBS is caused by the selection of the *scope*(23) of the event consumptions:

- *NO consumption:* the evaluation and the execution of the conditions part of the enabled/instantiated triggers have no impact on the triggering event. In essence, this means that the same event can enable a particular trigger over and over again. Typically, such behavior is found in the production rule systems used in expert systems (15).
- *Local consumption:* once an instantiated trigger has proceeded with its condition part evaluation, that trigger can no longer be enabled by the same event. However, that particular event remains eligible for evaluation of the condition the other triggers that it has enabled. This feature is the most common in the existing active database systems. In the setting of our motivational scenario, assume that we have another trigger, for example, Maintain-Statistics, which also reacts to an insertion of new employees by increasing properly the total number of the hired employees in the respective departmental relations. Upon insertion of a set of new employees, both New-Employee-Salary-Check and Maintain-Statistics triggers will be enabled. Under the local consumption mode, in case the New-Employee-Salary-Check trigger executes first, it is no longer enabled by the same insertion. The Maintain-Statistics trigger, however, is left enabled and will check its condition and/or execute its action.
- *Global consumption:* Essentially, global consumption means that once the first trigger has been selected for its processing, a particular event can no longer be used to enable any other triggers. In the settings of our motivational scenario, once the given trigger New-Employee-Salary-Check has been selected for evaluation of its condition, it would also disable the Maintain-Statistics despite that it never had its condition checked. In general, this type of consumption is appropriate for the settings in which one can distinguish among "regular" rules and "exception" rules that are enabled by the same event. The "exception" not only has a higher priority, but it also disables the processing of the "regular" rule.

A particular kind of event composition, which is encountered in practice, frequently is the *event net effect*. The basic distinction is whether the system should consider the impact of the occurrence of a particular event, regardless of what are the subsequent events in the transaction, or consider the possibility of invalidating some of the events that have occurred earlier. As a particular example, the following intuitive policy for computing the net effects has been formalized and implemented in the Starburst system (2):

- If a particular tuple is created (and possibly updated) in a transaction, and subsequently deleted within that same transaction, the net effect is *null.*

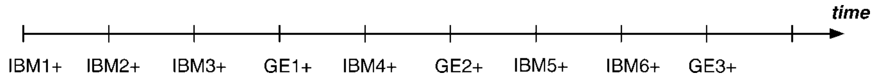


Figure 3. Composite events and consumption.

- If a particular tuple is created (respectively, updated) in a transaction, and that tuple is updated subsequently several times, the net effect is the creation of the final version of that tuple (respectively, the single update equivalent to the final value).
- If a particular tuple is updated and deleted subsequently in a given transaction, then the net effect is the deletion of the original tuple.

Combining the computation of the net effects in systems that allow specification of composite events via an event algebra (5,17) is a very complex problem. The main reason is that in a given algebra, the detection of a particular composite event may be in a state in which several different instances of one of its constituent events have occurred. Now, the question becomes what is the policy for *consuming* the primitive events upon a detection of a composite one. An illustrative example is provided in Fig. 3. Assume that the elementary (primitive) events correspond to tickers from the stockmarket and the user is interested in the composite event: *CE = (two consecutive increases of the IBM stock) AND (two consecutive increases of the General Electric [GE] stock).* Given the timeline for the sequence of events illustrated in Fig. 3, upon the second occurrence of the GE stock increase (GE2+), the desired composite event CE can be detected. However, now the question becomes *which* of the primitive events should be used for the detection of CE (6 ways exist to couple IBM-based events), and *how* should the rest of the events from the history be consumed for the future (e.g., if GE2+ is not consumed upon the detection of CE, then when GE3+ occurs, the system will be able to detect another instance of CE). Chakravarthy et al. (5) have identified four different contexts (*recent, chronicle, continuous,* and *cumulative*) of consuming the earlier occurrences of the primitive constituent events which enabled the detection of a given composite event.

Data Evolution

In many ADBSs, it is important to query the history concerning the execution of the transaction(s). For example, in our motivational scenario, one may envision a modified constraint that states that the average salary increase in the enterprise should not exceed 5% from its previous value when new employees are and/or inserted when the salaries of the existing employees are updated. Clearly, in such settings, the conditions part of the respective triggers should compare the current state of the database with the older state.

When it comes to past database states, a special syntax is required to specify properly the queries that will retrieve the correct information that pertains to the prior database states. It can be speculated that every single state that starts from the *begin* point of a particular transaction should be available for inspection; however, in practice, only a few such states are available (c.f. Ref. (23)):

- *Pretransaction state:* the state of the database just before the execution of the transaction that generated the enabling event.
- *Last consideration state:* given a particular trigger, the state of the database after the last time that trigger has been considered (i.e., for its condition evaluation).
- *Pre-event state:* given a particular trigger, the state of the database just before the occurrence of the event that enabled that trigger.

Typically, in the existing commercially available DBMS that offers active capabilities, the ability to query the past states refers to the pretransaction state. The users are given the keywords OLD and NEW to specify declaratively which part needs to be queried when specifying the condition part of the triggers (11).

Another option for inspecting the history of the active database system is to query explicitly the set of occurred events. The main benefit of this option is the increased flexibility to specify the desired behavioral aspects of a given application. For example, one may wish to query not all the items affected by a particular transaction, but only the ones that participated in the generation of the given composite event that enabled a particular trigger (5). Some prototype systems, [e.g., Chimera (25) offer this extended functionality, however, the triggers in the commercially available DBMS that conform to the SQL standard are restricted to querying the database states only (c.f., the OLD and NEW above).

Recent works (26) have addressed the issues of extending the capabilities of the commercially available ORDBMS Oracle 10g (27) with features that add a flexibility for accessing various portions (states) of interest throughout the evolution of the ADBS, which enable sophisticated management of events for wide variety of application domains.

Effects Ordering

We assumed that the execution of the action part of a particular trigger occurs not only *after* the occurrence of the event, but also after the effects of executing the modifications that generated that event have been incorporated. In other words, the effects of executing a particular trigger were adding to the effects of the modifications that were performed by its enabling event. Although this seems to be the most intuitive approach, in some applications, such as alerting or security monitoring, it may be desirable to have the action part of the corresponding trigger execute *before* the modifications of the events take place, or even *instead* of the modifications.

Typical example is a trigger that detects when an unauthorized user has attempted to update the value of a particular tuple in a given relation. Before executing the user's request, the respective log-file needs to be updated properly. Subsequently, the user-initiated transaction must be aborted; instead, an alert must be issued to the database administrator. Commercially available DBMS offer the flexibility of stating the BEFORE, AFTER, and INSTEAD preferences in the specification of the triggers.

Conflict Resolution and Atomicity of Actions

We already mentioned that if more than one trigger is enabled by the occurrence of a particular event, some selection must be performed to evaluate the respective conditions and/or execute the actions part. From the most global perspective, one may distinguish between the *serial* execution, which selects a single rule according to a predefined policy, and a *parallel* execution of all the enabled triggers. The latter was envisioned in the HiPAC active database systems (c.f. Ref. (28)) and requires sophisticated techniques for concurrency management. The former one can vary from specifying the total priority ordering completely by the designer, as done in the Postgres system (4), to partial ordering, which specifies an incomplete precedence relationship among the triggers, as is the case in the Starburst system (20). Although the total ordering among the triggers may enable a deterministic behavior of the active database, it may be too demanding on the designer, who always is expected to know exactly the intended behavior of all the available rules (23). Commercial systems that conform with the SQL99 standard do not offer the flexibility of specifying an ordering among the triggers. Instead, the default ordering is by the timestamp of their creation.

When executing the action part of a given trigger, a particular modification may constitute an enabling event for some other trigger, or even for a new instance of the same trigger whose action's execution generated that event. One option is to interrupt the action of the currently executing trigger and process the triggers that were "awoken" by it, which could result in cascaded invocation where the execution of the trigger that produced the event is suspended temporarily. Another option is to ignore the occurrence of the generated event temporarily, until the action part of the currently executing trigger is completed (atomic execution). This action illustrates that the values in different semantic dimensions are indeed correlated. Namely, the choice of the atomicity of the execution will impact the value of the E-C/C-A coupling modes: one cannot expect an immediate coupling if the execution of the actions is to be atomic.

Expressiveness Issues

As we illustrated, the choice of values for a particular semantic dimension, especially when it comes to the relationship with the transaction model, may yield different outcomes of the execution of a particular transaction by the DBMS (e.g., deferred coupling will yield different behavior from the immediate coupling). However, another subtle aspect of the active database systems is dependent strongly on their chosen semantic dimensions – the expressive power. Picouet and Vianu (29) introduced a broad model for active databases based on the unified framework of relational Turing machines. By restricting some of the values of the subset of the semantic dimensions and thus capturing the interactions between the sequence of the modifications and the triggers, one can establish a yardstick to compare the expressive powers of the various ADBSs. For example, it can be demonstrated that:

- The A-RDL system (30) under the immediate coupling mode is equivalent to the Postgres system (4) on ordered databases.
- The Starburst system (2) is incomparable to the Postgres system (4).
- The HiPAC system (28) subsumes strictly the Starburst (2) and the Postgres (4) systems.

Although this type of analysis is extremely theoretical in nature, it is important because it provides some insights that may have an impact on the overall application design. Namely, when the requirements of a given application of interest are formalized, the knowledge of the expressive power of the set of available systems may be a crucial factor to decide which particular platform should be used in the implementation of that particular application.

OVERVIEW OF ACTIVE DATABASE SYSTEMS

In this section, we outline briefly some of the distinct features of the existing ADBS—both prototypes as well as commercially available systems. A detailed discussion of the properties of some systems will also provide an insight of the historic development of the research in the field of ADBS, can be found in Refs. (1) and (7).

Relational Systems

A number of systems have been proposed to extend the functionality of relational DBMS with active rules. Typically, the events in such systems are mostly database modifications (insert, delete, update) and the language to specify the triggers is based on the SQL.

- *Ariel* (31): The Ariel system resembles closely the traditional *Condition* → *Action* rules from expert systems literature (15), because the specification of the Event part is optional. Therefore, in general, NO event consumption exists, and the coupling modes are immediate.
- *Starburst* (2): This system has been used extensively for database-internal applications, such as integrity constraints and views maintenance. Its most notable features include the set-based execution model and the introduction of the net effects when considering the modifications that have led to the occurrence of a particular event. Another particular feature

introduced by the Starburst system is the concept of *rule processing points*, which may be specified to occur during the execution of a particular transaction or at its end. The execution of the action part is atomic.

- *Postgres* (4): The key distinction of Postgres is that the granularity of the modifications to which the triggers react is tuple (row) oriented. The coupling modes between the E-C and the C-A parts of the triggers are immediate and the execution of the actions part is interruptable, which means that the recursive enabling of the triggers is an option. Another notable feature of the Postgres system is that it allows for INSTEAD OF specification in its active rules.

Object-Oriented Systems

One of the distinct features of object-oriented DBMS (OODBMS) is that it has *methods* that are coupled with the definition of the classes that specify the structure of the data objects stored in the database. This feature justifies the preference for using OODBMS for advanced application domains that include extended behavior management. Thus, the implementation of active behavior in these systems is coupled tightly with a richer source of events for the triggers (e.g., the execution of any method).

- *ODE* (32): The ODE system was envisioned as an extension of the C++ language with database capabilities. The active rules are of the C-A type and are divided into constraints and triggers for the efficiency of the implementations. Constraints and triggers are both defined at a class level and are considered a property of a given class. Consequently, they can be inherited. One restriction is that the updates of the individual objects, caused by private member functions, cannot be monitored by constraints and triggers. The system allows for both immediate coupling (called *hard constraints*) and deferred coupling (called *soft constraints*), and the triggers can be declared as executing once-only or perpetually (reactivated).
- *HiPAC* (28): The HiPAC project has pioneered many of the ideas that were used subsequently in various research results on active database systems. Some of the most important contributions were the introduction of the coupling modes and the concept of composite events. Another important feature of the HiPAC system was the extension that provided the so called *delta-relation*, which monitors the net effect of a set of modifications and made it available as a part of the querying language. HiPAC also introduced the visionary features of parallel execution of multiple triggers as subtransactions of the original transaction that generated their enabling events.
- *Sentinel* (5): The Sentinel project provided an active extension of the OODBMS, which represented the active rules as database objects and focused on the efficient integration of the rule processing module within the transaction manager. One of the main novelties discovered this particular research project was the introduction of a rich mechanism for to specify and to detect composite events.
- *SAMOS* (18): The SAMOS active database prototype introduced the concept of an *interval* as part of the functionality needed to manage composite events. A particular novelty was the ability to include the monitoring intervals of interest as part of the specification of the triggers. The underlying mechanism to detect the composite events was based on Colored Petri-Nets.
- *Chimera* (25): The Chimera system was envisioned as a tool that would seamlessly integrate the aspects of object orientation, deductive rules, and active rules into a unified paradigm. Its model has strict underlying logical semantics (fixpoint based) and very intuitive syntax to specify the active rules. It is based on the EECA (Extended-ECA) paradigm, specified in Ref. (23), and it provides the flexibility to specify a wide spectrum of behavioral aspects (e.g., semantic dimensions). The language consists of two main components: (1) declarative, which is used to specify queries, deductive rules, and conditions of the active rules; and (2) procedural, which is used to specify the nonelementary operations to the database, as well as the action parts of the triggers.

Commercially Available Systems

One of the earliest commercially available active database systems was DB2 (3), which integrated trigger processing with the evaluation and maintenance of declarative constraints in a manner fully compatible with the SQL92 standard. At the time it served as a foundation model for the draft of the SQL3 standard. Subsequently, the standard has migrated to the SQL99 version (11), in which the specification of the triggers is as follows:

```
<trigger definition> ::=
  CREATE TRIGGER <trigger name>
  {BEFORE | AFTER} <trigger event> ON
    <table name>
  [REFERENCING <old or new values alias list>]
  [FOR EACH {ROW | STATEMENT}]
  [<trigger condition>]
  <trigger action>
<trigger event> ::= INSERT | DELETE | UPDATE
    [OF <column name list>]
<old or new values alias list> ::= {OLD | NEW}
    [AS] <identifier> | {OLD_TABLE |
        NEW_TABLE} [AS] <identifier>
```

The condition part in the SQL99 triggers is optional and, if omitted, it is considered to be true; otherwise, it can be any arbitrarily complex SQL query. The action part, on the other hand, is any sequence of SQL statements, which includes the invocation of stored procedures, embedded within a single BEGIN – END block. The only statements that are excluded from the available actions pertain to connections, sessions, and transactions processing.

Commercially available DBMS, with minor variations, follow the guidelines of the SQL99 standards.

In particular, the Oracle 10g (27), an object-relational DBMS (ORDBMS), not only adheres to the syntax specifications of the SQL standard for triggers (28), but also provides some additions: The triggering event can be specified as a logical disjunction (ON INSERT OR UPDATE) and the INSTEAD OF option is provided for the action's execution. Also, some system events (startup/shutdown, server error messages), as well as user events (logon/logoff and DDL/DML commands), can be used as enabling events in the triggers specification. Just like in the SQL standard, if more than one trigger is enabled by the same event, the Oracle server will attempt to assign a priority for their execution based on the timestamps of their creation. However, it is not guarantees that this case will actually occur at run time. When it comes to dependency management, Oracle 10g server treats triggers in a similar manner to the stored procedures: they are inserted automatically into the data dictionary and linked with the referenced objects (e.g., the ones which are referenced by the action part of the trigger). In the presence of integrity constraints, the typical executional behavior of the Oracle 10g server is as follows:

1. Run all BEFORE *statement* triggers that apply to the statement.
2. Loop for each row affected by the SQL statement.
   a. Run all BEFORE *row* triggers that apply to the statement.
   b. Lock and change row, and perform integrity constraint checking. (The lock is not released until the transaction is committed.)
   c. Run all AFTER *row* triggers that apply to the statement.
3. Complete deferred integrity constraint checking.
4. Run all AFTER *statement* triggers that apply to the statement.

The Microsoft Server MS-SQL also follows closely the syntax prescribed by the SQL99 standard. However, it has its own additions; for example, it provides the INSTEAD OF option for triggers execution, as well as a specification of a restricted form of composite events to enable the particular trigger. Typically, the statements execute in a tuple-oriented manner for each row. A particular trigger is associated with a single table and, upon its definition, the server generates a virtual table automatically for to access the old data items. For example, if a particular trigger is supposed to react to INSERT on the table Employee, then upon insertion to Employee, a virtual relation called Inserted is maintained for that trigger.

NOVEL CHALLENGES FOR ACTIVE RULES

We conclude this article with a brief description of some challenges for the ADBSs in novel application domains, and with a look at an extended paradigm for declaratively specifying reactive behavior.

Application Domains

Workflow management systems (WfMS) provide tools to manage (modeling, executing, and monitoring) *workflows*, which are viewed commonly as processes that coordinate various cooperative activities to achieve a desired goal. Workflow systems often combine the *data centric* view of the applications, which is typical for information systems, with their *process centric* behavioral view. It has already been indicated (12) that WfMS could benefit greatly by a full use of the tools and techniques available in the DBMS when managing large volumes of data. In particular, Shankar et al. (12) have applied active rules to the WfMS settings, which demonstrates that data-intensive scientific workflows can benefit from the concept of *active tables* associated with the programs. One typical feature of workflows is that many of the activities may need to be executed by distributed agents (*actors* of particular *roles*), which need to be synchronized to optimize the concurrent execution. A particular challenge, from the perspective of triggers management in such distributed settings, is to establish a common (e.g., transaction-like) context for their main components—events, conditions, and actions. As a consequence, the corresponding triggers must execute in a detached mode, which poses problems related not only to the consistency, but also to their efficient scheduling and execution (33).

Unlike traditional database applications, many novel domains that require the management of large quantities of information are characterized by the high volumes of data that arrive very fast in a stream-like fashion (8). One of the main features of such systems is that the queries are no longer instantaneous; they become *continuous*/*persistent* in the sense that users expect the answers to be updated properly to reflect the current state of the streamed-in values. Clearly, one of the main aspects of the continuous queries (CQ) management systems is the ability to react quickly to the changes caused by the variation of the streams and process efficiently the modification of the answers. As such, the implementation of CQ systems may benefit from the usage of the triggers as was demonstrated in the Niagara project (9). One issue related to the scalability of the CQ systems is the very scalability of the triggers management (i.e., many instances of various triggers may be enabled). Although it is arguable that the problem of the scalable execution of a large number of triggers may be coupled closely with the nature of the particular application domain, it has been observed that some general aspects of the scalability are applicable universally. Namely, one can identify similar predicates (e.g., in the conditions) across many triggers and group them into equivalence classes that can be indexed on those predicates. This project may require a more involved system catalog (34), but the payoff is a much more efficient execution of a set of triggers. Recent research has also demonstrated that, to capture the intended semantics of the application domain in dynamic environments, the events may have to be assigned an interval-based semantics (i.e., duration may need to be associated with their detection). In particular, in Ref. (35), the authors have demonstrated that if the commonly accepted instantaneous semantics for events occurrence is used in traffic management settings, one may obtain an unintended meaning for the composite events.

Moving objects databases (MODs) are concerned with the management of large volumes of data that pertain to the location-in-time information of the moving entities, as well as efficient processing of the spatio-temporal queries that pertain to that information (13). By nature, MOD queries are continuous and the answers to the pending queries change because of the changes in the location of the mobile objects, which is another natural setting for exploiting an efficient form of a reactive behavior. In particular, Ref. (14) proposed a framework based on the existing triggers in commercially available systems to maintain the correctness of the continuous queries for trajectories. The problem of the scalable execution of the triggers in these settings occurs when a traffic abnormality in a geographically small region may cause changes to the trajectories that pass through that region and, in turn, invalidate the answers to spatio-temporal queries that pertain to a much larger geographic area. The nature of the continuous queries' maintenance is dependent largely on the model adopted for the mobility representation, and the MOD-field is still very active in devising efficient approaches for the queries management which, in one way or another, do require some form of active rules management.

Recently, the wireless sensor networks (WSNs) have opened a wide range of possibilities for novel applications domains in which the whole process of gathering and managing the information of interest requires new ways of perceiving the data management problems (36). WSN consist of hundreds, possibly thousands, of low-cost devices (sensors) that are capable of measuring the values of a particular physical phenomenon (e.g., temperature, humidity) and of performing some elementary calculations. In addition, the WSNs are also capable of communicating and self-organizing into a network in which the information can be gathered, processed, and disseminated to a desired location. As an illustrative example of the benefits of the ECA-like rules in WSN settings, consider the following scenario (c.f. Ref. (6)): whenever the sensors deployed in a given geographic area of interest have detected that the average level of carbon monoxide in the air over any region larger than 1200 ft^2 exceeds 22%, an alarm should be activated. Observe that here the event corresponds to the updates of the (readings of the) individual sensors; the condition is a continuous query evaluated over the entire geographic zone of interest, and with a nested sub-query of identifying the potentially-dangerous regions. At intuitive level, this seems like a straightforward application of the ECA paradigm. Numerous factors in sensor networks affect the efficient implementation of this type of behavior: the energy resource of the individual nodes is very limited, the communication between nodes drains more current from the battery than the sensing and local calculations, and unlike the traditional systems where there are few vantage points to generate new events, in WSN settings, any sensor node can be an event-generator. The detection of composite events, as well as the evaluation of the conditions, must to be integrated in a fully distributed environment under severe constraints (e.g., energy-efficient routing is a paramount). Efficient implementation of the reactive behavior in a WSN-based databases is an ongoing research effort.

The $(ECA)^2$ Paradigm

Given the constantly evolving nature of the streaming or moving objects data, along with the consideration that it may be managed by distributed and heterogeneous sources, it is important to offer a declarative tool in which the users can actually specify how the triggers themselves should evolve. Users can adjust the events that they monitor, the conditions that they need to evaluate, and the action that they execute. Consider, for example, a scenario in which a set of motion sensors deployed around a region of interest is supposed to monitor whether an object is moving continuously toward that region for a given time interval. Aside from the issues of efficient detection of such an event, the application may require an alert to be issued when the status of the closest air field is such that fewer than a certain number of fighter jets are available. In this setting, both the event detection and the condition evaluation are done in distributed manner and are continuous in nature. Aside from the need of their efficient synchronization, the application demands that when a particular object ceases to move continuously toward the region, the condition should not be monitored any further for that object. However, if the object in question is closer than a certain distance (after moving continuously toward the region of interest for a given time), in turn, another trigger may be enabled, which will notify the infantry personnel. An approach for declarative specification of triggers for such behavior was presented in Ref. (37) where the $(ECA)^2$ paradigm (evolving and context-aware event-condition-action) was introduced. Under this paradigm, for a given trigger, the users can embed children triggers in the specifications, which will become enabled upon the occurrences of certain events in the environment, and only when their respective parent triggers are no longer of interest. The children triggers may consume their parents either completely, by eliminating them from any consideration in the future or partially, by eliminating only the particular instance from the future consideration, but allowing a creation of subsequent instances of the parent trigger. Obviously, in these settings, the coupling modes among the E-C and C-A components of the triggers must to be detached, and for the purpose of their synchronization the concept of meta-triggers was proposed in Ref. (37). The efficient processing of such triggers is still an open challenge.

BIBLIOGRAPHY

1. J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing*, San Francisco: Morgan Kauffman, 1996.
2. J. Widom, The Starburst Active Database Rule System, *IEEE Trans. Knowl. Data Enginee.*, **8**(4): 583–595, 1996.
3. R. Cochrane, H. Pirahesh and N. M. Mattos, *Integrating Triggers and Declarative Constraints in SQL Database Systems*, International Conference on Very Large Databases, 1996.

4. M. Stonebraker, The integration of rule systems and database systems, *IEEE Trans. Knowl. Data Enginee.*, **4**(5): 416–432, 1992.

5. S. Chakravarthy, V. Krishnaprasad, E. Answar, and S. K. Kim, *Composite Events for Active Databases: Semantics, Contexts and Detection*, International Conference on Very Large Databases (VLDB), 1994.

6. M. Zoumboulakis, G. Roussos, and A. Poulovassilis, *Active Rules for Wireless Networks of Sensors & Actuators*, International Conference on Embedded Networked Sensor Systems, 2003.

7. N. W. Paton, *Active Rules in Database Systems*, New York: Springer Verlag, 1999.

8. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, *Models and Issues in Data Stream Systems*, International Conference on Principles of Database Systems, 2002.

9. J. Chen, D.J. DeWitt, F. Tian, and Y. Wang, *NiagaraCQ: A Scalable Continuous Query System for Internet Databases*, ACM SIGMOD International Conference on Management of Data, 2000.

10. A. Carzaniga, D. Rosenblum, and A. Wolf, *Achieving Scalability and Expressiveness in an Internet-scale Event Notification Service,* ACM Symposium on Principles of Distributed Computing, 2000.

11. ANSI/ISO International Standard: Database Language SQL. Available: http://webstore.ansi.org.

12. S. Shankar, A. Kini, D. J. DeWitt, and J. F. Naughton, Integrating databases and workflow systems, *SIGMOD Record*, **34**(3): 5–11, 2005.

13. R.H. Guting and M. Schneider, *Moving Objects Databases*, San Francisco: Morgan Kaufman, 2005.

14. G. Trajcevski and P. Scheuermann, Reactive maintenance of continuous queries, *Mobile Comput. Commun. Rev.*, **8**(3): 22–33, 2004.

15. L. Brownston, K. Farrel, E. Kant, and N. Martin, *Programming Expert Systems in OPS5: An Introduction to Rule-Base Programming*, Boston: Addison-Wesley, 2005.

16. A.P. Sistla and O. Wolfson, *Temporal Conditions and Integrity Constraints in Active Database Systems ACM SIGMOD*, International Conference on Management of Data, 1995.

17. S. Gatziu and K. R. Ditrich, *Events in an Active Object-Oriented Database System,* International Workshop on Rules in Database Systems, 1993.

18. K. Dittrich, H. Fritschi, S. Gatziu, A. Geppert, and A. Vaduva, SAMOS in hindsight: Experiences in building an active object-oriented DBMS, *Informat. Sys.*, **30**(5): 369–392, 2003.

19. S.D. Urban, T. Ben-Abdellatif, S. Dietrich, and A. Sundermier, Delta abstractions: a technique for managing database states in runtime debugging of active database rules, *IEEE-TKDE*, **15**(3): 597–612, 2003.

20. C. Baral, J. Lobo, and G. Trajcevski, *Formal Characterization of Active Databases: Part II,* International Conference on Deductive and Object-Oriented Databases (DOOD), 1997.

21. E. Baralis and J. Widom, An algebraic approach to static analysis of active database rules, *ACM Trans. Database Sys.*, **27**(3): 289–332, 2000.

22. S.D. Urban, M.K. Tschudi, S.W. Dietrich, and A.P. Karadimce, Active rule termination analysis: an implementation and evaluation of the refined triggering graph method, *J. Intell. Informat. Sys.*, **14**(1): 29–60, 1999.

23. P. Fraternali and L. Tanca, A structured approach for the definition of the semantics of active databases, *ACM Trans. Database Sys.*, **22**(4): 416–471, 1995.

24. G. Weikum and G. Vossen, *Transactional Information Systems: Theory, Algorithms and the Practice of Concurrency Control*, San Francisco: Morgan Kauffman, 2001.

25. P. Fraternali and S. Parabochi, Chimera: a language for designing rule applications, in: N. W. Paton (ed.), *Active Rules in Database System*, Berlin: Springer-Verlog, 1999.

26. M. Thome, D. Gawlick, and M. Pratt, *Event Processing with an Oracle Database*, SIGMOD International Conference on Management of Data, 2005.

27. K. Owens, *Programming Oracle Triggers and Stored Procedures*, (3rd ed.), O'Reily Publishers, 2003.

28. U. Dayal, A.P. Buchmann, and S. Chakravarthy, The HiPAC project, in J. Widom and S. Ceri, *Active Database Systems. Triggers and Rules for Advanced Database Processing*, San Francisco: Morgan Kauffman, 1996.

29. P. Picouet and V. Vianu, Semantics and expressiveness issues in active databases, *J. Comp. Sys. Sci.*, **57**(3): 327–355, 1998.

30. E. Simon and J. Kiernan, The A-RDL system, in J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing*, San Francisco: Morgan Kauffman, 1996.

31. E. N. Hanson, *Rule Condition Testing and Action Execution in Ariel*, ACM SIGMOD International Conference on Management of Data, 1992.

32. N. Gehani and H.V. Jagadish, *ODE as an Active Database: Constraints and Triggers*, International Conference on Very Large Databases, 1992.

33. S. Ceri, C. Gennaro, S. Paraboschi, and G. Serazzi, Effective scheduling of detached rules in active databases, *IEEE Trans. Knowl. Data Enginee.*, **16**(1): 2–15, 2003.

34. E.N. Hanson, S. Bodagala, and U. Chadaga, Trigger condition testing and view maintenance using optimized discrimination networks, *IEEE Trans. Know. Data Enginee.*, **16**(2): 281–300, 2002.

35. R. Adaikkalvan and S. Chakravarthy, *Formalization and Detection of Events Using Interval-Based Semantics*, International Conference on Advances in Data Management, 2005.

36. F. Zhao and L. Guibas, *Wireless Sensor Networks: An Information Processing Approach*, San Francisco: Morgan Kaufmann, 2004.

37. G. Trajcevski, P. Scheuermann, O. Ghica, A. Hinze, and A. Voisard, *Evolving Triggers for Dynamic Environments*, International Conference on Extending the Database Technology, 2006.

PETER SCHEUERMANN
GOCE TRAJCEVSKI
Northwestern University
Evanston, Illinois

A

ALGEBRAIC CODING THEORY

INTRODUCTION

In computers and digital communication systems, information almost always is represented in a binary form as a sequence of bits each having the values 0 or 1. This sequence of bits is transmitted over a *channel* from a sender to a receiver. In some applications the channel is a storage medium like a DVD, where the information is written to the medium at a certain time and retrieved at a later time. Because of the physical limitations of the channel, some transmitted bits may be corrupted (the channel is *noisy*) and thus make it difficult for the receiver to reconstruct the information correctly.

In algebraic coding theory, we are concerned mainly with developing methods to detect and correct errors that typically occur during transmission of information over a noisy channel. The basic technique to detect and correct errors is by introducing redundancy in the data that is to be transmitted. This technique is similar to communicating in a natural language in daily life. One can understand the information while listening to a noisy radio or talking on a bad telephone line, because of the redundancy in the language.

For example, suppose the sender wants to communicate one of 16 different messages to a receiver. Each message m can then be represented as a binary quadruple (c_0, c_1, c_2, c_3). If the message (0101) is transmitted and the first position is corrupted such that (1101) is received, this leads to an uncorrectable error because this quadruple represents a different valid message than the message that was sent across the channel. The receiver will have no way to detect and to correct a corrupted message in general, because any quadruple represents a valid message.

Therefore, to combat errors the sender *encodes* the data by introducing redundancy into the transmitted information. If M messages are to be transmitted, the sender selects a subset of M binary n-tuples, where $M < 2^n$. Each of the M messages is encoded into an n-tuple. The set consisting of the M n-tuples obtained after encoding is called a binary (n, M) *code*, and the elements are called *codewords*. The codewords are sent over the channel.

It is customary for many applications to let $M = 2^k$, such that each message can be represented uniquely by a k-tuple of information bits. To encode each message the sender can append $n - k$ parity bits depending on the message bits and use the resulting n bit codeword to represent the corresponding message.

A binary code C is called a linear code if the sum (modulo 2) of two codewords is again a codeword. This is always the case when the parity bits are linear combinations of the information bits. In this case, the code C is a vector space of dimension k over the binary field of two elements, containing $M = 2^k$ codewords, and is called an $[n, k]$ code. The main reason for using linear codes is that these codes have more algebraic structure and are therefore often easier to analyze and to decode in practical applications.

The simplest example of a linear code is the $[n, n - 1]$ *even-weight code* (or parity-check code). The encoding consists of appending a single parity bit to the $n - 1$ information bits so that the codeword has an even number of ones. Thus, the code consists of all 2^{n-1} possible n-tuples of even weight, where the *weight* of a vector is the total number of ones in its components. This code can detect all errors in an odd number of positions, because if such an error occurs, the received vector also will have odd weight. The even-weight code, however, can only detect errors. For example, if (000…0) is sent and the first bit is corrupted, then (100…0) is received. Also, if (110…0) was sent and the second bit was corrupted, then (100…0) is received. Hence, the receiver cannot correct this single error or, in fact, any other error.

An illustration of a code that can correct any single error is shown in Fig. 1. The three circles intersect and divide the plane into seven finite areas and one infinite area. Each finite area contains a bit c_i for $i = 0, 1,\ldots, 6$. Each of the 16 possible messages, denoted by (c_0, c_1, c_2, c_3), is encoded into a codeword $(c_0, c_1, c_2, c_3, c_4, c_5, c_6)$, in such a way that the sum of the bits in each circle has an even parity.

In Fig. 2, an example is shown of encoding the message (0011) into the codeword (0011110). Because the sum of two codewords also obeys the parity-checks and thus is a codeword, the code is a linear [7, 4] code.

Suppose, for example, that the transmitted codeword is corrupted in the bit c_1 such that the received word is (0111110). Then, calculating the parity of each of the three circles, we see that the parity fails for the upper circle as well as for the leftmost circle, whereas the parity of the rightmost circle is correct. Hence, from the received vector, indeed we can conclude that bit c_1 is in error and should be corrected. In the same way, any single error can be corrected by this code.

LINEAR CODES

An (n, M) code is simply a set of M vectors of length n with components from a finite field $F_2 = \{0, 1\}$, where addition and multiplication are done modulo 2. For practical applications it is desirable to provide the code with more structure. Therefore, linear codes often are preferred. A linear $[n, k]$ code C is a k-dimensional subspace C of F_2^n, where F_2^n is the vector space of n-tuples with coefficients from the finite field F_2.

A linear code C usually is described in terms of a generator matrix or a parity-check matrix. A *generator matrix* G of C is a $k \times n$ matrix whose row space is the code C; i.e.,

$$C = \{\mathbf{x}G | \mathbf{x} \in F_2^k\}$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

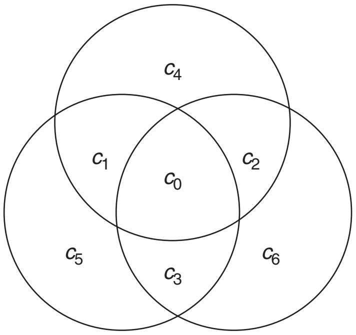


Figure 1. The message (c_0, c_1, c_2, c_3) is encoded into the codeword $(c_0, c_1, c_2, c_3, c_4, c_5, c_6)$, where c_4, c_5, c_6 are chosen such that there is an even number of ones within each circle.

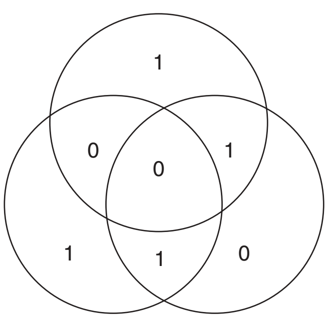


Figure 2. Example of the encoding procedure given in Fig. 1. The message (0011) is encoded into (0011110). Note that there is an even number of ones within each circle.

A *parity-check matrix* H is an $(n - k) \times n$ matrix such that

$$C = \{c \in F_2^n | cH^{tr} = 0\}$$

where H^{tr} denotes the transpose of H.

Example. The codewords in the code in the previous section are the vectors $(c_0, c_1, c_2, c_3, c_4, c_5, c_6)$ that satisfy the following system of *parity-check equations*:

$$\begin{array}{ccccccccc} c_0 & + & c_1 & + & c_2 & + & c_4 & = & 0 \\ c_0 & + & c_1 & + & c_3 & + & c_5 & = & 0 \\ c_0 & + & c_2 & + & c_3 & + & c_6 & = & 0 \end{array}$$

where all additions are modulo 2. Each of the three parity-check equations corresponds to one of the three circles.

The coefficient matrix of the parity-check equations is the *parity-check matrix*

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \qquad (1)$$

The code C is therefore given by

$$C = \{\mathbf{c} = (c_0, c_1, \ldots, c_6) | cH^{tr} = 0\}$$

A generator matrix for the code in the previous example is given by

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Two codes are *equivalent* if the codewords in one of the codes can be obtained by a fixed permutation of the positions in the codewords in the other code. If G (respectively, H) is a generator (respectively, parity-check) matrix of a code, then the matrices obtained by permuting the columns of these matrices in the same way give the generator matrix (respectively, parity-check) matrix of the permuted code.

The *Hamming distance* between $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$ in F_2^n is the number of positions in which they differ. That is,

$$d(\mathbf{x}, \mathbf{y}) = |\{i | x_i \neq y_i, 0 \leq i \leq n - 1\}|$$

The Hamming distance has the properties required to be a metric:

1. $d(\mathbf{x}, \mathbf{y}) \geq 0$ for all $\mathbf{x}, \mathbf{y} \in F_2^n$ and equality holds if and only if $\mathbf{x} = \mathbf{y}$.
2. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in F_2^n$.
3. $d(\mathbf{x}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$ for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in F_2^n$.

For any code C, one of the most important parameters is its *minimum distance*, defined by

$$d = \min\{d(\mathbf{x}, \mathbf{y}) | \mathbf{x} \neq \mathbf{y}, \mathbf{x}, \mathbf{y} \in C\}$$

The *Hamming weight* of a vector $\mathbf{x}$ in F_2^n is the number of nonzero components in $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$. That is,

$$w(\mathbf{x}) = |\{i | x_i \neq 0, 0 \leq i \leq n - 1\}| = d(\mathbf{x}, \mathbf{0})$$

Note that because $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{x} - \mathbf{y}, \mathbf{0}) = w(\mathbf{x} - \mathbf{y})$ for a linear code C, it follows that

$$d = \min\{w(\mathbf{z}) | \mathbf{z} \in C, \mathbf{z} \neq \mathbf{0}\}$$

Therefore, finding the minimum distance of a linear code is equivalent to finding the minimum nonzero weight among all codewords in the code.

If $w(\mathbf{c}) = i$, then $\mathbf{c}H^{tr}$ is the sum of i columns of H. Hence, an alternative description of the minimum distance of a linear code is as follows: the smallest d such that d linearly dependent columns exist in the parity-check matrix. In particular, to obtain a binary linear code of minimum distance of at least three, it is sufficient to select the columns of a parity-check matrix to be distinct and nonzero.

Sometimes we include d in the notation and refer to an $[n, k]$ code with minimum distance d as an $[n, k, d]$ code. If t components are corrupted during transmission of a codeword, we say that t errors have occurred or that an error $\mathbf{e}$ of weight t has occurred [where $\mathbf{e} = (e_0, e_1, \dots, e_{n-1}) \in F_2^n$, where $e_i = 1$ if and only if the ith component was corrupted, that is, if $\mathbf{c}$ was sent, $\mathbf{c} + \mathbf{e}$ was received].

The *error-correcting capability* of a code is defined as

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor$$

where $\lfloor x \rfloor$ denotes the largest integer $\leq x$.

A code with minimum distance d can correct all errors of weight t or less, because if a codeword $\mathbf{c}$ is transmitted and an error $\mathbf{e}$ of weight $e \leq t$ occurs, the received vector $\mathbf{r} = \mathbf{c} + \mathbf{e}$ is closer in Hamming distance to the transmitted codeword $\mathbf{c}$ than to any other codeword. Therefore, decoding any received vector to the closest codeword corrects all errors of weight $\leq t$.

The code can be used for error detection only. The code can detect all errors of weight $< d$ because if a codeword is transmitted and the error has weight $< d$, then the received vector is not another codeword.

The code also can be used for a combination of error-correction and error-detection. For a given $e \leq t$, the code can correct all errors of weight $\leq e$ and in addition can detect all errors of weight at most $d - e - 1$. This is caused by the fact that no vector in F_2^n can be at distance $\leq e$ from one codeword and at the same time at a distance $\leq d - e - 1$ from another codeword. Hence, the algorithm in this case is to decode a received vector to a codeword at distance $\leq e$ if such a codeword exists and otherwise detect an error.

If C is an $[n, k]$ code, the *extended code* C^{ext} is the $[n + 1, k]$ code defined by

$$c^{\text{ext}} = \left\{ (c_{\text{ext}}, c_0, c_1, \dots c_{n-1}) \middle| (c_0, c_1, \dots, c_{n-1}) \in C, \right.$$
$$\left. c_{\text{ext}} = \sum_{i=0}^{n-1} c_i \right\}$$

That is, each codeword in C is extended by one parity bit such that the Hamming weight of each codeword becomes even. In particular, if C has odd minimum distance d, then the minimum distance of C^{ext} is $d + 1$. If H is a parity-check matrix for C, then a parity check matrix for C^{ext} is

$$\begin{pmatrix} 1 & \mathbf{1} \\ 0^{tr} & H \end{pmatrix}$$

where $\mathbf{1} = (11\dots1)$.

For any linear $[n, k]$ code C, the *dual code* $C^{\perp}$ is the $[n, n - k]$ code defined by

$$C^{\perp} = \{\mathbf{x} \in F_2^n | (\mathbf{x}, c) = 0 \text{ for all } \mathbf{c} \in C\}$$

where $(\mathbf{x}, \mathbf{c}) = \sum_{i=0}^{n-1} x_i c_i$. We say that $\mathbf{x}$ and $\mathbf{c}$ are *orthogonal* if $(\mathbf{x}, \mathbf{c}) = 0$. Therefore $C^{\perp}$ consists of all n-tuples that are orthogonal to all codewords in C and vice versa; that is, $(C^{\perp})^{\perp} = C$. It follows that $C^{\perp}$ has dimension $n - k$ because it consists of all vectors that are solutions of a system of equations with coefficient matrix G of rank k. Hence, the parity-check matrix of $C^{\perp}$ is a generator matrix of C, and similarly the generator matrix of $C^{\perp}$ is a parity-check matrix of C. In particular, $GH^{tr} = O$ [the $k \times (n - k)$ matrix of all zeros].

Example. Let C be the $[n, n - 1, 2]$ even-weight code where

$$G = \begin{pmatrix} 1 & 0 & \dots & 0 & 1 \\ 0 & 1 & \dots & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 1 \end{pmatrix}$$

and

$$H = (\,1 \quad 1 \quad \dots \quad 1 \quad 1\,)$$

Then $C^{\perp}$ has H and G as its generator and parity-check matrices, respectively. It follows that $C^{\perp}$ is the $[n, 1, n]$ *repetition code* consisting of the two codewords $(00 \cdots 000)$ and $(11 \cdots 111)$.

Example. Let C be the $[2^m - 1, 2^m - 1 - m, 3]$ code, where H contains all nonzero m-tuples as its columns. This is known as the *Hamming code*. In the case when $m = 3$, a parity-check matrix already is described in Equation (1). Because all columns of the parity-check matrix are distinct and nonzero, the code has minimum distance of at least 3. The minimum distance is indeed 3 because three columns exist whose sum is zero; in fact, the sum of any two columns of H equals another column in H for this particular code.

The dual code $C^{\perp}$ is the $[2^m - 1, m, 2^{m-1}]$ *simplex code*, all of whose nonzero codewords have weight 2^{m-1}. This follows because the generator matrix has all nonzero vectors as its columns. In particular, taking any linear combination of rows, the number of columns with odd parity in the corresponding subset of rows equals 2^{m-1} (and the number with even parity is $2^{m-1} - 1$).

The extended code of the Hamming code is a $[2^m, 2^m - 1 - m, 4]$ code. Its dual code is a $[2^m, m + 1, 2^{m-1}]$ code that is known as the first-order Reed–Muller code.

SOME BOUNDS ON CODES

The *Hamming bound* states that for any (n, M, d) code, we have

$$M \sum_{i=0}^{e} \binom{n}{i} \leq 2^n$$

where $e = \lfloor (d-1)/2 \rfloor$. This follows from the fact that the M spheres

$$S_{\mathbf{c}} = \{\mathbf{x} | d(\mathbf{x}, \mathbf{c}) \le e\}$$

centered at the codewords $\mathbf{c} \in C$ are disjoint and that each sphere contains

$$\sum_{i=0}^{e} \binom{n}{i}$$

vectors.

If the spheres fill the whole space, that is,

$$\bigcup_{\mathbf{c} \in C} S_{\mathbf{c}} = F_2^n$$

then C is called *perfect*. The binary *linear* perfect codes are as follows:

- the $[n, 1, n]$ repetition codes for all odd n
- the $[2^m - 1, 2^m - 1 - m, 3]$ Hamming codes H_m for all $m \ge 2$
- the [23, 12, 7] Golay code G_{23}

We will return to the Golay code later.

GALOIS FIELDS

Finite fields, also known as Galois fields, with p^m elements exist for any prime p and any positive integer m. A Galois field of a given order p^m is unique (up to isomorphism) and is denoted by F_{p^m}.

For a prime p, let $F_p = \{0, 1, \ldots, p-1\}$ denote the integers modulo p with the two operations addition and multiplication modulo p.

To construct a Galois field with p^m elements, select a polynomial $f(x)$ with coefficients in F_p, which is irreducible over F_p; that is, $f(x)$ cannot be written as a product of two polynomials with coefficients from F_p of degree ≥ 1 (irreducible polynomials of any degree m over F_p exist).

Let

$$F_{p^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_0 | a_0, \ldots, a_{m-1} \in F_p\}$$

Then F_{p^m} is a finite field when addition and multiplication of the elements (polynomials) are done modulo $f(x)$ and modulo p. To simplify the notations, let α denote a zero of $f(x)$, that is, $f(\alpha) = 0$. If such an α exists, formally it can be defined as the equivalence class of x modulo $f(x)$. For coding theory, $p = 2$ is by far the most important case, and we assume this from now on. Note that for any $a, b \in F_{2^m}$,

$$(a+b)^2 = a^2 + b^2$$

Example. The Galois field F_{2^4} can be constructed as follows. Let $f(x) = x^4 + x + 1$ that is an irreducible polynomial over F_2. Then $\alpha^4 = \alpha + 1$ and

$$F_{2^4} = \{a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0 | a_0, a_1, a_2, a_3 \in F_2\}$$

Computing the powers of α, we obtain

$$\alpha^5 = \alpha \cdot \alpha^4 = \alpha(\alpha + 1) = \alpha^2 + \alpha$$
$$\alpha^6 = \alpha \cdot \alpha^5 = \alpha(\alpha^2 + \alpha) = \alpha^3 + \alpha^2$$
$$\alpha^7 = \alpha \cdot \alpha^6 = \alpha(\alpha^3 + \alpha^2) = \alpha^4 + \alpha^3 = \alpha^3 + \alpha + 1$$

and, similarly, all higher powers of α can be expressed as a linear combination of $\alpha^3, \alpha^2, \alpha$, and 1. In particular, $\alpha^{15} = 1$. We get the following table of the powers of α. In the table the polynomial $a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0$ is represented as $a_3a_2a_1a_0$.

i	α^i	i	α^i	i	α^i
0	0001	5	0110	10	0111
1	0010	6	1100	11	1110
2	0100	7	1011	12	1111
3	1000	8	0101	13	1101
4	0011	9	1010	14	1001

Hence, the elements $1, \alpha, \alpha^2, \ldots, \alpha^{14}$ are all the nonzero elements in F_{2^4}. Such an element α that generates the nonzero elements of F_{2^m} is called a *primitive element* in F_{2^m}. An irreducible polynomial $g(x)$ with a primitive element as a zero is called a primitive polynomial. Every finite field has a primitive element, and therefore, the multiplicative subgroup of a finite field is cyclic.

All elements in F_{2^m} are roots of the equation $x^{2^m} + x = 0$. Let β be an element in F_{2^m}. It is important to study the polynomial $m(x)$ of smallest degree with coefficients in F_2, which has β as a zero. This polynomial is called the *minimal polynomial* of β over F_2.

First, observe that if $m(x) = \sum_{i=0}^{\kappa} m_i x^i$ has coefficients in F_2 and β as a zero, then

$$m(\beta^2) = \sum_{i=0}^{\kappa} m_i \beta^{2i} = \sum_{i=0}^{\kappa} m_i^2 \beta^{2i} = \left(\sum_{i=0}^{\kappa} m_i \beta^i\right)^2 = (m(\beta))^2 = 0$$

Hence, $m(x)$ has $\beta, \beta^2, \ldots, \beta^{2^{\kappa-1}}$ as zeros, where κ is the smallest integer such that $\beta^{2^\kappa} = \beta$. Conversely, the polynomial with exactly these zeros can be shown to be a binary irreducible polynomial.

Example. We will find the minimal polynomial of all the elements in F_{2^4}. Let α be a root of $x^4 + x + 1 = 0$; that is, $\alpha^4 = \alpha + 1$. The minimal polynomials over F_2 of α^i for $0 \le i \le 14$ are denoted $m_i(x)$. Observe by the above argument that $m_{2i}(x) = m_i(x)$, where the indices are taken modulo 15. It follows that

$$\begin{aligned}
m_0(x) &= (x+\alpha^0) = x+1\\
m_1(x) &= (x+\alpha)(x+\alpha^2)(x+\alpha^4)(x+\alpha^8) = x^4+x+1\\
m_3(x) &= (x+\alpha^3)(x+\alpha^6)(x+\alpha^{12})(x+\alpha^9)\\
&= x^4+x^3+x^2+x+1\\
m_5(x) &= (x+\alpha^5)(x+\alpha^{10}) = x^2+x+1\\
m_7(x) &= (x+\alpha^7)(x+\alpha^{14})(x+\alpha^{13})(x+\alpha^{11}) = x^4+x^3+1\\
m_9(x) &= m_3(x)\\
m_{11}(x) &= m_7(x)\\
m_{13}(x) &= m_7(x)
\end{aligned}$$

To verify this, one simply computes the coefficients and uses the preceding table of F_{2^4} in the computations. For example,

$$\begin{aligned}
m_5(x) &= (x+\alpha^5)(x+\alpha^{10}) = x^2 + (\alpha^5+\alpha^{10})x + \alpha^5\cdot\alpha^{10}\\
&= x^2+x+1
\end{aligned}$$

This also leads to a factorization into irreducible polynomials,

$$\begin{aligned}
x^{2^4}+x &= x\prod_{j=0}^{14}(x+\alpha^j)\\
&= x(x+1)(x^2+x+1)(x^4+x+1)\\
&\quad (x^4+x^3+x^2+x+1)(x^4+x^3+1)\\
&= xm_0(x)m_1(x)m_3(x)m_5(x)m_7(x)
\end{aligned}$$

In fact, in general it holds that $x^{2^m}+x$ is the product of all irreducible polynomials over F_2 of degree that divides m.

Let $C_i = \{i\,2^j \mod n | j = 0, 1, \ldots\}$, which is called the *cyclotomic coset* of i (mod n). Then, the elements of the cyclotomic coset C_i (mod 2^m-1) correspond to the exponents of the zeros of $m_i(x)$. That is,

$$m_i(x) = \prod_{j\in C_i}(x-\alpha^j)$$

The cyclotomic cosets (mod n) are important in the next section when cyclic codes of length n are discussed.

CYCLIC CODES

Many good linear codes that have practical and efficient decoding algorithms have the property that a cyclic shift of a codeword is again a codeword. Such codes are called cyclic codes.

We can represent the set of n-tuples over F_2^n as polynomials of degree $< n$ in a natural way. The vector

$$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$$

is represented as the polynomial

$$c(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1}.$$

A *cyclic shift*

$$\sigma(\mathbf{c}) = (c_{n-1}, c_0, c_1, \ldots, c_{n-2})$$

of $\mathbf{c}$ is then represented by the polynomial

$$\begin{aligned}
\sigma(c(x)) &= c_{n-1} + c_0x + c_1x^2 + \cdots + c_{n-2}x^{n-1}\\
&= x(c_{n-1}x^{n-1} + c_0 + c_1x + \cdots + c_{n-2}x^{n-2})\\
&\quad + c_{n-1}(x^n+1)\\
&\equiv xc(x) \pmod{x^n+1}
\end{aligned}$$

Example. Rearranging the columns in the parity-check matrix of the [7, 4] Hamming code in Equation (1), an equivalent code is obtained with parity-check matrix

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1\\ 0 & 1 & 0 & 1 & 1 & 1 & 0\\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{2}$$

This code contains 16 codewords, which are represented next in polynomial form:

$$\begin{array}{lrr}
1000110 \leftrightarrow & x^5+x^4+1 = & (x^2+x+1)g(x)\\
0100011 \leftrightarrow & x^6+x^5+x = & (x^3+x^2+x)g(x)\\
1010001 \leftrightarrow & x^6+x^2+1 = & (x^3+x+1)g(x)\\
1101000 \leftrightarrow & x^3+x+1 = & g(x)\\
0110100 \leftrightarrow & x^4+x^2+x = & xg(x)\\
0011010 \leftrightarrow & x^5+x^3+x^2 = & x^2g(x)\\
0001101 \leftrightarrow & x^6+x^4+x^3 = & x^3g(x)\\
0010111 \leftrightarrow & x^6+x^5+x^4+x^2 = & (x^3+x^2)g(x)\\
1001011 \leftrightarrow & x^6+x^5+x^3+1 = & (x^3+x^2+x+1)g(x)\\
1100101 \leftrightarrow & x^6+x^4+x+1 = & (x^3+1)g(x)\\
1110010 \leftrightarrow & x^5+x^2+x+1 = & (x^2+1)g(x)\\
0111001 \leftrightarrow & x^6+x^3+x^2+x = & (x^3+x)g(x)\\
1011100 \leftrightarrow & x^4+x^3+x^2+1 = & (x+1)g(x)\\
0101110 \leftrightarrow & x^5+x^4+x^3+x = & (x^2+x)g(x)\\
0000000 \leftrightarrow & 0 = & 0\\
1111111 \leftrightarrow & x^6+x^5+x^4+x^3+x^2+x+1 & \\
 & = & (x^3+x^2+1)g(x)
\end{array}$$

By inspection it is easy to verify that any cyclic shift of a codeword is again a codeword. Indeed, the 16 codewords in the code are $\mathbf{0}$, $\mathbf{1}$ and all cyclic shifts of (1000110) and (0010111). The unique nonzero polynomial in the code of lowest possible degree is $g(x) = x^3+x+1$, and $g(x)$ is called the *generator polynomial* of the cyclic code. The code consists of all polynomials $c(x)$, which are multiples of $g(x)$. Note that the degree of $g(x)$ is $n-k=3$ and that $g(x)$ divides x^7+1 because

$$x^7+1 = (x+1)(x^3+x+1)(x^3+x^2+1).$$

Therefore the code has a simple description in terms of the set of code polynomials as

$$C = \{c(x)|c(x) = u(x)(x^3+x+1),\quad deg(u(x)) < 4\}$$

This situation holds in general for any cyclic code.

For any cyclic $[n, k]$ code C, we have

$$C = \{c(x)|c(x) = u(x)g(x),\quad deg(u(x)) < k\}$$

for a polynomial $g(x)$ of degree $n-k$ that divides x^n+1.

We can show this as follows: Let $g(x)$ be the generator polynomial of C, which is the nonzero polynomial of smallest degree r in the code C. Then the cyclic shifts $g(x)$, $xg(x)$, $\cdots$, $x^{n-r-1}g(x)$ are codewords as well as any linear combination $u(x)g(x)$, where $\deg(u(x)) < r$. These are the only 2^{n-r} codewords in the code C, because if $c(x)$ is a codeword, then

$$c(x) = u(x)g(x) + s(x), \text{where} \quad deg(s(x)) < deg(g(x))$$

By linearity, $s(x)$ is a codeword and therefore $s(x) = 0$ because $\deg(s(x)) < \deg(g(x))$ and $g(x)$ is the nonzero polynomial of smallest degree in the code. It follows that C is as described previously. Since C has 2^{n-r} codewords, it follows that $n - r = k$; i.e., $\deg(g(x)) = n - k$.

Finally, we show that $g(x)$ divides $x^n + 1$. Let $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$ be a nonzero codeword shifted such that $c_{n-1} = 1$. Then the cyclic shift of $c(x)$ given by $\sigma(c(x)) = c_{n-1} + \ c_0x + c_1x + \cdots + c_{n-2}x^{n-1}$ also is a codeword and

$$\sigma(c(x)) = xc(x) + (x^n + 1)$$

Because both codewords $c(x)$ and $\sigma(c(x))$ are divisible by $g(x)$, it follows that $g(x)$ divides $x^n + 1$.

Because the generator polynomial of a cyclic code divides $x^n + 1$, it is important to know how to factor $x^n + 1$ into irreducible polynomials. Let n be odd. Then an integer m exists such that $2^m \equiv 1 \pmod{n}$ and an element $\alpha \in F_{2^m}$ exists of order n [if ω is a primitive element of F_{2^m}, then α can be taken to be $\alpha = \omega^{(2^m-1)/n}$].

We have

$$x^n + 1 = \prod_{i=0}^{n-1}(x + \alpha^i)$$

Let $m_i(x)$ denote the minimal polynomial of α^i; that is, the polynomial of smallest degree with coefficients in F_2 and having α^i as a zero. The generator polynomial $g(x)$ can be written as

$$g(x) = \prod_{i \in I}(x + \alpha^i)$$

where I is a subset of $\{0, 1, \ldots, n - 1\}$, called the *defining set* of C with respect to α. Then $m_i(x)$ divides $g(x)$ for all $i \in I$. Furthermore, $g(x) \prod_{j=1}^{l} m_{i_j}(x)$ for some i_1, $i_2, \ldots, i_l$.

Therefore we can describe the cyclic code in alternative equivalent ways as

$$\begin{aligned} C &= \{c(x) | m_i(x) \text{divides } c(x), \text{for all } i \in I\} \\ C &= \{c(x) | c(\alpha^i) = 0, \text{for all } i \in I\} \\ C &= \{\mathbf{c} \in F_q^n | \mathbf{c}H^{tr} = 0\} \end{aligned}$$

where

$$H = \begin{pmatrix} 1 & \alpha^{i_1} & \alpha^{2i_1} & \cdots & \alpha^{(n-1)i_1} \\ 1 & \alpha^{i_2} & \alpha^{2i_2} & \cdots & \alpha^{(n-1)i_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{i_l} & \alpha^{2i_l} & \cdots & \alpha^{(n-1)i_l} \end{pmatrix}$$

The encoding for cyclic codes usually is done in one of two ways. Let $u(x)$ denote the information polynomial of degree $< k$. The two ways are as follows:

1. Encode into $u(x)g(x)$.
2. Encode into $c(x) = x^{n-k}u(x) + s(x)$, where $s(x)$ is the polynomial such that
   - $s(x) \equiv x^{n-k}u(x) \pmod{g(x)}$ [thus $g(x)$ divides $c(x)$],
   - $\deg(s(x)) < \deg(g(x))$.

The last of these two methods is systematic; that is, the last k bits of the codeword are the information bits.

BCH CODES

An important task in coding theory is to design codes with a guaranteed minimum distance d that correct all errors of Hamming weight $\lfloor \frac{d-1}{2} \rfloor$. Such codes were designed independently by Bose and Ray-Chaudhuri (1) and by Hocquenghem (2) and are known as BCH codes. To construct a BCH code of *designed distance d*, the generator polynomial is chosen to have $d - 1$ "consecutive" powers of α as zeros

$$\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+d-2}$$

That is, the defining set I with respect to α contains a set of $d - 1$ consecutive integers (mod n). The parity-check matrix of the BCH code is

$$H = \begin{pmatrix} 1 & \alpha^b & \alpha^{2b} & \cdots & \alpha^{(n-1)b} \\ 1 & \alpha^{b+1} & \alpha^{2(b+1)} & \cdots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{b+d-2} & \alpha^{2(b+d-2)} & \cdots & \alpha^{(n-1)(b+d-2)} \end{pmatrix}$$

To show that this code has a minimum distance of at least d, it is sufficient to show that any $d - 1$ columns are linear independent. Suppose a linear dependency between the $d - 1$ columns corresponds to $\alpha^{i_1b}, \alpha^{i_2b}, \ldots, \alpha^{i_{d-1}b}$. In this case the $(d - 1) \times (d - 1)$ submatrix obtained by retaining these columns in H has determinant

$$\begin{aligned} &\begin{vmatrix} \alpha^{i_1b} & \alpha^{i_2b} & \cdots & \alpha^{i_{d-1}b} \\ \alpha^{i_{1(b+1)}} & \alpha^{i_{2(b+1)}} & \cdots & \alpha^{i_{d-1}(b+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{i_1(b+d-2)} & \alpha^{i_2(b+d-2)} & \cdots & \alpha^{i_{d-1}(b+d-2)} \end{vmatrix} \\ = \quad & \alpha^{b(i_1+i_2+\ldots+i_{d-2})} \begin{vmatrix} 1 & 1 & \cdots & 1 \\ \alpha^{i_1} & \alpha^{i_2} & \cdots & \alpha^{i_{d-1}} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha^{(d-2)i_1} & \alpha^{(d-2)i_2} & \cdots & \alpha^{(d-2)i_{d-1}} \end{vmatrix} \\ = \quad & \alpha^{b(i_1+i_2+\ldots+i_{d-2})} \prod_{k<r}(\alpha^{i_k} - \alpha^{i_r}) \neq 0 \end{aligned}$$

because the elements $\alpha^{i_1}, \alpha^{i_2}, \cdots, \alpha^{i_{d-1}}$ are distinct (the last equality follows from the fact that the last determinant is a Vandermonde determinant). It follows that the BCH code has a minimum Hamming distance of at least d.

If $b = 1$, which is often the case, the code is called a *narrow-sense* BCH code. If $n = 2^m - 1$, the BCH code is called a *primitive* BCH code. A binary single error-correcting primitive BCH code is generated by $g(x) = m_1(x)$. The zeros of $g(x)$ are α^{2^i}, $i = 0, 1, \ldots, m - 1$. The parity-check matrix is

$$H = (\,1 \quad \alpha^1 \quad \alpha^2 \quad \ldots \quad \alpha^{2^m-2}\,)$$

This code is equivalent to the Hamming code because α is a primitive element of F_{2^m}.

To construct a binary double-error-correcting primitive BCH code, we let $g(x)$ have α, α^2, α^3, α^4 as zeros. Therefore, $g(x) = m_1(x)m_3(x)$ is a generator polynomial of this code. The parity-check matrix of a double-error-correcting BCH code is

$$H = \begin{pmatrix} 1 & \alpha^1 & \alpha^2 & \ldots & \alpha^{2^m-2} \\ 1 & \alpha^3 & \alpha^6 & \ldots & \alpha^{3(2^m-2)} \end{pmatrix}$$

In particular, a binary double-error correcting BCH code of length $n = 2^4 - 1 = 15$, is obtained by selecting

$$\begin{aligned} g(x) &= m_1(x)m_3(x) \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) \\ &= x^8 + x^7 + x^6 + x^4 + 1 \end{aligned}$$

Similarly, a binary triple-error correcting BCH code of the same length is obtained by choosing the generator polynomial

$$\begin{aligned} g(x) &= m_1(x)m_3(x)m_5(x) \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1) \\ &= x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1 \end{aligned}$$

The main interest in BCH codes is because they have a very fast and efficient decoding algorithm. We will describe this later.

AUTOMORPHISMS

Let C be a binary code of length n. Consider a permutation π of the set $\{0, 1, \ldots, n - 1\}$; that is, π is a one-to-one function of the set of coordinate positions onto itself.

For a codeword $\mathbf{c} \in C$, let

$$\pi(\mathbf{c}) = (c_{\pi(0)}, c_{\pi(1)}, \ldots, c_{\pi(n-1)})$$

That is, the coordinates are permuted by the permutation π. If

$$\{\pi(\mathbf{c}) | \mathbf{c} \in C\} = C$$

then π is called an *automorphism* of the code C.

Example. Consider the following (nonlinear code):

$$C = \{101, 011\}$$

The actions of the six possible permutations on three elements are given in the following table. The permutations, which are automorphisms, are marked by a star.

$\pi(0)$	$\pi(1)$	$\pi(2)$	$\pi((101))$	$\pi((011))$	
0	1	2	101	011	⋆
0	2	1	110	011	
1	0	2	011	101	⋆
1	2	0	011	110	
2	0	1	110	101	
2	1	0	101	110	

In general, the set of automorphisms of a code C is a group, the *Automorphism group* Aut(C). We note that

$$\sum_{i=0}^{n-1} x_i y_i = \sum_{i=0}^{n-1} x_{\pi(i)} y_{\pi(i)}$$

and so $(\mathbf{x}, \mathbf{y}) = 0$ if and only if $(\pi(\mathbf{x}), \pi(\mathbf{y})) = 0$. In particular, this implies that

$$\mathrm{Aut}(C) = \mathrm{Aut}(C^{\perp})$$

That is, C and $C^{\perp}$ have the same automorphism group.

For a *cyclic* code C of length n, we have by definition $\sigma(\mathbf{c}) \in C$ for all $\mathbf{c} \in C$, where $\sigma(i) \equiv i - 1 \pmod{n}$. In particular, $\sigma \in \mathrm{Aut}(C)$. For n odd, the permutation δ defined by $\delta(j) = 2j \pmod{n}$ also is contained in the automorphism group. To show this permutation, it is easier to show that $\delta^{-1} \in \mathrm{Aut}(C)$. We have

$$\begin{aligned} \delta^{-1}(2j) &= j & &\text{for } j = 0, 1, \ldots, (n-1)/2 \\ \delta^{-1}(2j+1) &= (n+1)/2 + j & &\text{for } j = 0, 1, \ldots, (n-1)/2 - 1 \end{aligned}$$

Let $g(x)$ be a generator polynomial for C, and let $\sum_{i=0}^{n-1} c_i x^i = a(x)g(x)$. Because $x^n \equiv 1 \pmod{x^n + 1}$, we have

$$\begin{aligned} \sum_{i=0}^{n-1} c_{\delta^{-1}(i)} x^i &= \sum_{j=0}^{(n-1)/2} c_j x^{2j} + \sum_{j=0}^{(n-1)/2} c_{(n+1)/2+j} x^{2j+1} \\ &= \sum_{j=0}^{(n-1)/2} c_j x^{2j} + \sum_{j=(n+1)/2}^{n-1} c_j x^{2j} \\ &= a(x^2)g(x^2) = (a(x^2)g(x))g(x), \quad (\mathrm{mod}\, x^n + 1) \end{aligned}$$

and so $\delta^{-1}(\mathbf{c}) \in C$; that is, $\delta^{-1} \in \mathrm{Aut}(C)$ and so $\delta \in \mathrm{Aut}(C)$.

The automorphism group Aut(C) is *transitive* if for each pair (i, j) a $\pi \in \mathrm{Aut}(C)$ exists such that $\pi(i) = j$. More general, Aut(C) is t-fold transitive if, for distinct $i_1, i_2, \ldots, i_t$ and distinct $j_1, j_2, \ldots, j_t$, a $\pi \in \mathrm{Aut}(C)$ exists such that $\pi(i_1) = j_1$, $\pi(i_2) = j_2, \ldots, \pi(i_t) = j_t$.

Example. Any cyclic $[n, k]$ code has a transitive automorphism group because σ repeated s times, where $s \equiv i - j \pmod{n}$, maps i to j.

Example. The (nonlinear) code $C = \{101, 011\}$ was considered previously. Its automorphism group is not transitive because there is no automorphism π such that $\pi(0) = 2$.

Example. Let C be the $[9, 3]$ code generated by the matrix

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

This is a cyclic code, and we will determine its automorphism group. The all zero and the all one vectors in C are transformed into themselves by any permutation. The vectors of weight 3 are the rows of the generator matrix, and the vectors of weight 6 are the complements of these vectors. Hence, we see that π is an automorphism if and only if it leaves the set of the three rows of the generator matrix invariant, that is, if and only if the following conditions are satisfied:

$$\begin{aligned} \pi(0) \equiv \pi(3) \equiv \pi(6) \pmod 3 \\ \pi(1) \equiv \pi(4) \equiv \pi(7) \pmod 3 \\ \pi(2) \equiv \pi(5) \equiv \pi(8) \pmod 3 \end{aligned}$$

Note that the two permutations σ and δ defined previously satisfy these conditions, as they should. They are listed explicitly in the following table:

i	0	1	2	3	4	5	6	7	8
$\sigma(i)$	8	0	1	2	3	4	5	6	7
$\delta(i)$	0	2	4	6	8	1	3	5	7

The automorphism group is transitive because the code is cyclic but not doubly transitive. For example, no automorphism π exists such that $\pi(0) = 0$ and $\pi(3) = 1$ because 0 and 1 are not equivalent modulo 3. A simple counting argument shows that Aut(C) has order 1296: First choose $\pi(0)$; this can be done in nine ways. Then two ways exist to choose $\pi(3)$ and $\pi(6)$. Next choose $\pi(1)$; this can be done in six ways. There are again two ways to choose $\pi(4)$ and $\pi(7)$. Finally, there are 3×2 ways to choose $\pi(2)$, $\pi(5)$, $\pi(8)$. Hence, the order is $9 \times 2 \times 6 \times 2 \times 3 \times 2 = 1296$.

Example. Consider the extended Hamming code H_m^{ext}. The positions of the codewords correspond to the elements of F_{2^m} and are permuted by the affine group

$$AG = \{\pi | \pi(x) = ax + b, a, b \in F_{2^m}, a \neq 0\}$$

This is the automorphism group of H_m^{ext}. It is double transitive.

THE WEIGHT DISTRIBUTION OF A CODE

Let C be a binary linear $[n, k]$ code. As we noted,

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{x} - \mathbf{y}, 0) = w(\mathbf{x} - \mathbf{y})$$

If $\mathbf{x}$, $\mathbf{y} \in C$, then $\mathbf{x} - \mathbf{y} \in C$ by the linearity of C. In particular, this means that the set of distances from a fixed codeword to all the other codewords is independent of which codeword we fix; that is, the code looks the same from any codeword. In particular, the set of distances from the codeword **0** is the set of *Hamming weights* of the codewords. For $i = 0, 1, \ldots, n$, let A_i denote the number of codewords of weight i. The sequence

$$A_0, A_1, A_2, \ldots, A_n$$

is called the *weight distribution* of the code C. The corresponding polynomial

$$A_C(z) = A_0 + A_1 z + A_2 z^2 + +A_n z^n$$

is known as the *weight enumerator polynomial* of C.

The polynomials $A_C(z)$ and $A_{C^\perp}(z)$ are related by the fundamental MacWilliams identity:

$$A_{C^\perp}(z) = 2^{-k}(1 + z)^n A_C\left(\frac{1 - z}{1 + z}\right)$$

Example. The $[2^m - 1, m]$ simplex code has the weight distribution polynomial $1 + (2^m - 1)z^{2^{m-1}}$. The dual code is the $[2^m - 1,\ 2^m - 1 - m]$ Hamming code with weight enumerator polynomial

$$\begin{aligned} &2^{-m}(1 + z)^{2^m - 1}\left(1 + (2^m - 1)\left(\frac{1 - z}{1 + z}\right)^{2^{m-1}}\right) \\ &\quad = 2^{-m}(1 + z)^{2^m - 1} + (1 - 2^{-m})(1 - z)^{2^{m-1}}(1 + z)^{2^{m-1} - 1} \end{aligned}$$

For example, for $m = 4$, we get the weight distribution of the [15, 11] Hamming code:

$$\begin{aligned} &1 + 35z^3 + 105z^4 + 168z^5 + 280z^6 + 435z^7 + 435z^8 + 280z^9 \\ &\quad + 168z^{10} + 105z^{11} + 35z^{12} + z^{15} \end{aligned}$$

Consider a binary linear code C that is used purely for error detection. Suppose a codeword $\mathbf{c}$ is transmitted over a binary symmetric channel with bit error probability p. The probability of receiving a vector $\mathbf{r}$ at distance i from $\mathbf{c}$ is $p^i(1 - p)^{n-i}$, because i positions are changed (each with probability p) and $n - i$ are unchanged (each with probability $1 - p$). If $\mathbf{r}$ is not a codeword, then this will be discovered by the receiver. If $\mathbf{r} = \mathbf{c}$, then no errors have occurred. However, if $\mathbf{r}$ is another codeword, then an undetectable error has occurred. Hence, the *probability of undetected error* is given by

$$\begin{aligned} P_{\text{ue}}(C, p) &= \sum_{\mathbf{c}\prime \neq \mathbf{c}} p^{d(\mathbf{c}\prime, \mathbf{c})}(1 - p)^{n - d(\mathbf{c}\prime, \mathbf{c})} \\ &= \sum_{\mathbf{c}' \neq 0} p^{w(\mathbf{c}')}(1 - p)^{n - w(\mathbf{c}')} \\ &= \sum_{i=1}^{n} A_i p^i (1 - p)^{n-i} \\ &= (1 - p)^n A_C\left(\frac{p}{1 - p}\right) - (1 - p)^n \end{aligned}$$

From the MacWilliams identity, we also get

$$P_{\text{ue}}(C^\perp, p) = 2^{-k} A_C(1 - 2p) - (1 - p)^n$$

Example. For the $[2^m - 1, 2^m - 1 - m]$ Hamming code H_m, we get

$$P_{\text{ue}}(H_m, p) = 2^{-m}(1 + (2^m - 1)(1 - 2p)^{2^{m-1}}) - (1 - p)^{2^{m-1}}$$

More information on the use of codes for error detection can be found in the books by Kløve and Korzhik (see Further Reading).

THE BINARY GOLAY CODE

The Golay code G_{23} has received much attention. It is practically useful and has a several interesting properties. The code can be defined in various ways. One definition is that G_{23} is the cyclic code generated by the irreducible polynomial

$$x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$$

which is a factor of $x^{23} + 1$ over F_2. Another definition is the following: Let H denote the [7, 4] Hamming code, and let H^* be the code whose codewords are the reversed of the codewords of H. Let

$$C = \{(\mathbf{u} + \mathbf{x}, \mathbf{v} + \mathbf{x}, \mathbf{u} + \mathbf{v} + \mathbf{x}) | \mathbf{u}, \mathbf{v} \in H^{\text{ext}}, \mathbf{x} \in (H^*)^{\text{ext}}\}$$

where H^{ext} is the [8, 4] extended Hamming code and $(H^*)^{\text{ext}}$ is the [8, 4] extended H^*. The code C is a [24, 12, 8] code. Puncturing the last position, we get a [23, 12, 7] code that is (equivalent to) the Golay code.

The weight distribution of G_{23} is given by the following table:

i	A_i
0, 23	1
7, 16	253
8, 15	506
11, 12	1288

The automorphism group Aut(G_{23}) of the Golay code is the Mathieu group M_{23}, a simple group of order $10200960 = 2^7 \times 3^2 \times 5 \times 7 \times 11 \times 23$, which is fourfold transitive.

Much information about G_{23} can be found in the book by MacWilliams and Sloane and in the *Handbook of Coding Theory* (see Further Reading).

DECODING

Suppose that a codeword $\mathbf{c}$ from the $[n, k]$ code C was sent and that an error $\mathbf{e}$ occurred during the transmission over the noisy channel. Based on the received vector $\mathbf{r} = \mathbf{c} + \mathbf{e}$, the receiver has to make an estimate of what was the transmitted codeword. Because error patterns of lower weight are more probable than error patterns of higher weight, the problem is to estimate an error $\hat{\mathbf{e}}$ such that the weight of $\hat{\mathbf{e}}$ is as small as possible. He will then decode the received vector $\mathbf{r}$ into $\hat{\mathbf{c}} = \mathbf{r} + \hat{\mathbf{e}}$.

If H is a parity-check matrix for C, then $\mathbf{c}H^{tr} = \mathbf{0}$ for all codewords $\mathbf{c}$. Hence,

$$\mathbf{r}H^{tr} = (\mathbf{c} + \mathbf{e})H^{tr} = \mathbf{c}H^{tr} + \mathbf{e}H^{tr} = \mathbf{e}H^{tr} \tag{3}$$

The vector

$$\mathbf{s} = \mathbf{e}H^{tr}$$

is known as the *syndrome* of the error $\mathbf{e}$; Equation (3) shows that $\mathbf{s}$ can be computed from $\mathbf{r}$. We now have the following outline of a decoding strategy:

1. Compute the syndrome $\mathbf{s} = \mathbf{r}H^{tr}$.
2. Estimate an error $\hat{\mathbf{e}}$ of smallest weight corresponding to the syndrome $\mathbf{s}$.
3. Decode to $\hat{\mathbf{c}} = \mathbf{r} + \hat{\mathbf{e}}$.

The hard part is, of course, step 2.

For any vector $\mathbf{x} \in F_2^n$, the set $\{\mathbf{x} + \mathbf{c} \mid \mathbf{c} \in C\}$ is a *coset* of C. All the elements of the coset have the same syndrome, namely, $\mathbf{x}H^{tr}$. There are 2^{n-k} cosets, one for each syndrome in F_2^{n-k}, and the set of cosets is a partition of F_2^n. We can rephrase step 2 as follows: Find a vector $\mathbf{e}$ of smallest weight in the coset with syndrome $\mathbf{s}$.

Example. Let C be the [6, 3, 3] code with parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

A *standard array* for C is the following array (the eight columns to the right):

000	000000	001011	010101	011110	100110	101101	110011	111000
110	100000	101011	110101	111110	000110	001101	010011	011000
101	010000	011011	000101	001110	110110	111101	100011	101000
011	001000	000011	011101	010110	101110	100101	111011	110000
100	000100	001111	010001	011010	100010	101001	110111	111100
010	000010	001001	010111	011100	100100	101111	110001	111010
001	000001	001010	010100	011111	100111	101100	110010	111001
111	100001	101010	110100	111111	000111	001100	010010	011001

Each row in the array is a listing of a coset of C; the first row is a listing of the code itself. The vectors in the first column have minimal weight in their cosets and are known as *coset leaders*. The choice of coset leader may not be unique. For example, in the last coset there are three vectors of minimal weight. Any entry in the array is the sum of the codeword at the top of the column and the coset leader (at the left in the row). Each vector of F_2^6 is listed exactly once in the array. The standard array can be used to decode: Locate $\mathbf{r}$ in the array and decode the codeword at the top of the corresponding column (that is, the coset leader is assumed to be the error pattern). However, this method is not practical; except for small n, the standard array of 2^n entries is too large to store (also locating $\mathbf{r}$ may be a problem). A step to simplify the method is to store a table of coset leaders corresponding to the 2^{n-k} syndromes. In the table above, this method is illustrated by listing the syndromes at the left. Again this alternative is possible only if $n - k$ is small. For carefully designed codes, it is possible to *compute* $\mathbf{e}$ from the syndrome. The simplest case is single errors: If $\mathbf{e}$ is an error pattern of weight 1, where the 1 is in the ith position, then the corresponding syndrome is in the ith column of H; hence, from H and the syndrome, we can determine i.

Example. Let H be the $m \times (2^m - 1)$ parity-check matrix where the ith column is the binary expansion of the integer i for $i = 1, 2, \ldots, 2^m - 1$. The corresponding $[2^m - 1, 2^m - 1 - m, 3]$ Hamming code corrects all single errors. Decoding is done as follows: Compute the syndrome $\mathbf{s} = (s_0, s_1, \ldots, s_{m-1})$. If $\mathbf{s} \neq \mathbf{0}$, then correct position $i = \sum_{j=0}^{m-1} s_j 2^j$.

Example. Let

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3(n-1)} \end{pmatrix}$$

where $\alpha \in F_{2^m}$ and $n = 2^m - 1$. This is the parity-check matrix for the double error-correcting BCH code. It is convenient to have a similar representation of the syndromes:

$$\mathbf{s} = (S_1, S_3) \quad \text{where} \quad S_1, S_3 \in F_{2^m}$$

Depending on the syndrome, there are several cases:

1. If no errors have occurred, then clearly $S_1 = S_3 = 0$.
2. If a single error has occurred in the ith position (that is, the position corresponding to α^i), then $S_1 = \alpha^i$ and $S_3 = \alpha^{3i}$. In particular, $S_3 = S^3{}_1$.
3. If two errors have occurred in positions i and j, then

$$S_1 = \alpha^i + \alpha^j, S_3 = \alpha^{3i} + \alpha^{3j}$$

This implies that $S^3{}_1 = S_3 + \alpha^i\alpha^j S_1 \neq S_3$. Furthermore, $x_1 = \alpha^{-i}$ and $x_2 = \alpha^{-j}$ are roots of the equation

$$1 + S_1 x + \frac{S_1^3 + S_3}{S_1} x^2 = 0. \qquad (4)$$

This gives the following procedure to correct two errors:

- Compute S_1 and S_3.
- If $S_1 = S_3 = 0$, then assume that no errors have occurred.
- Else, if $S^3 = S_1^3 \neq 0$, then one error has occurred in the ith position determined by $S_1 = \alpha^i$.
- Else (if $S_3 \neq S_1^3$), consider the equation

$$1 + S_1 x + (S_1^3 + S_3)/S_1 x^2 = 0$$

- If the equation has two roots α^{-i} and α^{-j}, then errors have occurred in positions i and j.

Else (if the equation has no roots in F_{2^m}), then more than two errors have occurred.

Similar explicit expressions (in terms of the syndrome) for the coefficients of an equation with the error positions as roots can be found for t error-correcting BCH codes when $t = 3$, $t = 4$, etc., but they become increasingly complicated. However, an efficient algorithm to determine the equation exists, and we describe this is some detail next.

Let α be a primitive element in F_{2^m}. A parity-check matrix for the primitive t-error-correcting BCH code is

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t-1} & \alpha^{2(2t-1)} & \cdots & \alpha^{(2t-1)(n-1)} \end{pmatrix}$$

where $n = 2^m - 1$. Suppose errors have occurred in positions $i_1, i_2, \ldots, i_\tau$, where $\tau \leq t$. Let $X_j = \alpha^{i_j}$ for $j = 1, 2, \ldots, \tau$. The *error locator polynomial* $\Lambda(x)$ is defined by

$$\Lambda(x) = \prod_{j=1}^{\tau}(1 + X_j x) = \sum_{l=0}^{\tau} \lambda_l x^l$$

The roots of $\Lambda(x) = 0$ are $X^{-1}{}_j$. Therefore, if we can determine $\Lambda(x)$, then we can determine the locations of the errors. Expanding the expression for $\Lambda(x)$, we get

$$\begin{aligned}
\lambda_0 &= 1 \\
\lambda_1 &= X_1 + X_2 + \cdots + X_\tau \\
\lambda_2 &= X_1X_2 + X_1X_3 + X_2X_3 + \cdots + X_{\tau-1}X_\tau \\
\lambda_3 &= X_1X_2X_3 + X_1X_2X_4 + X_2X_3X_4 + \cdots + X_{\tau-2}X_{\tau-1}X_\tau \\
&\vdots \\
\lambda_\tau &= X_1X_2 \cdots X_\tau \\
\lambda_l &= 0 \text{ for } l > \tau
\end{aligned}$$

Hence λ_l is the lth elementary symmetric function of $X_1, X_2, \ldots, X_\tau$.

From the syndrome, we get $S_1, S_3, \ldots, S_{2t-1}$, where

$$\begin{aligned}
S_1 &= X_1 + X_2 + \cdots + X_\tau \\
S_2 &= X_1^2 + X_2^2 + \cdots + X_\tau^2 \\
S_3 &= X_1^3 + X_2^3 + \cdots + X_\tau^3 \\
&\vdots \\
S_{2t} &= X_1^{2t} + X_2^{2t} + + X_\tau^{2t}
\end{aligned}$$

Furthermore,

$$S_{2r} = X_1^{2r} + X_2^{2r} + \cdots + X_r^{2r} = (X_1^r + X_2^r + \cdots + X_r^r)^2 = S_r^2$$

for all r. Hence, from the syndrome we can determine the polynomial

$$S(x) = 1 + S_1 x + S_2 x^2 + \cdots + S_{2t} x^{2t}$$

The *Newton equations* is a set of relations between the power sums S_r and the symmetric functions λ_l, namely

$$\sum_{j=0}^{l-1} S_{l-j}\lambda_j + l\lambda_l = 0 \text{ for } l \geq 1$$

Let

$$\Omega(x) = S(x)\Lambda(x) = \sum_{l \geq 0} \omega_l x^l \qquad (5)$$

Because $\omega_l = \sum_{j=0}^{l-1} S_{l-j}\lambda_j + \lambda_l$, the Newton equations imply that

$$\omega_l = 0 \text{ for all odd } l,\ 1 \le l \le 2t - 1 \tag{6}$$

The *Berlekamp–Massey algorithm* is an algorithm that, given $S(x)$, determines the polynomial $\Lambda(x)$ of smallest degree such that Equation (6) is satisfied, where the ω_l are defined by Equation (5). The idea is, for $r = 0, 1, \ldots, t$, to determine polynomials $\Lambda^{(r)}$ of lowest degree such that

$$\omega_l^{(r)} = 0 \text{ for all odd } l, 1 \le l \le 2r - 1$$

where

$$\sum_{l \ge 0} \omega_l^{(r)} x^l = S(x)\Lambda^{(r)}(x)$$

For $r = 0$, clearly we can let $\Lambda^{(0)}(x) = 1$. We proceed by induction. Let $0 \le r < t$, and suppose that polynomials $\Lambda^{(\rho)}(x)$ have been constructed for $0 \le \rho \le r$. If $\omega_{2r+1}^{(r)} = 0$, then we can choose

$$\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x)$$

If, on the other hand, $\omega_{2r+1}^{(r)} \neq 0$, then we modify $\Lambda^{(r)}(x)$ by adding another suitable polynomial. Two cases to consider are as follows: First, if $\Lambda^{(r)}(x) = 1$ [in which case $\Lambda^{(\tau)}(x) = 1$ for $0 \le \tau \le r$], then

$$\Lambda^{(r+1)}(x) = 1 + \omega_{2r+1}^{(r)} x^{2r+1}$$

will have the required property. If $\Lambda^{(r)}(x) \neq 1$, then a maximal positive integer $\rho < r$ such that $\omega_{2\rho+1}^{(\rho)} \neq 0$ exists and we add a suitable multiple of $\Lambda^{(\rho)}$:

$$\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x) + \omega_{2r+1}^{(r)} (\omega_{2\rho+1}^{(\rho)})^{-1} x^{2r-2\rho} \Lambda^{(\rho)}(x)$$

We note that this implies that

$$\Lambda^{(r+1)}(x)S(x) = \sum_{l \ge 0} \omega_l^{(r)} x^l + \omega_{2r+1}^{(r)} (\omega_{2\rho+1}^{(\rho)})^{-1} \sum_{l \ge 0} \omega_l^{(\rho)} x^{l+2r-2\rho}$$

Hence for odd l we get

$$\omega_l^{(r+1)} = \begin{cases} \omega_l^{(r)} = 0 & \text{for } 1 \le l \le 2r - 2\rho - 1 \\ \omega_l^{(r)} + \omega_{2r+1}^{(r)} (\omega_{2\rho+1}^{(\rho)})^{-1} \omega_{l-2r+2\rho}^{(\rho)} & \\ = 0 + 0 = 0 & \text{for } 2r - 2\rho + 1 \le l \le 2r - 1 \\ \omega_{2r+1}^{(r)} + \omega_{2r+1}^{(r)} (\omega_{2\rho+1}^{(\rho)})^{-1} \omega_{2\rho+1}^{(\rho)} & \\ = \omega_{2r+1}^{(r)} + \omega_{2r+1}^{(r)} = 0 & \text{for } l = 2r + 1 \end{cases}$$

We now formulate these ideas as an algorithm (in a Pascal-like syntax). In each step we keep the present $\Lambda(x)$ [the superscript (r) is dropped] and the modifying polynomial $[x^{2r-2\rho-1}$ or $(\omega_{2\rho+1}^{(\rho)})^{-1} x^{2r-2\rho-1} \Lambda^{(\rho)}(x)]$, which we denote by $B(x)$.

Berlekamp–Massey Algorithm in the Binary Case

```
Input: $t$ and $S(x)$.
$\Lambda(x) := 1$;   $B(x) := 1$;
for $r := 1$ to $t$ do
begin
     $\omega$ := coefficient of $x^{2r-1}$ in $S(x)\Lambda(x)$;
     if $\omega = 0$ then $B(x) := x^2 B(x)$
       else
          $[\Lambda(x), B(x)] := [\Lambda(x) + \omega x B(x),\ \ x\Lambda(x)/\omega]$ :
end;
```

The assignment following the **else** is two assignments to be done in parallel; the new $\Lambda(x)$ and $B(x)$ are computed from the old ones.

The Berlekamp–Massey algorithm determines the polynomial $\Lambda(x)$. To find the roots of $\Lambda(x) = 0$, we try all possible elements of F_{2^m}. In practical applications, this can be efficiently implemented using shift registers (usually called the *Chien search*).

Example. We consider the [15, 7, 5] double-error correcting BCH code; that is, $m = 4$ and $t = 2$. As a primitive root, we choose α such that $\alpha^4 = \alpha + 1$. Suppose that we have received a vector with syndrome $(S_1, S_3) = (\alpha^4, \alpha^5)$. Since $S_3 \neq S^3{}_1$, at least two errors have occurred. Equation (4) becomes

$$1 + \alpha^4 x + \alpha^{10} x^2 = 0$$

which has the zeros α^{-3} and α^{-7}. We conclude that the received vector has two errors (namely, in positions 3 and 7).

Now consider the Berlekamp–Massey algorithm for the same example. First we compute $S_2 = S^2{}_1 = \alpha^8$ and $S_4 = S^2{}_2 = \alpha$. Hence

$$S(x) = 1 + \alpha^4 x + \alpha^8 x^2 + \alpha^5 x^3 + \alpha x^4.$$

The values of r, ω, $\Lambda(x)$, and $B(x)$ after each iteration of the for-loop in the Berlekamp–Massey algorithm are shown in the following table:

r	ω	$\Lambda(x)$	$B(x)$
		1	1
1	α^4	$1 + \alpha^4 x$	$\alpha^{11} x$
2	α^{14}	$1 + \alpha^4 x + \alpha^{10} x^2$	$\alpha x + \alpha^5 x^2$

Hence, $\Lambda(x) = 1 + \alpha^4 x + \alpha^{10} x^2$ (as before).

Now consider the same code with syndrome of received vector $(S_1, S_3) = (\alpha, \alpha^9)$. Because $S_3 \neq S_1^3$, at least two errors have occurred. We get

$$\Lambda(x) = 1 + \alpha x + x^2$$

However, the equation $1 + \alpha x + x^2 = 0$ does not have any roots in F_{2^4}. Hence, at least three errors have occurred, and the code cannot correct them.

REED–SOLOMON CODES

In the previous sections we have considered binary codes where the components of the codewords belong to the finite field $F_2 = \{0, 1\}$. In a similar way we can consider codes with components from any finite field F_q.

The *Singleton bound* states that for any $[n, k, d]$ code with components from F_q, we have

$$d \leq n + k - 1$$

A code for which $d = n - k + 1$ is called *maximum distance separable* (MDS). The only binary MDS codes are the trivial $[n, 1, n]$ repetition codes and $[n, n - 1, 2]$ even-weight codes. However, important nonbinary MDS codes exist (in particular, the Reed–Solomon codes, which we now will describe).

Reed–Solomon codes are t-error-correcting cyclic codes with symbols from a finite field F_q, even though they can be constructed in many different ways. They can be considered as the simplest generalization of BCH codes. Because the most important case for applications is $q = 2^m$, we consider this case here. Each symbol is an element in F_{2^m} and can be considered as an m-bit symbol.

One construction of a cyclic Reed–Solomon code is as follows: Let α be a primitive element of F_{2^m}, and let $x_i = \alpha^i$. Because $x_i \in F_{2^m}$ for all i, the minimal polynomial of α^i over F_{2^m} is just $x - x_i$. The generator polynomial of a (primitive) t-error-correcting Reed–Solomon code of length $2^m - 1$ has $2t$ consequtive powers of α as zeros:

$$\begin{aligned} g(x) &= \prod_{i=0}^{2t-1} (x - \alpha^{b+i}) \\ &= g_0 + g_1 x + \cdots + g_{2t-1} x^{2t-1} + x^{2t} \end{aligned}$$

The code has the following parameters:

Block length: $n = 2^m - 1$
Number of parity-check symbols: $n - k = 2t$
Minimum distance: $d = 2t + 1$

Thus, the Reed–Solomon codes satisfy the Singleton bound with equality $n - k = d + 1$. That is, they are MDS codes.

An alternative description of the Reed–Solomon code is

$$\{(f(x_1), f(x_2), \ldots, f(x_n))| \\ f \text{ polynomial of degree less than } k\}.$$

The weight distribution of the Reed–Solomon code is (for $i \geq d$)

$$A_i \binom{n}{i} \sum_{j=0}^{i-d} (-1)^j \binom{i}{j} \left(2^{m(i-d-j+1)} - 1\right)$$

The encoding of Reed–Solomon codes is similar to the encoding of binary cyclic codes. One decoding is similar to the decoding of binary BCH codes with one added complication. Using a generalization of the Berlekamp–Massey algorithm, we determine the polynomials $\Lambda(x)$ and $\Omega(x)$. From $\Lambda(x)$ we can determine the locations of the errors. In addition, we must determine the value of the errors (in the binary case, the values are always 1). The value of the error at location X_j easily can be determined using $\Omega(x)$ and $\Lambda(x)$; we omit further details.

An alternative decoding algorithm can sometimes decode errors of weight more than half the minimum distance. We sketch this algorithm, first giving the simplest version, which works if the errors have weight less than half the minimum distance; that is, we assume a codeword $c = (f(x_1), f(x_2), \ldots, f(x_n))$ was sent and $r = (r_1, r_2, \ldots, r_n)$ was received, and $w(r - c) < (n - k + 1)/2$. It is easy to show that if $Q(x,y)$ is a nonzero polynomial in two variables of the form

$$Q(x,y) = Q_0(x) + Q_1(x)y$$

where

$$\begin{aligned} &Q(x_i, r_i) = 0 \text{ for } i = 1, 2, \ldots, n \\ &Q_0(x) \text{ has degree at most } n - 1 - t \\ &Q_1(x) \text{ has degree at most } n - k - t \end{aligned}$$

then $Q_0(x) + Q_1(x) f(x) = 0$, and so $Q(x,y) = Q_1(x)(y - f(x))$. Moreover, such a polynomial $Q(x,y)$ does exist. An algorithm to find $Q(x,y)$ is as follows:

```
Input: $r = (r_1, r_2, \ldots r_n)$.
Solve the equations $\sum_{j=0}^{n-1} a_j x_i^j + \sum_{j=0}^{n-k-t} b_j r_i x_i^j = 0$,
  where $i = 1, 2, \ldots, n$, for the unknown $a_j$ and $b_j$;
$Q_0(x) := \sum_{j=0}^{n-1-t} a_j x^j$;
$Q_1(x) := \sum_{j=0}^{n-k-t} b_j x^j$;
```

We now recover *f(x)* from the relation

$$Q_0(x) + Q_1(x)\ f(x) = 0.$$

To correct (some) errors of weight more than half the minimum distance, the method above must be generalized. The idea is to find a nonzero polynomial *Q(x, y)* of the form

$$Q(x,y) = Q_0(x) + Q_1(x)y + Q_2(x)y^2 + \cdots + Q_l(x)y^l$$

where now, for some integer τ and s,

(x_i, r_i) for $i = 1, 2, \ldots, n$, are zeros of $Q(x,y)$ of multiplicity s
$Q_m(x)$ has degree at most $s(n - \tau) - 1 - m(k - 1)$
for $m = 0, 1, \ldots, l$

For such a polynomial one can show that if the weight of the error is at most τ, then $y - f(x)$ divides *Q(x,y)*. Therefore, we can find all codewords within distance τ from **r** by finding all factors of *Q(x,y)* of the form $y - h(x)$, where *h(x)* is a polynomial of degree less than k. In general, there may be more than one such polynomial *h(x)*, of course, but in some cases, it is unique even if τ is larger than half the minimum distance. This idea is the basis for the Guruswami–Sudan algorithm.

Guruswami–Sudan Algorithm.

```
Input: $r = (r_1, r_2, \ldots r_n)$, $\tau$, $s$.

Solve the $n\binom{s+1}{2}$ equations
  $\sum_{m=u}^{l} \sum_{j=v}^{s(n-\tau)-1-m(k-1)} \binom{m}{u}\binom{j}{v} a_{m,j} x_i^{m-u} r_i^{j-v} = 0$,
  where $i = 1, 2, \ldots, n$, and $0 \leq u + v < s$,
  for the unknown $a_{m,j}$;
for $m$:=0 to $l$ do
  $Q_m(x) := \sum_{j=0}^{s(n-\tau)-1-m(k-1)} a_{m,j} x^j$;
$Q(x,y) := \sum_{m=0}^{l} Q_m(x) y^m$;
for each polynomial $h(x)$ of degree less than $k$ do
  if $y - h(x)$ divides $Q(x,y)$, then output $h(x)$.
```

We remark that the last loop of the algorithm is formulated in this simple form to explain the idea. In actual implementations, this part can be made more efficient. If $\tau < \frac{n(2l-s+1)}{2(l+1)} - \frac{l(k-1)}{2s}$, then the polynomial $f(x)$ of the sent codeword is among the output. The Guruswami–Sudan algorithm is a recent invention. A textbook covering it is the book by Justesen and Høholdt given in Further Reading.

ALGEBRAIC GEOMETRY CODES

Algebraic geometry codes can be considered as generalizations of Reed–Solomon codes, but they offer a wider range of code parameters. The Reed–Solomon codes over F_q have maximum length $n = q - 1$ whereas algebraic geometry codes can be longer. One common method to construct algebraic geometry codes is to select an algebraic curve and a set of functions that are evaluated on the points on the curve. In this way, by selecting different curves and sets of functions, one obtains different algebraic geometry codes. To treat algebraic codes in full generality is outside the scope of this article. We only will give a small flavor of the basic techniques involved by restricting ourselves to considering the class of Hermitian codes, which is the most studied class of algebraic geometry codes. This class of codes can serve as a good illustration of the methods involved.

The codewords in a Hermitian code have symbols from an alphabet F_{q^2}. The Hermitian curve consists of all points $(x,y) \in F_{q^2}^2$ given by the following equation in two variables:

$$x^{q+1} - y^q - y = 0$$

It is a straightforward argument to show that the number of different points on the curve is $n = q^3$. To select the functions to evaluate over the points on the curve, one needs to define an order function ρ. The function ρ is the mapping from the set of polynomials in two variable over F_{q^2} to the set of integers such that

$$\rho(x^i y^j) = iq + j(q+1)$$

and $\rho(f(x,y))$ is the maximum over all nonzero terms in $f(x,y)$.

Let $P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_n = (x_n, y_n)$ denote all the points on the Hermitian curve. The Hermitian code C_s is defined by

$$C_s = \{(f(P_1), f(P_2), \ldots, f(P_n)) | \rho(f(x,y)) \leq s \\ \text{and } \deg_x(f(x,y)) \leq q\}$$

where $\deg_x(f(x,y))$ is the maximum degree of x in $f(x,y)$.

Using methods from algebraic geometry, one obtains the following parameters of the Hermitian codes. The length of the code is $n = q^3$, and its dimension is

$$k = \begin{cases} \mu_s & \text{if } 0 \leq s \leq q^2 - q - 2 \\ s + 1 - (q^2 - q)/2 & \text{if } q^2 - q - 2 < s < n - q^2 + q \end{cases}$$

where μ_s is the number of monomials with $\rho(x^i y^j) \leq s$ and $i \leq q$, and the minimum distance is

$$d \geq n - s \text{ if } q^2 - q - 2 < s < n - q^2 + q$$

It is known that the class of algebraic geometry codes contains some very good codes with efficient decoding algorithms. It is possible to modify the Guruswami–Sudan algorithm to decode these codes.

NONLINEAR CODES FROM CODES OVER Z_4

In the previous sections mainly we have considered binary *linear* codes; that is, codes where the sum of two codewords is again a codeword. The main reason has been that the linearity greatly simplified construction and decoding of the codes.

A binary nonlinear (n, M, d) code C is simply a set of M binary n-tuples with pairwise distance at least d, but without any further imposed structure. In general, to find the minimum distance of a nonlinear code, one must compute the distance between all pairs of codewords. This is, of course, more complicated than for linear codes, where it suffices to find the minimum weight among all the nonzero codewords. The lack of structure in a nonlinear code also makes it difficult to decode in an efficient manner.

However, some advantages to nonlinear codes exist. For given values of length n and minimum distance d, sometimes it is possible to construct nonlinear codes with more codewords than is possible for linear codes. For example, for $n = 16$ and $d = 6$, the best linear code has dimension $k = 7$ (i.e., it contains 128 codewords). The code of length 16 obtained by extending the double-error-correcting primitive BCH code has these parameters.

In 1967, Nordstrom and Robinson (3) found a nonlinear code with parameters $n = 16$ and $d = 6$ containing $M = 256$ codewords, which has twice as many codewords as the best linear code for the same values of n and d.

In 1968, Preparata (4) generalized this construction to an infinite family of codes having parameters

$$(2^{m+1}, 2^{2^{m+1}-2m-2}, 6), m \text{ odd}, m \geq 3$$

A few years later, in 1972, Kerdock (5) gave another generalization of the Nordstrom–Robinson code and constructed another infinite class of codes with parameters

$$(2^{m+1}, 2^{2m+2}, 2^m - 2^{\frac{m-1}{2}}), \quad m \text{ odd}, m \geq 3$$

The Preparata code contains twice as many codewords as the extended double-error-correcting BCH code and is optimal in the sense of having the largest possible size for the given length and minimum distance. The Kerdock code has twice as many codewords as the best known linear code. In the case $m = 3$, the Preparata code and the Kerdock codes both coincide with the Nordstrom–Robinson code.

The Preparata and Kerdock codes are *distance invariant*, which means that the distance distribution from a given codeword to all the other codewords is independent of the given codeword. In particular, because they contain the all-zero codeword, their weight distribution equals their distance distribution.

In general, no natural way to define the dual code of a nonlinear code exists, and thus the MacWilliams identities have no meaning for nonlinear codes. However, one can define the weight enumerator polynomial $A(z)$ of a nonlinear code in the same way as for linear codes and compute its *formal dual* $B(z)$ from the MacWilliams identities:

$$B(z) = \frac{1}{M}(1+z)^n A\left(\frac{1-z}{1+z}\right)$$

The polynomial $B(z)$ obtained in this way has no simple interpretation. In particular, it may have coefficients that are nonintegers or even negative. For example, if $C = \{(110), (101), (111)\}$, then $A(z) = 2z^2 + z^3$ and $B(z) = (3 - 5z + z^2 + z^3)/3$.

An observation that puzzled the coding theory community for a long time was that the weight enumerator of the Preparata code $A(z)$ and the weight enumerator of the Kerdock code $B(z)$ satisfied the MacWilliams identities, and in this sense, these *nonlinear* codes behaved like dual linear codes.

Hammons et al. (6) gave a significantly simpler description of the family of Kerdock codes. They constructed a linear code over $Z_4 = \{0, 1, 2, 3\}$, which is an analog of the binary first-order Reed–Muller code. This code is combined with a mapping called the Gray map that maps the elements in Z_4 into binary pairs. The Gray map ϕ is defined by

$$\phi(0) = 00,\ \phi(1) = 01,\ \phi(2) = 11,\ \phi(3) = 10$$

The Lee weight of an element in Z_4 is defined by

$$w_L(0) = 0,\ w_L(1) = 1,\ w_L(2) = 2,\ w_L(3) = 1$$

Extending ϕ in a natural way to a map $\phi : Z_4^n \rightarrow Z_2^{2n}$ one observes that ϕ is a distance preserving map from Z_4^n (under the Lee metric) to Z_2^{2n} (under the Hamming metric).

A linear code over Z_4 is a subset of Z_4^n such that any linear combination of two codewords is again a codeword. From a linear code C of length n over Z_4, one obtains usually a binary nonlinear code $C = \phi(C)$ of length $2n$ by replacing each component in a codeword in C by its image under the Gray map. This code usually is nonlinear.

The minimum Hamming distance of C equals the minimum Lee distance of C and is equal to the minimum Lee weight of C because C is linear over Z_4.

Example. To obtain the Nordstrom–Robinson code, we will construct a code over Z_4 of length 8 and then apply the Gray map. Let $f(x) = x^3 + 2x^2 + x + 3 \in Z_4[x]$. Let β be a zero of $f(x)$; that is, $\beta^3 + 2\beta^2 + \beta + 3 = 0$. Then we can express all powers of β in terms of 1, β, and β^2, as follows:

$$\begin{aligned}
\beta^3 &= 2\beta^2 + 3\beta + 1 \\
\beta^4 &= 3\beta^2 + 3\beta + 2 \\
\beta^5 &= \beta^2 + 3\beta + 3 \\
\beta^6 &= \beta^2 + 2\beta + 1 \\
\beta^7 &= 1
\end{aligned}$$

Consider the code C over Z_4 with generator matrix given by

$$\begin{aligned}
G &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \beta & \beta^2 & \beta^3 & \beta^4 & \beta^5 & \beta^6 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 2 & 3 & 1 \\ 0 & 0 & 1 & 0 & 3 & 3 & 3 & 2 \\ 0 & 0 & 0 & 1 & 2 & 3 & 1 & 1 \end{bmatrix}
\end{aligned}$$

where the column corresponding to β^i is replaced by the coefficients in its expression in terms of 1, β, and β^2. Then the Nordstrom–Robinson code is the Gray map of C.

The dual code $\mathcal{C}^\perp$ of a code $\mathcal{C}$ over Z_4 is defined similarly as for binary linear codes, except that the inner product of the vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ with components in Z_4 is defined by

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} x_i y_i \pmod 4$$

The dual code $\mathcal{C}^\perp$ of $\mathcal{C}$ is then

$$\mathcal{C}^\perp = \{\mathbf{x} \in Z_4^n | (\mathbf{x}, \mathbf{c}) = 0 \text{ for all } \mathbf{c} \in Z_4^n\}$$

For a linear code C over Z_4, a MacWilliams relation determines the complete weight distribution of the dual code $\mathcal{C}^\perp$ from the complete weight distribution of $\mathcal{C}$. Therefore, one can compute the relation between the Hamming weight distributions of the nonlinear codes $C = \phi(\mathcal{C})$ and $C_\perp = \phi(\mathcal{C}^\perp)$, and it turns out that the MacWilliams identities hold.

Hence, to find nonlinear binary codes related by the MacWilliams identities, one can start with a pair of Z_4-linear dual codes and apply the Gray map. For any odd integer $m \geq 3$, the Gray map of the code K_m over Z_4 with generator matrix

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 0 & 1 & \beta & \beta^2 & \ldots & \beta^{2^m-2} \end{bmatrix}$$

is the binary, nonlinear $\left(2^{m+1}, 2^{2m+2}, 2^m - 2^{\frac{m-1}{2}}\right)$ Kerdock code. The Gray map of $K_m^{\perp}$ has the same weight distribution as the $(2^{m+1}, 2^{2^{m+1}-2m-2}, 6)$ Preparata code. It is not identical, however, to the Preparata code and is therefore denoted the "Preparata" code. Hence the Kerdock code and the "Preparata" code are the Z_4-analogy of the first-order Reed–Muller code and the extended Hamming code, respectively.

Hammons et al. (6) also showed that the binary code defined by $C = \phi(C)$, where C is the quaternary code with parity-check matrix given by

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & \beta & \beta^2 & \dots & \beta^{2^m-2} \\ 0 & 2 & 2\beta^3 & 2\beta^6 & \dots & 2\beta^{3(2^m-2)} \end{bmatrix}$$

is a binary nonlinear $(2^{m+1}, 2^{2^{m+1}-3m-2}, 8)$ code whenever $m \geq 3$ is odd. This code has the same weight distribution as the Goethals code, which is a nonlinear code that has four times as many codewords as the comparable linear extended triple-error-correcting primitive BCH code. The code $C_\perp = \phi(C^\perp)$ is identical to a binary nonlinear code, which was constructed in a much more complicated way by Delsarte and Goethals (7–9) more than 30 years ago.

To analyze codes obtained from codes over Z_4 in this manner, one is led to study Galois rings instead of Galois fields. Similar to a Galois field, a Galois ring can be defined as $Z_{p^e}[x]/(f(x))$, where $f(x)$ is a monic polynomial of degree m that is irreducible modulo p. The richness in structure of the Galois rings led to several recently discovered good nonlinear codes that have an efficient and fast decoding algorithm.

BIBLIOGRAPHY

1. R. C. Bose and D. K. Ray-Chaudhuri, On a class of error correcting binary group codes, *Inform. Contr.*, **3**: 68–79, 1960.
2. A. Hocquenghem, Codes correcteur d'erreurs, *Chiffres*, **2**: 147–156, 1959.
3. A. W. Nordstrom and J. P. Robinson, An optimum nonlinear code, *Inform. Contr.*, **11**: 613–616, 1967.
4. F. P. Preparata, A class of optimum nonlinear double-error correcting codes, *Inform. Contr.*, **13**: 378–400, 1968.
5. A. M. Kerdock, A class of low-rate nonlinear binary codes, *Inform. Contr.*, **20**: 182–187, 1972.
6. A. R. Hammons, P. V. Kumar, A. R. Calderbank, N. J. A. Sloane, and P. Solé, The Z_4-linearity of Kerdock, Preparata, Goethals, and related codes, *IEEE Trans. Inform. Theory*, **40**: 301–319, 1994.
7. P. Delsarte and J. M. Goethals, Alternating bilinear forms over GF(q), *J. Combin. Theory, Series A*, **19**: 26–50, 1975.
8. J. M. Goethals, Two dual families of nonlinear binary codes, *Electronic Letters*, **10**: 471–472, 1974.
9. J. M. Goethals, Nonlinear codes defined by quadratic forms over GF(2), *Inform. Contr.*, **31**: 43–74, 1976.

FURTHER READING

R. Blahut, *The Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley, 1983.

R. Blahut, *Algebraic Codes for Data Transmission*. Cambridge, MA: Cambridge Univ. Press, 2003.

R. Hill, *A First Course in Coding Theory*. Oxford: Clarendon Press, 1986.

J. Justesen and T. Høholdt, *A Course in Error-Correcting Codes*. European Mathematical Society Publ. House, 2004.

T. Kløve, *Codes for Error Detection*. Singapore: World Scientific, 2007.

T. Kløve and V. I. Korzhik, *Error-Detecting Codes*. Boston, MA: Kluwer Academic, 1995.

R. Lidl and H. Niederreiter, *Finite Fields*, vol. 20 of *Encyclopedia of Mathematics and Its Applications*. Reading, MA: Addison-Wesley, 1983.

S. Lin and D. J. Costello, Jr., *Error Control Coding*. 2nd edition. Englewood Cliffs, NJ: Prentice Hall, 2004.

J. H. vanLint, *Introduction to Coding Theory*. New York, NY: Springer-Verlag, 1982.

F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam: North-Holland, 1977.

W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. Cambridge, MA: The MIT Press, 1972.

V. S. Pless and W. C. Huffman (eds.), *Handbook of Coding Theory*, Vol. I & II. Amsterdam: Elsevier, 1998.

M. Purser, *Introduction to Error-Correcting Codes*. Boston, MA: Artech House, 1995.

H. vanTilborg, *Error-Correcting Codes—A First Course*. Lund: Studentlitteratur, 1993.

S. A. Vanstone and P. C. van Oorschot, *An Introduction to Error-Correcting Codes with Applications*. Boston, MA: Kluwer Academic, 1989.

TOR HELLESETH
TORLEIV KLØVE
University of Bergen
Bergen, Norway

BIOINFORMATIC DATABASES

At some time during the course of any bioinformatics project, a researcher must go to a database that houses biological data. Whether it is a local database that records internal data from that laboratory's experiments or a public database accessed through the Internet, such as NCBI's GenBank (1) or EBI's EMBL (2), researchers use biological databases for multiple reasons.

One of the founding reasons for the fields of bioinformatics and computational biology was the need for management of biological data. In the past several decades, biological disciplines, including molecular biology and biochemistry, have generated massive amounts of data that are difficult to organize for efficient search, query, and analysis. If we trace the histories of both database development and the development of biochemical databases, we see that the biochemical community was quick to embrace databases. For example, E. F. Codd's seminal paper, "A Relational Model of Data for Large Shared Data Banks" (3), published in 1970 is heralded as the beginning of the relational database, whereas the first version of the Protein Data Bank (PDB) was established at Brookhaven National Laboratories in 1972 (4).

Since then, especially after the launching of the human genome sequencing project in 1990, biological databases have proliferated, most embracing the World Wide Web technologies that became available in the 1990s. Now there are hundreds of biological databases, with significant research efforts in both the biological as well as the database communities for managing these data. There are conferences and publications solely dedicated to the topic. For example, Oxford University Press dedicates the first issue of its journal *Nucleic Acids Research* (which is freely available) every year specifically to biological databases. The database issue is supplemented by an online collection of databases that listed 858 databases in 14 categories in 2006 (5), including both new and updated ones.

Biological database research now encompasses many topics, such as biological data management, curation, quality, integration, and mining (6). Biological databases can be classified in many different ways, from the topic they cover, to how heavily annotated they are or which annotation method they employ, to how highly integrated the database is with other databases. Popularly, the first two categories of classification are used most frequently. For example, there are archival nucleic acid data repositories [GenBank, the EMBL Data Library, and the DNA Databank of Japan (7)] as well as protein sequence motif/domain databases, like PROSITE (8), that are derived from primary source data.

Modern biological databases comprise not only data, but also sophisticated query facilities and bioinformatic data analysis tools; hence, the term "bioinformatic databases" is often used. This article presents information on some popular bioinformatic databases available online, including sequence, phylogenetic, structure and pathway, and microarray databases. It highlights features of these databases, discussing their unique characteristics, and focusing on types of data stored and query facilities available in the databases. The article concludes by summarizing important research and development challenges for these databases, namely knowledge discovery, large-scale knowledge integration, and data provenance problems. For further information about these databases and access to all hyperlinks presented in this article, please visit http://www.csam.montclair.edu/~herbert/bioDatabases.html.

SEQUENCE DATABASES

Genome and protein sequence databases represent the most widely used and some of the best established biological databases. These databases serve as repositories for wetlab results and the primary source for experimental results. Table 1 summarizes these data repositories and gives their respective URLs.

GenBank, EMBL, and the DNA Databank of Japan

The most widely used biological data bank resource on the World Wide Web is the genomic information stored in the U.S.'s National Institutes of Health's GenBank, the European Bioinformatics Institutes' EMBL, and Japan's National Institute of Genetics DNA Databank of Japan (1, 2, 7). Each of these three databases was developed separately, with GenBank and EMBL launching in 1980 (4). Their collaboration started soon after their development, and DDBJ joined the collaboration shortly after its creation in 1986. The three databases, under the direction of the International Nucleotide Sequence Database Collaboration (INSDC), gather, maintain, and share mainly nucleotide data, each catering to the needs of the region in which it is located (4).

The Ensembl Genome Database

The Ensembl database is a repository of stable, automatically annotated human genome sequences. It is available either as an interactive website or downloadable as flat files. Ensembl annotates and predicts new genes, with annotation from the InterPro (9) protein family databases and with additional annotations from databases of genetic disease [OMIM (10)], expression [SAGE (11,12)] and gene family (13). As Ensembl endeavors to be both portable and freely available, software available at Ensembl is based on relational database models (14).

GeneDB Database

GeneDB (15) is a genome database for prokaryotic and eukaryotic organisms. It currently contains data for 37 genomes generated from the Pathogen Sequencing Unit (PSU) at the Welcome Trust Sanger Institute. The GeneDB

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Table 1. Summary of Genome and Protein Sequence Databases

Database	URL	Feature
GenBank	http://www.ncbi.nlm.nih.gov/	NIH's archival genetic sequence database
EMBL	http://www.ebi.ac.uk/embl/	EBI's archival genetic sequence database
DDBJ	http://www.ddbj.nig.ac.jp/	NIG's archival genetic sequence database
Ensembl	http://www.ensembl.org/	Database that maintains automatic annotation on selected eukaryotic genomes
GeneDB	http://www.cebitec.uni-bielefeld.de/groups/brf/software/gendb_info/	Database that maintains genomic information about specific species related to pathogens
TAIR	http://www.arabidopsis.org/	Database that maintains genomic information about *Arabidopsis thaliana*
SGD	http://www.yeastgenome.org/	A repository for baker's yeast genome and biological data
dbEST	http://www.ncbi.nlm.nih.gov/dbEST/	Division of GenBank that contains expression tag sequence data
Protein Information Resource (PIR)	http://pir.georgetown.edu/	Repository for nonredundant protein sequences and functional information
Swiss-Prot/TrEMBL	http://www.expasy.org/sprot/	Repository for nonredundant protein sequences and functional information
UniProt	http://www.pir.uniprot.org/	Central repository for PIR, Swiss-Prot, and TrEMBL

database has four key functionalities. First, the database stores and frequently updates sequences and annotations. Second, GeneDB provides a user interface, which can be used for access, visualization, searching, and downloading of the data. Third, the database architecture allows integration of different biological datasets with the sequences. Finally, GeneDB facilitates querying and comparisons between species by using structured vocabularies (15).

The Arabidopsis Information Resource (TAIR)

TAIR (16) is a comprehensive genome database that allows for information retrieval and data analysis pertaining to *Arabidopsis thaliana* (a small annual plant belonging to the mustard family). *Arabidopsis thaliana* has been of great interest to the biological community and is one of the few plants whose genome is completely sequenced (16). Due to the complexity of many plant genomes, *Arabidopsis thaliana* serves as a model for plant genome investigations. The database has been designed to be simple, portable, and efficient. One innovate aspect of the TAIR website is MapViewer (http://www.arabidopsis.org/servlets/mapper). MapViewer is an integrated visualization tool for viewing genetic, physical, and sequence maps for each *Arabidopsis* chromosome. Each component of the map contains a hyperlink to an output page from the database that displays all the information related to this component (16).

SGD: Saccharomyces Genome Database

The Saccharomyces Genome Database (SGD) (17) provides information for the complete *Saccharomyces cerevisiae* (baker's and brewer's yeast) genomic sequence, along with its genes, gene products, and related literature. The database contains several types of data, including DNA sequence, gene-encoded proteins, and the structures and biological functions of any known gene products. It also allows full-text searches of articles concerning *Saccharomyces cerevisiae*. The SGD database is not a primary sequence repository (17), but a collection of DNA and protein sequences from existing databases [GenBank (1), EMBL (2), DDBJ (7), PIR (18), and Swiss-Prot (19)]. It organizes the sequences into datasets to make the data more useful and easily accessible.

dbEST Database

dbEST (20) is a division of GenBank that contains sequence data and other information on short, "single-pass" cDNA sequences, or Expressed Sequence Tags (ESTs), generated from randomly selected library clones (http://www.ncbi.nlm.nih.gov/dbEST/). dbEST contains approximately 36,843,572 entries from a broad spectrum of organisms. Access to dbEST can be obtained through the Web, either from NCBI by anonymous ftp or through Entrez (21). The dbEST nucleotide sequences can be searched using the BLAST sequence search program at the NCBI website. In addition, TBLASTN, a program that takes a query amino acid sequence and compares it with six-frame translations of dbEST DNA sequences can also be useful for finding novel coding sequences. EST sequences are available in the FASTA format from the "/repository/dbEST" directory at ftp.ncbi.nih.gov.

The Protein Information Resource

The Protein Information Resource (PIR) is an integrated public bioinformatics resource that supports genomic and proteomic research and scientific studies. PIR has provided many protein databases and analysis tools to the scientific community, including the PIR-International Protein Sequence Database (PSD) of functionally annotated protein sequences. The PIR-PSD, originally created as the Atlas of Protein Sequence and Structure edited by Margaret Dayhoff, contained protein sequences that were highly annotated with functional, structural, bibliographic, and sequence data (5,18). PIR-PSD is now merged with UniProt Consortium databases (22). PIR offers the

PIRSF protein classification system (23) that classifies proteins, based on full-length sequence similarities and their domain architectures, to reflect their evolutionary relationships. PIR also provides the iProClass database that integrates over 90 databases to create value-added views for protein data (24). In addition, PIR supports a literature mining resource, iProLINK (25), which provides multiple annotated literature datasets to facilitate text mining research in the areas of literature-based database curation, named entity recognition, and protein ontology development.

The Swiss-Prot Database

Swiss-Prot (19) is a protein sequence and knowledge database and serves as a hub for biomolecular information archived in 66 databases (2). It is well known for its minimal redundancy, high quality of annotation, use of standardized nomenclature, and links to specialized databases. Its format is very similar to that of the EMBL Nucleotide Sequence Database (2). As Swiss-Prot is a protein sequence database, its repository contains the amino acid sequence, the protein name and description, taxonomic data, and citation information. If additional information is provided with the data, such as protein structures, diseases associated with the protein or splice isoforms, Swiss-Prot provides a table where these data can be stored. Swiss-Prot also combines all information retrieved from the publications reporting new sequence data, review articles, and comments from enlisted external experts.

TrEMBL: A Supplement to Swiss-Prot

Due to the large number of sequences generated by different genome projects, the Swiss-Prot database faces several challenges related to the processing time required for manual annotation. For this reason, the European Bioinformatics Institute, collaborating with Swiss-Prot, introduced another database, TrEMBL (translation of EMBL nucleotide sequence database). This database consists of computer-annotated entries derived from the translation of all coding sequences in the nucleotide databases. This database is divided into two sections: SP-TrEMBL contains sequences that will eventually be transferred to Swiss-Prot and REM-TrEMBL contains those that will not go into Swiss-Prot, including patent application sequences, fragments of less than eight amino acids, and sequences that have proven not to code for a real protein (19, 26, 27).

UniProt

With protein information spread over multiple data repositories, the efforts from PIR, SIB's Swiss-Prot and EBI's TrEMBL were combined to develop the UniProt Consortium Database to centralize protein resources (22). UniProt is organized into three layers. The UniProt Archive (UniParc) stores the stable, nonredundant, corpus of publicly available protein sequence data. The UniProt Knowledgebase (UniProtKB) consists of accurate protein sequences with functional annotation. Finally, the UniProt Reference Cluster (UniRef) datasets provide nonredundant reference clusters based primarily on UniProtKB. UniProt also offers users multiple tools, including searches against the individual contributing databases, BLAST and multiple sequence alignment, proteomic tools, and bibliographic searches (22).

PHYLOGENETIC DATABASES

With all of the knowledge accumulating in the genomic and proteomic databases, there is a great need for understanding how all these types of data relate to each other. As all biological things have come about through the evolutionary process, the patterns, functions, and processes that they possess are best analyzed in terms of their phylogenetic histories. The same gene can evolve a different timing of its expression, a different tissue where it is expressed, or even gain a whole new function along one phylogenetic branch as compared with another. These changes along a branch affect the biology of all descendant species, thereby leaving phylogenetic patterns in everything we see. A detailed mapping between biological data and phylogenetic histories must be accomplished so that the full potential of the data accumulation activities can be realized. Otherwise it will be impossible to understand why certain drugs work in some species but not others, or how we can design therapies against evolving disease agents such as HIV and influenza.

The need to query data using sets of evolutionary related taxa, rather than on single species, has brought up the need to create databases that can serve as repositories of phylogenetic trees, generated by a variety of methods. Phylogeny and phylogenetic trees give a picture of the evolutionary history among species, individuals, or genes. Therefore, there are at least two distinct goals of a phylogenetic database: archival storage and analysis (28). Table 2 summarizes these repositories.

Many of the aforementioned data repositories offer functionalities for browsing phylogenetic and taxonomic information. NCBI offers users the Taxonomy Databases (1, 13), which organize the data maintained in its repositories from the species perspective and allows the user to hierarchically browse data with respect to a Tree of Life organization. NEWT is a taxonomy database (http://www.ebi.ac.uk/newt/) that connects UniProtKB data to the NCBI taxonomy data. For every species, NEWT provides information about the taxon's scientific name, common name and synonym(s), lineage, number of UniProtKB protein entries in the given taxon, and links to each entry.

Tree of Life

The Tree of Life (29) is a phylogenetic repository that aims to provide users with information from a whole-species point of view. The Tree of Life allows users to search for pages about specific species through conventional keyword search mechanisms. Most interestingly, a user can also navigate through the "tree of life" using hierarchical browsing starting at the root organism, popularly referred to as "Life," and traverse the tree until a species of interest is reached. The species web page contains information gathered and edited by recognized experts about the species

Table 2. Summary of Phylogenetic Data Repositories

Database	URL	Feature
NCBI Taxonomy Database	http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Taxonomy	Whole-species view of genomic and proteomic data stored in GenBank
Tree of Life	http://tolweb.org/tree/	Species-centric hierarchical browsing database modeling the evolutionary relationships between species
TreeFam	http://www.treefam.org/	Repository for phylogenetic trees based on animal genomes
TreeBASE	http://www.treebase.org/treebase/	Archival peer-reviewed phylogenetic tree repository
SYSTERS	http://systers.molgen.mpg.de/	Protein cluster repository with significant phylogenetic functionalities
PANDIT	http://www.ebi.ac.uk/goldman-srv/pandit/	Protein domains repository with inferred phylogenetic trees

as well as peer-reviewed resources accessible through hyperlinks (29).

TreeFam

TreeFam is a database of phylogenetic trees of animal gene families. The goal of TreeFam is to develop a curated database that provides accurate information about ortholog and paralog assignments and evolutionary histories of various gene families (30). To create and curate the trees and families, TreeFam has gathered sequence data from several protein repositories. It contains protein sequences for human (*Homo sapiens*), mouse (*Mus musculus*), rat (*Rattus norvegicus*), chicken (*Gallus gallus*), pufferfish (*Takifugu rubripes*), zebrafish (*Danio rerio*), and fruitfly (*Drosophila melanogaster*), which were retrieved from Ensembl (14), WormBase (31), SGD (17), GeneDB (15), and TIGR (32). The protein sequences in TreeFam are grouped into families of genes that descended from a single gene in the last common ancestor of all animals, or that first appeared in animals. From the above sources, families and trees are automatically generated and then manually curated based on expert review. To manage these data, TreeFam is divided into two parts. TreeFAM-B consists of the automatically generated trees. It obtains clusters from the PhIGs (33) database and uses BLAST (34), MUSCLE (35), and HMMER (36) and neighbor-joining algorithms (37) to generate the trees. TreeFAM-A contains the manually curated trees, which exploit algorithms similar to the DLI algorithm (DLI: H. Li, unpublished data) and the SDI algorithm (38). TreeFAM contains 11,646 families including about 690 families that have curated phylogenetic trees. Therefore, as more trees get curated, the TreeFam-A database increases, whereas TreeFam-B decreases in size.

TreeBASE

TreeBASE (39) was developed to help harness the explosively high growth in the number of published phylogenetic trees. It is a relational database and contains phylogenetic trees and the data underlying those trees. TreeBASE is available at http://www.treebase.org and allows the user to search the database according to different keywords and to see graphical representations of the trees. The user can also access information such as data matrices, bibliographic information, taxonomic names, character states, algorithms used, and analyses performed. Phylogenetic trees are submitted to TreeBASE by the authors of the papers that describe the trees. For data to be accepted by TreeBASE, the corresponding paper must pass the journal's peer review process (39).

SYSTERS Database

SYSTERS is a protein clustering database based on sequence similarity (40). It can be accessed at http://SYSTERS.molgen.mpg.de/. SYSTERS contains 185,000 disjoint protein families gathered from existing sequence repositories: Swiss-Prot (19), TrEMBL (19) and complete genomes: Ensembl (14), The Arabidopsis Information Resource (16), SGD (17), and GeneDB (15). Two innovative features of this repository are the SYSTERS Table and SYSTERS Tree. The SYSTERS Table for a family cluster contains a variety of information, most notably accession numbers as well as accession numbers for a variety of external databases [IMB (41), MSD (42), ENZYME (43), INTERPRO (9), PROSITE (8), GO (44)]. There can be several redundant entries in the table for one protein sequence. As SYSTERS data rely on external protein databases, there is always an entry name (protein name) and an accession number for each entry but there may not be a gene name. For each family cluster that consists of more than two nonredundant entries, a phylogenetic tree is available. The phylogenetic trees are constructed using the UPGMA (45) method. No more than 200 entries are displayed in a tree; the selection process when a cluster contains more than 200 entries is not clear.

PANDIT (Protein and Associated Nucleotide Domains with Inferred Trees)

PANDIT is a nonredundant repository of multiple sequence alignments and phylogenetic trees. It is available at http://www.ebi.ac.uk/goldman-srv/pandit. The database consists of three portions: protein domain sequence alignments from Pfam Database (46), alignments of nucleotide sequences derived from EMBL Nucleotide Sequence Database (2), and phylogenetic trees inferred from each alignment. Currently PANDIT contains 7738 families of homologous protein sequences with corresponding DNA sequences and phylogenetic trees. All alignments are based on Pfam-A (47) seed alignments, which are manually curated and, therefore, make PANDIT data high quality and comparable with alignments used to study evolution. Each family contains three alignments: PANDIT-aa contains the exact Pfam-A seed protein sequence alignment; and PANDIT-dna contains the DNA sequences encoding the protein sequences in PANDIT-aa that could be recovered; and PANDIT-aa-restricted contains only those protein sequences for which a DNA sequence has been recovered. The DNA sequences have been retrieved using cross-references to the EMBL Nucleotide Sequence Database from the Swiss-Prot (19) and TrEMBL (19) databases. To ensure accuracy, PANDIT performs a translation of the cross-referenced DNA sequences back to the corresponding protein sequences.

PANDIT database is intended for studying the molecular evolution of protein families. Therefore, phylogenetic trees have been constructed for families of more than two sequences. For each family, five different methods for tree estimation have been used to produce candidate trees. These methods include neighbor-joining (37), BioNJ (48), Weighbor (49), FastME (50), and PHYML (51). Neighbor-joining, BioNJ and Weighbor are methods used to produce phylogenetic tree estimates from a pairwise distance matrix. FastME uses a minimum evolution criterion with local tree-rearrangements to estimate a tree, and Phyml uses maximum likelihood with local tree searching. At the end, the likelihood of each tree from the candidate set is computed and the tree with the highest likelihood is added to the database.

STRUCTURE AND PATHWAY DATABASES

Knowledge of protein structures and of molecular interactions is key to understanding protein functions and complex regulatory mechanisms underlying many biological processes. However, computationally, these datasets are highly complex. The most popular ways to model these datasets are through text, graphs, or images. Text data tend not to have the descriptive power needed to fully model this type of data. Graphical and the image data require complex algorithms that are computationally expensive and not reliably accurate. Therefore, structural and pathway databases become an interesting niche from both the biological and the computational perspectives. Table 3 lists several prominent databases in this field.

The Protein Data Bank

The Protein Data Bank (PDB) is an archive of structural data of biological macromolecules. PDB is maintained by the Research Collaboratory for Structural Bioinformatics (RCSB). It allows the user to view data both in plain text and through a molecular viewer using Jmol. A key goal of the PDB is to make the data as uniform as possible while improving data accessibility and providing advanced querying options (52, 53).

To have complete information regarding the features of macromolecular structures, PDB allows a wide spectrum of queries through data integration. PDB collects and integrates external data from scientists' deposition, Gene Ontology (GO) (54), Enzyme Commission (55), KEGG Pathways (56), and NCBI resources (57). PDB realizes data integration through data loaders written in Java, which extract information from existing databases based on common identification numbers. PDB also allows data extraction at query run time, which means implemented Web services extract information as the query is executing.

The Nucleic Acid Database

Nucleic Acid Database, also curated by RCSB and similar to the PDB and the Cambridge Structural Database (58), is a repository for nucleic acid structures. It gives users access to tools for extracting information from nucleic acid structures and distributes data and software. The data are stored in a relational database that contains tables of primary and derivative information. The primary information includes atomic coordinates, bibliographic references, crystal data, data collection, and other structural descriptions. The derivative information is calculated from the primary information and includes chemical bond lengths, and angles, virtual bond lengths, and other measures according to various algorithms (59, 60). The experimental data in the NDB database have been collected from published literature, as well as from one of the standard crystallographic archive file types (60, 61) and other sources. Primary information has been encoded in ASCII format file (62). Several programs have been developed to convert between different file formats (60, 63, 64, 65).

The Kyoto Encyclopedia of Genes and Genomes

The Kyoto Encyclopedia of Genes and Genomes (KEGG) (56) is the primary resource for the Japanese GenomeNet service that attempts to define the relationships between

Table 3. Summary of Structural and Pathway Databases

Database	URL	Feature
The Protein Data Bank (PDB)	http://www.rcsb.org/pdb/	Protein structure repository that provides tools for analyzing these structures
The Nucleic Acid Database (NDB)	http://ndbserver.rutgers.edu/	Database housing nucleic acid structural information
The Kyoto Encyclopedia of Genes and Genomes (KEGG)	http://www.genome.jp/kegg/	Collection of databases integrating pathway, genomic, proteomic, and ligand data
The BioCyc Database Collection	http://www.biocyc.org/	Collection of over 200 pathway and genomic databases

the functional meanings and utilities of the cell or the organism and its genome information. KEGG contains three databases: PATHWAY, GENES, and LIGAND. The PATHWAY database stores computerized knowledge on molecular interaction networks. The GENES database contains data concerning sequences of genes and proteins generated by the genome projects. The LIGAND database holds information about the chemical compounds and chemical reactions that are relevant to cellular processes. KEGG computerizes the data and knowledge as graph information. The KEGG/PATHWAY database contains reference diagrams for molecular pathways and complexes involving various cellular processes, which can readily be integrated with genomic information (66). It stores data objects called generalized protein interaction networks (67, 68). The PATHWAY database is composed of four levels that can be accessed through the Web browser. The top two levels contain information about metabolism, genetic information processing, environmental information processing, cellular processes, and human diseases. The others relate to the pathway diagram and the ortholog group table, which is a collection of genes and proteins.

The BioCyc Database Collection

The BioCyc Database Collection (69) is a compilation of pathway and genome information for different organisms. Based on the number of reviews and updates, BioCyc databases are organized into several tiers. Tier 1 consists of three databases, EcoCyc (70), which describes *Escherichia coli* K-12; MetaCyc (71), which describes pathways for more than 300 organisms; and the BioCyc Open Compounds Database (69), which contains a collection of chemical compound data from BioCyc databases. Tier 2 contains 12 databases computationally generated by the Pathologic program. These databases have been updated and manually curated to varying degrees. Tier 3 is composed of 191 databases computationally generated by the Pathologic program with no review and updating (69).

The BioCyc website allows scientists to perform certain operations, e.g., to visualize individual metabolic pathways, to view the complete metabolic map of an organism, and to analyze, metabolomics data using the Omics Viewer. The website also provides a spectrum of browsing capabilities such as moving from a display of an enzyme to a display of a reaction that the enzyme catalyzes or to the gene that encodes the enzyme (69).

MICROARRAY AND BOUTIQUE BIOINFORMATIC DATABASES

Both microarray databases and boutique databases offer interesting perspectives on biological data. The microarray databases allow users to retrieve and interact with data from microarray experiments. Boutique databases offer users specialty services concerning a particular aspect of biological data. This section reviews such databases and synopsizes these reviews in Table 4.

The Stanford Microarray Database

The Stanford Microarray Database (SMD) (72) allows researchers to retrieve, analyze, and visualize gene expression data from microarray experiments. The repository also contains literature data and integrates multiple related resources, including SGD (17), YPD and WormPD (73), UniGene (74), dbEST (20), and Swiss-Prot (19).

Due to the large number of experiments and datasets, SMD uses comprehensive interfaces allowing users to efficiently query the database. For each experiment, the database stores the name of the researcher, the source organism of the microarray probe sequences, along with a category and subcategory that describe the biological view of the experiment. The user can create a query using any of these criteria to narrow down the number of experiments.

The Yale Microarray Database

The Yale Microarray Database (YMD) (75) is another repository for gene expression data. It is Web-accessible and enables users to perform several operations, e.g., tracking DNA samples between source plates and arrays and finding common genes/clones across different microarray platforms. Moreover, it allows the user to access the image file server, to enter data, and to get integrated data through linkage of gene expression data to annotation databases for functional analysis (75). YMD provides several means of querying the database. The website contains a query criteria interface (75), which allows the user to perform common queries. The interface also enables the user to choose the format of the output, e.g., which columns to be included and the type of output display (HTML, EXCEL, TEXT, or CLUSTER). Finally, the query output can also be dynamically linked to external annotation databases such as DRAGON (76).

Table 4. Summary of Microarray and Boutique Databases

Database	URL	Feature
The Stanford Microarray Database	http://genome-www5.stanford.edu/	Repository for raw and normalized microarray data
The Yale Microarray Database	http://www.med.yale.edu/microarray/	Repository for raw and normalized microarray data
The Stem Cell Database	http://stemcell.princeton.edu/	Database for human and mice stem cell data
The BrainML Data Server	http://www.neurodatabase.org,	Databases containing information necessary for understanding brain processes

The Stem Cell Database

The Stem Cell Database (SCDb) (77), supported by Princeton University and the University of Pennsylvania, is a unique repository that contains information about hematopoietic stem cells from mice and humans. It is closely associated with the Stromal Cell Database (http://stromalcell.princeton.edu/), also supported by Princeton University and the University of Pennsylvania. Data for this repository are obtained from various peer-reviewed sources, publications, and libraries. Users can query on various aspects of the data, including gene name and other annotations, as well as sequence data (77).

The BrainML Data Server

The BrainML Data Server is a repository containing data that pertain to the understanding of neural coding, information transmission, and brain processes and provides a venue for sharing neuro-physiological data. It acquires, organizes, annotates, archives, delivers, and displays single- and multi-unit neuronal data from mammalian cerebral cortex (78). Users can obtain the actual datasets, provided by several laboratories, all in common format and annotated compatibly. The Web interface provides a tool called QueryTool that allows the user to search by metadata terms submitted by the researchers. Another Tool, Virtual Scilloscope Java Tool, displays time-series and histogram datasets dynamically. The datasets can also be downloaded for analysis.

RESEARCH CHALLENGES AND ISSUES

Although extensive efforts have been made to catalog and store biological and chemical data, there is still a great amount of work to be done. Scientists are figuratively drowning in data. Therefore, there is a strong need for computational tools that allow scientists to slice through the mounds of data to pinpoint information needed for experiments. Moreover, with research methodologies changing from library-based to Web-based, new methods for maintaining the quality of the data are needed. Maintenance and updates on bioinformatic databases require not only automatic tools but in most cases also in the curation process. This process involves manual checks from biologists to ensure that data are valid and accurate before integrating this data into the database. There are two major research challenges in the area of bioinformatic databases: (1) development of software tools that are reliable, scalable, downloadable, platform-independent, user-friendly, high performance, and open source for discovering, extracting, and delivering knowledge from large amounts of text and biomedical data; and (2) development of large-scale ontology-assisted knowledge integration systems. The two issues also give rise to others, such as how we can maintain the quality (79) and the proper provenance of biological data when it is heavily integrated. Some work has been done toward the first issue, as discussed in Ref. 80.

Knowledge Discovery from Data (KDD)

The KDD process, in its most fundamental form, is to extract interesting, nontrivial, implicit, previously unknown, and potentially useful information from data. When applied to bioinformatic databases, KDD refers to diverse activities, including bioinformatic data cleaning and preprocessing, pattern and motif discovery, classification and clustering, biological network modeling, and bioinformatic data visualization, to name a few. An annual KDD Cup is organized as the Data Mining and Knowledge Discovery competition by the ACM Special Interest Group (81, 82). Various KDD tools have been developed to analyze DNA and protein sequences, whole genomes, phylogeny and evolutionary trees, macromolecule structures, and biological pathways. However, many of these tools suffer from inefficiency, low accuracy, and unsatisfactory performance due to factors, including experimental noise, unknown model complexity, visualization difficulties with very high-dimensional data, and the lack of sufficient samples for computational validation. Another problem is that some KDD tools are platform dependent and their availability is limited.

One emerging trend in KDD is to apply machine learning, natural language processing, and statistical techniques to text and biomedical literature mining. The goal is to establish associations between biological objects and publications from literature databases such as MEDLINE, for example, finding all related literature studying the same proteins from different aspects. It has been shown that incorporating information obtained from biomedical literature mining into sequence alignment tools such as BLAST can increase the accuracy of alignment results. This shows an example of combining KDD methods with traditional sequence analysis tools to improve their performance. However, these KDD methods are not yet fully reliable, scalable, or user-friendly, and many of the methods still need to be improved.

Large-Scale Knowledge Integration (LKI)

LKI of heterogeneous, distributed bioinformatic data is supposed to offer users a seamless view of knowledge. However, with a few exceptions, many current bioinformatic systems use hyperlink navigation techniques to integrate World Wide Web repositories. These techniques result in semantically meaningless integrations. Often, websites are not maintained, datasets are poorly curated, or in some cases, the integration has been done improperly. With these concerns, efforts based on current biological data integration that create advanced tools to help deliver knowledge to the bioinformatics community fail or become dataset dependent.

A major research challenge in bioinformatics is integrating and representing knowledge effectively. The informatics community has effectively integrated and visualized data. However, research must be taken to the next phase where knowledge integration and knowledge management becomes a key interest. The informatics community must work with the biomedical community from the ground up. Effective, structured knowledge bases need to be created that are also relatively easy to use. The computer science community is starting to address this challenge with projects in the areas of the Semantic Web and semantic integration. The bioinformatics community has started to create such knowledge bases with projects like the Gene Ontology (GO) and Stanford's biomedical ontology (http://bioontology.org/) (more are listed under the Open Biological Ontology, http://obo.sourceforge.net/). Ontologies and meta-data are only the beginning. It is well known in the computer science community that meta-data management can be a tricky, complicated process. Attempting this in the biomedical realm is downright difficult. Researchers currently must wield complicated ontologies to classify even more complex data. Extensive research is needed into how to develop better ontologies as well as to manipulate them more effectively.

Ontology also assumes that there is a general consensus within the bioinformatics field as to the format and structure of the data, with mechanisms for minimizing synonyms and homonyms. This is not true for many types of data. For example, many plant species have binomial names identical to animal species. Many genes have been given different names when found in one species or one tissue as compared with another. In almost every area of medicine as well as biology, researchers can identify contentious nomenclature issues. This standardized naming problem has serious consequences.

KDD and LKI are not separate; rather, they interact with each other closely. For example, as mentioned, one area of KDD is extracting knowledge from peer-reviewed journal articles for clinical use. However, due to the variety of ways to specify biological objects such as species, regulatory pathways, and gene names, KDD tools have difficulty extracting knowledge from these articles. These articles often represent a majority of data the scientific community have concerning the various biological objects. Due to the lack of standardized representations, one can only employ information retrieval algorithms and give the user a confidence level to the knowledge extracted. Great amounts of knowledge are lost because we cannot exploit a standardized knowledge base while examining peer-reviewed literature. As another example, the GO contains a graph structure that illustrates the relationship among molecular functions attributed to genes. If this structure can be combined with KDD processes such as clustering and classification algorithms, one can produce more biologically meaningful clusters or classification outcomes. These examples illustrate the importance of combining KDD and LKI, which is a challenging problem in the field.

Data Provenance

As demonstrated by the above databases as well as the previous issues, there are large quantities of data interchanging between tens if not hundreds of databases regularly. Furthermore, scientists are revolutionizing how research is done by relying more and more on the biological databases and less and less on original journal articles. Thus, the issue of preserving how the data are obtained becomes a paramount concern (83). The field of data provenance investigates how to maintain meta-data describing the history of a data item within the database. With databases cross-listing each other's entries, and with data mining and knowledge discovery algorithms generating new information based on data published in these databases, the issue of data provenance becomes more and more significant.

BIBLIOGRAPHY

1. D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, D. L. Wheeler, GenBank, *Nuc. Acids Res.*, **28**: 15–18, 2000.
2. G. Cochrane, P. Aldebert, N. Althorpe, M. Andersson, W. Baker, A. Baldwin, et al., EMBL Nucleotide Sequence Database: developments in 2005, *Nuc. Acids Res.*, **34**(1): D10–D15, 2006.
3. E. F. Codd, A relational model of data for large shared data banks, *CACM*, **13**(6): 377–387, 1970.
4. A. M. Lesk, *Database Annotation in Molecular Biology*. West Sussex, England: John Wiley & Sons, 2005.
5. M. Y. Galperin, The molecular biology database collection: 2006 update, *Nuc. Acids Res.*, **34**: D3–D5, 2006.
6. J. T. L. Wang, C. H. Wu, and P. P. Wang, *Computational Biology and Genome Informatics*, Singapore: World Scientific Publishing, 2003.
7. K. Okubo, H. Sugawara, T. Gojobori, and Y. Tateno, DDBJ in preparation for overview of research activities behind data submissions *Nuc. Acids Res.*, **34**(1): D6–D9, 2006.
8. N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. DeCastro, P. S. Langendijk-Genevaux, M. Pagni, C. J. A. Sigrist. The PROSITE database. *Nuc. Acids Res.*, **34**(1): D227–D230, 2006.
9. N. J. Mulder, R. Apweiler, T. K. Attwood, A. Bairoch, A. Bateman, D. Binns, et al., InterPro, progress and status in 2005. *Nuc. Acids Res.*, **33**: D201–205, 2006.
10. S. E. Antonarakis and V. A. McKusick, OMIM passes the 1,000-disease-gene mark, *Nature Genet.*, **25**: 11, 2000.
11. V. E. Velculescu, L. Zhang, B. Vogelstein, and K. W. Kinzler, Serial analysis of gene expression. *Science*, **270**, 484–487, 1995.
12. D. L. Wheeler, D. M. Church, A. E. Lash, D. D. Leipe, T. L. Madden, J. U. Pontius, G. D. Schuler, L. M. Schriml, T. A. Tatusova, L. Wagner, and B. A. Rapp, Database resources of the National Center for Biotechnology Information, *Nuc. Acids Res.*, **29**: 11–16, 2001, Updated article: *Nuc. Acids Res.*, **30**: 13–16, 2002.
13. A. J. Enright, I. Iliopoulos, N. C. Kyrpides, and C. A. Ouzounis, Protein interaction maps for complete genomes based on gene fusion events, *Nature*, **402**, 86–90, 1999.
14. T. Hubbard, D. Barker, E. Birney, G. Cameron, Y. Chen, L. Clark, et al., The Ensembl genome database project. *Nuc. Acids Res.*, **30**, 38–41, 2002.
15. C. Hertz-Fowler, C. S. Peacock, V. Wood, M. Aslett, A. Kerhornou, P. Mooney, et al., GeneDB: A resource for prokaryotic and eukaryotic organisms, *Nuc. Acids Res.*, **32**: D339–D343, 2004.
16. D. W. Meinke, J. M. Cherry, C. Dean, S. D. Rounsley, and M. Koornneef, Arabidopsis thaliana: A model plant for genome analysis, *Science*, **282**: 679–682, 1998.
17. J. M. Cherry, C. Adler, C. Ball, S. A. Chervitz, S. S. Dwight, E. T. Hester, Y. Jia, G. Juvik, T. Roe, M. Schroeder, S. Weng, and D. Botstein, SGD: Saccharomyces Genome Database, *Nuc. Acids Res.*, **26**: 73–79, 1998.

18. C. H. Wu, L. S. Yeh, H. Huang, L. Arminski, J. Castro-Alvear, Y. Chen, Z. Z. Hu, R. S. Ledley, P. Kourtesis, B. E. Suzek, C. R. Vinayaka, J. Zhang, W. C. Barker, The protein information resource, *Nuc. Acids Res.*, **31**: 345–347, 2003.
19. C. O'Donovan, M. J. Martin, A. Gattiker, E. Gasteiger, A. Bairoch, and R. Apweiler, High-quality protein knowledge resource: SWISSPROT and TrEMBL. *Brief. Bioinform.*, **3**: 275–284, 2002.
20. M. S. Boguski, T. M. Lowe, and C. M. Tolstoshev, dbEST — database for expressed sequence tags, *Nature Genet.*, **4**: 332–333, 1993.
21. G. D. Schuler, J. A. Epstein, H. Ohkawa, and J. A. Kans, Entrez: Molecular biology database and retrieval system, *Methods Enzymol.*, **266**: 141–162, 1996.
22. C. H. Wu, R. Apweiler, A. Bairoch, D. A. Natale, W. C. Barker, B. Boeckmann, et al., The Universal Protein Resource (UniProt): AN expanding universe of protein information. *Nuc. Acids Res.*, **34**(1): D187–191, 2006.
23. C. H. Wu, A. Nikolskaya, H. Huang, L. S. Yeh, D. A. Natale, C. R. Vinayaka, et al., PIRSF: Family classification system at the Protein Information Resource. *Nuc. Acids Res.*, **32**: D112–114, 2004.
24. C. H. Wu, H. Huang, A. Nikolskaya, Z. Z. Hu, and W. C. Barker, The iProClass integrated database for protein functional analysis. *Comput Biol Chem.*, **28**: 87–96, 2004.
25. Z. Z. Hu, I. Mani, V. Hermoso, H. Liu, C. H. Wu, iProLINK: An integrated protein resource for literature mining. *Comput Biol Chem.*, **28**: 409–416, 2004.
26. E. Gasteiger, E. Jung, and A. Bairoch, SWISS-PROT: Connecting biomolecular knowledge via a protein database, *Curr. Issues Mol. Biol.*, **3**: 47–55, 2001.
27. T. Etzold, and P. Argos, SRS—an indexing and retrieval tool for flat file data libraries. *Comput. Appl. Biosci.*, **9**: 49–57, 2003.
28. J. T. L. Wang, M. J. Zaki, H. T. T. Toivonen, and D. Shasha (eds), *Data mining in Bioinformatics*, London, UK: Springer, 2005.
29. D. R. Maddison, and K.-S. Schulz (eds.). The Tree of Life Web Project. Available: http://tolweb.org. Last accessed July 26, 2006.
30. W. M. Fitch, Distinguishing homologous from analogous proteins, *Syst. Zool.*, **19**: 99–113, 1970.
31. N. Chen, T. W. Harris, I. Antoshechkin, C. Bastiani, T. Bieri, D. Blasiar, et al., WormBase: A comprehensive data resource for Caenorhabditis biology and genomics, *Nuc. Acids Res.*, **33**: D383–D389, 2005.
32. B. J. Haas, J. R. Wortaman, C. M. Ronning, L. I. Hannick, R. K. Smith Jr., et al., Complete reannotation of the Arabidopsis genome: Methods, tools, protocols and the final release. *BMC Biol.*, **3**:7, 2005.
33. P. Dehal, and J. L. Boore, Two rounds of whole genome duplication in the ancestral vertebrate, *PLoS Biol.*, **3**: e314, 2005.
34. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, *Nuc. Acids Res.*, **25**: 3389–3402, 1997.
35. R. C. Edgar, MUSCLE: A multiple sequence alignment method with reduced time and space complexity, *BMC Bioinformatics*, **5**:113, 2004.
36. S. R. Eddy, Profile hidden Markov models. *Bioinformatics*, **14**: 755–763, 1998.
37. N. Saitou and M. Nei, The neighbor-joining method: a new method for reconstructing phylogenetic trees, *Mol. Biol. Evol.*, **4**: 406–425, 1987.
38. C. M. Zmasek and S. R. Eddy, A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics*, **17**: 821–828, 2001.
39. M. J. Sanderson, M. J. Donoghue, W. H. Piel, and T. Eriksson, TreeBASE: A prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life, *Am. J. Bot.*, **81**(6): 163 1994.
40. T. Meinel, A. Krause, H. Luz, M. Vingron, and E. Staub, The SYSTERS Protein Family Database in 2005. *Nuc. Acids Res.*, **33**: D226–D229, 2005.
41. J. Reichert, J. Suhnel, The IMB jena image library of biological macromolecules: 2002 update, *Nuc. Acids Res.*, **30**: 253–254, 2002.
42. H. Boutzelakis, D. Dimitropoulos, J. Fillon, A. Golovin, K. Henrick, A. Hussain, et al., E-MSD: The Eurepoean Bioinformatics Institute Macromolecular Structure Database. *Nuc. Acids Res.*, **31**: 458–462, 2003.
43. A. Bairoch, The ENZYME database in 2000. *Nuc. Acids Res.*, **28**: 304–305, 2000.
44. M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, et al., Gene ontology: Tool for the unification of biology, The Gene Ontology Consortium, *Nature Genetics*, **25**: 25–29, 2000.
45. R. C. Dubes and A. K. Jain. *Algorithms for Clustering Data*, Englewood Cliffs, NJ: Prentice Hall, 1988.
46. A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S. R. Eddy, S. Griffiths-Jones, K. L. Howe, M. Marshall, E. L. L. Sonnhammer, The Pfam protein families database, *Nuc. Acids Res.*, **30**: 276–280, 2002.
47. E. L. L. Sonnhammer, S. R. Eddy, and R. Durbin, Pfam: A comprehensive database of protein domain families based on seed alignments, *Proteins: Struct. Funct. Gene.*, **28**: 405–420, 1998.
48. O. Gascuel, BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data, *Mol. Biol. Evol.*, **14**: 685–695, 1997.
49. W. J. Bruno, N. D. Socci, and A. L. Halpern, Weighted neighbor joining: A likelihood-based approach to distance-based phylogeny reconstruction, *Mol. Biol. Evol.*, **17**: 189–197, 2000.
50. R. Desper and O. Gascuel, Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle, *J. Comput. Biol.*, **9**: 687–705, 2002.
51. S. Guindon and O. Gascuel, A simple, fast and accurate method to estimate large phylogenies by maximum-likelihood, *Syst. Biol.*, **52**: 696–704, 2003.
52. T. N. Bhat, P. Bourne, Z. Feng, G. Gilliland, S. Jain, V. Ravichandran, et al., The PDB data uniformity project. *Nuc. Acids Res.*, **29**, 214–218, 2001.
53. N. Deshpande, K. J. Addess, W. F. Bluhm, J. C. Merino-Ott, W. Townsend-Merino, Q. Zhang, et al., The RCSB Protein Data Bank: A redesigned query system and relational database based on the mmCIF schema, *Nuc. Acids Res.*, **33**: D233–D237, 2005.
54. The Gene Ontology Consortium, Gene Ontology: Tool for the unification of biology, *Nature Genetics*, **25**: 25–29, 2000.
55. G. P. Moss (2006, March 16). Enzyme Nomenclature: Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology on the Nomenclature and Classification of Enzymes by the Reactions they Catalyse, Available: http://www.chem.qmul.ac.uk/iubmb/enzyme/. Accessed: July 27, 2006.
56. M. Kanehisa and S. Goto, KEGG: Kyoto Encyclopedia of Genes and Genomes, *Nuc. Acids Res.*, **28**: 27–30, 2000.

57. D. L. Wheeler, D. M. Church, R. Edgar, S. Federhen, W. Helmberg, T. L. Madden, et al., Database resources of the National Center for Biotechnology Information: update, *Nuc. Acids Res.*, **32**: D35–D40, 2004.

58. F. H. Allen, S. Bellard, M. D. Brice, B. A. Cartwright, A. Doubleday, H. Higgs, et al., The Cambridge crystallographic data centre: Computer-based search, retrieval, analysis and display of information. *Acta Cryst.*, **35**: 2331–2339, 1979.

59. M. S. Babcock and W. K. Olson, A new program for the analysis of nucleic acid structure: implications for nucleic acid structure interpretation, *Computation of Biomolecular Structures: Achievements, Problems, and Perspectives*, Heidelberg: Springer-Verlag, 1992.

60. K. Grzeskowiak, K. Yanagi, G. G. Prive, and R. E. Dickerson, The structure of B-helical C-G-A-T-C-G-A-T-C-G, and comparison with C-C-A-A-C-G-T-T-G-G: the effect ofbase pair reversal. *J. Bio. Chem.*, **266**: 8861–8883, 1991.

61. R. Lavery and H. Sklenar, The definition of generalized helicoidal parameters and of axis curvature for irregular nucleic acids, *J. Biomol. Struct. Dynam.***6**: 63–91, 655–667, 1988.

62. H. M. Berman, A. Gelbin, J. Westbrook, and T. Demeny. *The Nucleic Acid Database File Format*. New Brunswick, NJ: Rutgers University, 1991.

63. S.-H Hsieh. *Ndbfilter. A Suite of Translator Programs for Nucleic Acid Database Crystallographic Archive File Format*. New Brunswick, NJ: Rutgers University, 1992.

64. J. Westbrook, T. Demeny, and S.-H. Hsieh. *Ndbquery. A Simplified User Interface to the Nucleic Acid Database*. New Brunswick, NJ: Rutgers University, 1992.

65. A. R. Srinivasan and W. K. Olson, Yeast tRNAPhC conformation wheels: A novel probe of the monoclinic and orthorhombic models. *Nuc. Acid Res.*, **8**: 2307–2329, 1980.

66. M. Kanehisa, S. Goto, S. Kawashima, and A. Nakaya, The KEGG databases at GenomeNet. *Nuc. Acid Res.*, **30**: 42–46, 2002.

67. M. Kanehisa, *Post-genome Informatics*. Oxford, UK: Oxford University Press, 2000.

68. M. Kanehisa, Pathway databases and higher order function. *Adv. Protein Chem.*, **54**: 381–408, 2000.

69. P. D. Karp, C. A. Ouzounis, C. Moore-Kochlacs, L. Goldovsky, P. Kaipa, D. Ahren, S. Tsoka, N. Darzentas, V. Kunin, and N. Lopez-Bigas, Expansion of the BioCyc collection of pathway/genome databases to 160 genomes, *Nuc. Acids Res.*, **19**: 6083–6089, 2005.

70. R. Caspi, H. Foerster, C. A. Fulcher, R. Hopkinson, J. Ingraham, P. Kaipa, M. Krummenacker, S. Paley, J. Pick, S. Y. R., C. Tissier, P. Zhang and P. D. Karp, MetaCyc: A multiorganism database of metabolic pathways and enzymes, *Nuc. Acids Res.*, **34**: D511–D516, 2006.

71. P. Romero, J. Wagg, M. L. Green, D. Kaiser, M. Krummenacker, and P. D. Karp, Computational prediction of human metabolic pathways from the complete human genome, *Genome Biology*, **6**: 1–17, 2004.

72. C. A. Ball, I. A. Awad, J. Demeter, J. Gollub, J. M. Hebert, T. Hernandez-Boussard, H. Jin, J. C. Matese, M. Nitzberg, F. Wymore, Z. K. Zachariah, P. O. Brown, G. Sherlock, The Stanford Microarray Database accommodates additional microarray platforms and data formats, *Nuc. Acids Res*, **33**: D580–582, 2005.

73. M. C. Costanzo, J. D. Hogan, M. E. Cusick, B. P. Davis, A. M. Fancher, P. E. Hodges, et al., The Yeast Proteome Database (YPD) and *Caenorhabditis elegans* Proteome Database (WormPD): Comprehensive resources for the organization and comparison of model organism protein information. *Nuc. Acids Res.*, **28**: 73–76, 2000.

74. G. D. Schuler, Pieces of the puzzle: Expressed sequence tags and the catalog of human genes, *J. Mol. Med.*, **75**: 694–698, 1997.

75. K. H. Cheung, K. White, J. Hager, M. Gerstein, V. Reinke, K. Nelson, et al., YMD: A microarray database for large-scale gene expression analysis. *Proc. of the American Medical Informatics Association 2002 Annual Symposium*, San Antonio, Texas, November 9–11, 2002, pp. 140–144.

76. C. M. Bouton and J. Pevsner, DRAGON: Database Referencing of Array Genes Online. *Bioinformatics*, **16**(11): 1038–1039, 2000.

77. I. R. Lemischka, K. A. Moore, and C. Stoeckert. (2005) SCDb: The Stem Cell Database, Available: http://stemcell.princeton.edu/. Accessed: July 28, 2006.

78. D. Gardner, M. Abato, K. H. Knuth, R. DeBellis, and S. M Erde, *Philosophical Transactions of the Royal Society B: Biological Sciences*. **356**: 1229–1247, 2001.

79. K. G. Herbert, N. H. Gehani, W. H. Piel, J. T. L. Wang, and C. H. Wu, BIO-AJAX: An Extensible Framework for Biological Data Cleaning, *ACM SIGMOD Record*, **33**: 51–57, 2004.

80. G. Chang, M. Haley, J. A. M. McHugh, J. T. L. Wang, *Mining the World Wide Web*, Norwell, MA: 2001.

81. H. Shatkay, N. Chen, and D. Blostein, Integrating image data into biomedical text categorization. *Bioinformatics*, **22**(14): 446–453, 2006.

82. A. S. Yeh, L. Hirschman, and A. A. Morgan, Evaluation of text data mining for database curation: lessons learned from the KDD Challenge Cup. *Bioinformatics*, **19** (Suppl 1): i331–339, 2003.

83. P. Buneman, A. Chapman, and J. Cheney, Provenance Management in Curated Databases, *Proc. of ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois June 26–29, 2006.

KATHERINE G. HERBERT
Montclair State University
Montclair, New Jersey

JUNILDA SPIROLLARI
JASON T. L. WANG
New Jersey Institute of Technology
Newark, New Jersey

WILLIAM H. PIEL
Peabody Museum of Natural History, Yale University
New Haven, Connecticut

JOHN WESTBROOK
Protein Data Bank and Rutgers, The State University of New Jersey
Piscataway, New Jersey

WINONA C. BARKER
ZHANG-ZHI HU
CATHY H. WU
Protein Information Resource and Georgetown University Medical Center
Washington, D.C.

CONTENT-BASED MULTIMEDIA RETRIEVAL

With rapid advances in storage devices, networks, and compression techniques, large-scale multimedia data has become available to average users. How to index and search multimedia data according to its real content is a challenging problem, which has been studied for decades. Since the 1990s, content-based multimedia retrieval (CBMR) had become an increasingly active field, which is defined as searching for desired multimedia data (images or video/audio segments/clips) relevant with issued queries, such as image/audio/video examples, keywords, phrases, sentences, or any combination of them.

Different from text retrieval, CBMR is a more challenging task, as certain understanding of the content of multimedia data is desired. However, the state-of-the-art techniques for multimedia content understanding are still far from satisfactory. It is well-known that a large gap exists between the semantic meaning of multimedia content (what we really want) and the features of multimedia data (what we can actually get). To reduce this gap, CBMR mainly involves two basic problems. One problem is how to represent queries and multimedia content. The other problem is how to match the representations of queries and multimedia content.

Extensive works have been published on CBMR (1), and different paradigms and techniques have been proposed, such as query-by-example (QBE), annotation-based retrieval (ABR), and multimodality retrieval. The target of all these approaches is to address the above two problems. Table 1 illustrates several techniques and their corresponding solutions in terms of these two problems. QBE is the most typical scenario for multimedia retrieval before the year of 2000, whereas ABR and querying by combination of text and examples have become two new mainstream scenarios thereafter. In this article, we will introduce both classical CBMR and the current mainstream. In detail, we will introduce the following topics: feature extraction, query representation, high-dimensional feature indexing, annotation-based retrieval, and interactive multimedia retrieval. Besides that, we will also introduce several specific techniques for web videos search, including categorization, presentation, and recommendation. Before we introduce these concepts in detail, we give an overview of all these techniques.

OVERVIEW

Features for multimedia retrieval can be categorized into two general types: low-level features and high-level features. Low-level features include global features (such as color histogram, color moment, and texture) and local features (such as regional-level features and features extracted from key-points) (2). High-level features are semantic concepts (or keywords) used to describe the content of multimedia data in text (3–5). The approaches to high-level feature extraction are also called video/image annotation or semantic concept detection. Manual annotation of multimedia data is not only labor-intensive but also a time-consuming process. Thus, automatic annotation methods have been widely applied. Typically, it can be accomplished by machine learning methods. For a given concept, its annotation can be formulated as a binary classification task. As a consequence, multimedia annotation has benefited a lot from the advances of machine learning. More sophisticated methods for multimedia annotation include semisupervised learning, multiinstance learning, multilabel learning, and so on (6).

Queries and multimedia content can be described by low-level features, high-level features, or both. QBE adopts low-level features to retrieve desired data, whereas ABR uses high-level features. In QBE, multimedia data are indexed by low-level features, where users provide examples (such as images or video clips) to retrieve similar results. However, in many cases average users would prefer using text to describe what they want. ABR aims to address this issue, where multimedia data is annotated with a lexicon of semantic concepts (i.e., high-level features), and then queries are also mapped to these concepts and multimedia data can thus be retrieved by text-based matching. Another advantage of ABR based approach is that we can leverage existing text-based indexing and searching technologies to index and search multimedia content.

Multimodality retrieval, which has been actively studied recently, is a combination of these two methods (QBE and ABR) (5). For certain types of queries, for example, those about persons and sports, QBE can achieve better performance, whereas for some queries, such as those about scenes, ABR may be better. Thus, multimodal retrieval can outperform each individual method. It can be accomplished by analyzing the type of queries and then fusing the two methods or adopting the most suitable one. Fusing retrieval results from multimodalities also often generates better ones.

For QBE-based multimedia retrieval, typically low-level features of high dimension are required. Computation cost will be increased significantly when feature dimension is high and the size of the multimedia database is large. Therefore, high-dimensional indexing methods for efficient retrieval, such as k-d tree, are proposed to tackle this difficulty. Dimension reduction is also a frequently applied approach to deal with high-dimensional features (7,8).

As mentioned previously, annotation-based retrieval uses semantic concepts to index and search multimedia content. The main research problem in ABR is automatic annotation, which is typically based on a set of predefined semantic concepts, or we can also call it "concept lexicon," or "multimedia semantic ontology." The semantic ontology should satisfy several requirements, such as sufficient broadness and high feasibility. A frequently applied

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Table 1. Several Techniques for CBMR and their Descriptions

Technique	Description
Query-By-Example (QBE)	Represent queries and multimedia content by low-level features.
Annotation-Based Retrieval (ABR)	Represent queries and multimedia content by high-level features (i.e., semantic concepts).
Multimodality Retrieval	Represent queries and multimedia content by multiple modalities, such as visual features, audio and textual features, or low-level features and high-level features.
Relevance Feedback	Interactively bridging the gap between queries and multimedia content.

semantic ontology in academia is large-scale concept ontology for multimedia (LSCOM) (9).

The gap between queries and multimedia content is a central problem in CBMR. Frequently QBE, ABR, and multimodal retrieval cannot bridge this gap. Interaction is a promising approach to deal with this difficulty, which incorporates human factors into the retrieval loops. Relevance feedback is such an approach, which works in an iterative manner. In each round, users manually label some retrieved samples by indicating whether they are relevant, and then retrieval models can be refined based on these labeled samples. In this way, the semantic gap between high-level queries and low-level features can be gradually bridged. In addition, in ABR, users can improve the retrieval performance by manually revising their query representations (for example, changing/adding/removing keywords) or model parameters according to the current retrieved results.

Nowadays, many video-sharing websites are getting popular and attracting more and more attentions. How to access and manage web videos efficiently has got great interests recently. Already several research efforts have focused on web videos, including categorization, retrieval, and recommendation. Note that web videos have richer information than other videos as they typically have surrounding textual information from web pages accompanied with visual and audio content.

FEATURE EXTRACTION

Feature extraction is the basis of content-based image/video/audio retrieval. The features here include low-level visual/audio features and high-level semantic features. Typical low-level visual features include color, texture, shape, color layout, and spatial location. Low-level audio features generally are computed on a window basis. These window-based features can be considered as a short time description of the signal for that particular moment in time. In some cases, the low-level feature is not efficient for content based retrieval (10). If users want to search images or video segments by text query then semantic concepts are required to describe the images and video segments. These concepts may include, for example, human face, car, water, mountain, beach, people walking, indoor, landscape, and so on. In this section, we will have a brief introduction of the widely-applied low-level features and the normally used visual-semantic mapping methods.

Low-Level Visual Features

To perform content-based retrieval, visual features can be either extracted from the entire image/key-frame (global features) or from regions or feature points (local features). Global feature-based retrieval is comparatively simpler, whereas local features-based representation of images is proved to be more consistent to human perception.

Color Feature. Color feature is one of the most widely used features in image and video retrieval. Color features can be defined based on different color spaces, and some typical ones include RGB, LAB, LUV, HSV, and YCrCb (2,10). Generally color features are relatively robust to complex background and nearly invariant to image scale and orientation. Color features can either be extracted from the entire image or regions.

Histogram is the most commonly used color feature for image and video retrieval. It denotes the joint probability of the intensities of one or more of the three color channels. Besides histogram, color moments and color sets are also frequently applied. To overcome the quantization effects in the color histogram, Strieker and Orengo (11) proposed to use the color moments. To facilitate fast search over large-scale image collections, Smith and Chang (12) proposed color sets as an approximation to the color histogram. Correlogram is also a frequently applied feature. It describes image by a table indexed by color pairs, where the kth entry for $<i, j>$ specifies the probability of finding a pixel color j at a distance k from a pixel of color i in the image. Such an image feature turns out to be robust in tolerating large changes in appearance of the same scene caused by changes in viewing positions, changes in the background scene, partial occlusions, camera zoom that causes radical changes in shape, and so on (13).

Texture. Texture provides important structural information of many real-world images, such as fruit skin, trees, clouds, and fabric. Texture features describe the visual patterns that have properties of homogeneity.

Usually applied texture features for describing the visual information include spectral features, such as Gabor texture or wavelet texture, and statistical features (characterizing texture according to local statistical measures), such as the six Tamura texture features and the wold features (2). Among the six Tamura features, that is, *coarseness*, *directionality*, *regularity*, *contrast*, *line-*

likeness, *contrast*, and *roughness*, the first three are more effective, and the other three are related to the first three and have less effectiveness on texture description. Among the various texture features, Gabor texture and wavelet texture are widely used for image retrieval and have been reported to well match the perception of human vision (2). Texture can also be extracted from both the entire image and regions.

Shape. Some content-based visual information retrieval applications require the shape representation of objects that are generally invariant to translation, rotation, and scaling, while others do not. These features include aspect ratio, circularity, Fourier descriptors, moment invariants, consecutive boundary segments, and so on (2,14).

Spatial Location. Besides color and texture, spatial location is also useful for region-based retrieval. For example, "sky" and "sea" could have similar color and texture features, but their spatial locations are typically different in images and videos. "Sky" usually appears at the top of an image, whereas "sea" at the bottom or middle. Spatial location can be defined by two approaches. The first one is absolute spatial location, such as *upper*, *bottom*, *top*, and *centroid*, according to the location of the region in an image, and the other one is relative spatial relationship, such as the directional relationships between objects: *left*, *right*, *above*, and *below*.

Handling Layout Information. Although the global feature (color, texture, edge, etc.) is simple to calculate and can provide reasonable discriminating power in visual information retrieval, it tends to give too many false positives when the image collection is large. Many research results suggested that using layout (both features and spatial relations) is a better solution to image retrieval. To extend the global feature to a local one, a natural approach is to divide the whole image into subblocks and extract features from each subblock. A variation of this approach is the quadtree-based layout approach, in which the entire image was split into a quadtree structure and each tree branch had its own feature to describe its content.

Low-Level Audio Features

Typically, audio analysis algorithms are based on features computed on a window basis. These window-based features can be considered as a short time description of the signal for that particular moment in time. A wide range of audio features exist for audio computing tasks. These features can be divided into two categories: time domain and frequency domain features. The most typical audio features include mel frequency cepstral coefficient (MFCC), zero crossing rates, and short time energy.

Temporal Low-Level Features

Temporal low-level features are calculated from a set of consecutive frames (or a period of time) that have two typical forms: scalar and vector. Scalar temporal features generally are statistical measures along a set of consecutive frames (for visual features) or a period of time (for audio features), say, a shot, a scene, or a 1-second window. Typical scalar temporal feature is the average of general low-level features on a set of frames (for visual features) or a set of time windows (for audio features), for example, average color histogram of the frames in a shot, and average onset rate in a scene. Another exemplary scalar temporal feature is average motion intensity and motion intensity variance within a shot.

Vector temporal feature generally describes the temporal pattern or variance of a given video clip. For example, a trajectory of a moving object, curve of frame-based feature differences, camera motion speed, speaking rate, onset, and so on, are all vector temporal features.

High-Level Semantic Features

Various high-level semantic concepts, such as *Indoor/ Outdoor, People, Car,* and *Water,* occur frequently in the images and video segments. Type of audio, such as music, speech and noise; and type of music, such as rock, classical, and jazz can be regarded as semantic audio features.

Semantic concepts derived from the image or the video segments are very useful for content-based retrieval. However, the low-level features cannot well represent the semantics, which is the well-known semantic gap between low-level feature and high-level semantic. Lots of approaches are proposed to map the low-level feature to high-level semantics. For video data, a great many approaches to high-level semantic extraction have been proposed in the TRECVID competition organized by NIST (15).

The techniques used to derive high-level semantics include (*1*) using ontology to help detect high-level concepts, (*2*) using machine learning methods to map low-level features to high-level semantic concepts, (*3*) generating semantic template to support high-level image retrieval, and (*4*) making use of both the textual information obtained from the Web and the visual content of images for Web multimedia data annotation. Most systems exploit one or more of the above techniques to implement high-level semantic-based retrieval. As some of these topics will be discussed in other sections, we will just have a brief introduction on the concept ontology and machine learning methods.

Concept Ontology. In some cases, semantics can be derived easily from our daily language. For example, sky can be described as *upper, uniform*, and *blue region.* Using such simple semantics, different intermediate-level descriptors are defined to describe the low-level image features, for example, *light green, medium green*, and *dark green.* These descriptors form a simple vocabulary that is the so-called *object-ontology.* Images or video keyframes can be classified into different categories by mapping such descriptors to high-level semantics (keywords) based on our knowledge, for example, *sky* can be defined as *light blue* (color), *uniform* (texture), and *upper* (spatial location).

As automatic annotation techniques attract increasingly more attention in content based retrieval, it becomes ever more important to standardize the defined semantic concepts. Doing so can direct the multimedia research on a well-defined set of semantics. The LSCOM is designed to optimize simultaneously utility to facilitate end-user access, cover a large semantic space, make automated extraction feasible, and increase observability in diverse broadcast news video data sets (15).

In multimedia retrieval, however, ontology typically only consists of a set of terms (concepts/keywords), it actually also contains the relationships of these terms, as well as certain properties of the terms. Ontology with relationship and properties has been introduced to improve the performance of multimedia annotation in recent research.

Machine Learning. In most cases, to derive high-level semantic features requires the use of formal tools such as machine learning. The most commonly used learning techniques in CBMR include supervised learning, semi-supervised learning, and multiple-instance learning. Supervised learning predicts the semantic category label based on a set of input training samples. It is the mostly used learning technique for feature–concept mapping. By leveraging unlabeled data with certain assumptions, semi-supervised learning methods are promising to build more accurate models compared with supervised learning methods. Some have been applied to enhance the performance of multimedia annotation and retrieval. Multiple-instance learning (MIL) is a type of learning algorithms to tackle the problems with coarsely labeled information on bags of instances. This model assumes that instances are contained in bags and the instance labels are hidden. The bag label is related to the hidden labels of the instances as follows: the bag is labeled as positive if any single instance in it is positive, otherwise it is labeled as negative. MIL is widely applied in the content-based image retrieval (CBIR) systems, in which each image is deemed as a labeled bag with multiple instances, and the segmented regions in the images correspond to the instances in the bags. In addition, semantics from video data can be modeled as a multilayer MIL (MLMIL) learning problem, in which both shot and key-frame can be bag when key-frames of the shot and regions in a key-frames are instances, respectively (16).

EVALUATION MEASURES FOR MULTIMEDIA RETREIVAL

Recall and precision are two widely applied ratios to measure the success of a retrieval system. Recall is defined as the number of relevant (or correct) items returned by a query divided by the total number of relevant items in the corresponding data collection. Precision is defined as the number of relevant items returned by a query divided by the total number of items returned by the query. Usually, precision and recall have an inverse relationship: the higher the recall, the lower the precision, and vice versa. The precision-recall (PR) curve gives a more comprehensive illustration of a retrieval system's performance.

Average precision (AP) is also a measure that is often used in CBMR. Different from precision and recall, which actually do not count the rank of the retrieved result, average precision measures the performance of a ranked list (a list of retrieved items in descending order of relevance score). Letx_k be a variable representing the degree of relevance of the kth item in a ranked list that a retrieval system generates for a given query. $x_k = 1$ if the kth item is relevant; otherwise $x_k = 0$. Then the precision of the top-m items is

$$p_m = \frac{1}{m}\sum_{k=1}^{m} x_k \tag{1}$$

Suppose n items are in the ranked list, and R is the number of relevant items in the list, then the average precision of this list is defined as

$$\mathrm{AP} = \frac{1}{R}\sum_{m=1}^{n} x_m p_m \tag{2}$$

Average precision is sensitive to the entire ranking, that is, it will be changed if the order of relevant items and irrelevant ones is changed (order change within relevant items and within irrelevant items will not change average precision). However, this measure is also stable as a small change in ranking only makes a relatively small change in the final score. In addition, average precision has both precision and recall oriented factors, as it is computed over all relevant retrieved items.

Recently researchers proposed a new measure called average typicality precision (ATP) (16), which is more sensitive to order change. As mentioned above, average precision will not change if the order of two relevant (irrelevant) items is changed. That is to say, average precision actually is not sensitive to the "ranks" or the degree of relevant of the retrieved relevant items. ATP takes the degree of relevant (or the typicality score of the retrieved relevant items) into account. Different from counting AP, in which we only care whether a retrieved item is relevant or irrelevant to the query, a score for each retrieved item to indicate the degree of typicality is used to calculate ATP. ATP is defined as

$$\mathrm{ATP} = \frac{1}{n}\sum_{m=1}^{n} tp_m \tag{3}$$

$$tp_m = \begin{cases} \frac{\sum_{r=1}^{m} g_r}{\sum_{r=1}^{m} o_r} & if \quad g_r > 0 \\ 0 & if \quad g_r = 0 \end{cases} \tag{4}$$

where g_r is the ground-truth of the rth sample's typicality score in the ranked list, and o_r is the r-th highest typicality score in the ground-truth. ATP increases when more and more highly relevant items are at the top of the ranked list.

Besides recall, precision, average precision, and average typicality precision, receiver operating characteristic (ROC) curve (which actually is equivalent to PR curve but presented in a different space) and AUC (area under ROC curve) are applied seldom in multimedia retrieval.

QUERY REPRESENTATION

A query is represented in some specific forms by which the user delivers the information of interested multimedia data. Several types of query representation include: QBE, query by text, and multimodality representation.

Query by Text

A text used to specify the query may consist of one or more keywords, phrases, or sentences, which may be matched with the tags, annotations, title, and/or filename of the desired content, to deliver high-level semantic information for the interested multimedia data. Most commercial image and video search engines make use of the associated textual information to perform query by text.

The text can describe an object, a scene, or an action that appears in the media (e.g. video). For example, the keyword, "the great wall," means that the ideal video should contain the object. A query, represented by a phrase "opening a door" may aim to find a video that contains the behavior of opening a door. The text can also reflect the genre of the queried video. For example, a query by word "comedy" means that the user is interested in the comedy videos.

A query represented by a text is an easy way to deliver users' query requirements. Extracting the semantic information from the video content is still a challenging problem, and at present it is far away from practicality. It is very useful in the case that some textual descriptions, including the title, synopsis, tags, and so on, are associated with the videos in the database. And automatic annotation is also another type of text information that can be used to match with textual query. Then the techniques of document retrieval are borrowed for multimedia retrieval.

Query by Example (QBE)

QBE is a method that allows a user to specify a query condition by providing examples that contain the features of interest. Using this representation, the query condition may be directly comparable with the objects. For subsequent comparison, the multimedia data items in the database may be put into a preprocessor to extract the representative features.

Compared with query by text, query by example is an intuitive way of representing the user's interest and also an effective way to deliver the interest information because it can represent subtle difference more easily than representation by words. However, extracting the hidden interest information from the examples is not as straightforward as query by text. To accomplish this goal, video and/or audio analyzing and understanding techniques are required.

QBE for Image Retrieval. In QBE for image retrieval, five popular example representations can be used with different specified features: image, shape, spatial relation, color, and texture (1,2,10,17–20).

The simplest representation of QBE is just a sample image that is much related or similar to the desired images in the database. This representation is easily given by a user, but it is not a good way to pass the manifest interest feature. A subsequent interactive relevance feedback, such as correcting the retrieved results, may help clarify this query. For example, a user may label some retrieved images as relevant, and some retrieved image as irrelevant. Exploring this feedback and the original query specification will make the interest information clearer so that the retrieval will be refined (21,22). More details about relevance feedback will be introduced later in this article.

Query by shape aims to retrieve an image object by evaluating the shapes of object. This query representation is usually applied to the shape-related retrieval task. Two forms of shape specification include: an image with a shape to be retrieved and a shape drawn by a user (or query by sketch).

Query by spatial relation is mainly adopted in geographical image retrieval (e.g., building and river retrieval). This query specifies one or more component objects as an example, in which the important information, the spatial relation among them, is delivered. The possible relations consist of disjoint, meet, overlap, contain, cover, inside, covered-by, equal, and others.

Query by color aims to find images such that they have principal color values and/or spatial distribution similar to the query specification. The query can be specified by just giving colors and/or the spatial distribution. Query by color is often accompanied by the specification of spatial relation of color components specified. In query evaluation, the color difference, its spatial distribution, and the spatial relation of color components may be valued together or separately to the degree of meeting to the query specification.

Query by texture is typically applied to the task that images with specific pattern are queried. This query is specified by a texture image, and then the image database is searched to find images that contain textures similar to the specification.

QBE for Video Retrieval. A video consists of a series of images. The simple way is just viewing it as a collection of images, and then the techniques of query by example for image retrieval may be used directly to specify the query for video retrieval. However, the most important characteristic of videos, which are different from the image, is the temporal relation. A possible query may be a video clip in which the temporal relation is latently encoded. Signal-processing and computer-vision techniques are required to analyze the spatial and temporal relations of video clips to extract the interest information, and sometimes reanalysis is necessary with given relevance feedback. Besides, query specification techniques that use the temporal relations mainly include query by motions and query by spatio-temporal relations.

Query by motions is an approach to retrieve intrinsic features of video data, that is the motion of objects that appear in video. Extracting the motion information directly from a video using the computer vision techniques is still a difficult problem. Motion by sketch is a practical method to realize motion-based retrieval. An example of motion may be specified by moving a mouse, and the trajectory and velocity are sampled. Additionally, size change of the object may also be specified by drawing rectangles along with the timeline. An alternative method is allowing a user to specify trajectory, duration and scaling as well as the basic feature of image such as color, texture, and shape.

Query by spatio-temporal relations aims at specifying the spatio-temporal correlation of multiple objects in the video. A user may define two or more sets of spatial relations sequentially in accordance with the time, which represent spatio-temporal relations of the objects. The temporal relation of each object can be specified by giving the trajectory, velocity, and size. The spatio-temporal relation can also be used for multiple color components, in which the spatial and temporal relations of different color components are specified simultaneously.

QBE for Audio Retrieval. A general approach to content-based audio indexing and retrieval is taken as (23):

(1) Audio is classified into a set of predefined types such as speech, music, and noise. Some types can be divided into small types. For example, music can be classified into classical, rock, jazz, dance, and so on.
(2) Different audio types may be processed and indexed in different ways. For example, if the audio type is speech, then speech recognition is typically applied and the speech is indexed based on recognized text. For other types, a set of low-level and/or perceptual features are often extracted. Low-level features include mel frequency cepstral coefficient (MFCC), zero crossing rates, and short time energy. Exemplary perceptual features include melody, pitch, onset, and mood.
(3) Query audio pieces are then similarly classified, processed, and indexed.
(4) Audio pieces are retrieved based on the similarity between the query index and the audio indices in the database.

QBE for audio retrieval are more applied for retrieving data from music and speech data collections. Query-by-humming (QBH) (24) is a typical setting for music retrieval. QBH system aims at searching a desired piece of music by singing or whistling its tune. It is very useful when you want to find a song from music library but forget its title or artist. In a QBH system, a person humming is entered to the system as an example of a musical phrase. The system retrieves the melody of songs that well match this given example. Humming as the way of representing a query condition for music retrieval is intuitive to a general user. More audio retrieval techniques can be found in Refs. 2,10, and 23.

Query by Multimodality. A query may be specified by multimodality information related with the queried objects, which usually occurs when multiple modalities associated with the queried media are available at hand with a user. Query by multimodality is a more promising specification scheme than any single query specification, and it potentially makes the retrieval easier because it provides more solid interest information. For example, in video retrieval, besides a keyword or phrase, we also can specify a query by giving a video clip or an audio clip at the same time. Multimodality representation also brings a hot topic, that is, how to fuse the formation derived from multimodality.

The most substantial work in this field is presented in the TREC Video Retrieval Evaluation (16). It focuses its efforts to promote progress in content-based retrieval from video via an open, metrics-based evaluation, which is based on the common video datasets and a standard set of queries. The queries include text plus example images and example videos optionally.

A typical multimodal video search system consists of several main components, which include query analysis, uni-modal search, reranking and multimodal fusion. A generic video search framework is illustrated as Fig. 1. By analyzing the query, the multimodal query (i.e., text, key-frames, and shot) are input to individual search models, such as text-based, example-based, and annotation-based model. Then a fusion and reranking model is applied to aggregate the search results.

Usually, video retrieval systems tend to get the most improvement in a multimodal fusion and reranking by leveraging the above three uni-modal search models. In most multimodal fusion systems for video search, different fusion models are constructed for different query classes, with the involvement of human knowledge. However, some query classification methods are designed for a certain video collection, and they may not be appropriate to other collections. How to fuse these uni-model search models remain a challenge and a meaningful research topic. Reranking is a frequently applied technique to tackle this problem. More details about this topic will be presented later in this article.

HIGH-DIMENSIONAL FEATURE INDEXING

The main operations in content-based retrieval, especially for QBE, include similarity evaluation between the features of two objects and feature indexing in the database. The former is time-expensive when the feature is high dimensional, and the latter may suffer from the large-scale database. To make content-based retrieval scalable, especially for QBE, two techniques, dimension reduction and multi-dimensional indexing, are explored for efficient retrieval. Dimension reduction maps high-dimensional features into lower-dimensional space to save computation cost used for feature comparison (similarity calculation). Multidimensional indexing is used to find efficient index structure of a multimedia database to speed up the feature comparison process between the query and the entire database.

Dimension Reduction

Dimension reduction aims to map a higher-dimensional space to a lower-dimensional space (7). Two categories of dimension reduction methods include linear and nonlinear dimension reduction.

Linear Dimension Reduction. Linear dimension reduction consists of two basic steps: linear transformation and dimension discarding. Linear transformation is optimized with various criteria. Principal component analysis (equivalent to Karhunen-Loeve Transformation) finds a linear transformation such that the transformed features are uncorrelated, or equivalently, the covariance matrix of the transformed data is diagonal. The linear transformation is obtained by performing singular value decomposition on the covariance matrix of the original data. The next step discards some dimensions that have smaller variances. By incorporating supervised information, the resulted linear

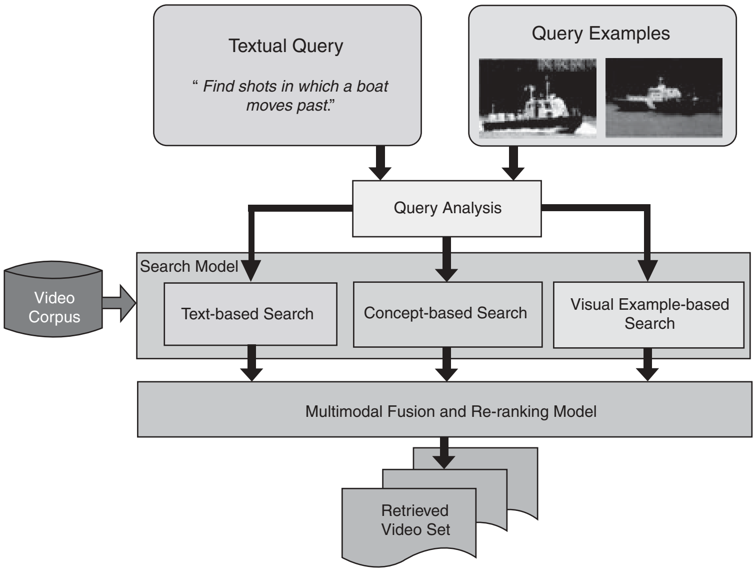


Figure 1. Framework of multimodal video search system.

transformation may be discriminative. Linear discriminant analysis is just such a linear transformation to maximize the between-class variance and minimize the within-class variance in the lower-dimensional space.

Nonlinear Dimension Reduction. Many researchers advocate the use of multidimensional scaling for content-based retrieval applications. MDS comes in different flavors and lacks a general precise definition. A typical approach is to map space R^n into R^m using m transformations, each of which is a linear combination of appropriate radial basis functions. Isometric feature mapping is a special MDS to find a lower-dimensional space such that the Euclidean distance is "close enough" to the "geodesic distance" on the original space. Another popular method, called locally linear embedding, aims to find a lower-dimensional space such that the locally linear reconstructions in the original space are kept. An augmented relation embedding is proposed to map the image space into a semantic manifold that faithfully grasps the user's preferences by combining the similarity relations entailed by the content-based features, and the relevance relations specified in the feedback.

Geometric hashing consists of hashing from a high-dimensional space to a very low-dimensional space. In general, hashing functions are not data dependent, and the metric properties of the hashed space can be significantly different from those of the original space. Ideally, the hash function should spread the database uniformly across the range of the low-dimensionality space, but the design of the hashing function becomes increasingly complex with the dimensionality of the original space. Hence it can be applied to image database only when the dimensionality of the original space is not so high.

The local linear dimension reduction methods, which include clustering and singular value decomposition and vector quantization principal component analysis, are another categorization of nonlinear dimension-reduction method. Those methods consist of two steps. First, the data points are clustered. Second, a linear dimension reduction method is adopted on each local cluster.

Multidimensional Indexing

Given an object database with multidimensional features, an appropriate indexing structure should be constructed to make the retrieval task efficient and effective. Many methods can be used for indexing, which can be roughly divided into two classes: vector-space and metric-space methods (8).

Vector-Space Methods. Vector-space methods represent objects (or feature vectors) as *sets* or *points* in a d-dimensional vector space. For example, gray histograms of images can be viewed as points in high-dimensional (typically 255-dimensional) space, in which each coordinate corresponds to a different bin of the histogram. Algorithmically, vector-space methods can be divided into nonhierarchical methods, hierarchical decomposition methods, and projection-based methods.

The basic nonhierarchical method is a brute-force approach. It scans the whole database table sequentially, and it apparently is very time consuming and not efficient. Besides this naive method, mainly two other classes are suggested: mapping a d-dimensional space onto a real line through a space-filling curve, and partitioning the space into nonoverlapping cells of known size. Most former methods order the database using the positions of the individual items on a space-filling curve, such as the Hibert or Peano-Hilbert curve, or the z-ordering, and obtain a one-dimensional representation. Then, a one-dimensional indexing structure is used to index the mapped records. The space-filling curves tend to map nearby points in the original space into nearby points on a real line, so the one-dimensional indexing techniques, such as range queries, nearest-neighbor queries, and α-cut queries, may be reasonably approximated by executing them in the projected space. The latter partition the search space into a predefined number of nonoverlapping fixed-size regions, which do not dependent on the actual data contained in the database. These two methods are well suited to index low-dimensional spaces (when $d < 10$), but their efficiency decays exponentially when $d > 20$.

Locality sensitive hashing is a recent technique which proposed a locality sensitive hashing function to perform approximate nearest neighbor searching in high-dimensions. The basic idea is to hash the input items so that similar items are mapped to the same buckets with high probability (1).

Hierarchical decomposition methods recursively partition the search space into progressively smaller regions that depend on the dataset. The hierarchical decomposition can be finally represented by a tree. The decomposition techniques vary at the partitioning step with the different tree representation. The typical methods include quadtrees, k-d trees, and R-trees. Those methods were originally developed for low-dimensional search spaces, and unsurprisingly they suffer from the curse of dimensionality and become ineffective in high-dimension cases (when $d > 20$). One of recent researches adopted kd-trees to perform approximate nearest neighbor searching in arbitrarily high dimensions (7,8). The hierarchical clustering methods, such as hierarchical k-means or hierarchical Gaussian mixture models, can also be used for indexing.

Projection-based methods are indexing structures that support approximate nearest-neighbor queries. The techniques vary with different types of approximation performed. Basically two categories exist: fixed-radius queries and $(1 + \varepsilon)$-nearest-neighbor queries. Some former methods project the database onto the coordinate axes, maintain a list for each collection of projections, and use the list to identify a region of the search space quickly that contains a hyper-sphere of radius centered on the query point. Other methods project the database onto appropriate (d + 1)-dimensional hyperplanes, and they find nearest neighbors by tracing an approximate line query point and finding its intersection with the hyperspaces. The latter methods project the high-dimension database into selected or randomly generated lines and index the projections. Both methods are proper for high-dimension queries.

Metric-Space Methods. Metric-space methods index the distances between database items rather than the individual database items. They are useful when the distances are provided with the dataset or where the selected metric is too computationally complex for interactive retrieval (and therefore it is more convenient to compute the pairwise distances while adding items to the database). Most metric-space methods are designed for nearest-neighbor queries, few support alpha-cut, and almost no methods support range queries.

The Voronoi diagram is a method that associates a Voronoi region to each database item if the distance function is given. Different distance functions produce different sets of Voronoi regions. Examples of Voronoi diagrams include cell-and X-tree-based methods.

The ordered list method is proper when all pairwise distances between database items are given. Each database item is associated with an ordered list of all the other items, which are sorted in ascending order of distance. Nearest-neighbor queries are reduced to a point query followed by scanning the list.

Vantage-point methods build a tree structure such that each internal node indexes a disjoint subset of the database, has two children nodes, and is associated with a database item called vantage point. The items indexed by an internal node are well organized according the distance. For example, the median distance is computed, the items closer to the vantage point than the median distance are associated with the left subtree and the remaining ones with the right subtree.

ANNOTATION-BASED RETRIEVAL

Early multimedia search, especially image search, can be traced back to the 1970s. Since the 1990s, it has witnessed strong renaissance in the multimedia search, especially the classic content-based retrieval. Three paradigms exist on the methodological spectrum of the content-based multimedia search. At the earliest extreme, it is the pure manual-labeling paradigm that labels multimedia content, for example, images and video clips, manually with some text labels or concepts and then use text retrieval techniques to search multimedia content indirectly. At the other extreme, it is the automatic content-based search paradigm that can be fully automatic by using the low-level features from multimedia analysis. QBE is the most typical method in this paradigm. However, some difficulties develop in these two paradigms. As for the first manual-based method, a large amount of human labors are required, and the manual labels suffer from the subjectivity of human perception on multimedia content. However, the latter paradigm of the fully automatic method is subject to the well-known "semantic gap" between the low-level features and high-level semantic concepts.

In the past few years, a promising paradigm of the automatic annotation-based multimedia search has been brought into many practical search systems. Compared with the above two extremal paradigms on the methodological spectrum, the third annotation-based paradigm strikes a better balance in the middle and it is an automated method as well (6,9,25). However, this approach is not purely automatic because we need to label some content at the beginning as the training set. It is not purely manual either because once a concept detector is trained based on the labeled training set, the detector can automatically annotate the same concept for other new images and video clips.

Figure 2 illustrates a general framework for automatic annotation (it is also called high-level feature extraction, semantic concept detection, or concept detection in brief). First a set of pre-labeled training samples are used to learn the models of a set of semantic concepts or keywords. These learned models are trained based on some extracted features from training samples, such as color moments, color histograms, color correlogram, wavelet textures and some region features (e.g., SIFT descriptors, shape-based features etc.). These obtained models can then be used to predict the keywords or concepts of any unlabeled images or video clips. Accordingly the trained models here are referred to "classifiers." With these predicted keywords or concepts, the text-based retrieval techniques can be adopted to search the multimedia collections.

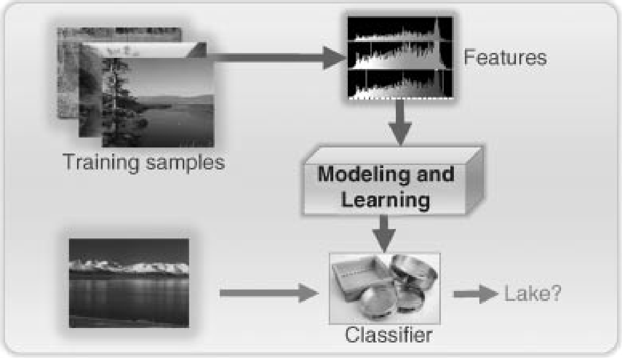


Figure 2. General framework for annotation-based multimedia retrieval.

It is not difficult to recognize that the key factor of the annotation-based paradigm is the classifiers that are used to annotate the keywords to each images and video clips. Some well-defined classifiers have been successfully adopted in content-based retrieval system. These classifiers can be categorized into two main approaches. The first one is the generative models. They explicitly assume the multimedia data is generated by some predefined distributions, such as Hidden Markov Model, Gaussian Mixture Model and so on. These models define a joint probability distribution $P(x, y)$ over the observed low-level features x and their corresponding labels y. In the step of prediction, a conditional distribution $P(y|x)$ is formed to predict the keywords of images or video clips. Opposite to the generative model, the other genre of classifiers is discriminative model. It directly models the dependence of the labels y on the low-level features x, that is, the conditional distribution $P(y|x)$ Some examples of discriminative models used in multimedia retrieval include support vector machine (SVM), boosting, conditional random field, and so on. Both generative and discriminative models are called by "supervised models" for they are all constructed on a prelabeled training set.

In recent years, another type of so-called "semisupervised model" is also applied into annotation-based retrieval. Different from supervised models that only involve training samples in the modeling step, the semisupervised model takes into account the unlabeled samples as well. By leveraging the distribution information revealed by a large amount of unlabeled samples, the semisupervised model can tackle the problem of insufficiency of training samples, and hence the prediction accuracy can be improved. In multimedia retrieval community, existing semisupervised models include manifold models, cotraining models, and so on.

Extracted semantic concepts can be used to index multimedia content directly using text-based technologies, but typically they are integrated with other textual information, such as recognized speech, closed captions, and surrounding text, or even integrated into a multimodality multimedia retrieval system in which QBE techniques are also applied.

The annotation accuracy of the state-of-the-art of concept-detection algorithms is still not satisfactory in the current stage. However, researchers proved that when the number of semantic concepts is relatively large, even it the annotation accuracy is low, annotated semantic concepts (with low accuracy) still can improve the accuracy of the search results significantly (9).

INTERACTIVE MULTIMEDIA RETRIEVAL

As fully automatic content-based retrieval (whether under the typical query-by-example scenario, text-based, or annotation-based scenario) frequently gives unsatisfied results, loop human factor into the retrieval process has been proposed to refine the retrieval accuracy gradually. Typical interactive retrieval method is relevance feedback, whereas interactions can be in other forms, such as manual parameter adjustment and modality selection (3–5,26).

Relevance Feedback

Relevance feedback was originally developed for textual document retrieval and then introduced to CBIR, mainly for QBE scenario. Since then, this topic has attracted tremendous attention in the CBIR community, and a variety of solutions has been proposed within a short period (19,20).

Atypical framework for relevance feedback in CBIR contains the following three main steps, which can be expanded easily to handle video and audio data:

Step 1: For a given query, the machine first provides a list of ranked images according a certain similarity metric.

Step 2: The user marks certain number of ranked images as relevant to the query (positive samples). or irrelevant (negative samples).

Step 3: The machine learns and tries again.

Generally speaking, the earlier CBIR systems without interaction suffer from the two difficulties:

(1) The gap between high-level concepts and low-level features: Low-level visual similarity measures, such as color histogram, do not necessarily match the high-level semantics of images. For example, all three images in Fig. 3 are about cars, but they are of different colors and backgrounds. Their color histograms are very different, although their semantics are close.

(2) Subjectivity of human's perception on media content: Different persons, or the same person under different circumstances, may perceive the same visual content differently. For example, in Fig. 4, one may be interested in the eagle in (a), so (b) is more relevant to (a) compared with (c). But another person may be more interested in the water in (a), and thus (c) is more relevant to what he/she is looking for.

The early relevance feedback schemes for CBIR systems can be classified into two approaches: query point

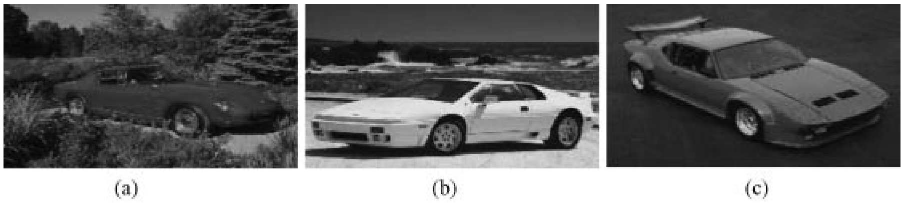


Figure 3. The gap between high-level concepts and low-level features.

movement (query refinement) and reweighting (similarity measure refinement).

The query point movement method tries to improve the estimate of the "ideal query point" by moving it toward positive example points and away from bad example points in the query space. Various ways can be used to update the query. A frequently used technique is Rocchio's formula. That is, for a set of relevant documents D_R and nonrelevant documents D_N given by a user, the optimal query is defined as:

$$Q' = \alpha Q + \beta\left(\frac{1}{N_R}\sum_{i \in D_R} D_i\right) - \gamma\left(\frac{1}{N_N}\sum_{i \in D_N} D_i\right) \qquad (5)$$

where α, β, and γ are suitable constants; N_R and N_N are the number of documents in D_R and D_N, respectively. This technique is also referred to as a learning query vector. It was used in a multimedia retrieval system called "MARS" to replace the document vector with visual feature vectors.

The reweighting method enhances the importance of a feature's dimensions that help retrieve relevant images and reduce the importance of the dimensions that hinder this process. This process is achieved by updating the weights of the feature vectors in the distance metric.

One of the most influential classic researches on relevance feedback has clear roots in the document retrieval field. For example, learning based on "term frequency" and "inverse document frequency" in the text domain was transformed into learning based on ranks of the positive and negative images along each feature axis in the continuous feature space.

The classic query point movement method and reweighting method are both simple learning methods. Relevance feedback can be considered as a learning problem: A user provides feedback about examples retrieved as a result of a query, and the system learns from such examples how to refine the retrieval results.

Three basic issues are generated in relevant feedback for multimedia retrieval, which include small sample size, asymmetry of training samples, and real-time requirement.

The number of training samples is typically less than 20 per round of iteration, which depends on the user's patience and willingness to cooperate. It is very small relative to the dimension of the feature space (from dozens to hundreds, or even more). For such small sample size, some existing learning machines such as SVMS cannot give stable or meaningful results, unless more training samples can be elicited from the user.

In the relevance feedback process, there are usually many more negative feedback samples than positive ones. Learning algorithms tend to be overwhelmed by the major class and ignore the minor one.

Because the user is interacting with the machine in real time, the algorithm should be sufficiently fast and should avoid heavy computations over the whole dataset.

Most published work has focused on the above three issues. However, it should be noted these do not cover all the issues when addressing relevance in CBIR as a learning problem. Other important ones include how to accumulate knowledge learned from the feedback and how to integrate low-level visual and high-level semantic features in query.

Interactive Search

Image/video search is defined as searching for the relevant image/video with issued textual queries (keywords, phrases, or sentences) and/or provided image examples or video clips (or some combination of the two). In the TRECVID video search task hold by NIST, prior work has established that a human searcher in the loop significantly outperforms fully automated video search systems without such a human searcher.

Actually, relevance feedback can also be regarded as a type of interactive search, whereas the interactions in interactive search can be in different forms instead of only indicating whether a certain number of retrieved results are positive or negative. Interactive search is first proposed in TRECVID video search.

In TRECVID video search, interactive search allows users to fuse together multiple searches within each query, which was typically done for answering the query topics. A successful example of interactive search is the IBM Marvel Multimedia Analysis and Retrieval System (IBM, Armonk, NY). It provides search facilities for classic content-based retrieval, model-based (annotation-based) retrieval, and text/speech-based retrieval, and any combination of them. Given the statement of information need and query content, the

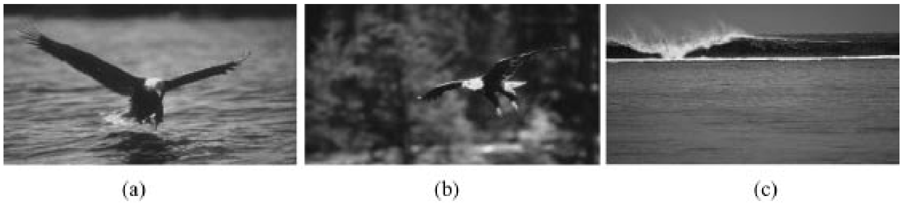


Figure 4. Subjectivity of human perception.

user would typically issue multiple searches based on the example content, models/concepts, and speech terms.

Multiple searches are combined using a set of aggregation functions. Consider scored results list R_k for query k, where $D_k(n)$ gives the score of item with id = n and $Q_d(n)$ the scored result for each item n in the current user-issued search, then the aggregation function rescores the items using function $D_{i+1}(n) = F_a(D_i(n), Q_d(n))$. The following aggregation functions are frequently applied:

Average: Taking the average of the scores of prior result list and current user-search. "Average" deals with "and" semantics, that is, it can be used for queries like "retrieve items that are indoors and contain faces":

$$D_{i+1}(n) = \frac{1}{2}(D_i(n), Q_d(n)) \tag{6}$$

Minimum: Retaining lowest score from prior result list and current user-issued search. "Minimum" provides "or" semantics. This function can be useful in searches such as "retrieve items that are outdoors or have music"

$$D_{i+1}(n) = \min(D_i(n), Q_d(n)) \tag{7}$$

Sum: Taking the sum of scores of prior results list and current user-search. Provides "and" semantics.

$$D_{i+1}(n) = D_i(n) + Q_d(n) \tag{8}$$

Product: Taking the product of scores of prior results list and current user-search. Provides "and" semantics and better favors those matches that have low scores compared to "average."

$$D_{i+1}(n) = D_i(n) \times Q_d(n) \tag{9}$$

Consider user looking for items showing a beach scene. Without interaction, the user can issue a query with the following sequence of searches:

(1) Search for images with color similar to example query images of beach scenes.
(2) Combine results with model = "sky" using "average" aggregation function
(3) Combine with model = "water" using "product" aggregation function
(4) Combine with text = "beach" using "minimum" aggregation function

On the other hand, for interactive search operations the user can issue the following sequence of searches in which the user views the results at each stage, which corresponds to the following query:

(1) Search for text = "beach",
(2) Then, select results that best depict beach scenes and search again based similar color. And then combine with previous results using "product" aggregation function,
(3) Combine with model = "sky" using "average" aggregation function,
(4) Combine with model = "water" using "product" aggregation function

The user interactively builds a query by choosing sequentially among the descriptors and by selecting from various combining and score aggregation functions to fuse results of individual searches. The additional interactivity improves the retrieval performance.

CONTENT-BASED WEB VIDEO RETRIEVAL

Content-based video retrieval, as a challenging yet promising approach to video search, has attracted the attentions of many researchers during the last decades. Many methods are proposed, and some systems are also realized. Currently, with the advent of new video techniques and ever developing network technology, video-sharing websites and video-related Internet-based services are becoming more and more important on the Web. As a result, many videos are generated or uploaded onto the Web every day. This situation brings new challenges and difficulties to content-based video retrieval (27).

Large-Scale Web Video Categorization

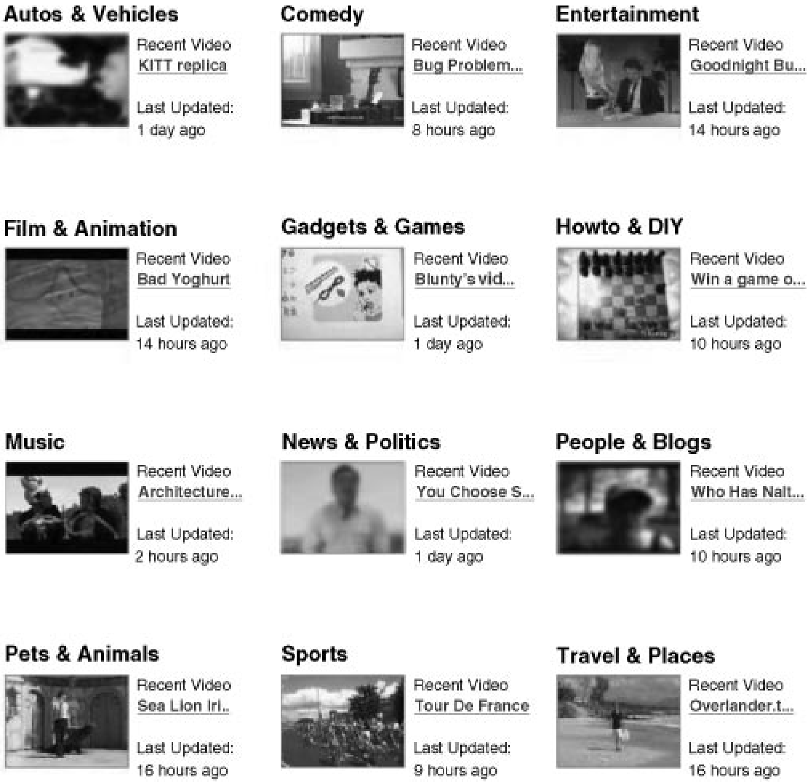
Video categorization is used to classify videos into some predefined categories. Traditionally, most methods of video categorization categorize videos into certain typical genres, such as movies, commercials, cartoons, and so on. However, for web videos, the categories include not only genres, but also semantic concepts, such as autos, animals, and so on. Figure 5 shows the category definition excerpted from YouTube (YouTube, LLC, Mountain View, CA).

Using multimodality is a promising approach to categorize web videos. Low-level and semantic features from different information sources, such as visual, audio, and the surrounding text in the web page affiliated with the video can be applied. A classifier can be trained based on one type of features, and then improved results can be obtained by fusing all the output from these classifiers (27).

Content-Based Search Result Re-Ranking

For video retrieval, many information sources can contribute to the final ranking, including visual, audio, concept detection, and text (ASR, OCR, etc.). Combining the evidence from the various modalities often improves the performance of content-based search significantly (28).

One intuitive solution to combine such modalities is to fuse the ranking scores outputted by the search engines using each modality. However, simply fusing them will lead to inaccuracy and maybe propagate the noise. Some researchers propose to train the fusion model separately for different query classes. Some efforts focus on how to



Autos & Vehicles
Recent Video
KITT replica
Last Updated:
1 day ago

Comedy
Recent Video
Bug Problem...
Last Updated:
8 hours ago

Entertainment
Recent Video
Goodnight Bu...
Last Updated:
14 hours ago

Film & Animation
Recent Video
Bad Yoghurt
Last Updated:
14 hours ago

Gadgets & Games
Recent Video
Blunty's vid...
Last Updated:
1 day ago

Howto & DIY
Recent Video
Win a game o...
Last Updated:
10 hours ago

Music
Recent Video
Architecture...
Last Updated:
2 hours ago

News & Politics
Recent Video
You Choose S...
Last Updated:
1 day ago

People & Blogs
Recent Video
Who Has Nalt...
Last Updated:
10 hours ago

Pets & Animals
Recent Video
Sea Lion Iri..
Last Updated:
16 hours ago

Sports
Recent Video
Tour De France
Last Updated:
9 hours ago

Travel & Places
Recent Video
Overlander.t...
Last Updated:
16 hours ago

Figure 5. The categories excerpted from YouTube.

classify the queries automatically, which still remains an open problem.

In the experiments of video retrieval, text information often leads to better performance than other individual modalities. Hence, some methods emerge that first use text information to rank the videos and then use other evidences to adjust the order of videos in the rank list. It is called reranking. Some researchers used the technique called pseudo relevance feedback to rerank the video search results. They assume that the top and bottom fractions of the rank list can be regarded as the positive and negative samples with significant confidences, and then they use these samples to train a model to rerank the search result.

However, such assumption cannot be satisfied in most situations because the state-of-the-art performance of text-based video search is low. One proposed two methods to rerank the video search result from different viewpoints. One is information theory driven reranking, called IB-reranking. It assumes the videos that have many visually similar videos with high text relevance should be ranked top in the search result. The other is context ranking, which formulate the reranking problem into a random work framework.

Content-based search result reranking, although attracted the attentions from the researchers and is applied successfully to TRECVID video search task, many issues need to be solved. The most important is deploying it into real-world video search system to observe whether it can improve the performance.

Content-Based Search Result Presentation

Search result presentation is a critical component for web video retrieval as it is the direct interface to the users. A good search result presentation makes users browse the videos effectively and efficiently.

Typical video presentation forms include static thumbnail, motion thumbnail, and multiple thumbnails.

Static Thumbnail: an image chosen from the video frame sequence, that can represent the video content briefly.

Motion Thumbnail: a video clip composed of the smaller clips selected from the original video sequence based on some criteria to maximize the information preserved in the thumbnail. Motion thumbnail generation uses the same or similar technology for video summarization.

Multiple Thumbnails: selecting multiple representative frames from the video sequence and show them to the user so that the user can quickly recognize the meaning of the video. The number of thumbnails can be determined by the complexity of the video content or can be scalable (that is, can be changed according to users' request). Multiple thumbnails are also generated by video summarization algorithms.

Content-Based Duplicate Detection

As the techniques and tools for copying, recompressing, and transporting videos can be acquired easily, the same videos, although with different formats, frame rates, frame sizes, or even some editing, so-called duplicates, are distributed over the Internet. In some video services, such as video search and video recommendation, duplicated videos that appear in the results will degrade the user experience. Therefore, duplicate detection is becoming an ever important problem, especially when facing web videos. In addition, integrating the textual information of video duplicates also help build a better video index.

Video duplicate detection is similar to QBE-based video search in that they are both to find similar videos, with the difference in similarity measure between videos. For video search, the returned videos should be the videos semantically related to or matching the query video, whereas for duplicate detection, only the duplicates should be returned. Three components for duplicate detection are video signature generation, similarity computation, and search strategy.

Video signature is a compact and effective video representation. Because videos are a temporal image sequence, a sequence of the feature vectors of the frames sampled from the video can be viewed as the video signature. Also, shot detection can be processed first and then the feature vector sequence extracted from the shot key frames can be regarded as video signature. In addition, the statistical information of the frames can be employed as video signature.

The similarity measure depends on the video signature algorithm, which includes L1 norm, L2 norm, and so on. The high-dimensional indexing techniques can be used to speed up the search for duplicate detection, and some heuristic methods that use the characteristics of the specific video signatures can also be employed.

The difficulties of video duplicate detection lie in that: (*1*) It requires low computational cost for video signature generation as well as the duplicate search so as to be applied in large-scale video set and (*2*) the forms of the video duplicates are diverse so that it requires an effective and robust video signature and similarity measure.

A closely related topic to duplicate detection is copy detection, which is a more general algorithm to detect duplicates. Technologies for copy detection often are also called video signatures. Different from duplicate detection, which only cares about videos of the same or close durations, copy detection also needs to detect whether a short clip exists in a longer video, or even whether two videos share one or more segments. Copy detection is generally used against pirating.

Web Video Recommendation

Helping users to find their desired information is a central problem for research and industry. For video information, basically three approaches are dedicated to this problem: video categorization, video search, and video recommendation. Many video-sharing websites, like YouTube, MSN Soapbox, and even most video search websites, like Google (Mountain View, CA) and Microsoft Live Search (Microsoft, Redmond, WA), have provided video recommendation service.

Video recommendation, in some sense, is related to video search in that they both find the videos related to the query (for video search) or a certain video (for video recommendation). However, they are also different in two aspects. First, video search finds the videos mostly matching the query, whereas video recommendation is to find the videos related to the source video. For example, “Apple” video is not a good search result for a query “Orange”, whereas it may be a good recommendation for an “Orange” video. Second, the user profile is an important component for video recommendation.

Most recommendation systems use the user profiles to provide recommended video lists to the specific user. The user’s click and selection of videos and ratings to some items can be parts of user profile, besides the user location and other user information. As well, most video recommendation systems use the text information (such as the name of actors, actresses, directors for the movie, and the surrounding text for the web video) to generate the recommendation list. Content-based video analysis techniques can provide important evidence besides user profile and text to improve the recommendation. With such information, video recommendation can be formulated into a multi-modality analysis framework (29).

CONCLUSION

As aforementioned, CBMR is more challenging than a text search as it requires certain understanding the content of multimedia data, whereas the state-of-the-art of visual and audio content understanding is still far from applicable. Most existing content-based systems are based on low-level feature comparison, and some are based on limited semantic analysis. One of the most significant problems of existing CBMR systems is that they are basically only applicable for relatively smaller data collections. Existing widely-applied large-scale multimedia retrieval systems, such as, video and image search engines provided by Yahoo (Sunnyvale, CA), Microsoft, and Google, are mostly based on textual keyword indexing and search (surrounding text, file metadata, tags and recognized speech). Development has A long way to go for applicable large-scale content-based multimedia retrieval.

High-dimensional feature indexing is still a challenging problem for QBE-based multimedia retrieval, especially for enabling large-scale data indexing and search. Representative content-aware features are also still not satisfactory in the current stage. Multimedia annotation-based retrieval is a promising direction, but current applicable keywords are still limited and large-scale (in term of both semantic concepts and multimedia dataset) and comprehensive annotation is still difficult. Recently, by leveraging resources including existing textual information, tags, and community of users in CBIR attracts more and more interests of researchers. Some researchers believe leveraging these resources is promising to solve difficult problems as multimedia content understanding. Integrating active learning and/or relevance feedback alike technologies are possible solutions to using these information.

BIBLIOGRAPHY

1. V. N. Gudivada and V. V. Raghavan, Content based image retrieval systems, *Computer* **28** (9): 18–22, 1995.
2. J. Foote, An overview of audio information retrieval, Multim. Syst., **7**: 2–10, 1999.
3. T.-S. Chua et al., TRECVID 2004 search and feature extraction task by NUS PRIS, *TREC Video Retrieval Evaluation Online Proc.*, 2004.

4. M. Campbell, et al., IBM research trecvid-2006 video retrieval system, In *TREC Video Retrieval Evaluation (TRECVID) Proc.*, 2006.
5. S.-F. Chang, et al., Columbia University trecvid-2006 video search and high-level feature extraction, In *TREC Video Retrieval Evaluation (TRECVID) Proc.*, 2006.
6. G.-J. Qi, X.-S. Hua, Y. Rui, J. Tang, T. Mei, and H.-J. Zhang, Correlative multi-label video annotation, *ACM Internat. Conf. Multimedia (ACM MM)*, Augsburg, Germany, 2007.
7. M. Beatty and B. S. Manjunath, Dimensionality reduction using multi-dimensional scaling for content-based retrieval, *ICIP* **2**: 835–838, 1997.
8. V. Castelli, Multidimensional indexing structures for content-based retrieval, *Image Datab.* 373–433, 2002.
9. A. G. Hauptmann, R. Yan, and W.-H. Lin, How many high-level concepts will fill the semantic gap in news video retrieval? *Internat. Conf. on Image and Video Retrieval (CIVR)*, 2007.
10. A. Yoshitaka and T. Ichikawa, A survey on content-based retrieval for multimedia databases, *IEEE Trans. Knowl. Data Eng.*, **11**(1): 81–93, 1999.
11. M. Strieker and M. Orengo, Similarity of color images, *Proc. Of SPIE: Storage and Retrieval for Image and Video Databases III*, Vol. 2420, 1995, pp. 381–392.
12. J. R. Smith and S.-F. Chang, Single color extraction and image query, *Proc. IEEE Int. Conf. on Image Proc.*, 1995.
13. J. Huang, S. R. Kumar, M. Mitra, W.-J. Zhu, and R. Zabih, Image indexing using color correlograms, *Proc. IEEE Comp. Soc. Conf. Comp. Vis. And Patt. Rec.*, 1997, pp. 762–768.
14. K. N. Plataniotis and A. N. Venetsanopoulos, *Color Image Processing and Applications*, Berlin: Springer, 2000.
15. A. F. Smeaton, P. Over and W. Kraaij, 2006. Evaluation campaigns and TRECVid, *Proc. of the 8th ACM International Workshop on Multimedia Information Retrieval (MIR)*, Santa Barbara, California, 2006. ACM Press, New York, 321–330.
16. Z. Gu, Tao Mei, J. Tang, X. Wu and X.-S. Hua. MILC^2: A multi-layer multi-instance learning approach to video concept detection, *Int. Conf. on Multi-Media Modeling (MMM)* Kyoto, Japan, 2008.
17. W. Niblack, R. Barber, et al., The QBIC project: querying images by content using color, texture and shape, *Proc. SPIE Storage and Retrieval for Image and Video Databases*, 1994.
18. W. Y. Ma and B. Manjunath, Netra: a toolbox for navigating large image databases, *Proc. of the IEEE International Conference on Image Processing*, 1997, pp. 568–571.
19. A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, Content-based image retrieval at the end of the early years, *Trans. Patt. Analy. Mach. Intell.*, **22**(12): 1349–1380, 2000.
20. F. Long, H. J. Zhang, and D. G. Feng, Fundamental of content-based image retrieval, In: *Multimedia Information Retrieval and Management*, Berlin: Springer Press, 2002.
21. Y. Rui, T. S. Huang, M. Ortega, and S. Mehrotra, Relevance feedback: a power tool for interactive content-based image retrieval, *IEEE Transactions on Circuits and Systems for Video (CSVT)*, 1998.
22. X. S. Zhou and T. S. Huang, Relevance feedback in image retrieval: a comprehensive review, *Multimedia Sys.*, 2003.
23. G. Lu, Indexing and retrieval of audio: a survey, *Multime. Tools Applicat.*, **15**: 269–290, 2001.
24. A. Ghias, J. Logan, and D. Chamberlin, Query by humming, *Proc. ACM Multimedia*, 1995, pp. 231–236.
25. Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma, A survey of content-based image retrieval with high-level semantics, *Pattern Recognit.*, **40**: 262–282, 2007.
26. J. R. Smith, A. Jaimes, C. Y. Lin, M. Naphade, and A. P. Natsev, Interactive search fusion methods for video database retrieval, *Proc. of International conference on Image Processing (ICIP)*, 2003.
27. L. Yang, J. Liu, X. Yang, and X.-S. Hua, Multi-Modal Web Video Categorization, *ACM SIGMM International Conference Workshop on Multimedia Information Retrieval (ACM MIR)*, in conjunction with ACM Multimedia, Augsburg, Germany, 2007.
28. W. H. Hsu, L. Kennedy, and S.-F. Chang, Reranking methods for visual search, *IEEE Multime. Mag.*, **13**(3), 2007.
29. B. Yang, T. Mei, X.-S. Hua, L. Yang, S.-Q. Yang, and M. Li, Online Video Recommendation Based on Multimodal Fusion and Relevance Feedback, *ACM Internat. Conf. on Image and Video Retrieval (CIVR 2007)*, Amsterdam, The Netherlands, 2007.

FURTHER READING

Y. Rui, T. S. Huang, and S.-F. Chang, Image retrieval: current techniques, promising directions, and open issues, *J. Vis. Communicat. Image Represen.* **10**:39–62,1999.

X.-S. Hau and G.-J. Qi, Online multi-label active annotation – towards large-scale content-based video search, *ACM Multimedia 2008*, Vancouver, Canada, 2008.

X. Tian, L. Yang, et al., Bayesian video search reranking, *ACM Multimedia 2008*, Vancouver, Canada, 2008.

XIAN-SHENG HUA
Microsoft Research Asia
Beijing, China

YONG RUI
Microsoft Advanced Technology Center
Beijing, China

COOPERATIVE DATABASE SYSTEMS

Consider posing a query to a human expert. If the posed query has no answer or the complete data for an answer are not available, one does not simply get a null response. The human expert attempts to understand the gist of the query, to suggest or answer related questions, to infer an answer from data that are accessible, or to give an approximate answer. The goal of cooperative database research is to create information systems with these characteristics (1). Thus, the system will provide answers that cooperate with the user. The key component in cooperative query answering is the integration of a knowledge base (represents data semantics) with the database. Research in cooperative answering stems from three areas: natural language interface and dialogue systems, database systems, and logic programming and deductive database systems. In this article, we shall place emphasis on cooperative databases.

We shall first provide an overview of cooperative database systems which covers such topics as presuppositions, misconceptions, intensional query answering, user modeling, query relaxation, and associative query answering. Then, we present the concept of the Type Abstraction Hierarchy (TAH) which provides a structured approach for query relaxation. Methodologies for automatic TAH generation are discussed. Next, we present the cooperative primitives for query relaxation and selected query examples for relational databases. Then, we present the relaxation controls for providing efficient query processing and the filtering of unsuitable answers for the user. The case-based approach for providing relevant information to query answers is then presented. The performance of a set of sample queries generated from an operational cooperative database system (CoBase) on top of a relational database is reported. Finally, we discuss the technology transfer of successful query relaxation to transportation, logistics planning applications, medical image databases, and electronic warfare applications.

OVERVIEW

Presuppositions

Usually when one asks a query, one not only presupposes the existence of all the components of the query, but one also presupposes an answer to the query itself. For example, suppose one asks “Which employees own red cars?” One assumes there is an answer to the query. If the answer is “nobody owns a red car,” the system should provide the user with further explanation (e.g., in the case where no employee owns a red car because no employee owns a car at all). To avoid misleading the user, the answer should be “There are no employees who own a red car because no employee owns a car at all.” Therefore in many queries, “No” as an answer does not provide the user with sufficient information. Further clarification is necessary to resolve the presupposition problem (2). False presuppositions usually occur with respect to the database’s state and schema. Presuppositions assume that the query has an answer. If any presuppositions are false, the query is nonsensical. The following is a method to detect false presuppositions. Let us represent a query as a graph consisting of arcs at the nodes and binary relations between the arcs. The graph is a semantic network, and the query is reexpressed in binary notation. The query answering system checks to see that each connected subgraph is nonempty. If any is empty, this indicates a failed presupposition. A prototype system called COOP (A Cooperative Query System) was constructed and operated with a CODASYL database to demonstrate such cooperative concepts (3).

Misconceptions

A query may be free of any false presuppositions but can still cause misconceptions. False presuppositions concern the schema of the knowledge base. Misconceptions concern the scope of the domain of the knowledge base. Misconceptions arise when the user has a false or unclear understanding of what is necessarily true or false in the database. For example, for the query, “Which teachers take CS10?”, the corresponding answer will be “None” followed by the explanation from the domain knowledge, “Teachers teach courses” and “Students take courses” (4). Whenever the user poses a query that has no answer, the system infers the probable mismatches between the user’s view of the world and the knowledge in the knowledge base. The system then answers with a correction to rectify the mismatch (5).

Intensional Query Answering

Intensional query answering provides additional information about the extensional answer such as information about class hierarchies that define various data classes and relationships, integrity constraints to state the relationships among data, and rules that define new classes in terms of known classes. Intensional query answering can also provide abstraction and summarization of the extensional answer. As a result, the intensional answers can often improve and compliment extensional answers. For example, consider the query “Which cars are equipped with air bags?” The extensional answer will provide a very long list of registration numbers of all the cars that are equipped with air bags. However, an intensional answer will provide a summaried answer and state “All cars built after 1995 are equipped with air bags.” Note that intensional answering gives more meaning to the answer than does the extensional answer. Furthermore, intensional answers take less time to compute than extensional answers. There are different approaches to compute intensional query answers which yield different quality of answers (6–12). The effectiveness of the answer can be measured by completeness, nonredundancy, optimality, relevance, and efficiency (13).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

User Models

Cooperative query answering depends on the user and context of the query. Thus, a user model will clearly aid in providing more specific query answering and thus improve search efficiency.

User models contain a representation of characteristic information about the user as well as a description of the user's intentions and goals. These models help interpret the content of a user's query and effectively customize results by guiding the query facility in deriving the answer.

Three types of knowledge about a user that are relevant to cooperative query answering are *interests and preferences, needs*, and *goals and intentions*. *Interests and preferences* direct the content and type of answers that should be provided. For example, (14) and (15) rewrite queries to include relevant information that is of interest to the user. *User needs* may vary from user to user. They can be represented by user constraints (16). The notion of user constraints is analogous to the integrity constraints in databases. Unlike integrity constraints, user constraints do not have to be logically consistent with the database. *Goals and intentions* do not vary from user to user. Rather, they vary from session to session and depend on the user who is attempting to achieve the goal. Past dialogue, user models, and other factors can help a system to determine the probable goals and intentions of the user (17–20) and also clarify the user's goals (21). The system can also explain the brief of the system that conflicts with the user's belief to resolve the user's misconceptions (22,23). Hemerly et al. (24) use a predefined user model and maintain a log of previous interactions to avoid misconstruction when providing additional information.

Query Relaxation

In conventional databases, if the required data is missing, if an exact answer is unavailable, or if a query is not well-formed with respect to the schema, the database just returns a null answer or an error. An intelligent system would be much more resourceful and cooperative by relaxing the query conditions and providing an approximate answer. Furthermore, if the user does not know the exact database schema, the user is permitted to pose queries containing concepts that may not be expressed in the database schema.

A user interface for relational databases has been proposed (25) that is tolerant of incorrect user input and allows the user to select directions of relaxation. Chu, et al. (26) proposed to generalize queries by relaxing the query conditions via a knowledge structure called Type Abstraction Hierarchy (TAH). TAHs provide multilevel representation of domain knowledge. Relaxation can be performed via generalization and specialization (traversing up and down the hierarchy). Query conditions are relaxed to their semantic neighbors in the TAHs until the relaxed query conditions can produce approximate answers. Conceptual terms can be defined by labeling the nodes in a type abstraction hierarchy. To process a query with conceptual terms, the conceptual terms are translated into numeric value ranges or a set of nonnumeric information under that node. TAHs can then be generated by clustering algorithms from data sources. There are numerical TAHs that generate by clustering attributes with numerical databases (27,28) and nonnumerical TAHs that generate by rule induction from nonnumerical data sources (29).

Explicit relaxation operators such as approximate, near-to (distance range), and similar-to (based on the values of a set of attributes) can also be introduced in a query to relax the query conditions. Relaxation can be controlled by users with operators such as nonrelaxable, relaxation order, preference list, the number of answers, etc., which can be included in the query. A cooperative languge for relational databases, CoSQL, was developed (30,31) and extended the Structured Query Language (SQL) with these constructs. A cooperative database interface called CoBase was developed to automatically rewrite a CoSQL query with relaxation and relaxation control into SQL statements. As a result, CoBase can run on top of conventional relational databases such as Oracle, Sybase, etc., to provide query relaxation as well as conceptual query answering (answering to a query with conceptual terms) (27,31).

Gaasterland, et al. (32) have used a similar type of abstraction knowledge representation for providing query relaxation in deductive databases by expanding the scope of query constraints. They also used a meta-interpreter to provide users with choices of relaxed queries.

Associative Query Answering

Associative Query Answering provides the user with additional useful relevant information about a query even if the user does not ask for or does not know how to ask for such information. Such relevant information can often expedite the query answering process or provide the user with additional topics for dialogue to accomplish a query goal. It can also provide valuable past experiences that may be helpful to the user in problem solving and decision making. For example, consider the query “Find an airport that can land a C5.” In addition to the query answer regarding the location of the airport, additional relevant information for a pilot may be the *weather* and *runway conditions* of the airport. The additional relevant information for a transportation planner may be the existence of *railway facilities* and *storage facilities* nearby the airport. Thus associative information is both user- and context-sensitive. Cuppens and Demolombe (14) use a rule-based approach to rewrite queries by adding additional attributes to the query vector to provide additional relevant information. They defined a meta-level definition of a query, which specifies the query in three parts: entity, condition, and retrieved attributes. Answers to queries provide values to the variables designated by the retrieved attributes. They have defined methods to extend the retrieved attributes according to heuristics about topics of interest to the user.

CoBase uses a case-based reasoning approach to match past queries with the posed query (33). Query features consist of the query topic, the output attribute list, and the query conditions (15). The similarity of the query features can be evaluated from a user-specific semantic model based on the database schema, user type, and context. Cases with the same topic are searched first. If insufficient cases were found, then cases with related topics are searched. The attributes in the matched cases are then extended to the original query. The extended query is then processed to derive additional relevant information for the user.

STRUCTURED APPROACH FOR QUERY RELAXATION

Query relaxation relaxes a query scope to enlarge the search range or relaxes an answer scope to include additional information. Enlarging and shrinking a query scope can be accomplished by viewing the queried objects at different conceptual levels because an object representation has wider coverage at a higher level and, inversely, more narrow coverage at a lower level. We propose the notion of a type abstraction hierarchy (27–29) for providing an efficient and organized framework for cooperative query processing. A TAH represents objects at different levels of abstraction. For example, in Fig. 1, the Medium-Range (i.e., from 4000 to 8000 ft) in the TAH for runway length is a more abstract representation than a specific runway length in the same TAH (e.g., 6000 ft). Likewise, SW Tunisia is a more abstract representation than individual airports (e.g., Gafsa). A higher-level and more abstract object representation corresponds to multiple lower levels and more specialized object representations. Querying an abstractly represented object is equivalent to querying multiple specialized objects.

A query can be modified by relaxing the query conditions via such operations as generalization (moving up the TAH) and specialization (moving down the TAH, moving, for example, from 6000 ft to Medium-Range to (4000 ft, 8000 ft). In addition, queries may have conceptual conditions such as runway-length = Medium-Range. This condition can be transformed into specific query conditions by specialization. Query modification may also be specified explicitly by the user through a set of cooperative operators such as similar-to, approximate, and near-to.

The notion of multilevel object representation is not captured by the conventional semantic network and object-oriented database approaches for the following reasons. Grouping objects into a class and grouping several classes into a superclass provide only a common *title* (type) for the involved objects without concern for the object instance values and without introducing abstract object representations. Grouping several objects together and identifying their aggregation as a single (complex) object does not provide abstract instance representations for its component objects. Therefore, an object-oriented database deals with information only at two general layers: the metalayer and the instance layer. Because forming an object-oriented type hierarchy does not introduce new instance values, it is impossible to introduce an additional instance layer. In the TAH, instances of a supertype and a subtype may have different representations and can be viewed at different instance layers. Such multiple-layer knowledge representation is essential for cooperative query answering.

Knowledge for query relaxation can be expressed as a set of logical rules, but such a rule-based approach (14) lacks a systematic organization to guide the query transformation process. TAHs provide a much simpler and more intuitive representation for query relaxation and do not have the complexity of the inference that exists in the rule-based system. As a result, the TAH structure can easily support flexible relaxation control, which is important to improve relaxation accuracy and efficiency. Furthermore, knowledge represented in a TAH is customized; thus changes in one TAH represent only a localized update and do not affect other TAHs, simplifying TAH maintenance (see subsection entitled "Maintenance of TAHs"). We have developed tools to generate TAHs automatically from data sources (see the next section), which enable our system to scale up and extend to large data sources.

AUTOMATIC KNOWLEDGE ACQUISITION

The automatic generation of a knowledge base (TAHs) from databases is essential for CoBase to be scalable to large systems. We have developed algorithms to generate automatically TAHs based on database instances. A brief discussion about the algorithms and their complexity follow.

Numerical TAHs

COBWEB (34), a conceptual clustering system, uses category utility (35) as a quality measure to classify the objects described by a set of attributes into a classification tree. COBWEB deals only with categorical data. Thus, it cannot be used for abstracting numerical data. For providing

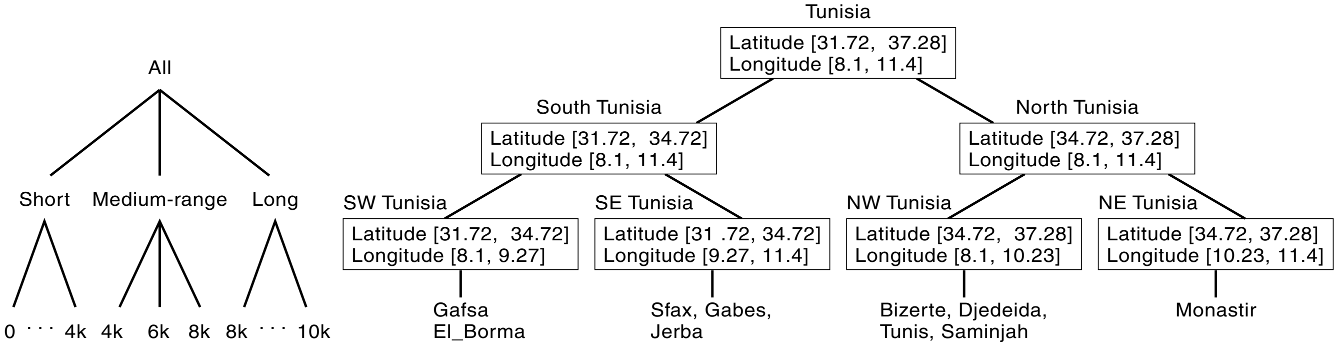


Figure 1. Type abstraction hierarchies: (a) runway length and (b) airport location in Tunisia.

approximate answers, we want to build a classification tree that minimizes the difference between the desired answer and the derived answer. Specifically, we use relaxation error as a measure for clustering. The relaxation error (RE) is defined as the average difference between the requested values and the returned values. $\mathrm{RE}_1(C)$ can also be interpreted from the standpoint of query relaxation. Let us define the relaxation error of x_i, $\mathrm{RE}_1(x_i)$, as the average difference from x_i to x_j, $j = 1, \ldots, n$. That is,

$$\mathrm{RE}_1(x_i) = \sum_{j=1}^{n} P(x_j)|x_i - x_j| \tag{1}$$

where $P(x_j)$ is the occurrence probability of x_j in C. $\mathrm{RE}_1(x_i)$ can be used to measure the quality of an approximate answer where x_i in a query is relaxed to x_j, $j = 1, \ldots, n$. Summing $\mathrm{RE}_1(x_i)$ over all values x_i in C, we have

$$\mathrm{RE}_1(C) = \sum_{i=1}^{n} P(x_i)\mathrm{RE}_1(x_i) \tag{2}$$

Thus, $\mathrm{RE}_1(C)$ is the expected error of relaxing any value in C.

If $\mathrm{RE}_1(C)$ is large, query relaxation based on C may produce very poor approximate answers. To overcome this problem, we can partition C into subclusters to reduce relaxation error. Given a partition $P = \{C_1, C_2, \ldots, C_N\}$ of C, the relaxation error of the partition P is defined as

$$\mathrm{RE}_1(P) = \sum_{k=1}^{N} P(C_k)\mathrm{RE}_1(C_k) \tag{3}$$

where $P(C_k)$ equals the number of tuples in C_k divided by the number of tuples in C. In general, $\mathrm{RE}_1(P) < \mathrm{RE}_1(C)$.

Relaxation error is the expected pairwise difference between values in a cluster. The notion of relaxation error for multiple attributes can be extended from single attributes.

Distribution Sensitive Clustering (DISC) (27,28) partitions sets of numerical values into clusters that minimize the relaxation error. We shall now present a class of DISC algorithms for clustering numerical values. We shall present the algorithm for a single attribute and then extend it for multiple attributes.

The Clustering Algorithm for a Single Attribute. Given a cluster with n distinct values, the number of partitions is exponential with respect to n, so the best partition takes exponential time to find. To reduce computation complexity, we shall consider only binary partitions. Later we shall show that a simple hill-climbing strategy can be used for obtaining N-ary partitions from binary partitions.

Our method is top down: we start from one cluster consisting of all the values of an attribute, and then we find *cuts* to partition recursively the cluster into smaller clusters. (A *cut* c is a value that separates a cluster of numbers $\{x | a \leq x \leq b\}$ into two subclusters $\{x | a \leq x \leq c\}$ and $\{x | c < x \leq b\}$.) The partition result is a concept hierarchy called type abstraction hierarchy. The clustering algorithm is called the DISC method and is given in Table 1.

In Ref. 30, an implementation of the algorithm BinaryCut is presented whose time complexity is $O(n)$. Because DISC needs to execute BinaryCut $n - 1$ times at most to generate a TAH, the worst case time complexity of DISC is $O(n^2)$. [The average case time complexity of DISC is $O(n \log n)$.]

***N*-ary Partitioning.** N-ary partitions can be obtained from binary partitions by a hill-climbing method. Starting from a binary partition, the subcluster with greater relaxation error is selected for further cutting. We shall use RE as a measure to determine if the newly formed partition is better than the previous one. If the RE of the binary partition is less than that of the trinary partition, then the trinary partition is dropped, and the cutting is terminated. Otherwise, the trinary partition is selected, and the cutting process continues until it reaches the point where a cut increases RE.

The Clustering Algorithm for Multiple Attributes. Query relaxation for multiple attributes using multiple single-attribute TAHs relaxes each attribute independently disregarding the relationships that might exist among attributes. This may not be adequate for the applications where

Table 1. The Algorithms DISC and BinaryCut

```
Algorithm DISC(C)
  if the number of distinct values $\in C < T$, return /* $T$ is a threshold */
  let $cut$ = the best cut returned by BinaryCut(C)
  partition values in $C$ based on $cut$
  let the resultant subclusters be $C_1$ and $C_2$
    call DISC($C_1$) and DISC($C_2$)

Algorithm BinaryCut(C)
/* input cluster $C = \{x_1, \ldots, x_n\}$ */
  for $h = 1$ to $n - 1$/* evaluate each cut*/
    Let $P$ be the partition with clusters $C_1 = \{x_1, \ldots, x_h\}$ and $C_2 = \{x_{h+1}, \ldots, x_n\}$
    compute $\mathrm{RE}_1(P)$
    if $\mathrm{RE}_1(P) <$ MinRE then
      MinRE $= \mathrm{RE}_1(P)$, $cut = h$/* the best cut */
  Return $cut$ as the best cut
```

attributes are dependent. (Dependency here means that all the attributes as a whole define a coherent concept. For example, the length and width of a rectangle are said to be "semantically" dependent. This kind of dependency should be distinguished from the functional dependency in database theory.) In addition, using multiple single-attribute TAHs is inefficient because it may need many iterations of query modification and database access before approximate answers are found. Furthermore, relaxation control for multiple TAHs is more complex because there is a large number of possible orders for relaxing attributes. In general, we can rely only on simple heuristics such as best first or minimal coverage first to guide the relaxation (see subsection entitled "Relaxation Control"). These heuristics cannot guarantee best approximate answers because they are rules of thumb and not necessarily accurate.

Most of these difficulties can be overcome by using Multiattribute TAH (MTAH) for the relaxation of multiple attributes. Because MTAHs are generated from semantically dependent attributes, these attributes are relaxed together in a single relaxation step, thus greatly reducing the number of query modifications and database accesses. Approximate answers derived by using MTAH have better quality than those derived by using multiple single-attribute TAHs. MTAHs are context- and user-sensitive because a user may generate several MTAHs with different attribute sets from a table. Should a user need to create an MTAH containing semantically dependent attributes from different tables, these tables can be joined into a single view for MTAH generation.

To cluster objects with multiple attributes, DISC can be extended to Multiple attributes–DISC or M-DISC (28). MTAHs are generated. The algorithm DISC is a special case of M-DISC, and TAH is a special case of MTAH. Let us now consider the time complexity of M-DISC. Let m be the number of attributes and n be the number of distinct attribute values. The computation of relaxation error for a single attribute takes $O(n \log n)$ to complete (27). Because the computation of RE involves computation of relaxation error for m attributes, its complexity is $O(mn \log n)$. The nested loop in M-DISC is executed mn times so that the time complexity of M-DISC is $O(m^2n^2 \log n)$. To generate an MTAH, it takes no more than n calls of M-DISC; therefore, the worst case time complexity of generating an MTAH is $O(m^2n^3 \log n)$. The average case time complexity is $O[m^2n^2(\log n)^2]$ because M-DISC needs only to be called $\log n$ times on the average.

Nonnumerical TAHs

Previous knowledge discovery techniques are inadequate for clustering nonnumerical attribute values for generating TAHs for Cooperative Query Answering. For example, Attribute Oriented Induction (36) provides summary information and characterizes tuples in the database, but is inappropriate since attribute values are focused too closely on a specific target. Conceptual Clustering (37,38) is a top-down method to provide approximate query answers, iteratively subdividing the tuple-space into smaller sets. The top-down approach does not yield clusters that provide the best correlation near the bottom of the hierarchy. Cooperative query answering operates from the bottom of the hierarchy, so better clustering near the bottom is desirable. To remedy these shortcomings, a bottom-up approach for constructing attribute abstraction hierarchies called Pattern-Based Knowledge Induction (PKI) was developed to include a nearness measure for the clusters (29).

PKI determines clusters by deriving rules from the instance of the current database. The rules are not 100% certain; instead, they are rules-of-thumb about the database, such as

If the car is a sports car, then the color is red

Each rule has a *coverage* that measures how often the rule applies, and *confidence* measures the validity of the rule in the database. In certain cases, combining simpler rules can derive a more sophisticated rule with high confidence.

The PKI approach generates a set of useful rules that can then be used to construct the TAH by clustering the premises of rules sharing a similar consequence. For example, if the following two rules:

If the car is a sports car, then the color is red
If the car is a sports car, then the color is black

have high confidence, then this indicates that for sports cars, the colors *red* and *black* should be clustered together. Supporting and contradicting evidence from rules for other attributes is gathered and PKI builds an initial set of clusters. Each invocation of the clustering algorithm adds a layer of abstraction to the hierarchy. Thus, attribute values are clustered if they are used as the premise for rules with the same consequence. By iteratively applying the algorithm, a hierarchy of clusters (TAH) can be found. PKI can cluster attribute values with or without expert direction. The algorithm can be improved by allowing domain expert supervision during the clustering process. PKI also works well when there are NULL values in the data. Our experimental results confirm that the method is scalable to large systems. For a more detailed discussion, see (29).

Maintenance of TAHs

Because the quality of TAH affects the quality of derived approximate answers, TAHs should be kept up to date. One simple way to maintain TAHs is to regenerate them whenever an update occurs. This approach is not desirable because it causes overhead for the database system. Although each update changes the distribution of data (thus changing the quality of the corresponding TAHs), this may not be significant enough to warrant a TAH regeneration. TAH regeneration is necessary only when the cumulative effect of updates has greatly degraded the TAHs. The quality of a TAH can be monitored by comparing the derived approximate answers to the expected relaxation error (e.g., see Fig. 7), which is computed at TAH generation time and recorded at each node of the TAH. When the derived approximate answers significantly deviate from the expected quality, then the quality of the TAH is deemed to be inadequate and a regeneration is necessary. The following incremental TAH regeneration procedure can be used. First, identify the node within the TAH that has the worst query relaxations. Apply partial TAH regeneration for all the database instances covered by the node. After several such partial regenerations, we then initiate a complete TAH regeneration.

The generated TAHs are stored in UNIX files, and a TAH Manager (described in subsection entitled "TAH Facility") is responsible to parse the files, create internal representation of TAHs, and provide operations such as generalization and specialization to traverse TAHs. The TAH Manager also provides a directory that describes the characteristics of TAHs (e.g., attributes, names, user type, context, TAH size, location) for the users/systems to select the appropriate TAH to be used for relaxation.

Our experience in using DISC/M-DISC and PKI for ARPA Rome Labs Planning Initiative (ARPI) transportation databases (94 relations, the biggest one of which has 12 attributes and 195,598 tuples) shows that the clustering techniques for both numerical and nonnumerical attributes can be generated from a few seconds to a few minutes depending on the table size on a SunSPARC 20 Workstation.

COOPERATIVE OPERATIONS

The cooperative operations consist of the following four types: context-free, context-sensitive, control, and interactive.

Context-Free Operators

- *Approximate operator* $\wedge v$ relaxes the specified value v within the approximate range predefined by the user. For example, `∧9am` transforms into the interval (8am, 10am).
- *Between* (v_1, v_2) specifies the interval for an attribute. For example, time `between (7am, ∧9am)` transforms into (7am, 10am). The transformed interval is prespecified by either the user or the system.

Context-Sensitive Operators

- *Near-to X* is used for specification of spatial nearness of object X. The near-to measure is context- and user-sensitive. "Nearness" can be specified by the user. For example, `near-to 'BIZERTE'` requests the list of cities located within a certain Euclidean distance (depending on the context) from the city Bizerte.
- *Similar-to X based-on* [(a_1 w_1)(a_2 w_2) $\cdots$ (a_n w_n)] is used to specify a set of objects semantically similar to the target object *X* based on a set of attributes (a_1, a_2, ..., a_n) specified by the user. Weights (w_1, w_2, ..., w_n) may be assigned to each of the attributes to reflect the relative importance in considering the similarity measure. The set of similar objects can be ranked by the similarity. The similarity measures that computed from the nearness (e.g., weighted mean square error) of the prespecified attributes to that of the target object. The set size is bound by a prespecified nearness threshold.

Control Operators

- *Relaxation-order* (a_1, a_2, ..., a_n) specifies the order of the relaxation among the attributes (a_1, a_2, ..., a_n) (i.e., a_i precedes a_{i+1}). For example, `relaxation-order (runway_length, runway_width)` indicates that if no exact answer is found, then runway_length should be relaxed first. If still no answer is found, then relax the runway_width. If no relaxation-order control is specified, the system relaxes according to its default relaxation strategy.
- *Not-relaxable* (a_1, a_2, ..., a_n) specifies the attributes (a_1, a_2, ..., a_n) that should not be relaxed. For example, `not-relaxable location_name` indicates that the condition clause containing location_name must not be relaxed.
- *Preference-list* (v_1, v_2, ..., v_n) specifies the preferred values (v_1, v_2, ..., v_n) of a given attribute, where v_i is preferred over v_{i+1}. As a result, the given attribute is relaxed according to the order of preference that the user specifies in the preference list. Consider the attribute "food style"; a user may prefer Italian food to Mexican food. If there are no such restaurants within the specified area, the query can be relaxed to include the foods similar to Italian food first and then similar to Mexican food.
- *Unacceptable-list* (v_1, v_2, ..., v_n) allows users to inform the system not to provide certain answers. This control can be accomplished by trimming parts of the TAH from searching. For example, `avoid airlines X and Y` tells the system that airlines X and Y should not be considered during relaxation. It not only provides more satisfactory answers to users but also reduces search time.
- *Alternative-TAH (TAH-name)* allows users to use the TAHs of their choices. For example, a vacation traveler may want to find an airline based on its fare, whereas a business traveler is more concerned with his schedule. To satisfy the different needs of the users, several TAHs of airlines can be generated, emphasizing different attributes (e.g., price and nonstop flight).
- *Relaxation-level* (v) specifies the maximum allowable range of the relaxation on an attribute, i.e., [0, v].
- *Answer-set* (s) specifies the minimum number of answers required by the user. CoBase relaxes query conditions until enough number of approximate answers (i.e., $\geq s$) are obtained.
- *Rank-by* ((a_1, w_1), (a_2, w_2), ..., (a_n, w_n)) *METHOD* (*method* $-$ *name*) specifies a method to rank the answers returned by CoBase.

User/System Interaction Operators

- *Nearer, Further* provide users with the ability to control the near-to relaxation scope interactively. Nearer reduces the distance by a prespecified percentage, whereas further increases the distance by a prespecified percentage.

Editing Relaxation Control Parameters

Users can browse and edit relaxation control parameters to better suit their applications (see Fig. 2). The parameters include the relaxation range for the approximately-equal operator, the default distance for the near-to operator, and the number of returned tuples for the similar-to operator.

Cooperative SQL (CoSQL)

The cooperative operations can be extended to the relational database query language, SQL, as follows: The context-free and context-sensitive cooperative operators can be used in conjunction with attribute values specified in the WHERE clause. The relaxation control operators can be used only on attributes specified in the WHERE clause, and the control operators must be specified in the WITH clause after the WHERE clause. The interactive operators can be used alone as command inputs.

Examples. In this section, we present a few selected examples that illustrate the capabilities of the cooperative operators. The corresponding TAHs used for query modification are shown in Fig. 1, and the relaxable ranges are shown in Fig. 2.

Query 1. List all the airports with the runway length greater than 7500 ft and runway width greater than 100 ft. If there is no answer, relax the runway length condition first. The following is the corresponding CoSQL query:

```
SELECT aport_name, runway_length_ft,
  runway_width_ft
FROM aports
WHERE runway_length_ft > 7500 AND
  runway_width_ft > 100
WITH RELAXATION-ORDER (runway_length_ft,
  runway_width_ft)
```

Approximate operator relaxation range

Relation name	Attribute name	∧Range
Aports	Runway_length_ft	500
Aports	Runway_width_ft	10
Aports	Parking_sq_ft	100000
GEOLOC	Latitude	0.001
GEOLOC	Longitude	0.001

Near-to operator relaxation range

Relation name	Attribute name	Near-to range	Nearer/further
Aports	Aport_name	100 miles	50%
GEOLOC	Location_name	200 miles	50%

Figure 2. Relaxation range for the approximate and near-to operators.

Based on the TAH on runway length and the relaxation order, the query is relaxed to

```
SELECT aport_name, runway_length_ft,
  runway_width_ft
FROM aports
WHERE runway_length_ft >= 7000 AND
  runway_width_ft > 100
```

If this query yields no answer, then we proceed to relax the range runway width.

Query 2. Find all the cities with their geographical coordinates near the city Bizerte in the country Tunisia. If there is no answer, the restriction on the country should not be relaxed. The near-to range in this case is prespecified at 100 miles. The corresponding CoSQL query is as follows:

```
SELECT location_name, latitude, longitude
FROM GEOLOC
WHERE location_name NEAR-TO ‘Bizerte’
  AND country_state_name = ‘Tunisia’
WITH NOT-RELAXABLE country_state_name
```

Based on the TAH on location Tunisia, the relaxed version of the query is

```
SELECT location_name, latitude, longitude
FROM GEOLOC
WHERE location_name IN {‘Bizerte’ , ‘Djedeida’ ,
  ‘Gafsa’ ,
      ‘Gabes’ , ‘Sfax’ , ‘Sousse’ , ‘Tabarqa’ ,
        ‘Tunis’}
    AND country_state_name_= ‘Tunisia’
```

Query 3. Find all airports in Tunisia similar to the Bizerte airport. Use the attributes runway_length_ft and runway_width_ft as criteria for similarity. Place more similarity emphasis on runway length than runway width; their corresponding weight assignments are 2 and 1, respectively. The following is the CoSQL version of the query:

```
SELECT aport_name
FROM aports, GEOLOC
WHERE aport_name SIMILAR-TO ‘Bizerte’
      BASED-ON ((runway_length_ft 2.0)
        (runway_width_ft 1.0))
  AND country_state_name = ‘TUNISIA’
  AND GEOLOC.geo_code = aports.geo_code
```

To select the set of the airport names that have the runway length and runway width similar to the ones for the airport in Bizerte, we shall first find all the airports in Tunisia and, therefore, transform the query to

```
SELECT aport_name
FROM aports, GEOLOC
WHERE country_state_name_ = ‘TUNISIA’
  AND GEOLOC.geo_code = aports.geo_code
```

After retrieving all the airports in Tunisia, based on the runway length, runway width, and their corresponding weights, the similarity of these airports to Bizerte can be computed by the prespecified nearness formula (e.g.,

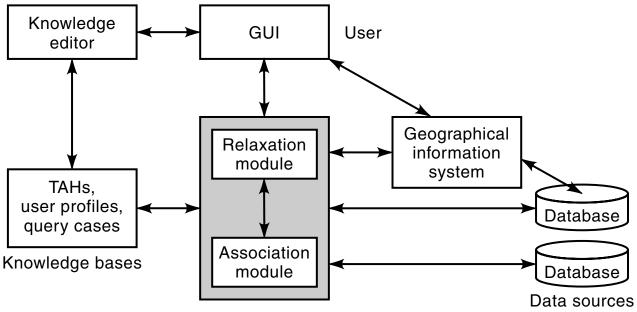


Figure 3. CoBase functional architecture.

weighted mean square error). The order in the similarity set is ranked according to the nearness measure, and the size of the similarity set is determined by the prespecified nearness threshold.

A SCALABLE AND EXTENSIBLE ARCHITECTURE

Figure 3 shows an overview of the CoBase System. Type abstraction hierarchies and relaxation ranges for the explicit operators are stored in a knowledge base (KB). There is a TAH directory storing the characteristics of all the TAHs in the system. When CoBase queries, it asks the underlying database systems (DBMS). When an approximate answer is returned, context-based semantic nearness will be provided to rank the approximate answers (in order of nearness) against the specified query. A graphical user interface (GUI) displays the query, results, TAHs, and relaxation processes. Based on user type and query context, associative information is derived from past query cases. A user can construct TAHs from one or more attributes and modify the existing TAH in the KB.

Figure 4 displays the various cooperative modules: Relaxation, Association, and Directory. These agents are connected selectively to meet applications' needs. An application that requires relaxation and association capabilities, for example, will entail a linking of Relaxation and Association agents. Our architecture allows incremental growth with application. When the demand for certain modules increases, additional copies of the modules can be added to reduce the loading; thus, the system is scalable. For example, there are multiple copies of relaxation agent and association agent in Fig. 4. Furthermore, different types of agents can be interconnected and communicate with each other via a common communication protocol [e.g., FIPA (http.//www.fipa.org), or Knowledge Query Manipulation Language (KQML) (39)] to perform a joint task. Thus, the architecture is extensible.

Relaxation Module

Query relaxation is the process of understanding the semantic context, intent of a user query and modifying the query constraints with the guidance of the customized knowledge structure (TAH) into near values that provide best-fit answers. The flow of the relaxation process is depicted in Fig. 5. When a CoSQL query is presented to the Relaxation Agent, the system first go through a preprocessing phase. During the preprocessing, the system first relaxes any context-free and/or context-sensitive cooperative operators in the query. All relaxation control operations specified in the query will be processed. The information will be stored in the relaxation manager and be ready to be used if the query requires relaxation. The modified SQL query is then presented to the underlying database system for execution. If no answers are returned, then the cooperative query system, under the direction of the Relaxation Manager, relaxes the queries by query modification. This is accomplished by traversing along the TAH node for performing generalization and specialization and rewriting the query to include a larger search scope. The relaxed query is then executed, and if there is no answer, we repeat the relaxation process until we obtain one or more approximate answers. If the system fails to produce an answer due to overtrimmed TAHs, the relaxation manager will deactivate certain relaxation rules to restore part of a trimmed TAH to broaden the search scope until answers are found. Finally, the answers are postprocessed (e.g., ranking and filtering).

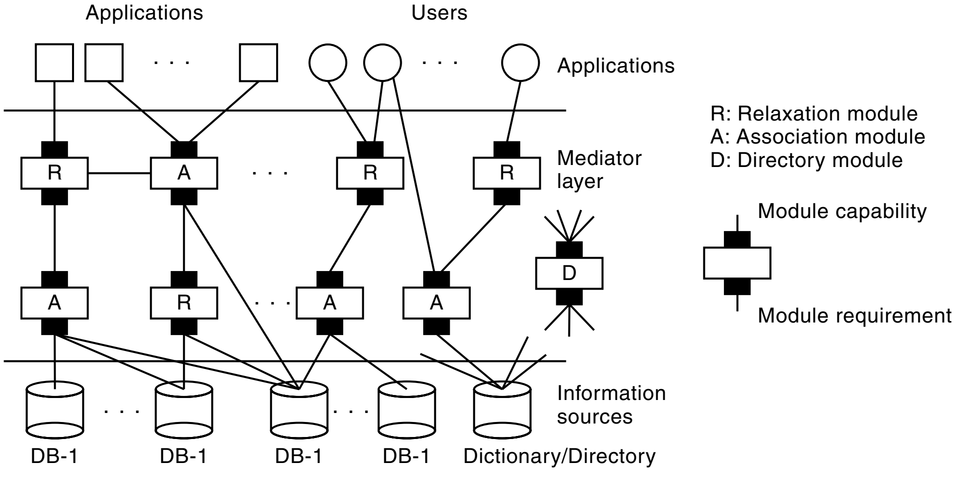


Figure 4. A scalable and extensible cooperative information system.

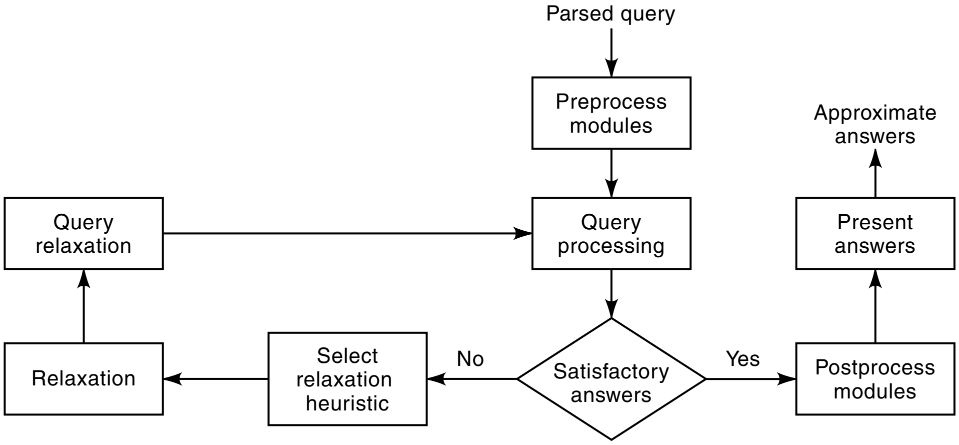


Figure 5. Flow chart for processing CoBase queries.

Relaxation Control. Relaxation without control may generate more approximations than the user can handle. The policy for relaxation control depends on many factors, including user profile, query context, and relaxation control operators as defined previously. The Relaxation Manager combines those factors via certain policies (e.g., minimizing search time or nearness) to restrict the search for approximate answers. We allow the input query to be annotated with control operators to help guide the agent in query relaxation operations.

If control operators are used, the Relaxation Manager selects the condition to relax in accordance with the requirements specified by the operators. For example, a relaxation-order operator will dictate "relax location first, then runway length." Without such user-specified requirements, the Relaxation Manager uses a default relaxation strategy by selecting the relaxation order based on the minimum coverage rule. Coverage is defined as the ratio of the cardinality of the set of instances covered by the entire TAH. Thus, coverage of a TAH node is the percentage of all tuples in the TAH covered by the current TAH node. The minimum coverage rule always relaxes the condition that causes the minimum increase in the scope of the query, which is measured by the coverage of its TAH node. This default relaxation strategy attempts to add the smallest number of tuples possible at each step, based on the rationale that the smallest increase in scope is likely to generate the close approximate answers. The strategy for choosing which condition to be relaxed first is only one of many possible relaxation strategies; the Relaxation Manager can support other different relaxation strategies as well.

Let us consider the following example of using control operators to improve the relaxation process. Suppose a pilot is searching for an airport with an 8000 ft runway in Bizerte but there is no airport in Bizerte that meets the specifications. There are many ways to relax the query in terms of location and runway length. If the pilot specifies the relaxation order to relax the location attribute first, then the query modification generalizes the location Bizerte to NW Tunisia (as shown in Fig. 1) and specifies the locations Bizerte, Djedeida, Tunis, and Saminjah, thus broadening the search scope of the original query. If, in addition, we know that the user is interested only in the airports in West Tunisia and does not wish to shorten the required runway length, the system can eliminate the search in East Tunisia and also avoid airports with short and medium runways, as shown in Fig. 6. As a result, we can limit the query relaxation to a narrower scope by trimming the TAHs, thus improving both the system performance and the answer relevance.

Spatial Relaxation and Approximation. In geographical queries, spatial operators such as located, within, contain, intersect, union, and difference are used. When there are no exact answers for a geographical query, both its spatial and nonspatial conditions can be relaxed to obtain the approximate answers. CoBase operators also can be used for describing approximate spatial relationships. For example, "an aircraft-carrier is *near* seaport Sfax." Approximate spatial operators, such as near-to and between are developed for the approximate spatial relationships. Spatial approximation depends on contexts and domains (40,41). For example, a hospital near to LAX is different from an airport near to LAX. Likewise, the nearness of a hospital in a metropolitan area is different from the one in a rural area. Thus, spatial conditions should be relaxed differently in different circumstances. A common approach to this problem is the use of prespecified ranges. This approach requires experts to provide such information for all possible situations, which is difficult to scale up to larger applications or to extend to different domains. Because TAHs are user- and context-sensitive, they can be used to provide context-sensitive approximation. More specifically, we can generate TAHs based on multidimensional spatial attributes (MTAHs).

Furthermore, MTAH (based on latitude and longitude) is generated based on the distribution of the object locations. The distance between nearby objects is context-sensitive: the denser the location distribution, the smaller the distance among the objects. In Fig. 7, for example, the default neighborhood distance in Area 3 is smaller than the one in Area 1. Thus, when a set of airports is clustered based on the locations of the airports, the ones in the same cluster of the MTAH are much closer to each other than to those outside the cluster. Thus, they can be considered near-to each other. We can apply the same approach to other approximate spatial operators, such as between (i.e., a cluster near-to the center of two objects). MTAHs also can

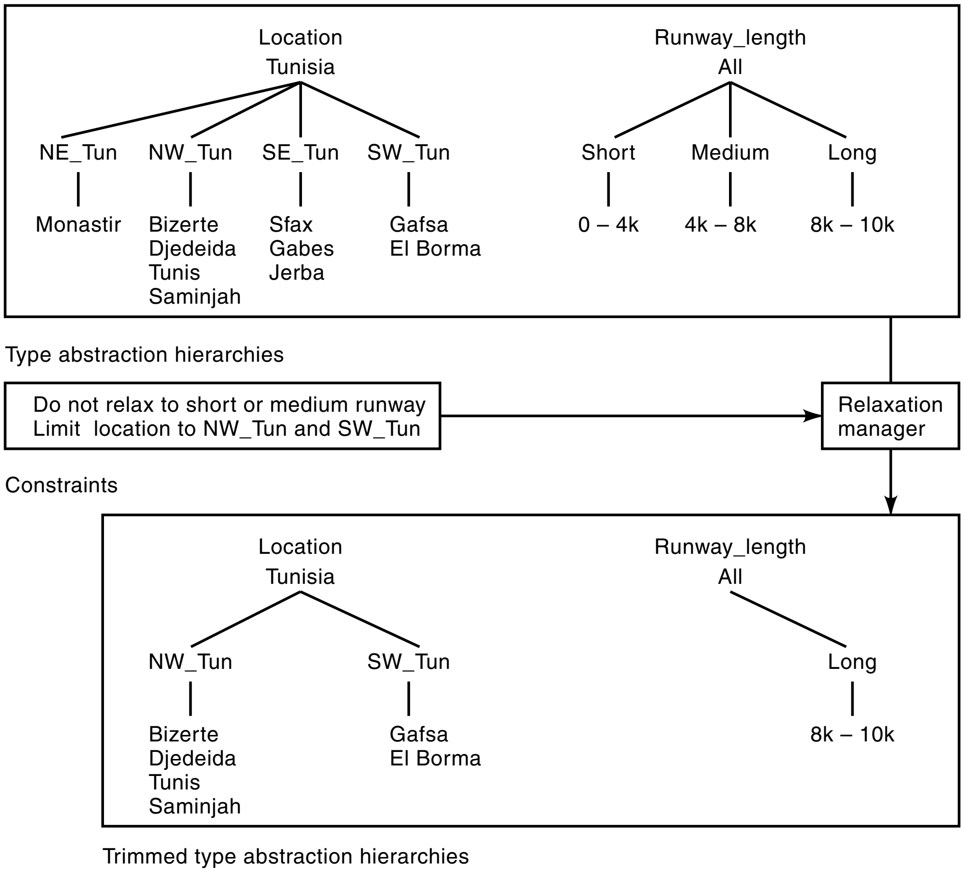


Figure 6. TAH trimming based on relaxation control operators.

be used to provide context-sensitive query relaxation. For example, consider the query: "Find an airfield at the city Sousse." Because there is no airfield located exactly at Sousse, this query can be relaxed to obtain approximate answers. First, we locate the city Sousse with latitude 35.83 and longitude 10.63. Using the MTAH in Fig. 7, we find that Sousse is covered by Area 4. Thus, the airport Monastir is returned. Unfortunately, it is not an airfield. So the query is further relaxed to the neighboring cluster—the four airports in Area 3 are returned: Bizerte, Djedeida, Tunis, and Saminjah. Because only Djedeida and Saminjah are airfields, these two will be returned as the approximate answers.

MTAHs are automatically generated from databases by using our clustering method that minimizes relaxation error (27). They can be constructed for different contexts and user type. For example, it is critical to distinguish a friendly airport from an enemy airport. Using an MTAH for friendly airports restricts the relaxation only within the set of friendly airports, even though some enemy airports are

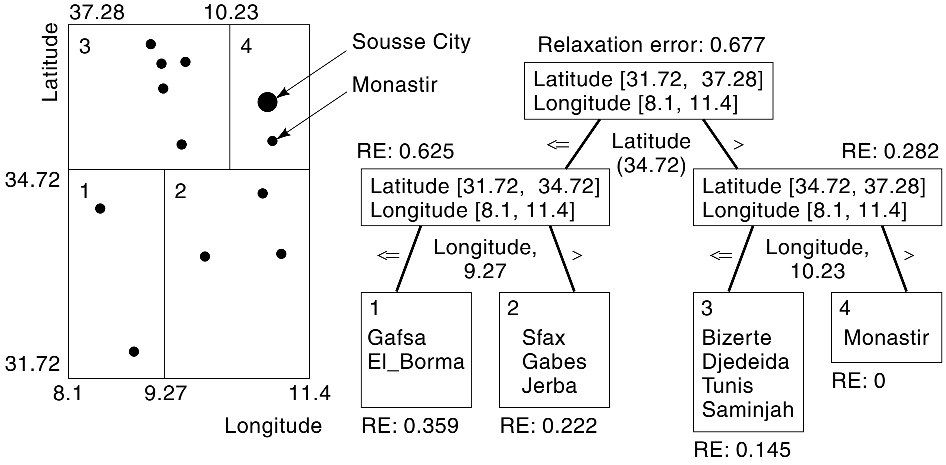


Figure 7. An MTAH for the airports in Tunisia and its corresponding two-dimensional space.

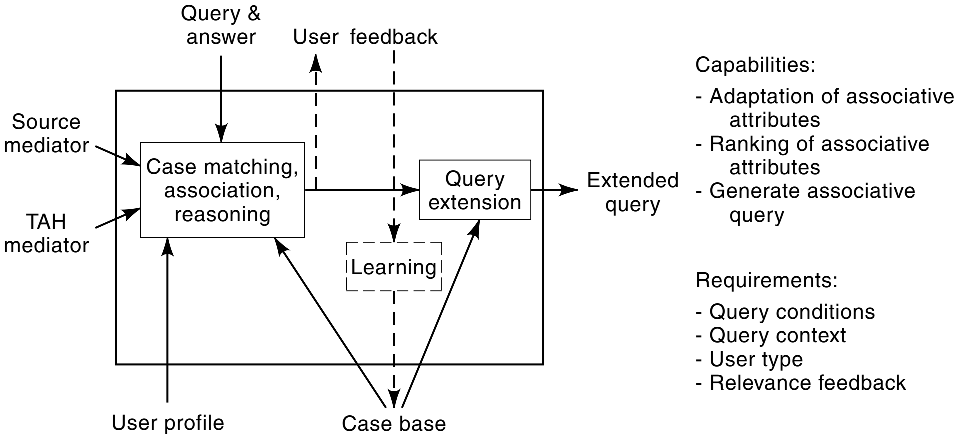


Figure 8. Associative query answering facility.

geographically nearby. This restriction significantly improves the accuracy and flexibility of spatial query answering. The integration of spatial and cooperative operators provides more expressiveness and context-sensitive answers. For example, the user is able to pose such queries as, "find the airports similar-to LAX and near-to City *X*." When no answers are available, both near-to and similar-to can be relaxed based on the user's preference (i.e., a set of attributes). To relax near-to, airports from neighboring clusters in the MTAH are returned. To relax similar-to, the multiple-attribute criteria are relaxed by their respective TAHs.

Cooperativeness in geographic databases was studied in Ref. 42. A rule-based approach is used in their system for approximate spatial operators as well as query relaxation. For example, they define that "P is near-to Q iff the distance from P to Q is less than $n^{*}length\_unit$, where *length_unit* is a context dependent scalar parameter, and n is a scalar parameter that can be either unique for the application and thus defined in domain model, or specific for each class of users and therefore defined in the user models." This approach requires n and *length_unit* be set by domain experts. Thus, it is difficult to scale up. Our system uses MTAHs as a representation of the domain knowledge. The MTAHs can be generated automatically from databases based on contexts and provide a structured and context-sensitive way to relax queries. As a result, it is scalable to large applications. Further, the relaxation error at each node is computed during the construction of TAHs and MTAHs. It can be used to evaluate the quality of relaxations and to rank the nearness of the approximate answers to the exact answer.

Associative Query Answering via Case-Based Reasoning

Often it is desirable to provide additional information relevant to, though not explicitly stated in, a user's query. For example, in finding the location of an airport satisfying the runway length and width specifications, the association module (Fig. 8) can provide additional information about the runway quality and weather condition so that this additional information may help the pilot select a suitable airport to land his aircraft. On the other hand, the useful relevant information for the same query if posed by a transportation planner may be information regarding railway facilities and storage facilities nearby the airport. Therefore, associative information is user- and context-sensitive.

Association in CoBase is executed as a multistep post-process. After the query is executed, the answer set is gathered with the query conditions, user profile, and application constraints. This combined information is matched against query cases from the case base to identify relevant associative information (15,33). The query cases can take the form of a CoBase query, which can include any CoBase construct, such as conceptual conditions (e.g., runway_length_ft = short) or explicitly cooperative operations (city near-to 'BIZERTE').

For example, consider the query

```
SELECT name, runway_length_ft
FROM airports
WHERE runway_length_ft > 6000
```

Based on the combined information, associative attributes such as runway conditions and weather are derived. The associated information for the corresponding airports is retrieved from the database and then appended to the query answer, as shown in Fig. 9.

Our current case base, consisting of about 1500 past queries, serves as the knowledge server for the association module. The size of the case base is around 2 Mb. For association purposes, we use the 300-case set, which is composed of past queries used in the transportation domain. For testing performance and scalability of the

Query answer		Associative information	
Name	Runway_length	Runway_condition	Weather
Jerba	9500	Damaged	Sunny
Monastir	6500	Good	Foggy
Tunis	8500	Good	Good

Figure 9. Query answer and associative information for the selected airports.

system, we use a 1500-case set, which consists of randomly generated queries based on user profile and query template over the transportation domain. Users can also browse and edit association control parameters such as the number of association subjects, the associated links and weights of a given case, and the threshold for association relevance.

PERFORMANCE EVALUATION

In this section, we present the CoBase performance based on measuring the execution of a set of queries on the CoBase testbed developed at UCLA for the ARPI transportation domain. The performance measure includes response time for query relaxation, association, and the quality of answers. The response time depends on the type of queries (e.g., size of joins, number of joins) as well as the amount of relaxation, and association, required to produce an answer. The quality of the answer depends on the amount of relaxation and association involved. The user is able to specify the relaxation and association control to reduce the response time and also to specify the requirement of answer accuracy. In the following, we shall show four example queries and their performances. The first query illustrates the relaxation cost. The second query shows the additional translation cost for the “similar-to” cooperative operator, whereas the third query shows the additional association cost. The fourth query shows the processing cost for returned query answers as well as the quality of answers by using TAH versus MTAH for a very large database table (about 200,000 tuples).

Query 4. Find nearby airports can land C-5.

Based on the airplane location, the relaxation module translates *nearby* to a prespecified or user-specified latitude and longitude range. Based on the domain knowledge of C-5, the mediator also translates *land* into required runway length and width for landing the aircraft. The system executes the translated query. If no airport is found, the system relaxes the distance (by a predefined amount) until an answer is returned. In this query, an airport is found after one relaxation. Thus, two database retrievals (i.e., one for the original query and one for the relaxed query) are performed. Three tables are involved: Table GEOLOC (50,000 tuples), table RUNWAYS (10 tuples), and table AIRCRAFT_AIRFIELD_CHARS (29 tuples). The query answers provide airport locations and their characteristics.

```
Elapsed time: 5 seconds processing time for
                relaxation
              40 seconds database retrieval time
```

Query 5. Find at least three airports similar-to Bizerte based on runway length and runway width.

The relaxation module retrieves runway characteristics of Bizerte airport and translates the similar-to condition into the corresponding query conditions (runway length and runway width). The system executes the translated query and relaxes the runway length and runway width according to the TAHs until at least three answers are returned. Note that the TAH used for this query is a Runway-TAH based on runway length and runway width, which is different from the Location-TAH based on latitude and longitude (shown in Fig. 7). The nearness measure is calculated based on weighted mean square error. The system computes similarity measure for each answer obtained, ranks the list of answers, and presents it to the user. The system obtains five answers after two relaxations. The best three are selected and presented to the user. Two tables are involved: table GEOLOC (50000 tuples) and table RUNWAYS (10 tuples).

```
Elapsed time: 2 seconds processing time for
                relaxation
              10 seconds database retrieval time
```

Query 6. Find seaports in Tunisia with a refrigerated storage capacity of over 50 tons.

The relaxation module executes the query. The query is not relaxed, so one database retrieval is performed. Two tables are used: table SEAPORTS (11 tuples) and table GEOLOC (about 50,000 tuples).

```
Elapsed time: 2 seconds processing time for
                relaxation
              5 seconds database retrieval time
```

The association module returns relevant information about the seaports. It compares the user query to previous similar cases and selects a set of attributes relevant to the query. Two top-associated attributes are selected and appended to the query. CoBase executes the appended query and returns the answers to the user, together with the additional information. The two additional attributes associated are location name and availability of railroad facility near the seaports.

```
Elapsed time: 10 seconds for association
                computation time
```

Query 7. Find at least 100 cargos of code ‘3FKAK’ with the given volume (length, width, height), code is nonrelaxable.

The relaxation module executes the query and relaxes the height, width, and length according to MTAH, until at least 100 answers are returned. The query is relaxed four times. Thus, five database retrievals are performed. Among the tables accessed is table CARGO_DETAILS (200,000 tuples), a very large table.

```
Elapsed time: 3 seconds processing time for
                relaxation using MTAH
              2 minutes database retrieval time
                for 5 retrievals
```

By using single TAHs (i.e., single TAHs for height, width, and length, respectively), the query is relaxed 12 times. Thus, 13 database retrievals are performed.

```
Elapsed time: 4 seconds for relaxation by
                single TAHs
              5 minutes database retrieval time
                for 13 retrievals
```

For queries involving multiple attributes in the same relation, using an MTAH that covers multiple attributes would provide better relaxation control than using a combination of single-attribute TAHs. The MTAH compares favorably with multiple single-attribute TAHs in both quality and efficiency. We have shown that an MTAH yields a better relaxation strategy than multiple single-attribute TAHs. The primary reason is that MTAHs capture attribute-dependent relationships that cannot be captured when using multiple single-attribute TAHs.

Using MTAHs to control relaxation is more efficient than using multiple single-attribute TAHs. For this example, relaxation using MTAHs require an average of 2.5 relaxation steps, whereas single-attribute TAHs require 8.4 steps. Because a database query is posed after each relaxation step, using MTAHs saves around six database accesses on average. Depending on the size of tables and joins involved, each database access may take from 1 s to about 30 s. As a result, using MTAHs to control relaxation saves a significant amount of user time.

With the aid of domain experts, these queries can be answered by conventional databases. Such an approach takes a few minutes to a few hours. However, without the aid of the domain experts, it may take hours to days to answer these queries. CoBase incorporates domain knowledge as well as relaxation techniques to enlarge the search scope to generate the query answers. Relaxation control plays an important role in enabling the user to control the relaxation process via relaxation control operators such as relaxation order, nonrelaxable attributes, preference list, etc., to restrict the search scope. As a result, CoBase is able to derive the desired answers for the user in significantly less time.

TECHNOLOGY TRANSFER OF COBASE

CoBase stemmed from the transportation planning application for relaxing query conditions. CoBase was linked with SIMS (43) and LIM (44) as a knowledge server for the planning system. SIMS performs query optimizations for distributed databases, and LIM provides high-level language query input to the database. A Technical Integration Experiment was performed to demonstrate the feasibility of this integrated approach. CoBase technology was implemented for the ARPI transportation application (45). Recently, CoBase has also been integrated into a logistical planning tool called Geographical Logistics Anchor Desk (GLAD) developed by GTE/BBN. GLAD is used in locating desired assets for logistical planning which has a very large database (some of the tables exceed one million rows). CoBase has been successfully inserted into GLAD (called CoGLAD), generating the TAHs from the databases, providing similarity search when exact match of the desired assets are not available, and also locating the required amount of these assets with spatial relaxation techniques. The spatial relaxation avoids searching and filtering the entire available assets, which greatly reduces the computation time.

In addition, CoBase has also been successfully applied to the following domains. In electronic warfare, one of the key problems is to identify and locate the emitter for radiated electromagnetic energy based on the operating parameters of observed signals. The signal parameters are radio frequency, pulse repetition frequency, pulse duration, scan period, and the like. In a noisy environment, these parameters often cannot be matched exactly within the emitter specifications. CoBase can be used to provide approximate matching of these emitter signals. A knowledge base (TAH) can be constructed from the parameter values of previously identified signals and also from the peak (typical, unique) parameter values. The TAH provides guidance on the parameter relaxation. The matched emitters from relaxation can be ranked according to relaxation errors. Our preliminary results have shown that CoBase can significantly improve emitter identification as compared to conventional database techniques, particularly in a noisy environment. From the line of bearing of the emitter signal, CoBase can locate the platform that generates the emitter signal by using the near-to relaxation operator.

In medical databases that store x rays and magnetic resonance images, the images are evolution and temporal-based. Furthermore, these images need to be retrieved by object features or contents rather than patient identification (46). The queries asked are often conceptual and not precisely defined. We need to use knowledge about the application (e.g., age class, ethnic class, disease class, bone age), user profile and query context to derive such queries (47). Further, to match the feature exactly is very difficult if not impossible. For example, if the query "Find the treatment methods used for tumors *similar to* X_i ($location_{x_i}$, $size_{x_i}$) on 12-year-old Korean males" cannot be answered, then, based on the TAH shown in Fig. 10, we can relax tumor X_i to tumor Class X, and 12-year-old Korean male to pre-teen Asian, which results in the following relaxed query: "Find the treatment methods used for tumor Class X on pre-teen Asians." Further, we can obtain such relevant information as the success rate, side effects, and cost of the treatment from the association operations. As a result, query relaxation and modification are essential

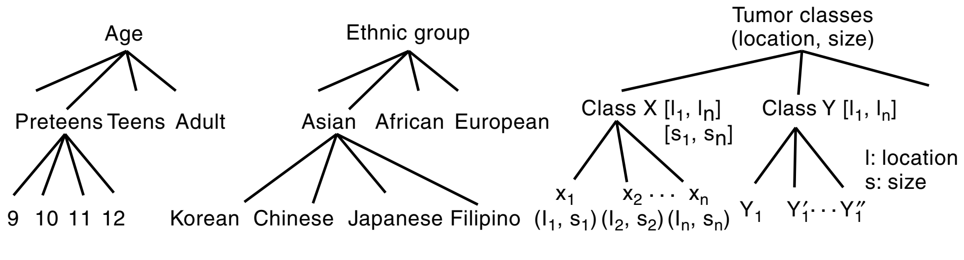


Figure 10. Type abstraction hierarchies for the medical query example.

to process these queries. We have applied CoBase technology to medical imaging databases (48). TAHs are generated automatically based on context-specific (e.g., brain tumor) image features (e.g., location, size, shape). After the TAHs for the medical image features have been constructed, query relaxation and modification can be carried out on the medical features (49).

The use of CoSQL constructs such as *similar-to*, *near-to*, and *within* can be used in combination, thus greatly increasing the expressibility for relaxation. For example, we can express "Find tumors *similar-to* the tumor x *based-on* (shape, size, location) and *near-to* object O *within* a specific range (e.g., angle of coverage)." The relaxation control operators, such as matching tumor features in accordance to their importance, can be specified by the operator *relaxation-order* (location, size, shape), to improve the relaxation quality.

CONCLUSIONS

After discussing an overview of cooperative database systems, which includes such topics as presuppositions, misconceptions, intensional query answering, user modeling, query relaxation, and associative query answering, we presented a structured approach to query relaxation via Type Abstraction Hierarchy (TAH) and a case-based reasoning approach to provide associative query answering. TAHs are user- and context-sensitive and can be generated automatically from data sources for both numerical and nonnumerical attributes. Therefore, such an approach for query relaxation can scale to large database systems. A set of cooperative operators for relaxation and relaxation control was presented in which these operators were extended to SQL to form a cooperative SQL (CoSQL). A cooperative database (CoBase) has been developed to automatically translate CoSQL queries into SQL queries and can thus run on top of conventional relational databases to provide query relaxation and relaxation control.

The performance measurements on sample queries from CoBase reveal that the cost for relaxation and association is fairly small. The major cost is due to database retrieval which depends on the amount of relaxation required before obtaining a satisfactory answer. The CoBase query relaxation technology has been successfully transferred to the logistics planning application to provide relaxation of asset characteristics as well as spatial relaxation to locate the desired amount of assets. It has also been applied in a medical imaging database (x ray, MRI) for approximate matching of image features and contents, and in electronic warfare for approximate matching of emitter signals (based on a set of parameter values) and also for locating the platforms that generate the signals via spatial relaxation.

With the recent advances in voice recognition systems, more and more systems will be providing voice input features. However, there are many ambiguities in the natural language. Further research in cooperative query answering techniques will be useful in assisting systems to understand users' dialogue with the system.

ACKNOWLEDGMENTS

The research and development of CoBase has been a team effort. I would like to acknowledge the past and present CoBase members—Hua Yang, Gladys Kong, X. Yang, Frank Meng, Guogen Zhang, Wesley Chuang, Meng-feng Tsai, Henrick Yau, and Gilles Fouques—for their contributions toward its design and implementation. The author also wishes to thank the reviewers for their valuable comments.

BIBLIOGRAPHY

1. T. Gaasterland, P. Godfrey, and J. Minker, An overview of cooperative answering, *J. Intell. Inf. Sys.*, **1**: 123–157, 1992.
2. A. Colmerauer and J. Pique, About natural logic, in H. Gallaire, et al. (eds.), *Proc. 5th ECAI*, Orsay, France, 1982, pp. 343–365.
3. S. J. Kaplan, Cooperative Responses from a portable natural language query system, *Artificial Intelligence*, **19**(2): 165–187, 1982.
4. E. Mays, Correcting misconceptions about database structure, *Proc. CSCSI 80*, 1980.
5. K. McCoy, Correcting object-related misconceptions, *Proc. COLING10*, Stanford, CA, 1984.
6. L. Cholvy and R. Demolombe, Querying a rule base, *Proc. 1st Int. Conf. Expert Database Syst.*, 1986, pp. 365–371.
7. T. Imielinski, Intelligent query answering in rule based systems, in J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Washington, DC: Morgan Kaufman, 1988.
8. A. Motro, Using integrity constraints to provide intensional responses to relational queries, *Proc. 15th Int. Conf. Very Large Data Bases*, Los Altos, CA, 1989, pp. 237–246.
9. A. Pirotte, D. Roelants, and E. Zimanyi, Controlled generation of intensional answers, *IEEE Trans. Knowl. Data Eng.*, **3**: 221–236, 1991.
10. U. Chakravarthy, J. Grant, and J. Minker, Logic based approach to semantic query optimization, *ACM Trans. Database Syst.*, **15**(2): 162–207, 1990.
11. C. Shum and R. Muntz, Implicit representation for extensional answers, in L. Kershberg (ed.), *Expert Database Systems*, Menlo Park, CA: Benjamin/Cummings, 1986, pp. 497–522.
12. W. W. Chu, R. C. Lee, and Q. Chen, Using type inference and induced rules to provide intensional answers, *Proc. IEEE Comput. Soc. 7th Int. Conf. Data Eng.*, Washington, DC, 1991, pp. 396–403.
13. A. Motro, Intensional answers to database queries, *IEEE Trans. Knowl. Database Eng.*, **6**(3): 1994, pp. 444–454.
14. F. Cuppens and R. Demolombe, How to recognize interesting topics to provide cooperative answering, *Inf. Syst.*, **14**(2): 163–173, 1989.
15. W. W. Chu and G. Zhang, Associative query answering via query feature similarity, *Int. Conf. Intell. Inf. Syst.*, pp. 405–501, Grand Bahama Island, Bahamas, 1997.
16. T. Gaasterland, J. Minker, and A. Rajesekar, Deductive database systems and knowledge base systems, *Proc. VIA 90*, Barcelona, Spain, 1990.
17. B. L. Webber and E. Mays, Varieties of user misconceptions: Detection and correction, *Proc. 8th Int. Conf. Artificial Intell.*, Karlsruhe, Germany, 1983, pp. 650–652.

18. W. Wahlster et al., Over-answering yes-no questions: Extended responses in a NL interface to a vision system, *Proc. IJCAI 1983*, Karlsruhe, West Germany, 1983.

19. A. K. Joshi, B. L. Webber, and R. M. Weischedel, Living up to expectations: Computing expert responses, *Proc. Natl. Conf. Artificial. Intell.*, Univ. Texas at Austin: The Amer. Assoc. Artif. Intell., 1984, pp. 169–175.

20. J. Allen, *Natural Language Understanding*, Menlo Park, CA: Benjamin/Cummings.

21. S. Carberry, Modeling the user's plans and goals, *Computational Linguistics*, **14**(3): 23–37, 1988.

22. K. F. McCoy, Reasoning on a highlighted user model to respond to misconceptions, *Computational Linguistics*, **14**(3): 52–63, 1988.

23. A. Quilici, M. G. Dyer, and M. Flowers, Recognizing and responding to plan-oriented misconceptions, *Computational Linguistics*, **14**(3): 38–51, 1988.

24. A. S. Hemerly, M. A. Casanova, and A. L. Furtado, Exploiting user models to avoid misconstruals, in R. Demolombe and T. Imielinski (eds.), *Nonstandard Queries and Nonstandard Answers*, Great Britain, Oxford Science, 1994, pp. 73–98.

25. A. Motro, FLEX: A tolerant and cooperative user interface to database, *IEEE Trans. Knowl. Data Eng.*, **4**: 231–246, 1990.

26. W. W. Chu, Q. Chen, and R. C. Lee, Cooperative query answering via type abstraction hierarchy, in S. M. Deen (ed.), *Cooperating Knowledge Based Systems*, Berlin: Springer-Verlag, 1991, pp. 271–292.

27. W. W. Chu and K. Chiang, Abstraction of high level concepts from numerical values in databases, *Proc. AAAI Workshop Knowl. Discovery Databases*, 1994.

28. W. W. Chu et al., An error-based conceptual clustering method for providing approximate query answers [online], *Commun. ACM, Virtual Extension Edition*, **39**(12): 216–230, 1996. Available: http://www.acm.org/cacm/extension.

29. M. Merzbacher and W. W. Chu, Pattern-based clustering for database attribute values, *Proc. AAAI Workshop on Knowl. Discovery*, Washington, DC, 1993.

30. W. W. Chu and Q. Chen, A structured approach for cooperative query answering, *IEEE Trans. Knowl. Data Eng.*, **6**: 738–749, 1994.

31. W. Chu et al., A scalable and extensible cooperative information system, *J. Intell. Inf. Syst.*, pp. 223–259, 1996.

32. T. Gaasterland, P. Godfrey, and J. Minker, Relaxation as a platform of cooperative answering, *J. Intell. Inf. Syst.*, **1**: 293–321, 1992.

33. G. Fouque, W. W. Chu, and H. Yau, A case-based reasoning approach for associative query answering, *Proc. 8th Int. Symp. Methodologies Intell. Syst.*, Charlotte, NC, 1994.

34. D. H. Fisher, Knowledge acquisition via incremental conceptual clustering, *Machine Learning*, **2**(2): 139–172, 1987.

35. M. A. Gluck and J. E. Corter, Information, uncertainty, and the unity of categories, *Proc. 7th Annu. Conf. Cognitive Sci. Soc.*, Irvine, CA, 1985, pp. 283–287.

36. Y. Cai, N. Cercone, and J. Han, Attribute-oriented induction in relational databases, in G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in Databases*, Menlo Park, CA: 1991.

37. J. R. Quinlan, The effect of noise on concept learning, in R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning*, volume **2**, 1986.

38. R. E. Stepp III and R. S. Michalski, Conceptual clustering: Inventing goal-oriented classifications of structured objects, in R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning*, 1986.

39. T. Finin et al., KQML as an agent communication language, *Proc. 3rd Int. Conf. Inf. Knowl. Manage.*, Gaithersburg, MD, 1994, pp. 456–463.

40. D. M. Mark and A. U. Frank, Concepts of space and spatial language, *Proc. 9th Int. Symp. Comput.-Assisted Cartography*, Baltimore, MD, 1989, pp. 538–556.

41. R. Subramanian and N. R. Adam, Ill-defined spatial operators in geographic databases: Their nature and query processing strategies, *Proc. ACM Workshop Advances Geographical Inf. Syst.*, Washington, DC, 1993, pp. 88–93.

42. A. S. Hemerly, A. L. Furtado, and M. A. Casanova, Towards cooperativeness in geographic databases, *Proc. 4th Int. Conf. Database Expert Syst. Appl.*, Prague, Czech Republic, 1993.

43. Y. Arens and C. Knoblock, Planning and reformulating queries for semantically-modelled multidatabase systems, *Proc. 1st Int. Conf. Inf. Knowl. Manage. (CIKM)*, Baltimore, MD, 1992, pp. 92–101.

44. D. P. McKay, J. Pastor, and T. W. Finin, View-concepts: Knowledge-based access to databases, *Proc. 1st Int. Conf. Inf. Knowl. Manage. (CIKM)*, Baltimore, MD, 1992, pp. 84–91.

45. J. Stillman and P. Bonissone, Developing new technologies for the ARPA-Rome Planning Initiative, *IEEE Expert*, **10**(1): 10–16, Feb. 1995.

46. W. W. Chu, I. T. Ieong, and R. K. Taira, A semantic modeling approach for image retrieval by content, *J. Very Large Database Syst.*, **3**: 445–477, 1994.

47. W. W. Chu, A. F. Cardenas, and R. K. Taira, KMeD: A knowledge-based multimedia medical distributed database system, *Inf. Syst.*, **20**(2): 75–96, 1995.

48. H. K. Huang and R. K. Taira, Infrastructure design of a picture archiving and communication system, *Amer. J. Roentgenol.*, **158**: 743–749, 1992.

49. C. Hsu, W. W. Chu, and R. K. Taira, A knowledge-based approach for retrieving images by content, *IEEE Trans. Knowl. Data Eng.*, **8**: 522–532, 1996.

WESLEY W. CHU
University of California
at Los Angeles
Los Angeles, California

CoXML: COOPERATIVE XML QUERY ANSWERING

INTRODUCTION

As the World Wide Web becomes a major means in disseminating and sharing information, there has been an exponential increase in the amount of data in web-compliant format such as HyperText Markup Language (HTML) and Extensible Markup Language (XML). XML is essentially a textual representation of the hierarchical (tree-like) data where a meaningful piece of data is bounded by matching starting and ending tags, such as `<name>` and `</name>`. As a result of the simplicity of XML as compared with SGML and the expressiveness of XML as compared with HTML, XML has become the most popular format for information representation and data exchange.

To cope with the tree-like structure in the XML model, many XML-specific query languages have been proposed (e.g., XPath[1] and XQuery (1)). All these query languages aim at the exact matching of query conditions. Answers are found when those XML documents match the given query condition *exactly*, which however, may not always be the case in the XML model. To remedy this condition, we propose a cooperative query answering framework that derives *approximate* answers by relaxing query conditions to *less* restricted forms. Query relaxation has been successfully used in relational databases (e.g., Refs. 2–6) and is important for the XML model because:

1. Unlike the relational model where users are given a relatively small-sized schema to ask queries, the schema in the XML model is substantially bigger and more complex. As a result, it is unrealistic for users to understand the full schema and to compose complex queries. Thus, it is desirable to relax the user's query when the original query yields null or not sufficient answers.
2. As the number of data sources available on the Web increases, it becomes more common to build systems where data are gathered from heterogeneous data sources. The structures of the participating data source may be different even though they use the same ontologies about the same contents. Therefore, the need to be able to query differently structured data sources becomes more important (e.g., (7,8)). Query relaxation allows a query to be structurally relaxed and routed to diverse data sources with different structures.

Query relaxation in the relational model focuses on value aspects. For example, for a relational query "*find a person with a salary range 50K–55K*," if there are no answers or insufficient results available, the query can be relaxed to "*find a person with a salary range 45K–60K*." In the XML model, in addition to the value relaxation, a new type of relaxation called *structure relaxation* is introduced, which relaxes the structure conditions in a query. Structure relaxation introduces new challenges to the query relaxation in the XML model.

FOUNDATION OF XML RELAXATION

XML Data Model

We model an XML document as an ordered, labeled tree in which each element is represented as a node and each element-to-subelement relationship is represented as an edge between the corresponding nodes. We represent each data node u as a triple (*id, label*, <*text*>), where *id* uniquely identifies the node, *label* is the name of the corresponding element or attribute, and *text* is the corresponding element's text content or attribute's value. *Text* is optional because not every element has a text content.

Figure 1 presents a sample XML data tree describing an article's information. Each circle represents a node with the node *id* inside the circle and *label* beside the circle. The text of each node is represented in italic at the leaf level.

Due to the hierarchical nature of the XML data model, we consider the text of a data node u as part of the text of any of u's ancestor nodes in the data tree. For example, in the sample XML data tree (Fig. 1), the node 8 is an ancestor of the node 9. Thus, the text of the node 9 (i.e., "*Algorithms for mining frequent itemsets*...") is considered part of the text of the node 8.

XML Query Model

A fundamental construct in most existing XML query languages is the tree-pattern query or *twig*, which selects elements or attributes with a tree-like structure. In this article, we use the twig as our basic query model. Similar to the tree representation of XML data, we model a query twig as a rooted tree. More specifically, a query twig T is a tuple (*root, V, E*), where

- *root* is the root node of the twig;
- V is the set of nodes in the twig, where each node is a tripe (*id, label*, <*cont*>), where *id* uniquely identifies the node, *label* is the name of the corresponding element or attribute, and *cont* is the content condition on the corresponding node. *cont* is optional because not every query node may have a content condition;
- The content condition for a query node is either a database-style value constraint (e.g., a Boolean condition such as equality, inequality, or range constraint) or an IR-style keyword search. An IR-style content condition consists of a set of terms, where

[*] This work is supported by NSF Award ITR#: 0219442

[1] See http://www.w3.org/TR/xpath/.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

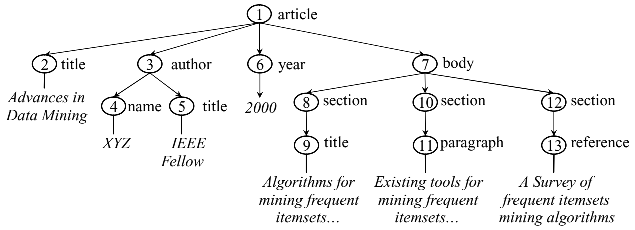


Figure 1. A sample XML data tree.

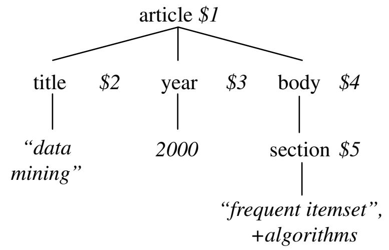


Figure 2. As sample XML twig.

each term is either a single word or a phrase. Each term may be prefixed with modifiers such as "+" or "−" for specifying preferences or rejections over the term. An IR-style content condition is to be processed in a nonBoolean style; and

- E is the set of edges in the twig. An edge from nodes $\$u$[2] to $\$v$, denoted as $e_{\$u,\$v}$, represents either a parent-to-child (i.e., "/") or an ancestor-to-descendant (i.e., "//") relationship between the nodes $\$u$ and $\$v$.

Given a twig T, we use $T.root$, $T.V$, and $T.E$ to represent its root, nodes, and edges, respectively. Given a node $\$v$ in the twig T (i.e., $v \in T.V$), we use $\$v.id$, $\$v.label$, and $\$v.cont$ to denote the unique ID, the name, and the content condition (if any) of the node respectively. The IDs of the nodes in a twig can be skipped when the labels of all the nodes are distinct.

For example, Fig. 2 illustrates a sample twig, which searches for articles with a title on "*data mining*," a year in 2000, and a body section about "*frequent itemset algorithms*." In this query, the user has a preference over the term *algorithm*. The twig consists of five nodes, where each node is associated with an unique *id* next to the node. The text under a twig node, shown in italic, is the content or value condition on the node.

The terms "twig" and "query tree" will be used interchangeably throughout this article.

XML Query Answer

With the introduction of XML data and query models, we shall now introduce the definition of an XML query answer. An answer for a query twig is a set of data nodes that satisfy the structure and content conditions in the twig. We formally define a query answer as follows:

Definition 1. ***Query Answer*** *Given an XML data tree D and a query twig T, an answer for the twig T, denoted as*$\mathcal{A}_D^T$, *is a set of nodes in the data D such that*:

- $\forall \$u \in T.V$, *there exists an unique data node u in* $\mathcal{A}_D^T$ *s.t* $\$u.label = u.label$. *Also, if* $\$u.cont \neq null$ *and* $\$u.cont$ *is a database-style value constraint, then the text of the data node* $u.text$ *satisfies the value constraint. If* $\$u:cont \neq null$ *and* $\$u.cont$ *is an IR-style content condition, then the text of* $u.text$ *should contain all the terms that are prefixed with "+" in* $\$u.cont$ *and must not contain any terms that are prefixed with "−" in* $\$u.cont$;
- $\forall e_{\$u,\$v} \in T.E$, *let u and v be the data nodes in* $\mathcal{A}_D^T$ *that correspond to the query node* $\$u$ *and* $\$v$, *respectively, then the structural relationship between u and v should satisfy the edge constraint* $e_{\$u,\$v}$.

For example, given the twig in Fig. 2, the set of nodes {1, 2, 6, 7, 8} in the sample XML data tree (Fig. 1) is an answer for the query, which matches the query nodes {\$1, \$2, \$3, \$4, \$5}, respectively. Similarly, the set of nodes {1, 2, 6, 7, 12} is also an answer to the sample query. Although the text of the data node 10 contain the phrase "*frequent itemset*," it does not contain the term *algorithm*, which is prefixed with "+." Thus, the set of data nodes {1, 2, 6, 7, 10} is not an answer for the twig.

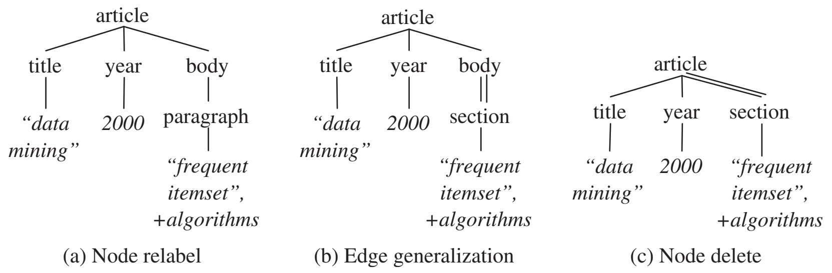
XML Query Relaxation Types

In the XML model, there are two types of query relaxations: *value relaxation* and *structure relaxation*. A value relaxation expands a value scope to allow the matching of additional answers. A structure relaxation, on the other hand, derives approximate answers by relaxing the constraint on a node or an edge in a twig. Value relaxation is orthogonal to structure relaxation. In this article, we focus on structure relaxation.

Many structure relaxation types have been proposed (8–10). We use the following three types, similar to the ones proposed in Ref. 10, which capture most of the relaxation types used in previous work.

- **Node Relabel.** With this relaxation type, a node can be relabeled to similar or equivalent labels according to domain knowledge. We use $rel(\$u, l)$ to represent a relaxation operation that renames a node $\$u$ to label l. For example, the twig in Fig. 2 can be relaxed to that in Fig. 3(a) by relabeling the node *section* to *paragraph*.
- **Edge Generalization.** With an edge relaxation, a parent-to-child edge ('/') in a twig can be generalized to an ancestor-to-descendant edge ('//'). We use $gen(e_{\$u,\$v})$ to represent a generalization of the edge between nodes $\$u$ and $\$v$. For example, the twig in Fig. 2 can be relaxed to that in Fig. 3(b) by relaxing the edge between nodes *body* and *section*.

[2] To distinguish a data node from a query node, we prefix the notation of a query node with a \$.



(a) Node relabel (b) Edge generalization (c) Node delete

Figure 3. Examples of structure relaxations for Fig.2.

- **Node Deletion.** With this relaxation type, a node may be deleted to derive approximate answers. We use *del*($*v*) to denote the deletion of a node $*v*. When $*v* is a leaf node, it can simply be removed. When $*v* is an internal node, the children of node $*v* will be connected to the parent of $*v* with ancestor-descendant edges ("//"). For instance, the twig in Fig. 2 can be relaxed to that in Fig. 3(c) by deleting the internal node *body*. As the root node in a twig is a special node representing the search context, we assume that any twig root cannot be deleted.

Given a twig T, a *relaxed twig* can be generated by applying one or more relaxation operations to T. Let m be the number of relaxation operations applicable to T, then there are at most $\binom{m}{1} + \ldots + \binom{m}{m} = 2^m$ relaxation operation combinations. Thus, there are at most 2^m relaxed twigs.

XML QUERY RELAXATION BASED ON SCHEMA CONVERSION

One approach to XML query relaxation is to convert XML schema, transform XML documents into relational tables with the converted schema, and then apply relational query relaxation techniques. A schema conversion tool, called XPRESS (*X*ml *P*rocessing and *R*elaxation in r*E*lational *S*torage *S*ystem) has been developed for these purposes: XML documents are mapped into relational formats so that queries can be processed and relaxed using existing relational technologies. Figure 4 illustrates the query relaxation flow via the schema conversion approach. This process first begins by extracting the schema information, such as DTD, from XML documents via tools such as XML Spy(see http://www.xmlspy.com.) Second, XML schema is transformed to relational schema via schema conversion ([e.g., XPRESS). Third, XML documents are parsed, mapped into tuples, and inserted into the relational databases. Then, relational query relaxation techniques [e.g., CoBase (3,6)] can be used to relax query conditions. Further, semi-structured queries over XML documents are translated into SQL queries. These SQL queries are processed and relaxed if there is no answer or there are insufficient answers available. Finally, results in the relational format are converted back into XML (e.g., the Nesting-based Translation Algorithm (NeT) and Constraints-based Translation Algorithm (CoT) (11) in XPRESS). The entire process can be done automatically and is transparent to users. In the following sections, we shall briefly describe the mapping between XML and relational schema.

Mapping XML Schema to Relational Schema

Transforming a hierarchical XML model to a flat relational model is a nontrivial task because of the following

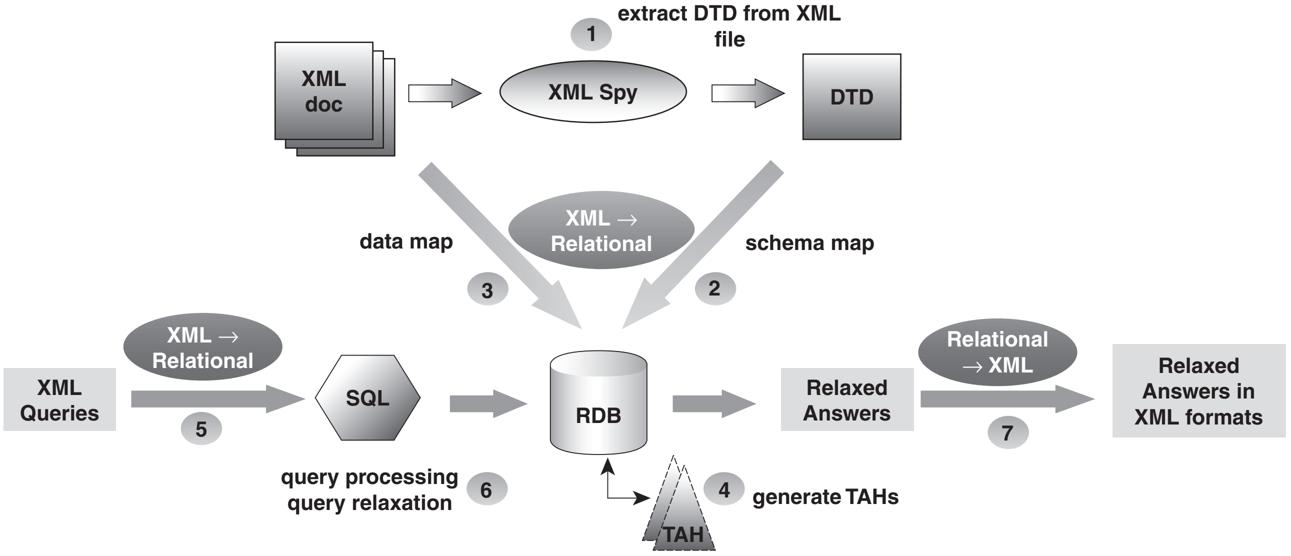


Figure 4. The processing flow of XML query relaxation via schema conversion.

inherent difficulties: the nontrivial 1-to-1 mapping, existence of set values, complicated recursion, and/or fragmentation issues. Several research works have been reported in these areas. Shanmugasundaram et al. (12) mainly focuses on the issues of structural conversion. The Constraints Preserving Inline (CPI) algorithm (13) considers the semantics existing in the original XML schema during the transformation. CPI inclines as many descendants of an element as possible into a single relation. It maps an XML element to a table when there is 1-to-$\{0,\ldots\}$ or 1-to-$\{1,\ldots\}$ cardinality between its parent and itself. The first cardinality has the semantics of "any," denoted by * in XML. The second means "at least," denoted by +. For example, consider the following DTD fragment:

```
<!ELEMENT author (name, address)>
<!ELEMENT name (firstname?, lastname)>
```

A naive algorithm will map every element into a separate table, leading to excessive fragmentation of the document, as follows:

```
author (address, name_id)
name (id, firstname, lastname)
```

The CPI algorithm converts the DTD fragment above into a single relational table as **author** (firstname, lastname, address).

In addition, semantics such as #REQUIRED in XML can be enforced in SQL with NOT NULL. Parent-to-child relationships are captured with KEYS in SQL to allow join operations. Figure 5 overviews the CPI algorithm, which uses a structure-based conversion algorithm (i.e., a hybrid algorithm) (13), as a basis and identifies various semantic constraints in the XML model. The CPI algorithm has been implemented in XPRESS, which reduces the number of tables generated while preserving most constraints.

Mapping Relational Schema to XML Schema

After obtaining the results in the relational format, we may need to represent them in the XML format before returning them back to users. XPRESS developed a Flat Translation (FT) algorithm (13), which translates tables in a relational schema to elements in an XML schema and columns in a relational schema to attributes in an XML schema. As FT translates the "flat" relational model to a "flat" XML model in a one-to-one manner, it does not use basic "non-flat" features provided by the XML model such as representing subelements though regular expression operator (e.g., "*" and "+"). As a result, the NeT algorithm (11) is proposed to decrease data redundancy and obtains a more intuitive schema by: (1) removing redundancies caused by multivalued dependencies; and (2) performing grouping on attributes. The NeT algorithm, however, considering tables one at a time, cannot obtain an overall picture of the relational schema where many tables are interconnected with each other through various other dependencies. The CoT algorithm (11) uses inclusion dependencies (INDs) of relational schema, such as foreign key constraints, to capture the interconnections between relational tables and represent them via parent-to-child hierarchical relationships in the XML model.

Query relaxation via schema transformation (e.g., XPRESS) has the advantage of leveraging on the well-developed relational databases and relational query relaxation techniques. Information, however, may be lost during the decomposition of hierarchical XML data into "flat" relational tables. For example, by transforming the following XML schema into the relational schema author (firstname, lastname, address), we lose the hierarchical relationship between element *author* and element *name*, as well as the information that element *firstname* is optional.

```
<!ELEMENT author (name, address)>
<!ELEMENT name (firstname?,lastname)>
```

Further, this approach does not support structure relaxations in the XML data model. To remedy these shortcomings, we shall perform query relaxation on the XML model directly, which will provide both value relaxation and structure relaxation.

A COOPERATIVE APPROACH FOR XML QUERY RELAXATION

Query relaxation is often user-specific. For a given query, different users may have different specifications about which conditions to relax and how to relax them. Most existing approaches on XML query relaxation (e.g., (10)) do not provide control during relaxation, which may yield undesired approximate answers. To provide user-specific approximate query answering, it is essential for an XML system to have a relaxation language that allows users to specify their relaxation control requirements and to have the capability to control the query relaxation process.

Furthermore, query relaxation usually returns a set of approximate answers. These answers should be ranked based on their relevancy to both the structure and the content conditions of the posed query. Most existing ranking models (e.g., (14,15)) only measure the content similarities between queries and answers, and thus are

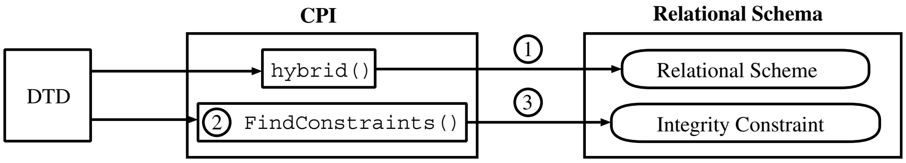


Figure 5. Overview of the CPI algorithm.

inadequate for ranking approximate answers that use structure relaxations. Recently, in Ref. (16), the authors proposed a family of structure scoring functions based on the occurrence frequencies of query structures among data without considering data semantics. Clearly, using the rich semantics provided in XML data in design scoring functions can improve ranking accuracy.

To remedy these shortcomings, we propose a new paradigm for XML approximate query answering that places users and their demands in the center of the design approach. Based on this paradigm, we develop a cooperative XML system that provides userspecific approximate query answering. More specifically, we first, develop a relaxation language that allows users to specify approximate conditions and control requirements in queries (e.g., preferred or unacceptable relaxations, nonrelaxable conditions, and relaxation orders).

Second, we introduce a relaxation index structure that clusters twigs into multilevel groups based on relaxation types and their distances. Thus, it enables the system to control the relaxation process based on users' specifications in queries.

Third, we propose a semantic-based tree editing distance to evaluate XML structure similarities, which is based on not only the number of operations but also the operation semantics. Furthermore, we combine structure and content similarities in evaluating the overall relevancy.

In Fig. 6, we present the architecture of our CoXML query answering system. The system contains two major parts: offline components for building relaxation indexes and online components for processing and relaxing queries and ranking results.

- *Building relaxation indexes.* The *Relaxation Index Builder* constructs relaxation indexes, XML Type Abstraction Hierarchy (XTAH), for a set of document collections.
- *Processing, relaxing queries, and ranking results.* When a user posts a query, the *Relaxation Engine* first sends the query to an *XML Database Engine* to search for answers that exactly match the structure conditions and approximately satisfy the content conditions in the query. If enough answers are found, the *Ranking Module* ranks the results based on their relevancy to the content conditions and returns the ranked results to the user. If there are no answers or insufficient results, then the *Relaxation Engine*, based on the user-specified relaxation constructs and controls, consults the relaxation indexes for the best relaxed query. The relaxed query is then resubmitted to the *XML Database Engine* to search for approximate answers. The *Ranking Module* ranks the returned approximate answers based on their relevancies to both structure and content conditions in the query. This process will be repeated until either there are enough approximate answers returned or the query is no longer relaxable.

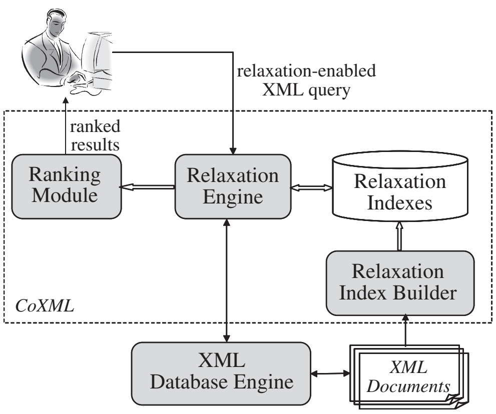


Figure 6. The CoXML system architecture.

The CoXML system can run on top of any existing XML database engine (e.g., BerkeleyDB[3], Tamino[4], DB2XML[5]) that retrieves exactly matched answers.

XML QUERY RELAXATION LANGUAGE

A number of XML approximate search languages have been proposed. Most extend standard query languages with constructs for approximate text search (e.g., XIRQL (15), TeXQuery (17), NEXI (18)). For example, TeXQuery extends XQuery with a rich set of full-text search primitives, such as proximity distances, stemming, and thesauri. NEXI introduces *about* functions for users to specify approximate content conditions. XXL (19) is a flexible XML search language with constructs for users to specify both approximate structure and content conditions. It, however, does not allow users to control the relaxation process. Users may often want to specify their preferred or rejected relaxations, nonrelaxable query conditions, or to control the relaxation orders among multiple relaxable conditions.

To remedy these shortcomings, we propose an XML relaxation language that allows users both to specify approximate conditions and to control the relaxation process. A relaxation-enabled query $\mathcal{Q}$ is a tuple ($\mathcal{T}$, $\mathcal{R}$, $\mathcal{C}$, $\mathcal{S}$), where:

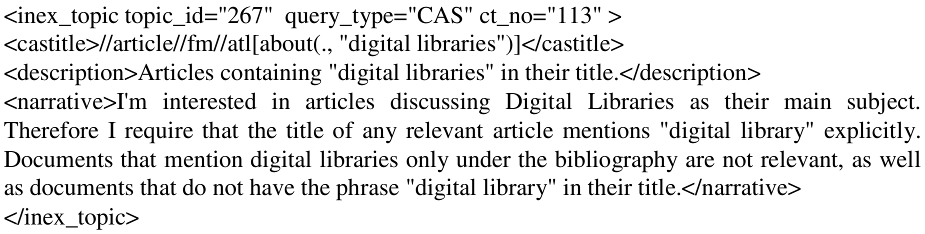
- $\mathcal{T}$ is a twig as described earlier;
- $\mathcal{R}$ is a set of relaxation constructs specifying which conditions in $\mathcal{T}$ may be approximated when needed;
- $\mathcal{C}$ is a boolean combination of relaxation control stating how the query shall be relaxed; and
- $\mathcal{S}$ is a stop condition indicating when to terminate the relaxation process.

The execution semantics for a relaxation-enabled query are as follows: We first search for answers that exactly match the query; we then test the stop condition to check whether relaxation is needed. If not, we repeatedly relax

[3] See http://www.sleepycat.com/

[4] See http://www.softwareag.com/tamino

[5] See http://www.ibm.com/software/data/db2/



```
<inex_topic topic_id="267"  query_type="CAS" ct_no="113" >
<castitle>//article//fm//atl[about(., "digital libraries")]</castitle>
<description>Articles containing "digital libraries" in their title.</description>
<narrative>I'm interested in articles discussing Digital Libraries as their main subject. Therefore I require that the title of any relevant article mentions "digital library" explicitly. Documents that mention digital libraries only under the bibliography are not relevant, as well as documents that do not have the phrase "digital library" in their title.</narrative>
</inex_topic>
```

Figure 8. Topic 267 in INEX 05.

the twig based on the relaxation constructs and control until either the stop condition is met or the twig cannot be further relaxed.

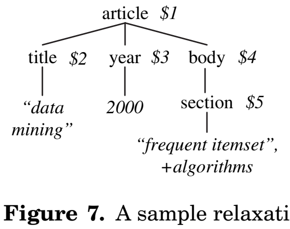
Given a relaxation-enabled query $\mathcal{Q}$, we use $\mathcal{Q}.\mathcal{T}$, $\mathcal{Q}.\mathcal{R}$, $\mathcal{Q}.\mathcal{C}$, and $\mathcal{Q}.\mathcal{S}$ to represent its twig, relaxation constructs, control, and stop condition, respectively. Note that a twig is required to specify a query, whereas relaxation constructs, control, and stop condition are optional. When only a twig is present, we iteratively relax the query based on similarity metrics until the query cannot be further relaxed.

A relaxation construct for a query $\mathcal{Q}$ is either a specific or a generic relaxation operation in any of the following forms:

- *rel*$(u,-)$, where $u \in \mathcal{Q}.\mathcal{T}.V$, specifies that node u may be relabeled when needed;
- *del*(u), where $u \in \mathcal{Q}.\mathcal{T}.V$, specifies that node u may be deleted if necessary; and
- *gen*$(e_{u,v})$, where $e_{u,v} \in \mathcal{Q}.\mathcal{T}.E$, specifies that edge $e_{u,v}$ may be generalized when needed.

The relaxation control for a query $\mathcal{Q}$ is a conjunction of any of the following forms:

- Nonrelaxable condition $!r$, where $r \in \{rel(u,-), del(u), gen\ (e_{u,v}) | u, v \in \mathcal{Q}.\mathcal{T}.V, e_{u,v} \in \mathcal{Q}, \mathcal{T}.E\}$, specifies that node u cannot be relabeled or deleted or edge $e_{u,v}$ cannot be generalized;
- $Pre\,fer(u, l_1, \ldots, ln)$, where $u \in \mathcal{Q}.\mathcal{T}.V$ and l_i is a label $(1 \leq \mathrm{i} \leq \mathrm{n})$, specifies that node u is preferred to be relabeled to the labels in the order of $(l_1, \ldots, l_n)$;
- $Reject(u, l_1, \ldots, l_n)$, where $u \in \mathcal{Q}.\mathcal{T}.V$, specifies a set of unacceptable labels for node u;
- $RelaxOrder(r_1, \ldots, r_n)$, where $r_i \in \mathcal{Q}.\mathcal{R}.\,(1 \leq \mathrm{i} \leq \mathrm{n})$, specifies the relaxation orders for the constructs in R to be $(r_1, \ldots, r_n)$; and
- $UseRType(rt_1, \ldots, rt_k)$, where $rt_i \in \{node\_relabel, node\_delete, edge\_generalize\} (1 \leq i \leq \mathrm{k} \leq 3)$, specifies the set of relaxation types allowed to be used. By default, all three relaxation types may be used.



$R = \{gen(e_{\$4,\$5}), del(\$3)\}$

$C = !del(\$4) \wedge !gen(e_{\$1,\$2}) \wedge !gen(e_{\$1,\$4}) \wedge$ UseRType(node_delete, edge_generalize)

$S = AtLeast(20)$

Figure 7. A sample relaxation-enabled query.

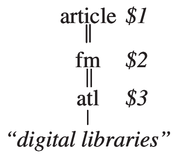
A stop condition $\mathcal{S}$ is either:

- $AtLeast(n)$, where n is a positive integer, specifies the minimum number of answers to be returned; or
- $\mathrm{d}(\mathcal{Q}.\mathcal{T}.T') \leq \tau$, where T' stands for a relaxed twig and τ a distance threshold, specifies that the relaxation should be terminated when the distance between the original twig and a relaxed twig exceeds the threshold.

Figure 7 presents a sample relaxation-enabled query. The minimum number of answers to be returned is 20. When relaxation is needed, the edge between *body* and *section* may be generalized and node *year* may be deleted. The relaxation control specifies that node *body* cannot be deleted during relaxation. For instance, a *section* about "*frequent itemset*" in an article's appendix part is irrelevant. Also, the edge between nodes *article* and *title* and the edge between nodes *article* and *body* cannot be generalized. For instance, an article with a reference to another article that possesses a title on "*data mining*" is irrelevant. Finally, only *edge generalization* and *node deletion* can be used.

We now present an example of using the relaxation language to represent query topics in INEX 05[6]. Figure 8 presents Topic 267 with three parts: *castitle* (i.e., the query formulated in an XPath-like syntax), *description*, and *narrative*. The *narrative* part describes a user's detailed information needs and is used for judging result relevancy.

The user considers an article's title (*atl*) non-relaxable and regards titles about "digital libraries" under the bibliography part (*bb*) irrelevant. Based on this narrative, we formulate this topic using the relaxation language as shown in Fig. 9. The query specifies that node *atl* cannot be relaxed (either deleted or relabeled) and node *fm* cannot be relabeled to *bb*.



$C = !rel(\$3, -) \wedge !del(\$3) \wedge Reject(\$2, bb)$

Figure 9. Relaxation specifications for Topic 267.

[6] Initiative for the evaluation of XML retrieval, See http://inex.is.informatik.uni-duisburg.de/

XML RELAXATION INDEX

Several approaches for relaxing XML or graph queries have been proposed (8,10,16,20,21). Most focus on efficient algorithms for deriving top-k approximate answers without relaxation control. For example, Amer-yahia et al. (16) proposed a DAG structure that organizes relaxed twigs based on their "consumption" relationships. Each node in a DAG represents a twig. There is an edge from twig T_A to twig T_B if the answers for T_B is a superset of those for T_A. Thus, the twig represented by an ancestor DAG node is always less relaxed and thus closer to the original twig than the twig represented by a descendant node. Therefore, the DAG structure enables efficient top-k searching when there are no relaxation specifications. When there are relaxation specifications, the approach in Ref. 16 can also be adapted to top-k searching by adding a postprocessing part that checks whether a relaxed query satisfies the specifications. Such an approach, however, may not be efficient when relaxed queries do not satisfy the relaxation specifications.

To remedy this condition, we propose an XML relaxation index structure, XTAH, that clusters relaxed twigs into multilevel groups based on relaxation types used by the twigs and distances between them. Each group consists of twigs using similar types of relaxations. Thus, XTAH enables a systematic relaxation control based on users' specifications in queries. For example, *Reject* can be implemented by pruning groups of twigs using unacceptable relaxations. *RelaxOrder* can be implemented by scheduling relaxed twigs from groups based on the specified order.

In the following, we first introduce XTAH and then present the algorithm for building an XTAH.

XML Type Abstraction Hierarchy—XTAH

Query relaxation is a process that enlarges the search scope for finding more answers. Enlarging a query scope can be accomplished by viewing the queried object at different conceptual levels.

In the relational database, a tree-like knowledge representation called Type Abstraction Hierarchy (TAH) (3) is introduced to provide systematic query relaxation guidance. A TAH is a hierarchical cluster that represents data objects at multiple levels of abstractions, where objects at higher levels are more general than objects at lower levels. For example, Fig. 10 presents a TAH for brain tumor sizes, in which a medium tumor size (i.e., 3–10 mm) is a more abstract representation than a specific tumor size (e.g., 10 mm). By such multilevel abstractions, a query can be relaxed by modifying its conditions via *generalization* (moving up the TAH) and *specialization* (moving down the TAH). In addition, relaxation can be easily controlled via TAH. For example, *REJECT* of a relaxation can be implemented by pruning the corresponding node from a TAH.

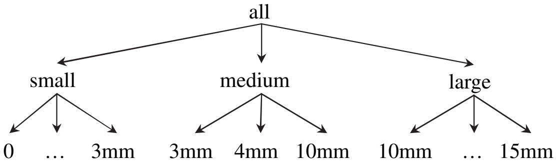


Figure 10. A TAH for brain tumor size.

To support query relaxation in the XML model, we propose a relaxation index structure similar to TAH, called XML Type Abstraction Hierarchy (XTAH). An XTAH for a twig structure T, denoted as XT_T , is a hierarchical cluster that represents relaxed twigs of T at different levels of relaxations based on the types of operations used by the twigs and the distances between them. More specifically, an XTAH is a multilevel labeled cluster with two types of nodes: internal and leaf nodes. A leaf node is a relaxed twig of T. An internal node represents a cluster of relaxed twigs that use similar operations and are closer to each other by distance. The label of an internal node is the common relaxation operations (or types) used by the twigs in the cluster. The higher level an internal node in the XTAH, the more general the label of the node, the less relaxed the twigs in the internal node.

XTAH provides several significant advantages: (1) We can efficiently relax a query based on relaxation constructs by fetching relaxed twigs from internal nodes whose labels satisfy the constructs; (2) we can relax a query at different granularities by traversing up and down an XTAH; and (3) we can control and schedule query relaxation based on users' relaxation control requirements. For example, relaxation control such as nonrelaxable conditions, Reject or UseRType, can be implemented by pruning XTAH internal nodes corresponding to unacceptable operations or types.

Figure 11 shows an XTAH for the sample twig in Fig. 3(a).[7] For ease of reference, we associate each node in the XTAH with a unique ID, where the IDs of internal nodes are prefixed with *I* and the IDs of leaf nodes are prefixed with *T*'.

Given a relaxation operation r, let I_r be an internal node with a label $\{r\}$. That is, I_r represents a cluster of relaxed twigs whose common relaxation operation is r. As a result of the tree-like organization of clusters, each relaxed twig belongs to only one cluster, whereas the twig may use multiple relaxation operations. Thus, it may be the case that not all the relaxed twigs that use the relaxation operation r are within the group I_r. For example, the relaxed twig T'_2, which uses two operations $gen(e_{\$1,\$2})$ and $gen(e_{\$4,\$5})$, is not included in the internal node that represents $\{gen(e_{\$4,\$5})\}$, I_7, because T'_2 may belong to either group I_4 or group I_7 but is closer to the twigs in group I_4.

To support efficient searching or pruning of relaxed twigs in an XTAH that uses an operation r, we add a virtual link from internal node I_r to internal node I_k, where I_k is not a descendant of I_r, but all the twigs within I_k use operation r. By doing so, relaxed twigs that use operation r are either within group I_r or within the groups connected to I_r by virtual links. For example, internal node I_7 is connected to internal nodes I_{16} and I_{35} via virtual links.

[7] Due to space limitations, we only show part of the XTAH here.

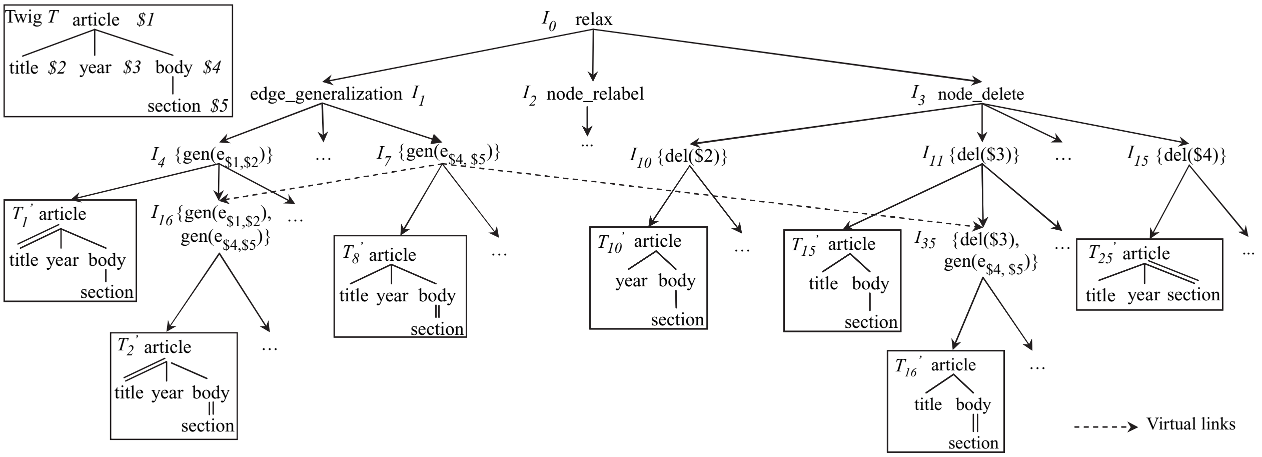


Figure 11. An example of XML relaxation index structure for the twig T.

Thus, all the relaxed twigs using the operation $gen(e_{\$4,\$5})$ are within the groups I_7, I_{16}, and I_{35}.

Building an XTAH

With the introduction of XTAH, we now present the algorithm for building the XTAH for a given twig T.

Algorithm 1 Building the XTAH for a given twig T

```
Input: T: a twig
    K: domain knowledge about similar node labels
Output: $XT_T$ : an XTAH for T
 1: $RO_T \leftarrow$ GerRelaxOperations(T, K) {GerRelaxOperations
    (T, K) returns a set of relaxation operations applicable to the
    twig T based on the domain knowledge K}
 2: let $XT_T$ be a rooted tree with four nodes: a root node relax
    with three child nodes node_relabel, node_delete and
    edge_generalization
 3: for each relaxation operation $r \in RO_T$ do
 4:     rtype ← the relaxation type of r
 5:     InsertXTNode(/relax/rtype, {r}) {InsertXTNode(p, n)
        inserts node n into $XT_T$ under path p}
 6:     $T' \leftarrow$ the relaxed twig using operation r
 7:     InsertXTNode (/relax/rtype, /{r}, T')
 8: end for
 9: for $k = 2$ to$|RO_T|$ do
10:     $S_k \leftarrow$ all possible combinations of k relaxation operations
        in $RO_T$
11:     for each combination $s \in S_k$ do
12:         let $s = \{r_1, \ldots, r_k\}$
13:         if the set of operations in s is applicable to T then
14:             $T' \leftarrow$ the relaxed twig using the operations in s
15:             $I_i \leftarrow$ the node representing $s - \{r_i\}(1 \le i \le k)$
16:             $I_j \leftarrow$ the node s.t. $\forall i, d(T', I_j) \le d(T', I_i)(1 \le \text{i,j} \le \text{k})$
17:             InsertXTNode(//$I_j$, $\{r_1, \ldots, r_k\}$)
18:             InsertXTNode(//$I_j$/$\{r_1, \ldots, r_k\}$, T')
19:             AddVLink(//$\{r_j\}$, //$I_j$) {AddV Link($p_1$,$p_2$) adds a
                virtual link from the node under path $p_1$ to the node
                under path $p_2$}
20:         end if
21:     end for
22: end for
```

In this subsection, we assume that a distance function is available that measures the structure similarity between twigs. Given any two twigs T_1 and T_2, we use $d(T_1, T_2)$ to represent the distance between the two twigs. Given a twig T and an XTAH internal node I, we measure the distance between the twig and the internal node, $d(T, I)$, as the average distance between T and any twig T′ covered by I.

Algorithm 1 presents the procedure of building the XTAH for twig T in a top-down fashion. The algorithm first generates all possible relaxations applicable to T (Line 1). Next, it initializes the XTAH with the top two level nodes (Line 2). In Lines 3–8, the algorithm generates relaxed twigs using one relaxation operation and builds indexes on these twigs based on the type of the relaxation used: For each relaxation operation r, it first adds a node to represent r, then inserts the node into the XTAH based on r's type, and places the relaxed twig using r under the node. In Lines 9–22, the algorithm generates relaxed twigs using two or more relaxations and builds indexes on these twigs. Let s be a set of k relaxation operations ($\text{k} \ge 2$), T′ a relaxed twig using the operations in s, and I an internal node representing s. Adding node I into the XTAH is a three-step process: (1) it first determines I's parent in the XTAH (Line 16). In principle, any internal node that uses a subset of the operations in s can be I's parent. The algorithm selects an internal node I_j to be I's parent if the distance between T' and I_j is less than the distance between T' and other parent node candidates; (2) It then connects node I to its parent I_j and adds a leaf node representing T' to node I (Lines 17 and 18). (3) Finally, it adds a virtual link from the internal node representing the relaxation operation r_j to node I (Line 19), where r_j is the operation that occurs in the label of I but not in label of its parent node I_j .

QUERY RELAXATION PROCESS

Query Relaxation Algorithm

Algorithm 2 Query Relaxation Process

```
Input: $XT_T$: an XTAH
       $\mathcal{Q} = \{\mathcal{T}, \mathcal{R}, \mathcal{C}, \mathcal{S}\}$: a relaxation-enabled query
Output: $\mathcal{A}$: a list of answers for the query $\mathcal{Q}$
 1: $\mathcal{A} \leftarrow$ SearchAnswer($\mathcal{Q}.T$); {Searching for exactly matched
    answers for $\mathcal{Q}.T$}
 2: if (the stop condition $\mathcal{Q}.\mathcal{S}$ is met) then
 3:     return $\mathcal{A}$
 4: end if
 5: if (the relaxation controls $\mathcal{Q}.\mathcal{C}$ are non-empty) then
 6:     PruneXTAH($XT_T$, $\mathcal{Q}.\mathcal{C}$) {Pruning nodes in $XT_T$ that
        contain relaxed twigs using unacceptable relaxation
        operations based on $\mathcal{Q}.\mathcal{C}$}
 7: end if
 8: if the relaxation constructs $\mathcal{Q}.\mathcal{R}$ are non-empty then
 9:     while ($\mathcal{Q}.\mathcal{S}$ is not met)&&(not all the constructs in $\mathcal{Q}.\mathcal{R}$ have
        been processed) do
10:         $T' \leftarrow$ the relaxed twig from $XT_T$ that best satisfies the
            relaxation specifications $\mathcal{Q}.\mathcal{R}$ & $\mathcal{Q}.\mathcal{C}$
11:         Insert SearchAnswer($T'$) into $\mathcal{A}$
12:     end while
13: end if
14: while ($\mathcal{Q}.\mathcal{T}$ is relaxable)&&($\mathcal{Q}.\mathcal{S}$ is not met) do
15:     $T' \leftarrow$ the relaxed twig from $XT_T$ that is closest to $\mathcal{Q}.\mathcal{T}$ based
        on distance
16:     Insert SearchAnswer($T'$) into $\mathcal{A}$
17: end while
18: return $\mathcal{A}$
```

Figure 12 presents the control flow of a relaxation process based on XTAH and relaxation specifications in a query. The *Relaxation Control* module prunes irrelevant XTAH groups corresponding to unacceptable relaxation operations or types and schedules relaxation operations based on *Prefer* and *RelaxOrder* as specified in the query. Algorithm 2 presents the detailed steps of the relaxation process:

1. Given a relaxation-enabled query $\mathcal{Q} = \{\mathcal{T}, \mathcal{R}, \mathcal{C}, \mathcal{S}\}$ and an XTAH for $\mathcal{Q}.\mathcal{T}$, the algorithm first searches for exactly matched answers. If there are enough number of answers available, there is no need for relaxation and the answers are returned (Lines 1–4).
2. If relaxation is needed, based on the relaxation control $\mathcal{Q}.\mathcal{C}$ (Lines 5–7), the algorithm prunes XTAH internal nodes that correspond to unacceptable operations such as nonrelaxable twig nodes (or edges), unacceptable node relabels, and rejected relaxation types. This step can be efficiently carried out by using internal node labels and virtual links. For example, the relaxation control in the sample query (Figure 7) specifies that only *node_delete* and *edge_generalization* may be used. Thus, any XTAH node that uses *node_relabel*, either within group I_2 or connected to I_2 by virtual links, is disqualified from searching. Similarly, the internal nodes I_{15} and I_4, representing the operations $del(\$4)$ and $gen(e_{\$1,\ \$2})$, respectively, are pruned from the XTAH by the *Relaxation Control* module.
3. After pruning disqualified internal groups, based on relaxation constructs and control, such as *RelaxOrder* and *Prefer*, the *Relaxation Control* module schedules and searches for the relaxed query that best satisfies users' specifications from the XTAH. This step terminates when either the stop condition is met or all the constructs have been processed. For example, the sample query contains two relaxation constructs: $gen(e_{\$4,\$5})$ and $del(\$3)$. Thus, this step selects the best relaxed query from internal groups, I_7 and I_{11}, representing the two constructs, respectively.
4. If further relaxation is needed, the algorithm then iteratively searches for the relaxed query that is closest to the original query by distance, which may use relaxation operations in addition to those specified in the query. This process terminates when either the stop condition holds or the query cannot be further relaxed.
5. Finally, the algorithm outputs approximate answers.

Searching for Relaxed Queries in an XTAH

We shall now discuss how to efficiently search for the best relaxed twig that has the least distance to the query twig from its XTAH in Algorithm 2.

A brute-force approach is to select the best twig by checking all the relaxed twigs at the leaf level. For a twig T with m relaxation operations, the number of relaxed twigs can be up to 2^m. Thus, the worst case time complexity for this approach is $O(2^m)$, which is expensive.

To remedy this condition, we propose to assign representatives to internal nodes, where a representative summarizes the distance characteristics of all the relaxed twigs covered by a node. The representatives facilitate the searching for the best relaxed twig by traversing an

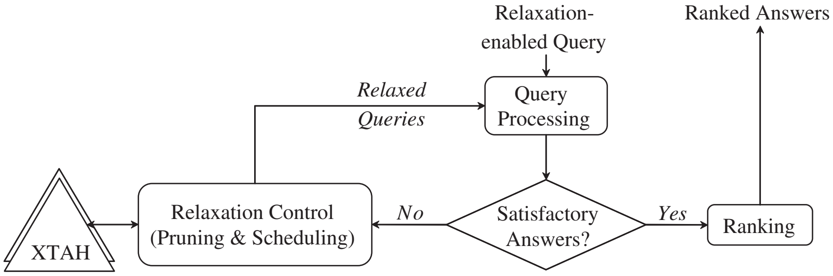


Figure 12. Query relaxation control flow.

XTAH in a top-down fashion, where the path is determined by the distance properties of the representatives. By doing so, the worst case time complexity of finding the best relaxed query is $O(d * h)$, where d is the maximal degree of an XTAH node and h is the height of the XTAH. Given an XTAH for a twig T with m relaxation operations, the maximal degree of any XTAH node and the depth of the XTAH are both $O(m)$. Thus, the time complexity of the approach is $O(m^2)$, which is far more efficient than the brute-force approach ($O(2^m)$).

In this article, we use M-tree (22) for assigning representatives to XTAH internal nodes. M-tree provides an efficient access method for similarity search in the "metric space," where object similarities are defined by a distance function. Given a tree organization of data objects where all the data objects are at the leaf level, M-tree assigns a data object covered by an internal node I to be the representative object of I. Each representative object stores the covering radius of the internal node (i.e., the maximal distance between the representative object and any data object covered by the internal node). These covering radii are then used in determining the path to a data object at the leaf level that is closest to a query object during similarity searches.

XML RANKING

Query relaxation usually generates a set of approximate answers, which need to be ranked before being returned to users. A query contains both structure and content conditions. Thus, we shall rank an approximate answer based on its relevancy to both the structure and content conditions of the posed query. In this section, we first present how to compute XML content similarity, then describe how to measure XML structure relevancy, and finally discuss how to combine structure relevancy with content similarity to produce the overall XML ranking.

XML Content Similarity

Given an answer $\mathcal{A}$ and a query $\mathcal{Q}$, the content similarity between the answer and the query, denoted as cont_sim(A and $\mathcal{Q}$), is the sum of the content similarities between the data nodes and their corresponding matched query nodes. That is,

$$cont\_sim(\mathcal{A}, \mathcal{Q}) = \sum_{v \in \mathcal{A}, \$u \in \mathcal{Q}.\mathcal{T}.v, u \; matches \; \$u} cont\_sim(v, \$u) \tag{1}$$

For example, given the sample twig in Fig. 2, the set of nodes {1, 2, 6, 7, 8} in the sample data tree is an answer. The content similarity between the answer and the twig equals to $cont\_sim(2, \$2) + cont\_sim(6, \$3) + cont\_sim(8, \$5)$.

We now present how to evaluate the content similarity between a data node and a query node. Ranking models in traditional IR evaluate the content similarity between a document to a query and thus need to be extended to evaluating the content similarity between an XML data node and a query node. Therefore, we proposed an extended vector space model (14) for measuring XML content similarity, which is based on two concepts: *weighted term frequency* and *inverse element frequency*.

Weighted Term Frequency. Due to the hierarchical structure of the XML data model, the text of a node is also considered as a part of the ancestor nodes' text, which introduces the challenge of how to calculate the content relevancy of an XML data node v to a query term t, where t could occur in the text of any node nested within the node v. For example, all three *section* nodes (i.e., nodes 8, 10, and 12) in the XML data tree (Fig. 1) contain the phrase "frequent itemsets" in their text parts. The phrase "frequent itemsets" occurs at the *title* part of the node 8, the *paragraph* part in the node 10, and the *reference* part in the node 12. The same term occurring at the different text parts of a node may be of different weights. For example, a "frequent itemset" in the *title* part of a *section* node has a higher weight than a "frequent itemset" in the *paragraph* part of a *section* node, which, in turn, is more important than a "frequent itemset" in the *reference* part of a *section* node. As a result, it may be inaccurate to measure the weight of a term t in the text of a data node v by simply counting the occurrence frequency of the term t in the text of the node v without distinguishing the term's occurrence paths within the node v.

To remedy this condition, we introduce the concept of "weighted term frequency," which assigns the weight of a term t in a data node v based on the term's occurrence frequency and the weight of the *occurrence path*. Given a data node v and a term t, let $p = v_1.v_2 \ldots .v_k$ be an *occurrence path* for the term t in the node v, where v_k is a descendant node of v, v_k directly contains the term t, and $v \rightarrow v_1 \rightarrow \ldots \rightarrow v_k$ represents the path from the node v to the node v_k. Let $w(p)$ and $w(v_i)$ denote the weight for the path p and the node v_i, respectively. Intuitively, the weight of the path $p = v_1.v_2 \ldots .v_k$ is a function of the weights of the nodes on the path (i.e., $w(p) = \mathrm{f}(w(v_1), \ldots w(v_k))$), with the following two properties:

1. $\mathrm{f}(w(v_1), w(v_2), \ldots, w(v_\mathrm{k}))$ is a monotonically increasing function with respect to $w(v_i)$ ($1 \leq \mathrm{i} \leq \mathrm{k}$); and
2. $\mathrm{f}(w(v_1), w(v_2), \ldots, w(v_\mathrm{k}))) = 0$ if any $w(v_i) = 0$ ($1 \leq \mathrm{i} \leq \mathrm{k}$).

The first property states that the path weight function is a monotonically increasing function. That is, the weight of a path is increasing if the weight of any node on the path is increasing. The second property states that if the weight of any node on the path is zero, then the weight of the path is zero. For any node v_i ($1 \leq \mathrm{i} \leq \mathrm{k}$) on the path p, if the weight of the node v_i is zero, then it implies that users are not interested in the terms occurring under the node v_i. Therefore, any term in the text of either the node v_i or a descendant node of v_i is irrelevant.

A simple implementation of the path weight function $\mathrm{f}(w(v_1), w(v_2), \ldots, w(v_k))$ that satisfies the properties stated above is to let the weight of a path equal to the

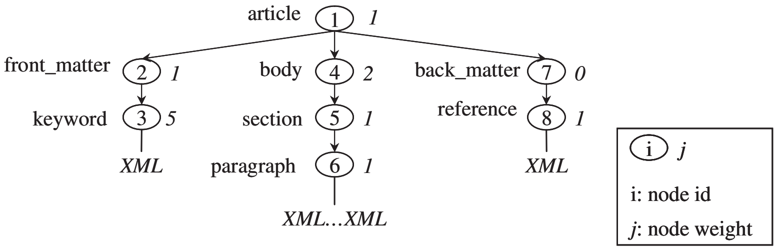


Figure 13. An example of weighted term frequency.

product of the weights of all nodes on the path:

$$w(p) = \prod_{i=1}^{k} w(v_i) \tag{2}$$

With the introduction of the weight of a path, we shall now define the weighted term frequency for a term t in a data node v, denoted as $tf_w(v,\, t)$, as follows:

$$tf_w(v,t) = \sum_{j=1}^{m} w(p_j) \times tf(v,p_j,t) \tag{3}$$

where m is the number of paths in the data node v containing the term t and $tf\ (v,\, p_j,\, t)$ is the frequency of the term t occurred in the node v via the path p_j.

For example, Fig. 13 illustrates an example of an XML data tree with the weight for each node shown in italic beside the node. The weight for the *keyword* node is 5 (i.e., $w(keyword) = 5$). From Equation (2), we have $w(front\_matter.keyword)$ = 5*1 = 5, $w(body.section.paragraph)$ = 2*1*1 = 2, and $w(back\_matter.reference)$ = 0*1 = 0, respectively. The frequencies of the term "XML" in the paths *front_matter.keyword*, *body.section.paragraph*, and *back_ matter.reference* are 1, 2, and 1, respectively. Therefore, from Equation (3), the weighted term frequency for the term "XML" in the data node *article* is 5*1 + 2*2 + 0*1 = 9.

Inverse Element Frequency. Terms with different popularity in XML data have different degrees of discriminative power. It is well known that a term frequency (*tf*) needs to be adjusted by the inverse document frequency (*idf*) (23). A very popular term (with a small *idf*) is less discriminative than a rare term (with a large *idf*). Therefore, the second component in our content ranking model is the concept of "inverse element frequencys" (*ief*), which distinguishes terms with different discriminative powers in XML data. Given a query Q and a term t, let \$$u$ be the node in the twig $Q.\mathcal{T}$ whose content condition contains the term t (i.e., $t \in \$u.cont$). Let DN be the set of data nodes such that each node in DN matches the structure condition related with the query node \$$u$. Intuitively, the more frequent the term t occurs in the text of the data nodes in DN, the less discriminative power the term t has. Thus, the inverse element frequency for the query term t can be measured as follows:

$$ief(\$u,\, t) = log\left(\frac{N_1}{N_2} + 1\right) \tag{4}$$

where N_1 denotes the number of nodes in the set DN and N_2 represents the number of the nodes in the set DN that contain the term t in their text parts.

For example, given the sample XML data tree (Fig. 1) and the query twig (Fig. 2), the inverse element frequency for the term "frequent itemset" can be calculated as follows: First, the content condition of the query node \$5 contains the term "frequent itemset"; second, there are three data nodes (i.e., nodes 8, 10, and 12) that match the query node \$5; and third, all the three nodes contain the term in their text. Therefore, the inverse element frequency for the term "frequent itemset" is log(3/3 + 1) = log2. Similarly, as only two nodes (i.e., nodes 8 and 12) contain the term "algorithms," the inverse element frequency for the term "algorithms" is log(3/2 + 1) = log(5/2).

Extended Vector Space Model. With the introduction "weighted term frequency" and "inverse element frequency," we now first present how we compute the content similarity between a data node and a query node and then present how we calculate the content similarity between an answer and a query.

Given a query node \$$u$ and a data node v, where the node v matches the structure condition related with the query node \$$u$, the content similarity between the nodes v and \$$u$ can be measured as follows:

$$cont_sim(v,\$u) = \sum_{t \in \$u.cont} w(m(t)) \times tf_w(v,t) \times ief(\$u,t) \tag{5}$$

where t is a term in the content condition of the node \$$u$, $m(t)$ stands for the modifier prefixed with the term t (e.g., "+", " ", "−"), and $w(m(t))$ is the weight for the term modifier as specified by users.

For example, given the *section* node, \$5, in the sample twig (Fig. 2), the data node 8 in Fig. 1 is a match for the twig node \$5. Suppose that the weight for a "+" term modifier is 2 and the weight for the *title* node is 5, respectively. The content similarity between the data node 8 and the twig node \$5 equals to tf_w(8, *"frequent itemset"*)× ief(\$5, *"frequent itemset"*) + w('+') × tf_w(8, *"algorithms"*) × *ief*(\$5, *"algorithms"*), which is 5 × log2 + 2 × 5 × log(5/2) = 18.22. Similarly, the data node 2 is a match for the twig node *title* (i.e., \$2) and the content similarity between them is tf_w(2, *"data mining"*) × ief(\$2, "data mining") = 1.

Discussions. The extended vector space model has shown to be very effective in ranking content similarities of SCAS retrieval results[8](14). SCAS retrieval results are usually of relatively similar sizes. For example, for the twig in Fig. 2, suppose that the node *section* is the target node (i.e., whose matches are to be returned as answers). All the SCAS retrieval results for the twig will be sections inside article bodies. Results that approximately match the twig, however, could be nodes other than *section* nodes, such as *paragraph*, *body*, or *article* nodes, which are of varying sizes. Thus, to apply the extended vector space model for evaluating content similarities of approximate answers under this condition, we introduce the factor of "weighted sizes" into the model for normalizing the biased effects caused by the varying sizes in the approximate answers (24):

$$cont\_sim(\mathcal{A}, \mathcal{Q}) = \sum_{v \in \mathcal{A}, \$u \in \mathcal{Q}.\mathcal{T}.\mathcal{V}, v\, matches\, \$u} \frac{cont\_sim(v, \$u)}{log_2\, wsize(v)} \tag{6}$$

where *wsize*(v) denotes the weighted size of a data node v.

Given an XML data node v, *wsize*(v) is the sum of the number of terms directly contained in node v's text, *size*-($v.text$), and the weighted size of all its child nodes adjusted by their corresponding weights, as shown in the following equation.

$$wsize(v) = size(v.text) + \sum_{v_i\ s.t.\ v|v_i} wsize(v_i) * w(v_i) \tag{7}$$

For example, the weighted size of the *paragraph* node equals the number of terms in its text part, because the *paragraph* node does not have any child node.

Our normalization approach is similar to the scoring formula proposed in Ref. 25, which uses the log of a document size to adjust the product of tf and *idf*.

Semantic-based Structure Distance

The structure similarity between two twigs can be measured using tree editing distance (e.g., (26)), which is frequently used for evaluating tree-to-tree similarities. Thus, we measure the structure distance between an answer $\mathcal{A}$ and a query $\mathcal{Q}$, *struct_dist*($\mathcal{A}$, $\mathcal{Q}$), as the editing distance between the twig $\mathcal{Q}\cdot\mathcal{T}$ and the least relaxed twig T', $d(\mathcal{Q}.\mathcal{T}, T')$, which is the total costs of operations that relax $\mathcal{Q}.\mathcal{T}$ to T':

$$struct\_dist(\mathcal{A}, \mathcal{Q}) = d(\mathcal{Q}.\mathcal{T}, T') = \sum_{i=1}^{k} cost(r_i) \tag{8}$$

where $\{r_1, \ldots, r_k\}$ is the set of operations that relaxes $\mathcal{Q}.\mathcal{T}$ to T' and $cost(r_i)$ $(0 \leq cost(r_i) \leq 1)$ is equal to the cost of the relaxation operation $r_i(1 \leq \mathrm{i} \leq \mathrm{k})$.

Existing edit distance algorithms do not consider operation cost. Assigning equal cost to each operation is simple, but does not distinguish the semantics of different operations. To remedy this condition, we propose a semantic-based relaxation operation cost model.

We shall first present how we model the semantics of XML nodes. Given an XML dataset D, we represent each data node v_i as a vector $\{w_{i1}, w_{i2}, \ldots, w_{iN}\}$, where N is the total number of distinct terms in D and w_{ij} is the weight of the *jth* term in the text of v_i. The weight of a term may be computed using *tf***idf* (23) by considering each node as a "document." With this representation, the similarity between two nodes can be computed by the cosine of their corresponding vectors. The greater the cosine of the two vectors, the semantically closer the two nodes.

We now present how to model the cost of an operation based on the semantics of the nodes affected by the operation with regard to a twig T as follows:

- Node Relabel – *rel*(u, l)
  A node relabel operation, *rel*(u, l), changes the label of a node u from $u.label$ to a new label l. The more semantically similar the two labels are, the less the relabel operation will cost. The similarity between two labels, $u.label$ and l, denoted as *sim*($u.label$, l), can be measured as the cosine of their corresponding vector representations in XML data. Thus, the cost of a relabel operation is:

$$cost(rel(u, l)) = 1 - sim(u.lable,\ l) \tag{9}$$

  For example, using the INEX 05 data, the cosine of the vector representing *section* nodes and the vector representing *paragraph* nodes is 0.99, whereas the cosine of the vector for *section* nodes and the vector for *figure* nodes is 0.38. Thus, it is more expensive to relabel node *section* to *paragraph* than to *figure*.

- Node Deletion – *del*(u)
  Deleting a node u from the twig approximates u to its parent node in the twig, say v. The more semantically similar node u is to its parent node v, the less the deletion will cost. Let $\mathrm{V}_{v/u}$ and V_v be the two vectors representing the data nodes satisfying v/u and v, respectively. The similarity between $v\,/\,u$ and v, denoted as *sim*($v\,/\,u$, v), can be measured as the cosine of the two vectors $V_{v\,/\,u}$ and V_v. Thus, a node deletion cost is:

$$cost(del(u)) = 1 - sim(v/u,\ u) \tag{10}$$

  For example, using the INEX 05 data, the cosine of the vector for *section* nodes inside *body* nodes and the vector for *body* nodes is 0.99, whereas the cosine of the vector for *keyword* nodes inside *article* nodes and the vector for *article* nodes is 0.2714. Thus, deleting the *keyword* node in Fig. 3(a) costs more than deleting the *section* node.

- Edge Generalization – *gen*($e_{v,u}$)
  Generalizing the edge between nodes $\$v$ and $\$u$ approximates a child node $v\,/\,u$ to a descendant node $v\,/\,/\,u$. The closer $v\,/\,u$ is to $v\,/\,/\,u$ in semantics, the less the edge generalization will cost. Let $V_{v\,/\,u}$ and $V_{v\,/\,/\,u}$ be two vectors representing the data nodes satisfying

[8] In a SCAS retrieval task, structure conditions must be matched exactly whereas content conditions are to be approximately matched.

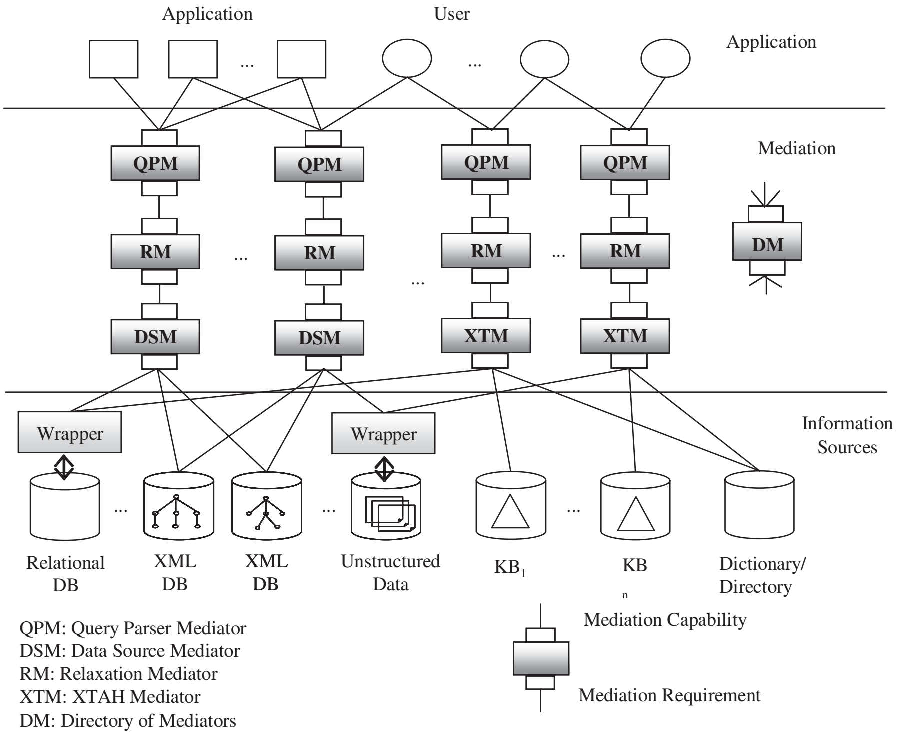


Figure 14. A scalable and extensible cooperative XML query answering system.

v/u and $v//u$, respectively. The similarity between v/u and $v//u$, denoted as $sim(v/u, v//u)$, can be measured as the cosine of the two vectors $V_{v/u}$ and $V_{v//u}$. Thus, the cost for an edge generalization can be measured as:

$$cost(gen(e_{v,u})) = 1 - sim(v/u, v//u) \qquad (11)$$

For example, relaxing *article/title* in Fig. 3(a) to *article//title* makes the title of an article's author (i.e., */article/author/title*) an approximate match. As the similarity between an article's title and an author's title is low, the cost of generalizing *article/title* to *article//title* may be high.

Note that our cost model differs from Amer-Yahia et al. (16) in that Amer-Yahia et al. (16) applies *idf* to twig structures without considering node semantics, whereas we applied *tf*idf* to nodes with regard to their corresponding data content.

The Overall Relevancy Ranking Model

We now discuss how to combine structure distance and content similarity for evaluating the overall relevancy.

Given a query $\mathcal{Q}$, the relevancy of an answer $\mathcal{A}$ to the query $\mathcal{Q}$, denoted as $sim(\mathcal{A}, \mathcal{Q})$, is a function of two factors: the structure distance between $\mathcal{A}$ and $\mathcal{Q}$ (i.e., $struct\_dist(\mathcal{A}, \mathcal{Q})$), and the content similarity between $\mathcal{A}$ and $\mathcal{Q}$, denoted as $cont\_sim(\mathcal{A}, \mathcal{Q})$. We use our extended vector space model for measuring content similarity (14). Intuitively, the larger the structure distance, the less the relevancy; the larger the content similarity, the greater the relevancy. When the structure distance is zero (i.e., exact structure match), the relevancy of the answer to the query should be determined by their content similarity only. Thus, we combine the two factors in a way similar to the one used in XRank (27) for combining element rank with distance:

$$sim(\mathcal{A}, \mathcal{Q}) = \alpha^{struct\_dist(\mathcal{A},\mathcal{Q})} * cost\_sim(\mathcal{A}, \mathcal{Q}) \qquad (12)$$

where α is a constant between 0 and 1.

A SCALABLE AND EXTENSIBLE ARCHITECTURE

Figure 14 illustrates a mediator architecture framework for a cooperative XML system. The architecture consists of an application layer, a mediation layer, and an information source layer. The information source layer includes a set of heterogeneous data sources (e.g., relational databases, XML databases, and unstructured data), knowledge bases, and knowledge base dictionaries or directories. The knowledge base dictionary (or directory) stores the characteristics of all the knowledge bases, including XTAH and domain knowledge in the system. Non-XML data can be converted into the XML format by wrappers. The mediation layer consists of data source mediators, query parser mediators, relaxation mediators, XTAH mediators, and directory mediators. These mediators are selectively interconnected to meet the specific application requirements. When the demand for certain mediators increases, additional copies of the mediators can be added to reduce the loading. The mediator architecture allows incremental growth with application, and thus the system is *scalable*. Further,

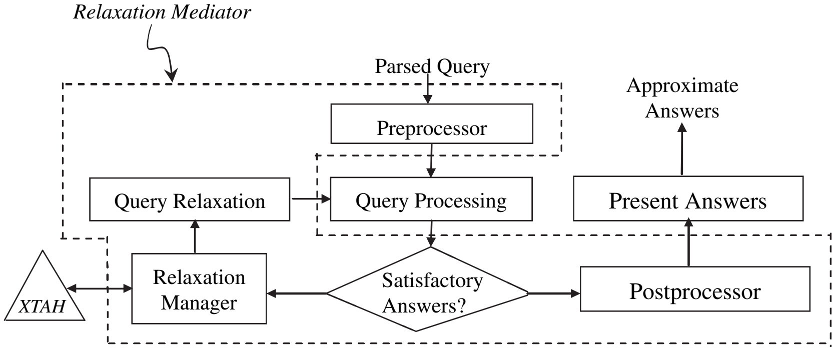


Figure 16. The flow chart of XML query relaxation processing.

different types of mediators can be interconnected and can communicate with each other via a common communication protocol (e.g., KQML (28), FIPA[9]) to perform a joint task. Thus, the architecture is *extensible*.

For query relaxation, based on the set of frequently used query tree structures, the XTAHs for each query tree structure can be generated accordingly. During the query relaxation process, the XTAH manager selects the appropriate XTAH for relaxation. If there is no XTAH available, the system generates the corresponding XTAH on-the-fly.

We shall now describe the functionalities of various mediators as follows:

- **Data Source Mediator (DSM)**
  The data source mediator provides a virtual database interface to query different data sources that usually have different schema. The data source mediator maintains the characteristics of the underlying data sources and provides a unified description of these data sources. As a result, XML data can be accessed from data sources without knowing the differences of the underlying data sources.
- **Query Parser Mediator (PM)**
  The query parser mediator parses the queries from the application layer and transforms the queries into query representation objects.
- **Relaxation Mediator (RM)**
  Figure 15 illustrates the functional components of the relaxation mediator, which consists of a pre-processor, a relaxation manager, and a post-processor. The flow of the relaxation process is depicted in Fig. 16. When a relaxation-enabled query is presented to the relaxation mediator, the system first goes through a pre-processing phase. During pre-processing, the system transforms the relaxation constructs into standard XML query constructs. All relaxation control operations specified in the query are processed and forwarded to the relaxation manager and are ready for use if the query requires relaxation. The modified query is then presented to the underlying databases for execution. If no answers are returned, then the relaxation manager relaxes the query conditions guided by the relaxation index (XTAH). We repeat the relaxation process until either the stop condition is met or the query is no longer relaxable. Finally, the returned answers are forwarded to the post-processing module for ranking.
- **XTAH Mediator (XTM)**
  The XTAH mediator provides three conceptually separate, yet interlinked functions to peer mediators: XTAH Directory, the XTAH Management, and the XTAH Editing facilities, as illustrated in Fig. 17.

  Usually, a system contains a large number of XTAHs. To allow other mediators to determine which XTAHs exist within the system and their characteristics, the XTAH mediator contains a directory. This directory is searchable by the XML query tree structures.

  The XTAH management facility provides client mediators with traversal functions and data extraction functions (for reading the information out of XTAH nodes). These capabilities present a common interface so that peer mediators can traverse and extract data from an XTAH. Further, the XTAH

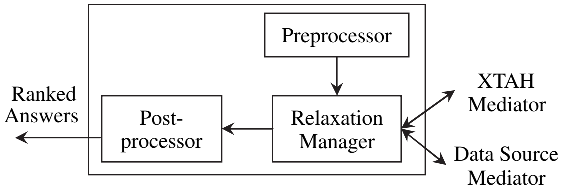


Figure 15. The relaxation mediator.

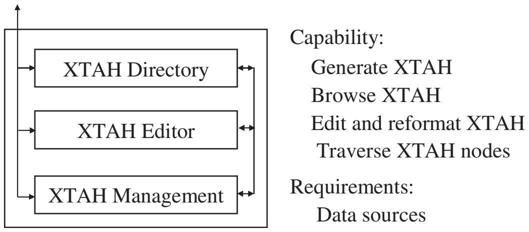


Figure 17. The XTAH mediator.

[9] See http://www.fipa.org.

mediator has an editor that allows users to edit XTAHs to suit their specific needs. The editor handles recalculation of all information contained within XTAH nodes during the editing process and supports exportation and importation of entire XTAHs if a peer mediator wishes to modify it.

- **Directory Mediator (DM)**
  The directory mediator provides the locations, characteristics, and functionalities of all the mediators in the system and is used by peer mediators for locating a mediator to perform a specific function.

A COOPERATIVE XML (CoXML) QUERY ANSWERING TESTBED

A CoXML query answering testbed has been developed at UCLA to evaluate the effectiveness of XML query relaxation through XTAH. Figure 18 illustrates the architecture of CoXML testbed, which consists of a query parser, a preprocessor, a relaxation manager, a database manager, an XTAH manager, an XTAH builder, and a post-processor. We describe the functionality provided by each module as follows:

- *XTAH Builder*. Given a set of XML documents and the domain knowledge, the *XTAH builder* constructs a set of XTAHs that summarizes the structure characteristics of the data.
- *Query Parser*. The *query parser* checks the syntax of the query. If the syntax is correct, then it extracts information from the parsed query and creates a query representation object.
- *Preprocessor*. The pre-processor transforms relaxation constructs (if any) in the query into the standard XML query constructs.
- *Relaxation Manager*. The relaxation manager performs the following services: (1) building a relaxation structure based on the specified relaxation constructs and controls; (2) obtaining the relaxed query conditions from the XTAH Manager; (3) modifying the query accordingly; and (4) retrieving the exactly matched answers.
- *Database Manager*. The database manager interacts with an XML database engine and returns exactly matched answers for a standard XML query.
- *XTAH Manager*. Based on the structure of the query tree, the XTAH manager selects an appropriate XTAH to guide the query relaxation process.
- *Post-processor*. The post-processor takes unsorted answers as input, ranks them based on both structure and content similarities, and outputs a ranked list of results.

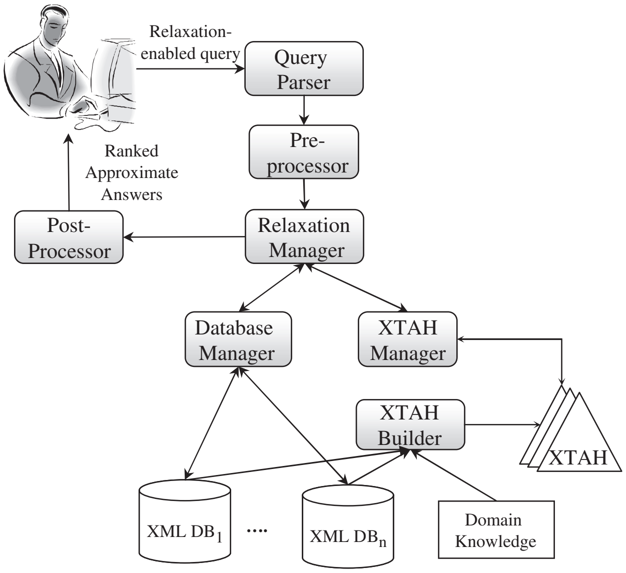


Figure 18. The architecture of the CoXML testbed.

EVALUATION OF XML QUERY RELAXATION

INEX is a DELOS working group[10] that aims to provide a means for evaluating XML retrieval systems in the form of a large heterogeneous XML test collection and appropriate scoring methods. The INEX test collection is a large set of scientific articles, represented in XML format, from publications of the IEEE Computer Society covering a range of computer science topics. The collection, approximately 500 megabytes, contains over 12,000 articles from 18 magazines/transactions from the period of 1995 to 2004, where an article (on average) consists of 1500 XML nodes. Different magazines/transactions have different data organizations, although they use the same ontology for representing similar content.

There are three types of queries in the INEX query sets: content-only (CO), strict content and structure (SCAS), and vague content and structure (VCAS). CO queries are traditional information retrieval (IR) queries that are written in natural language and constrain the content of the desired results. Content and structure queries not only restrict content of interest but also contain either explicit or implicit references to the XML structure. The difference between a SCAS and a VCAS query is that the structure conditions in a SCAS query must be interpreted exactly whereas the structure conditions in a VCAS query may be interpreted loosely.

To evaluate the relaxation quality of the CoXML system, we perform the VCAS retrieval runs on the CoXML testbed and compare the results against the INEX's relevance assessments for the VCAS task, which can be viewed as the "gold standard." The evaluaion studies reveal the expressiveness of the relaxation language and the effectiveness of using XTAH in providing user-desired relaxation control. The evaluation results demonstrate that our content similarity model has significantly high precision at low recall regions. The model achieves the highest average precision as compared with all the 38 official submissions in

[10] See http://www.iei.pi.cnr.it/DELOS

INEX 03 (14). Furthermore, the evaluation results also demonstrate that using the semantic-based distance function yields results with greater relevancy than using the uniform-cost distance function. Comparing with other systems in INEX 05, our user-centeric relaxation approach retrieves approximate answers with greater relevancy (29).

SUMMARY

Approximate matching of query conditions plays an important role in XML query answering. There are two approaches to XML query relaxation: either through schema conversion or directly through the XML model. Converting the XML model to the relational model by schema conversion can leverage on the mature relational model techniques, but information may be lost during such conversions. Furthermore, this approach does not support XML structure relaxation. Relaxation via the XML model approach remedies these shortcomings. In this article, a new paradigm for XML approximate query answering is proposed that places users and their demands at the center of the design approach. Based on this paradigm, we develop an XML system that cooperates with users to provide user-specific approximate query answering. More specifically, a relaxation language is introduced that allows users to specify approximate conditions and relaxation control requirements in a posed query. We also develop a relaxation index structure, XTAH, that clusters relaxed twigs into multilevel groups based on relaxation types and their interdistances. XTAH enables the system to provide user-desired relaxation control as specified in the query. Furthermore, a ranking model is introduced that combines both content and structure similarities in evaluating the overall relevancy of approximate answers returned from query relaxation. Finally, a mediatorbased CoXML architecture is presented. The evaluation results using the INEX test collection reveal the effectiveness of our proposed user-centric XML relaxation methodology for providing user-specific relaxation.

ACKNOWLEDGMENTS

The research and development of CoXML has been a team effort. We would like to acknowledge our CoXML members, Tony Lee, Eric Sung, Anna Putnam, Christian Cardenas, Joseph Chen, and Ruzan Shahinian, for their contributions in implementation, testing, and performance evaluation.

BIBLIOGRAPHY

1. S. Boag, D. Chamberlin, M. F Fernandez, D. Florescu, J. Robie, and J. S. (eds.), XQuery 1.0: An XML Query Language. Available http://www.w3.org/TR/xquery/.
2. W. W Chu ,Q. Chen, and A. Huang, Query Answering via Cooperative Data Inference. *J. Intelligent Information Systems (JIIS)*, **3** (1): 57–87, 1994.
3. W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson, CoBase: A scalable and extensible cooperative information system. *J. Intell. Inform. Syst.*, **6** (11), 1996.
4. S. Chaudhuri and L. Gravano, Evaluating Top-k Selection Queries. In *Proceedings of 25th International Conference on Very Large Data Bases*, September 7–10, 1999, Edinburgh, Scotland, UK.
5. T. Gaasterland, Cooperative answering through controlled query relaxation, *IEEE Expert*, **12** (5): 48–59, 1997.
6. W.W. Chu, Cooperative Information Systems, in B. Wah (ed.), *The Encyclopedia of Computer Science and Engineering*, New York: Wiley, 2007.
7. Y. Kanza, W. Nutt, and Y. Sagiv, Queries with Incomplete Answers Over Semistructured Data. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, May 31 – June 2, 1999, Philadelphia, Pennsylvania.
8. Y. Kanza and Y. Sagiv, In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, May 21–23, 2001, Santa Barbara, California.
9. T. Schlieder, In *Proceedings of 10th International Conference on Extending Database Technology*, March 26–31, 2006, Munich, Germany.
10. S. Amer-Yahia, S. Cho, and D. Srivastava, XML Tree Pattern Relaxation. In *Proceedings of 10th International Conference on Extending Database Technology*, March 26–31, 2006, Munich, Germany.
11. D. Lee, M. Mani, and W. W Chu, Schema Conversions Methods between XML and Relational Models, *Knowledge Transformation for the Semantic Web. Frontiers in Artificial Intelligence and Applications* Vol. 95, IOS Press, 2003, pp. 1–17.
12. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases*, September 7–10, 1999, Edinburgh, Scotland, UK.
13. D. Lee and W.W Chu, CPI: Constraints-preserving Inlining algorithm for mapping XML DTD to relational schema, *J. Data and Knowledge Engineering, Special Issue on Conceptual Modeling*, **39** (1): 3–25, 2001.
14. S. Liu, Q. Zou, and W. Chu, Configurable Indexing and Ranking for XML Information Retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, July 25–29, 2004, Sheffield, UK.
15. N. Fuhr and K. Groβjohann, XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* September 9–13, 2001, New Orleans Louisiana.
16. S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman, Structure and Content Scoring for XML. In *Proceedings of the 31st International Conference on Very Large Data Bases*, August 30– September 2, 2005, Trondheim, Norway.
17. S. Amer-Yahia, C. Botev, and J. Shanmugasundaram, TeXQuery: A Full-Text Search Extension to XQuery. In *Proceedings of 13th International World Wide Web Conference.* May 17–22, 2004, New York.
18. A. Trotman and B. Sigurbjornsson, Narrowed Extended XPath I NEXI. In *Proceedings of the 3rd Initiative of the Evaluation of XML Retrieval (INEX 2004) Workshop*, December 6–8, 2004, Schloss Dagstuhl, Germany,
19. A. Theobald and G. Weikum, Adding Relevance to XML. In *Proceedings of the 3rd International Workshop on the Web and*

Databases, WebDB 2000, Adam's, May 18–19, 2000, Dallas, Texas.

20. A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava, Adaptive Processing of Top-k Queries in XML. *In Proceedings of the 21st International Conference on Data Engineering, ICDE 2005*, April 5–8, 2005, Tokyo, Japan.
21. I. Manolescu, D. Florescu, and D. Kossmann, Answering XML Queries on Heterogeneous Data Sources. *In Proceedings of 27th International Conference on Very Large Data Bases*, September 11–14, 2001, Rome, Italy.
22. P. Ciaccia, M. Patella, and P. Zezula, M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. *In Proceedings of 23rd International Conference on Very Large Data Bases*, August 25–29, 1997, Athens, Greece.
23. G. Salton and M. J McGill, *Introduction to Modern Information Retrieval*, New York: McGraw-Hill, 1983.
24. S. Liu, W. Chu, and R. Shahinian, Vague Content and Structure Retrieval(VCAS) for Document-Centric XML Retrieval. *Proceedings of the 8th International Workshop on the Web and Databases (WebDB 2005)*, June 16–17, 2005, Baltimore, Maryland.
25. W. B Frakes and R. Baeza-Yates, *Information Retreival: Data Structures and Algorithms*, Englewood Cliffs, N.J.: Prentice Hall, 1992.
26. K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM J. Comput.*, **18** (6):1245– 1262, 1989.
27. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, XRANK: Ranked Keyword Search Over XML Document. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, June 9–12, 2003, San Diego, California.
28. T. Finin, D. McKay, R. Fritzson, and R. McEntire, KQML: An information and knowledge exchange protocol, in K. Fuchi and T. Yokoi, (eds), *Knowledge Building and Knowledge Sharing*, Ohmsha and IOS Press, 1994.
29. S. Liu and W. W Chu, CoXML: A Cooperative XML Query Answering System. Technical Report # 060014, Computer Science Department, UCLA, 2006.
30. T. Schlieder and H. Meuss, Querying and ranking XML documents, *J. Amer. So. Inf. Sci. Technol.*, **53** (6):489.
31. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB*, 1999.

Wesley W. Chu
Shaorong Liu
University of California,
Los Angeles
Los Angeles, California

DATA ANALYSIS

What is data analysis? Nolan (1) gives a definition, which is a way of making sense of the patterns that are in, or can be imposed on, sets of figures. In concrete terms, data analysis consists of an observation and an investigation of the given data, and the derivation of characteristics from the data. Such characteristics, or features as they are sometimes called, contribute to the insight of the nature of data. Mathematically, the features can be regarded as some variables, and the data are modeled as a realization of these variables with some appropriate sets of values. In traditional data analysis (2), the values of the variables are usually numerical and may be transformed into symbolic representation. There are two general types of variables, discrete and continuous. Discrete variables vary in units, such as the number of words in a document or the population in a region. In contrast, continuous variables can vary in less than a unit to a certain degree. The stock price and the height of people are examples of this type. The suitable method for collecting values of discrete variables is counting, and for continuous ones, it is measurement.

The task of data analysis is required among various application fields, such as agriculture, biology, economics, government, industry, medicine, military, psychology, and science. The source data provided for different purposes may be in various forms, such as text, image, or waveform. There are several basic types of purposes for data analysis:

1. Obtain the implicit structure of data.
2. Derive the classification or clustering of data.
3. Search particular objects in data.

For example, the stockbroker would like to get the future trend of the stock price, the biologist needs to divide the animals into taxonomies, and the physician tries to find the related symptoms of a given disease. The techniques to accomplish these purposes are generally drawn from statistics that provide well-defined mathematical models and probability laws. In addition, some theories, such as fuzzy-set theory, are also useful for data analysis in particular applications. This article is an attempt to give a brief introduction of these techniques and concepts of data analysis. In the following section, a variety of fundamental data analysis methods are introduced and illustrated by examples. In the second section, the methods for data analysis on two types of Internet data are presented. Advanced methods for Internet data analysis are discussed in the third section. At last, we give a summary of this article and highlight the research trends.

FUNDAMENTAL DATA ANALYSIS METHODS

In data analysis, the goals are to find significant patterns in the data and apply this knowledge to some applications. Data analysis is generally performed in the following stages:

1. Feature selection.
2. Data classification or clustering.
3. Conclusion evaluation.

The first stage consists of the selection of the features in the data according to some criteria. For instance, features of people may include their height, skin color, and fingerprints. Considering the effectiveness of human recognition, the fingerprint, which is the least ambiguous, may get the highest priority in the feature selection. In the second stage, the data are classified according to the selected features. If the data consist of at least two features, e.g., the height and the weight of people, which can be plotted in a suitable coordinate system, we can inspect so-called scatter plots and detect clusters or contours for data grouping. Furthermore, we can investigate ways to express data similarity. In the final stage, the conclusions drawn from the data would be compared with the actual demands. A set of mathematical models has been developed for this evaluation. In the following, we first divide the methods of data analysis into two categories according to different initial conditions and resultant uses. Then, we introduce two famous models for data analysis. Each method will be discussed first, followed by examples. Because the feature selection depends on the actual representations of data, we postpone the discussion about this stage until the next section. In this section, we focus on the classification/clustering procedure based on the given features.

A Categorization of Data Analysis Methods

There are a variety of ways to categorize the methods of data analysis. According to the initial conditions and the resultant uses, there can be two categories, supervised data analysis and unsupervised data analysis. The term *supervised* means that the human knowledge has to be provided for the process. In supervised data analysis, we specify a set of classes called a *classification template* and select some samples from the data for each class. These samples are then labeled by the names of the associated classes. Based on this initial condition, we can automatically classify the other data termed as *to-be-classified data*. In *unsupervised* data analysis, there is no classification template, and the resultant classes depend on the samples. The following are descriptions of supervised and unsupervised data analysis with an emphasis on their differences.

Supervised Data Analysis. The classification template and the well-chosen samples are given as an initial state and contribute to the high accuracy of data classification. Consider the K nearest-neighbor (K *NNs*) classifier, which is a typical example of supervised data analysis. The input

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

to the classifier includes a set of labeled samples S, a constant value K, and a to-be-classified datum X. The output after the classification is a label denoting a class to which X belongs. The classification procedure is as follows:

1. Find the K NNs of X from S.
2. Choose the dominant classes by K NNs.
3. If only one dominant class exists, label X by this class; otherwise, label X by any dominant class.
4. Add X to S and the process terminates.

The first step selects K samples from S such that the values of the selected features (also called patterns) of these K samples are closest to those of X. Such a similarity may be expressed in a variety of ways. The measurement of distances among the patterns is one of the suitable instruments, for example, the Euclidean distance as shown in Equation (1). Suppose the K samples belong to a set of classes; the second step is to find the set of dominant classes C’. A dominant class is a class that contains the majority of the K samples. If there is only one element in C’, say class C_i, we assign X to C_i. On the other hand, if C’ contains more than one element, X is assigned to an arbitrary class in C’. After deciding the class of X, we label it and add it into the set S.

$$\delta(X, Y) = \sqrt{\sum_{k=1}^{m} (X_k - Y_k)^2} \tag{1}$$

where each datum is represented by m features.

Example. Suppose there is a dataset about the salaries and ages of people. Table 1 gives such a set of samples S and the corresponding labels. There are three labels that denote three classes: Rich, Fair, and Poor. These classes are determined based on the assumption that the Richness depends on the values of the salary and age. In Table 1, we also append the rules of assigning labels for each age value. From the above, we can get the set membership of each class:

$$C_{\text{Rich}} = \{Y_1, Y_4, Y_8\}, \; C_{\text{Fair}} = \{Y_{2,} Y_5, Y_6, Y_{10}\}, \\ C_{\text{Poor}} = \{Y_3, Y_7, Y_9\}$$

Table 1. A Set of Samples with the Salary and Age Data

Sample	Age	Salary	Label	Assumed Rules to Assign Labels
Y_1	20	25k	Rich	Rich, >20k; Poor, <10k
Y_2	22	15k	Fair	Rich, >26k; Poor, <13k
Y_3	24	15k	Poor	Rich, >35k; Poor, <16k
Y_4	24	40k	Rich	
Y_5	28	25k	Fair	Rich, >44k; Poor, <22k
Y_6	30	40k	Fair	Rich, > 50k; Poor, <25k
Y_7	30	20k	Poor	
Y_8	32	60k	Rich	Rich, >56k; Poor, <28k
Y_9	36	30k	Poor	Rich, >68k; Poor, <34k
Y_{10}	40	70k	Fair	Rich, >80k; Poor, <40k
X	26	35k	Fair	Rich, >38k; Poor, <19k

If there is a to-be-classified datum X with age 26 and salary \$35,000 (35k), we apply the classification procedure to classify it. Here we let the value of K be 4 and use the Euclidean distance as the similarity measure:

1. The set of 4 NNs is $\{Y_4, Y_5, Y_6, Y_9\}$.
2. The dominant class is the class C_{Fair} because Y_6, $Y_5 \in C_{\text{Fair}}$, $Y_4 \in C_{\text{Rich}}$, and $Y_9 \in C_{\text{Poor}}$.
3. Label X by C_{Fair}.
4. New S contains an updated class $C_{\text{Fair}} = \{Y_2, Y_5, Y_6, Y_{10}, X\}$.

We can also give an assumed rule to decide the corresponding label for the age of X as shown in Table 1. Obviously, the conclusion drawn from the above classification coincides with such an assumption from human knowledge.

Unsupervised Data Analysis. Under some circumstances, data analysis consists of a partition of the whole data set into many subsets. Moreover, the data within each subset have to be similar to a high degree, whereas the data between different subsets have to be similar to a very low degree. Such subsets are called clusters, and the way to find a good partition is sometimes also called cluster analysis. A variety of methods have been developed to handle this problem. A common characteristic among them is the iterative nature of the algorithms.

The c-mean clustering algorithm is representative in this field. The input contains the sample set S and a given value c, which denotes the number of clusters in the final partition. Notice that no labels are assigned to the samples in S in advance. Before clustering, we must give an initial partition W_0 with c clusters. The algorithm terminates when it converges to a stable situation in which the current partition remains the same as the previous one. Different initial partitions can lead to different final results. One way to get the best partition is to apply this algorithm with all different W_0’s. To simplify the illustration, we only consider a given W_0 and a fixed c. The clustering procedure is as follows:

1. Let W be W_0 on S.
2. Compute the mean of each cluster in W.
3. Evaluate the nearest mean of each sample and move a sample if its current cluster is not the one corresponding to its nearest mean.
4. If any movement occurs, go to step 2; otherwise, the process terminates.

The first step sets the current partition W to be W_0. Then we compute a set of means M in W. In general, a mean is a virtual sample representing the whole cluster. It is straightforward to use averaging as the way to find M.

Next, we measure the similarity between each sample in S and every mean in M. Suppose a sample Y_j belongs to a cluster C_i in the previous partition W, whereas another cluster C_k has a mean nearest to Y_j. Then we move Y_j from C_i to C_k. Finally, if such a sample movement exists, the partition W would become a new one and requires more iteration. On the other hand, if no such movement occurs during iteration, the partition would become stable and the final clustering is produced.

Example. Consider the data in Table 1 again. Suppose there is no label on each sample and only the salary and the age data are used as the features for analysis. For clarity, we use a pair of values on the two features to represent a sample; for instance, the pair (20, 25k) refers to the sample Y_1. Suppose there is an initial partition containing two clusters C_1 and C_2. Let the means of these clusters be M_1 and M_2, respectively. The following shows the iterations for the clustering:

1. For the initial partition $W : C_1 = \{Y_1, Y_2, Y_3, Y_4, Y_5\}$, $C_2 = \{Y_6, Y_7, Y_8, Y_9, Y_{10}\}$.
   a. The first iteration: $M_1 = (23.6,\ 26\text{k})$, $M_2 = (33.6,\ 44\text{k})$.
   b. Move Y_4 from C_1 to C_2, move Y_7 and Y_9 from C_2 to C_1.
2. For the new partition $W : C_1 = \{Y_1, Y_2, Y_3, Y_5, Y_7, Y_9\}$, $C_2 = \{Y_4, Y_6, Y_8, Y_{10}\}$.
   a. The second iteration: $M_1 = (26.6,\ 21.6\text{k})$, $M_2 = (31.5,\ 52.5\text{k})$.
   b. There is no sample movement; the process terminates.

We can easily find a simple discriminant rule behind this final partition. All the samples with salaries lower than 40k belong to C_1, and the others belong to C_2. Hence we may conclude with a discriminant rule that divides S into two clusters by checking the salary data. If we use another initial partition, say W', where C_1 is $\{Y_1, Y_3, Y_5, Y_7, Y_9\}$ and C_2 is $\{Y_2, Y_4, Y_6, Y_8, Y_{10}\}$, the conclusion is the same. The following process yields another partition with three clusters.

1. For the initial partition $W : C_1 = \{Y_1, Y_4, Y_7\}$, $C_2 = \{Y_2, Y_5, Y_8\}$, $C_3 = \{Y_3, Y_6, Y_9, Y_{10}\}$.
   a. The first iteration: $M_1 = (24.6,\ 28.3\text{k})$, $M_2 = (27.3,\ 33.3\text{k})$, $M_3 = (32.5,\ 38.7\text{k})$.
   b. Move Y_4 from C_1 to C_2, move Y_2 and Y_5 from C_2 to C_1, move Y_8 from C_2 to C_3, move Y_3 from C_3 to C_1, move Y_9 from C_3 to C_2.
2. For the new partition $W : C_1 = \{Y_1, Y_2, Y_3, Y_5, Y_7\}$, $C_2 = \{Y_4, Y_9\}$, $C_3 = \{Y_6, Y_8, Y_{10}\}$.
   a. The second iteration: 2 $M_1 = (24.8,\ 20\text{k})$, $M_2 = (30,\ 35\text{k})$, $\mathrm{M}_3 = (34,\ 56.6\text{k})$.
   b. Move Y_6 from C_3 to C_2.
3. For the new partition $W : C_1 = \{Y_1, Y_2, Y_3, Y_5, Y_7\}$, $C_2 = \{Y_4, Y_6, Y_9\}$, $C_3 = \{Y_8, Y_{10}\}$.
   a. The third iteration: $M_1 = (24.8,\ 20\text{k})$, $M_2 = (30,\ 36.6\text{k})$, $M_3 = (36,\ 65\text{k})$.
   b. There is no sample movement; the process terminates.

After three iterations, we have a stable partition and conclude with a discriminant rule that all samples with salaries lower than 30k belong to C_1, the other samples with salaries lower than 60k belong to C_2 and the remainder belongs to C_3. The total number of iterations depends on the initial partition, the number of clusters, the given features, and the similarity measure.

Methods for Data Analysis

In the following, we introduce two famous methods for data analysis. One is Bayesian data analysis based on probability theory, and the other is fuzzy data analysis based on fuzzy-set theory.

Bayesian Data Analysis. Bayesian inference, as defined in Ref. 3, is the process of fitting a probability model to a set of samples, which results in a probability distribution to make predictions for to-be-classified data. In this environment, a set of samples is given in advance and labeled by their associated classes. Observing the patterns contained in these samples, we can obtain not only the distributions of samples for the classes but also the distributions of samples for the patterns. Therefore, we can compute a distribution of classes for these patterns and use this distribution to predict the classes of the to-be-classified data based on their patterns. A typical process of Bayesian data analysis contains the following stages:

1. Compute the distributions from the set of labeled samples.
2. Derive the distribution of classes for the patterns.
3. Evaluate the effectiveness of these distributions.

Suppose a sample containing the pattern a on some features is labeled class C_i. First, we compute a set of probabilities $P(C_i)$ that denotes a distribution of samples for different classes and let each $P(a|C_i)$ denote the conditional probability of a sample containing the pattern a, given that the sample belongs to the class C_i. In the second stage, the conditional probability of a sample belonging to the class C_i, given that the sample contains the pattern a, can be formulated as follows:

$$P(C_i|a) = \frac{P(a|C_i) \times P(C_i)}{P(a)}, \text{ where}$$
$$P(a) = \sum_j P(a|C_j) \times P(C_j) \qquad (2)$$

From Equation (2), we can derive the probabilities of a sample belonging to classes according to the patterns contained in the sample. Finally, we can find a way to determine the class by using these probabilities. The following is a simple illustration of data analysis based on this technique.

Example. Consider the data in Table 1. We first gather the statistics and transform the continuous values into discrete ones as in Table 2. Here we have two discrete

Table 2. A Summary of Probability Distribution for the Data in Table 1

Sample	Rich	Fair	Poor	Expressions of New Condensed Features
Young	2	2	1	Age is lower than 30
Old	1	2	2	The others
Low	1	2	3	Salary is lower than 36k
Median	1	1	0	The others
High	1	1	0	Salary is higher than 50k

levels, young and old, representing the age data, and three levels, low, median, and high, referring to the salary data. We collect all the probabilities and derive the ones for prediction based on Equation (2):

$P(\text{young, low} \mid C_{\text{Rich}}) = 1/3$, $P(\text{young, low} \mid C_{\text{Fair}}) = 1/2$, $P(\text{young, low} \mid C_{\text{Poor}}) = 1/3$,

$P(\text{young, median} \mid C_{\text{Rich}}) = 1/3$, $P(\text{young, median} \mid C_{\text{Fair}}) = 0$, $P(\text{young, median} \mid C_{\text{Poor}}) = 0, \ldots$

$P(\text{young, low}) = 4/10$, $P(\text{young, median}) = 1/10$, $P(\text{young, high}) = 0, \ldots$

$P(C_{\text{Rich}}) = 3/10$, $P(C_{\text{Fair}}) = 2/5$, $P(C_{\text{Poor}}) = 3/10$

$P(C_{\text{Rich}} \mid \text{young, low}) = 1/4$, $P(C_{\text{Fair}} \mid \text{young, low}) = 1/2$, $P(C_{\text{Poor}} \mid \text{young, low}) = 1/4$,

$P(C_{\text{Rich}} \mid \text{young, median}) = 1$, $P(C_{\text{Fair}} \mid \text{young, median}) = 0$, $P(C_{\text{Poor}} \mid \text{young, median}) = 0, \ldots$

Because there are two features representing the data, we compute the joint probabilities instead of individual probabilities. Here we assume that the two features have the same degree of significance. At this point, we have constructed a model to express the data with their two features. The derived probabilities can be regarded as a set of rules to decide the classes of to-be-classified data.

If there is a to-be-classified datum X whose age is 26 and salary is 35k, we apply the derived rules to label X. We transform the pattern of X to indicate that the age is young and the salary is low. To find the suitable rules, we can define a penalty function $\lambda(C_i|C_j)$, which denotes the payment when a datum belonging to C_j is classified into C_i. Let the value of this function be 1 if C_j is not equal to C_i and 0 if two classes are the same. Furthermore, we can define a distance measure $\iota(X, C_i)$ as in Equation (3), which represents the total amount of payments when we classify X into C_i. We conclude that the lower the value of $\iota(X, C_i)$, the higher the probability that X belongs to C_i. In this example, we label X by C_{Fair} because $\iota(X, C_{\text{Fair}})$ is the lowest.

$$\iota(X, C_i) = \sum_j \lambda(C_i|C_j) \times P(C_j|X)$$
$$\therefore\ \iota(X, C_{\text{Rich}}) = 3/4,\ \iota(X, C_{\text{Fair}}) = 1/2,\ \iota(X, C_{\text{Poor}}) = 3/4 \quad (3)$$

Fuzzy Data Analysis. Fuzzy-set theory, established by Zadeh (4) allows a gradual membership $MF_A(X)$ for any datum X on a specified set A. Such an approach more adequately models the data uncertainty than using the common notion of set membership. Take cluster analysis as an example. Each datum belongs to exactly one cluster after the classification procedure. Often, however, the data cannot be assigned exactly to one cluster in the real world, such as the jobs of a busy person, the interests of a researcher, or the conditions of the weather. In the following, we replace the previous example for supervised data analysis with the fuzzy-set notion to show its characteristic.

Consider a universe of data U and a subset A of U. Set theory allows us to express the membership of A on U by the characteristic function $F_A(X) : U \to \{0, 1\}$.

$$F_A(X) = \begin{cases} 1, X \in A \\ 0, X \notin A \end{cases} \quad (4)$$

From the above, it can be clearly determined whether X is an element of A. However, many real-world phenomena make such a unique decision impossible. In this case, expressing in a degree of membership is more suitable. A fuzzy set A on U can be represented by the set of pairs that describe the membership function $MF_A(X) : U \to [0, 1]$ as defined in Ref. 5.

$$A = \{(X, MF_A(X)) | X \in U, MF_A(X) \in [0, 1]\} \quad (5)$$

Example. Table 3 contains a fuzzy-set representation of the dataset in Table 1. The membership function of each sample is expressed in a form of possibility that stands for the degree of the acceptance that a sample belongs to a class. Under the case of supervised data analysis, the to-be-classified datum X needs to be labeled using an appropriate classification procedure. The distance between each sample and X is calculated using the two features and Euclidean distance:

1. Find the K NNs of X from S.
2. Compute the membership function of X for each class.
3. Label X by the class with a maximal membership.
4. Add X to S and stop the process.

The first stage in finding K samples with minimal distances is the same, so we have the same set of 4 NNs

Table 3. Fuzzy-Set Membership Functions for the Data in Table 1

Sample	Rich	Fair	Poor	Estimated Distance Between Each Sample and X
Y_1	0.5	0.2	0.3	11.66
Y_2	0.1	0.5	0.4	20.39
Y_3	0	0.2	0.8	20.09
Y_4	0.6	0.3	0.1	5.38
Y_5	0.2	0.5	0.3	10.19
Y_6	0.2	0.5	0.2	6.4
Y_7	0	0	1	15.52
Y_8	0.9	0.1	0	25.7
Y_9	0	0.3	0.7	11.18
Y_{10}	0.4	0.6	0	37.69
X	0.2	0.42	0.38	

$\{Y_4, Y_5, Y_6, Y_9\}$ when the value of $K = 4$. Let $\delta(X, Y_j)$ denote the distance between X and the sample Y_j. In the next stage, we calculate the membership function $MF_{C_i}(X)$ of X for each class C_i as follows:

$$MF_{C_i}(X) = \frac{\sum_j MF_{C_i}(Y_j) \times \delta(X, Y_j)}{\sum_j \delta(X, Y_j)}, \forall Y_j \in K \text{ NNs of } X \quad (6)$$

$$MF_{C_{\text{rich}}}(X) = \frac{0.6 \times 5.38 + 0.2 \times 10.19 + 0.2 \times 6.4 + 0 \times 11.18}{5.38 + 10.19 + 6.4 + 11.18} \approx 0.2$$

$$MF_{C_{\text{fair}}}(X) = \frac{0.3 \times 5.38 + 0.5 \times 10.19 + 0.6 \times 6.4 + 0.3 \times 11.18}{5.38 + 10.19 + 6.4 + 11.18} \approx 0.42$$

$$MF_{C_{\text{poor}}}(X) = \frac{0.1 \times 5.38 + 0.3 \times 10.19 + 0.2 \times 6.4 + 0.7 \times 11.18}{5.38 + 10.19 + 6.4 + 11.18} \approx 0.38$$

Because the membership of X for class C_{Fair} is higher than all others, we label X by C_{Fair}. The resultant membership directly gives a confidence measure of this classification.

INTERNET DATA ANALYSIS METHODS

The dramatic growth of information systems over the past years has brought about the rapid accumulation of data and an increasing need for information sharing. The World Wide Web (WWW) combines the technologies of the uniform resource locator (URL) and hypertext to organize the resources in the Internet into a distributed hypertext system (6). As more and more users and servers register on the WWW, data analysis on its rich content is expected to produce useful results for various applications. Many research communities such as network management (7), information retrieval (8), and database management (9) have been working in this field.

The goal of Internet data analysis is to derive a classification or clustering of Internet data, which can provide a valuable guide for the WWW users. Here the Internet data can be two kinds of materials, web page and web log. Each site within the WWW environment contains one or more web pages. Under this environment, any WWW user can make a request to any site for any web page in it, and the request is then recorded in its log. Moreover, the user can also roam through different sites by means of the anchors provided in each web page. Such an approach leads to the essential difficulties for data analysis:

1. Huge amount of data.
2. Frequent changes.
3. Heterogeneous presentations.

Basically the Internet data originate from all over the world; the amount of data is huge. As any WWW user can create, delete, and update the data, and change the locations of the data at any time, it is difficult to get a precise view of the data. Furthermore, the various forms of expressing the same data also reveal the status of the chaos on the WWW. As a whole, Internet data analysis should be able to handle the large amount of data and to control the uncertainty factors in a practical way. In this section, we first introduce the method for data analysis on web pages and then describe the method for data analysis on web logs.

Web Page Analysis

Many tools for Internet resource discovery (10) use the results of data analysis on the WWW to help users find the correct positions of the desired resources. However, many of these tools essentially keep a keyword-based index of the available web pages. Owing to the imprecise relationship between the semantics of keywords and the web pages (11), this approach clearly does not fit the user requests well. From the experiments in Ref. 12, the text-based classifier that is 87% accurate for Reuters (news documents) yields only 32% precision for Yahoo (web pages). Therefore, a new method for data analysis on web pages is required. Our approach is to use the anchor information in each web page, which contains much stronger semantics for connecting the user interests and those truly relevant web pages.

A typical data analysis procedure consists of the following stages:

1. Observe the data.
2. Collect the samples.
3. Select the features.
4. Classify the data.
5. Evaluate the results.

In the first stage, we observe the data and conclude with a set of features that may be effective for classifying the data. Next, we collect a set of samples based on a given scope. In the third stage, we estimate the fitness of each feature for the collected samples to determine a set of effective features. Then, we classify the to-be-classified data according to the similarity measure on the selected features. At last, we evaluate the classified results and find a way for the further improvement. In the following, we first give some results of the study on the nature of web pages. Then we show the feature selection stage and a procedure to classify the web pages.

Data Observation. In the following, we provide two directions for observing the web pages.

Semantic Analysis. We may consider the semantics of a web page as potential features. Keywords contained in a web page can be analyzed to determine the semantics such as which fields it belongs to or what concepts it provides. There are many ongoing efforts on developing techniques to derive the semantics of a web page. The research results of information retrieval (13,14) can also be applied for this purpose.

Observing the data formats of web pages, we can find several parts expressing the semantics of the web pages to some extent. For example, the title of a web page usually refers to a general concept of the web page. An anchor, which is constructed by the web page designer, provides a URL of another web page and makes a connection between the two web pages. As far as the web page designer is concerned, the anchor texts must sufficiently express the semantics of the whole web page to which the anchor points. As to the viewpoint of a WWW user, the motivation to follow an anchor is based on the fact that this anchor expresses desired semantics for the user. Therefore, we can make a proper connection between the user's interests and those truly relevant web pages. We can group the anchor texts to generate a corresponding classification of the web pages pointed to by these anchor texts. Through this classification, we can relieve the WWW users of the difficulties on Internet resource discovery through a query facility.

Syntactic Analysis. Because the data formats of web pages follow the standards provided on the WWW, for example, hypertext markup language (HTML), we can find potential features among the web pages. Consider the features shown in Table 4. The white pages, which mean the web pages with a list of URLs, can be distinguished from the ordinary web pages by a large number of anchors and the short distances between two adjacent anchors within a web page. Note that here the distance between two anchors means the number of characters between them. For the publication, the set of headings has to contain some specified keywords, such as "bibliography" or "reference." The average distance between two adjacent anchors has to be lower than a given threshold, and the placement of anchors has to center to the bottom of the web page.

According to these features, some conclusions may be drawn in the form of classification rules. For instance, the web page is designed for publication if it satisfies the requirements of the corresponding features. Obviously, this approach is effective only when the degree of support for such rules is high enough. Selection of effective features is a way to improve the precision of syntactic analysis.

Sample Collection. It is impossible to collect all web pages and thus, choosing a set of representative samples becomes a very important task. On the Internet, we have two approaches to gather these samples, as follows:

1. Supervised sampling.
2. Unsupervised sampling.

Supervised sampling means that the sampling process is based on the human knowledge specifying the scope of the samples. In supervised data analysis, a classification template that consists of a set of classes exists. The sampling scope can be set based on the template. The sampling is more effective when all classes in the template contain at least one sample. On the other hand, we consider unsupervised sampling if there is not enough knowledge about the scope, as in the case of unsupervised data analysis. The most trivial way to get samples is to choose any subset of web pages. However, this arbitrary sampling may not fit the requirement of random sampling well. We recommend the use of search engines that provide different kinds of web pages in the form of a directory.

Table 4. Potential Features for Some Kinds of Home Pages

Type of Web Page	Potential Feature
White page	Number of anchors, average distance between two adjacent anchors
Publication	Headings, average distance between two adjacent anchors, anchor position
Person	Title, URL directory
Resource	Title, URL filename

Feature Selection. In addition to collecting enough samples, we have to select suitable features for the subsequent classification. No matter how good the classification scheme is, the accuracy of the results would not be satisfactory without effective features. A measure for the effectiveness of a feature is to estimate the degree of class separability. A better feature implies higher class separability. This measure can be formulated as a criterion to select effective features.

Example. Consider the samples shown in Table 5. From Table 4, there are two potential features for white pages, the number of anchors (F_0) and the average distance between two adjacent anchors (F_1). We assume that $F_0 \geq 30$ and $F_1 \leq 3$ when the sample is a white page. However, a sample may actually belong to the class of white pages although it does not satisfy the assumed conditions. For example, Y_6 is a white page although its $F_0 < 30$. Therefore, we need to find a way to select effective features.

From the labels, the set membership of the two classes is as follows, where the class C_1 refers to the class of white pages:

$$C_0 = \{Y_1, Y_2, Y_3, Y_4, Y_5\}, C_1 = \{Y_6, Y_7, Y_8, Y_9, Y_{10}\}$$

We can begin to formulate the class separability. In the following formula, we assume that the number of classes is c, the number of samples within class C_j is n_j,

Table 5. A Set of Samples with Two Features

Sample	F_0	F_1	White Page
Y_1	8	5	No
Y_2	15	3.5	No
Y_3	25	2.5	No
Y_4	35	4	No
Y_5	50	10	No
Y_6	20	2	Yes
Y_7	25	1	Yes
Y_8	40	2	Yes
Y_9	50	2	Yes
Y_{10}	80	8	Yes

Note: F_0 denotes the number of anchors.
F_1 denotes the average distance for two adjacent anchors.
The labels are determined by human knowledge.

and Y_k^i denotes the kth sample in the class C_i. First, we define interclass separability D_b, which represents the ability of a feature to distinguish the data between two classes. Next, we define the intraclass separability D_w, which expresses the power of a feature to separate the data within the same class. The two measures are formulated in Equations (7) and (8) based on the Euclidean distance defined in Equation (1). As a feature with larger D_b and smaller D_w can get a better class separability, we define a simple criterion function D_{F_j} [Equation (9)] as a composition of D_b and D_w to evaluate the effectiveness of a feature F_j. Based on this criterion function, we get $D_{F_0} = 1.98$ and $D_{F_1} = 8.78$. Therefore, F_1 is more effective than F_0 because of its higher class separability.

$$D_b = \frac{1}{2}\sum_{i=1}^{c} P_i \sum_{j \neq i} P_j \frac{1}{n_i \times n_j} \sum_{k=1}^{n_i}\sum_{m=1}^{n_j} \delta(Y_k^i,\, Y_m^j), \text{ where}$$
$$P_i = \frac{n_i}{\sum_{j=1}^{c} n_j} \tag{7}$$

$$D_w = \frac{1}{2}\sum_{i=1}^{c} P_i \sum_{j = i} P_j \frac{1}{n_i \times n_j} \sum_{k=1}^{n_i}\sum_{m=1}^{n_j} \delta(Y_k^i, Y_m^i), \text{ where}$$
$$P_i = \frac{n_i}{\sum_{j=1}^{c} n_j} \tag{8}$$

$$D_{F_j} = D_b - D_w \tag{9}$$

We have several ways to choose the most effective set of features, as follows:

1. Ranking approach.
2. Top-down approach.
3. Bottom-up approach.
4. Mixture approach.

The ranking approach selects the features one by one according to the rank of their effectiveness. Each time we include a new feature from the rank, we compute the joint effectiveness of the features selected so far by Equations (7)–(9). When the effectiveness degenerates, the process terminates. Using a top-down approach, we consider all features as the initial selection and drop the features one by one until the effectiveness degenerates. On the contrary, the bottom-up approach adds a feature at each iteration. The worse case of the above two approaches occurs if we choose the bad features earlier in the bottom-up approach or the good features earlier in the top-down approach. The last approach allows us to add and drop the features at each iteration by combining the above two approaches. After determining the set of effective features, we can start the classification process.

Data Classification. In the following, we only consider the anchor semantics as the feature, which is based on the dependency between an anchor and the web page to which the anchor points. As mentioned, the semantics expressed by the anchor implies the semantics of the web page to which the anchor points, and describes the desired web pages for the users. Therefore, grouping the semantics of the anchors is equivalent to classifying the web pages into different classes. The classification procedure consists of the following stages:

1. Label all sample pages.
2. For each labeled page, group the texts of the anchors pointing to it.
3. Record the texts of the anchors pointing to the to-be-classified page.
4. Classify the to-be-classified page based on the anchor information.
5. Refine the classification process.

In the beginning, we label all samples and record all anchors pointing to them. Then we group together the anchor texts contained in the anchors pointing to the same page. In the third stage, we group the anchor texts contained in the anchors pointing to the to-be-classified page. After the grouping, we decide the class of the to-be-classified page according to the corresponding anchor texts. At last, we can further improve the effectiveness of the classification process. There are two important measures during the classification process. One is the similarity measure of two data, and the other is the criterion for relevance feedback.

Similarity Measure. After the grouping of samples, we have to measure the degree of membership between the to-be-classified page and each class. Considering the Euclidean distance again, there are three kinds of approaches for such measurement:

1. Nearest-neighbor approach.
2. Farthest-neighbor approach.
3. Mean approach.

The first approach finds the sample in each class nearest to the to-be-classified page. Among these representative samples, we can choose the class containing the one with a minimal distance and assign the page to it. On the other hand, we can also find the farthest sample in each class from the page. Then we assign the page to the class that contains the representative sample with a minimal distance. The last approach is to take the mean of each class into consideration. As in the previous approaches, the mean of each class represents a whole class, and the one with a minimal distance from the page would be chosen. An example follows by using the mean approach.

Example. Inspect the data shown in Table 6. There are several web pages and anchor texts contained in some anchors pointing to the web pages. Here we consider six types of anchor texts, T_1, T_2, ..., and T_6. The value of an anchor text for a web page stands for the number of the anchors pointing to the web page, which contain the anchor text. The labeling is the same as in the previous example.

Table 6. A Set of Home Pages with Corresponding Anchor Texts and Labels

Sample	T_1	T_2	T_3	T_4	T_5	T_6	White Page
Y_1	0	0	0	1	1	2	No
Y_2	0	1	2	0	0	2	No
Y_3	0	2	0	4	0	0	No
Y_4	0	0	3	0	0	1	No
Y_5	2	2	0	0	0	0	No
Y_6	1	3	0	0	2	3	Yes
Y_7	3	3	1	6	3	0	Yes
Y_8	4	2	5	0	1	0	Yes
Y_9	5	5	3	0	0	2	Yes
Y_{10}	8	4	4	1	4	2	Yes
X	5	2	0	0	5	0	Yes

Note: T_1 = “list”, T_2 = “directory”, T_3 = “classification”, T_4 = “bookmark”, T_5 = “hot”, and T_6 = “resource”.
The labels are determined by human knowledge.

We calculate the means of the two classes as follows:

$$M_0 = (0.4,\ 1,\ 1,\ 1,\ 0.2,\ 1),\ M_1 = (4.2,\ 3.4,\ 2.6,\ 1.4,\ 2,\ 1.4)$$

Suppose there is a web page X to be classified as shown in Table 6. We can compute the distances between X and the two means. They are $\delta(X, M_0) = 6.94$ and $\delta(X, M_1) = 4.72$. Thus, we assign X to class C_1.

Relevance Feedback. The set of samples may be enlarged after a successful classification by including the classified pages. However, the distance between a to-be-classified page and the nearest mean may be very large, which means that the current classification process does not work well on this web page. In this case, we reject to classify such a web page and wait until more anchor texts for this web page are accumulated. This kind of rejection not only expresses the extent of the current ability to classify web pages, but also it promotes the precision of the classified results. Furthermore, by the concept of class separability formulated in Equations (7)–(9), we can define a similar criterion function D_S to evaluate the performance of the current set of samples:

$$D_S = D_F(S) \tag{10}$$

where F is the set of all effective features and S is the current set of samples.

Example. Reconsider the data shown in Table 6. Before we assign X to C_1, the initial D_S equals to 0.75. When C_1 contains X, $D_{S\cup\{X\}}$ yields a smaller value 0.16. On the other hand, $D_{S\cup\{X\}}$ becomes 1.26 if we assign X to C_0. Hence, although X is labeled by C_1, it is not suitable to become a new sample for the subsequent classification. The set of samples can be enlarged only when such an addition of new samples gains a larger D_S value, which means the class separability is improved.

Web Log Analysis

The browsing behavior of the WWW users is also interesting to data analyzers. Johnson and Fotouhi (15) propose a technique to aid users in roaming through the hypertext environment. They gather and analyze the browsing paths of some users to generate a summary as a guide for other users. Within the WWW environment, the browsing behavior can be found in three positions, the history log of the browser, the access log of the proxy server, and the request log of the web server. Data analysis on the web logs at different positions will lead to different applications. Many efforts have been made to apply the results of web log analysis, for example, the discovery of marketing knowledge (16) the dynamic generation of anchors (17) and the quality evaluation of website design (18).

For web log analysis, we can also follow the typical data analysis procedure as described previously. Here we illustrate with an example of unsupervised data analysis. In the first stage, we observe the content of the web log to identify a set of features that can represent the desired user behavior. Next, we may collect a set of samples from the entire log to reduce the processing cost. In the third stage, we choose a proper subset of feature values according to some criteria such as arrival time and number of occurrences. Then, we divide the users into clusters according to the similarity measure on the features representing their behaviors. At last, we evaluate the clustered results and find a way for further improvement. In the following, we first introduce two kinds of representations of user behavior, or user profiles as they are sometimes called (19). Then we show a procedure to derive user clusters from the web log.

Data Observation and Feature Selection. The web log usually keeps the details of each request. The following is an example record:

```
890984441.324 0 140.117.11.12 UDP_MISS/000 76 ICP_QUERY http://www.yam.org.tw/b5/yam/...
```

According to application needs, only parts of the fields are retrieved, for instance, arrival time (890984441.324), IP address (140.117.11.12), and URL (http://www.yam.org.tw/b5/yam/). The records with the same IP address can be concatenated as a long sequence by the order of their arrival times. Such a long sequence often spans a long time and implies the user behavior composed of more than one browses, where a browse means a series of navigations for a specific goal. We can examine the arrival times of every two consecutive requests to cut the sequence into shorter segments, which exactly correspond to the individual browses. In this way, a browse is represented as a sequence of requests (URLs) within a time limit.

We may also represent the browsing behavior without temporal information. For a sequence of URLs, we can count the number of occurrences for each URL in it and represent this sequence as a vector. For example, we can represent a sequence $<a, b, a, b, c, d, b, c>$ as a vector [2 3 2 1], where each value stands for the number of occurrences of a, b, c, and d, respectively. Compared with the previous one, this representation emphasizes the frequency of each URL instead of their order in a browse.

User Clustering. In the following, we consider the vector representation as the browsing behavior and assume that similar vectors imply two browses with similar information needs. Therefore, clustering the vectors is equivalent to clustering the users' information needs. We use the leader algorithm (20) and Euclidean distance as defined in Equation (1) for clustering. The input is a set of vectors S and the output is a set of clusters C. Two thresholds *min* and *max* are also given to limit, respectively, the sum of values in a vector and the distance between two vectors in the same cluster. The algorithm processes the vectors one by one and terminates after all of them are processed. Different orderings of vectors can lead to different final partitions. One way to get the best partition is to try this algorithm on all different orderings. Without loss of generality, we consider that a fixed ordering of vectors and the clustering procedure is as follows:

For each vector v in S, do the following steps:

1. If the sum of values in v is smaller than *min*, discard v.
2. Compute the distance between v and the mean of each cluster in C; let the cluster with the minimum distance be c, and let d denote the minimum distance.
3. If d is larger than *max*, create a new cluster $\{v\}$ and add it to C; otherwise, assign v to c.
4. Compute the mean of each cluster in C.

The first step filters out the vector with a small number of requests because it is not very useful in applications. Then the distance between the vector v and the mean of each cluster is computed. The cluster c with the minimum distance d is then identified. If d is large, we treat v as the mean of a new cluster; otherwise, v is assigned to c.

Example. Consider the data in Table 7. There is a set of vectors on four URLs. Let *min* and *max* be set to 2 and 6, respectively. The following shows the iterations for the clustering:

For $Y_1 = [0\ 0\ 0\ 1]$,
  1 The sum of value $< min \Rightarrow \text{discard}\ Y_1$.

For $Y_2 = [1\ 3\ 0\ 0]$,
  1–3 Create $C_1 = \{Y_2\}$.
  4 Set the median $M_1 = [1\,3\,0\,0]$.

For $Y_3 = [3\ 3\ 1\ 6]$,
  1,2 Compute the distance between Y_3 and M_1: $d_1 = 6.4$.
  3 $d_1 > max \Rightarrow \text{Create}\ C_2 = \{Y_3\}$.
  4 Set the median $M_2 = [3\,3\,1\,6]$.

Table 7. A Set of Vectors on Six URLs

Vector	URL$_1$	URL$_2$	URL$_3$	URL$_4$
Y_1	0	0	0	1
Y_2	1	3	0	0
Y_3	3	3	1	6
Y_4	4	2	5	0
Y_5	5	5	3	0

For $Y_4 = [4\ 2\ 5\ 0]$,
  1,2 Compute the distance between Y_4 and M_1: $d_1 = 5.9$; the distance between Y_4 and M_2: $d_2 = 7.3$.
  3 $d_1 < max \Rightarrow \text{assign}\ Y_4\ \text{to}\ C_1 = \{Y_2,\ Y_4\}$.
  4 Compute the new median $M_1 = [2.5\ 2.5\ 2.5\ 0]$.

For $Y_5 = [5\ 5\ 3\ 0]$,
  1,2 Compute the distance between Y_5 and M_1: $d_1 = 3.6$; the distance between Y_5 and $M_2 : d_2 = 6.9$.
  3 $d_1 < max \Rightarrow \text{assign}\ Y_5\ \text{to}\ C_1 = \{Y_2,\ Y_4,\ Y_5\}$.
  4 Compute the new median $M_1 = [3.3\ 3.3\ 2.7\ 0]$.

ADVANCED INTERNET DATA ANALYSIS METHODS

Although the previous procedures fit the goal of data analysis well, there are still problems, such as speed or memory requirements and the complex nature of real-world data. We have to use some advanced techniques to improve the performance. For example, the number of clusters given in unsupervised data analysis has a significant impact on the time spent in each iteration and the quality of final partition. Notice that the initial partition may contribute to a specific sequence of adjustments and then to a particular solution. Therefore, we have to find an ideal number of clusters during the analysis according to the given initial partition. The bottom-up approach with decreasing the number of clusters in iterations is a way to adjust the final partition. Given a threshold of similarity among the clusters, we can merge two clusters that are similar enough to become a new single cluster. The number of clusters is determined when there are no more similar clusters to be merged. In the following subsections, we introduce two advanced techniques for Internet data analysis.

Techniques for Web Page Analysis

The approach to classifying Web pages by anchor semantics requires a large amount of anchor texts. These anchor texts may be contained in the anchors pointing to the Web pages in different classes. An anchor text is said to be indiscernible when it cannot be used to distinguish the Web pages in different classes. We use the rough-set theory (21, 22) to find the indiscernible anchor texts, which will then be removed. The remaining anchor texts will contribute to a higher degree of accuracy for the subsequent classification. In addition, the cost of distance computation can also be reduced. In the following, we introduce the basic idea of the rough-set theory and an example for the reduction of anchor texts (23).

Rough-Set Theory. By the rough-set theory, an information system is modeled in the form of a 4-tuple (U, A, V, F), where U represents a finite set of objects, A refers to a finite set of attributes, V is the union of all domains of the attributes in A, and F is a binary function $(F :\ U \times A \rightarrow V)$. The attribute set A often consists of two subsets, one refers to condition attribute C and the other stands for decision attribute D. In the classification on web pages, U stands for all web pages, A is the union of the anchor texts

(C) and the class of web pages (D), V is the union of all the domains of the attributes in A, and F handles the mappings. Let B be a subset of A. A binary relation called the indiscernibility relation is defined as

$$\mathrm{IND}_B = \{(X_i, X_j) \in U \times U | \forall\, p \in B,\ p(X_i) = p(X_j)\} \quad (11)$$

That is, X_i and X_j are indiscernible by the set of attributes B if $p(X_i)$ is equal to $p(X_j)$ for every attribute p in B. IND_B is an equivalence relation that produces an equivalence class denoted as $[X_i]_B$ for each sample X_i. Two web pages X_i and X_j, which have the same statistics for each anchor text in C, belong to the same equivalence class $[X_i]_C$ (or $[X_j]_C$). Let U' be a subset of U. A lower approximation $\mathrm{LOW}_{B,U'}$, which contains all samples in each equivalence class $[X_i]_B$ contained in U', is defined as

$$\mathrm{LOW}_{B,U}' = \{X_i \in U | [X_i]_B \subset U'\} \quad (12)$$

Based on Equation (12), $\mathrm{LOW}_{C,[X_i]_D}$ contains the web pages in the equivalence classes produced by IND_C, and these equivalence classes are contained in $[X_i]_D$ for a given X_i. A positive region $\mathrm{POS}_{C,D}$ is defined as the union of $\mathrm{LOW}_{C,[X_i]_D}$ for each equivalence class produced by IND_D·$\mathrm{POS}_{C,D}$ refers to the samples that belong to the same class when they have the same anchor texts. As defined in Ref. 24, C is independent on D if each subset C_i in C satisfies the criterion that $\mathrm{POS}_{C,D} \neq \mathrm{POS}_{C_i,D}$; otherwise, C is said to be dependent on D. The degree of dependency $\gamma_{C,D}$ is defined as

$$\gamma_{C,D} = \frac{\mathrm{card}(\mathrm{POS}_{C,D})}{\mathrm{card}(U)} \quad (13)$$

where card denotes set cardinality.

$$\mathrm{CON}_{p,\gamma_{C,D}} = \gamma_{C,D} - \gamma_{C-\{p\},D} \quad (14)$$

From these equations, we define the contribution $\mathrm{CON}_{p,\gamma_{C,D}}$ of an anchor text p in C to the degree of dependency $\gamma_{C,D}$ by using Equation (14). According to Equation (13), we say an anchor text p is dispensable if $\gamma_{C-\{p\},D} = \gamma_{C,D}$. That is, the anchor text p makes no contribution to $\gamma_{C,D}$ and the value of $\mathrm{CON}_{p,\gamma_{C,D}}$ equals 0. The set of indispensable anchor texts is the core of the reduced set of anchor texts. The remaining task is to find a minimal subset of C called a reduct of C, which satisfies Equation (15) and the condition that the minimal subset is independent on D.

$$\mathrm{POS}_{C,D} = \mathrm{POS}_{\text{minimal subset of } C,D} \quad (15)$$

Reduction of Anchor Texts. To employ the concepts of the rough-set theory for the reduction of anchor texts, we transform the data shown in Table 6 into those in Table 8. The numeric value of each anchor text is transformed into a symbol according to the range in which the value falls. For instance, a value in the range between 0 and 2 is transformed into the symbol L. This process is a generalization technique usually used for a large database.

Table 8. A Set of Data in Symbolic Values Transformed from Table 6

Sample	T_1	T_2	T_3	T_4	T_5	T_6	White Page
Y_1	L	L	L	L	L	L	No
Y_2	L	L	L	L	L	L	No
Y_3	L	L	L	M	L	L	No
Y_4	L	L	M	L	L	L	No
Y_5	L	L	L	L	L	L	No
Y_6	L	M	L	L	L	M	Yes
Y_7	M	M	L	H	M	L	Yes
Y_8	M	L	M	L	L	L	Yes
Y_9	M	M	M	L	L	L	Yes
Y_{10}	H	M	M	L	M	L	Yes
X	M	L	L	L	M	L	Yes

Note: L = [0, 2], M = [3, 5], H = [6, 8].

By Equation (14), we can compute $\mathrm{CON}_{p,\gamma_{C,D}}$ for each anchor text p and sort them in ascending order. In this case, all $\mathrm{CON}_{p,\gamma_{C,D}}$ are 0 except $\mathrm{CON}_{T_1,\gamma_{C,D}}$. That is, only the anchor text T_1 is indispensable, which becomes the unique core of C. Next, we use a heuristic method to find a reduct of C because such a task has been proved to be NP-complete in Ref. 25. Based on an arbitrary ordering of the dispensable anchor texts, we check the first anchor text to see whether it is dispensable. If it is, then remove it and continue to check the second anchor text. This process continues until no more anchor texts can be removed.

Example. Suppose we sort the dispensable anchor texts as the sequence <T_2, T_3, T_4, T_5, T_6>, we then check one at a time to see whether it is dispensable. At last, we obtain the reduct $\{T_1, T_6\}$. During the classification process, we only consider these two anchor texts for a similarity measure. Let the symbols used in each anchor text be transformed into three discrete values, 0, 1, and 2. The means of the two classes are $M_0 = (0, 0)$ and $M_1 = (1, 0.8)$. Finally, we classify X into the class C_1 due to its minimum distance. When we use the reduct $\{T_1, T_6\}$ to classify data, the class separability $D_{\{T_1, T_6\}}$ is 0.22. Different reducts may result in different values of class separability. For instance, the class separability becomes 0.27 if we choose the reduct $\{T_1, T_2\}$.

Techniques for Web Log Analysis

The approach to clustering the user behaviors must adapt itself to the rich content, large amount, and dynamic nature of log data. As mentioned, the web log can provide a variety of descriptions of a request. Consider the web log of an online bookstore. As the user browses an article in it, the web log may record several fields about that article, such as URL, title, authors, keywords, and categories. Notice that the last three fields represent a request as a set of items, and thus, a browse is represented as a sequence of itemsets. In addition, the dynamic nature of log data implies that the users may change their interests or behaviors as time passes. Therefore, extracting the most significant information from a long history of user browsing is important. We apply the association mining concepts (26) to extract the most representative information (use profile) from a sequence of itemsets. In the following, we introduce the basic concepts used in the data mining field and illustrate by an example how to derive the user profile with data mining techniques (27).

Association Mining Concepts. According to the definition in Ref. 26, a transaction has two fields, transaction-time and the items purchased. An itemset is a nonempty set of items, and thus, each transaction is also an itemset. The length of an itemset is the total number of items in it. A k-itemset stands for an itemset with length k. Furthermore, if a transaction contains an itemset I, we call that the transaction supports I. The support of an itemset is the percentage of transactions that support it in the entire database. If the support of an itemset is larger than a minimum support threshold, we call it a frequent itemset. Given a transaction database, the goal of association mining is to efficiently find all the frequent itemsets.

Consider the web log analysis for an online bookstore. We can regard a transaction as the set of categories (or authors or keywords) logged during a time period that the user enters the online bookstore. The categories in a transaction are equally treated. Thus, for each user, there will be a series of transactions recorded in the web log. Moreover, we can identify a set of categories, for instance, {"data mining", "database", "multimedia"}, which has frequently appeared in these transactions, to represent the user's interests.

Profile Derivation. We employ the association mining concepts to derive the user profile from the transactions in the web log. Due to the accumulation of transactions, the profile derivation may incur high overheads if we use the traditional mining method (28). Therefore, we adopt an incremental method to derive the user profile. The core of our method is the *interest table*, which is built for each user to keep the categories in the transactions. An interest table consists of four columns, while each row of the table refers to a category. An example with four transactions is shown in Table 9. The Category column lists the categories that appear in the transactions. The First and Last columns record the first and the last transactions in which each category appears, respectively. Finally, the Count column keeps the number of occurrences for each category since its first transaction. From the interest table, we can compute the support of each category as follows (where c is a category and T is the current transaction):

$$\text{Support}(c) = \text{Count}(c)/(T - \text{First}(c) + 1) \tag{16}$$

This formula indicates that only the transactions after the first transaction of this category are considered in the support measure. In other words, we ignore the effects of the transactions before the first transaction of this category. Consider the scenario that a category first appears in $T91$ and then continually shows up from $T92$ to $T100$. By the traditional method, its support is small (10%). In contrast, our formula computes the support at 100%, indicating that this category appears frequently in the recent transactions. Given a threshold α, the user profile is $\{c|Support(c) \geq \alpha\}$.

By Equation (16), when a category first appears, its support will be always 100%, which is not reasonable. Therefore, we define a threshold, called minimal count (denoted by β), to filter out the categories with very small counts. On the other hand, we observe that the support of a category can be very low if its first transaction is very far from the current transaction. Consider the scenario that a category first appears in $T1$, disappears from $T2$ to $T91$, and continually shows up from $T92$ to $T100$. By Equation (16), its support is just 10%. However, the fact that this category appears frequently in the recent transactions implies that the user is getting interested in it recently. Therefore, we define another threshold, called expired time (denoted by γ), to restrict the interval between the last and the current transactions. When this interval exceeds γ, both the First and the Last columns of the category are changed to the current transaction.

Example. Take the four transactions in Table 9 as an example. As category c appears in $T1$, $T2$, and $T4$, its count has a value 3. Moreover, the first and the last transactions of category c are $T1$ and $T4$, respectively. The rightmost column of Table 9 lists the support of each category. In this example, the thresholds α, β, and γ are set to 75%, 2, and 4, respectively. We do not compute the support of category a because its count is less than β. As a result, $\{c, d, e\}$ is derived as the user profile.

When a new transaction $T5$ arrives, we recalculate the supports of the categories that appear in $T5$ or the user profile to update the interest table. As Table 10 shows, the supports of all categories except a are recalculated. By α, we derive the new user profile $\{b, c, e, f\}$. Note that category d is removed from the user profile because its new support is lower than α.

Table 9. Four Transactions and the Interest Table

Transactions	Category	First	Last	Count	Support
T1 {*a*, *c*, *e*}	*a*	*T1*	*T1*	1	N/A
T2 {*b*, *c*, *e*, *f*}	*b*	*T2*	*T4*	2	67%
T3 {*d*, *e*, *f*}	*c*	*T1*	*T4*	3	75%
T4 {*b*, *c*, *d*}	*d*	*T3*	*T4*	2	100%
	e	*T1*	*T3*	3	75%
	f	*T2*	*T3*	2	67%

Table 10. The Interest Table After T5 Arrives

Transactions	Category	First	Last	Count	Support
T1 {*a*, *c*, *e*}	*a*	*T1*	*T1*	1	N/A
T2 {*b*, *c*, *e*, *f*}	*b*	*T2*	*T5*	3	75%
T3 {*d*, *e*, *f*}	*c*	*T1*	*T5*	4	80%
T4 {*b*, *c*, *d*}	*d*	*T3*	*T4*	2	67%
T5 {*b*, *c*, *e*, *f*, *g*}	*e*	*T1*	*T5*	4	80%
	f	*T2*	*T5*	3	75%
	g	*T5*	*T5*	1	N/A

SUMMARY

In this article, we describe the techniques and concepts of data analysis. A variety of data analysis methods are introduced and illustrated by examples. Two categories, supervised data analysis and unsupervised data analysis, are presented according to their different initial conditions and resultant uses. Two methods for data analysis are also described, which are based on probability theory and fuzzy-set theory, respectively. The methods for data analysis on two types of Internet data are presented. Advanced techniques for Internet data analysis are also discussed.

Research Trends

The research trend of data analysis can be seen from two viewpoints. From the viewpoint of data, new data types such as transactions, sequences, trees, graphs, and relational tables, bring new challenges to data analysis. From the viewpoint of knowledge, or the result of data analysis, new interestingness measures of patterns, such as support, confidence, lift, chi-square value, and entropy gain, also bring new challenges to data analysis. It is not easy to come up with a single method that can meet all combinations of these challenges. Several ideas and solutions have been proposed, but many challenges remain. In the following, we introduce two research directions that have recently attracted great attention from the data analyzers.

In many of the new applications, several continuous data streams rapidly flow through computers to be either read or discarded by people. Examples are the data flows on computer, traffic, and telecommunication networks. Recently, data analysis on continuous data streams has become widely recognized (29). Its applications include intrusion detection, trend analysis, and grid computing. The characteristics of data analysis on continuous data streams are as follows:

1. Continuity—Data are continuous and arrive at a variable rate.
2. Infinity—The total amount of data is unbounded.
3. Uncertainty—The order of data cannot be predetermined.
4. Expiration—Data can be read only once.
5. Multiplicity—More than one data stream may arrive at a single site for data analysis.

Data analysis on graphs has become popular due to a variety of new applications emerging in the fields like biology, chemistry, and networking (30). The prevalent use of HTML and XML formats also pushes the data analyzers toward this research direction. The characteristics of data analysis on graphs are as follows:

1. Variety—Graph has various substructures such as a connected subgraph, an ordered tree, and a path.
2. Isomorphism—It is NP-complete to decide whether one graph is a subgraph of another one.
3. Interestingness—The measure of pattern interestingness depends on application needs.
4. Complexity—A very efficient mechanism is required to keep the complex structure of graph data.

BIBLIOGRAPHY

1. B. Nolan, *Data Analysis: An Introduction*, Cambridge, UK: Polity Press, 1994.
2. J. W. Tukey, *Exploratory Data Analysis*, Reading, MA: Addison-Wesley, 1977.
3. A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian Data Analysis*, London: Chapman & Hall, 1995.
4. L. A. Zadeh, Fuzzy sets, *Information Control*, **8**: 338–353, 1965.
5. H. Bandemer and W. Nather, *Fuzzy Data Analysis*, Dordrecht: Kluwer, 1992.
6. T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, The world wide web, *Communications of the ACM*, **37** (8): 76–82, 1994.
7. M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, Enhancing the web's infrastructure: From caching to replication, *IEEE Internet Computing*, **1**(2): 18–27, 1997.
8. V. N. Gudivada, V. V. Raghavan, W. I. Grosky, and R. Kasanagottu, Information retrieval on the world wide web, *IEEE Internet Computing*, **1** (5): 58–68, September/October 1997.
9. D. Florescu, A. Levy, and A. Mendelzon, Database techniques for the world wide web: A survey, *ACM SIGMOD Record*, **27** (3): 59–74, September 1998.
10. K. Obraczka, P. B. Danzig, and S. H. Li, Internet resource discovery services, *IEEE Comp. Mag.*, **26** (9): 8–22, 1993.
11. C. S. Chang and A. L. P. Chen, Supporting conceptual and neighborhood queries on www, *IEEE Trans. on Systems, Man, and Cybernetics*, **28** (2): 300–308, 1998.
12. S. Chakrabarti, B. Dom, and P. Indyk, Enhanced hypertext categorization using hyperlinks, *Proc. of ACM SIGMOD Conference on Management of Data*, Seattle, WA, 1998, pp. 307–318.
13. G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*, New York: McGraw-Hill, 1983.
14. G. Salton, *Automatic Text Processing*, Reading, MA: Addison Wesley, 1989.
15. A. Johnson and F. Fotouhi, Automatic touring in hypertext systems, *Proc. of IEEE Phoenix Conference on Computers and Communications*, Phoenix, AZ, 1993, pp. 524–530.
16. A. Büchner and M. D. Mulvenna, Discovering internet marketing intelligence through online analytical web usage mining, *ACM SIGMOD Record*, **27** (4): 54–61, 1998.
17. T. W. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal, From user access patterns to dynamic hypertext linking, *Computer Networks and ISDN Systems*, **28**: 1007–1014, 1996.
18. M. Perkowitz and O. Etzioni, Adaptive web sites, *Communications of the ACM*, **43** (8): 152–158, 2000.
19. Y. H. Wu and A. L. P. Chen, Prediction of web page accesses by proxy server log, *World Wide Web: Internet and Web Information Systems*, **5** (1): 67–88, 2002.
20. J. A. Hartigan, *Clustering Algorithms*, New York: Wiley, 1975.
21. Z. Pawlak, Rough sets, *Communications of the ACM*, **38** (11): 88–95, 1995.
22. Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Knowledge*, Norwell, MA: Kluwer, 1991.
23. A. L. P. Chen and Y. H. Wu, Data analysis, *Wiley Encyclopedia of Electrical and Electronics Engineering*, New York: Wiley, 1999.

24. X. Hu and N. Cercone, Mining knowledge rules from databases: A rough-set approach, *Proc. of IEEE Conference on Data Engineering*, New Orleans, LA, 1996, pp. 96–105.

25. R. Slowinski (ed.), *Handbook of Applications and Advances of the Rough Sets Theory*, Norwell MA: Kluwer Academic Publishers, 1992.

26. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, New York: Morgan Kaufman Publishers, 2000.

27. Y. H. Wu, Y. C. Chen, and A. L. P. Chen, Enabling personalized recommendation on the web based on user interests and behaviors, *Proc. of IEEE Workshop on Research Issues in Data Engineering*, Heidelberg, Germany, 2001, pp. 17–24.

28. R. Agrawal and R. Srikant, Fast algorithms for mining association rules, *Proc. of Conference on Very Large Data Base*, Santiago de Chile, Chile, 1994, pp. 487–499.

29. L. Golab and M. T. Özsu, Issues in data stream management, *ACM SIGMOD Record*, **32** (2): 5–14, 2003.

30. T. Washio and H. Motoda, State of the art of graph-based data mining, *ACM SIGKDD Explorations*, **5** (1): 2003.

YI-HUNG WU
Chung Yuan Christian
University
Chungli, Taiwan
ARBEE L. P. CHEN
National Chengchi University
Taipei, Taiwan

DATABASE LANGUAGES

BACKGROUND

The DBMS has a DDL complier that processes DDL statements to generate the schema descriptions and store them into the DBMS catalog. The catalog contains metadata that are consulted before the actual data are accessed.

There are two types of DMLs: *low-level* or *procedural* DMLs and *high-level* or *nonprocedural* DMLs. A low-level DML requires the user to specify *how* the data are manipulated. This type of DML typically retrieves and processes individual records from a database. Therefore, a low-level DML is also called a *record-at-a-time* DML. A high-level DML allows users to specify *what* the result is, leaving the decision about how to get the result to the DBMS. Therefore, a high-level DML is also called a *declarative* DML. A high-level DML can specify and retrieve many records in a single statement and is hence called a *set-at-a-time* or *set-oriented* DML. A declarative DML is usually easier to learn and use than a procedural DML. In addition, a declarative DML is sometimes faster in overall processing than its procedural counterpart by reducing the number of communications between a client and a server. A low-level DML may be embedded in a general-purpose programming language such as COBOL, C/C++, or Java. A general-purpose language in this case is called the *host language*, and the DML is called the *data sublanguage*. On the other hand, a high-level DML called a *query language* can be used in a standard-alone and interactive manner.

The main criterion used to classify a database language is the data model based on which the language is defined. The existing data models fall into three different groups: *high-level* or *conceptual data models, implementation data models*, and *physical data models* (1). Conceptual models provide concepts that are close to the way users perceive data, whereas physical data models provide concepts for describing the details about how data are stored. Implementation data models specify the overall logical structure of a database and yet provide a high-level description of the implementation.

High-level data models are sometimes called *object-based* models because they mainly describe the objects involved and their relationships. The *entity-relationship* (E–R) model and the *object-oriented* model are the most popular high-level data models. The E–R model (2) is usually used as a high-level conceptual model that can be mapped to the relational data model. The object-oriented data model adapts the object-oriented paradigm for a database by adding concepts in object-oriented programming such as persistence and collection.

Implementation data models are used most frequently in a commercial DBMS. There are three widely used implementation data models: *relational*, *network*, and *hierarchical*. They organize the data in record structures and hence are sometimes called *record-based* data models. Many database systems in early days were built based on either the network model or the hierarchical model. However, because these two models only support low-level queries and record-at-a-time retrievals, their importance has decreased over the years. The relational model has been successfully used in most commercial database management systems today. This is because the relational model and relational database languages provide high-level query specifications and set-at-a-time retrievals.

The *object-relational* data model is a hybrid of the object-oriented and the relational models. It extends the relational data model by providing an extended type system and object-oriented concepts such as object identity, inheritance, encapsulation, and complex objects.

With the popularity of Extensible Markup Language (XML), XML is becoming a standard tool for data representation and exchange between multiple applications and database systems. Later, we will discuss some advanced database languages developed based on temporal, spatial, and active models.

RELATIONAL DATA MODEL, RELATIONAL ALGEBRA, RELATIONAL CALCULUS, AND RELATIONAL QUERY LANGUAGES

Relational Data Model

The relational data model was introduced by Codd (3). It was developed based on the mathematical notion of a *relation*. Its root in mathematics allows it to provide the simplest and the most uniform and formal representation. The relational data model represents the data in a database as a collection of relations. Using the terms of the E–R model, both entity sets and relationship sets are relations. A relation is viewed as a two-dimensional table in which the rows of the table correspond to a collection of related data values and the values in a column all come from the same domain. In database terminology, a row is called a *tuple* and a column name is called an *attribute*. A *relation schema* R, denoted by $R(A_1, A_2, \ldots, A_n)$, is a set of attributes $R = \{A_1, A_2, \ldots, A_n\}$. Each attribute A_i is a descriptive property in a domain D. The domain of A_i is denoted by dom(A_i). A relation schema is used to describe a relation, and R is called the name of the relation. The degree of a relation is the number of the attributes of its relation schema. A relation instance r of the relation schema $R(A_1, A_2, \ldots, A_n)$, also denoted by $r(R)$, is a set of m-tuples $r = \{t_1, t_2, \ldots, t_m\}$. Each tuple t is an ordered list of n values $t = <v_1, v_2, \ldots, v_n>$, where each value v_i is an element of dom(A_i).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Relational Algebra

A query language is a language in which users can request data from a database. Relational algebra is a procedural language. The fundamental operations in the relational algebra are usually divided into two groups. The first group includes *select*, *project*, and *join* operations that are developed specifically for relational databases. The second group includes set operations from mathematical set theory such as *union, interaction, difference*, and *Cartesian product*. These operations allow us to perform most data retrieval operations.

The Select Operation. The select operation selects a subset of the tuples from a relation. These tuples must satisfy a given predicate. The lowercase Greek letter sigma (σ) is used to denote the select operation and followed by a predicate as a subscript to σ. The argument relation R is given in the parentheses following σ. Suppose we have a sample *University* database that consists of three relations: *Faculty*, *Department*, and *Membership* as shown in Fig. 1. If we wish to find information of all faculty members whose salary is greater than $50,000, we would write

$$\sigma_{\text{salary} > 50000}(\text{Faculty})$$

In general, the select operation is denoted by

$$\sigma_{\Sigma\text{condition}i}(R)$$

Note that the select operation is commutative; that is,

$$\sigma_{\text{condition1}}(\sigma_{\text{condition2}}(R)) = \sigma_{\text{condition2}}(\sigma_{\text{condition1}}(R))$$

In addition, we can combine a cascade of select operations into a single select operation with logical "AND" connectives; that is,

$$\sigma_{\text{condition1}}(\sigma_{\text{condition2}}(\ldots\sigma_{\text{conditionn}}R)))$$
$$= \sigma_{\text{condition1 AND}}\sigma_{\text{condition2 AND ... AND}}\ \sigma_{\text{conditionn}}(R)$$

The Project Operation. The project operation produces a vertical subset of the table by extracting the values from a set of specified columns, eliminating duplicates and placing the values in a new table. The lowercase Greek letter pi (π) is used to denote the project operation and followed by a list of attribute names that we wish to extract in the result as a subscript to π. For example, the following operation produces the names and birthdates of all faculty members:

$$\pi_{\text{name, birthdate}}(\text{Faculty})$$

Relational Expressions. As each relational algebra operation produces a set, we can combine several relational algebra operations into a relational expression. For example, suppose that we want the names of all faculty members whose salary is greater than 50,000 (Fig. 1). We can write a relational expression as follows:

$$\pi_{\text{name}}(\sigma_{\text{salary} > 50000}(\text{Faculty}))$$

The Cartesian Product Operation. The Cartesian product operation, denoted by a cross (X), allows us to combine tuples from two relations so that the result of $R1(A_1, A_2, \ldots, A_n)$ X $R2(B_1, B_2, \ldots, B_m)$ is a relation Q with $n + m$ attributes $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$ in

Faculty

fid	*name*	*birthdate*	*salary (dollar)*	*dcode*
91239876	John Smith	01/09/67	40000	2
33489783	Perry Lee	12/08/55	68000	2
78738498	Tom Bush	06/20/41	79000	4
12323567	Jane Rivera	09/15/72	35000	3
87822384	Joyce Adams	11/10/59	54000	1
78898999	Frank Wong	07/31/79	30000	3
22322123	Alicia Johnson	08/18/80	28000	1

Department

dcode	*name*
1	Chemical Engineering
2	Computer Science
3	Electrical Engineering
4	Mechanical Engineering

Membership

fid	*society*
91239876	American Database Association
33489783	American Software Engineering Society
78738498	International Mechanical Engineering Association
12323567	International Electrical Engineering Association
87822384	International Chemical Engineering Association
78898999	International Electrical Engineering Association
22322123	Advocate for Women in Engineering

Figure 1. Instance of the *University* database.

fid	*Faculty.name*	*birthdate*	*salary (dollar)*	*Faculty. dcode*	*Department. dcode*	*Department.name*
91239876	John Smith	01/09/67	40000	2	2	Computer Science
33489783	Perry Lee	12/08/55	68000	2	2	Computer Science
78738498	Tom Bush	06/20/41	79000	4	4	Mechanical Engineering
12323567	Jane Rivera	09/15/72	35000	3	3	Electrical Engineering
87822384	Joyce Adams	11/10/59	54000	1	1	Chemical Engineering
78898999	Frank Wong	07/31/79	30000	3	3	Electrical Engineering
22322123	Alicia Johnson	08/18/80	28000	1	1	Chemical Engineering

Figure 2. Result of an equijoin of *Faculty* and *Department*.

that order. The tuples of the relation Q are all combinations of the rows from $R1$ and $R2$. Specifically, it combines the first row of $R1$ with the first row of $R2$, then with the second row of $R2$, and so on, until all combinations of the first row of $R1$ with all the rows of $R2$ have been formed. The procedure is repeated for the second row of $R1$, then the third row of $R1$, and so on. Assume that we have p tuples in $R1$ and q tuples in $R2$. Then, there are $p \times q$ tuples in Q.

The Join Operation. In fact, the Cartesian product is rarely used by itself. We can define several useful operations based on the Cartesian product operation. The join operation, denoted by $\bowtie$, is used to combine *related tuples* from two relations. The general form of the join operation on two relations $R1(A_1, A_2, \ldots, A_n)$ and $R2\ (B_1, B_2, \ldots, B_m)$ is

$$R1 \bowtie_{<\text{join condition}>} R2$$

The result of the join operation is a relation $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$ that satisfies the join condition. The join condition is specified on the attributes of $R1$ and $R2$ and is evaluated for each tuple from the Cartesian product $R1 \times R2$. The join condition is of the form:

$$<\text{condition}> \text{ AND } <\text{condition}> \text{ AND } \ldots \text{ AND } <\text{condition}>$$

where each condition is of the form $A_i \Theta B_j$, A_i is an attribute of $R1, B_j$ is an attribute of $R2$, both A_i and B_j belong to the same domain, and Θ is one of the following comparison operators: $=, \leq, <, >, \geq, \neq$. A join operation with such a general join condition is called a *theta join*.

The most common join operation involves only equality comparisons. When a join only involves join conditions with equality operators, such a join is called an *equijoin*. As an illustration of an equijoin join, the following query retrieves the names of all faculty members of each department in the *University* database:

$$\pi_{\text{name}}(\text{Faculty} \bowtie_{\text{Faculty.dcode}=\text{Department.dcode}} \text{Department})$$

The result of the equijoin is shown in Fig. 2. You may notice that we always have at least two attributes that have identical values in every tuple as shown in Fig. 2. As it is unnecessary to include repeated columns, we can define a *natural join* by including an equality comparison between every pair of common attributes and eliminating the repeated columns in the result of an equijoin. A natural join for the above example can be expressed as follows in Fig. 3.

Relational Calculus

Relational calculus is a declarative query language in which users specify what data should be retrieved, but not how to retrieve them. It is a formal language, based on *predicate calculus*, a branch of mathematical logic. There are two types of relational calculus: *tuple-related calculus* and *domain-related calculus*. Both are subsets of predicate calculus, which deals with quantified variables and values.

A query in the tuple-related calculus is expressed as

$$\{t|P(t)\}$$

which designates the set of all tuples t such that the predicate *P(t)* is true.

We may connect a set of predicates by the logical connectives AND ($\wedge$), OR ($\vee$), and NOT ($\sim$) to form compound predicates such as *P(t)* AND *Q(t)*, *P(t)* OR NOT *Q(t)*, and NOT *P(t)* OR *Q(t)*, which can be written as $P(t) \wedge Q(t)$, $P(t) \vee \sim Q(t)$, and $\sim P(t) \vee Q(t)$, respectively. A *conjunction* consists of predicates connected by a set of logical ANDs,

fid	*Faculty.name*	*birthdate*	*salary (dollar)*	*Faculty. dcode*	*Department.name*
91239876	John Smith	01/09/67	40000	2	Computer Science
33489783	Perry Lee	12/08/55	68000	2	Computer Science
78738498	Tom Bush	06/20/41	79000	4	Mechanical Engineering
12323567	Jane Rivera	09/15/72	35000	3	Electrical Engineering
87822384	Joyce Adams	11/10/59	54000	1	Chemical Engineering
78898999	Frank Wong	07/31/79	30000	3	Electrical Engineering
22322123	Alicia Johnson	08/18/80	28000	1	Chemical Engineering

Figure 3. Result of the natural join of *Faculty* and *Department*.

a *disjunction* consists of predicates connected by a set of logical ORs, and a *negation* is a predicate preceded by a NOT. For example, to retrieve all faculty members whose salary is above \$50,000, we can write the following tuple-related calculus expression:

$$\{t|Faculty(t) \wedge t.Salary > 50000\}$$

where the condition *Faculty(t)* specifies that the range of the tuple variable t is *Faculty* and each *Faculty* tuple t satisfies the condition $t.Salary > 50000$.

There are two quantifiers, the *existential quantifier* ($\exists$) and the *universal quantifier* ($\forall$), used with predicates to qualify tuple variables. An existential quantifier can be applied to a variable of predicate to demand that the predicate must be true for at least one tuple of the variable. A universal quantifier can be applied to a variable of a predicate to demand that the predicate must be true for all possible tuples that may instantiate the variable. Tuple variables without any quantifiers applied are called *free variables*. A tuple variable that is quantified by a $\exists$ or $\forall$ is called a *bound variable*.

In domain-related calculus, we use variables that take their values directly from individual domains to form a tuple variable. An expression in domain-related calculus is of the form

$$\{<x_1, x_2, \ldots, x_n>|P(x_1, x_2, \ldots, x_n)\}$$

where $<x_1, x_2, \ldots, x_n>$ represents domain variables and $P(x_1, x_2, \ldots, x_n)$ stands for a predicate with these variables. For example, to retrieve the names of all faculty members whose salary is above \$50,000, we can write the following domain-related calculus expression:

$$\{<n>|(\exists\, n)Faculty(f, n, b, s, d) \wedge s > 50000\}$$

where *f, n, b, s,* and *d* are variables created from the domain of each attribute in the relation *Faculty*, i.e., *fid, name, birthdate, salary*, and *dcode*, respectively.

Structured Query Language (SQL)

A formal language such as relational algebra or relational calculus provides a concise notation for representing queries. However, only a few commercial database languages have been proposed based directly on a formal database language. A Structured Query Language (SQL) is a high-level relational database language developed based on a combination of the relational algebra and relational calculus. SQL is a comprehensive database language that includes features for data definition, data manipulation, and view definition.

Originally, SQL was designed and implemented in a relational DBMS called SYSTEM R developed by IBM. In 1986, the America National Standards Institute (ANSI) published an SQL standard, called SQL86, and the International Standards Organization (ISO) adopted SQL86 in 1987. The U. S. Government's Federal Information Processing Standard (FIPS) adopted the ANSI/ISO standard. In 1989, a revised standard known commonly as SQL89 was published.

The SQL92 was published to strength the standard in 1992. This standard addressed several weaknesses in SQL89 as well as extended SQL with some additional features. The main updated features include session management statements; connection management, including CONNECT, SET CONNECTION, and DISCONNECT; self-referencing DELETE, INSERT, and UPDATE statements; subqueries in a CHECK constraint; and deferrable constraints.

More recently, in 1999, the ANSI/ISO SQL99 standard was released. The SQL99 (4,5) is the newest SQL standard. It contains many additional features beyond SQL92. This standard addresses some of the more advanced and previously nonaddressed areas of modern SQL systems, such as object-oriented, call-level interfaces, and integrity management.

Data Definition. The basic commands in SQL for data definition include CREATE, ALTER, and DROP. They are used to define the attributes of a table, to add an attribute to a table, and to delete a table, respectively. The basic format of the CREATE command is

CREATE TABLE table-name < attribute-name >: < attribute-type >: [< constraints >]

where each attribute is given its name, a data type defines its domain, and possibly some constraints exist.

The data types available are basic types including numeric values and strings. Numeric data types include integer number, real number, and formatted number. A String data type may have a fixed length or a variable length. There are also special data types, such as date and currency.

As SQL allows NULL (which means "unknown") to be an attribute value, a constraint NOT NULL may be specified on an attribute if NULL is not permitted for that attribute. In general, the *primary key* attributes of a table are restricted to be NOT NULL. The value of the primary key can identify a tuple uniquely. The same constraint can also be specified on any other attributes whose values are required to be NOT NULL. The *check clause* specifies a predicate P that must be satisfied by every tuple in the table. The table defined by the CREATE TABLE statement is called a *base table*, which is physically stored in the database. Base tables are different from *virtual tables* (*views*), which are not physically stored in the database. The following example shows how a *Faculty* table can be created using the above CREATE TABLE command:

```
CREATE TABLE Faculty
(fid: char (10) NOT NULL,
name: char(20) NOT NULL,
"birthdate": date,
salary: integer,
dcode: integer
PRIMARY KEY (fid),
CHECK (salary >= 0))
```

If we want to add an attribute to the table, we can use the ALTER command. In this case, the new attributes may have NULL as the value of the new attribute. For example, we can add an attribute SSN (Social Security Number) to the *Faculty* table with the following command:

```
ALTER TABLE Faculty
ADD SSN char(9)
```

If the *Faculty* table is no longer needed, we can delete the table with the following command:

```
DROP TABLE Faculty
```

Data Manipulation—Querying. SQL has one basic statement for retrieving information from a database: the SELECT statement. The basic form of the SELECT statement consists of three clauses: SELECT, FROM, and WHERE and has the following form:

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>
```

where the SELECT clause specifies a set of attributes to be retrieved by the query, the FROM clause specifies a list of tables to be used in executing the query, and the WHERE clause consists of a set of predicates that qualifies the tuples of the tables involved in forming the final result.

The followings are three example queries, assuming that the tables *Faculty*, *Department*, and *Membership* are defined as follows:

Faculty (fid, name, birthdate, salary, dcode)
Department (dcode, name)
Membership (fid, society)

Query 1. Retrieve the faculty IDs and names of all faculty members who were born on August 18, 1980.

```
SELECT fid, name
FROM Faculty
WHERE birthdate = '08/18/80'
```

Query 2. Retrieve the names of all faculty members associated with the Computer Science department.

```
SELECT Faculty.name
FROM Faculty, Department
WHERE Department.name = 'Computer Science' AND
  Faculty.dcode = Department.dcode
```

Query 3. Retrieve the faculty IDs and names of all faculty members who are members of any society of which Frank Wong is a member.

```
SELECT Faculty.fid, Faculty.name
FROM Faculty, Membership
WHERE Faculty.fid = Membership.fid AND
  society IN (SELECT society
    FROM Membership, Faculty
    WHERE Faculty.fid = Membership.fid AND
        Faculty.name = 'Frank Wong')
```

Note that Query 3 is a *nested query*, where the *inner query* returns a set of values, and it is used as an operand in the *outer query*.

Aggregate functions are functions that take a set of values as the input and return a single value. SQL offers five built-in aggregate functions: COUNT, SUM, AVG, MAX, and MIN. The COUNT function returns the number of values in the input set. The functions SUM, AVG, MAX, and MIN are applied to a set of numeric values and return the sum, average, maximum, and minimum, respectively. In many cases, we can apply the aggregate functions to subgroups of tuples based on some attribute values. SQL has a GROUP BY clause for this purpose. Some example queries are as follows:

Query 4. Find the average salary of all faculty members associated with the Computer Science department.

```
SELECT AVG (salary)
FROM Faculty, Department
WHERE Faculty.dcode = Department.code AND
  Department.name = 'Computer Science'
```

Query 5. For each department, retrieve the department name and the average salary.

```
SELECT Department.name, AVG(salary)
FROM Faculty, Department
WHERE Faculty.dcode = Department.dcode
GROUP BY Department.name
```

Sometimes it is useful to state a condition that applies to groups rather than to tuples. For example, we might be interested in only those departments where the average salary is more than $60,000, This condition does not apply to a single tuple but applies to each group of tuples constructed by the GROUP BY clause. To express such a query, the HAVING clause is provided, which is illustrated in Query 6.

Query 6. Find the departments whose the average salary is more than $60,000.

```
SELECT Department.name, AVG(salary)
FROM Faculty, Department
WHERE Faculty.dcode = Department.dcode
GROUP BY Department.name
HAVING AVG(salary) > 60000
```

Data Manipulation—Updates. In SQL, there are three commands to modify a database: INSERT, DELETE, and UPDATE. The INSERT command is used to add one or more tuples into a table. The values must be listed in the same order as the corresponding attributes are defined in the schema of the table if the insert column list is not specified explicitly. The following example shows a query to insert a new faculty member into the *Faculty* table:

```
INSERT Faculty
VALUES ('78965456', 'Gloria Smith', '12/12/81',
  25000, 1)
```

The DELETE command removes tuples from a table. It includes a WHERE clause to select the tuples to be deleted. Depending on the number of tuples selected by the condition in the WHERE clause, the tuples can be deleted by a single DELETE command. A missing WHERE clause indicates that all tuples in the table are to be deleted. We must use the DROP command to remove a table completely. The following example shows a query to delete those faculty members with the highest salary:

```
DELETE FROM Faculty
WHERE salary IN (SELECT MAX (salary) FROM Faculty)
```

Note that the subquery is evaluated only once before executing the command.

The UPDATE command modifies certain attribute values of some selected tuples. As in the DETELTE command, a WHERE clause in the UPDATE command selects the tuples to be updated from a single table. A SET clause specifies the attributes to be modified and their new values. The following example shows a query to increase by 10% the salary of each faculty member in the Computer Science department:

```
UPDATE Faculty
SET salary = salary * 1.1
WHERE dcode IN (SELECT code
    FROM Department
    WHERE name='Computer Science')
```

View Definition. A view in SQL is a table that is derived from other tables. These other tables can be base tables or previously defined views. A view does not exist in the physical form, so it is considered a virtual table in contrast to the base tables that physically exist in a database. A view can be used as a table in any query as if it existed physically. The command to define a view is as follows:

```
CREATE VIEW <view name>
AS <query statement>
```

The following example shows the definition of a view called *Young-Faculty-Members* who were born after 01/01/1970:

```
CREATE VIEW Young-Faculty-Members
AS SELECT name, birth-date
    FROM Faculty
    WHERE birth-date > '01/01/1970'
```

Object-Oriented Database Model and Languages

The object-oriented data model was developed based on the fundamental object-oriented concepts from a database perspective. The basic concepts in the object-oriented model include encapsulation, object identity, inheritance, and complex objects.

The object-oriented concepts had been first applied to programming. The object-oriented approach to programming was first introduced by the language, Simular67. More recently, C++ and Java have become the most widely known object-oriented programming languages. The object-oriented database model extends the features of object-oriented programming languages. The extensions include object identity and pointer, persistence of data (which allows transient data to be distinguished from persistent data), and support for collections.

A main difference between a programming language and a database programming language is that the latter directly accesses and manipulates a database (called *persistent data*), whereas the objects in the former only last during program execution. In the past, two major approaches have been taken to implement database programming languages. The first is to embed a database language such as SQL in conventional programming languages; these languages are called embedded languages. The other approach is to extend an existing programming language to support persistent data and the functionality of a database. These languages are called *persistent programming languages*.

However, the use of embedded languages leads to a major problem, namely *impedance mismatch*. In other words, conventional languages and database languages differ in their ways of describing data structures. The data type systems in most programming languages do not support relations in a database directly, thus requiring complex mappings from the programmer. In addition, because conventional programming languages do not understand database structures, it is not possible to check for type correctness.

In a persistent programming language, the above mismatch can be avoided. The query language is fully integrated with the host language, and both share the same type system. Objects can be created and stored in a database without any explicit type change. Also, the code for data manipulation does not depend on whether the data it manipulated are short-lived or persistent. Despite these advantages, however, persistent programming languages have some drawbacks. As a programming language accesses a database directly, it is relatively easy to make programming errors that damage the database. The complexity of such languages also makes high-level optimization (e.g., disk I/O reduction) difficult. Finally, declarative querying is, in general, not supported.

Object Database Management Group (ODMG). The Object Database Management Group (ODMG), which is a consortium of object-oriented DBMS vendors, has been working on standard language extensions including class libraries to C++ and Smalltalk to support persistency. The standard includes a common architecture and a definition for object-oriented DBMS, a common object model with an object definition language, and an object query language for C++, Smalltalk, and Java. Since ODMG published ODMG1.0 for their products in 1993, ODMG 2.0 and ODMG 3.0 were released in 1997 and 2000, respectively (6).

The components of the ODMG specification include an Object Model, an Object Definition Language (ODL), an Object Query Language (OQL), and language bindings to Java, C++, and Smalltalk:

ODMG Object Model is a superset of the Object Management Group (OMG) Object Model that gives it database capabilities, including relationships, extents, collection classes, and concurrency control. It is the unifying concept for the ODMG standard and is completely language-independent.

Object Definition Language (ODL) is used to define a database schema in terms of object types, attributes, relationships, and operations. The resulting schema can be moved from one database to another and is programming language-independent. ODL is a superset of OMG's Interface Definition Language (IDL).

Object Query Language (OQL) is an ODMG's query language. It closely resembles SQL99, and it includes support for object sets and structures. It also has object extensions to support object identity, complex objects, path expressions, operation invocation, and inheritance.

The *language bindings* to Java, C++, and Smalltalk are extensions of their respective language standards to allow the storage of persistent objects. Each binding includes support for OQL, navigation, and transactions. The advantage of such an approach is that users can build an entire database application from within a single programming language environment.

The following example shows the schema of two object types, *Faculty* and *Course*, in ODMG ODL:

```
interface Faculty (extent Faculties, key fid) {
    attribute integer fid;
    attribute string name;
    attribute birthdate date;
    attribute salary integer;
    relationship Set<Course> teachCourses
    inverse Course::faculties;
};
interface Course (extent Courses, key CID) {
    attribute string CID;
    attribute string title;
    relationship Set<Faculty>Faculties
    inverse Faculty::teachCourses;
};
```

where *interface* names an ODL description, *extent* names the set of objects declared, *key* declares the key, *attribute* declares an attribute, *set* declares a collection type, *relationship* declares a relationship, and *inverse* declares an inverse relationship to specify a referential integrity constraint.

The following example shows an example query in ODMG OQL to retrieve the course IDs and course titles of the courses instructed by John Smith:

```
SELECT f.teachCourse.CID, f. teachCourse.title
FROM Faculties f
WHERE f.name = 'John Smith';
```

Note that the FROM clause *f* refers to the extent *Faculties*, not the class *Faculty*. The variable *f* is used to range over the objects in *Faculties*. Path expressions using a dot (.) to separate attributes at different levels are used to access any property (either an attribute or a relationship) of an object.

Object-Relational Database Model and Languages

The object-relational data model extends the relational data model by providing extended data types and object-oriented features such as inheritance and references to objects. Extended data types include nested relations that support nonatomic data types such as collection types and structured types. Inheritance provides reusability for attributes and methods of exiting types. A reference is conceptually a pointer to tuples of a table. An object-relational database language extends a relational language such as SQL by adding object-oriented features. The following discussion is primarily based on the SQL99 standard (4,5):

Object Type. Objects are typed in an object-relational database. A value of an object type is an instance of that type. An object instance is also called an object. An object type consists of two parts: attributes and methods. Attributes are properties of an object and hold values at a given moment. The values of attributes describe an object's state. An attribute may be an instance of a declared object type, which can, in turn, be of another object type. Methods are invoked and executed in response to a message. An object type may have any number of methods or no method at all. For instance, we can define an object type *Address* as follows:

```
CREATE TYPE Address AS
    (street varchar(20),
    city varchar(20),
    state varchar(2))
METHOD change-address()
```

In the above, the object type *Address* contains the attributes *street*, *city*, and *state*, and a method *change-address()*.

Collection Type. Collection types such as sets, arrays, and multisets are used to permit mulivalued attributes. Arrays are supported by SQL99. The following attribute definition illustrates the declaration of a *Course-array* that contains up to 10 course names taught by a faculty member:

```
Course-array varchar(20) ARRAY [10]
```

Large Object Type. To deal with multimedia data types such as texts, audios, images, and video streams, SQL99 supports large object types. Two types of large objects (LOBs) can be defined depending on their locations: *internal* LOB and *external* LOB. Internal LOBs are stored in the database space, whereas external LOBs, or BFILEs (Binary FILEs), are stored in the file system that is outside of the database space. The following example shows the declaration of some LOB types:

```
Course-review CLOB (10KB)
Course-images BLOB (10MB)
Course-video BLOB (2GB)
```

Structured Type. In the relational model, the value of an attribute must be primitive as required by the first normal form. However, the object relational model extends the relational model so that the value of an attribute can be an object or a set of objects. For example, we may define an object type *Faculty* as follows:

```
create type Address AS
    (street varchar(20),
    city varchar(20),
    state varchar(2))
create type Name as
    (firstname varchar(20),
    middlename varchar(20),
    lastname varchar(20))
create type Faculty as
    (fid integer,
    name Name,
    address Address,
    Course-array varchar(20) array[10])
```

Type Inheritance. Inheritance allows an existing type's attributes and methods to be reused. Thus, it enables users to construct a type hierarchy to support *specialization* and *globalization*. The original type that is used to derive a new one is called a *supertype* and the derived type is called a *subtype*. For example, we can define two types *Graduate-student* and *University-staff* as two subtypes of *University-personnel* as follows:

```
CREATE TYPE University-personnel AS
    (name Name,
    address Address)

CREATE TYPE Graduate-student
UNDER University-personnel
    (student-id char(10),
    department varchar(20))

CREATE TYPE University-staff
UNDER University-personnel
    (years-of-experience: integer
    salary: integer)
METHOD compute-salary()
```

Both *Graduate-student* and *University-staff* types inherit the attributes from the type *University-personnel* (i.e., *name* and *address*). Methods of a supertype are inherited by its subtypes, just as attributes are. However, a subtype can redefine the method it inherits from its supertype. Redefining an inherited method allows the subtype to execute its own method. Executing its own method results in *overriding* an inherited method. For example, the type *Research-assistant* has its own method *compute-salary()*, which overrides *computer-salary()* defined in *Graduate-student*:

```
CREATE TYPE Research-assistant
UNDER University-staff, Graduate-student
METHOD compute-salary()
```

In the above example, the subtype *Research-assistant* inherits the attributes and methods defined in *University-staff* and *Graduate-student*. This is called *multiple inheritance*.

Reference Type. A reference can refer to any object created from a specific object type. A reference enables users to navigate between objects. The keyword REF is used to declare an attribute to be of the reference type. The restriction that the scope of a reference is the objects in a table is mandatory in SQL99. For example, we can include a reference to the *Faculty* object type in the *Graduate-student* object type as follows:

```
CREATE TYPE Graduate-student
UNDER University-personnel
    (student-id char(10),
    department varchar(20)
    advisor REF(Faculty) scope Faculty)
```

Here, the reference is restricted to the objects of the *Faculty* table.

Extensible Markup Language (XML)

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents, which are well formed (7). XML is a subset of Standard Generalized Markup Language (SGML) and was originally developed for document management in the World Wide Web (WWW) environment as was the Hyper-Text Markup Language (HTML). However, unlike SGML and HTML, XML is also playing an important role in representing and exchanging data.

XML can be used for *data-centric documents* as well as *document-centric documents*. In general, document-centric documents are used by humans and characterized by less regular structures and larger grained data. On the other hand, data-centric document are designed to facilitate the communication between applications. When XML is used for data-centric documents, many database issues develop, including mapping from the logical model to the XML data model, organizing, manipulating, querying, and storing XML data.

As other markup languages, XML takes the form of tags enclosed in a pair of angle brackets. Tags are used in pairs with <tag-name> and </tag-name> delimiting the beginning and the end of the *elements*, which are the fundamental constructs in an XML document. An XML document consists of a set of elements that can be nested and may have attributes to describe each element. The nested elements are called *subelements*. For example, we may have a nested XML representation of a university information document as follows:

```
<university>
    <faculty>
        <fid> 91239876 </fid>
        <name> John Smith </name>
        <birthdate> 01/09/67 </birthdate>
        <salary> 40000 </salary>
        <dcode> 2 </dcode>
    </faculty>
    <faculty>
        <fid> 33489783 </fid>
        <name> Perry Lee </name>
        <birthdate> 12/08/55 </birthdate>
        <salary> 68000 </salary>
        <dcode> 2 </dcode>
    </faculty>
</university>
```

XML Data Modeling. The mapping from the entity-relationship model to the XML data model is fairly straightforward. An entity becomes an *element*, and the attributes of an entity often become *subelements*. These subelements are sometimes consolidated into a complex type. Moreover, XML supports the concepts of collections including one-to-many and many-to-many relationships and references.

Like the mapping from the E–R model to the XML data model, mapping an XML schema to a relational schema is straightforward. An XML element becomes a table and it's attributes become columns. However, for an XML element that contains complex elements, key attributes must be considered depending on whether the element represents a one-to-many or many-to-many relationship.

XML Document Schema. The document type definition (DTD) language allows users to define the structure of a type of XML documents. A DTD specification defines a set of tags, the order of tags, and the attributes associated with each tag. A DTD is declared in the XML document using the !DOCTYPE tag. A DTD can be included within an XML document, or it can be contained in a separate file. If a DTD is in a separate file, say document.dtd, the corresponding XML document may reference it using:

```
<!DOCTYPE Document SYSTEM "document.dtd">
```

The following shows a part of an example DTD for a university information document:

```
<!DOCTYPE university [
        <!ELEMENT university (faculty)>
        <!ELEMENT faculty (fid name birthdate
            salary dcode)>
        <!ELEMENT fid (#PCDATA)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT birth-date (#PCDATA)>
        <!ELEMENT salary (#PCDATA)>
        <!ELEMENT dcode (#PCDATA)>
]>
```

where the keyword # PCDATA refers to text data and its name was derived from "parsed character data."

XML Schema (8) offers facilities for describing the structures and constraining the contents of XML documents by exploiting the XML *namespace* facility. The XML Schema, which is itself represented in XML and employs namespaces, substantially reconstructs and considerably extends the capabilities supported by DTDs. XML Schema provides several benefits: It provides user-define types, it allows richer and more useful data types, it is written in the XML syntax, and it supports namespaces. The following shows an XML Schema for a university information document:

```
<xsd:schema xmlns:xsd="http://www.w3.org/
  2001/XMLschema.xsd">
<xsd:element name="university" type=
  "UniveristyType">
<xsd:elemant name="faculty">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="fid" type="xsd:string"/>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="birthdate" type="xsd:string"/>
      <xsd:element name="salary" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="UniversityType">
  <xsd:sequence>
      <xsd:element ref="faculty" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

XML Query Language. The World Wide Web Consortium (W3C) has been developing XQuery (9), a query language of XML. XQuery uses the structure of XML and expresses queries across diverse data sources including structured and semi-structured documents, relational databases, and object repositories. XQuery is designed to be broadly applicable across many types of XML data sources.

XQuery provides a feature called a *FLWOR* expression that supports iteration and binding of variables to intermediate results. This kind of expression is often useful for computing joins between two or more documents and for restructuring data. The name *FLWOR*, pronounced "flower," came from the keywords *for*, *let*, *where*, *order by*, and *return*.

The *for* clause in a *FLWOR* expression generates a sequence of variables. The *let* clause allows complicated expressions to be assigned to variable names for simplicity. The *where* clause serves to filter the variables, retaining some result and discarding the others. The *order by* clause imposes an ordering on the result. The *return* clause constructs the result of the *FLWOR* expression.

A simple *FLWOR* expression that returns all faculty members whose salary is over $50,000 by checking the salaries based on an XML document is as follows:

```
for $x in /university/faculty/name
where $x/salary > 50000
return <fid>$x </fid>
```

where /university/faculty is a *path expression* (10) that shows a sequence of locations separated by "/". The initial "/" indicated the root of the document. The path expression /university/faculty/name would return <name>John Smith</name> <name>Perry Lee</name>. With a condition, $x/salary > 50000, the tuple <name>Perry Lee</name> is returned as the result for the above XQuery.

ADVANCED DATABASE MODELS AND LANGUAGES

Although relation, object-oriented, and object-relational databases have been used successfully in various areas, specialized databases are required for many applications. For example, in order to deal with temporal data, spatial data, and rules, we may need a temporal database, spatial database, and active database, respectively.

Temporal Database Models and Languages

Time is an important aspect of real-world applications. However, most nontemporal database models do not capture the *time-varying* nature of the problems they model. Nontemporal databases represent the current state of a database and their languages provide inadequate support for time-varying applications.

In general, a temporal database must support time points, time intervals, and relationships involving time such as *before*, *after*, and *during*. Temporal data models also need to represent time-varying information and time-invariant information separately. The temporal relational model (11) extends the relational model based on the above consideration. In this model, a database is classified as two sets of relations *Rs* and *Rt*, where *Rs* is the set of time-invariant relations and *Rt* is the set of time-varying relations. Every time-variant relation must have two time stamps (stored as attributes): time start (*Ts*) and time end (*Te*). An attribute value is associated with *Ts* and *Te* if it is valid in [*Ts, Te*].

TSQL2 is an extension of SQL92 with temporal constructs. TSQL2 allows both time-varying relations and time-invariant relations. Thus, SQL92 is directly applicable to time-invariant relations. TSQL2 has been proposed to be included as a part of SQL99. TSQL2 has a number of major temporal constructs as illustrated by the following example relation:

Faculty (fid, name, birthdate, rank, salary, Ts, Te)

where a tuple (f, n, b, *r*, s, Ts, Te) states the salary history of a faculty member whose name is n and birth-date is b.

For example, suppose that the following table stores Perry Lee's salary history:

fid	name	birth-date	rank	salary	Ts	Te
33489783	Perry Lee	12/08/55	Assistant	40000	01/01/91	12/31/97
33489783	Perry Lee	12/08/55	Associate	52000	01/01/98	12/31/02
33489783	Perry Lee	12/08/55	Full	68000	01/01/03	12/31/03

The following query retrieves Perry Lee's history in TSQL2;

Query: Retrieve Perry Lee's salary history.

```
SELECT salary
FROM Faculty
WHERE Name = 'Perry Lee'
```

Spatial Data Models and Languages

Spatial databases deal with spatial objects. For modeling spatial objects, the fundamental abstractions are *point*, *line*, and *region*. A point can be represented by a set of numbers. In a two-dimensional space, a point can be modeled as (x, y). A *line segment* can be represented by two endpoints. A *polyline* consists of a connected sequence of line segments. An arbitrary curve can be modeled as a set of polylines. We can represent a polygon by a set of vertices in order. An alternative representation of polygon is to divide a polygon into a set of triangles. The process of dividing a more complex polygon into simple triangles is called *triangulation*. Geographic databases are a subset of spatial databases created for geographic information management such as maps. In a geographic database, a city may be modeled as a point. Roads, rivers, and phone cables can be modeled as polylines. A country, a lake, or a national park can be modeled as polygons.

In nonspatial databases, the most common queries are exact match queries, in which all values of attributes must match. In a spatial database, approximations such as *nearness* queries and *region* queries are supported. Nearness queries search for objects that lie near a specified location. A query to find all gas stations that lie within a given distance of a car is an example of a nearness query. A *nearest-neighbor* query searches for the objects that are the nearest to a specified location. Region queries (range queries) search for objects that lie inside or partially inside a specified region. For example, a query to locate all the restaurants in a given square-mile area is a region query. Queries may also request intersections and unions of regions. Extensions of SQL have been proposed to permit relational databases to store and retrieve spatial information to support spatial queries as illustrated below (12):

Query: Find all gas stations within 5 miles from the current location of the car '1001'

```
SELECT g.name
FROM Gas AS g, Car AS c
WHERE c.id = '1001' and
distance (c.location , g.location) < 5
```

Table 1. Comparison of database languages

	Relational	Object-Relational	Object-Oriented	XML	Temporal	Spatial	Active
Structure	Flat table	Nested table	Class	XML document	Table with time	Table	Table with trigger
Query type	Declarative	Declarative	Procedural, Declarative	Declarative	Declarative	Declarative	Declarative
Language	SQL	SQL99	Persistent C++	XQuery	TSQL, SQL99	SQL99	SQL99
Optimization	System	System	User	System	System	System	System

ACTIVE DATABASE MODELS AND LANGUAGES

Conventional database systems are passive. In other words, data are created, retrieved, and deleted only in response to operations posed by the user or from application programs. Sometimes, it is more convenient that a database system itself performs certain operations automatically in response to certain events or conditions that must be satisfied. Such systems are called *active*. Typically, an active database supports (1) specification and monitoring of general integrity constraints, (2) flexible timing of constraint verification, and (3) automatic execution of actions.

A major construct in active database systems is the notion of *event-condition-action* (EAC) rules. An active database rule is triggered when its associated event occurs; in the meantime, the rule's condition is checked and, if the condition is true, it's action is executed. Typical triggering events include data modifications (i.e., insertion, deletion, and update), data retrieval, and user-defined events. The condition part of an ECA rule is a WHERE clause and an action could be a data modification, data retrieval, or call to a procedure in an application program. The following SQL-like statement illustrates the use of an ECA rule:

```
<EVENT>:       UPDATE Faculty
               Set salary = salary * 1.1
<CONDITION>:   salary > 100000
<ACTION>:      INSERT INTO High-paid-faculty-member
```

Several commercial relational database systems support some forms of active database rules, which are usually referred to as triggers. In SQL99, each trigger reacts to a specific data modification on a table. The general form of a trigger definition is as follows (13):

```
<SQL99 trigger>::= CREATE TRIGGER <trigger name>
                   {BEFORE|AFTER}<trigger event>
                   ON <table name>
                   [FOR EACH{ROW|STATEMENT}]
                   WHEN <condition>
                   <SQL procedure statements>
<trigger-event>::= INSERT|DELETE|UPDATE
```

where <trigger-event> is a monitored database operation, <condition> is an arbitrary SQL predicate, and <action> is a sequence of SQL procedural statements that are serially executed. A trigger may be executed BEFORE or AFTER the associated event, where the unit of data that can be processed by a trigger may be a tuple or a transaction. A trigger can execute FOR EACH ROW (i.e., each modified tuple) or FOR EACH STATEMENT (i.e., an entire SQL statement).

CONCLUSION

We have considered several modern database languages based on their underlying data models. The main criterion used to classify database languages is the data model on which they are defined. A comparison of the database languages discussed in this article is summarized in Table 1.

BIBLIOGRAPHY

1. A. Silberschatz, H. F. Korth, and S. Sudarshan, Data models, *ACM Comput. Surveys*, **28** (1): 105–108, 1996.
2. P. P. Chen, The entity-relationship model: Toward a unified view of data, *ACM Trans. Database Syst.*, **1** (1): 9–36, 1976.
3. E. F. Codd, A relational model for large shared data banks, *Commun. ACM*, **13** (6): 377–387, 1970.
4. *IBM DB2 Universal Database SQL Reference* Volume 1 Version 8, 2002.
5. *Oracle 9i Application Developer's Guide – Object-Relational Features*, Release 1 (9.0.1), June 2001, Part No. A88878-01.
6. ODMG3.0. Available: http://www.service-architecture.com/database/articles/odmg_3_0.html.
7. Extensible Markup Language (XML) 1.0. Available: http://www.w3.org/TR/1998/REC-xml-19980210#dt-xml-doc.
8. XML Schema Part 1: Structures. Available: http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/structures.xml.
9. XQuery 1.0: An XML Query Language. Available: http://www.w3.org/TR/xquery/.
10. XML Path Language (XPath) Version 1.0. Available: http://www.w3.org/TR/xpath.

11. T. Theory, *Database Modeling & Design*. San Francisco, CA: Morgan Kaufmann, 1999.

12. M. Stonebaker and P. Brown, *Object-Relational DBMSs – Tracking the Next Great Wave*. San Francisco, CA: Morgan Kaufmann, 1999.

13. J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules For Advanced Database Processing*. San Mateo, CA: Morgan Kaufmann, 1996.

FURTHER READING

R. Elmarsri and S. B. Navathe, *Fundamentals of Database Systems*. Menlo Park, CA: Benjamin/Cummings, 1994.

A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*. New York: McGraw-Hill, 2002.

M. Stonebraker and J. Hellerstein (eds.), *Readings in Database Systems*, San Francisco, CA: Morgan Kaufmann, 1998.

C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, V. Subrahmanian, and R. Zicari, *Advanced Database Systems*. San Francisco, CA: Morgan Kaufmann, 1997.

GEORGE (TAEHYUNG) WANG
California State University,
Northridge
Northridge, California
PHILLIP C-Y SHEU
University of California, Irvine
Irvine, California
ATSUSHI KITAZAWA
HIROSHI YAMAGUCHI
NEC Soft, Ltd.
Japan

DATA CLASSIFICATION

INTRODUCTION

To succeed in data classification, we have to begin to survey data and usually to preprocess it to have suitable forms of data to be input to a classification program. In the following discussion, a cavalcade of focal classification methods are briefly described and shown how they can be used for an actual classification problem stemmed from medical informatics. First, a dataset and its features are introduced because these naturally have influence on the preprocessing needed and on the opportunities to apply classification methods.

An ample variety of classification methods are available that use different approaches to group objects into different classes. Discriminant analysis, cluster analysis, nearest-neighbor searching, Bayesian rule, and decision trees have been applied (1). Later, new methods have been developed such as neural networks and support vector machines (1, 2).

In the following discussion, classification algorithms are not presented thoroughly to delimit the text, but a practical view is introduced for how they can be applied to a nontrivial dataset. Numerous literary sources are available for classification methods (1–4). Instead, fewer presentations are given to explain how to exploit them for real-world data.

VARIABLES AND DATA

Variables are also called attributes or features. Although a dataset can comprise one or more variable types, most datasets consist of one variable type, usually binary, integer, or real values. The statistical convention to categorize them is very suitable: binary, nominal, ordinal, interval, or ratio scales. Actually, binary variables are a special case of nominal variables, but here a variable is called nominal if it has more than two discrete categories. The difference in interval and ratio scales is that the latter has a genuinely fixed zero point, where the quantity represented by the variable disappears. Temperature can have different scales, but length and weight have exactly one constant zero value. For classification purpose, these two are recognized as one common type.

Data can include symbolic variables, but such variable values have to be coded with numeric values in some suitable way. If only two variable values or categories are used, then the binary type is selected. For example, for the replies of a question, two alternatives “no” and “yes”, are typically encoded with 0 and 1. For more alternatives, a nominal variable is used. For instance, for “black”, “brown”, and “white”, the categories would be 0, 1, and 2. Because these values are nominal, it is important to remember which statistics can be calculated for them. For example, the mode can be determined, but not the mean. If there is an order between them, like “no symptom”, “mild”, “strong”, and “severe”, the data are encoded as an ordinal variable of categories 0, 1, 2, and 3. If we measure some phenomenon with a device understood—it could even be a mere ruler—very widely, we obtain a recording of a sample or several samples. These values form a sample sequence, which is always discrete even if it describes some continuous phenomenon recorded typically in the course of time. Such samples are of interval or ratio scale types encoded with integer or real values. We can also name the variable types as qualitative, which are binary and nominal, and quantitative, which are the others.

It is straightforward to assume a dataset to include only integers or real values and then to apply different distance measures in their computation. However, what would a difference or distance be for “black”, “brown”, or “white”? When these values are encoded with 0, 1, and 2, a difference can “technically” be computed, but it would be meaningless. The consequences of this error may be difficult to observe. Therefore, it is important to employ proper distance measures. We call a difference to be distance, but dissimilarity is used, and it is the opposite of similarity. The latter naturally requires slightly modified equations, for instance,

$$s(x,y) = 1 - ds(x,\ y)$$

in which similarity s of instances x and y is defined to be equal to 1 if they are entirely similar (i.e., identical), and 0 if they are entirely different. Dissimilarity ds is defined in the opposite way. The minimum and maximum of a distance can likewise be other than 0 and 1 (e.g., 0 and 100).

Obviously, the best-known distance measure is Euclidean distance

$$D(x,y) = \sqrt{\sum_{i=1}^{n} d(x_i,y_i)^2}, d(x_i,y_i) = \frac{|x_i - y_i|}{R_i}$$

where the differences of n variables are computed for instances x and y and where R_i is the range (the minimum subtracted from the maximum) of variable i used to normalize the absolute difference. Normalization or scaling with R_i is often, but not always, important to set all variables to the same scale. Some classification methods can be sensitive to different scales and could give more influence on variables, (e.g., with the scale of [0,100] than for those of [0,1]). It is important to remember that this distance is inappropriate for nominal variables (except binary).

For a nominal variable, an appropriate method is to calculate differences between variables with the following formula:

$$d(x_i,y_i) = \begin{cases} 1, x_i \neq y_i \\ 0, x_i = y_i \end{cases}$$

This formula determines the minimum of 0 for identical values and the maximum of 1 for different values.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Several distance measures have been designed especially for binary data (2). Perhaps the Hamming distance is the simplest measure. Let element $n_{1,1}$ be equal to the number of binary variables that have value pair $i = 1$ and $j = 1$ (i.e., both are 1), $n_{0,0}$ equal to the number of $i = 0$ and $j = 0$, and $n_{1,0}$ and $n_{0,1}$ be the opposite cases. Hamming distance is defined as the number of variables with opposite values.

$$H(i, j) = n_{1,0} + n_{0,1}$$

Several distance measures of binary data are known. The following equations are given as exemplar similarity measures: Jaccard coefficient J and Dice coefficient Di.

$$J(i, j) = \frac{n_{1,1}}{n_{1,1} + n_{1,0} + n_{0,1}}$$

$$Di(i, j) = \frac{2n_{1,1}}{2n_{1,1} + n_{1,0} + n_{0,1}}$$

We often expect no missing value, which is not always self-evident. If a person is interviewed according to a list of questions, then it is possible that some questions receive no reply. The alternatives to treat such a situation in classification are either (1) to leave out such an instance or even the whole variable or (2) to estimate the missing value somehow (5). This preprocessing step is often necessary before classification. However, some classification methods like decision trees can be programmed to cope with data with missing values, but it can mean that the program does not use the variable with a missing value at all for some decision steps, which may partly lose decision information slightly. Several missing values is a problem. If a considerable quantity of the values of individual variables is missing, then these variables can be useless, and it can be best to abandon them. The significance or usefulness of variables for classification can be important, and it is reasonable to investigate with statistical methods like correlation and statistical tests. Algorithms are used to evaluate the usefulness of variables based on their values and distributions (4, 6). “Dead” variables rarely exist, which are constant for all instances. All dead variables can be discarded self-evidently, which also decreases dimensionality.

A possible way to handle missing values is to modify a distance measure that defines a maximum difference when one or both values are unknown or missing. This “pessimistic” method assumes the worst case (i.e., the farthest distance). For binary and nominal variables, it can be as follows:

$$d(x_i, y_i) = \begin{cases} 1, x_i \neq y_i \\ 1, x_i \text{ or } y_i \text{ unknown} \\ 0, x_i = y_i \end{cases}$$

For quantitative variables we could use the following definition:

$$d(x_i, y_i) = \begin{cases} \dfrac{|x_i - y_i|}{R_i} \\ 1, x_i \text{ or} y_i \text{ unknown} \end{cases}$$

where range R_i (the minimum subtracted from the maximum) of variable i is used as above.

A dataset may contain several types of variables. Distance measures are used for these variables. For instance, a heterogeneous Euclidean-overlap metric (7) uses the preceding $d(x_i,y_i)$ values and the above formula $D(x,y)$. In fact, this measure is not metric but pseudometric, because distance $D(a,a) = 0$ for the case of instance a is not true if it includes missing values. The other conditions required that a measure be a metric for instances a, b, and c are $D(a,b) > 0$, if $a \neq b$, $D(a,b) = D(b,a)$, and $D(a,c) \leq D(a,b) + D(b,c)$ (triangular inequality). The metric property is very useful for the purpose of distance computation; but even without it, we can come through, even with a more limited selection of computational methods.

A DATASET

In the following discussion, we consider a medical dataset of vertiginous patients to show how it is prepared for classification and how the methods selected classify its instances. Medical data are interesting, because their classification is often a complicated and elaborate task. Often, several different types of data are present in such a dataset. Vertigo or dizziness and other balance disorders are a common nuisance and can be symptoms of a serious disease. These problems are investigated in otoneurology (8) to find their causes, to devise new treatments, and to prevent possible accidents originated from such harms. For this purpose, computational classification methods can be used to identify a patient’s disease and to separate disease instances from each other.

The dataset consisted of 815 patient cases, whose diagnoses were made by two specialists in otoneurology. Six diseases appeared with frequencies as follows: 130 vestibular schwannoma cases (16%), 146 benign positional cases (18%), 313 Menière’s disease cases (38%), 41 sudden deafness cases (5%), 65 traumatic vertigo cases (8%), and 120 vestibular neuritis cases (15%). In all, 38 variables were discovered among patients’ symptoms, medical history, and clinical findings (Table 1). These variables were found to be the most important of a larger variable set in our research (8) and fairly infrequently contained missing values. Overall, 11% of values were missing in the whole dataset. They were imputed or substituted using modes for 11 binary variables and 1 nominal variable and using medians for 10 ordinal and 16 quantitative variables. The mean could have been applied to the quantitative variables, but as known from statistics, in principle, it is a poorer central tendency estimate under the circumstances of scarce data or biased-value distributions. The only genuine nominal (four-valued) variable [14] in Table 1 was still substituted by three binary variables to enable the use of a Euclidean measure in its strictest way. As mentioned, the Euclidean measure cannot be applied to nominal attributes. When its four values were “no hearing loss”, “sudden”, “progressive”, and “both disorders”, the last three corresponded to new three binary variables with value 1. If all of them were equal to 0, then it was the same as “no hearing loss” of the original variable. Thus, we had 40 variables altogether.

Table 1. Variables and their Types: B = Binary, N = Nominal, O = Ordinal, and Q = Quantitative; Category Numbers After the Nominal and Ordinal

Variable	Type	Variable	Type	Variable	Type
[1] patient's age	Q	[14] hearing loss type	N 4	[27] caloric asymmetry %	Q
[2] time from symptoms	O 7	[15] severity of tinnitus	O 4	[28] nystagmus to right	Q
[3] frequency of spells	O 6	[16] time of first tinnitus	O 7	[29] nystagmus to left	Q
[4] duration of attack	O 6	[17] ear infection	B	[30] pursuit eye movement amplitude gain %	Q
[5] severity of attack	O 5	[18] ear operation	B	[31] and its latency (ms)	Q
[6] rotational vertigo	Q	[19] head or ear trauma with noise injury	B	[32] audiometry 500 Hz right ear (dB)	Q
[7] swinging or floating vertigo or unsteadiness	Q	[20] chronic noise exposure	B	[33] audiometry 500 Hz left ear (dB)	Q
[8] Tumarkin-type drop attacks	O 4	[21] head trauma	B	[34] audiometry 2 kHz right (dB)	Q
[9] positional vertigo	Q	[22] ear trauma	B	[35] and left ear (dB)	Q
[10] unsteadiness outside attacks	O 4	[23] spontaneous nystagmus	B	[36] nausea or vomiting	O 4
[11] duration of hearing symptoms	O 7	[24] swaying velocity of posturography eyes open (cm/s)	Q	[37] fluctuation of hearing	B
[12] hearing loss of right ear between attacks	B	[25] swaying velocity of posturography eyes closed (cm/s)	Q	[38] lightheadedness	B
[13] hearing loss of left ear between attacks	B	[26] spontaneous nystagmus (eye movement) velocity (°/s)	Q		

Imputation was carried out class by class, it was important to follow class-wise variable distributions because it is essential that differences exist between classes, which is actually the ground of any classification. Using modes and medians in imputation is straightforward but sufficient in this case, because the number of the missing values was pretty small. We observed (10) that the more sophisticated techniques of linear regression and the Expectation-Maximization algorithm (11) gave no better results while using discriminant analysis for the classification of otoneurological data. Of course, such a property of missing values is not common. Each dataset has its own peculiarities that must be analyzed before classification (12).

Often, numbers that describe the central tendency, for example, modes for binary and nominal variables and medians or means for other variable types, are used to impute data. It is realistic to assume that the data are missing at random (11) (i.e., the missing values of a variable may depend on the values of other variables, but not on the values of the variable itself). Then likelihood-based methods, such as the Expectation-Maximization method, are appropriate. Linear regression and nearest-neighbor estimator (11) are also used for imputation.

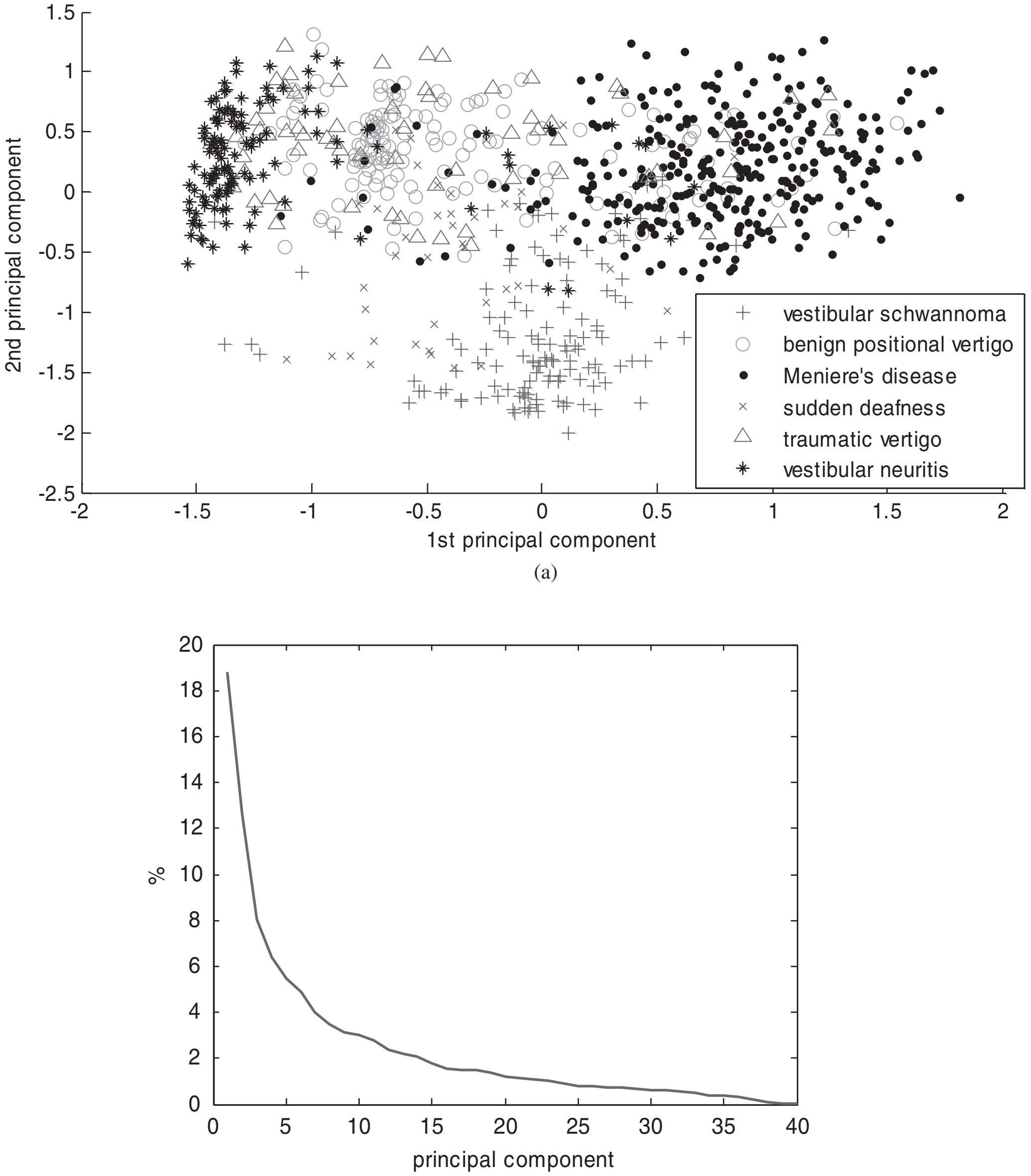
Outliers (1, 13) are data that are exceptional or inconsistent with other values of one or more variables. They are often some measurement or input errors. They can usually be revealed considering variable distributions. In the current dataset of vertiginous patients, 14 subjects were originally used, whose data included zero ages. This was observed on the basis of the distribution of age variable (1) (Table 1). Because it was surely known for the sake of medical reasons that the data could not contain ages below six years, these values were erroneous. If such values had not been corrected. then these patient cases should have been eliminated from the dataset.

The subset of Menière's disease was essentially larger than two small subsets of sudden deafness and traumatic vertigo. This imbalanced class distribution is an ordinary nuisance in actual datasets. It is a difficulty for some classification methods, at least for multilayer perceptron neural networks. Because neural networks are learning methods, they may learn majority classes and lose minority classes.

Principal component analysis (1, 2, 13, 14) is often used to reduce a high dimensionality. It can be also applied to visualize data distribution crudely between different classes. It can be understood as a rotation of the axes of the original coordinate system to a new system of orthogonal axes ordered in terms of the magnitude of variation that they account for in the original data. Forty principal components were computed from the original data. The most important two components are shown in Fig. 1(a), which indicates that the six disease classes are more or less divided into separate areas, which is essential for classification. Their importance was calculated on the basis of their eigenvalues [Fig. 1(b)], which explained 31.5% of data variance with the two first principal components. The principal components slowly decreased down to almost zero while advancing to the fortieth component. Including no clear abrupt decrease between the successive eigenvalues, this indicated that it was not sensible to reduce the dimensionality of the data, but almost all 40 components included useful information.

CLASSIFICATION METHODS AND THEIR TESTING

The following methods are examined and compared with the given dataset: Bayesian decision rule, k–nearest-neighbor searching, discriminant analysis, k-means clustering, decision trees, multilayer perceptron neural networks, Kohonen networks (self-organizing maps), and support-vector machines. All implementations were programmed with Matlab using various values of control parameters like numbers of clusters or network hidden nodes.



(a)

(b)

Figure 1. (a) The data distribution of the vertiginous patient cases into six classes according to the two first principal components. (b) Eigenvalue spectrum (in percents) of 40 principal components as the ratio between each eigenvalue and the sum of all eigenvalues.

Maybe the simplest classification method is based on conditional probabilities determined by Bayesian rule

$$p(c_{k|}\,\mathbf{x}) = \frac{p(\mathbf{x}|c_k)p(c_k)}{\sum\limits_{i=1}^{k} p(\mathbf{x}|c_i)p(c_i)}$$

in which the formula estimates the probability of class c_k given instance (vector of variables) $\mathbf{x}$. Altogether, there are K classes whose a priori probabilities $p(c_i)$ are known or estimated from data. Actually, the formula is based on finding all identical instances in each class and counting the numbers of such identical instances. Unfortunately, there were several variables (40), and it was then very improbable that there would be any identical instances. An additional difficulty was that there were some real (continuous) variables that make it virtually impossible to find any identical instances. Thus, there were none in our dataset.

We can sometimes attempt to bypass the preceding problem by means of discretization of quantitative variables. Although it is not important in the current context, it can be useful for such classification or other data-analysis methods that provide discrete integer values. Several algorithms for this purpose can divide the range of a variable into uniform intervals (e.g., 10 intervals). More sophisticated techniques function on the basis of the value distribution of a variable (6). We could apply another way, which is often a good and reliable choice: the use of expert knowledge. Two experienced otoneurologists defined 2–5 suitable values for the quantitative variables in Table 1: [1], [6], [7], [9], [24]–[35]. For instance, they discretised the audiometric variables of [32]–[35] with 5 values: 0 for hearing decrease of 0–15 dB (normal hearing), 1 for 16–30 dB (slight hearing loss), 2 for 31–60 dB (moderate hearing loss), 3 for 61–90 dB (severe hearing loss), and 4 for 91–120 dB (deaf). The specialists' discretization is not possible if hundreds or more variables are used.

Notwithstanding discretization, no identical instances were encountered in the current dataset. This result is understandable because there were 40 variables. If all the variables were binary (i.e., including the minimum of categories per variable), then there would be $2^{40} = 1024^4$ different permutations of variable values. Consequently, identical instances would be very improbable; thus, we gave up the current discretization and returned to the original data of 40 variables and applied the Bayesian rule (13, 14) in a more sophisticated manner as follows. Let us assume that a normal distribution underlies every variable. We computed the Bayesian decision rule for the q-dimensional ($q = 40$) Gaussian (normal) distribution (3), which gives the probability for instance vector $\mathbf{x}$ provided class $c_j, j = 1, \ldots, 6,$ when mean $\mathbf{s}_j$ and covariance matrix $\sum_j$ were computed from the instances of class j of a training set.

$$p(\mathbf{x}|c_j) = 2\pi^{-\frac{q}{2}} \left|\sum_j\right|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{s}_j)^T \sum_j^{-1}(\mathbf{x}-\mathbf{s}_j)\right)$$

Let $p(c_j)$ be the probability of class j. The Bayesian decision rule then predicted class j for test case $\mathbf{x}$ if

$$p(\mathbf{x}|c_j) > \frac{p(c_k)}{p(c_j)}\, p(\mathbf{x}|c_k)$$

for all $k \neq j, k$ in $\{1, \ldots, 6\}$.

Nonetheless, data matrices required by Bayesian rule were singular for the current dataset because they incorporated such variables that included purely zeroes for some classes (not for the whole data). These crucial variables were impossible to eliminate. Singular matrices would rule out their inversions required. If no singular matrices occur that highly depend on data, the naïve Bayesian method can give good results.

A crude way is to ignore dependence between variables, which is not made above. If we assume them to be independent, then we obtain the following joint probability.

$$p(\mathbf{x}|c_j) = p(x_1, \ldots, x_n|c_j) = p(c_j)\prod_{i=1}^{n} p(x_i|c_j)$$

Independent conditional probabilities $p(x_i|c_j)$ can be estimated from data. Because this computation includes no iterative training algorithm, its execution is normally far faster than those of machine learning methods that will be dealt with subsequently. By allowing one or more dependences between variables, frequently efficient Bayesian networks (1) can be formed. However, these networks are skipped here.

To test classification methods, we need training and testing sets that are independent of each other but extracted from the same data source. If such a data source encompassed virtually almost unlimited number of data, we would extract many separate training and test sets. A training set is used to train a classification algorithm programmed, and the corresponding test set is used to test its capability to classify correctly. Several training and test-set pairs are necessary to justify statistically a final conclusion about the capability of the method. Inasmuch as we survey actual datasets, the preceding ideal circumstances are infrequently met. Therefore, we have to use more complicated techniques to infer statistical evidence about tests. One common way is to employ cross-validation.

We divided our dataset of N instances into 10 pairs of a learning subset (90% of instances) and a test subset (10%) in accordance with 10-fold cross-validation, so that every instance was incorporated in exactly one test set. The selection into the subsets was performed randomly, but in accordance with the class distribution. In other words, when the frequency of Menière's disease was 38% in our dataset, the same approximate proportion was preserved in all training and test sets. Most classification methods involve random initializations. Therefore, 10 runs for each crossvalidation pair and 100 runs altogether were repeated with an exception. Two test and training set pairs were discarded for discriminant analysis, because they consisted of matrices, which were not positive definitive (i.e., their processing could not be continued). Accordingly, 80 runs were executed for it. The same pairs of training and test sets were applied to all classification methods to be presented.

Other ways to test with a restricted dataset are leave-one-out and bootstrapping (4). In the former, each instance is used alternately as a single test set, and the other instances are used as the corresponding training set. In a way, this method is the extreme N-fold cross-validation situation, where the size of every training set is maximal in accordance with the principle that such a training set is statistically as representative as possible. The latter efficiently exploits random sampling, in which extracted instances are returned to the dataset for the later process, (i.e., they are replaced, which is not done in cross-validation).

Means and standard deviations can be calculated for sensitivity, specificity, positive predictive value, and accuracy. Let us define *tp* as the number of true positive classifications, *fn* the number of false-negative classifications, *fp* as the number of false-positive classifications, and *tn* as the number of true negative classifications. Their sum is naturally equal to the whole number N of instances. The term "positive" now means that some certain disease class is predicted to an instance. It is either true or false, depending on the matter whether this instance really belongs to the predicted class. The term "negative" means that an instance is classified into some other than the certain class, either correctly or incorrectly.

Table 2. Means and Standard Deviations of Nearest-Neighbor Searching (%) for 100 Test Runs

Number of Neighbors	Static	Vestibular Schwannoma	Benign Positional Vertigo	Menière's Disease	Sudden Deafness	Traumatic Vertigo	Vestibular neuritis
1-nearest	Sensitivity	88.5 ± 9.8	80.7 ± 19.7	87.6 ± 6.5	92.5 ± 12.1	82.6 ± 19.7	86.7 ± 13.1
	Accuracy	96.4 ± 1.6	91.8 ± 3.6	90.4 ± 4.8	99.4 ± 1.0	97.8 ± 2.2	96.6 ± 2.1
3-nearest	Sensitivity	89.2 ± 10.4	77.3 ± 23.6	87.9 ± 5.6	92.5 ± 12.1	78.8 ± 16.5	84.2 ± 13.9
	Accuracy	96.0 ± 2.5	91.2 ± 4.3	89.6 ± 5.2	99.4 ± 1.0	97.6 ± 2.4	96.5 ± 2.5
5-nearest	Sensitivity	79.2 ± 16.2	83.3 ± 18.9	89.5 ± 5.8	90.5 ± 12.3	80.2 ± 16.4	90.0 ± 11.7
	Accuracy	96.4 ± 2.7	92.2 ± 3.4	89.1 ± 6.0	99.4 ± 0.9	97.9 ± 1.7	97.1 ± 2.2
weighted	Sensitivity	88.5 ± 10.4	77.3 ± 23.6	87.9 ± 5.6	92.5 ± 12.1	84.5 ± 15.2	84.2 ± 13.9
3-nearest	Accuracy	95.8 ± 2.7	91.6 ± 4.3	89.5 ± 5.2	99.4 ± 1.0	98.0 ± 2.3	96.5 ± 2.5
weighted	Sensitivity	79.2 ± 16.2	82.7 ± 19.4	90.1 ± 6.1	92.5 ± 12.1	83.1 ± 15.8	90.8 ± 11.4
5-nearest	Accuracy	96.5 ± 2.6	92.5 ± 3.3	89.6 ± 5.3	99.5 ± 0.9	98.0 ± 1.7	97.1 ± 2.2

Sensitivity is given as the ratio:

$$sens = \frac{tp}{tp + fn} 100\%$$

Specificity is defined as the formula:

$$spec = \frac{tn}{tn + fp} 100\%$$

Positive predictive value is equal to:

$$ppv = \frac{tp}{tp + fp} 100\%$$

Accuracy is as follows:

$$a = \frac{tp + tn}{tp + tn + fp + fn} 100\% = \frac{tp + tn}{N} 100\%$$

In the following discussion, sensitivity and accuracy are given to delimit the number of results presented. Other validation quantities are applied to testing, but many of them, [e.g., negative predictive value, receiver operating characteristic (ROC) curves, and area under ROC curves (AUC)], are related to the aforementioned basic quantities. In addition, some quantities are unsuitable for multiclass circumstances; instead, they are meant for two-class classification. These terms are used commonly in medical informatics. Elsewhere various terms can be used, often for the same concepts. For instance, recall and precision are used. The former is equivalent to sensitivity, and the latter is equal to positive predictive value.

Nearest-neighbor searching techniques (1, 2, 13, 14) are straightforward. They are heuristic, which means that nothing guarantees that a correct solution is found. Despite this theoretical concept, they can function efficiently in practice. They search for the nearest instance from the training set and label a test instance according to the class of the nearest neighbor. Neighbors can be weighted, for example, using the inverse of the distance between the test instance and the near instance next to be processed. The nearer the training case is to the test case, the greater weight is given it. We computed tests of nearest-neighbor searching for one, three, and five nearest neighbors using Euclidean metric. Their average results are presented in Table 2. An increase of nearest neighbors k did not improve otherwise satisfactory results, which is possible depending on the data. The results of vestibular schwannoma and benign positional vertigo varied depending on k, which represents that their instances were mixed erroneously between the two classes. The weighted nearest-neighbor searching produced similar results. When the accuracy value of each class was weighted with its number of the instances, mean weighted accuracies were as follows: 93.5 ± 2.3, 93.0 ± 2.9 and 93.2 ± 2.8% for k equal to 1, 3, and 5, and 93.1 ± 2.9 and 93.5 ± 2.8% for the weighted versions of k equal to 3 and 5. When all means are considerably high and standard deviations are small, these examples show how this simple method classified this complicated data.

In discriminant analysis (13, 14), the aim is to split the variable space including all instances so that its parts represent the different classes. Bounds between such parts discriminate the classes to separate regions. The simplest choice is linear discriminant analysis, which is also run for the current dataset by applying the Euclidean metric. Its results are characterized in Table 3. This analysis was exceptionally successful for the class of sudden deafness, which can be hard to distinguish medically because it is the smallest class. The other classes were also detected successfully. A mean weighted accuracy of 95.5 ± 1.8% was obtained.

Table 3. Means and Standard Deviations of Linear Discriminant Analysis (%) for 80 Test Runs

Static	Vestibular Schwannoma	Benign Positional Vertigo	Menière's Disease	Sudden Deafness	Traumatic Vertigo	Vestibular Neuritis
Sensitivity	87.5 ± 8.6	87.5 ± 15.5	89.6 ± 4.8	100.0 ± 0.0	90.5 ± 13.3	95.8 ± 8.4
Accuracy	97.5 ± 1.5	93.2 ± 2.4	93.4 ± 3.1	99.5 ± 0.6	98.8 ± 1.2	98.2 ± 1.7

Table 4. Means and Standard Deviations of k-means Clustering (%) for 100 Test Runs

Number k of Clusters	Static	Vestibular Schwannoma	Benign Positional Vertigo	Menière's Disease	Sudden Deafness	Traumatic Vertigo	Vestibular Neuritis
6	Sensitivity	81.2 ± 14.2	66.2 ± 33.4	92.8 ± 7.3	12.0 ± 32.7	53.8 ± 44.4	84.8 ± 20.2
	Accuracy	93.7 ± 3.2	88.1 ± 6.0	88.0 ± 6.7	95.5 ± 1.6	95.3 ± 3.4	94.9 ± 5.1
8	Sensitivity	80.5 ± 15.0	70.1 ± 29.9	93.1 ± 6.5	41.5 ± 48.4	78.1 ± 30.2	86.9 ± 14.6
	Accuracy	94.6 ± 3.2	91.4 ± 5.0	88.3 ± 6.2	97.0 ± 2.3	97.0 ± 3.1	96.2 ± 4.0
12	Sensitivity	80.8 ± 13.1	71.7 ± 28.5	93.3 ± 6.2	68.3 ± 40.0	87.2 ± 16.3	86.3 ± 14.4
	Accuracy	95.7 ± 2.7	92.5 ± 4.9	88.7 ± 6.2	98.2 ± 2.1	97.8 ± 2.6	96.4 ± 3.2
20	Sensitivity	81.2 ± 13.3	74.9 ± 26.0	91.2 ± 8.1	88.8 ± 17.5	85.7 ± 18.4	87.2 ± 13.6
	Accuracy	96.0 ± 2.4	92.3 ± 5.4	88.6 ± 6.4	99.3 ± 1.0	97.9 ± 2.2	96.8 ± 2.2
30	Sensitivity	83.0 ± 13.9	74.8 ± 25.4	90.3 ± 9.0	87.5 ± 17.9	83.9 ± 17.9	85.8 ± 13.5
	Accuracy	96.4 ± 2.5	91.4 ± 5.8	88.7 ± 6.5	99.2 ± 1.0	97.6 ± 2.6	96.7 ± 2.5

By attempting to apply Mahalanobis (generalized Euclidean metric) or quadratic discriminant function, discriminant analysis suffered from the similar complication of unsatisfying the requirement of positive-definite matrices as the naïve Bayesian rule.

Clustering (1, 2, 13, 14) is an old technique to group instances by computing distances between instances and assigning them to separate groups or clusters on the basis of the mutual distances between clusters and between instances. Many ways can be used to compute distances between clusters and between instances (1, 3). Table 4 introduces results yielded by k-means clustering with Euclidean metric; from the results, several values of k five alternatives are given. Here, as with most other methods, more variations were computed to find proper limits of classifier structures, for instance, for k in clustering. We present conditions with small system-parameter values of k here and then increase it to reasonably large values related to the methods and to the numbers of instances and variables. The minimum of the clusters was six, which is the same number of the classes. Apart from sudden deafness, the number k of clusters between 12 and 40 produced equally high average results. For sudden deafness, 20 clusters sufficed to evolve sensitivities up to 89%. Benign positional vertigo remained the poorest recognized class here. Mean weighted accuracies of 90.9 ± 3.8, 92.8 ± 3.5, 92.9 ± 3.6, and 92.8 ± 3.7% were achieved for k equal to 6, 12, 20, and 30.

A decision tree (2, 12) selects a variable, which, at the current stage of process, best divides instances between classes and sets the variable to the next node built in the tree. Such an evaluation of variables is based on various principles of information theory. We exploited the pruning leaves of decision trees to estimate the best tree size according to residual variance. The best size gained was 36.2 leaves on average. As to the current data, decision trees were not effective for benign positional vertigo and especially sudden deafness (Table 5), although their variances were large (i.e., some trees were good but others were poor). This evaluation stemmed from a special situation that variable [14] named 'hearing loss type' in Table 2; is this variable crucial for sudden deafness. Thus, decision trees often "raised" it high in their process, in which variables are chosen according to criteria based on their information theoretic values. Unfortunately, in this specific data, lots of values were missing for this variable, in which for the six diseases 88%, 54%, 59%, 15%, 34%, and 62% of the values were absent. Altogether, 53% of all values of variable (14) were missing. Although sudden deafness was the best of all with only 15% of data missing, the others suffered from too many missing values. Note that the missing values were imputed, but because of their high numbers, most instances received the same mode within each class. In this method, the variable did not aid separate instances successfully. Therefore, decision trees occasionally misclassified data, whereas other methods were not so dependent on single variables, because their computation approaches were different. A mean weighted accuracy of 89.4 ± 2.5% was gained for the decision trees.

Multilayer percepton networks (12–17) could not classify the dataset tending to put most cases to the largest Menière's disease class and to lose the two smallest classes entirely. Hence, principal component analysis (1) was first computed as described at the end of the third section, using transformed values according to all 40 principal components as input data. Its main role is to reduce the dimensions of highly variable numbers. In a way, via its coordinate system, transform data are rearranged to a more appropriate form. Here, the idea was not to decrease the number of variables, but to use its rearrangement. Every perceptron network comprised 40 input nodes for principal components and 6 output nodes after the diseases. The only free parameter in the network topology was the number of hidden nodes. Adaptive learning and momentum coefficient were used with the backpropagation training algorithm, which was to run no more than 200 epochs to

Table 5. Means and Standard Deviations of Decision Trees (%) for 100 Test Runs

Static	Vestibular Schwannoma	Benign Positional Vertigo	Menière's Disease	Sudden Deafness	Traumatic Vertigo	Vestibular Neuritis
Sensitivity	72.7 ± 22.4	63.8 ± 32.5	87.4 ± 10.7	43.6 ± 39.4	81.1 ± 17.3	80.1 ± 17.0
Accuracy	95.3 ± 3.5	89.0 ± 4.5	82.6 ± 4.8	94.9 ± 2.4	96.6 ± 2.1	95.4 ± 3.1

Table 6. Means and Standard Deviations of Multilayer Perceptron Networks (%) for 100 Test Runs

Number of Hidden Nodes	Static	Vestibular Schwannoma	Benign Positional Vertigo	Menière's Disease	Sudden Deafness	Traumatic Vertigo	Vestibular Neuritis
4	Sensitivity	86.5 ± 15.7	77.6 ± 21.3	91.9 ± 5.9	32.8 ± 47.0	58.6 ± 43.2	87.5 ± 20.3
	Accuracy	95.2 ± 4.5	91.6 ± 5.1	90.9 ± 5.9	96.4 ± 2.2	95.3 ± 3.1	94.9 ± 4.2
6	Sensitivity	89.3 ± 9.6	79.0 ± 20.8	91.53 ± 5.5	93.2 ± 23.4	86.9 ± 16.7	91.9 ± 10.9
	Accuracy	97.7 ± 1.8	93.0 ± 4.1	91.7 ± 5.4	99.3 ± 1.2	98.2 ± 1.6	97.3 ± 2.4
10	Sensitivity	89.2 ± 9.0	80.3 ± 18.6	91.8 ± 5.7	99.8 ± 2.5	88.9 ± 12.9	92.8 ± 10.6
	Accuracy	97.7 ± 1.5	93.5 ± 4.0	92.3 ± 5.3	99.7 ± 0.5	98.4 ± 1.4	97.6 ± 2.0
16	Sensitivity	89.4 ± 8.7	80.1 ± 18.3	91.5 ± 5.6	99.8 ± 2.5	89.4 ± 12.9	92.8 ± 10.7
	Accuracy	97.7 ± 1.4	93.3 ± 3.8	92.2 ± 5.2	99.6 ± 0.6	98.5 ± 1.3	97.6 ± 2.1

Table 7. Means and Standard Deviations of Kohonen Networks (%) for 100 Test Runs

Number of nodes	Static	Vestibular Schwannoma	Benign Positional Vertigo	Menière's Disease	Sudden Deafness	Traumatic Vertigo	Vestibular Neuritis
3 × 3	Sensitivity	78.5 ± 17.2	66.1 ± 27.2	94.1 ± 6.6	32.2 ± 28.5	13.9 ± 27.6	84.5 ± 16.5
	Accuracy	94.3 ± 2.8	86.6 ± 4.9	87.2 ± 7.0	93.9 ± 2.3	91.6 ± 2.9	96.2 ± 2.8
5 × 5	Sensitivity	82.2 ± 14.1	71.4 ± 27.3	90.1 ± 7.1	57.4 ± 32.3	80.0 ± 19.2	82.8 ± 15.4
	Accuracy	95.1 ± 2.5	91.1 ± 5.0	88.2 ± 5.8	96.3 ± 2.4	96.8 ± 2.4	96.1 ± 2.7
8 × 8	Sensitivity	86.2 ± 12.9	74.6 ± 24.5	88.1 ± 7.7	79.4 ± 26.6	84.8 ± 18.2	84.6 ± 15.4
	Accuracy	95.9 ± 2.5	91.5 ± 5.2	88.7 ± 5.9	98.4 ± 1.8	97.7 ± 2.3	96.4 ± 2.6
10 × 10	Sensitivity	84.3 ± 14.0	75.1 ± 24.4	87.4 ± 7.6	83.7 ± 18.8	85.5 ± 17.4	85.4 ± 14.6
	Accuracy	95.8 ± 2.5	91.3 ± 5.0	88.8 ± 5.6	98.6 ± 1.3	97.9 ± 2.2	96.7 ± 2.4

prevent overlearning. Overlearning means learning continued too long so that the weights of the network start to adapt to random noises present in the data. Tests were conducted using 4–16 hidden nodes. A recommendation states that the number of connections (weights) in a perceptron neural network should not be greater than one tenth of a training set, so that efficient learning would succeed (1, 17). Nevertheless, the networks with 6–16 hidden nodes produced valid results (Table 6). Benign positional vertigo was the most difficult to detect. Mean weighted accuracies of 92.9 ± 3.5%, 94.9 ± 2.9%, 95.0 ± 3.0%, and 95.0 ± 2.9% were achieved for 4, 6, 10, and 16 hidden nodes.

Kohonen neural networks (13–17) were run varying their sizes from 3 × 3 to 10 × 10 nodes. A hexagonal neighborhood pattern was employed as the topological structure with link distance. For every network, 400 learning epochs were completed. According to the results, 7 × 7 nodes were sufficient for the current data, and data as small as 5 × 5 nodes were available for the other than sudden deafness. Table 7 includes a breakdown for four network sizes. Benign positional vertigo and sudden deafness obtained slightly poorer results compared with the remaining four classes. Mean weighted accuracies of 90.3 ± 3.7%, 92.1 ± 3.5%, 92.7 ± 3.3%, and 92.7 ± 3.3% were attained for these networks.

Support vector machines (1, 13) map data nonlinearly to a multidimensional space where it can be linearly classified. Kernel functions are used to decrease the number of computation needed in optimization. We employed two-degree radial basis and polynomial functions as kernels (Table 8). Their mean weighted accuracies were 93.8 ± 2.3% and 93.7 ± 2.3%. Linear and Gaussian kernels did not facilitate reasonable outcomes.

The running time (100 tests with a 2.6-GHz processor) of discriminant analysis was approximately 10 seconds. They were 23–104 seconds for clusterings, 68–90 seconds for nearest-neighbor searchings, 5 minutes 30 seconds for decision trees and multilayer perceptron networks, 23–29 hours for Kohonen networks and 2–3 minutes for support vector machines. Because these were slow Matlab programs and not optimized at all, running times could naturally be decreased. Although Kohonen networks were very slow, it is worth noting that the training phase took most of the time. In classification, however, training is seldom necessary in actual applications, because a trained classifier is run mostly to classify, not to test its effectiveness.

Table 8. Means and Standard Deviations of Support Vector Machines (%) for 100 Test Runs

Type	Static	Vestibular Schwannoma	Benign Positional Vertigo	Menière's Disease	Sudden Deafness	Traumatic Vertigo	Vestibular Neuritis
Radial basis	Sensitivity	86.9 ± 8.5	80.7 ± 17.8	92.7 ± 3.8	92.5 ± 11.5	80.2 ± 18.1	83.3 ± 14.0
	Accuracy	96.9 ± 1.3	94.3 ± 2.6	89.9 ± 4.0	99.4 ± 0.6	97.3 ± 2.5	96.6 ± 2.2
Polynomial	Sensitivity	83.8 ± 11.2	79.3 ± 19.8	94.2 ± 3.8	92.5 ± 11.5	83.6 ± 17.9	82.5 ± 14.7
	Accuracy	96.6 ± 1.7	94.5 ± 3.1	89.4 ± 3.9	99.4 ± 0.6	97.9 ± 2.3	96.6 ± 2.2

CONCLUSION AND SUMMARY

By adopting the best results from Tables 2–8, simple linear discriminant analysis was the best method by correctly classifying with the mean weighted accuracy of 95.5%, and decision trees yielded the poorest results of 89.4%. Statistically significant differences between the most of the methods were found according to t test (0.001 significance level). The results of individual disease classes varied between the methods and between the classifier structures (table rows). The results typically become more reliable when a structure was extended. Such an increase cannot be continued further, since the size of training sets was limited. Naturally, results always depend on data. Another dataset could result in different outcomes. No general conclusion between the methods could be drawn according to the computations of merely one dataset. Their motivation was to demonstrate actual applications.

Methods other than multilayer perceptron networks did not benefit from using principal components as input because the results were very reliable. For linear discriminant analysis, the results were identical independent of this preprocessing for the sake of its linear character as is also in principal component analysis.

This article focused on a practical approach to how classification methods can be applied to a complicated dataset. Data preprocessing was emphasised because this sector is often neglected or ignored, which, nevertheless, is crucial for successful classification.

To observe the classification algorithms in detail, the attached book references are a good start.

ACKNOWLEDGMENTS

The author is grateful to Docent Erna Kentala, M.D., and Prof. Ilmari Pyykkö, M.D., for the data and medical information. The dataset was collected and screened at the Department of Otorhinolaryngology, Helsinki University Central Hospital, Finland.

BIBLIOGRAPHY

1. A.R. Webb, *Statistical Pattern Recognition*, 2nd ed. Chichester, England: Wiley, 2002.
2. K.J. Cios, W. Pedrycz, R.W. Swiniarski, and L.A. Kurgan, *Data Mining, A Knowledge Discovery Approach*, New York: Springer Science+Business Media, 2007.
3. R. Schalkoff, *Pattern Recognition, Statistical, Structural and Neural Approaches*, New York: Wiley, 1992.
4. I.H. Witten, and E. Frank, *Data Mining, Practical Machine Learning Tools and Techniques with Java Implementations*, San Francisco, CA: Morgan Kaufmann, 2000.
5. I. Fortes, L. Mora-López, R. Morales, and F. Triguero, Inductive learning models with missing values, *Math. Comput. Model.*, **44**, 790–806, 2006.
6. D. Pyle, *Data Preparation for Data Mining*, San Francisco, CA: Morgan Kaufmann, 1999.
7. D.R. Wilson, and T.R. Martinez, Improved heterogeneous distance functions, *J. Artif. Intelligence Research*, **6**, 1–34, 1997.
8. J. Furman (ed.), Otoneurology, in: *Neurologic Clinics*, Vol. 23–2, Netherlands: Elsevier, 2005.
9. E. Kentala, I. Pyykkö, Y. Auramo, and M. Juhola, Database for vertigo, *Otolaryngology, Head Neck Surgery*, **112**, 383–390, 1995.
10. J. Laurikkala, E. Kentala, M. Juhola, I. Pyykkö, and S. Lammi, Usefulness of imputation for the analysis of incomplete otoneurological data, *Int. J. Med. Inform.*, **58–59**, 235–242, 2001.
11. R.J.A. Little, and D.B. Rubin, *Statistical Analysis with Missing Data*, New York: Wiley, 1987.
12. M.Y. Kiang, A comparative assessment of classification methods, *Decision Support Systems*, **35**, 441–454, 2003.
13. R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification*, 2nd ed., New York: Wiley, 2001.
14. S. Theodoridis, and K. Koutroumbas, *Pattern Recognition*, 3rd ed., San Diego, CA: Academic Press (Elsevier), 2006.
15. C.M. Bishop, *Neural Networks for Pattern Recognition*, New York: Oxford University Press, 1997.
16. S. Haykin, *Neural Networks, A Comprehensive Foundation*, Upper Saddle River, NJ: Prentice-Hall, 1994.
17. K. Swingler, *Applying Neural Networks, Practical Guide*, London, UK: Academic Press, 1996.

Martti Juhola
University of Tampere
Tampere, Finland

DATA COMMUNICATION

INTRODUCTION

This chapter considers data communication as the data exchange process between devices linked directly by a transmission medium with or without multiaccess, and without intermediate switching nodes. The scope of discussion is confined to the physical and data link layers in the Open Systems Interconnection (OSI) reference model.

The physical layer focuses on the exchange of an unstructured data bit stream via modulated electromagnetic signals (i.e., electrical and optical) transmission over a physical link channel. Modulation determines the spectral efficiency or the ratio of transmitted data bit rate to the signal bandwidth. Data bits may be received in error when signal waveforms are impaired by channel noises. Long-distance transmission involves signal propagation and regeneration by repeaters. Unlike analog transmission, digital transmission must recover the binary data content first before regenerating the signals. The physical layer specifies the mechanical and the electromagnetic characteristics of the transmission medium, and the access interface between the device and the transmission medium. Transmission between two devices can be simplex (unidirectional), full-duplex (simultaneous bidirectional), or half-duplex (bidirectional but one direction at a time).

The data link layer is concerned with the exchange of structured data bit stream via frames over a logical link channel with error recovery control to optimize link reliability. Channel coding adds structured redundancy to the data bit stream to enable error detection or error correction. Each frame consists of data payload and control overhead bits, which convey control procedures between sending and receiving devices. Link control procedures include flow control, error recovery, and device addressing in multiaccess medium.

Elements of Data Communication System

Figure 1 illustrates the elements of a data communication system. The elements include digital source, digital sink, transmitter interface, receiver interface, and physical medium of a communication channel.

The digital source consists of data source and source encoder. The *data source* generates either an analog signal (e.g., audio, video), or a digital data stream (sequence of discrete symbols or binary bits stored in computing memory) to the source encoder. The analog-to-digital converter (ADC) converts an analog signal to a digital data stream of bits through discrete sampling and quantization of the analog signal. Common ADC schemes include pulse code modulation (PCM) that maps each sampled signal-amplitude to a digital data, and delta modulation (DM) that maps the change from the previous sampled value to a digital data.

The *source encoder* employs data compression coding to reduce data redundancy and communication capacity requirements for more efficient data transmission over the communication channel. Lossless compression algorithms (e.g., Huffman) result in decompressed data that is identical to the original data. Lossy compression algorithms (e.g., discrete cosine transform, wavelet compression) result in decompressed data that is a qualified approximation to the original data. Information theory defines the compression ratio limit without data information loss for a given data stream (1).

The transmitter interface consists of the channel encoder and the signal waveform generator in the form of a digital modulator. The *channel encoder* employs error-detection or error-correction coding to introduce structured redundancy in digital data to combat signal degradation (noise, interference, etc.) in channel. The *digital modulator* enables baseband or passband signaling for digital data. The digital baseband modulator maps the digital data stream to the modulated pulse train of discrete-level digital signals. The digital passband modulator maps the digital data stream to the modulated sinusoidal waveform of continuous-level analog signals.

The transmitted digital or analog signals propagate through the physical medium (wire or wireless) of a communication channel. Signal transmission impairments can be caused by the attenuation of signal energy along propagation path, delay distortions because of the propagation velocity variation with frequency, and noises because of the unwanted energy from sources other than transmitter. Thermal noise is caused by the random motion of electrons. Cross talk noise is caused by the unwanted coupling between signal paths. Impulse noise is a spike caused by external electromagnetic disturbances such as lightning and power line disturbances.

The receiver interface consists of the channel decoder and the signal waveform detector or digital demodulator. The *digital demodulator* converts the incoming modulated pulse train or sinusoidal waveform to a digital data stream with potential bit errors incurred by signal transmission impairments. The *channel decoder* detects or corrects bit errors in received data.

The digital sink consists of a source decoder and a data sink. The *source decoder* maps incoming compressed digital data into original digital data transmitted by the data source. The digital-to-analog converter (DAC) converts a digital data stream to an analog signal. The *data sink* receives either an analog signal or a digital data from the source decoder.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

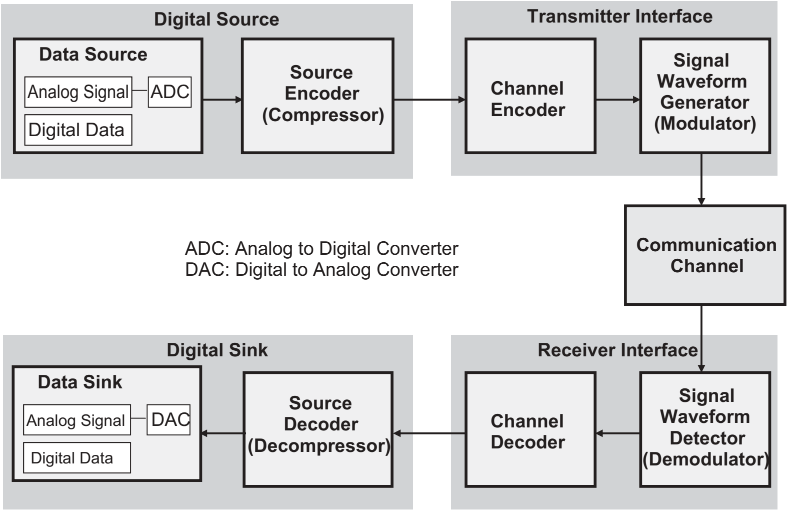

Digital Source
Data Source
Analog Signal
ADC
Digital Data
Source Encoder (Compressor)
Transmitter Interface
Channel Encoder
Signal Waveform Generator (Modulator)
Communication Channel
ADC: Analog to Digital Converter
DAC: Digital to Analog Converter
Digital Sink
Data Sink
Analog Signal
DAC
Digital Data
Source Decoder (Decompressor)
Receiver Interface
Channel Decoder
Signal Waveform Detector (Demodulator)

Figure 1. Elements of data communication system.

PHYSICAL LAYER

Communication Channel

The physical medium of a communication channel is used to transmit electromagnetic signals from the transmitter to the receiver. The physical media can be either guided media for wired transmission, or unguided media for wireless transmission. A signal that propagates along a guided medium is directed and is contained within a physical conductor. An unguided medium transports signal without using a physical conductor.

Guided media include twisted-pair cable, coaxial cable, and optical fiber. Figure 2 shows the electromagnetic spectrum used by wired transmission in these guided media. Consisting of two insulated copper conductors twisted together; twisted-pair cables commonly are used in subscriber loops to telephone networks. Coaxial cable consists of an inner solid wire conductor surrounded by an outer conductor of metal braid, with an insulting ring between the two conductors. Coaxial cables are used widely in cable TV networks, local area networks, and long-distance telephone networks. Coaxial cable offers a larger bandwidth and less interference or crosstalk than that of twisted-pair cable.

Consisting of a thin glass or plastics central core surrounded by a cladding layer, optical fiber transports light signal through the core by means of reflection. Optical fiber offers a wider bandwidth, lower attenuation, and lighter weight than that of coaxial or twisted-pair cables. The wider bandwith is achieved at a relatively higher cost. Optical fiber cables commonly are used in backbone networks and in long-haul undersea links because its wide bandwidth is cost effective.

Unguided media include the air atmosphere, outer space, and water. Wireless transmission and the reception of electromagnetic signals in unguided media are accomplished through an antenna. For unidirectional wireless transmission, the transmitted signal is a focused electromagnetic beam and the receiving antenna must be aligned properly for signal reception. For omni-directional wireless transmission, the transmitted signal spreads out in all directions to enable reception by many antennas. Antenna size depends on the operating frequency of electromagnetic wave (2).

Wireless transmission in the air medium commonly is classified according to the operational frequency ranges or bands of the electromagnetic signals. These signals include radio waves that range from 3 KHz to 1 GHz, microwaves that range from 1 GHz to 300 GHz, and infrared waves that range from 300 GHz to 400 THz. Radio waves mostly are omni-directional; these waves are appropriate for multicasting or broadcasting communication. Their applications include AM and FM radio, television, and cordless phones. Microwaves are unidirectional and are appropriate for unicast communication. Their applications include cellular, satellite, and wireless local area networks. Infrared waves are unidirectional and cannot penetrate physical barriers, thus they can be used for short-range unicast communication. Their applications include communications between computers and peripheral devices such as keyboards, mice, and printers. Figure 2 shows the electromagnetic spectrum used by wireless transmission in the air medium.

Channel capacity is the maximum rate at which data can be transmitted over a medium, using whatever means, with negligible errors. For binary digital data, the transmission rate is measured in bits-per-seconds (bps). Channel capacity increases with transmission bandwidth (W in Hz) and increases with the ratio of received signal power (S in Watts) to noise power (R in Watts); also it is limited by the Shannon theorem: Channel Capacity $= \mathrm{W} \cdot \log_2 (1 + S/R)$.

Channel Transmission Impairments. Communication channels impair transmitted signals, and the exact form of signal degradation depends primarily on the type of transmission medium. Affecting all channels, *additive*

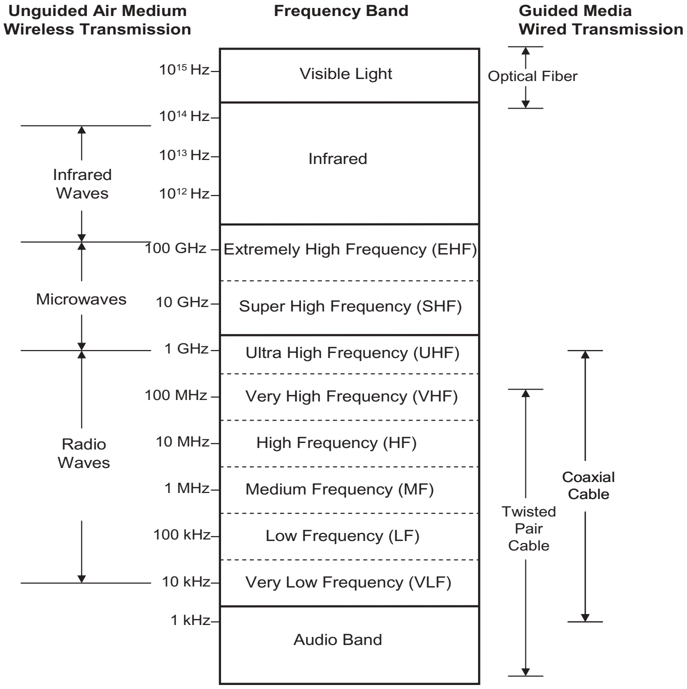


Figure 2. Electromagnetic spectrum used for wired and wireless transmission.

noise is an unwanted electromagnetic signal that is superimposed on the transmitted waveform. Noise is the single most important factor that limits the performance of a digital communication system. Other channel impairments include *attenuation, dispersion*, and *fading.* Attenuation is the decrease of transmitted signal strength as observed over distance. Dispersion is caused by delay distortions. Dispersion in wireless medium is caused by multipath propagation, which occurs when a transmitted signal is received via multiple propagation paths of different delays. It leads to inter-symbol interference (ISI), or the smearing of transmission pulses that represent transmitted symbols causing mutual interference between adjacent symbols. Fading is another problem that is associated with multipath propagation over a wireless medium. It occurs when signal components that arrive via different paths add destructively at the receiver.

In wireline medium, the dispersion and the ISI is caused by amplitude and phase distortion properties of the particular medium. The ISI could be minimized by several techniques, including limiting data rate R to less than twice the channel bandwidth W (i.e., $R < 2{\cdot}W$), pulse shaping (e.g., raised cosine spectrum), adaptive equalization, and correlative coding.

The objective of the data communication system design is to implement a system that minimizes channel impairments. The performance of the digital channel is measured in terms of bit-error-probability or bit-error-rate (BER), block-error probability, and error-free seconds. The communication problem can be stated as follows: given the communication medium, data rate, design transmitter/receiver so as to minimize BER subject to given power and bandwidth constraints; or to minimize power and/or bandwidth subject to given BER objective.

Communication Channel Models. The simplest and most used model for both wireline and wireless communication channels is the additive noise channel. The cumulative noises are caused by thermal noise in the channel and electronic noise in devices. Under this model, the communication channel is assumed to add a random noise process on the transmitted signal. When the statistics of the noise process is assumed to be uncorrelated (white) Gaussian, the model is known as *additive white Gaussian noise (AWGN) channel*. Channel attenuation can be incorporated into this model by introducing a multiplicative term. The *linear filter channel* model is similar to the additive noise channel, which differs only in the linear filtering operation that precedes noise addition. For wireline channels, the filtering effect usually is assumed to be time-invariant.

Digital transmission over wireless mobile channels must account for signal fading because of multipath propagation and Doppler shift. Multipath propagation occurs when a direct line-of-sight (LOS) path and multiple indirect paths exist between a transmitter and a receiver. Signal components that arrive at a receiver with different delay shifts and combine in a distorted version of the transmitted signal. Frequency-selective fading occurs when the amplitudes and the phases of the signal's frequency components are distorted or faded independently.

Doppler shift occurs when a transmitter and a receiver are in a relative motion, which causes the multipath received signal components to undergo different frequency shifts and to combine in a distorted version of the transmitted signal. Time-selective fading occurs when the signal's time components are faded independently.

A wireless mobile channel is modeled as a *Rayleigh fading channel* when a dominant path does not exist among direct and indirect paths between the transmitter and the receiver. The rayleigh model can be applied in complex outdoor environments with several obstructions, such as urban downtown. A wireless mobile channel is modeled as a *Rican fading channel* when a dominant direct or LOS path exists. The rican model can be applied in an open-space, outdoor environment or an indoor environment.

Digital Modulation: Signal Waveform Generation

Digital modulation enables a digital data stream to encode its information contents onto a pulse train signal via *baseband modulation,* or onto a sinusoidal carrier signal via *passband modulation*. Each signal element of the resulting waveform can carry one or more data elements or bits of the data stream. The data rate and the signal rate measures respectively the number of bits and signal elements transmitted in one second. The signal rate is also referred as the baud rate or modulation rate. The signal rate is proportional directly to the date rate, and proportional inversely to the number of data elements carried per signal element.

Besides single-carrier modulation (baseband and passband), this section will also present *spreading code modulation* (i.e., spread spectrum) that hides the signal behind the noise to minimize signal jamming and interception. *Multicarrier modulation* maximizes the spectral efficiency of a frequency-selective fading channel.

Digital Baseband Modulation. Digital baseband modulation or line coding is the process of converting a digital data stream to an amplitude modulated pulse train of discrete-level for transmission in low-pass communication channel. The pulse amplitude modulation formats or line codes include unipolar, polar, bipolar, and biphase. For unipolar codes, the signal has single polarity pulses, which results in a direct current (DC) component. Unipolar codes include unipolar return-to-zero (RZ) that specifies "on" pulses for half bit duration followed by a return to the zero level; and unipolar non-return-to-zero (NRZ) that specifies "on" pulses for full bit duration.

For polar and bipolar codes, the signal has opposite polarity pulses, which reduce the DC component. Polar codes include polar NRZ with signal levels that include positive and negative, and polar RZ with signal levels that include positive, negative, and zero. Bipolar codes specify that data bits with one binary level are represented by zero, whereas successive data bits with the other binary level are represented by pulses with alternative polarity. Bipolar codes include bipolar alternate mark inversion or pseudo-ternary with signal levels that include positive, negative, and zero.

For biphase codes, a transition at the middle of each data bit period always exists to provide clocking synchronization. Biphase codes include Manchester that specifies low-to-high and high-to-low transitions to represent "1" and "0" bits, respectively, and differential Manchester that combines transitions at the beginning and the middle of data bit periods.

The selection of signaling type depends on design considerations such as bandwidth, timing recovery, error detection, and implementation complexity. Bipolar and biphase signaling have no DC spectral component. Also, the main lobe of biphase spectrum is twice as wide as NRZ and bipolar (tradeoff between synchronization or timing recovery and bandwidth).

Signal waveform detection or demodulation recovers the original data sequence from the incoming signal at the receiver end. The original data sequence is subjected to channel impairments. Consider the simplest case of NRZ signaling. The simplest detection is to sample the received pulses at the mid-points of bit periods. This detection method suffers from the poor noise immunity. Optimum detection employs a correlator or a matched filter (integrate and dump for NRZ). This achieves better noise immunity because the noise perturbation is averaged over bit period.

Digital Passband Modulation. Digital passband modulation is the process of converting a digital data stream to a modulated sinusoidal waveform of continuous-level for transmission in band-pass communication channel, which passes frequencies in some frequency bands (3). The conversion process imbeds a digital data stream onto a sinusoidal carrier by changing (keying) one or more of its key parameters such as amplitude, phase, or frequency. Digital passband modulation schemes include amplitude shift keying (ASK), phase shift keying (PSK), and frequency shift keying (FSK).

In binary ASK, the binary data values are mapped to two distinct amplitude variations; one of them is zero and the other one is specified by the presence of the carrier signal. In binary FSK, the binary data values are mapped to two distinct frequencies usually that are of equal but opposite offset from the carrier frequency. In binary PSK, the binary data values are mapped to two distinct phases of the carrier signal; one of them with a phase of 0 degrees, and the other one with a phase of 180 degrees.

In M-ary or multilevel ASK, FSK, and PSK modulation, each signal element of M-levels carries $\log_2 M$ bits. Quadrature amplitude modulation (QAM) combines ASK and PSK. In QAM, two carriers (one shifted by 90 degrees in phase from the other) are used with each carrier that is ASK modulated.

Spreading Code Modulation: Spread Spectrum. Spreading spectrum (4) enables a narrow-band signal to spread across a wider bandwidth of a wireless channel by imparting its information contents onto a spreading code (generated by a pseudo-noise or pseudo-number generator), with the combined bit stream that attains the data rate of the original spreading code bit stream. Spread spectrum technique hides the signal behind the noise to minimize signal jamming and interception. Two common techniques to spread signal bandwidth are frequency hopping spreading spectrum (FHSS) and direct sequence spread spectrum (DSSS).

On the transmitter side, FHSS enables the original signal to modulate or to hop between a set of carrier frequencies at fixed intervals. A pseudo-random number generator or pseudo-noise (PN) source generates a spreading sequence of numbers, with each one serving as an index into the set of carrier frequencies. The resulting bandwidth of the modulated carrier signal is significantly greater than that of the original signal. On the receiver side, the same spreading sequence is used to demodulate the spread spectrum signal.

On the transmitter side, DSSS spreads the original signal by replacing each data bit with multiple bits or chips using a spreading code. A PN source generates a pseudo-random bit stream, which combines with the original signal through multiplication or an exclusive-OR operation to produce the spread spectrum signal. On the receiver side, the same spreading code is used to despread the spread spectrum signal. Multiple DSSS signals can share the same air link if appropriate spreading codes are used such that the spread signals are mutually orthogonal (zero cross-correlation), or they do not interfere with the dispreading operation of a particular signal.

Multi-carrier Modulation: Orthogonal Frequency Division Multiplexing (OFDM). OFDM coverts a data signal into multiple parallel data signals, which modulate multiple closely spaced orthogonal subcarriers onto separate subchannels. Each subcarrier modulation employs conventional single-carrier passband signaling scheme, such as quadrature phase shift keying (QPSK). When a rectangular subcarrier pulse is employed, OFDM modulation can be performed by a simple inverse discrete fourier transform (IDFT) to minimize equipment complexity. OFDM increases the spectral efficiency (ratio of transmitted data rate to signal bandwidth) of a transmission channel by minimizing subcarrier frequency channel separation while maintaining orthogonality of their corresponding time domain waveforms; and by minimizing intersymbol interference (e.g., in mulitpath fading channel) with lower bit-rate subchannels.

Multiplexing

Multiplexing is a methodology to divide the resource of a link into accessible components, to enable multiple data devices to share the capacity of the link simultaneously or sequentially. Data devices access a link synchronously through a single multiplexer. In a later section on general multiple access methodology, data devices can access a link synchronously or asynchronously through distributed access controllers.

Frequency Division Multiplexing (FDM). FDM divides a link into a number of frequency channels assigned to different data sources; each has full-time access to a portion of the link bandwidth via centralized access control. In FDM, signals are modulated onto different carrier frequencies. Each modulated signal is allocated with a frequency channel of specified bandwidth range or a band centered on its carrier frequency. Guard bands are allocated between adjacent frequency channels to counteract inter-channel interference. Guard bands reduce the overall capacity of a link.

Wavelength division multiplexing is similar to FDM, but it operates in the optical domain. It divides an optical link or a fiber into multiple optical wavelength channels assigned to different data signals, which modulate onto optical carriers or light beams at different optical wavelengths. Optical multiplexers and demultiplexers can be realized via various mechanical devices such as prisms and diffraction gratings.

Time Division Multiplexing (TDM). TDM divides a link into a number of time-slot channels assigned to different signals; each accesses the full bandwidth of the channel alternately at different times via centralized transmission access control. The time domain is organized into periodic TDM frames, with each frame containing a cycle of time slots. Each signal is assigned with one or more time-slots in each frame. Therefore, the time-slot channel is the sequence of slots assigned from frame to frame. TDM requires a guard-time for each time slot to counteract inter-channel interference in the time domain.

Instead of preassigning time slots to different signals, *statistical time division multiplexing* (asynchronous TDM with on-demand sharing of time slots) improves bandwidth efficiency by allocating time slots dynamically only when data sources have data to transmit. In statistical TDM, the number of time slots in each frame is less than the number of data sources or input lines to the multiplexer. Data from each source is accompanied by the line address such that the demultiplexer can output the data to the intended output line. Short-term demand exceeding link capacity is handled by buffering within the multiplexer.

Code Division Multiplexing (CDM). CDM divides a wireless channel into code channels assigned to different data signals, each of which modulates an associated unique spreading code to span the whole spectrum of the wireless channel (see previous section on spread spectrum modulation). A receiver retrieves the intended data signal by demodulating the input with the associated spreading code. CDM employs orthogonal spreading codes (e.g., Walsh codes) in which all pair-wise cross correlations are zero.

OFDM. OFDM divides a link into multiple frequency channels with orthogonal subcarriers. Unlike FDM, all of the subchannels are assigned to a single data signal. Furthermore, OFDM specifies the subcarriers to maintain orthogonality of their corresponding time domain waveforms. This technigue allows OFDM to minimize guard-band overhead even though signal spectra that corresponds to the subcarriers overlap in the frequency domain. OFDM can be used to convert a broadband and a wireless channel with frequency-selective fading into multiple frequency-flat subcarrier channels (i.e., transmitted signals in subcarrier channels experience narrowband fading). The multi-carrier modulation aspect of OFDM has been discussed in a previous section.

Multiplexed Transmission Network: Synchronous Optical Network (SONET). TDM-based transmission networks employ either back-to-back demultiplexer–multiplexer pairs or add-drop multiplexers (ADMs) to add or to extract lower-rate signals (tributaries) to or from an existing high-rate data stream. Former plesiochronous TDM-based transmission networks require demultiplexing of a whole multiplexed stream to access a single tributary, and then remultiplexing of the reconstituted tributaries via back-to-back demultiplexer–multiplexer pairs. Current synchronous TDM-based transmission networks employ ADMs that allow tributaries addition and extraction without interfering with other bypass tributaries.

The SONET employs electrical ADMs and converts the high-rate data streams into optical carrier signals via electrical-optical converters for transmission over optical fibers. SONET ADMs can be arranged in linear and ring topologies to form a transmission network. SONET standard specifies the base electrical signal or synchronous transport signal level-1 (STS-1) of 51.84 Mbps, and the multiplexed higher-rate STS-n electrical signals with corresponding optical carrier level-n (OC-n) signal. Repeating 8000 times per second, a SONETSTS-1 frame (9×90 byte-array) consists of transport overhead (9×3 byte-array) that relates to the transmission section and line, and to the synchronous payload envelope (9×87 byte-array) including user data and transmission path overhead. A pointer structure in the transport overhead specifies the beginning of a tributary, and this structure enables the synchronization of frames and tributaries even when their clock frequencies differ slightly.

DATA LINK LAYER

The physical layer of data communications provides an exchange of data bits in the form of a signal over a transmission link between two directly connected devices or stations. The data link layer organizes the bit stream into blocks or frames with control information overhead, and provides a reliable exchange of data frames (blocks of bits) over a data communication link between transmitter and receiver devices. Using a signal transmission service provided by the physical layer, the data link layer provides a data communication service to the network layer. Main data link control functions include frame synchronization, flow control and error recovery (particularly important when a transmission link is error prone), and medium access control to coordinate multiple devices to share and to access the medium of a transmission link.

FRAME SYNCHRONIZATION

A data frame consists of a data payload field and control information fields in the form of header, trailer, and flags. Headers may include start of frame delimiter, character or bit constant, control signaling, and sender and receiver address. Trailes may include an error detection code and an end of frame delimiter. The framing operation packages a sending data message (e.g., network layer data packet) into data frames for transmission over the data link. The frame design should enable a receiver to identify an individual frame after an incoming bit stream is organized into a continuous sequence of frames.

The beginning and the ending of a data frame must be identifiable uniquely. Frames can be of fixed or variable size. Fixed size frames can be delimited implicitly. Variable size frames can be delimited explicitly through a size count encoded in a header field. Transmission error could corrupt the count field, so usually the count field is complemented with starting/ending flags to delimit frames.

In a character-oriented protocol, a data frame consists of a variable number of ASCII-coded characters. Special flag characters are used to delimit the beginning and the end of frames. When these special flag characters appear in the data portion of the frame, character stuffing is used by the sender to insert an extra escape character (ESC) character just before each flag character, followed by the receiver to remove the ESC character.

In a bit-oriented protocol, a data frame consists of a variable number of bits. Special 8-bit pattern flag 01111110 is used to delimit the frame boundary. To preserve data transparency through bit stuffing, the sender inserts an extra 0 bit after sending five consecutive 1 bits, and the receiver discards the 0 bit that follows five consecutive 1 bits.

Channel Coding

Channel encoding is the process of introducing structured redundancy in digital data to allow the receiver to detect or to correct data errors caused by to transmission impairments in the channel. The ratio of redundant bits to the data bits and their relationship determines the error detection and the correction capability of a coding scheme. Channel coding schemes include block coding and convolutional coding schemes (5).

In block coding, data bits are grouped into blocks or datawords (each of k bits); and r redundant bits are added to each block to form a codeword (each of $n = k + r$ bits). The number of possible codewords is 2^{n} and 2^{k} of them are valid codewords, which encode the datawords.

The Hamming distance d_H between two codewords is the number of bit positions in which the corresponding bits differ, e.g., d_H (01011, 11110) = 3. The minimum distance $d_{\min}$ of a block code (that consists of a set of valid codewords) is the minimum d_{H} between any two different valid codewords in the set, e.g., $d_{\min}$(00000, 01011, 10101, 11110) = 3. A block code with $d_{\min}$ can detect all combinations of $(d_{\min}-1)$ errors. A block code with $d_{\min}$ can correct all combinations of errors specified by the largest integer that is less than or equal to $(d_{\min}-1)/2$.

Figure 3 illustrates the error detection operation based on the parity check code. The channel encoder adds a parity bit to the 7-bit dataword (1100100) to produce an even-parity 8-bit codeword (11001001). The signal waveform generator maps the codeword into a NRZ-L baseband signal. The received signal was corrupted by channel noise, one of the 0 bits was received and it was detected as a 1 bit. The channel decoder concludes that the received odd-parity codeword (11000001) is invalid.

Block codes that are deployed today are mostly linear, rather than nonlinear. For linear block codes, the modulo-2 addition of two valid codewords creates another valid codeword. Common linear block coding schemes include simple parity check codes, hamming codes, and cyclic redundancy check (CRC) codes. In a CRC code, the cyclic shifted version of a codeword is also a codeword. This allows for simple implementation by using shift registers. Bose–Chaudhuri–Hocqunghem (BCH) codes constitute a large class of powerful cyclic block codes. Reed–Solomon (RS) codes commonly are used in commercial applications, such as in digital subscriber line for Internet access and in video broadcasting for digital video. For integers m and t, RS coding operates in k symbols of data information (with m bit per symbol), n symbols $(2^m - 1)$ of data block, and $2t$ symbols of redundancy with $d_{\min} = 2t + 1$.

If the data source and the sink transmit and receive in a continuous stream, a block code would be less convenient than a code that generates redundant bits continuously without bits blocking. In convolutional coding, the coding structure extends over the entire transmitted data stream effectively, instead of being limited to data blocks or codewords as in block coding. Convolutional or trellis coding is applicable for communication systems configured with simple encoders and sophisticated decoders; such as space and satellite communication systems.

The channel coding selection for a channel should account for the channel condition, including channel error characteristics. All block and convolutional codes can be used in AWGN channels that produce random errors. However, convolutional codes tend to be ineffective in fading channels that produce burst errors. On the other hand, RS block codes are much more capable of correcting burst errors. For block codes designed for correcting random

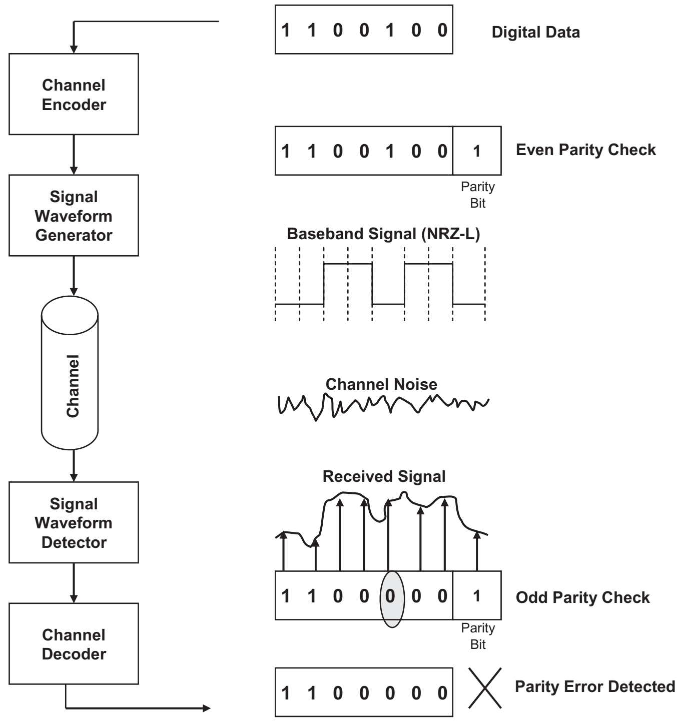


Figure 3. Example of parity-based error detection.

errors, they can be combined with block interleaving to enhance the capability of burst error correction. In block interleaving, L codewords of n-bit block to be transmitted are stored row-by-row in a rectangular $L \times n$ array buffer, and then data is transmitted over the channel column-by-column. The interleaving depth L determines the distribution of the errors of a burst over the codewords. Burst error correction is effective if the resulting number of errors to be distributed in each codeword is within the block code's error correction capability.

Flow Control

Flow control enables the receiver to slow down the sender so that the data frames are not sent at a rate higher than the receiver could handle. Otherwise, the receive buffers would overflow, which results in data frame loss. The sender is allowed to send a maximum number of frames without receiving acknowledgment from the receiver. Those frames sent but not yet acknowledged (outstanding) would be buffered by the sender in case they need to be retransmitted because of transmission error. The receiver can flow control the sender by either passively withholding acknowledgment, or actively signaling the sender explicitly that it is not ready to receive.

Error Control

Transmission errors can result in either lost frames or damaged frames with corrupted bits because of noise and interference in the channel. Error recovery enables a reliable data transfer between the sender and the receiver. When a return channel is available for data acknowledgment and recovery control, the receiver employs error detection via channel coding violation; and then requests for retransmission via automatic repeat request (ARQ) procedure. When the return channel is not available, the receiver employs forward error correction that detects an error via channel coding violation; and then the receiver corrects it by estimating the most likely data transmitted by sender.

ARQ Protocols

ARQ is a general error recovery approach that includes retransmission procedures at the sender to recover from data frames detected with error at receiver, or missing data frames (i.e., very corrupted frames not recognized by receiver). It also includes a procedure to recover from missing acknowledgment frames. With data frames that incorporate error detection codes, the receiver discards data frames with detected errors. The receiver sends an acknowledgment (ACK) frame to indicate the successful reception of one or more data frames. The receiver sends an negative acknowledgment (NAK) frame to request retransmission of one or more data frames in error. The sender sets a transmit timer when a data frame is sent, and retransmits a frame when the timer expires to account for a potentially missing data frame.

Data frames and corresponding ACK/NAK frames are identified by k-bit modulo-2 sequence numbers (N_s). The sequence numbers enable the receiver to avoid receiving duplicate frames because of lost acknowledgments. Data frames could carry a piggyback acknowledgment for data frames transmitted in the opposite direction. Acknowledgment sequence number (N_r) identifies the next data frame expected by the receiver, and acknowledges all outstanding frames up to $N_r - 1$. Outstanding data frames that have been sent but not yet acknowledged would be buffered in transmit buffers in case they need to be retransmitted because of a transmission error.

ARQ protocols combine error recovery control with *sliding-window* flow control. The sender can send multiple consecutive data frames without receiving acknowledgment from the receiver. The sender and the receiver maintain flow control state information through sending and receiving windows.

The sending window contains sequence numbers of data frames that the sender is allowed to send, and it is partitioned into two parts. From the trailing edge to the partition, the first part consists of outstanding frames. From the partition to the leading edge, the second part consists of frames yet to be sent. The sending window size (Ws) equals the maximum number of outstanding frames or the number of transmit buffers. The sender stops accepting data frames for transmission because of flow control when the partition overlaps the leading edge, i.e., all frames in window are outstanding. When an ACK is received with N_r, the window slides forward so that the trailing edge is at N_r.

The receiving window contains the sequence numbers of the data frames that the receiver is expecting to receive. Receiving window size (W_r) is equal to the number of receive buffers. The receive data frames with sequence numbers outside receive window are discarded. The accepted data frames are delivered to the upper layer (i.e., network layer) if their sequence numbers follow immediately that of the last frame delivered; or else they are stored in receive buffers. The acknowledgment procedures are as follows. First, the receiver sets an acknowledgment timer when a frame is delivered. If it is possible to piggyback an ACK, the receiver does so and resets the timer. Second, when an acknowledgment timer expires, it will send a self-contained ACK frame. For both cases, the receiver sets N_r to be the sequence number at the trailing edge of the window at the time the ACK is sent.

Common ARQ protocols include the stop-and-wait ARQ, and the sliding-window based go-back-n ARQ and selective-repeat ARQ. The sending and receiving window sizes specified for these protocols are summarized in Table 1.

Stop-and-Wait ARQ. The stop-and-wait ARQ specifies that both the sending and the receiving window sizes are limited to one. Single-bit modulo-2 sequence numbers are used. The sender and receiver process one frame at a time. This ARQ is particularly useful for the half-duplex data link as the two sides never transmit simultaneously. After sending one data frame, the sender sets the transmit timer and waits for an ACK before sending the next data frame. The sender retransmits the data frame if NAK is received or if the transmit timer expires.

Go-Back-N ARQ. The go-back-n ARQ specifies that the sending window size can range from one to $2^k - 1$; whereas the receiving window size is limited to one. If a data frame is received in error, all subsequent frames will not fall within the receiving window and will be discarded. Consequently, retransmission of all these frames will be necessary. The receiver may seek retransmission actively by sending an NAK with N_r being set to sequence number of the receive window, and then resetting the running acknowledgment timer.

The sender retransmission procedures are as follows. First, the sender sets a transmit timer when a frame is transmitted or retransmitted and resets the timer when frame is acknowledged. Second, when the transmit timer for the oldest outstanding frame expires, the sender retransmits all of the outstanding frames in the sending window. Third, if NAK with Nr is received, the sender slides a sending window to Nr and retransmits all of the outstanding frames in the new window.

Selective-Repeat ARQ. The selective-repeat ARQ specifies that the sending window size can range from one to $2^{(k-1)}$; whereas the receiving window size is less than or equal to the sending window size. Instead of retransmitting every frame in pipeline when a frame is received in error, only the frame in error is retransmitted. Received frames with a sequence number inside receiving window but out of order are stored in buffers. Then the receiver sends a NAK with N_r set to the trailing edge of the receive window, and resets the acknowledgment timer.

The sender retransmission procedures are as follows: The sender retransmits each outstanding frame when the transmit timer for that frame times out. If NAK is received, the sender retransmits the frame indicated by the Nr field in the NAK, slides the window forward if necessary, and sets the transmit timer.

Medium Access Control (MAC)

For the multiaccess or the shared-medium link as illustrated in Fig. 4, the OSI reference model divides the data link layer into the upper logical link control (LLC) sublayer and the lower medium access control (MAC) sublayer. Like the standard data link layer that provides services to the network layer, the LLC sublayer enables transmission control between two directly connected data devices. The MAC sublayer provides services to the LLC sublayer, and enables distributed data devices to access a shared-medium. Although the MAC protocols are medium specific, the LLC protocols are medium independent. The MAC protocol can be classified based on its *multiple access* and *resource sharing* methodologies.

Multiple Access Methodology. In multiplexing control, data devices access a link synchronously through a single multiplexer. In multiple access control, distributed data devices access a link synchronously or asynchronously through distributed controllers. An example *of synchronous* multiple access would be down-link transmissions from a cellular base station to wireless mobile devices. An example of *asynchronous* multiple access would be an up-link transmission from a wireless mobile device to a cellular base station. Like multiplexing, multiples access divides the resource of a transmission channel into accessible portions or subchannels identified by frequency subbands, time-slots, or spreading codes.

Frequency division multiple access (FDMA) schemes divide the frequency-domain resource of a link into portions of spectrum bands that are separated by guard-bands to reduce inter-channel interference. Data devices can transmit simultaneously and continuously on assigned subbands. For a wireless fading channel with mobile devices, FDMA requires a larger guard-band between the adjacent frequency channels to counteract the Doppler frequency spread caused by the data sources mobility.

Time division multiple access (TDMA) schemes divide the time-domain resource of a link into time-slots, separated by guard-times. Data devices take turn and transmit in assigned time-slots, which makes use of the entire link bandwidth. For the wireless fading channel, TDMA requires a larger guard-time for each time slot to counteract the delay spread caused by the multi-path fading.

Code division multiple access (CDMA) schemes divide the code-domain resource of a link into a collection of spreading codes. Data devices transmit their spread spectrum signals via assigned codes. In *synchronous CDMA*, orthogonal spreading codes are employed in which all pairwise cross-correlations are zero. In *asynchronous CDMA*, data devices access the code channels via distributed access controls without coordination, which prohibit the employment of orthogonal spreading codes because of arbitrary random transmission instants. Spreading pseudo-noise sequences are employed instead, which are statistically uncorrelated. This causes multiple access interference

Table 1. Window Sizes for ARQ Protocols

ARQ Protocol	Sending Window Size (W_s)	Receiving Window Size (W_r)
Stop-and-wait	1	1
Go-Back-N	$1 < W_s < 2^k - 1$	1
Selective Repeat	$1 < W_s < 2^{(k-1)}$	$1 < W_r < W_s$

Note: k = Number of bits available for frame sequence number.

(MAI), which is proportional directly to the number of spreading signals received at the same power level.

Orthogonal frequency division multiple access (OFDMA) schemes divide the frequency-domain resource into multiple frequency channels with orthogonal subcarriers. As discussed in a previous section, OFDM is considered as a multi-carrier modulation technique, rather than a multiplexing technique, when all of the subchannels are assigned to a single user signal. In OFDMA, multiple subsets of subchannels are assigned to different user signals.

Resource Sharing Methodology. Resource sharing control can occur either through a *centralized* resource scheduler with global resource availability information, or a *distributed* scheduler with local or isolated resource availability information. Resource sharing schemes include *preassignment, random access (contention), limited-contention*, and *on-demand assignment*. Preassignment allows devices with predetermined and fixed allocation of link resources regardless of their needs to transmit data. Random access allows devices to contend for the link resources by transmitting data whenever available. On-demand assignment allows devices to reserve data for link resources based on their dynamic needs to transmit data.

Preassignment MAC Protocols. These protocols employ static channelization. Link capacity is preassigned or allocated a priori to each station in the frequency domain via FDMA or in the time domain via TDMA. With FDMA-based protocols, disjointed frequency bands (separated by guard bands) are assigned to the links that connect the stations. For a single channel per carrier assignment, each frequency band carries only one channel of nonmultiplexed traffic. With TDMA-based protocols, stations take turns to transmit a burst of data over the same channel. Stations must synchronize with the network timing and the TDMA frame structure. These protocols achieve high efficiency if continuous streams of traffic are exchanged between stations, but they incur poor efficiency under bursty traffic.

Random Access (Contention) MAC Protocols. With random access or contention MAC protocols, no scheduled times exist for stations to transmit and all stations compete against each other to access to the full capacity of the data link. Thus, it is a distributed MAC protocol. Simultaneous transmissions by two or more stations result in a collision that causes all data frames involved to be either destroyed or modified. Collision is recovered by retransmissions after some random time. Delay may become unbounded under heavy traffic, and channel use is relatively poor compared with other MAC protocols that avoid collision, resolve collision, or limit contention. Common random access MAC protocols include Pure ALOHA, Slotted ALOHA, carrier sense multiple access (CSMA), and CSMA with collision detection or with collision avoidance.

Pure ALOHA protocol allows the station to send data frame whenever it is ready, with no regard for other stations (6). The station learns about the collision of its data frame by monitoring its own transmission (distributed control), or by a lack of acknowledgment from a control entity within a time-out period (centralized control). The station retransmits the collided data frame after a random delay or a back-off time. Suppose a station transmits a frame at time t_o, and all data frames are of equal duration T_{frame}. Collision occurs if one of more stations transmit frames within the time interval $(t_o - T_{\text{frame}}, t_o + T_{\text{frame}})$, which is referred to as the vulnerable period of length $2T_{\text{frame}}$.

Heavy traffic causes high collision probability, which increases traffic because of retransmission. In turn, this traffic increases collision probability, and the vicious cycle would cause system instability because of the loss of collision recovery capability. Short term load fluctuations could knock the system into instability. Stabilizing strategies include increasing the range of back-off with increasing number of retransmissions (e.g., *binary exponential back-off*), and limiting the maximum number of retransmissions.

Slotted ALOHA protocol is similar to that of pure ALOHA, but stations are synchronized to time slots of duration equal to the frame duration T_{frame}. Stations may access the medium only at the start of each time slot. Thus, arrivals during each time slot are transmitted at the beginning of the next time slot. Collision recovery is the same as pure ALOHA. Collisions are in the form of a complete frame overlap, and the vulnerable period is of length T_{frame}, which is half that of pure ALOHA. The maximum throughput achieved is twice that of pure ALOHA.

CSMA protocols reduce the collision probability by *sensing* the state of the medium before transmission. Each station proceeds to transmit the data frame if the medium is sensed to be *idle*. Each station defers the data frame transmission and repeats sensing if the medium is sensed to be *busy*. Collision still could occur because the propagation delay must be accounted for the first bit to reach every station and for every station to sense it. The vulnerable period is the maximum propagation delay incurred for a signal to propagate between any two stations. It can be shown that CSMA protocols outperform ALOHA protocols if propagation delay is much smaller than the frame duration. Depending on how they react when the medium is sensed to be busy or idle, CSMA schemes are classified additionally into nonpersistent CSMA, 1-persistent CSMA, and p-persistent CSMA.

With *nonpersistent CSMA* (slotted or unslotted), a station waits for a random delay to sense again if the medium is busy; it transmits as soon as medium is idle. With *1-persistent CSMA* (slotted or unslotted), a station continues to sense if the medium is busy; it transmits as soon as

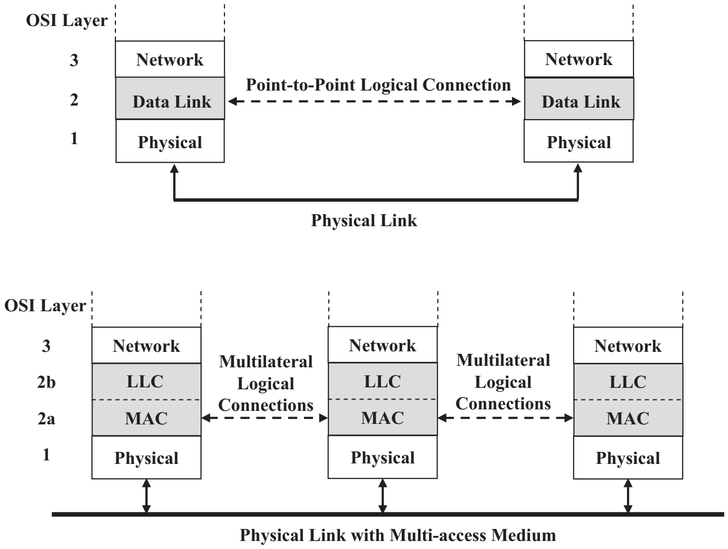


Figure 4. Data link layer.

the medium is idle. With *p-persistent CSMA* (slotted), a station continues sensing if the medium is busy; it transmits in the next slot with probability p, and delay one slot with probability (1-p) if the medium is idle.

The similarities and the differences between these CSMA schemes are summarized in Table 2.

CSMA with Collision Detection (CSMA/CD) protocol enhances the additionally CSMA operations by enabling the stations to continue sensing the channel and detecting collision while transmitting data frames (7). If a collision is *detected* (e.g., received signal energy is greater than that of sent signal, and encoding violation), the involved station would abort frame transmission. After transmitting a brief jamming signal to assure other stations would detect the collision, the involved station would back off and repeat the carrier sensing. Successful frame transmission is assured if no collision is detected within round-trip propagation delay (with regard to the furthest station) of frame transmission.

CSMA with Collision Avoidance (CSMA/CA) protocol enables stations to avoid collisions in wireless networks, where collision detection is not practical because the dynamic range of wireless signals is very large. If the medium is sensed to be idle, a station continues to monitor the medium for a time interval known as the inter-frame space (IFS). If the medium remains idle after IFS, the station chooses the time after a binary exponential back off to schedule its transmission, and then continues to monitor the medium. If the medium is sensed to be busy either during IFS or back off waiting, the station continues to monitor the medium until the medium becomes idle. At the scheduled time, the station transmits the data frame and waits for its positive acknowledgment. The station increases the back off if its transmission fails, and continues to monitor the medium until the medium becomes idle.

Limited-Contention MAC Protocols. These MAC protocols limit the number of contending stations per slot to maximize the probability of success. Different groups of stations contend for repetitive series of different slots. The large group size and the high repetition rate are assigned for light traffic, whereas a small group size and low repetition rate are assigned for heavy traffic. These protocols combine the best properties of contention and contention-free protocols. It achieves low access delay under light traffic and high throughput under heavy traffic. It is applicable to slotted ALOHA and slotted CSMA.

The splitting algorithms (8) are employed commonly by limited-contention MAC protocols. These algorithms split the colliding stations into two groups or subsets; one subset is allowed to transmit in the next slot. The splitting continues until the collision is resolved. Each station can choose whether or not to transmit in the successive slots (i.e., belong to the transmitting subset) according to the random selection, the arrival time of its collided data, or the unique identifier of the station.

On-Demand Assignment MAC. These types of protocols set up an orderly access to the medium. These protocols require the stations to exchange status information of readiness to transmit data frames, and a station can transmit only when it has been authorized by other stations or a centralized control entity. Common on-demand assignment MAC protocols include reservation-based protocols and token-passing protocols.

Reservation-based protocols require a station to reserve link resources before transmitting data. The reservation request could be signaled to other distributed stations or to a centralized reservation control entity. For example, in TDMA-based channelization, each TDMA frame could consist of data slots and reservation slots (usually smaller in size). Each station could signal a request in its own reservation slot, and transmit data in the next TDMA frame.

Token-passing protocols require a special bit pattern (token) to be passed from station to station in a round robin fashion. The token can be passed via a point-to-point line that connects one station to the next (token ring), or via a broadcast bus that connects all stations (token bus). Only a station that posses a token is allowed to send data. A station passes the token to the next station after sending data, or if it has no data to send. Each station is given the chance to send data in turns, and no contention occurs. The average access delay is the summation of the time to pass token through half the stations and the frame transmission time. The average access delay could be fairly long, but it is bounded even under heavy traffic. Token-passing protocols rely on distributed control with no need for network synchronization.

MULTI-INPUT MULTI-OUTPUT (MIMO) TRANSMISSION

MIMO and MIMO-OFDM have emerged as the next generation wireless transmission technologies with antenna arrays for mobile broadband wireless communications. As illustrated in Fig. 5, MIMO (9) is a multi-dimensional wireless transmission system with a transmitter that has multiple waveform generators (i.e., transmit antennas), and a receiver that has multiple waveform detectors (i.e., receive antennas). Multiplexing and channel coding in a MIMO channel would operate over the temporal dimension and the spatial dimension of transmit antennas. MIMO systems can be used either to achieve a diversity gain through space time coding (STC) to counteract multipath fading, or to achieve a throughput capacity gain through space division multiplexing (SDM) to counteract the bandwidth limitation of a wireless channel.

STC

STC is a two-dimensional channel coding method to maximize spectral efficiency in a MIMO channel by coding over the temporal dimension and the spatial dimension of transmit antennas. STC schemes can be based on either space time block codes (STBCs) or by space time trellis codes (STTCs). For a given number of transmit antennas (N_{TA}), an STBC scheme maps a number of modulated symbols (N_{MS}) into a two-dimensional $N_{TA} \times N_{MS}$ codeword that spans in the temporal and the spatial dimensions. On the other hand, the STTC scheme encodes a bit stream through a convolutional code, and then maps the coded bit stream into the temporal and the spatial dimensions.

SDM

SDM is a two-dimensional multiplexing method to increase data throughput in a MIMO channel by enabling simultaneous transmissions of different signals at the same carrier frequency through multiple transmit antennas. The mixed signals that arrived at the receiver can be demultiplexed by employing spatial sampling via multiple receive antennas and corresponding signal processing algorithms.

Combined MIMO–OFDM Transmission

As discussed in a previous section, OFDM can be used to convert a broadband wireless channel with frequency-selective fading into a several frequency-flat, subcarrier channels. A combined MIMO–OFDM system performs MIMO transmission per subcarrier channel with narrowband fading, thus enabling MIMO systems to operate with more optimality in broadband communications. Multiplexing and coding in a MIMO–OFDM system would operate over three dimensions (i.e., temporal, spatial, and subcarrier dimensions).

CONCLUSION

The maximum error-free data rate or capacity of an AWGN channel with a given channel bandwidth is determined by the Shannon capacity. Continuous improvements have been made to design modulation and coding schemes to approach the theoretical error-free capacity limit under the limitations of channel bandwidth and transmission power. Design objectives have always been to increase the spectral efficiency and the link reliability. Modulation can be used to increase spectral efficiency by sacrificing link reliability through modulation, whereas channel coding can be used to increase link reliability by sacrificing spectral efficiency. The performance of modulation and channel coding schemes depend on channel conditions. Existing AWGN-based modulation and coding schemes do not perform optimally in Rayleigh fading channels, whose error-free capacity are considerably less than that of AWGN channels. As fading channels have been increasing used to deploy cellular and Wi-Fi access applications, increasingly needs exist to deploy the next-generation wireless mobile technologies that are best suited for fading channels under all sorts of channel conditions (i.e., flat, frequency-selective, and time-selective fading). The emerging MIMO-OFDM technologies are promising. MIMO optimizes throughput

Table 2. CSMA Schemes Classification

	CSMA Schemes		
Medium Sensing Status	Non-Persistent (Slotted or Unslotted)	l-Persistent (Slotted or Unslotted)	p-Persistent (Slotted)
Idle	Send with Probability = 1	Send with Probability = 1	Send with Probability = p
Busy	Wait random delay, then sense channel	Continue sensing channel until idle	Continue sensing channel until idle
Collision	Wait random delay, then start over carrier sensing		

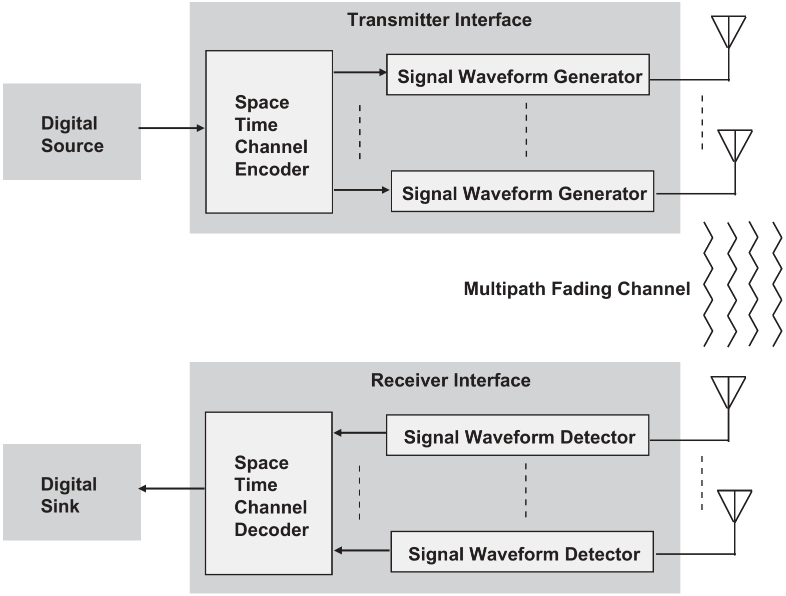


Figure 5. MIMO transmission.

capacity and link reliability by using antenna arrays; whereas OFDM optimizes spectral efficiency and link reliability by converting a broadband frequency-selective fading channel into a set of frequency flat fading channels with less signal distortions.

BIBLIOGRAPHY

1. D. Hankerson, G. Harris, and P. Johnson, *Introduction to Information Theory and Data Compression*, Boca Raton, FL: CRC Press, 1998.
2. W. Stallings, *Wireless Communications and Networks*, Englewood Cliffs, NJ: Prentice Hall, 2001.
3. S. G. Wilson, *Digital Modulation and Coding*, Upper Saddle River, NJ: Prentice Hall, 1996, pp. 1–286.
4. R. E. Ziemer, R. L. Peterson, and D. E. Borth, *Introduction to Spread Spectrum Communications*, Upper Saddle River, NJ: Prentice Hall, 1995.
5. T. K. Moon, *Error Correcting Coding: Mathematical Methods and Algorithms*, Hoboken, NJ: John Wiley & Sons Inc., 2005.
6. N. Abramson andF. Kuo,*The ALOHA system in Computer Communication Networks*,Englewood Cliffs, NJ:Prentice-Hall,1973, pp.501–518.
7. IEEE Standard 802.3, CSMA/CD Access Method
8. D. Bertsekas,R. Gallager,*Data Networks*,2nd ed.,Englewood Cliffs, NJ:Prentice Hall,1992.
9. G. L. Stuber,J. R. Barry,S. W. McLaughlin, andY. Li,“Broadband MIMO-OFDM wireless communications,”*Proc. of the IEEE*,**92**(2):2004,271–294.

FURTHER READING

S. Haykin, *Communication Systems*, 4th ed., New York: John Wiley & Sons, Inc., 2001.

B. Sklar, *Digital Communications: Fundamentals and Applications*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2001.

B. P. Lathi, *Modern Digital and Analog Communication Systems*, 3rd ed., New York: Oxford, 1998.

OLIVER YU
University of Illinois at Chicago
Chicago, Illinois

D

DATA COMPRESSION CODES, LOSSY

INTRODUCTION

In many data compression applications, one does not insist that the reconstructed data (= decoder output) is absolutely identical to the original source data (= encoder input). Such applications typically involve image or audio compression or compression results of various empirical measurements. Small reconstruction errors may be indistinguishable to a human observer, or small compression artifacts may be tolerable to the application. For example, human eye and ear are not equally sensitive to all frequencies, so information may be removed from less important frequencies without visual or audible effect.

The advantage of lossy compression stems from the fact that allowing reconstruction errors typically results in much higher compression rates than is admitted by lossless methods. This higher compression rate is understandable because several similar inputs now are represented by the same compressed file; in other words, the encoding process is not one to one. Hence, fewer compressed files are available, which results in the reduction in the number of bits required to specify each file.

It also is natural that a tradeoff exists between the amounts of reconstruction error and compression: Allowing bigger error results in higher compression. This tradeoff is studied by a subfield of information theory called the rate-distortion theory.

Information theory and rate-distortion theory can be applied to general types of input data. In the most important lossy data compression applications, the input data consists of numerical values, for example, audio samples or pixel intensities in images. In these cases, a typical lossy compression step is the quantization of numerical values. Quantization refers to the process of rounding some numerical values into a smaller range of values. Rounding an integer to the closest multiple of 10 is a typical example of quantization. Quantization can be performed to each number separately (scalar quantization), or several numbers can be quantized together (vector quantization). Quantization is a lossy operation as several numbers are represented by the same value in the reduced range.

In compression, quantization is not performed directly on the original data values. Rather, the data values are transformed first in such a way that quantization provides maximal compression gain. Commonly used transformations are orthogonal linear transformations (e.g., discrete cosine transform DCT). Also, prediction-based transformations are common: Previous data values are used to form a prediction for the next data value. The difference between the actual data value and the predicted value is the transformed number that is quantized and then included in the compressed file. The purpose of these transformations is to create numerical representations where many data values are close to zero and, hence, will be quantized to zero. The quantized data sets have highly uneven distributions, including many zeroes, and therefore they can be effectively compressed using entropy coding methods such as Huffman or arithmetic coding. Notice that quantization is the only operation that destroys information in these algorithms. Otherwise they consist of lossless transformations and entropy coders.

For example, the widely used lossy image compression algorithm JPEG is based on DCT transformation of 8×8 image blocks and scalar quantization of the transformed values. DCT transformation separates different frequencies so the transformed values are the frequency components of the image blocks. Taking advantage of the fact that the human visual system is more sensitive to low than high frequencies, the high frequency components are quantized more heavily than the low frequency ones. Combined with the fact that typical images contain more low than high frequency information, the quantization results in very few non zero high-frequency components. The components are ordered according to the frequency and entropy coded using Huffman coding.

The following sections discuss in more detail the basic ideas of lossy information theory, rate-distortion theory, scalar and vector quantization, prediction-based coding, and transform coding. Image compression algorithm JPEG is presented also.

THEORY

The Shannon information theory (1) allows one to analyze achievable rate-distortion performance for lossy compression. For simplicity, in the following illustration of the key concepts we assume a finite, independent, and identically distributed (or *iid*) source X. This means that the input data to be compressed consists of a sequence x_1,x_2,x_3,... of symbols from a finite source alphabet X where each member $x_i \in X$ of the sequence is produced with the same fixed probability distribution P on X, independent of other members of the sequence. The *entropy* of such a source is

$$H(X) = -\sum_{x \in X} P(x) \log P(x)$$

that is, the weighted average of the *self-information* $I(x) = -\log P(x)$ of elements $x \in X$. All logarithms are taken in base two, assuming that the compressed data is expressed in bits. The entropy measures the average amount of uncertainty that we have about the output symbol emitted by the source. It turns out that the entropy $H(X)$ is the best expected bitrate per source symbol that any lossless code for this source can

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

achieve (1). See Ref. 2 or any other standard information theory text for more details.

Lossy Information Theory

Suppose then that we allow errors in the reconstruction. In the general set-up we can assume an arbitrary reconstruction alphabet Y. Typically $Y = X$, as it is natural to reconstruct into the original symbol set, but this is not required. Let us specify desired *conditional reconstruction probabilities* $P(y|x)$ for all $x \in X$ and $y \in Y$, where for every $x \in X$ we have

$$\sum_{y \in Y} P(y|x) = 1$$

Value $P(y|x)$ is the probability that source symbol x will be reconstructed as y. Our goal is to design a code that realizes these conditional reconstruction probabilities and has as low an expected bitrate as possible. Note that the code can reconstruct source symbol x into various reconstruction symbols $y \in Y$, depending on the other elements of the input sequence. This is due to the fact that we do not encode individual symbols separately, but coding is done on longer segments of source symbols. Conditional probabilities $P(y|x)$ provide the desired reconstruction distribution. Formula for the best achievable bitrate, as provided by the Shannon theory (1,3), is presented below. See also Refs. 2 and 4.

Example 1. Consider a binary *iid* source X with source alphabet {0, 1} and source probabilities $P(0) = P(1) = \frac{1}{2}$. We encode source sequences in segments of three bits by assigning code c_0 to segments with more 0s than 1s and code c_1 to segments with more 1s than 0s. The entropy of the code is 1 bit per code, or $\frac{1}{3}$ bits per source symbol. Hence, we obtain compression ratio 1:3.

Let us reconstruct code c_0 as segment 000 and c_1 as 111. Then, source segments 000,001,010, and 100 get all reconstructed as 000, whereas segments 111,110,101, and 011 get reconstructed as 111. The encoding/decodong process is deterministic, but source symbols 0 get reconstructed as 1 one quarter of the time (whenever they happen to be in the same segment with two 1s). We say that the conditional reconstruction probabilities are

$$\begin{aligned} P(0|0) = P(1|1) &= \tfrac{3}{4} \\ P(1|0) = P(0|1) &= \tfrac{1}{4} \end{aligned} \qquad \square$$

For every fixed $x \in X$, the conditional reconstruction probabilities $P(y|x)$ are a probability distribution on Y, whose entropy

$$H(Y|x) = -\sum_{y \in Y} P(y|x) \log P(y|x)$$

measures the amount of uncertainty about the reconstruction if the source symbol is known to be x. Their average, weighted by the source probabilities $P(x)$, is the *conditional entropy*

$$H(Y|X) = \sum_{x \in X} P(x) H(Y|x) = -\sum_{x \in X} \sum_{y \in Y} P(x,y) \log P(y|x)$$

where $P(x,y) = P(x)P(y|x)$ is the joint probability of $x \in X$ and $y \in Y$. Conditional entropy $H(Y|X)$ measures the expected amount of uncertainty about the reconstruction to an observer that knows the corresponding source symbol.

The source probabilities $P(x)$ and the conditional reconstruction probabilities $P(y|x)$ also induce an overall probability distribution on the reconstruction alphabet Y, where for every $y \in Y$

$$P(y) = \sum_{x \in X} P(x) P(y|x)$$

We additionally can calculate the conditional source probabilities given the reconstruction symbol $y \in Y$ as

$$P(x|y) = \frac{P(x,y)}{P(y)}$$

The reconstruction entropy $H(Y)$ and the conditional source entropy $H(X|Y)$ in this direction are calculated as follows:

$$\begin{aligned} H(Y) &= -\sum_{y \in Y} P(y) \log P(y) \\ H(X|Y) &= -\sum_{x \in X} \sum_{y \in Y} P(x,y) \log P(x|y) \end{aligned}$$

They key concept of *average mutual information* is defined as the difference

$$I(X,Y) = H(X) - H(X|Y)$$

between the source entropy $H(X)$ and the conditional source entropy $H(X|Y)$. It measures the expectation of how much the uncertainty about the source symbol is reduced if the corresponding reconstruction symbol is known. It can be proved easily that $I(X,Y) \geq 0$ and that $I(X,Y) = I(Y,X)$ where we denote $I(Y,X) = H(Y) - H(Y|X)$ (2). Hence, the average mutual information also is the amount by which the uncertainty about the reconstruction is reduced if the source symbol is known.

It turns out that $I(X,Y)$ is the best bitrate that any code realizing the desired conditional reconstruction probabilities $P(y|x)$ can achieve. More precisely, this statement contains two directions: the direct statement that states that $I(X,Y)$ is a bitrate that can be achieved, and the converse statement that states that no code can have a bitrate below $I(X, Y)$. More precisely, the *direct lossy source coding theorem* can be formulated as follows: For every positive number $\varepsilon > 0$, no matter how small, a code exists such that

(i) For every $x \in X$ and $y \in Y$ the probability that x gets reconstructed as y is within ε of the desired $P(y|x)$, and

(ii) Source sequences are compressed into at most $I(X, Y) + \varepsilon$ expected bits per source symbol.

The *converse of the lossy source coding theorem* states that no code that realizes given reconstruction probabilities $P(y|x)$ can have an expected bitrate below $I(X, Y)$.

The proof of the converse statement is easier. The direct theorem is proved using the Shannon random coding technique. This technique is an example of the *probabilistic argument*, successfully applied in many areas in combinatorics and graph theory. The proof shows the existence of good codes, but it does not provide practical means of constructing such codes. Approaching the limit $I(X, Y)$ requires that long segments of the input sequence are encoded together.

Example 2. Let us return to the setup of Example 1, and let us use the obtained conditional reconstruction probabilities

$$P(0|0) = P(1|1) = \tfrac{3}{4}$$
$$P(1|0) = P(0|1) = \tfrac{1}{4}$$

as our target. The average mutual information is then

$$I(X, Y) = \frac{3}{4}\log_2 3 - 1 \approx 0.1887$$

This value is significantly lower than the rate $\frac{1}{3}$ bits obtained by the code of Example1 . According to the Shannon theory, by increasing the block length of the codes, one can approach rate $I(X, Y) \approx 0.1887$ while keeping the conditional reconstruction probabilities arbitrarily close to the target values. □

This section only discussed the source coding of finite *iid* sources. The main theorem remains valid in more general setups as well, in particular for continuous valued sources and ergodic, stationary Markov sources, that is, sources with memory.

Rate-Distortion Theory

In lossy compression, usually one does not want to specify for each source symbol the whole conditional probability distribution of reconstruction symbols. Rather, one is concerned with the amount of distortion made in the reconstruction. A *distortion function* on source alphabet X and reconstruction alphabet Y is a mapping

$$d : X \times Y \rightarrow R_+$$

into the non-negative real numbers. The value $d(x,y)$ represents the amount of distortion caused when source symbol x gets reconstructed as y. Typically (but not necessarily) $X = Y$, $d(x,x) = 0$ for all $x \in X$, and $d(x,y) > 0$ when $x \neq y$. Common distortion functions in the continuous valued case $X = Y = \mathbb{R}$ are the *squared error distortion*

$$d(x,y) = (x - y)^2$$

and the *absolute error distortion*

$$d(x,y) = |x - y|$$

Hamming distortion refers to the function

$$d(x,y) = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{if } x \neq y \end{cases}$$

The distortion between two sequences of symbols is defined as the average of the distortions between the corresponding elements. One talks about the *mean squared error* (MSE) or the *mean absolute error* (MAE) if the distortion function is the squared error or the absolute error distortion, respectively. These measures are the most widely used error measures in digital signal processing.

Consider the setup of the previous section where source symbols x are reconstructed as symbols y with probabilities $P(y|x)$. This setup imposes distortion $d(x,y)$ with probability $P(x,y) = P(x)P(y|x)$, so the *expected distortion* is

$$\sum_{x \in X} \sum_{y \in Y} P(x)P(y|x)d(x,y)$$

The goal is to find the highest achievable compression rate for a given target distortion D, which leads to the problem of identifying conditional reconstruction probabilities $P(y|x)$ such that the expected distortion is at most D while $I(X, Y)$ is as small as possible. Note how we no longer "micromanage" the reconstruction by preselecting the conditional probabilities. Instead, we only care about the overall reconstruction error induced by the probabilities and look for the probabilities that give the lowest bitrate $I(X, Y)$.

This process leads to the definition of the *rate-distortion function* $R(D)$ of the source:

$$R(D) = \min\{I(X, Y)|P(y|x) \text{ such that } \sum_{x \in X}\sum_{y \in Y} P(x)P(y|x)d(x,y) \leq D\}$$

The minimization is over all conditional probabilities $P(y|x)$ that induce at most distortion D. According to the source coding statement of the previous section, $R(D)$ is the best achievable rate at distortion D.

Example 3. (2) Consider a binary source that emits symbols 0 and 1 with probabilities p and $1-p$, respectively, and let us use the Hamming distortion function. By symmetry we do not lose generality if we assume that

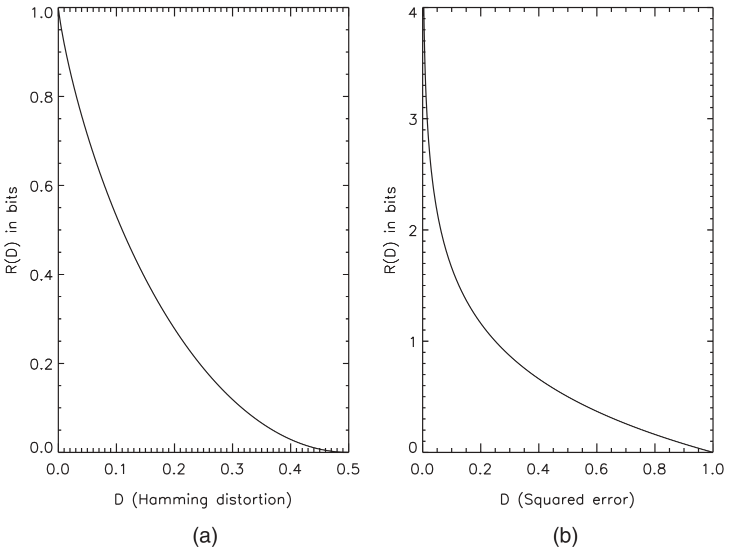


Figure 1. The rate-distortion function of (a) binary source with $p = 0.5$ and (b) Gaussian source with $\sigma = 1$. Achievable rates reside above the curve.

$p \leq 1 - p$. The rate-distortion function of this source is

$$R(D) = \begin{cases} h(p) - h(D), & \text{for } 0 \leq D \leq p \\ 0, & \text{for } D > p \end{cases}$$

where $h(r) = -r\log(r) - (1-r)\log(1-r)$ is the binary entropy function. See Fig. 1(a) for a plot of function $R(D)$.

For a continuous valued example, consider a source that emits real numbers *iid* under the Gaussian probability distribution with variance σ^2, that is, the probability density function of the source is

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$$

Consider the squared error distortion function. The corresponding rate distortion function is

$$R(D) = \begin{cases} \log(\sigma) - \frac{1}{2}\log(D), & \text{for } 0 < D \leq \sigma^2 \\ 0, & \text{for } D > \sigma^2 \end{cases}$$

see Fig. 1(b). □

Denote by D_{min} the smallest possible expected distortion at any bitrate (i.e., the distortion when the bitrate is high enough for lossless coding) and by D_{max} the smallest expected distortion corresponding to bitrate zero (i.e., the case where the reconstruction is done without receiving any information from the source). In the Gaussian example above, $D_{min} = 0$ and $D_{max} = \sigma^2$, and in the binary source example, $D_{min} = 0$ and $D_{max} = p$. Values of $R(D)$ are interesting only in the interval $D_{min} \leq D \leq D_{max}$. Function $R(D)$ is not defined for $D < D_{min}$, and it has value zero for all $D > D_{max}$. It can be proved that in the open interval $D_{min} < D < D_{max}$ the rate-distortion function $R(D)$ is strictly decreasing, continuously differentiable and convex downward. Its derivative approaches $-\infty$ when D approaches D_{min}(3,5,4).

In Example 3 , we had analytic formulas for the rate-distortion functions. In many cases such formulas cannot be found, and numerical calculations of rate-distortion functions become important. Finding $R(D)$ for a given finite *iid* source is a convex optimization problem and can be solved effectively using, for example, the Arimoto–Blahut algorithm (6,7).

QUANTIZATION

In the most important application areas of lossy compression, the source symbols are numerical values. They can be color intensities of pixels in image compression or audio samples in audio compression. Quantization is the process of representing approximations of numerical values using a small symbol set. In scalar quantization each number is treated separately, whereas in vector quantization several numbers are quantized together. Typically, some numerical transformation first is applied to initialize the data into a more suitable form before quantization. Quantization is the only lossy operation in use in many compression algorithms. See Ref. 8 for more details on quantization.

Scalar Quantization

A *scalar quantizer* is specified by a finite number of available *reconstruction values* $r_1, \ldots, r_n \in \mathbb{R}$, ordered $r_1 < r_2 < \ldots < r_n$, and the corresponding *decision intervals*. Decision intervals are specified by their boundaries $b_1, b_2, \ldots,$

$b_{n-1} \in \mathbb{R}$, also ordered $b_1 < b_2 < \ldots < b_{n-1}$. For each $i = 1, \ldots, n$, all input numbers in the decision interval (b_{i-1}, b_i) are quantized into r_i, where we use the convention that $b_0 = -\infty$ and $b_n = +\infty$. Only index i needs to be encoded, and the decoder knows to reconstruct it as value r_i. Note that all numerical digital data have been quantized initially from continuous valued measurements into discrete values. In digital data compression, we consider, additional quantization of discrete data into a more coarse representation to improve compression.

The simplest form of quantization is *uniform quantization*. In this case, the quantization step sizes $b_i - b_{i-1}$ are constant. This process can be viewed as the process of rounding each number x into the closest multiple of some positive constant b. Uniform quantizers are used widely because of their simplicity.

The choice of the quantizer affects both the distortion and the bitrate of a compressor. Typically, quantization is the only lossy step of a compressor, and quantization error becomes distortion in the reconstruction. For given distortion function and probability distribution of the input values, one obtains the expected distortion. For the analysis of bitrate, observe that if the quantizer outputs are entropy coded and take advantage of their uneven probability distribution, then the entropy of the distribution provides the achieved bitrate. The quantizer design problem then is the problem of selecting the decision intervals and reconstruction values in a way that provides minimal entropy for given distortion.

The classic Lloyd–Max (9,10) algorithm ignores the entropy and only minimizes the expected distortion of the quantizer output for a fixed number n of reconstruction values. The algorithm, hence, assumes that the quantizer output is encoded using fixed length codewords that ignore the distribution. The Lloyd–Max algorithm iteratively repeats the following two steps:

(i) Find reconstruction values r_i that minimize the expected distortion, while keeping the interval boundaries b_i fixed, and

(ii) Find optimal boundaries b_i, while keeping the reconstruction values fixed.

If the squared error distortion is used, the best choice of r_i in step (i) is the centroid (mean) of the distribution inside the corresponding decision interval $[b_{i-1}, b_i)$. If the error is measured in absolute error distortion, then the best choice of r_i is the median of the distribution in $[b_{i-1}, b_i)$. In step (ii), the optimal decision interval boundaries b_i in both distortions are the midpoints $\frac{r_i + r_{i+1}}{2}$ between consecutive reconstruction values.

Each iteration of (i) and (ii) lowers the expected distortion so the process converges toward a local minimum. A Lloyd–Max quantizer is any quantizer that is optimal in the sense that (i) and (ii) do not change it.

Modifications of the Lloyd–Max algorithm exist that produce entropy-constrained quantizers (8). These quantizers incorporate the bitrate in the iteration loop (i)–(ii). Also, a shortest path algorithm in directed acyclic graph can be used in optimal quantizer design (11).

Vector Quantization

From information theory, it is known that optimal lossy coding of sequences of symbols requires that long blocks of symbols are encoded together, even if they are independent statistically. This means that scalar quantization is suboptimal. In *vector quantization*, the input sequence is divided into blocks of length k, and each block is viewed as an element of $\mathbb{R}^k$, the k-dimensional real vector space. An n-level vector quantizer is specified by n reconstruction vectors $\vec{r}_1, \ldots, \vec{r}_n \in \mathbb{R}^k$ and the corresponding decision regions. Individual reconstruction vectors are called *code vectors*, and their collection often is called a *code book*.

Decision regions form a partitioning of $\mathbb{R}^k$ into n parts, each part representing the input vectors that are quantized into the respective code vector. Decision regions do not need to be formed explicitly, rather, one uses the distortion function to determine for any given input vector which of the n available code vectors $\vec{r}_i$ gives optimal representation. The encoder determines for blocks of k consecutive input values the best code vector $\vec{r}_i$ and communicates the index i to the decoder. The decoder reconstructs it as the code vector $\vec{r}_i$. Typically, each vector gets quantized to the closest code vector available in the code book, but if entropy coding of the code vectors is used, then the bitrate also may be incorporated in the decision.

Theoretically, vector quantizers can encode sources as close to the rate-distortion bound as desired. This encoding, however, requires that the block length k is increased, which makes this fact have more theoretical than practical significance.

The Lloyd–Max quantizer design algorithm easily generalizes into dimension k. The generalized Lloyd–Max algorithm, also known as the LBG algorithm after Linde, Buzo, and Gray (12), commonly is used with training vectors $\vec{t}_1, \ldots \vec{t}_m \in \mathbb{R}^k$. These vectors are sample vectors from representative data to be compressed. The idea is that although the actual probability distribution of vectors from the source may be difficult to find, obtaining large collections of training vectors is easy, and it corresponds to taking random samples from the distribution.

The LBG algorithm iteratively repeats the following two steps. At all times the training vectors are assigned to decision regions.

(i) For each decision region i find the vector $\vec{r}_i$ that minimizes the sum of the distortions between $\vec{r}_i$ and the training vectors previously assigned to region number i.

(ii) Change the assignment of training vectors: Go over the training vectors one by one and find for each training vector $\vec{t}_j$ the closest code vector $\vec{r}_i$ under the given distortion. Assign vector $\vec{t}_j$ in decision region number i.

In step (i), the optimal choice of $\vec{r}_i$ is either the coordinate wise mean or median of the training vectors in decision region number i, depending on whether the squared error or absolute error distortion is used.

Additional details are needed to specify how to initialize the code book and what is done when a decision region becomes empty. One also can incorporate the bitrate in the iteration loop, which creates entropy-constrained vector quantizers. For more details on vector quantization see, for example, Ref. 13.

DATA TRANSFORMATIONS

In data compression situations, it is rarely the case that the elements of source sequences are independent of each other. For example, consecutive audio samples or neighboring pixel values in images are highly correlated. Any effective compression system must take advantage of these correlations to improve compression—there is no sense in encoding the values independently of each other as then their common information gets encoded more than once. Vector quantization is one way to exploit correlations. Scalar quantization, however, requires some additional transformations to be performed on the data to remove correlation before quantization.

Prediction-Based Transformations

One way to remove correlation is to use previously processed data values to calculate a *prediction* $\hat{x}_i \in \mathbb{R}$ for the next data value $x_i \in \mathbb{R}$. As the decoder calculates the same prediction, it is enough to encode the *prediction error*

$$e_i = x_i - \hat{x}_i$$

In lossy compression, the prediction error is scalar quantized. Let $\bar{e}_i$ denote e_i quantized. Value $\bar{e}_i$ is entropy coded and included in the compressed file. The decoder obtains $\bar{e}_i$ and calculates the reconstruction value

$$\bar{x}_i = \bar{e}_i + \hat{x}_i$$

Note that the reconstruction error $x_i - \bar{x}_i$ is identical to the quantization error $e_i - \bar{e}_i$ so the quantization directly controls the distortion.

In order to allow the encoder and decoder to calculate an identical prediction $\hat{x}_i$, it is important that the predictions are based on the previously reconstructed values $\bar{x}_j$, not the original values $x_j, j < i$. In other words,

$$\hat{x}_i = f(\bar{x}_{i-1}, \bar{x}_{i-2}, \bar{x}_{i-3}, \ldots)$$

where f is the prediction function. In *linear prediction*, function f is a linear function, that is,

$$\hat{x}_i = a_1\bar{x}_{i-1} + a_2\bar{x}_{i-2} + \ldots + a_k\bar{x}_{i-k}$$

for some constants $a_1, a_2, \ldots, a_k \in \mathbb{R}$. Here, k is the *order* of the predictor. Optimal values of parameters $a_1, a_{2,} \ldots, a_k$ that minimize the squared prediction error can be found by solving a system of linear equations, provided autocorrelation values of the input sequences are known (14).

In prediction-based coding, one has strict control of the maximum allowed reconstruction error in individual source elements x_i. As the reconstruction error is identical to the quantization error, the step size of a uniform quantizer provides an upper bound on the absolute reconstruction error. Compression where a tight bound is imposed on the reconstruction errors of individual numbers rather than an average calculated over the entire input sequence is termed *near-lossless compression*.

Linear Transformations

An alternative to prediction-based coding is to perform a linear transformation on the input sequence to remove correlations. In particular, *orthogonal transformations* are used because they keep mean squared distances invariant. This method allows easy distortion control: The MSE quantization error on the transformed data is identical to the MSE reconstruction error on the original data.

Orthogonal linear transformation is given by an orthogonal square matrix, that is, an $n \times n$ matrix M whose rows (and consequently also columns) form an orthonormal basis of $\mathbb{R}^n$. Its inverse matrix is the same as its transpose M^T. The rows of M are the *basis vectors* of the transformation.

A data sequence of length n is viewed as an n-dimensional column vector $\vec{x} \in \mathbb{R}^n$. This vector can be the entire input, or, more likely, the input sequence is divided into blocks of length n, and each block is transformed separately. The transformed vector is $M\vec{x}$, which is also a sequence of n real numbers. Notice that the elements of the transformed vector are the coefficients in the expression of $\vec{x}$ as a linear combination of the basis vectors.

The inverse transformation is given by matrix M^T. In the following, we denote the i'th coordinate of $\vec{x}$ by $\vec{x}(i)$ for $i = 1$, 2,..., n. If the transformed data is quantized into $\vec{y} \in \mathbb{R}^n$, then the reconstruction becomes $M^T\vec{y}$. Orthogonality guarantees that the squared quantization error $\|M\vec{x} - \vec{y}\|^2$ is identical to the squared reconstruction error $\|\vec{x} - M^T\vec{y}\|^2$. Here, we use the notation

$$\|\vec{r}\|^2 = \sum_{i=1}^{n} \vec{r}(i)^2$$

for the square norm in $\mathbb{R}^n$.

KLT Transformation. The goal is to choose a transformation that maximally decorrelates the coordinates of the data vectors. At the same time, one hopes that as many coordinates as possible become almost zero, which results in low entropy in the distribution of quantized coordinate values. The second goal is called energy compaction.

Let us be more precise. Consider a probability distribution on the input data vectors with mean $\vec{m} \in \mathbb{R}^n$. The *variance* in coordinate i is the expectation of $(\vec{x}(i) - \vec{m}(i))^2$ when $\vec{x} \in \mathbb{R}^n$ is drawn from the distribution. The *energy* in coordinate i is the expectation of $\vec{x}(i)^2$. The total variance (energy) of the distribution is the sum of the n coordinate-wise variances (energies). Orthogonal transformation keeps the total variance and energy unchanged, but a shift of variances and energies between coordinates can occur. Variance (energy) compaction means that as much variance (energy) as possible gets shifted into as few coordinates as possible.

The *covariance* in coordinates i and j is the expectation of

$$(\vec{x}(i) - \vec{m}(i))(\vec{x}(j) - \vec{m}(j))$$

Coordinates are called uncorrelated if the covariance is zero. Decorrrelation refers to the goal of making the coordinates uncorrelated. Analogously, we can call the expectation of $\vec{x}(i)\vec{x}(j)$ the "coenergy" of the coordinates. Notice that the variances and covariances of a distribution are identical to the energies and coenergies of the translated distribution of $\mathbb{R}^n$ by the mean $\vec{m}$.

Example 4. Let $n = 2$, and let us build a distribution of vectors by sampling blocks from a grayscale image. The image is partitioned into blocks of 1×2 pixels, and each such block provides a vector $(x, y) \in \mathbb{R}^2$ where x and y are the intensities of the two pixels. See Fig. 2 (a) for a plot of the distribution. The two coordinates are highly correlated, which is natural because neighboring pixels typically have similar intensities. The variances and energies in the two coordinates are

	Energy	Variance
x	17224	2907
y	17281	2885
total	34505	5792

The covariance and coenergy between the x and y coordinates are 17174 and 2817, respectively.

Let us perform the orthogonal transformation

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

that rotates the points 45 degrees clockwise, see Fig. 2 (b) for the transformed distribution. The coordinates after the transformation have the following variances and energies:

	Energy	Variance
x	34426	5713
y	79	79
total	34505	5792

Energy and variance has been packed in the first coordinate, whereas the total energy and variance remained invariant. The effect is that the second coordinates became very small (and easy to compress), whereas the first coordinates increased.

After the rotation, the covariance and the coenergy are -29 and 11, respectively. The transformed coordinates are uncorrelated almost, but not totally. □

The two goals of variance compaction and decorrelation coincide. An orthogonal transformation called KLT (Karhunen–Loève transformation) makes all covariances between different coordinates zero. It turns out that at the same time KLT packs variance in the following, optimal way: For every $k \leq n$ the variance in coordinates 1,2,. . ., k after KLT is as large as in any k coordinates after any orthogonal transformation. In other words, no orthogonal transformation packs more variance in the first coordinate than KLT nor more variance in two coordinates than KLT, and so forth. The rows of the KLT transformation matrix are orthonormal eigenvectors of the *covariance matrix*, that is, the matrix whose element i,j is the covariance in coordinates i and j for all $1 \leq i, j \leq n$. If energy compaction rather than variance compaction is the goal, then one should replace the covariance matrix by the analogous coenergy matrix.

Example 5. In the case of Example 4 the covariance matrix is

$$\begin{pmatrix} 2907 & 2817 \\ 2817 & 2885 \end{pmatrix}$$

The corresponding KLT matrix

$$\begin{pmatrix} 0.708486 & 0.705725 \\ -0.705725 & 0.708486 \end{pmatrix}$$

consists of the orthonormal eigenvectors. It is the clockwise rotation of $\mathbb{R}^2$ by 44.88 degrees. □

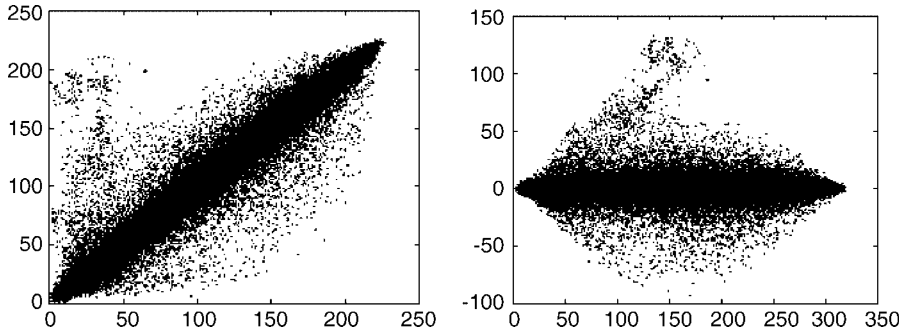


Figure 2. The sample distribution (a) before and (b) after the orthogonal transformation.

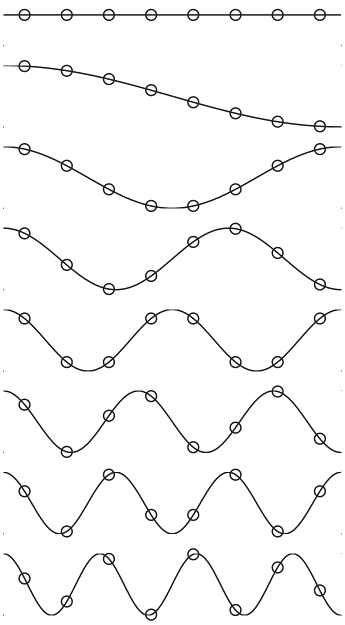


Figure 3. Size $n = 8$ DCT basis vectors.

KLT transformation, although optimal, is not used widely in compression applications. No fast way exists to take the transformation of given data vectors—one has to multiply the data vector by the KLT matrix. Also, the transformation is distribution dependent, so that the KLT transformation matrix or the covariance matrix has to be communicated to the decoder, which adds overhead to the bitrate. In addition, for real image data it turns out that the decorrelation performance of the widely used DCT transformation is almost as good. KLT is, however, an important tool in data analysis where it is used to lower the dimension of the data set by removing the less significant dimensions. This method also is known as the principal component analysis.

DCT Transformation. The *discrete cosine transform* (DCT) is an orthogonal linear transformation whose basis vectors are obtained by uniformly sampling the cosine function, see Fig. 3. More precisely, the element i, j of the $n \times n$ DCT matrix is

$$C_i \cos(i\pi \frac{2j+1}{2n})$$

where $i,j = 0,1,\ldots, n\text{–}1$ and C_i is the normalization factor

$$C_i = \begin{cases} \sqrt{1/n}, & \text{if } i = 0 \\ \sqrt{2/n}, & \text{otherwise} \end{cases}$$

Different basis vectors of DCT capture different frequencies present in the input. The first basis vector ($i = 0$) is constant, and the corresponding transformed value is called the DC coefficient. The DC coefficient gives the average of the input. Other transformed values are AC coefficients. DCT has good energy compaction properties where typically most energy is packed into the low frequency coefficients.

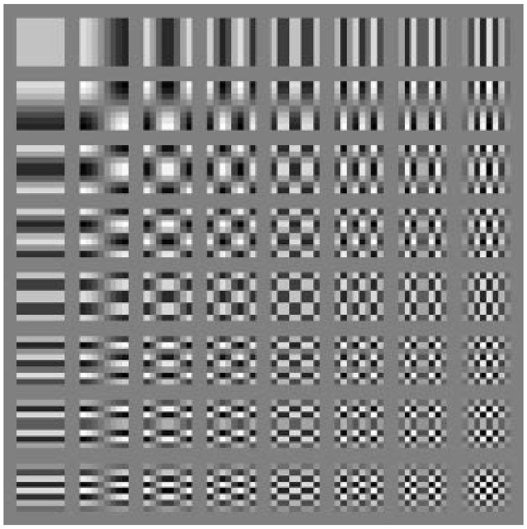


Figure 4. Basis vectors of the 8×8 DCT.

There are fast DCT algorithms that perform the transformation faster than the straightforward multiplication of the data by the transformation matrix. See Ref. 15 for these and other details on DCT.

Linear Transformations in Image Compression. In image data, the pixels have a two-dimensional arrangement. Correlations between pixels also are current in both horizontal and vertical directions. DCT, however, is natively suitable for one-dimensional signals because it separates frequencies in the direction of the signal. Also, other common transformations are designed for signals along a one-dimensional line. To use them on images, the transformation is done twice: first along horizontal lines of the image and then again on vertical lines on the output of the first round. (The same result is obtained if the vertical transformation is executed first, followed by the horizontal direction.) The combined effect again is an orthogonal transformation of the input image.

For example, the basis vectors of DCT on 8×8 image data is shown in Fig. 4. The transformation decomposes an input image into a linear combination of the shown basis vectors. Typically, high-frequency coefficients corresponding to basis vectors at the lower right corner are very small, whereas the low frequency basis vectors at the upper left corner capture most of the image energy. This fact is exploited in the JPEG image compression algorithm discussed below.

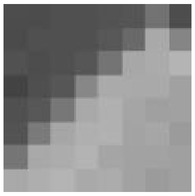
JPEG

This section outlines the JPEG (Joint Photographic Experts Group) image compression standard as an illustration of the concepts discussed in previous sections. JPEG is based on the DCT transformation of 8×8 image blocks;

uniform scalar quantization of the transformed values, where different step sizes can be used at different frequency components; ordering of the quantized values from the low frequency to the high frequency; and Huffman coding of the ordered, quantized coefficients. The bitrate and the quality are controlled through the selection of the quantizer step sizes. For details of JPEG, see Ref. 16.

If the input image is a color image, it has three color components, each of which is compressed as a grayscale image. But the commonly used RGB (red, green, and blue) color representation first is converted into a luminance–chrominance representation where the luminance component is the intensity and two chrominance components give the color. The chrominance components typically compress much better than the luminance component that contains most of the image information. In addition, the human visual system is not as sensitive to the chrominance data, so the chrominance components often are subsampled by removing half of the rows and/or columns. Also, the chrominance components can be quantized more heavily.

Next, the image is partitioned into non overlapping 8 × 8 squares. As an example, take the image block



or

77	76	80	80	83	85	114	77
75	80	80	80	87	105	169	133
81	77	80	86	116	167	171	180
67	79	86	135	170	169	169	161
80	87	119	168	176	165	159	161
83	122	166	175	177	163	166	155
117	168	172	179	165	162	162	159
168	174	180	169	172	162	155	160

extracted from a test picture. The intensity values are integers in the interval 0...255. First, 128 is subtracted from all intensities, and the result is transformed using the 8 × 8 DCT. The transformed values of the sample block—rounded to the closest integer—are

28	−158	−47	−5	−14	8	−17	5
−212	−61	56	32	−9	22	−9	7
−27	104	44	−12	−23	7	−6	9
−30	29	−50	−25	−1	11	−10	5
−11	21	−23	19	25	0	0	1
−3	2	−12	4	−4	−12	−2	−9
2	0	1	6	−1	−5	19	3
0	0	11	2	3	−2	11	−11

Notice that the high frequency values at the lower right corner are relatively small. This phenomenon is common as typical images contain more low than high frequencies. The next step is the lossy step of scalar quantization. Different frequencies may be quantized differently. Typically, the high-frequency values are quantized more coarsely than the low frequencies because the human visual system is less sensitive to high frequencies. The quantizer step sizes are specified in an 8 × 8 array, called the quantization table. In our example, let us use the following quantization table:

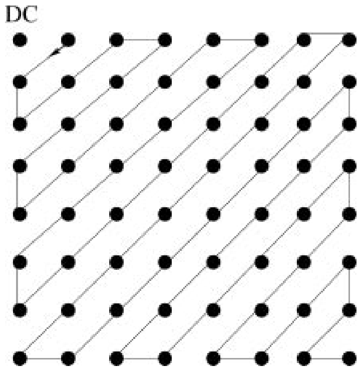
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

The transformed data values are divided by the corresponding element of the quantization table and rounded to the closest integer. The result is

2	−14	−5	0	−1	0	0	0
−18	−5	4	2	0	0	0	0
−2	8	3	0	−1	0	0	0
−2	2	−2	−1	0	0	0	0
−1	1	−1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Notice the large number of zeroes in the high-frequency part of the table. This large number is because of two factors: Higher frequencies were quantized more, and the input contain less high frequency information (energy) in the first place.

Next, the quantized values need to be entropy coded. Huffman coding (or arithmetic coding) is used. The DC coefficient (number 2 in our example) simply is subtracted from the DC coefficient of the previous block, and the difference is entropy coded. The other 63 values—the AC coefficients—are lined up according to the following zigzag order:



The purpose is to have the large low-frequency values first, and hopefully many zeroes at the end. In our example, this produces the sequence

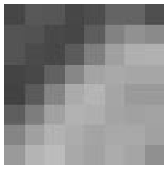
$$-14,\ -18,\ -2,\ -5,\ -5, 0, 4, 8,\ -2,\ -1, 2, 3, 2, -1, 0, 0, 0,$$
$$-2, 1, 0, 0, 0,\ -1,\ -1,\ -1, 0, 0, \ldots$$

where the end of the sequence contains only zeroes. Huffman coding is used where each Huffman code word represents a non zero value and the count of zeroes before it. In other words, Huffman code words are used for the non zero values only, although these code words also specify the positions of zeroes. A specific code word also exists for the end of the block that is inserted after the last non zero value. In this way, the long sequence of zeroes at the end can be skipped.

From the compressed file, the decoder obtains the quantized DCT output. Multiplying the values by the entries of the quantization table and applying the inverse DCT produces the reconstructed image block. In our example, this block is

70	87	86	71	77	100	100	78
84	82	72	69	93	130	145	138
84	74	71	91	127	160	175	177
68	72	97	135	163	169	166	166
66	90	133	171	180	165	154	154
94	125	161	178	174	164	161	163
132	160	178	172	164	168	168	161
154	180	188	169	162	172	165	144

or



The level of the compression artifacts is controlled by the selection of the quantization table. Typical artifacts in JPEG with coarse quantization are: blocking along the boundaries of the 8 × 8 blocks, blending of colors because of high quantization of chrominance components, and ringing on edges because of the loss of high-frequency information.

SUBBAND CODING AND WAVELETS

Typical image data contains localized high frequencies on the boundaries of objects. Within an object, usually large low frequency areas exist. Classical transformations into the frequency domain, such as DCT, measure the total amounts of various frequencies in the input. To localize the effect of sharp edges, DCT commonly is applied on relatively small image blocks. In this way, image edges only affect the compression of the block that contains the edge. This method, however, leads to blocking artifacts seen in JPEG.

Alternative linear, even orthogonal, transformations exist that measure frequencies locally. A simple example is the Haar transformation. It is based an a multilevel application of a 45 degree rotation of $\mathbb{R}^2$, specified by the matrix

$$M = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

On the first level, the input signal of length n is partitioned into segments of two samples and each segment is transformed using M. The result is shuffled by grouping together the first and the second coefficients from all segments, respectively. This shuffling provides two signals of length $n/2$, where the first signal consists of moving averages and the second signal of moving differences of the input. These two signals are called subbands, and they also can be viewed as the result of filtering the input signal by FIR filters whose coefficients are the rows of matrix M, followed by subsampling where half of the output values are deleted.

The high-frequency subband that contains the moving differences typically has small energy because consecutive input samples are similar. The low-frequency subband that contains the moving averages, however, is a scaled version of the original signal. The second level of the Haar transformation repeats the process on this low-frequency subband. This process again produces two subbands (signals of length $n/4$ this time) that contain moving averages and differences. The process is repeated on the low-frequency output for as many levels as desired.

The combined effect of all the levels is an orthogonal transformation that splits the signal into a number of subbands. Notice that a sharp transition in the input signal now only affects a small number of values in each subband, so the effect of sharp edges remains localized in a small number of output values.

Haar transformation is the simplest subband transformation. Other subband coders are based on the similar idea of filtering the signal with two filters—one low-pass and one high-pass filter—followed by subsampling the outputs by a factor of two. The process is repeated on the low-pass output and iterated in this fashion for as many levels as desired. Different subband coders differ in the filters they use. Haar transformation uses very simple filters, but other filters with better energy compaction properties exist. See, for example, Ref. 17 for more details on wavelets and subband coding.

When comparing KLT (and DCT) with subband transformations on individual inputs, one notices that although KLT packs energy optimally in fixed, predetermined coefficients, subband transformations pack the energy better in fewer coefficients, however, the positions of those coefficients depend on the input. In other words, for good compression one cannot order the coefficients in a predetermined zigzag order as in JPEG, but one has to specify also the position of each large value.

Among the earliest successful image compression algorithms that use subband transformations are the embedded zerotree wavelet (EZW) algorithm by J. M. Shapiro (18) and the set partitioning in hierarchical trees (SPIHT) algorithm by A. Said and W. A. Pearlman (19). The JPEG 2000 image compression standard is based on subband coding as well (20).

CONCLUSION

A wide variety of specific lossy compression algorithms based on the techniques presented here exist for audio, speech, image, and video data. In audio coding, linear prediction commonly is used. Also, subband coding and wavelets are well suited. If DCT type transformations are used, they are applied commonly on overlapping segments of the signal to avoid blocking artifacts.

In image compression, JPEG has been dominant. The wavelet approach has become more popular after the introduction of the wavelet-based JPEG 2000. At high quantization levels, JPEG 2000 exhibits blurring and ringing on edges, but it does not suffer from the blocking artifacts typical to JPEG.

Video is an image sequence, so it is not surprising that the same techniques that work in image compression also work well with video data. The main difference is the additional temporal redundancy: Consecutive video frames are very similar, and this similarity can be used to get good compression. Commonly, achieving this good compression is done through block-based motion compensation, where the frame is partitioned into blocks (say of size 16 × 16) and for each block a motion vector that refers to a reconstructed block in a previously transmitted frame is given. The motion vector specifies the relative location of the most similar block in the previous frame. This reference block is used by the decoder as the first approximation. The difference of the correct block and the approximation then is encoded as a still image using DCT. Common standards such as MPEG-2 are based on this principle. At high compression, the technique suffers from blocking artifacts, especially in the presence of high motion. This difficulty is because the motion vectors of neighboring blocks can differ significantly, which results in visible block boundaries, and because at high-motion areas the motion compensation leaves large differences to be encoded using DCT, which can be done within available bit budget only by increasing the level of quantization.

See Refs. 21–24 for more information on lossy compression and for details of specific algorithms.

BIBLIOGRAPHY

1. C. E. Shannon, A mathematical theory of communication, *Bell System Technical Journal*, **27**: 379–423; 623–656, 1948.
2. T. Cover and J. Thomas, *Elements of Information Theory*. New York: Wiley & Sons, 1991.
3. C. E. Shannon, Coding theorems for a discrete source with a fidelity criterion, *IRE Nat. Conv. Rec.*, **7**: 142–163, 1959.
4. T. Berger and J. D. Gibson, Lossy source coding, *IEEE Transactions on Information Theory*, **44**(6), 2693–2723, 1998.
5. T. Berger, *Rate Distortion Theory: A Mathematical Basis for Data Compression*, Englewood Cliffs, NJ: Prentice Hall, 1971.
6. S. Arimoto, An algorithm for computing the capacity of arbitrary discrete memoryless channels, *IEEE Transactions on Information Theory*, **18**(1): 14–20, 1972.
7. R. E. Blahut, Computation of channel capacity and rate- distortion function, *IEEE Trans. Information Theory*, **18**(4): 460–473, 1972.
8. R. M. Gray and D. L. Neuhoff, Quantization, *IEEE Transactions on Information Theory*, **44**: 2325–2384, 1998.
9. S. P. Lloyd, Least squared quantization in PCM. Unpublished, Bell Lab. 1957. Reprinted in *IEEE Trans. Information Theory*, **28**: 129–137, 1982.
10. J. Max, Quantizing for minimum distortion, *IEEE Trans. Information Theory*, **6**(1), 7–12, 1960.
11. D. Mureson and M. Effros, Quantization as histogram segmentation: Globally optimal scalar quantizer design in network systems, *Proc. IEEE Data Compression Conference 2002*, 2002, pp. 302–311.
12. Y. Linde, A. Buzo, and R. M. Gray, An algorithm for vector quantizer design, *IEEE Trans. Communications*, **28**: 84–95, 1980.
13. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. New York: Kluwer Academic Press, 1991.
14. J. D. Markel and A. H. Gray, *Linear Prediction of Speech*. New York: Springer Verlag, 1976.
15. K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. New York: Academic Press, 1990.
16. W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Compression Standard*. Amsterdam, the Netherlands: Van Nostrand Reinhold, 1992.
17. G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.
18. J. M. Shapiro, Embedded Image Coding Using Zerotrees of Wavelet Coefficients. *IEEE Trans. Signal Processing*, **41**(12), 3445–3462, 1993.
19. A. Said and W. A. Pearlman, A new fast and efficient image codec based on set partitioning in hierarchical trees, *IEEE Trans. Circuits and Systems for Video Technology*, **6**: 243–250, 1996.

20. D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression, Fundamentals, Standards, and Practice*. New York: Kluwer Academic Press, 2002.

21. K. Sayood, *Introduction to Data Compression*, 2nd ed. Morgan Kaufmann, 2000.

22. D. Salomon, *Data Compression: the Complete Reference*, 3rd ed. New York: Springer, 2004.

23. M. Nelson and J. L. Gailly, *The Data Compression Book*, 2nd ed. M & T Books, 1996.

24. M. Rabbani and P. W. Jones, *Digital Image Compression Techniques*. Society of Photo-Optical Instrumentation Engineers (SPIE) Tutorial Text, Vol. TT07, 1991.

Jarkko Kari
University of Turku
Turku, Finland

DATA HANDLING IN INTELLIGENT TRANSPORTATION SYSTEMS

INTRODUCTION

Within the domain of computer science, *data handling* is the coordinated movement of data within and between computer systems. The format of the data may be changed during these movements, possibly losing information during conversions. The data may also be filtered for privacy, security, or efficiency reasons. For instance, some information transmitted from an accident site may be filtered for privacy purposes before being presented to the general public (e.g., license numbers and names). The motivation for this article is to be an updated reference of the various mechanisms and best practices currently available for handling *intelligent transportation systems* (ITSs) data. In the sections that follow, data handling within ITS is discussed in detail.

What are Intelligent Transportation Systems?

Intelligent transportation systems apply computer, communication, and sensor technologies in an effort to improve surface transportation. The application of these technologies within surface transportation typically has been limited (e.g., car electronics and traffic signals). One goal of ITS is to broaden the use of these technologies to integrate more closely travelers, vehicles, and their surrounding infrastructure. Used effectively, ITS helps monitor and manage traffic flow, reduce congestion, provide alternative routes to travelers, enhance productivity, respond to incidents, and save lives, time, and money. Over the past ten years, the public and private sectors have invested billions of dollars in ITS research and development and in initial deployment of the resulting products and services.

Given the potential benefits of ITS, the U.S. Congress specifically supported an ITS program in the Transportation Equity Act for the 21st Century (TEA-21) in 1998. As defined by TEA-21, the ITS program provides for the research, development, and operational testing of ITSs aimed at solving congestion and safety problems, improving operating efficiencies in transit and commercial vehicles, and reducing the environmental impact of growing travel demand. Technologies that were cost effective were deployed nationwide within surface transportation systems as part of TEA-21.

Intelligent transportation systems can be broken down into three general categories: advanced traveler information systems, advanced traffic management systems, and incident management systems.

Advanced Traveler Information Systems (ATISs) deliver data directly to travelers, empowering them to make better choices about alternative routes or modes of transportation. When archived, this historical data provide transportation planners with accurate travel pattern information, optimizing the transportation planning process.

Advanced Traffic Management Systems (ATMSs) employ a variety of relatively inexpensive detectors, cameras, and communication systems to monitor traffic, optimize signal timings on major arterials, and control the flow of traffic.

Incident Management Systems, for their part, provide traffic operators with the tools to allow a quick and efficient response to accidents, hazardous spills, and other emergencies. Redundant communications systems link data collection points, transportation operations centers, and travel information portals into an integrated network that can be operated efficiently and “intelligently.”

Some example ITS applications include on-board navigation systems, crash notification systems, electronic payment systems, roadbed sensors, traffic video/control technologies, weather information services, variable message signs, fleet tracking, and weigh in-motion technologies.

MAJOR CHALLENGES

One major challenge in handling ITS data involves managing the broad spectrum of requirements inherent in a transportation system. ITS data must flow from and between a variety of locations and devices such as in-vehicle sensors, police dispatch centers, infrastructure sensors, computers, and databases. Each of these options has different bandwidth, formatting, and security requirements. A “one-solution-fits-all” approach is not appropriate in this type of environment. Fortunately, many standards can be applied in concert with one another to cover all requirements. These standards and how they are used are discussed in the sections that follow.

DATA HANDLING IN ITS

Data within an ITS originates at various sensors, such as in-pavement inductive loop detectors for measuring the presence of a vehicle, transponders for collecting toll fees, video cameras, and operator input. These data typically are collected and aggregated by a transportation agency for use in managing traffic flow and in detecting and responding to accidents. These data are often archived, and later data-mining techniques are used to detect traffic trends. These data also make their way to information providers for use in radio, television, and Web traffic reports.

The different requirements for ITS data handling depend on the medium used. For instance, the requirements for data handling in communications stress the efficient use of bandwidth and low latency, whereas data handling for data-mining applications require fast lookup capabilities.

Data Types and Formats

Various types of data are stored and manipulated by an ITS. The types of data are discussed in the sections that follow.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Traffic Statistics. Speed, travel time, volume, and occupancy data or other numeric measurements are used to characterize the flow of vehicles at a specific point or over a specific segment of roadway. These data can be generated from many types of detection systems, such as loop detectors, microwaves, infrared or sonic detectors, video image detection, automatic vehicle identification, license plate matching systems, and wireless phone probes.

Weather/Environmental. Wind speed, wind direction, temperature, humidity, visibility, and precipitation data are collected by weather stations. This information can be used to determine whether road salt is needed because of icing conditions or whether danger exists to large vehicles because of high wind speeds.

Video/Images. Video cameras are mounted high to provide unobstructed views of traffic. Video data are in the form of encoded data streams or still images. Video feeds are used to determine the cause and location of incidents and to determine the overall level of service (congestion).

Incident/Event Reports. Incidents, construction/ maintenance, events, and road conditions are some types of data collected. This information usually is entered manually into the "system" because an automated means of detecting an incident has yet to be developed. The reports provide descriptive information on planned or unplanned occurrences that affect or may affect traffic flow.

Data Attributes. The following attributes are common within ITS data: *Accuracy*—How precise is the data? *Detail*—Is the data a direct measurement or an estimated/indirect measurement? *Timeliness*—How fresh is the data? *Availability/Reliability*—How dependable is the flow of data? *Density*—How close together/far apart are the data collection sources? *Accessibility*—How easy is the data accessed by a data consumer? *Confidence*—Is the data trustworthy? *Coverage*—Where is the data collected?

Standards for Data Handling

Standards provide a mechanism to ensure compatibility among various disparate systems. These standards reduce costs because software and hardware can be developed to make use of a handful of standards rather than of hundreds of proprietary protocols.

The following organizations are involved in the development of ITS standards: the American Association of State Highway and Transportation Officials (AASHTO), American National Standards Institute (ANSI), American Society for Testing & Materials (ASTM), Consumer Electronics Association (CEA), Institute of Electrical and Electronics Engineers (IEEE), Institute of Transportation Engineers (ITE), Society of Automotive Engineers (SAE), National Electrical Manufacturers Association (NEMA), and the National Transportation Communications for ITS Protocol (NTCIP). Note that NTCIP is a joint effort among AASHTO, ITE, and NEMA.

The Federal Geographic Steering Committee (FGSC) provides staff to support the U.S. Department of the Interior to manage and facilitate the National Spatial Data Infrastructure (NSDI) (1).

Data Formats and Communication Protocols. Standards are built off of one another. For instance, ITS standards make use of standards for data formatting and communications, such as Extensible Markup Language (XML), Common Object Request Broker Architecture (CORBA), or Abstract Syntax Notation number One (ASN.1). To better understand ITS standards, these underlying standards will be discussed first.

Extensible Markup Language. XML is a standard of the World Wide Web Consortium (W3C) (2). XML is a means of encoding data so that computers can send and receive information. XML allows computers to understand the information content of the data and act on that content (e.g., process the information, display the information to a human, store the information in a database, and issue a command to a field device). Unlike most computer encoding standards (e.g., Basic Encoding Rules, Octet Encoding Rules, Packed Encoding Ruled, Common Data Representation, and Hypertext Markup Language), no single set of encoding rules exists for XML. Instead, XML encoding rules are customized for different applications. Furthermore, XML encoding rules include a mechanism for identifying each element of an XML document or message. See Ref. 3 for an overview of XML use in ITS applications.

The advantages of using XML for ITS data handling are for communications. It has pervasive support among software companies, standards organizations, and government agencies. Also, XML formatted data are often bundled with Web services over Hyper-Text Transmission Protocol (HTTP) via SOAP (4) or XML-RPC (5). This bundling allows the data and associated commands to traverse firewalls more easily. Some may see this process as a disadvantage, however, because it basically is hiding the services from network administrators and granting potentially unsecured access to important functions.

XML format is not well suited for data storage, however, because its hierarchal data structure does not lend itself to easy storage and retrieval from relational databases. However, XML databases, which natively can store, query, and retrieve XML-formatted documents, are now available (6).

A disadvantage to XML-formatted data is that the tags and the textual nature of the documents tend to make them much more verbose. This verbosity, in turn, requires higher bandwidth communication links and more data processing to encode and decode ITS data as compared with CORBA or ASN.1 (7,8).

Common Object Request Broker Architecture. CORBA is an Object Management Group (OMG) standard for communications between computers (9). As part of CORBA, the interface definition language (IDL) provides a platform and computer language-independent specification of the data to be transmitted and the services that are available to a CORBA application. The OMG requires CORBA implementations to make use of the Internet Inter-ORB Protocol

(IIOP) for encoding messages over the Internet. This protocol ensures that CORBA implementations from different vendors running on different operating systems can interact with one another. NTCIP has a CORBA-based standard for communications between transportation centers, NTCIP 2305 (10). CORBA inherently is object oriented, which allows for the definition of both data structures and for commands in the form of interfaces.

One advantage to using CORBA are its relatively lower bandwidth requirements as compared with XML-based communications. CORBA has wide industry support, and many open-source implementations are available (11,12). Because CORBA is based on the IIOP communications standard, the various commercial and open-source implementations can interoperate with one another. CORBA has the advantage, as compared with ASN.1 or XML, in that it describes both the structure of the data to be transmitted and what is to be done with the data. Finally, many counterpart extensions to CORBA handle such things as authentication.

The disadvantages to CORBA are that it requires specialized knowledge to integrate it into ITS data handling systems. Also, because CORBA does not make use of the HTTP protocol, it does not traverse Web firewalls as easily as XML-RPC or SOAP.

Abstract Syntax Notation. ASN.1 is a standard of the International Standards Organization (ISO) that defines a formalism for the specification of abstract data types. ASN.1 is a formal notation used for describing data transmission protocols, regardless of language implementation and physical representation of these data, whatever the application, whether complex or very simple. ASN/DATEX is a NTCIP standard (NTCIP 2304). ASN.1 is not a communication or data storage mechanism, but it is a standard for expressing the format of complex data structures in a machine-independent manner. Many encoding rules can encode and decode data via ASN.1 such as the basic encoding rules (BERs), canonical encoding rules (CERs), and distinguished encoding rules (DERs) (13).

ASN.1 has the advantage that it unambiguously defines the structure for ITS data and the various encoding schemes allow for efficient transmission of the data over even lower bandwidth communication systems. Many software libraries are also available that handle the encoding and decoding of ASN.1 data structures (14).

ITS Data Bus. Chartered in late 1995, the Data Bus Committee is developing the concept of a dedicated ITS data bus that may be installed on a vehicle to work in parallel with existing automotive electronics (15). When complete, the data bus will facilitate the addition of ITS electronics devices to vehicles without endangering any of its existing systems.

The ITS data bus will provide an open architecture to permit interoperability that will allow manufacturers, dealers, and vehicle buyers to install a wide range of electronics equipment in vehicles at any time during the vehicle's lifecycle, with little or no expert assistance required. The goal of the Data Bus Committee is to develop SAE recommended practices and standards that define the message formats, message header codes, node IDs, application services and service codes, data definitions, diagnostic connectors, diagnostic services and test mode codes, ITS data bus–vehicle bus gateway services and service codes, network management services/functionality, and other areas as may be needed.

National ITS Architecture. The National ITS Architecture provides a common framework for planning, defining, and integrating intelligent transportation systems (16). It is a mature product that reflects the contributions of a broad cross section of the ITS community: transportation practitioners, systems engineers, system developers, technology specialists, and consultants. The architecture provides high-level definitions of the functions, physical entities, and information flows that are required for ITS. The architecture is meant to be used as a planning tool for state and local governments.

Traffic Management Data Dictionary. The Data Dictionary for Advanced Traveler Information Systems, Society of Automotive Engineers (SAE) standard J2353, provides concise definitions for the data elements used in advanced traveler information systems (ATIS) (17). These definitions provide a bit-by-bit breakdown of each data element using ASN.1. The traffic management data dictionary is meant to be used in conjunction with at least two other standards, one for defining the message sets (e.g., SAE J2354 and SAE J2369) and the other for defining the communication protocol to be used.

TransXML. XML Schemas for the exchanges of Transportation Data (TransXML) is a fledgling standard with the goal to integrate various standards such as LandXML, aecXML, ITS XML, and OpenGIS into a common framework. This project has not had any results yet; see Ref. 18 for more information.

Geography Markup Language. Geography markup language (GML) is an OpenGIS standard based on XML that is used for encoding and storing geographic information, including the geometry and properties of geographic features (19). GML was developed by an international, nonprofit standards organization, the Open GIS Consortium, Inc. GML describes geographic features using a variety of XML elements such as features, coordinate referencing systems, geometry, topology, time, units of measure, and generalized values.

Spatial Data Transfer Standard. The Spatial Data Transfer Standard (SDTS) is a National Institute of Standards (NIST) standard for the exchange of digitized spatial data between computer systems (20). SDTS uses ISO 8211 for its physical file encoding and breaks the file down into modules, each of which is composed of records, which in turn are composed of fields. Thirty-four different types of modules currently are defined by the standard, some of which are specialized for a specific application.

LandXML. LandXML is an industry-developed standard schema for the exchange of data created during land planning, surveying, and civil engineering design processes by different applications software. LandXML was developed by an industry consortium of land developers, universities, and various government agencies. LandXML is used within GIS applications, survey field instruments, civil engineering desktop and computer aided design (CAD)-based applications, instant three-dimensional (3-D) viewers, and high-end 3D visualization rendering applications. LandXML has provisions for not only the storage of a feature's geometry, but also for its attributes, such as property owner and survey status. LandXML can be converted readily to GML format.

Scalable Vector Graphics. Scalable vector graphics (SVG) is an XML-based standard for the storage and display of graphics using vector data and graphic primitives. It primarily is used in the design of websites, but also it has application for the display and transfer of vector-based geographic data. It does not, however, have specific provisions for the attributes that are associated with geographic data such as road names and speed limits.

Location Referencing Data Model. The location referencing data model is a National Cooperative Highway Research Program (NCHRP) project that defines a location referencing system. Within this system, a location can be defined in various formats such as mile points or addresses. It also provides for the conversion of a location between the various formats and for the definition of a location in one, two, or more dimensions (21).

International Standards Organization Technical Committee 211. The International Standards Organization Technical Committee 211 (ISO TC/211) has several geographic standards that are of importance to intelligent transportation systems: Geographic information—Rules for Application Schema; ISO 19123, Geographic information—Schema for coverage geometry and functions; and ISO 19115, Geographic information—Metadata.

Data Mining and Analysis

Data mining and analysis of transportation data are useful for transportation planning purposes (e.g., transit development, safety analysis, and road design). Data quality especially is important because the sensors involved can malfunction and feed erroneous data into the analysis process (22). For this reason, ITS data typically needs to be filtered carefully to assure data quality. This filtering is accomplished by throwing out inconsistent data. For instance, a speed detector may be showing a constant free flow speed while its upstream and downstream counterparts show much slower speeds during peak usage hours. In this case, the inconsistent detector data would be thrown out and an average of the up and downstream detectors would be used instead.

BIBLIOGRAPHY

1. National Spatial Data Infrastructure. Available: http://www.geo-one-stop.gov/.
2. Extensible Markup Language (XML) 1.0 Specification. Available: http://www.w3.org/TR/REC-xml/.
3. XML in ITS Center-to-Center Communications. Available: http://www.ntcip.org/library/documents/pdf/9010v0107_XML_in_C2C.pdf.
4. SOAP Specifications. Available: http://www.w3.org/TR/soap/.
5. XML-RPC Home Page. Available: http://www.xmlrpc.com/.
6. XML:DB Initiative. Available: http://xmldb-org.sourceforge.net/.
7. James Kobielus, Taming the XML beast,. Available: http://www.networkworld.com/columnists/2005/011005kobielus.html.
8. R. Elfwing, U. Paulsson, and L. Lundberg, Performance of SOAP in web service environment compared to CORBA, *Proc. of the Ninth Asia-Pacific Software Engineering Conference*, 2002, pp. 84.
9. CORBA IIOP Specification. Available: http://www.omg.org/technology/documents/formal/corba_iiop.htm.
10. NTCIP 2305 - Application Profile for CORBA. Available: http://www.ntcip.org/library/documents/pdf/2305v0111b.pdf.
11. MICO CORBA. Available: http://www.mico.org/.
12. The Community OpenORB Project. Available: http://openorb.sourceforge.net/.
13. X.690 ASN.1 encoding rules: Specification of Basic Encoding Rules, Canonical Encoding Rules and Distinguished Encoding Rules. Available: http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf.
14. ASN.1 Tool Links. Available: http://asn1.elibel.tm.fr/links.
15. ITS Data Bus Committee. Available: http://www.sae.org/technicalcommittees/databus.htm.
16. National ITS Architecture. Available: http://itsarch.iteris.com/itsarch/.
17. SAE J2353 Advanced Traveler Information Systems (ATIS) DataDictionary. Available: http://www.standards.its.dot.gov/Documents/J2353.pdf.
18. XML Schemas for Exchange of Transportation Data (TransXML). Available: http://www4.trb.org/trb/crp.nsf/0/32a8d2b6bea6dc3885256d0b006589f9?OpenDocument.
19. OpenGIS® Geography Markup Language (GML) ImplementationSpecification. Available: http://www.opengis.org/docs/02-023r4.pdf.
20. Spatial Data Transfer Standard (SDTS). Available: http://ssdoo.gsfc.nasa.gov/nost/formats/sdts.html.
21. N. Koncz, and T. M. Adams, A data model for multi-dimensional transportation location referencing systems, *Urban Regional Informa. Syst. Assoc. J.*, **14**(2), 2002. Available: http://www.urisa.org/Journal/protect/Vol14No2/Koncz.pdf.
22. M. Flinner, and H. Horsey, Traffic data edit procedures pooled fund study traffic data quality (TDQ), 2000, SPR-2 (182).

JOHN F. DILLENBURG
PETER C. NELSON
University of Illinois at Chicago
Chicago, Illinois

DATA PRIVACY

The definition of privacy varies greatly among societies, across time, and even among individuals; with so much variation and interpretation, providing a single concrete definition is difficult. Several concepts that are related to privacy are as follows:

Solitude: The right of an individual to be undisturbed by, or invisible from, some or all other members of society, including protection of the home, work, or a public place from unwanted intrusion.

Secrecy: The ability to keep personal information secret from some or all others, including the right to control the initial and onward distribution of personal information and the ability to communicate with others in private.

Anonymity: The right to anonymity by allowing an individual to remain nameless in social or commercial interaction.

LEGAL BACKGROUND

In their influential article entitled "The Right to Privacy" published in the Harvard Law Review in 1890, Samuel D. Warren and Louis D. Brandeis described how "[i]nstantaneous photographs and newspaper enterprise have invaded the sacred precincts of private and domestic life; and numerous mechanical devices threaten to make good the prediction that 'what is whispered in the closet shall be proclaimed from the house-tops.'"

Warren and Brandeis described how U.S. common law could be used to protect an individual's right to privacy. The article has been very influential in the subsequent development of legal instruments to protect data privacy. In 1948, the General Assembly of the United Nations passed the Declaration of Human Rights, of which Article 12 states, "[n]o one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honour and reputation. Everyone has the right to the protection of the law against such interference or attacks." The majority of countries in the world now consider privacy as a right and have enacted laws describing how and when citizens can expect privacy.

The level of legal protection of privacy varies from one country to another. In the United States, there is no general privacy protection law to regulate private industry; instead specific federal legislation is introduced to deal with particular instances of the collection and processing of personal information. In addition, some states have amended their consititutions to include specific rights to privacy. In areas not covered by specific federal or state laws, companies are left to self-regulate. The new executive post of Chief Privacy Officer has been adopted by some companies, with responsibilities including the development of privacy policy, tracking legislation, monitoring competitors, training employees, and if necessary, performing damage control in the press.

The European Union has a comprehensive framework to control the collection and distribution of personally identifiable information by both state and private-sector institutions. The 1998 EU Data Protection Directive is the latest piece of European legislation that follows on from a series of national laws initially developed in the 1970s and earlier EU directives and OECD guidelines.

The EU Directive requires that all personal data are as follows:

- Processed fairly and lawfully
- Collected for explicit purposes and not further processed in any way incompatible with those purposes
- Relevant and not excessive in relation to the collected purposes
- Accurate and up to date
- Kept in a form that identifies an individual for no longer than neccessary to meet the stated purpose

Companies, governments, and in some cases individuals who process personal information must register their intent to do so with an independent supervisory authority and outline the requirement for, and purpose of, data processing. The supervisory authority has the power to investigate the processing of personal data and impose sanctions on those who do not adhere to the Directive. Individuals are entitled to receive a copy of their personal data held by a third-party "at reasonable intervals and without excessive delay or expense."

The European model has been widely copied elsewhere, in part because the Directive requires adequate levels of protection for the processing of personal data be available in all countries that receive personal data from within the EU. The provision of adequate protection is open to interpretation, and in some cases, it has been condemned for being too low; for example, the 2004 agreement between the European Commission and the United States to share airline passenger screening data has been widely criticized.

THE EFFECT OF TECHNOLOGY ON PRIVACY

Developments in science and technology continue to provide individuals, governments, and private industry with new ways to invade the privacy of others. In particular, the ability of computers to record, process, and communicate personal data has increased exponentially during the latter decades of the twentieth century. This increase in computing power has been used by governments, companies, and individuals to process an increasing quantity of personally identifiable data. Globalization has driven the desire to distribute personal information to different parts of the world and has encouraged the development of standards

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

such as the Internet Protocol suite and XML, enabling an efficient exchange of digital information on a global scale.

The widespread use of the personal computer and the Internet has provided a variety of both legitimate and malicous reasons for individuals to process personal data. Social networking websites allow people to search for user profiles matching certain criteria. Similarly, querying a Web search engine for an individual's name reveals a wealth of information about the person, including archived posts on mailing lists and newsgroups, personal home pages, business web pages, and telephone numbers. Often this information is intentionally published by, or on behalf of, the individual, but this copious amount of information can be easily misused. More serious invasions of privacy are possible when personal computers are connected to the Internet and contain sensitive personal data, for example, financial records and medical information. Many computers today are relatively insecure and are easy prey for malicious code in the form of viruses, trojans, and worms. Once compromised, a computer falls under the complete control of an attacker who may freely access any private data stored there.

The quantity and quality of personal information available on the Internet can be sufficient for a malicious person to perform *identity theft*, allowing them to impersonate a victim for illicit financial gain. *Cyberstalkers* use information available online to harrass their victims, sometimes with violent consequences. Anti-abortion activists have created websites detailing the home addresses of abortion doctors and photos of women who receive an abortion; these details have then been used by violent extremists, sometimes resulting in murder.

Governments process personal data for a variety of different purposes. Census data are collected from citizens in many countries to enable strategic planning of state and private sector services. Financial data such as bank account transactions are examined in the fight against organized crime, and biological data such as photographs, fingerprints, iris scans, and DNA profiles can be collected for both research and forensic analysis purposes. Traditionally, different government departments, like separate companies, have kept separate databases, but in some cases, disparate systems are in the process of, or have already been merged, enabling far more in-depth citizen profiling.

Companies compile personal information about customers to process orders, determine effective marketing strategies, and direct advertisements toward individuals who are likely to be receptive to them. Targetted advertisements are often used to increase consumer loyalty by providing discount points, tokens, or coupons on products of interest to a specific individual.

In economics, the act of charging different prices to different customers is known as *price discrimination* and is common practice in many markets worldwide. There are several different mechanisms by which sellers can perform price discrimination, including individual bartering, schemes of quantity discounts, and market segmentation. Sellers who have collected detailed personal information on buyers can employ a fourth mechanism. They can compute directly how much a user *can* pay for a product.

In markets with large up-front costs and low production costs, sellers can sacrifice the privacy of buyers to good effect, maximizing their profits without having to unnecessarily inconvenience anyone. For example, sellers no longer need to employ market segmentation to force poorer buyers to purchase third-class tickets; instead everyone can get the same high-quality service but for different prices: a potential social benefit. Societies have yet to decide whether this represents a reasonable invasion of privacy or something requiring government regulation.

Small and mobile computational devices are becoming increasingly popular. The GSM Association estimated that in early 2004, more than one billion people—one sixth of the world's population—had a GSM mobile phone. As devices like these continue to proliferate, the volume of personal information processed by computers will continue to increase, not just because there are more devices, but because these devices increasingly need to collect more personal information to function effectively (see UBIQUITOUS COMPUTING). As a consequence, providing privacy in the twenty-first century will become increasingly difficult and important.

USING TECHNOLOGY TO PROTECT PRIVACY

Although computer-related technologies are increasing the ability of individuals, companies, and governments to invade personal privacy, technology can also be used to increase the level of privacy available to individuals. Several solutions can be used depending on the level of trust placed in the recipients of private information.

Secrecy

Developments in cryptography have enabled information and communications to be encrypted, maintaining their secrecy in the presence of attacks by other individuals, companies, and even governments. In the latter part of the twentieth century, governments were reluctant to allow individuals access to strong encryption and therefore attempted to control the distribution of strong encryption technology. The development of the personal computer made governmental restriction on strong encryption ultimately impossible, because individuals could purchase computers powerful enough to perform both symmetric and public key cryptography in software.

Pretty Good Privacy (PGP) was one of the first programs developed to take advantage of the power of personal computers and offer individuals access to strong encryption schemes to keep documents secret from corporations and governments; it was so successful at this that its author, Phil Zimmerman, was charged with (but later acquitted of) violating U.S. export laws. More recently, Transport Layer Security (TLS) has been developed for the World Wide Web to enable end-to-end privacy of communications between two participants. As such, TLS has been a key enabling technology in the development of e-commerce over the Internet, enabling buyers and sellers to exchange credit card information in a secure way.

Access Control

Cryptography is excellent at protecting privacy when the data are controlled by its owner and it is shared with relatively few other parties who trust each other to keep the data secret. Sharing data with a larger set of participants increases the likelihood that trust is misplaced in at least one party, thereby jeopardizing data secrecy. The number of participants with access to private information can often be reduced by implementing *multilateral security*, where information is stored in compartments accessable to only a few individuals, and the flow of information between compartments is restricted (see COMPUTER SECURITY).

A recent example of using multilateral security to protect privacy is the British Medical Association model for controlling access to electronic patient records. In the model, patients do not have a single electronic record; rather, they have a set of records (or compartments), each of which has a separate list of health-care workers who have the permission to read or append information to it. The flow of information between different records is restricted to prevent particularly sensitive information detailed in one record, such as a positive HIV test, from being introduced into other records.

Reciprocity

Configuring fine-grained access control parameters can be time consuming and difficult to get right. An alternative solution is to use *reciprocity*, where two or more entities agree to the mutual exchange of private data. The exchange of information is symmetric if each party in the reciprocal transaction reveals the same piece of private data to all other parties; for example, three friends share their current mobile phone location with each other through their mobile phone operator. An exchange of data is asymmetric if information is provided in return for the knowledge of the recepient's identity. For example, in the United Kingdom, consumer credit ratings employ reciprocity: If a company queries a credit rating of an individual, that fact is recorded in the database so that when an individual queries their credit rating at a later date, they can see which companies examined their record.

Asymmetric reciprocity requires those requesting data to have a reputation worth protecting so that they obey acceptable social conventions when asking for information. Therefore, an abusive information request should reduce the attacker's reputation so that a requester with a poor reputation forfeits all future access. Symmetric reciprocity requires that the information shared by each participant is of similar value and therefore constitutes a fair exchange. This is not always true; for example, the home phone number of a famous individual may be considered more valuable than the phone number of an ordinary citizen.

A centralized authority is usually required to enforce reciprocity by authenticating both the identities of those who access private data and the actual content of the mutually shared information. Authentication guarantees the identities of the parties, and the content of the exchanged information cannot be forged; therefore, the central authority must be trusted by all parties who use the system.

Anonymity

Sometimes there no trust relationship exists between the owner of personal information and a third-party, and yet they still want to exchange information. For example, individuals may not want to reveal the contents of their medical records to drug companies; yet the development of pharmaceutical products clearly benefits from access to the information contained within such documents. It is often possible to provide access to a *subset* of the data that satisfies the demands of the third party and simultaneously protects the privacy of an individual through *anonymization*.

Anonymization protects an individual's privacy by removing all personally identifiable data before delivering it to an untrusted third party. Therefore, once data are successfully anonymized an adversary cannot infer the real-world individual represented by the data set. Anonymization is not an easy process; it is not sufficient to simply remove explicit identifiers such as a name or telephone number, because a combination of other attributes may enable a malicious data recipient to infer the individual represented by the data set. For example, in the set of medical records for Cambridge, England, there may be only one 42-year-old professor who has lost sight in one eye. If the data presented to the third party contains information concerning the individual's home city, profession, date of birth, and ophthalmology in sufficient detail, then it may be possible to associate this data with a real-world entity and therefore associate any other data in this record with a concrete identity. In this case, the privacy of the professor is effectively lost and the contents of his medical record, as presented to the third party, are revealed.

Successful anonymization may require the values in the released data set to be modified to prevent the third party from inferring the real-world identity associated with a record. For example, reducing the accuracy of the individual's age from 42 to the range (40–50) may prevent an attacker associating an identity with the record. Whether this reduction alone is sufficient depends on the data held in the other records (in the example above, it depends on the number of other professors between the ages of 40 and 50 with sight in only one eye). In general, anonymization of data is achieved through a thorough statistical analysis of the data set that takes into account the amount of information known by an attacker; such analysis is called Statistical Disclosure Control.

IS PRIVACY ALWAYS WORTH PROTECTING?

The rigorous protection of privacy is not always of benefit to the individual concerned or society at large. Professor Anita Allen observed that privacy in the home and workplace has been a problem for women where it led to "imposed modesty, chastity, and domestic isolation." Such enforced solitude can prevent the exposure of criminal acts such as domestic abuse. Allowing government ministers and departments privacy in their professional duties is at odds with the principles of openness and accountability. Most people would agree on the need for secrecy and privacy in the realm of international espionage but would hesitate in restricting the freedom of the press to uncover corruption and financial irregularities.

Similarly, company directors are ultimately accountable to shareholders; however, the desire for openness with investors must be reconciled with the need to protect business secrets from competitors. Most countries use legal measures to force businesses to make certain information publically available, for example, the filing of public accounts; yet financial scandals are still not uncommon. In many instances, an individual's right to privacy must be balanced with the need to protect themselves or others.

FURTHER READING

Web Links

Center for Democracy and Technology. Available: http://www.cdt.org.

Electronic Frontier Foundation. Available: http://www.eff.org.

Electronic Privacy Information Center. Available: http://www.epic.org.

Privacy International, Privacy and Human Rights Survey. Available: http://www.privacyinternational.org/survey.

Books

R. Anderson, *Security Engineering*, New York: Wiley, 2001.

S. Garfinkel, *Database Nation: The Death of Privacy in the 21st Century*, O'Reilly & Associates, 2001.

L. Lessig, *Code and Other Laws of Cyberspace*, Basic Books, 2000.

J. Rosen, *The Unwanted Gaze: The Distruction of Privacy in America*, Vintage Books, 2001.

L. Willenborg and T. de Waal, *Elements of Statistical Disclosure Control*, Lecture Notes in Statistics, Vol. 155, New York: Springer, 2001.

Alastair Beresford
David Scott
University of Cambridge
Cambridge, United Kingdom

DATA SEARCH ENGINE

INTRODUCTION

The World Wide Web was first developed by Tim Berners-Lee and his colleagues in 1990. In just over a decade, it has become the largest information source in human history. The total number of documents and database records that are accessible via the Web is estimated to be in the hundreds of billions (1). By the end of 2005, there were already over 1 billion Internet users worldwide. Finding information on the Web has become an important part of our daily lives. Indeed, searching is the second most popular activity on the Web, behind e-mail, and about 550 million Web searches are performed every day.

The Web consists of the Surface Web and the Deep Web (Hidden Web or Invisible Web). Each page in the Surface Web has a logical address called Uniform Resource Locator (URL). The URL of a page allows the page to be fetched directly. In contrast, the Deep Web contains pages that cannot be directly fetched and database records stored in database systems. It is estimated that the size of the Deep Web is over 100 times larger than that of the Surface Web (1).

The tools that we use to find information on the Web are called *search engines*. Today, over 1 million search engines are believed to be operational on the Web (2). Search engines may be classified based on the type of data that are searched. Search engines that search text documents are called *document search engines*, whereas those that search structured data stored in database systems are called *database search engines*. Many popular search engines such as Google and Yahoo are document search engines, whereas many e-commerce search engines such as Amazon.com are considered to be database search engines. Document search engines usually have a simple interface with a textbox for users to enter a query, which typically contains some key words that reflect the user's information needs. Database search engines, on the other hand, usually have more complex interfaces to allow users to enter more specific and complex queries.

Most search engines cover only a small portion of the Web. To increase the coverage of the Web by a single search system, multiple search engines can be combined. A search system that uses other search engines to perform the search and combines their search results is called a *metasearch engine*. Mamma.com and dogpile.com are metasearch engines that combine multiple document search engines whereas addall.com is a metasearch engine that combines multiple database search engines for books. From a user's perspective, there is little difference between using a search engine and using a metasearch engine.

This article provides an overview of some of the main methods that are used to create search engines and metasearch engines. In the next section, we describe the basic techniques for creating a document search engine. Then we sketch the idea of building a database search engine. Finally, we introduce the key components of metasearch engines, including both document metasearch engines and database metasearch engines, and the techniques for building them.

DOCUMENT SEARCH ENGINE

Architecture

Although the architectures of different Web search engines may vary, a typical document search engine generally consists of the following four main components as shown in Fig. 1: Web crawler, Indexer, Index database, and Query engine. A Web crawler, also known as a Web spider or a Web robot, traverses the Web to fetch Web pages by following the URLs of Web pages. The Indexer is responsible for parsing the text of each Web page into word tokens and then creating the Index database using all the fetched Web pages. When a user query is received, the Query engine searches the Index database to find the matching Web pages for the query.

Crawling the Web

A Web crawler is a computer program that fetches Web pages from remote Web servers. The URL of each Web page identifies the location of the page on the Web. Given its URL, a Web page can be downloaded from a Web server using the HTTP (HyperText Transfer Protocol). Starting from some initial URLs, a Web crawler repeatedly fetches Web pages based on their URLs and extracts new URLs from the downloaded pages so that more pages can be downloaded. This process ends when some termination conditions are satisfied. Some possible termination conditions include (1) no new URL remains and (2) a preset number of pages have been downloaded. As a Web crawler may interact with numerous autonomous Web servers, it is important to design scalable and efficient crawlers.

To crawl the Web quickly, multiple crawlers can be applied. These crawlers may operate in two different manners (i.e., centralized and distributed). Centralized crawlers are located at the same location running on different machines in parallel. Distributed crawlers are distributed at different locations of the Internet, and controlled by a central coordinator; each crawler just crawls the Web sites that are geographically close to the location of the crawler. The most significant benefit of distributed crawlers is the reduction in communication cost incurred by crawling activity. Centralized crawlers, however, are easier to implement and control than distributed crawlers.

As the Web grows and changes constantly, it is necessary to have the crawlers regularly re-crawl the Web and make the contents of the index database up to date. Frequent re-crawling of the Web will waste significant resources and make the network and Web servers over-

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

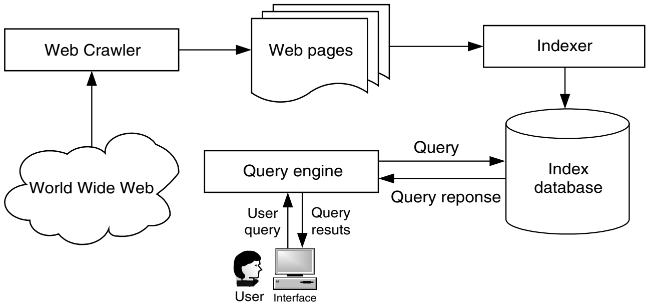


Figure 1. The general architecture of a document search engine.

loaded. Therefore, some incremental crawling strategies should be employed. One strategy is to re-crawl just the changed or newly added Web pages since the last crawling. The other strategy is to employ topic-specific crawlers to crawl the Web pages relevant to a pre-defined set of topics. Topic-specific crawling can also be used to build specialized search engines that are only interested in Web pages in some specific topics.

Conventional Web crawlers are capable of crawling only Web pages in the Surface Web. Deep Web crawlers are designed to crawl information in the Deep Web (3). As information in the Deep Web is often hidden behind the search interfaces of Deep Web data sources, Deep Web crawlers usually gather data by submitting queries to these search interfaces and collecting the returned results.

Indexing Web Pages

After Web pages are gathered to the site of a search engine, they are pre-processed into a format that is suitable for effective and efficient retrieval by search engines. The contents of a page may be represented by the words it has. Non-content words such as "the" and "is" are usually not used for page representation. Often, words are converted to their stems using a stemming program to facilitate the match of the different variations of the same word. For example, "comput" is the common stem of "compute" and "computing". After non-content word removal and stemming are performed on a page, the remaining words (called *terms* or *index terms*) are used to represent the page. Phrases may also be recognized as special terms. Furthermore, a weight is assigned to each term to reflect the importance of the term in representing the contents of the page.

The weight of a term t in a page p within a given set P of pages may be determined in a number of ways. If we treat each page as a plain text document, then the weight of t is usually computed based on two statistics. The first is its *term frequency* (*tf*) in p (i.e., the number of times t appears in p), and the second is its *document frequency* (*df*) in P (i.e., the number of pages in P that contain t). Intuitively, the more times a term appears in a page, the more important the term is in representing the contents of the page. Therefore, the weight of t in p should be a monotonically increasing function of its term frequency. On the other hand, the more pages that have a term, the less useful the term is in differentiating different pages. As a result, the weight of a term should be a monotonically decreasing function of its document frequency. Currently, most Web pages are formatted in HTML, which contains a set of tags such as *title* and *header*. The tag information can be used to influence the weights of the terms for representing Web pages. For example, terms in the title of a page or emphasized using bold and italic fonts are likely to be more important in representing a page than terms in the main body of the page with normal font.

To allow efficient search of Web pages for any given query, the representations of the fetched Web pages are organized into an *inverted file structure*. For each term t, an inverted list of the format $[(p_1, w_1), \ldots, (p_k, w_k)]$ is generated and stored, where each p_j is the identifier of a page containing t and w_j is the weight of t in p_j, $1 \leq j \leq k$. Only entries with positive weights are kept.

Ranking Pages for User Queries

A typical query submitted to a document search engine consists of some keywords. Such a query can also be represented as a set of terms with weights. The degree of match between a page and a query, often call the *similarity*, can be measured by the terms they share. A simple approach is to add up the products of the weights corresponding to the matching terms between the query and the page. This approach yields larger similarities for pages that share more important terms with a query. However, it tends to favor longer pages over shorter ones. This problem is often addressed by dividing the above similarity by the product of the lengths of the query and the page. The function that computes such type of similarities is called the *Cosine* function (4). The length of each page can be computed beforehand and stored at the search engine site.

Many methods exist for ranking Web pages for user queries, and different search engines likely employ different ranking techniques. For example, some ranking methods also consider the proximity of the query terms within a page. As another example, a search engine may keep track of the number of times each page has been accessed by users and use such information to help rank pages. Google (www.google.com) is one of the most popular search engines on the Web. A main reason why Google is successful is its powerful ranking method, which has the capability to differentiate more important pages from less important ones even when they all contain the query terms the same number of times. Google uses the linkage information among Web pages (i.e., how Web pages are linked) to derive the importance of each page. A link from page A to page B is placed by the author of page A. Intuitively, the existence of such a link is an indication that the author of page A considers page B to be of some value. On the Web, a page may be linked from many other pages and these links can be aggregated in some way to reflect the overall importance of the page. For a given page, *PageRank* is a measure of the relative importance of the page on the Web, and this measure is computed based on the linkage information (5). The following are the three main ideas behind the definition and computation of PageRank. (1) Pages that are linked from more pages are likely to be more important. In other words, the importance of a page should be reflected by the popularity of the page among the authors of all Web pages. (2) Pages that are linked from more important pages are likely to be more important themselves. (3) Pages that have links to more pages have less influence over the importance of each of the linked pages. In other words, if a page has more child pages, then it can only propagate a smaller fraction of its importance to each child page. Based on the above insights, the founders of Google developed a method to calculate the importance (PageRank) of each page on the Web (5). The PageRanks of Web pages can be combined with other, say content-based, measures to indicate the overall relevance of a page with respect to a given query. For example, for a given query, a page may be ranked based on a weighted sum of its similarity with the query and its PageRank. Among pages with similar similarities, this method will rank those that have higher PageRanks.

Effective and Efficient Retrieval

For a given query, a page is said to be *relevant* if the sender of the query finds the page useful. For a given query submitted by a user against a fixed set of pages, the set of relevant pages is also fixed. A good retrieval system should return a high percentage of relevant pages to the user and rank them high in the search result for each query. Traditionally, the effectiveness of a text retrieval system is measured using two quantities known as *recall* and *precision*. For a given query and a set of documents, recall is the percentage of the relevant documents that are retrieved and precision is the percentage of the retrieved documents that are relevant. To evaluate the effectiveness of a text retrieval system, a set of test queries is often used. For each query, the set of relevant documents is identified in advance. For each test query, a precision value at a different recall point is obtained. When the precision values at different recall values are averaged over all test queries, an average recall-precision curve is obtained, which is used as the measure of the effectiveness of the system. A system is considered to be more effective than another system if the recall-precision curve of the former is above that of the latter. A perfect text retrieval system should have both recall and precision equal to 1 at the same time. In other words, such a system retrieves exactly the set of relevant documents for each query. In practice, perfect performance is not achievable for many reasons, for example, a user's information needs usually cannot be precisely specified by the used query and the contents of documents and queries cannot be completely represented by weighted terms.

Using both recall and precision to measure the effectiveness of traditional text retrieval systems requires knowing all the relevant documents for each test query in advance. This requirement, however, is not practical for independently evaluating large search engines because it is impossible to know the number of relevant pages in a search engine for a query unless all the pages are retrieved and manually examined. Without knowing the number of relevant pages for each test query, the recall measure cannot be computed. As a result of this practical constraint, search engines are often evaluated using the average precision based on the top k retrieved pages for a set of test queries, for some small integer k, say 20, or based on the average position of the first relevant page among the returned results for each test query (6).

A large search engine may index hundreds of millions or even billions of pages, and process millions of queries on a daily basis. For example, by the end of 2005, the Google search engine has indexed about 10 billion pages and processed over 200 million queries every day. To accommodate the high computation demand, a large search engine often employs a large number of computers and efficient query processing techniques. When a user query is received by a search engine, the inverted file structure of the pre-processed pages, not the pages themselves, are used to find matching pages. Computing the similarity between a query and every page directly is very inefficient because the vast majority of the pages likely do not share any term with the query and computing the similarities of these pages with the query is a waste of resources. To process a query, a *hash table* is first used to locate the storage location of the inverted file list of each query term. Based on the inverted file lists of all the terms in the query, the similarities of all the pages that contain at least one term in common with the query can be computed efficiently.

Result Organization

Most search engines display search results in descending order of their matching scores with respect to a given query. Some search engines, such as the Vivisimo search engine (www.vivisimo.com), organize their results into groups such that pages that have certain common features are placed into the same group. Clustering/categorizing search results is known to be effective in helping users identify relevant results in two situations. One is when the number of results returned for a query is large, which is mostly true for large search engines, and the other is when a query submitted by a user is short, which is also mostly true as the average number of terms in a search engine query is slightly over two. When the number of results is large, clustering allows the searcher to focus the attention on a small number of promising groups. When a query is short, the query may be interpreted in different ways, in this case, clustering can group results based on different interpretations that allow the searcher to focus on the group with desired interpretation. For example, when query "apple" is submitted to the Vivisimo search engine, results related to Apple computer (Macintosh) forms one group and results related to fruit forms another group, which makes it easy for a user to focus on the results he/she wants.

Challenges of Document Search Engines

Although Web search engines like Google, Yahoo, and MSN are widely used by numerous users to find the desired information on the Web, there are still a number of challenges for enhancing their quality (7,8). In the following, we briefly introduce some of these challenges.

Freshness. Currently, most search engines depend on Web crawlers to collect Web pages from numerous Web sites and build the index database based on the fetched Web pages. To refresh the index database so as to provide up-to-date pages, they periodically (e.g., once every month) *recollect* Web pages from the Internet and *rebuild* the index database. As a result, pages that are added/deleted/changed since the last crawling are not reflected in the current index database, which makes some pages not accessible via the search engine, some retrieved pages not available on the Web (i.e., deadlinks), and the ranking of some pages based on obsolete contents. How to keep the index database up-to-date for large search engines is a challenging issue.

Coverage. It was estimated that no search engine indexes more than one-third of the "publicly indexable Web" (9). One important reason is that the Web crawlers can only crawl Web pages that are linked to the initial seed URLs. The "Bow Tie" theory about the Web structure (10) indicates that only 30% of the Web pages are strongly connected. This theory further proves the limitation of Web crawlers. How to fetch more Web pages, including those in the Deep Web, is a problem that needs further research.

Quality of Results. Quality of results refers to how well the returned pages match the given keywords query. Given a keywords query, a user wants the most relevant pages to be returned. Suppose a user submits "apple" as a query, a typical search engine will return all pages containing the word "apple" no matter if it is related to an apple pie recipe or Apple computer. Both the keywords-based similarity and the lack of context compromise the quality of returned pages. One promising technique for improving the quality of results is to perform a personalized search, in which a profile is maintained for each user that contains the user's personal information, such as specialty and interest, as well as some information obtained by tracking the user's Web surfing behaviors, such as which pages the user has clicked and how long the user spent on reading them; a user's query can be expanded based on his/her profile, and the pages are retrieved and ranked based on how well they match the expanded query.

Natural Language Query. Currently, most search engines accept only keywords queries. However, keywords cannot precisely express users' information needs. Natural language queries, such as "Who is the president of the United States?" often require clear answers that cannot be provided by most current search engines. Processing natural language queries requires not only the understanding of the semantics of a user query but also a different parsing and indexing mechanism of Web pages. Search engine ask.com can answer some simple natural language queries such as "Who is the president of the United States?" and "Where is Chicago?" using its *Web Answer* capability. However, ask.com does not yet have the capability to answer general natural language queries. There is still a long way to go before general natural language queries can be precisely answered.

Querying Non-Text Corpus. In addition to textual Web pages, a large amount of image, video, and audio data also exists on the Web. How to effectively and efficiently index and retrieve such data is also an open research problem in data search engines. Although some search engines such as Google and Yahoo can search images, their technologies are still mostly keywords-match based.

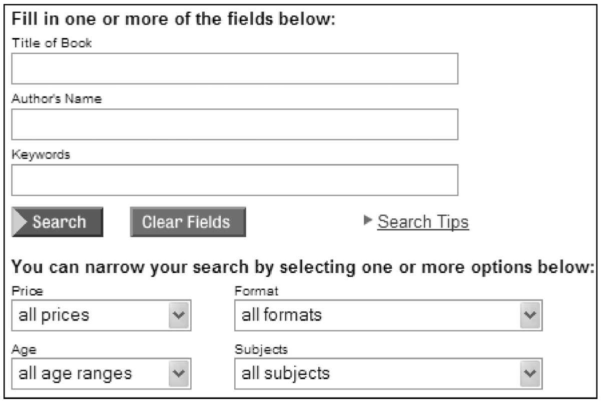
DATABASE SEARCH ENGINE

In comparison with document search engines, database search engines are much easier to build because they do not need crawlers to crawl the Web to build the index database. Instead, traditional database systems such as Oracle or SQL-server are usually used by database search engines to store and manage data. The stored data are often compiled and entered by human users. Unlike Web pages that have little structure, the data in database search engines are generally well structured. For example, the database of an online Web bookstore contains various books, and every book has attributes such as title, author, ISBN, publication date, and so on.

To make the data in a database search engine Web-accessible, an HTML form-based Web search interface like Fig. 2 is created on top of the underlying database system. The Web search interface often has multiple fields for users to specify queries that are more complex than the keywords queries for document Web search engines. For example, the search interface of bn.com (Fig. 2) contains fields like title, author, price, format, and so on. A user query submitted through the Web search interface of a database search engine is usually converted to a database query (e.g., SQL) that can be processed by the underlying database system; after the results that satisfy the query conditions are returned by the database system, they are wrapped by appropriate HTML tags and presented to the user on the dynamically generated Web page.

Database search engines are often used by organizations or companies that want to publish their compiled data on the Web for information sharing or business benefits. For example, a real estate company may employ a database search engine to post housing information, and an airline may use a database search engine to allow travelers to search and purchase airplane tickets.

It should be noted that structured data that are stored in database systems and are accessible via database search engines constitute a major portion of the Deep Web. A



Fill in one or more of the fields below:

Title of Book

Author's Name

Keywords

You can narrow your search by selecting one or more options below:

Price: all prices

Format: all formats

Age: all age ranges

Subjects: all subjects

Figure 2. The book search interface of bn.com.engine.

recent survey (2) estimated that, by April 2004, among the 450,000 search engines for the Deep Web, 348,000 were database search engines.

METASEARCH ENGINE

A metasearch engine is a system that provides unified access to multiple existing search engines. When a metasearch engine receives a query from a user, it sends the query to multiple existing search engines, and it then combines the results returned by these search engines and displays the combined results to the user. A metasearch engine makes it easy for a user to search multiple search engines simultaneously while submitting just one query. A big benefit of a metasearch engine is its ability to combine the coverage of many search engines. As metasearch engines interact with the search interfaces of search engines, they can use Deep Web search engines just as easily as Surface Web search engines. Therefore, metasearch engine technology provides an effective mechanism to reach a large portion of the Deep Web by connecting to many Deep Web search engines.

Metasearch Engine Architecture

A simple metasearch engine consists of a user interface for users to submit queries, a search engine connection component for programmatically submitting queries to its employed search engines and receiving result pages from them, a result extraction component for extracting the search result records from the returned result pages, and a result merging component for combining the results (11). If a metasearch engine employs a large number of search engines, then a search engine selection component is needed. This component determines which search engines are likely to contain good matching results for any given user query so that only these search engines are used for this query. Search engine selection is necessary for efficiency considerations. For example, suppose only the 20 best-matched results are needed for a query and there are 1000 search engines in a metasearch engine. It is clear that the 20 best-matched results will come from at most 20 search engines, meaning that at least 980 search engines are not useful for this query. Sending a query to useless search engines will cause serious inefficiencies, such as heavy network traffic caused by transmitting unwanted results and the waste of system resources for evaluating the query.

We may have metasearch engines for document search engines and metasearch engines for database search engines. These two types of metasearch engines, though conceptually similar, need different techniques to build. They will be discussed in the next two subsections.

Document Metasearch Engine

A document metasearch engine employs document search engines as its underlying search engines. In this subsection, we discuss some aspects of building a document metasearch engine, including search engine selection, search engine connection, result extraction, and merging.

Search Engine Selection. When a metasearch engine receives a query from a user, the metasearch engine makes a determination on which search engines likely contain useful pages to the query and therefore should be used to process the query. Before search engine selection can be performed, some information representing the contents of the set of pages of each search engine is collected. The information about the pages in a search engine is called the *representative* of the search engine (11). The representatives of all search engines used by the metasearch engine are collected in advance and are stored with the metasearch engine. During search engine selection for a given query, search engines are ranked based on how well their representatives match with the query.

Different search engine selection techniques exist and they often employ different types of search engine representatives. A simple representative of a search engine may contain only a few selected key words or a short description. This type of representative is usually produced manually by someone who is familiar with the contents of the search engine. When a user query is received, the metasearch engine can compute the similarities between the query and the representatives, and then select the search engines with the highest similarities. Although this method is easy to implement, this type of representative provides only a general description about the contents of search engines. As a result, the accuracy of the selection may be low.

More elaborate representatives collect detailed statistical information about the pages in each search engine. These representatives typically collect one or several pieces of statistical information for each term in each search engine. As it is impractical to find out all the terms that appear in some pages in a search engine, an approximate vocabulary of terms for a search engine can be used. Such an approximate vocabulary can be obtained from pages retrieved from the search engine using sample queries (12). Some of the statistics that have been used in proposed search engine selection techniques include, for each term, its *document frequency*, its average or maximum weight in all pages having the term, and the number of search engines that have the term. With the detailed statistics, more accurate estimation of the usefulness of each search engine with respect to any user query can be obtained. The collected statistics may be used to compute the similarity between a query and each search engine, to estimate the number of pages in a search engine whose similarities with the query are above a threshold value, and to estimate the similarity of the most similar page in a search engine with respect to a query (11). These quantities allow search engines to be ranked for any given query and the top-ranked search engines can then be selected to process the query.

It is also possible to generate search engine representatives by learning from the search results of past queries. In this case, the representative of a search engine is simply the knowledge indicating its past performance with respect to different queries. In the SavvySearch metasearch engine (13) (now www.search.com), the learning is carried out as follows. For a search engine, a weight is maintained for each term that has appeared in previous queries. The weight of a term for a search engine is increased or decreased depending on whether the search engine returns useful results for a query containing the term. Over time, if a search engine has a large positive (negative) weight for a term, the search engine is considered to have responded well (poorly) to the term in the past. When a new query is received by the metasearch engine, the weights of the query terms in the representatives of different search engines are aggregated to rank the search engines. The ProFusion metasearch engine also employs a learning-based approach to construct the search engine representatives (14). ProFusion uses training queries to find out how well each search engine responds to queries in 13 different subject categories. The knowledge learned about each search engine4 from training queries is used to select search engines to use for each user query and the knowledge is continuously updated based on the user's reaction to the search result (i.e., whether a particular page is clicked by the user).

Search Engine Connection. Usually, the search interface of a search engine is implemented using an HTML *form* tag with a query textbox. The form tag contains all information needed to connect to the search engine via a program. Such information includes the name and the location of the program (i.e., the search engine server) that processes user queries as well as the network connection method (i.e., the HTTP request method, usually GET or POST). The query textbox has an associated name and is used to fill out the query. The form tag of each search engine interface is pre-processed to extract the information needed for program connection. After a query is received by the metasearch engine and the decision is made to use a particular search engine, the query is assigned to the name of the query textbox of the search engine and sent to the server of the search engine using the HTTP request method supported by the search engine. After the query is processed by the search engine, a result page containing the search results is returned to the metasearch engine.

Search Result Extraction. A result page returned by a search engine is a dynamically generated HTML page. In addition to the search result records for a query, a result page usually also contains some unwanted information/ links such as advertisements, search engine host information, or sponsored links. It is essential for the metasearch engine to correctly extract the search result records on each result page. A typical search result record corresponds to a Web page found by the search engine and it usually contains the URL and the title of the page as well as some additional information about the page (usually the first few sentences of the page plus the date at which the page was created, etc.; it is often called the *snippet* of the page).

As different search engines organize their result pages differently, a separate result extraction program (also called *extraction wrapper*) needs to be generated for each search engine. To extract the search result records of a search engine, the structure/format of its result pages needs to be analyzed to identify the region(s) that contain the records and separators that separate different records (15). As a result, a wrapper is constructed to extract the results of any query for the search engine. Extraction wrappers can be manually, semi-automatically, or automatically constructed.

Result Merging. Result merging is the task of combining the results returned from multiple search engines into a single ranked list. Ideally, pages in the merged result should be ranked in descending order of the global matching scores of the pages, which can be accomplished by fetching/downloading all returned pages from their local servers and computing their global matching scores in the metasearch engine. For example, the Inquirus metasearch engine employs such an approach (7). The main drawback of this approach is that the time it takes to fetch the pages might be long.

Most metasearch engines use the local ranks of the returned pages and their snippets to perform result merging to avoid fetching the actual pages (16). When snippets are used to perform the merging, a matching score of each snippet with the query can be computed based on several factors such as the number of unique query terms that appear in the snippet and the proximity of the query terms in the snippet. Recall that when search engine selection is performed for a given query, the usefulness of each search engine is estimated and is represented as a score. The search engine scores can be used to adjust the matching scores of retrieved search records, for example, by multiplying the matching score of each record by the score of the search engine that retrieved the record. Furthermore, if the same result is retrieved by multiple search engines, the multiplied scores of the result from these search engines are aggregated, or added up, to produce the final score for the result. This type of aggregation gives preference to those results that are retrieved by multiple search engines. The search results are then ranked in descending order of the final scores.

Database Metasearch Engine

A database metasearch engine provides a unified access to multiple database search engines. Usually, multiple database search engines in the same application domain (e.g., auto, book, real estate, flight) are integrated to create a database metasearch engine. Such a metasearch engine over multiple e-commerce sites allows users to do comparison-shopping across these sites. For example, a metasearch engine on top of all book search engines allows users to find desired books with the lowest price from all booksellers.

A database metasearch engine is similar to a document metasearch engine in architecture. Components such as search engine connection, result extraction, and result merging are common in both types of metasearch engines, but the corresponding components for database metasearch engines need to deal with more structured data. For example, result extraction needs to extract not only the returned search records (say books) but also lower level semantic data units within each record such as the titles and prices of books. One new component needed for a database metasearch engine is the search interface integration component. This component integrates the search interfaces of multiple database search engines in the same domain into a unified interface, which is then used by users to specify queries against the metasearch engine. This component is not needed for document metasearch engines because document search engines usually have very simple search interfaces (just a textbox). In the following subsections, we present some details about the search interface integration component and the result extraction component. For the latter, we focus on extracting lower level semantic data units within records.

Search Interface Integration. To integrate the search interfaces of database search engines, the first step is to extract the search fields on the search interfaces from the HTML Web pages of these interfaces. A typical search interface of a database search engine has multiple search fields. An example of such an interface is shown in Fig. 2. Each search field is implemented by text (i.e., field label) and one or more HTML form control elements such as textbox, selection list, radio button, and checkbox. The text indicates the semantic meaning of its corresponding search field. A search interface can be treated as a partial schema of the underlying database, and each search field can be considered as an attribute of the schema. Search interfaces can be manually extracted but recently there have been efforts to develop techniques to automate the extraction (17). The main challenge of automatic extraction of search interfaces is to group form control elements and field labels into logical attributes.

After all the search interfaces under consideration have been extracted, they are integrated into a unified search interface to serve as the interface of the database metasearch engine. Search interface integration consists of primarily two steps. In the first step, attributes that have similar semantics across different search interfaces are identified. In the second step, attributes with similar semantics are mapped to a single attribute on the unified interface. In general, it is not difficult for an experienced human user to identify matching attributes across different search interfaces when the number of search interfaces under consideration is small. For applications that need to integrate a large number of search interfaces or need to perform the integration for many domains, automatic integration tools are needed. WISE-Integrator (18) is a tool that is specifically designed to automate the integration of search interfaces. It can identify matching attributes across different interfaces and produce a unified interface automatically.

Result Extraction and Annotation. For a document search engine, a search result record corresponds to a retrieved Web page. For a database search engine, however, a search result record corresponds to a structured entity in the database. The problem of extracting search result records from the result pages of both types of search engines is similar (see search result extraction section). However, a search result record of a database entity is more structured than that of a Web page, and it usually consists of multiple lower level semantic data units that need to be extracted and annotated with appropriate labels to facilitate further data manipulation such as result merging.

Wrapper induction in (19) is a semi-automatic technique to extract the desired information from Web pages. It needs users to specify what information they want to extract, and then the wrapper induction system induces the rules to construct the wrapper for extracting the corresponding data. Much human work is involved in such wrapper construction. Recently, research efforts (20,21) have been put on how to automatically construct wrapper to extract structured data. To automatically annotate the extracted data instances, currently there are three basic approaches: ontology-based (22), search interface schema-based, and physical layout-based. In the ontology-based approach, a task-specific ontology (i.e., conceptual model instance) is usually predefined, which describes the data of interest, including relationships, lexical appearance, and context keywords. A database schema and recognizers for constants and keywords can be produced by parsing the ontology. Then the data units can be recognized and structured using the recognizers and the database schema. The search interface schema-based approach (23) is based on the observation that the complex Web search interfaces of database search engines usually partially reflect the schema of the data in the underlying databases. So the data units in the returned result record may be the values of a search field on search interfaces. The search field labels are thereby assigned to corresponding data units as meaningful labels. The physical layout-based approach assumes that data units usually occur together with their class labels; thus it annotates the data units in such a way that the closest label to the data units is treated as the class label. The headers of a visual table layout are also another clue for annotating the corresponding column of data units. As none of the three approaches is perfect, a combination of them will be a very promising approach to automatic annotation.

Challenges of Metasearch Engines

Metasearch engine technology faces very different challenges compared with search engine technology, and some of these challenges are briefly described below.

Automatic Maintenance. Search engines used by metasearch engines may change their connection parameters and result display format, and these changes can cause the search engines not usable from the metasearch engines unless the corresponding connection programs and result extraction wrappers are changed accordingly. It is very expensive to manually monitor the changes of search engines and make the corresponding changes in the metasearch engine. Techniques that can automatically detect search engine changes and make the necessary adjustments in the corresponding metasearch engine components are needed.

Scalability. Most of today's metasearch engines are very small in terms of the number of search engines used. A typical metasearch engine has only about a dozen search engines. The current largest metasearch engine AllInOneNews (www.allinonenews.com) connects to about 1500 news search engines. But it is still very far away from building a metasearch engine that uses all of the over half million document search engines currently on the Web. There are number of challenges in building very large-scale metasearch engines, including automatic maintenance described above, and automatic generation and maintenance of search engine representatives that are needed to enable efficient and effective search engine selection.

Entity Identification. For database metasearch engines, data records retrieved from different search engines that correspond to the same real-world entities must be correctly identified before any meaningful result merging can be performed. Automatic entity identification across autonomous information sources has been a very challenging issue for a long time. A related problem is the automatic and accurate data unit annotation problem as knowing the meanings of different data units in a record can greatly help match the records. Although research on both entity identification and data unit annotation is being actively pursued, there is still long way to go to solve these problems satisfactorily.

ACKNOWLEDGMENT

This work is supported in part by the following NSF grants: IIS-0414981, IIS-0414939, and CNS-0454298. We also would like to thank the anonymous reviewers for valuable suggestions to improve the manuscript.

BIBLIOGRAPHY

1. M. Bergman. The Deep Web: Surfacing the Hidden Value. BrightPlanet White Paper. Available: http://www.brightplanet.com/images/stories/pdf/deepwebwhitepaper.pdf, October 16, 2006.
2. K. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the Web: observations and Implications. *SIGMOD Record,* **33** (3): 61–70, 2004.
3. S. Raghavan, and H. Garcia-Molina. Crawling the hidden Web. VLDB Conference, 2001.
4. G. Salton, and M. McGill. *Introduction to modern information retrieval.* New York: McCraw-Hill, 2001.
5. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bring Order to the Web. Technical Report, Stanford University, 1998.
6. D. Hawking, N. Craswell, P. Bailey, and K. Griffiths. Measuring search engine quality. *J. Inf. Retriev.,* **4** (1): 33–59, 2001.
7. S. Lawrence, and C. Lee Giles. Inquirus, the NECi Meta Search Engine. Seventh International World Wide Web conference, Brisbane, Australia, 1998, pp. 95–105.
8. M. R. Henzinger, and R. Motwani, and C. Silverstein. Challenges in Web Search Engines. Available http://citeseer.ist.psu.edu/henzinger02challenges.html, 2002.
9. S. Lawrence, and C. Lee Giles. Accessibility of information on the Web. *Nature,* **400**: 1999.
10. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph Structure in the Web. The 9th International World Wide Web Conference, Amsterdam, 2001.
11. W. Meng, C. Yu, and K. Liu. Building Efficient and Effective Metasearch Engines. *ACM Comput. Surv.,* **34** (1), 48–84, 2002.
12. J. Callan, M. Connell, and A. Du, Automatic Discovery of Language Models for Text Databases. ACM SIGMOD Conference, Philadelphia, PA, 1999, pp. 479–490.
13. D. Dreilinger, and A. Howe. Experiences with selecting search engines using metasearch. *ACM Trans. Inf. Syst.,* **15** (3): 195–222.
14. Y. Fan, and S. Gauch. Adaptive Agents for Information Gathering from Multiple, Distributed Information Sources. AAAI Symposium on Intelligent Agents in Cyberspace Stanford University, 1999, pp. 40–46.
15. H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. World Wide Web Conference, 2005.
16. Y. Lu, W. Meng, L. Shu, C. Yu, and K. Liu. Evaluation of result merging strategies for metasearch engines. *WISE Conference* , 2005.
17. Z. Zhang, B. He, and K. Chang. Understanding Web query interfaces: best-effort parsing with hidden syntax. ACM SIGMOD Conference, 2004.
18. H. He, W. Meng, C. Yu, and Z. Wu. Automatic integration of web search interfaces with wise-integrator. *VLDB J.* **13** (3): 256–273, 2004.
19. C. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the Web: a machine learning approach. *IEEE Data Eng. Bull.* **23** (4):2000.
20. A. Arasu, and H. Garcia-Molina. Extracting Structured Data from Web pages. SIGMOD Conference, 2003.
21. V. Crescenzi, G. Mecca, and P. Merialdo. RoadRUNNER: Towards Automatic Data Extraction from Large Web Sites. VLDB Conference, Italy, 2001.
22. D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y. K. Ng, R. D. Smith, Conceptual-model-based data extraction from multiple-record Web pages. *Data Knowledge Eng.,* **31** *(3):* 227–251, 1999.
23. J. Wang, and F. H. Lochovsky. Data Extraction and Label Assignment for Web Databases. WWW Conference, 2003.

FURTHER READING

S. Kirsch. The future of internet search: infoseek's experiences searching the Internet. *ACM SIGIR Forum* , **32** (2): 3–7, 1998.

Weiyi Meng
Hai He
State University of New York
Binghamton, New York

DATA SECURITY

INTRODUCTION

The term *data security* refers to the protection of information against possible violations that can compromise its *secrecy* (or confidentiality), *integrity,* or *availability.* Secrecy is compromised if information is disclosed to users not authorized to access it. Integrity is compromised if information is improperly modified, deleted, or tampered. Availability is compromised if users are prevented from accessing data for which they have the necessary permissions. This last problem is also known as *denial-of-service.*

The problem of protecting information has existed since information has been managed. However, as technology advances and information management systems become more and more powerful, the problem of enforcing information security also becomes more critical. The increasing development of information technology in the past few years, which has led to the widespread use of computer systems to store and manipulate information and greatly increased the availability and the processing and storage power of information systems, has also posed new serious security threats and increased the potential damage that violations may cause. Organizations more than ever today depend on the information they manage. A violation to the security of the information may jeopardize the whole system working and cause serious damages. Hospitals, banks, public administrations, and private organizations, all of them depend on the accuracy, availability, and confidentiality of the information they manage. Just imagine what could happen, for instance, if a patient's data were improperly modified, were not available to the doctors because of a violation blocking access to the resources, or were disclosed to the public domain.

Many are the threats to security to which information is exposed. Threats can be *nonfraudulent* or *fraudulent.* The first category comprises all threats resulting in nonintentional violations, such as natural disasters, errors or bugs in hardware or software, and human errors. The second category comprises all threats causing intentional violations. Such threats can be represented by authorized users *(insiders),* who can misuse their privileges and authority, or external users *(intruders),* who can improperly get access to the system and its resources. Ensuring protection against these threats requires the application of different protection measures. This article focuses, in particular, on the protection of information against possible violations by users, insiders, or intruders. The following services are crucial to the protection of data within this context (1):

1. *Identification and Authentication.* It provides the system with the ability of identifying its users and confirming their identity.
2. *Access Control.* It evaluates access requests to the resources by the authenticated users, and based on some access rules, it determines whether they must be granted or denied.
3. *Audit.* It provides a post facto evaluation of the requests and the accesses occurred to determine whether violations have occurred or have been attempted.
4. *Encryption.* It ensures that any data stored in the system or is sent over the network can be deciphered only by the intended recipient. In network communication, it can also be used to ensure the authenticity of the information transmitted and of the parties involved in the communication.

Figure 1 illustrates the position of these services within the system working. Their treatment is the focus of this article.

IDENTIFICATION AND AUTHENTICATION

Authentication is the process of certifying the identity of a party to another. In the most basic form, authentication certifies the identity of a human user to the computer system. Authentication is a prerequisite for a correct access control, because the correctness of the access control relies on the correctness of the identity of the subject requesting access. Good authentication is also important for *accountability,* whereby users can be retained accountable for the actions accomplished when connected to the system. In the authentication process, we can generally distinguish an *identification* phase, where users declare their identity to the computer and submit a proof for it; and an actual *authentication* phase, where the declared identity and the submitted proof are evaluated. The most common ways to enforce user to computer authentication are based on the use of:

- *Something the user knows,* such as a password.
- *Something the user possesses,* such as a magnetic card.
- *Something the user is or does,* such as her physical characteristics.

These techniques can be used in alternative or in combination, thus providing a stronger protection. For instance, a smartcard may require that a password be entered to unlock it.

Authentication Based on Knowledge

The most common technique based on user's knowledge uses secret keywords, named *passwords.* A password, known only to the user and the system, proves the identity of the user to the system. Users wishing to log into the computer enter their identity *(login)* and submit a secret keyword *(password)* as proof of their identity. Passwords

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

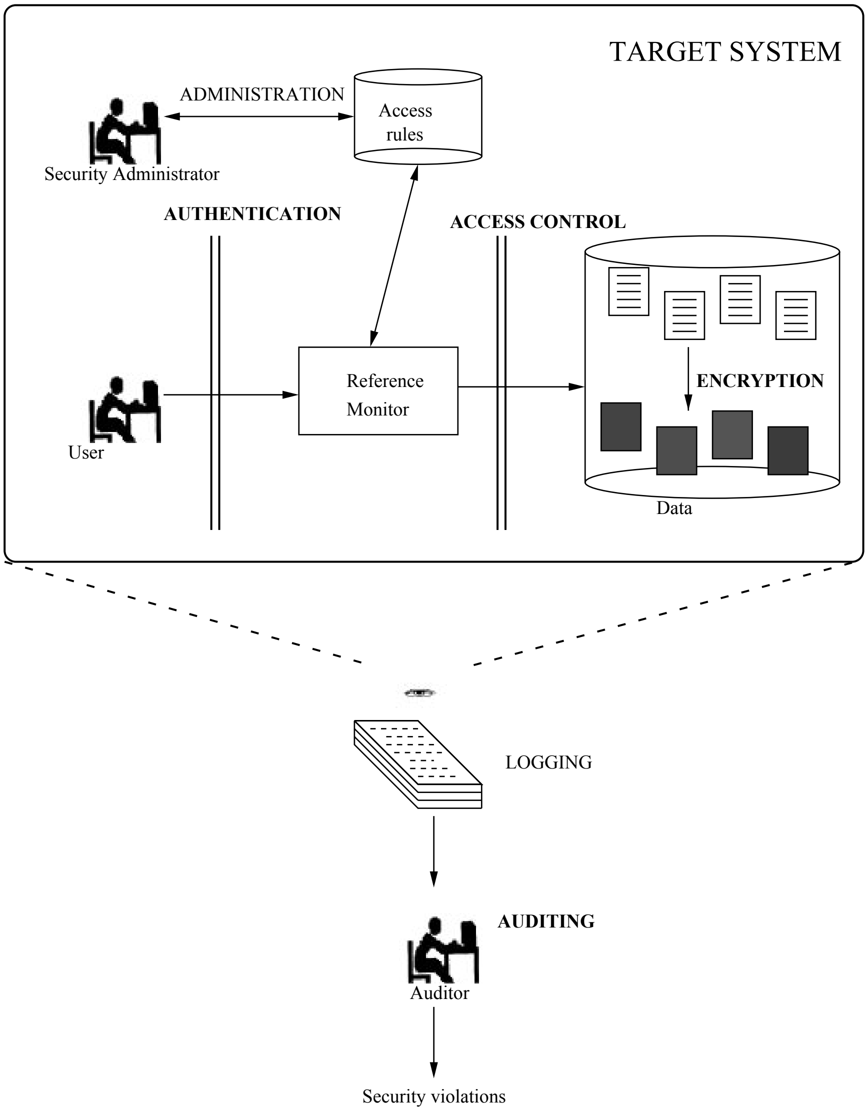


Figure 1. Authentication, access control, audit, and encryption.

are the most commonly used authentication techniques for controlling access to computers. The wide use of this technique is because it is very simple, cheap, and easily enforceable. As a drawback, however, this technique is vulnerable. Passwords can often be easily *guessed, snooped* by people observing the legitimate user keying it in, *sniffed* during transmission, or *spoofed* by attackers impersonating login interfaces. By getting a user's password, an attacker can then "impersonate" this user and enter the system. An important aspect necessary to limit the vulnerability of passwords is a good password management. A good password management requires users to change their password regularly, choose passwords that are not easy to guess, and keep the password private. Unfortunately these practices are not always followed. Having to remember passwords can become a burden for a user, especially when multiple passwords, necessary to access different accounts, need to be remembered. To avoid this problem, many systems enforce automatic controls regulating the specification and use of passwords. For instance, it is possible to enforce restrictions on the minimum number of digits a password must have, possibly requiring the use of both alphanumeric and non-alphanumeric characters. Also, often systems check passwords against language dictionaries and reject passwords corresponding to words of the language (which would be easily retrieved by attackers enforcing dictionary attacks). It is also possible to associate a maximum lifetime with passwords and require users to change their password when it expires. Passwords that remain unchanged for a long time are more vulnerable, and, if guessed and never changed, would allow attackers to freely access the system impersonating the legitimate users. A history log can also be kept to make sure users do not just pretend to change the password while reusing instead the same one. Sometimes a minimum lifetime can also be associated with passwords. The reason is for users to avoid reusing the same password over and over again despite the presence of lifetime and history controls. Without a minimum lifetime, a user required to change password but unwilling to do so could simply change it and then change it back right away to the old value. A minimum lifetime restriction would forbid this kind of operation.

Authentication Based on Possession

In this category, also called *token-based*, all techniques exist that require users to present a *token* as a proof of their identity. A token is a credit-size card device storing some information establishing and proving the token's identity. The simplest form of token is a memory card containing magnetically recorded information, which can be read by an appropriate card reader. Essentially, this technique authenticates the validity of the token, not of the user: Possession of the token establishes identity for the user. The main weakness of such an approach is that tokens can be forged, lost, or stolen. To limit the risk of security breaches due to such occurrences, often memory cards are used together with a *personal identification number (PIN),* generally composed of four numeric digits, that works like a password. To enter the system, a user needs both to present the token and to enter the PIN. Like passwords, PIN can be guessed or spoofed, thus possibly compromising authentication, because an attacker possessing the token and knowing the PIN will be able to impersonate the legitimate user and enter the system. To limit the vulnerability from attackers possessing a token and trying to guess the corresponding PIN to enter the system, often the authentication server terminates the authentication process, and possibly seizes the card, upon submission of few bad tries for a PIN. Like passwords, tokens can be shared among users, thus compromising accountability. Unlike with passwords, however, because possession of the token is necessary to enter the system, only one user at a time is able to enter the system. Memory cards are very simple and do not have any processing power. They cannot therefore perform any check on the PIN or encrypt it for transmission. This requires sending the PIN to the authentication server in the clear, exposing the PIN to sniffing attacks and requiring trust in the authentication server. The ATM (Automatic Teller Machine) cards are provided with processing power, which allows the checking and encrypting of the PIN before its transmission to the authentication server.

In token devices provided with processing capabilities, authentication is generally based on a challenge-response handshake. The authentication server generates a challenge that is keyed into the token by the user. The token computes a response by applying a cryptographic algorithm to the secret key, the PIN, and the challenge, and returns it to the user, who enters this response into the workstation interfacing the authentication server. In some cases, the workstation can directly interface the token, thus eliminating the need for the user to type in the challenge and the response. Smartcards are sophisticated token devices that have both processing power and direct connection to the system. Each smartcard has a unique private key stored within it. To authenticate the user to the system, the smartcard verifies the PIN. It then enciphers the user's identifier, the PIN, and additional information like date and time, and it sends the resulting ciphertext to the authentication server. Authentication succeeds if the authentication server can decipher the message properly.

Authentication Based on Personal Characteristics

Authentication techniques in this category establish the identity of users on the basis of their biometric characteristics. Biometric techniques can use physical or behavioral characteristics, or a combination of them. Physical characteristics are, for example, the retina, the fingerprint, and the palmprint. Behavioral characteristics include handwriting, voiceprint, and keystroke dynamics (2). Biometric techniques require a first phase in which the characteristic is measured. This phase, also called *enrollment,* generally comprises several measurements of the characteristic. On the basis of the different measurements, a template is computed and stored at the authentication server. A User's identity is established by comparing their characteristics with the stored templates. It is important to note that, unlike passwords, biometric methods are not exact. A password entered by a user either matches the one stored at the authentication server or it does not. A biometric characteristic instead cannot be required to exactly match the stored template. The authentication result is therefore based on how closely the characteristic matches the stored template. The acceptable difference must be determined in such a way that the method provides a high rate of successes (i.e., it correctly authenticates legitimate users and rejects attackers) and a low rate of unsuccesses. Unsuccesses can either deny access to legitimate users or allow accesses that should be rejected. Biometric techniques, being based on personal characteristics of the users, do not suffer of the weaknesses discusses above for password or token-based authentication. However, they require high-level and expensive technology, and they may be less accurate. Moreover, techniques based on physical characteristics are often not well accepted by users because of their intrusive nature. For instance, retinal scanners, which are one of the most accurate biometric methods of authentication, have raised concerns about possible harms that the infrared beams sent to the eye by the scanner can cause. Measurements of other characteristics, such as fingerprint or keystroke dynamics, have instead raised concerns about the privacy of the users.

ACCESS CONTROL

Access control evaluates the requests to access resources and services and determines whether to grant or deny them. In discussing access control, it is generally useful to distinguish between *policies* and *mechanisms*. Policies are high-level guidelines that determine how accesses are controlled and access decisions are determined. Mechanisms are low-level software and hardware functions implementing the policies. There are several advantages in abstracting policies from their implementation. First, it is possible to compare different policies and evaluate their properties without worrying about how they are actually implemented. Second, it is possible to devise mechanisms that enforce different policies so that a change of policy does not necessarily require changing the whole implementation. Third, it is possible to devise mechanisms that can enforce multiple policies at the same time, thus allowing users to choose the policy that best suits their needs when stating protection requirements on their data (3–6). The definition and formalization of a set of policies specifying the working of the access control system, providing thus an abstraction of the control mechanism, is called a *model.*

Access control policies can be divided into three major categories: *discretionary access control* (DAC), *mandatory access control* (MAC), and the most recent *role-based access control* (RBAC).

Discretionary Policies

Discretionary access control policies govern the access of users to the system on the basis of the user's identity and of rules, called *authorizations,* that specify for each user (or group of users) the types of accesses the user can/cannot exercise on each object. The objects to which access can be requested, and on which authorizations can be specified, may depend on the specific data model considered and on the desired granularity of access control. For instance, in operating systems, objects can be files, directories, or programs. In relational databases, objects can be databases, relations, views, and, possibly tuples or attributes within a relation. In object-oriented databases, objects include classes, instances, and methods. Accesses executable on the objects, or on which authorizations can be specified, may correspond to primitive operations like read, write, and execute, or to higher level operations or applications. For instance, in a bank organization, operations like, debit, credit, inquiry, an extinguish can be defined on objects of type accounts.

Policies in this class are called discretionary because they allow users to specify authorizations. Hence, the accesses to be or not to be allowed are at the discretion of the users. An authorization in its basic form is a triple ⟨`user`, `object`, `mode`⟩ stating that the `user` can exercise the access `mode` on the `object`. Authorizations of this form represent permission of accesses. Each request is controlled against the authorizations and allowed only if a triple authorizing it exists. This kind of policy is also called *closed* policy, because only accesses for which an explicit authorization is given are allowed, whereas the default decision is to deny access. In an *open* policy, instead, (negative) authorizations specify the accesses that should not be allowed. All access requests for which no negative authorizations are specified are allowed by default. Most systems support the closed policy. The open policy can be applied in systems with limited protection requirements, where most accesses are to be allowed and the specification of negative authorizations results is therefore more convenient.

Specification of authorizations for each single user, each single access mode, and each single object can become an administrative burden. By grouping users, modes, and objects, it is possible to specify authorizations holding for a group of users, a collection of access modes, and/or a set of objects (4,7–9). This grouping can be user defined or derived from the data definition or organization. For instance, object grouping can be based on the type of objects (e.g., files, directories, executable programs); on the application/activity in which they are used (e.g., ps-files, tex-files, dvi-files, ascii); on data model concepts (e.g., in object-oriented systems, a group can be defined corresponding to a class and grouping all its instances); or on other classifications defined by users. Groups of users generally reflect the structure of the organization. For instance, examples of groups can be `employee`, `staff`, `researchers`, or `consultants`. Most models considering user groups allow groups to be nested and nondisjoint. This means that users can belong to different groups and groups themselves can be members of other groups, provided that there are no cycles in the membership relation (i.e., a group cannot be a member of itself). Moreover, a basic group, called `public`, generally collects all users of the system.

The most recent authorization models supports grouping of users and objects, and both positive and negative authorizations (4,6,8,10–14). These features, toward the development of mechanisms able to enforce different policies, allow the support of both the closed and the open policy within the same system. Moreover, they represent a convenient means to support exceptions to authorizations. For instance, it is possible to specify that a group of users, with the exception of one of its members, can execute a particular access by granting a positive authorization for the access to the group and a negative authorization for the same access to the user. As a drawback for this added expressiveness and flexibility, support of both positive and negative authorizations complicates authorization management. In particular, conflicts may arise. To illustrate, consider the case of a user belonging to two groups. One group has a positive authorization for an access; the other has a negative authorization for the same access. Conflict control policies should then be devised that determine whether the access should in this case be allowed or denied. Different solutions can be taken. For instance, deciding on the *safest* side, the negative authorizations can be considered to hold *(denials take precedence).* Alternatively, conflicts may be resolved on the basis of possible relationships between the involved groups. For instance, if one of the groups is a member of the other one, then the authorization specified for the first group may be considered to hold *(most specific authorization takes precedence).* Another possible solution consists in assigning explicit priorities to authorizations; in the case of conflicts, the authorization with greater priority is considered to hold.

DAC Mechanisms. A common way to think of authorizations at a conceptual level is by means of an *access matrix.* Each row corresponds to a user (or group), and each column corresponds to an object. The entry crossing a user with an object reports the access modes that the user can exercise on the object. Figure 2(a) reports an example of an access matrix. Although the matrix represents a good conceptualization of authorizations, it is not appropriate for implementation. The access matrix may be very large and sparse. Storing authorizations as an access matrix may therefore prove inefficient. Three possible approaches can be used to represent the matrix:

- *Access Control List (ACL).* The matrix is stored by column. Each object is associated with a list, indicating, for each user, the access modes the user can exercise on the object.

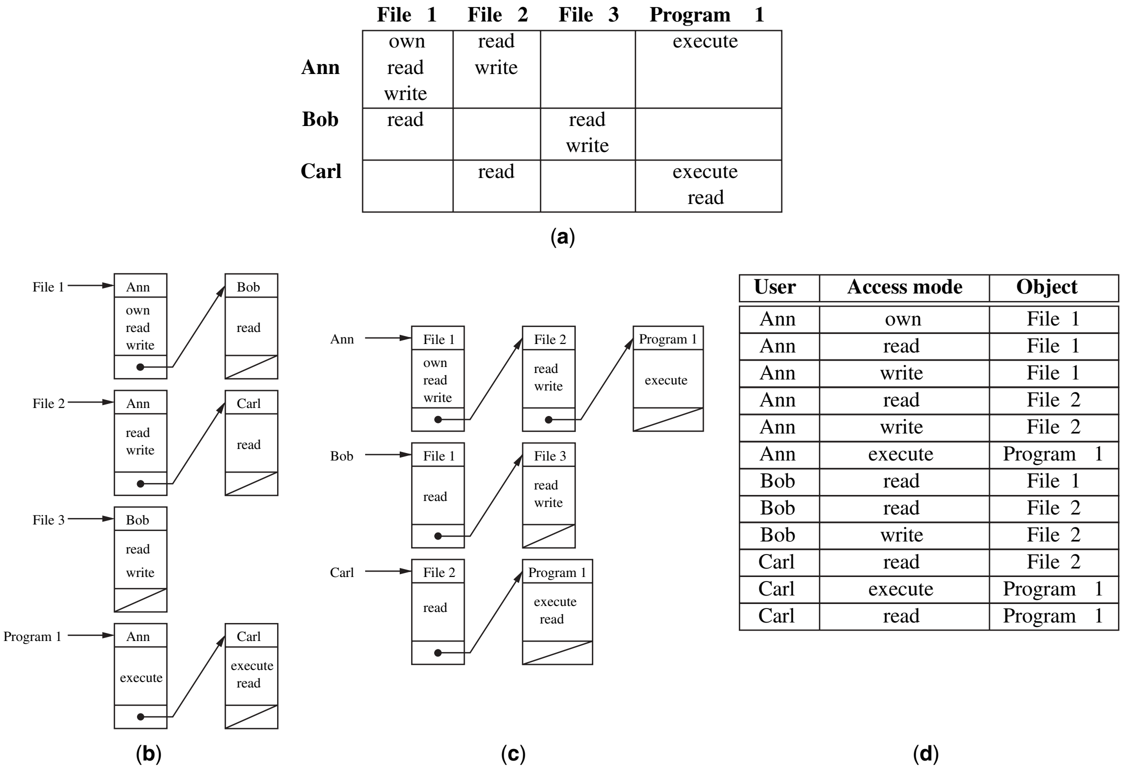
- *Capability*. The matrix is stored by row. Each user has associated a list, called a capability list, indicating for each object in the system the accesses the user is allowed to exercise on the object.
- *Authorization Table*. Nonempty entries of the matrix are reported in a three-column table whose columns are users, objects, and access modes, respectively. Each tuple in the table corresponds to an authorization.

Figure 2(b)–(d) illustrates the ACLs, capabilities, and authorization table, respectively, corresponding to the access matrix in Fig. 2(a).

Capabilities and ACLs present advantages and disadvantages with respect to authorization control and management. In particular, with ACLs, it is immediate to check the authorizations holding on an object, whereas retrieving all authorizations of a user requires the examination of the ACLs for all objects. Analogously, with capabilities, it is immediate to determine the privileges of a user, whereas retrieving all accesses executable on an object requires the examination of all different capabilities. These aspects affect the efficiency of authorization revocation upon deletion of either users or objects.

In a system supporting capabilities, it is sufficient for a user to present the appropriate capability to gain access to an object. This represents an advantage in distributed systems because it avoids multiple authentication of a subject. A user can be authenticated at a host, acquire the appropriate capabilities and present them to obtain accesses at the various servers of the system. Capabilities suffers, however, from a serious weakness. Unlike tickets, capabilities can be copied. This exposes capabilities to the risk of forgery, whereby an attacker gains access to the system by copying capabilities. For these reasons, capabilities are not generally used. Most commercial systems use ACLs. The Linux file system uses a primitive form of authorizations and ACLs. Each user in the system belongs to exactly one group, and each file has an owner (generally the user who created it). Authorizations for each file can be specified for either the owner, the group to which she belongs, or "the rest of the world." Each file has associated a list of nine privileges: *read, write*, and *execute,* each defined three times, at the level of *user, group,* and *other.* Each privilege is characterized by a single letter: `r` for *read;*`w` for *write;*`x` for *execute;* the absence of the privilege is represented by character -. The string presents first the *user* privileges, then *group,* and finally *other.* For instance, the ACL `rwxr-x--x` associated with a file indicates that the file can be read, written, and executed by its owner; read and executed by the group to which the owner belongs; and executed by all other users.

Weakness of Discretionary Policies: The Trojan Horse Problem. In discussing discretionary policies, we have referred to users and to access requests on objects submitted by users. Although it is true that each request is originated because of some user's actions, a more precise examination of the access control problem shows the utility



	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

(a)

(b)

(c)

User	Access mode	Object
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 2
Bob	write	File 2
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

(d)

Figure 2. (a) An example of access matrix, (b) corresponding ACL, (c) capabilities, and (d) and authorization table.

of separating *users* from *subjects*. Users are passive entities for whom authorizations can be specified and who can connect to the system. Once connected to the system, users originate processes (subjects) that execute on their behalf and, accordingly, submit requests to the system. Discretionary policies ignore this distinction and evaluate all requests submitted by a process running on behalf of some user against the authorizations of the user. This aspect makes discretionary policies vulnerable from processes executing malicious programs exploiting the authorizations of the user on behalf of whom they are executing. In particular, the access control system can be bypassed by Trojan Horses embedded in programs. A Trojan Horse is a computer program with an apparently or actually useful function, which contains additional *hidden* functions that surreptitiously exploit the legitimate authorizations of the invoking process. A Trojan Horse can improperly use any authorization of the invoking user; for example, it could even delete all files of the user (this destructive behavior is not uncommon in the case of viruses). This vulnerability to Trojan Horses, together with the fact discretionary policies do not enforce any control on the flow of information once this information is acquired by a process, makes it possible for processes to leak information to users not allowed to read it. This can happen without the cognizance of the data administrator/owner, and despite the fact that each single access request is controlled against the authorizations. To understand how a Trojan Horse can leak information to unauthorized users despite the discretionary access control, consider the following example. Assume that within an organization, Vicky, a top-level manager, creates a file Market containing important information about releases of new products. This information is very sensitive for the organization and, according to the organization's policy, should not be disclosed to anybody besides Vicky. Consider now John, one of Vicky's subordinates, who wants to acquire this sensitive information to sell it to a competitor organization. To achieve this, John creates a file, let's call it Stolen, and gives Vicky the authorization to write the file. Note that Vicky may not even know about the existence of Stolen or about the fact that she has the write authorization on it. Moreover, John modifies an application generally used by Vicky, to include two hidden operations, a read operation on file Market and a write operation on file Stolen [Fig. 3(a)]. Then, he gives the new application to his manager. Suppose now that Vicky executes the application. As the application executes on behalf of Vicky, every access is checked against Vicky's authorizations, and the read and write operations above will be allowed. As a result, during execution, sensitive information in Market is transferred to Stolen and thus made readable to the dishonest employee John, who can then sell it to the competitor [Fig. 3(b)].

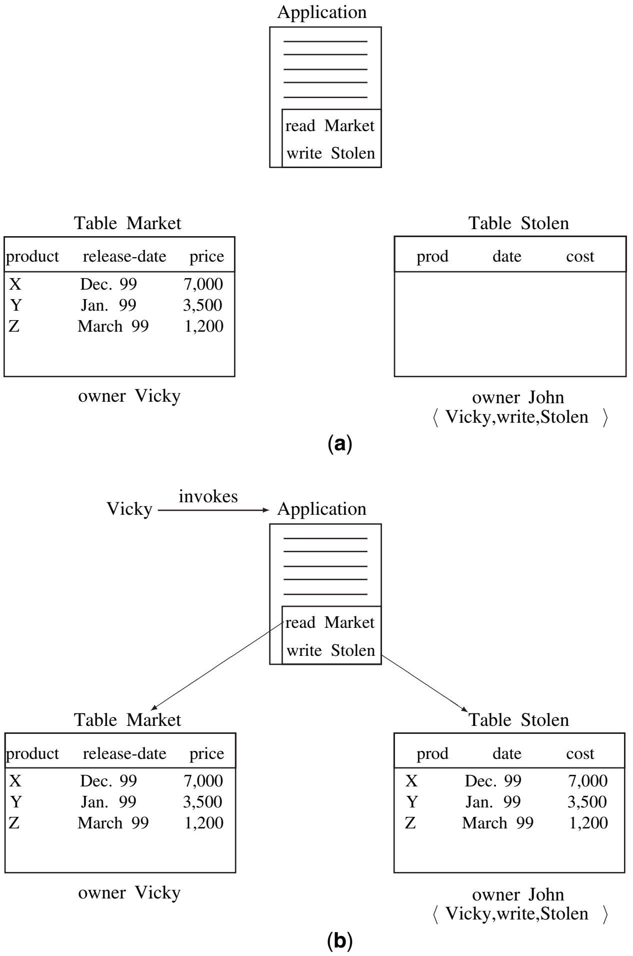


Figure 3. An example of a Trojan Horse.

The reader may object that there is little point in defending against Trojan Horses leaking information flow: Such an information flow could have happened anyway, by having Vicky explicitly tell this information to John, possibly even off-line, without the use of the computer system. Here is where the distinction between users and subjects operating on their behalf comes in. Although users are trusted to obey the access restrictions, subjects operating on their behalf are not. With reference to our example, Vicky is trusted not to release the sensitive information she knows to John, because, according to the authorizations, John cannot read it. However, the processes operating on behalf of Vicky cannot be given the same trust. Processes run programs that, unless properly certified, cannot be trusted for the operations they execute, as illustrated by the example above. For this reason, restrictions should be enforced on the operations that processes themselves can execute. In particular, protection against Trojan Horses leaking information to unauthorized users requires controlling the flows of information within process execution and possibly restricting them (13–18). Mandatory policies provide a way to enforce information flow control through the use of labels.

Mandatory Policies

Mandatory security policies enforce access control on the basis of classifications of *subjects* and *objects* in the system. Objects are the passive entities storing information, such as files, records, records' fields, in operating systems; or databases, tables, attributes, and tuples in relational database systems. Subjects are active entities that request access to the objects. An *access class* is defined as consisting of two components: a *security level* and a *set of categories*. The

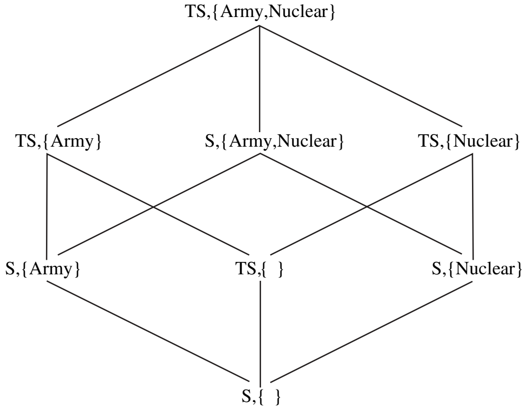


Figure 4. An example of a classification lattice.

security level is an element of a hierarchically ordered set. The levels generally considered are Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U), where TS > S > C > U. The set of categories is a subset of an unordered set, whose elements reflect functional or competence areas (e.g., NATO, Nuclear, Army for military systems; Financial, Administration, Research, for commercial systems). Access classes are partially ordered as follows: an access class c_1 *dominates* ($\geq$) an access class c_2 iff the security level of c_1 is greater than or equal to that of c_2 and the categories of c_1 include those of c_2. Two classes c_1 and c_2 are said to be incomparable if neither $c_1 \geq c_2$ nor $c_2 \geq c_1$ holds. Access classes together with the dominance relationship between them form a lattice. Figure 4 illustrates the security lattice for the security levels TS and S and the categories Nuclear and Army. Each object and each user in the system is assigned an access class. The security level of the access class associated with an object reflects the sensitivity of the information contained in the object, that is, the potential damage that could result from the unauthorized disclosure of the information. The security level of the access class associated with a user, also called *clearance,* reflects the user's trustworthiness not to disclose sensitive information to users not cleared to see it. Categories are used to provide finer grained security classifications of subjects and objects than classifications provided by security levels alone, and they are the basis for enforcing *need-to-know* restrictions. Users can connect to their system at any access class dominated by their clearance. A user connecting to the system at a given access class originates a subject at that access class. For instance, a user cleared `(Secret,Ø)` can connect to the system as a `(Secret,Ø)`, `(Confidential,Ø)`, or `(Unclassified,Ø)` subject. Requests by a subject to access an object are controlled with respect to the access class of the subject and the object and granted only if some relationship, depending on the requested access, is satisfied. In particular, two principles, first formulated by Bell and LaPadula (19), must be satisfied to protect information confidentiality:

No Read Up. A subject is allowed a read access to an object only if the access class of the subject dominates the access class of the object.

No Write Down. A subject is allowed a write access to an object only if the access class of the subject is dominated by the access of the object. (In most applications, subjects are further restricted to write only at their own level, so that no overwriting of sensitive information by low subjects not even allowed to see it can take place.)

Satisfaction of these two principles prevents information to flow from high-level subjects/objects to subjects/objects at lower levels, thereby ensuring the satisfaction of the protection requirements (i.e., no process will be able to make sensitive information available to users not cleared for it). This is illustrated in Fig. 5. Note the importance of controlling both read and write operations, because both can be improperly used to leak information. Consider the example of the Trojan Horse in Fig. 3. Possible classifications reflecting the specified access restrictions could be Secret for Vicky and Market, and Unclassified for John and Stolen. In the respect of the no-read-up and no-write-down principles, the Trojan Horse will never be able to complete successfully. If Vicky connects to the system as a Secret (or Confidential) subject, and thus the application runs with a Secret (or Confidential) access class, the write operation would be blocked. If Vicky invokes the application as an Unclassified subject, the read operation will be blocked instead.

Given the no-write-down principle, it is clear now why users are allowed to connect to the system at different access classes, so that they are able to access information at different levels (provided that they are cleared for it). For instance, Vicky has to connect to the system at a level below

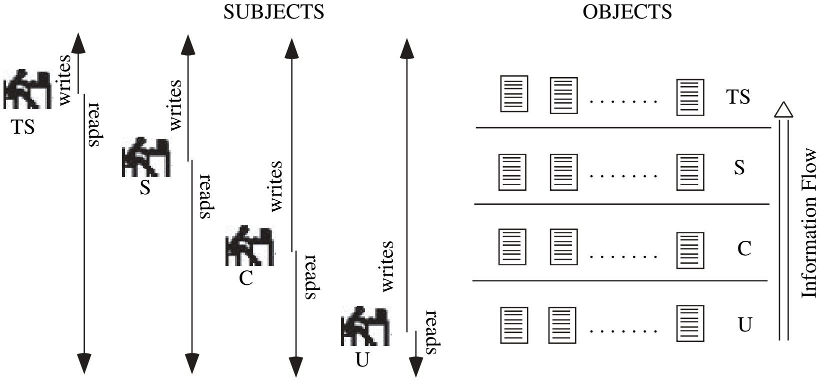


Figure 5. Controlling information flow for secrecy.

her clearance if she wants to write some Unclassified information, such as working instructions for John. Note also that a lower class does not mean “less” privileges in absolute terms, but only less reading privileges, as it is clear from the example above.

The mandatory policy that we have discussed above protects the confidentiality of the information. An analogous policy can be applied for the protection of the integrity of the information, to avoid untrusted subjects from modifying information they cannot write and compromise its integrity. With reference to our organization example, for example, integrity could be compromised if the Trojan Horse implanted by John in the application would write data in the file Market. Access classes for integrity comprise an integrity level and a set of categories. The set of categories is as seen for secrecy. The integrity level associated with a user reflects the user’s trustworthiness for inserting, modifying, or deleting information. The integrity level associated with an object reflects both the degree of trust that can be placed on the information stored in the object and the potential damage that could result from an unauthorized modification of the information. An example of integrity levels includes Crucial (C), Important (I), and Unknown (U). Access control is enforced according to the following two principles:

No Read Down. A subject is allowed a read access to an object only if the access class of the subject is dominated by the access class of the object.

No Write Up. A subject is allowed a write access to an object only if the access class of the subject is dominates the access class of the object.

Satisfaction of these principles safeguards integrity by preventing information stored in low objects (and therefore less reliable) to flow to high objects. This is illustrated in Fig. 6.

As it is visible from Figs. 5 and 6, secrecy policies allow the flow of information only from lower to higher (security) levels, whereas integrity policies allow the flow of information only from higher to lower security levels. If both secrecy and integrity have to be controlled, objects and subjects have to be assigned two access classes, one for secrecy control and one for integrity control.

The main drawback of mandatory protection policies is the rigidity of the control. They require the definition and application of classifications to subjects and objects. This may not always be feasible. Moreover, accesses to be allowed are determined only on the basis of the classifications of subjects and objects in the system. No possibility is given to the users for granting and revoking authorizations to other users. Some approaches have been proposed that complement flow control with discretionary access control (13,15,17).

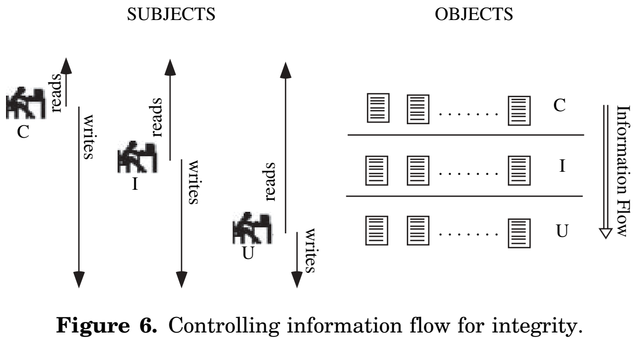


Figure 6. Controlling information flow for integrity.

Role-based Policies

A class of access control policies that has been receiving considerable attention recently is represented by role-based policies (20–22). Role-based policies govern the access of users to the information on the basis of their organizational role. A role can be defined as a set of actions and responsibilities associated with a particular working activity. Intuitively, a role identifies a task, and corresponding privileges, that users need to execute to perform organizational activities. Example of roles can be `secretary`, `dept-chair`, `programmer`, `payroll-officer`, and so on. Authorizations to access objects are not specified directly for users to access objects: Users are given authorizations to activate roles, and roles are given authorizations to access objects. By activating a given role (set of roles), a user will be able to execute the accesses for which the role is (set of roles are) authorized. Like groups, roles can also be organized in a hierarchy, along which authorizations can be propagated.

Note the different semantics that groups, presented in Section 3.1, and roles carry. Roles can be “activated” and “deactivated” by users at their discretion, whereas group membership always applies; that is, users cannot enable and disable group memberships (and corresponding authorizations) at their will. Note, however, that a same “concept” can be seen both as a group and as a role. To understand the difference between groups and roles, consider the following example. We could define a group, called `G_programmer`, consisting all users who are programmers. Any authorization specified for `G_programmer` is propagated to its members. Thus, if an authorization to read `tech-reports` is given to `G_programmer`, its members can exercise this right. We could also define a role, called `R_programmer`, and associate with it those privileges that are related to the programming activity and necessary for the programmers to perform their jobs (such as compiling, debugging, and writing reports). These privileges can be exercised by authorized users only when they choose to assume the role `R_programmer`. It is important to note that roles and groups are two complementary concepts; they are not mutually exclusive.

The enforcement of role-based policies present several advantages. Authorization management results are simplified by the separation of the users’ identity from the authorizations they need to execute tasks. Several users can be given the same set of authorizations simply by assigning them the same role. Also, if a user’s responsibilities change (e.g., because of a promotion), it is sufficient to disable her for the previous roles and enable her for a new set of roles, instead of deleting and inserting the many access authorizations that this responsibility change implies. A major advantage of role-based policies is represented by the fact that authorizations of a role are enabled only when the role is active for a user. This allows the enforcement of the least privilege principle, whereby a process is given only the authorizations it needs to complete successfully. This confinement of the process in a defined workspace is an important defense against attacks aiming at exploiting authorizations (as the Trojan Horse example illustrated). Moreover, the definition of roles and related authorizations fits with the information system organization and allows us to support related constraints, such as, for example, separation of duties (23–25). Separation of duties requires that no user should be given enough privileges to be able to misuse the system. For instance, the person authorizing a paycheck should not be the same person who prepares it. Separation of duties can be enforced statically, by controlling the specification of roles associated with each user and authorizations associated with each role, or dynamically, by controlling the actions actually executed by users when playing particular roles (4,24).

AUDIT

Authentication and access control do not guarantee complete security. Indeed, unauthorized or improper uses of the system can still occur. The reasons for this are various. First, security mechanisms, like any other software or hardware mechanism, can suffer from flaws that make them vulnerable to attacks. Second, security mechanisms have a cost, both monetary and in loss of system's performances. The more protection to reduce accidental and deliberate violations is implemented, the higher the cost of the system will be. For this reason, often organizations prefer to adopt less secure mechanisms, which have little impact on the system's performance, with respect to more reliable mechanisms, that would introduce overhead processing. Third, authorized users may misuse their privileges. This last aspect is definitely not the least, as misuse of privileges by internal users is one major cause of security violations.

This scenario raises the need for *audit control*. Audit provides a post facto evaluation of the requests and the accesses occurred to determine whether violations have occurred or have been attempted. To detect possible violations, all user requests and activities are registered in an *audit trail* (or *log),* for their later examination. An audit trail is a set of records of *computer events,* where a computer event is any action that happens on a computer system (e.g., logging into a system, executing a program, and opening a file). A computer system may have more than one audit trail, each devoted to a particular type of activity. The kind and format of data stored in an audit trail may vary from system to system; however, the information that should be recorded for each event includes the subject making the request, the object to which access is requested, the operation requested, the time and location at which the operation was requested, the response of the access control, and the amount of resources used. An audit trail is generated by an *auditing system* that monitors system activities. Audit trails have many uses in computer security:

- *Individual Accountability* An individual's actions are tracked in an audit trail making users personally accountable for their actions. Accountability may have a deterrent effect, as users are less likely to behave improperly if they know that their activities are being monitored.
- *Reconstructing Events* Audit trails can be used to reconstruct events after a problem has occurred. The amount of damage that occurred with an incident can be assessed by reviewing audit trails of system activity to pinpoint how, when, and why the incident occurred.
- *Monitoring* Audit trails may also be used as on-line tools to help monitoring problems as they occur. Such real-time monitoring helps in detecting problems like disk failures, over utilization of system resources, or network outages.
- *Intrusion Detection* Audit trails can be used to identify attempts to penetrate a system and gain unauthorized access.

It is easy to see that auditing is not a simple task, also due to the huge amount of data to be examined and to the fact that it is not always clear how violations are reflected in the users' or system's behaviors. Recent research has focused on the development of automated tools to help audit controls. In particular, a class of automated tools is represented by the so-called *intrusion detection systems,* whose purpose is to automate the data acquisition and their analysis. The issues to be addressed in data acquisition and analysis are as folows:

- *Audit data retention:* If the audit control is based on history information, then audit records al ready examined must be maintained. However, to avoid the "history log" to grow indefinitely, pruning operations should be executed removing records that do not need to be considered further.
- *Audit level:* Different approaches can be taken with respect to the level of events to be recorded. For instance, events can be recorded at the command level, at the level of each system call, at the application level, and so on. Each approach has some advantages and disadvantages, represented by the violations that can be detected and by the complexity and volume of audit records that have to be stored, respectively.
- *Recording time:* Different approaches can be taken with respect to the time at which the audit records are to be recorded. For instance, accesses can be recorded at the time they are requested or at the time they are completed. The first approach provides a quick response to possible violations, and the second provides more complete information for analysis.
- *Events monitored:* Audit analysis can be performed on any event or on specific events such as the events regarding a particular subject, object, operation, or occurring at a particular time or in a particular situation.

- *Audit control execution time:* Different approaches can be taken with respect to the time at which the audit control should be executed.
- *Audit control mechanism location:* The intrusion detection system and the monitored system may reside on the same machine or on different machines. Placing the audit control mechanism on different machines has advantages both in terms of performances (audit control does not interfere with normal system operation) and security, as the audit mechanism will not be affected by violations to the system under control.

DATA ENCRYPTION

Another measure for protecting information is provided by cryptography. Cryptographic techniques allow users to store, or transmit, encoded information instead of the actual data. An *encryption* process transforms the *plaintext* to be protected into an encoded *ciphertext,* which can then be stored or transmitted. A *decryption* process allows us to retrieve the *plaintext* from the *ciphertext.* The encryption and decryption functions take a *key* as a parameter. A user getting access to data, or sniffing them from the network, but lacking the appropriate decryption key, will not be able to understand them. Also, tampering with data results is difficult without the appropriate encryption key.

Cryptographic techniques must be proved resistant to attacks by cryptoanalysts trying to break the system to recover the plaintext or the key, or to forge data (generally, messages transmitted over the network). Cryptanalysis attacks can be classified according to how much information the cryptanalyst has available. In particular, with respect to secrecy, attacks can be classified as *ciphertext only, known-plaintext,* and *chosen-plaintext.* In ciphertext-only attacks, the cryptanalyst only knows the ciphertext, although she may know the encryption algorithm, the plaintext language, and possibly some words that are probably used in the plaintext. In known-plaintext attacks, the cryptanalyst also knows some plaintext and corresponding ciphertext. In chosen-plaintext attacks, the cryptanalyst can acquire the ciphertext corresponding to a selected plaintext. Most cryptographic techniques are designed to withstand chosen-plaintext attacks. The robustness of cryptographic algorithms relies on the amount of work and time that would be necessary to a cryptanalyst to break the system, using the best available techniques. With respect to protecting the authenticity of the information, there are two main classes of attacks: *impersonation attack,* in which the cryptanalyst creates a fraudulent ciphertext without knowledge of the authentic cipher-text; and *substitution attacks*, in which the cryptanalyst intercepts the authentic ciphertext and improperly modifies it.

Encryption algorithms can be divided into two main classes: *symmetric,* or *secret key,* and *asymmetric,* or *public key.* Symmetric algorithms encrypt and decrypt text by using the same key. Public key algorithms use, instead, two different keys. A *public* key is used to encrypt, and a *private* key, which cannot be guessed by knowing the public key, is used to decrypt. This is illustrated in Fig. 7. Symmetric algorithms rely on the secrecy of the key. Public key algorithms rely on the secrecy of the private key.

Symmetric Key Encryption

Most symmetric key encryption algorithms are *block ciphers*; that is, they break up the plaintext into blocks of a fixed length and encrypt one block at a time. Two well-known classes of block ciphers are based on substitution techniques and transposition techniques.

Substitution Algorithms. Substitution algorithms define a mapping, based on the key, between characters in the plaintext and characters in the ciphertext. Some substitution techniques are as follows.

Simple Substitution. Simple substitution algorithms are based on a one-to-one mapping between the plaintext alphabet and the ciphertext alphabet. Each character in the plaintext alphabet is therefore replaced with a fixed substitute in the ciphertext alphabet. An example of simple substitution is represented by the algorithms based on *shifted* alphabets, in which each letter of the plaintext is mapped onto the letter at a given fixed distance from it in the alphabet (wrapping the last letter with the first). An example of such algorithms is the Caesar cipher in which each letter is mapped to the letter three positions after it in the alphabet. Thus, A is mapped to D, B to E, and Z to C. For instance, `thistext` would be encrypted as `wklvwhaw`. Simple substitution techniques can be broken by analyzing single-letter frequency distribution (26).

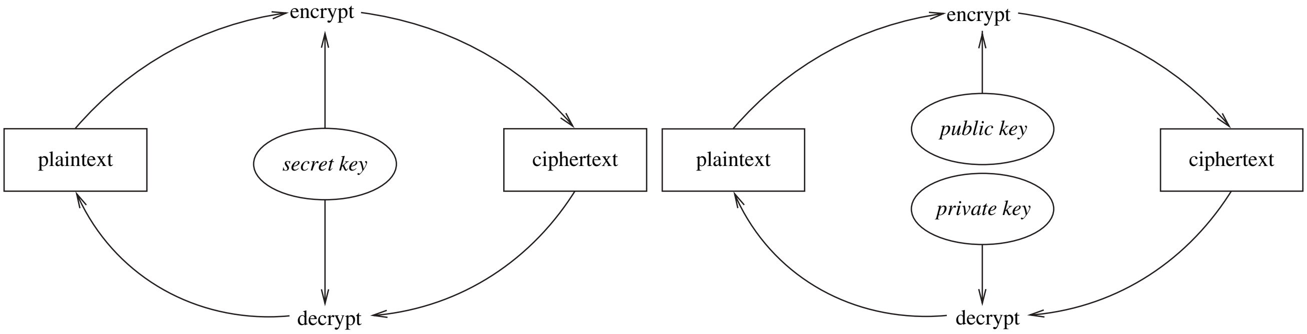


Figure 7. Secret key versus public key cryptography.

Homophonic Substitution. Homophonic substitution algorithms map each character of the plaintext alphabet onto a set of characters, called its *homophones,* in the ciphertext alphabet. There is therefore a one-to-many mapping between a plaintext character and the corresponding characters in the ciphertext. (Obviously, the vice versa is not true as decrypting cannot be ambiguous.) In this way, different occurrences of the same character in the plaintext may be mapped to different characters in the ciphertext. This characteristic allows the flattening of the letter frequency distribution in the ciphertext and proves a defense against attacks exploiting it. A simple example of homophonic substitution (although not used for ciphering) can be seen in the use of characters for phone numbers. Here, the alphabet of the plaintext are numbers, the alphabet of the ciphertext are the letters of the alphabet, but Q and Z which are not used, plus numbers 0 and 1 (which are not mapped to any letter). Number 2 maps to the first three letters of the alphabet, number 3 to the second three letters, and so on. For instance, number 6974663 can be enciphered as myphone, where the three occurrences of character 6 have been mapped to three different letters.

Polyalphabetic Substitution. Polyalphabetic substitution algorithms overcome the weakness of simple substitution through the use of multiple substitution algorithms. Most polyalphabetic algorithms use periodic sequences of alphabets. For instance, the Vigenère cipher uses a word as a key. The position in the alphabet of the *i*th character of the key gives the number of right shifts to be enforced on each *i*th element (modulo the key length) of the plaintext. For instance, if key crypt is used, then the first, sixth, eleventh, . . . , character of the plaintext will be shifted by 3 (the position of c in the alphabet); the second, seventh, twelfth, . . . , character will be shifted by 17 (the position of **r** in the alphabet); and so on.

Polygram Substitution. Although the previous algorithms encrypt a letter at the time, polygram algorithms encrypt blocks of letters. The plaintext is divided into blocks of letters. The mapping of each character of a block depends on the other characters appearing in the block. As an example, the Playfair cipher uses as key a 5 × 5 matrix, where the 25 letters of the alphabet (J was not considered) are inserted in some order. The plaintext is divided into blocks of length two. Each pair of characters is mapped onto a pair of characters in the ciphertext, where the mapping depends on the position of the two plaintext characters in the matrix (e.g., whether they are in the same column and/ or row). Polygram substitution destroys single-letter frequency distribution, thus making cryptanalysis harder.

Transposition Algorithms. Transposition algorithms determine the ciphertext by permuting the plaintext characters according to some scheme. The ciphertext, therefore, contains exactly the same characters as the plaintext but in different order. Often, the permutation scheme is determined by writing the plaintext in some geometric figure and then reading it by traversing the figure in a specified order. Some transposition algorithms, based on the use of matrixes, are as follows.

Column Transposition. The plaintext is written in a matrix by rows and re-read by columns according to an order specified by the key. Often the key is a word: The number of characters in the key determines the number of columns, and the position of the characters considered in alphabetical order determines the order to be considered in the reading process. For instance, key crypt would imply the use of a five-column matrix, where the order of the columns to be read is 14253 (the position in the key of the key characters is considered in alphabetical order). Let's say now that the plaintext to be encrypted is acme has discovered a new fuel. The matrix will look like as follows:

c	r	y	p	t
1	3	5	2	4
a	c	m	e	h
a	s	d	i	s
c	o	v	e	r
e	d	a	n	e
w	f	u	e	l

The corresponding ciphertext is aaceweienecsodfhsrelmdvau.

Periodic Transposition. It is a variation of the previous technique, where the text is also read by rows (instead of by columns) according to a specified column order. More precisely, instead of indicating the columns to be read, the key indicates the order in which the characters in each row must be read, and the matrix is read row by row. For instance, by using key crypt, the ciphertext is obtained by reading the first, fourth, second, fifth, and third character of the first row, then the second row is read in the same order, the third row, and so on. This process is equivalent to breaking the text in blocks with the same length as the key, and to permuting the characters in each block according to the order specified by the key. In this case, the ciphertext corresponding to the plaintext acme has discovered a new fuel is aechmaissdceorvendeaweflu.

Pure transposition and substitution techniques prove very vulnerable. For instance, transposition algorithms can be broken through anagramming techniques, because the characters in the ciphered text correspond exactly to the characters in the plaintext. Also, the fact that a transposition method has been used to encrypt can be determined by the fact that the ciphertext respects the frequency letter distribution of the considered alphabet. Simple substitution algorithms are vulnerable from attacks exploiting single-letter frequency distribution. Among them, shifted alphabet ciphers are easier to break, given that the mapping function applies the same transformation to all characters. Stronger algorithms can be obtained by combining the two techniques (27).

Advanced Encryption Standard (AES). In November 2001, the National Institute of Standards and Technology announced the approval of a new secret key cipher standard chosen among 15 candidates. This new standard algorithm was meant to replace the old DES algorithm (28), whose key

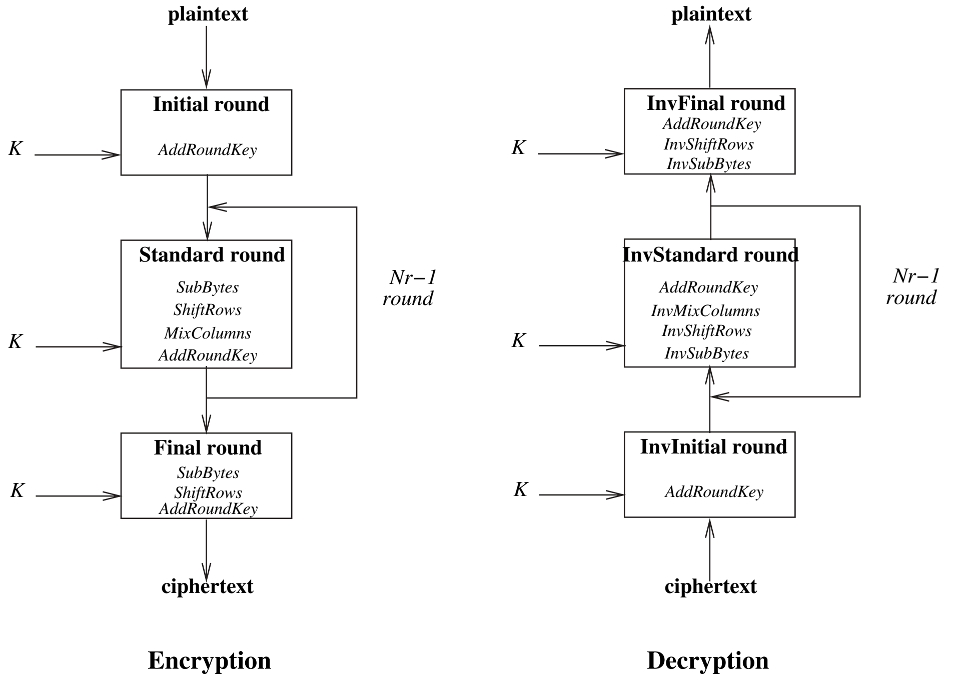


Figure 8. Encryption and decryption phases of the AES algorithm.

sizes were becoming too small. Rijndael, a compressed name taken from its inventors Rijmen and Daemen, was chosen to become the future Advanced Encryption Standard (AES) (29). AES is a block cipher that can process 128-bit block units using cipher keys with lengths of 128, 192, and 256 bits. Strictly speaking, AES is not precisely Rijndael (although in practice they are used interchangeably) as Rijndael supports a larger range of block and key sizes. Figure 8 illustrates the AES operational mode. It starts with an initial round followed by a number of standard rounds and ends with a final round. The number of standard round is dependent on the key length as illustrated in Table 1. AES operates on a 4 × 4 array of bytes (the intermediate cipher result), called the *state*.

AddRoundKey. The *AddRoundKey* operation is a simple EXOR operation between the *state* and the *roundkey*. The *roundkey* is derived from the secret key by means of a key schedule. The number of *roundkey* necessary to encrypt one block of the plaintext depends on the key length as this determines the number of rounds. For instance, for a key length of 128 bits, 11 *roundkeys* are needed. The keys are generated recursively. More precisely, key k_i of the ith round is obtained from the key expansion routine using subkey k_{i-1} of round $i - 1$-th and the secret key.

Table 1. Number of rounds

	Key length (words)	Number of rounds (Nr)
AES-128	4	10
AES-192	6	12
AES-256	8	14

[1]A word is 32 bits.

SubBytes. In the *SubBytes* operation, each byte in the state is updated using an 8-bit S-box. This operation provides the nonlinearity in the cipher. The AES S-box is specified as a matrix $M16 \times 16$. The first digit d_1 of the hexadecimal number corresponding to a state byte is used as a row index and the second digit d_2 as a column index. Hence, cell $M[d_1, d_2]$ contains the new hexadecimal number.

Shift Rows. In the *ShiftRows* operation, the rows of *state* are cyclically shifted with different offsets. The first row is shifted over 1 byte, the second row over 2 bytes, and the third row over 3 bytes.

MixColumns. The *MixColumns* operation operates on the columns of the state, combining the four bytes in each column using a linear transformation. Together with *ShiftRows, MixColumns* provides diffusion[1] in the cipher.

Note that in the AES algorithm encipherment and decipherment consists of different operations. Each operation that is used for encryption must be inverted to make it possible to decrypt a message (see Fig. 8).

Asymmetric Key Encryption

Asymmetric key encryption algorithms use two different keys for encryption and decryption. They are based on the application of one-way functions. A one-way function is a

[1] Diffusion refers to the property that redundancy in the statistics of the plaintext is "dissipated" in the statistics of the ciphertext (27). In a cipher with good diffusion, flipping an input bit should change each output bit with a probability of one half.

function that satisfies the property that it is computationally infeasible to compute the input from the result. Asymmetric key encryption algorithms are therefore based on hard-to-solve mathematical problems, such as computing logarithms, as in the proposals by Diffie and Hellman (30), (the proponents of public key cryptography) and by ElGamal (31), or factoring, as in the RSA algorithm illustrated next.

The RSA Algorithm. The most well-known public key algorithm is the RSA algorithm, whose name is derived from the initials of its inventor (Rivest, Shamir, and Adleman) (32). It is based on the idea that it is easy to multiply two large prime numbers, but it is extremely difficult to factor a large number. The establishment of the pair of keys works as follows. The user wishing to establish a pair of keys chooses two large primes p and q (which are to remain secret) and computes $n = pq$ and $\phi(n) = (p-1)(q-1)$, where $\phi(n)$ is the number of elements between 0 and $n-1$ that are relatively prime to n. Then, the user chooses an integer e between 1 and $\phi(n)-1$ that is relatively prime to $\phi(n)$, and computes its inverse d such that $ed \equiv 1 \bmod \phi(n)$. d can be easily computed by knowing $\phi(n)$. The encryption function E raises the plaintext M to the power e, modulo n. The decryption function D raises the ciphertext C to the power d, modulo n. That is, $E(M) = M^e \bmod n$ and $D(C) = C^d \bmod n$. Here, the public key is represented by the pair (e, n) and the private key by d. Because $\phi(n)$ cannot be determined without knowing the prime factors p and q, it is possible to keep d secret even if e and n are made public. The security of the algorithm depends therefore on the difficulty of factoring n into p and q. Usually a key with n of 512 bits is used, whose factorization would take a half-million MIPS-years with the best techniques known today. The algorithm itself, however, does not constraint the key length. The key length is variable. A longer key provides more protection, whereas a shorter key proves more efficient. The authors of the algorithm suggested using a 100-digit number for p and q, which would imply a 200-digit number for n. In this scenario, factoring n would take several billion years. The block size is also variable, but it must be smaller than the length of the key. The ciphertext block is the same length as the key.

Application of Cryptography

Cryptographic techniques can be used to protect the secrecy of information stored in the system, so that it will not be understandable to possible intruders bypassing access controls. For instance, password files are generally encrypted. Cryptography proves particularly useful in the protection of information transmitted over a communication network (33). Information transmitted over a network is vulnerable from *passive* attacks, in which intruders sniff the information, thus compromising its secrecy, and from *active* attacks, in which intruders improperly modify the information, thus compromising its integrity. Protecting against passive attacks means safeguarding the confidentiality of the message being transmitted. Protecting against active attacks requires us to be able to ensure the *authenticity of the message, its sender, and its receiver.* Authentication of the receiver means that the sender must be able to verify that the message is received by the recipient for which it was intended. Authentication of the sender means that the recipient of a message must be able to verify the identity of the sender. Authentication of the message means that sender and recipient must be able to verify that the message has not been improperly modified during transmission.

Both secret and public key techniques can be used to provide protection against both passive and active attacks. The use of secret keys in the communication requires the sender and the receiver to share the secret key. The sender encrypts the information to be transmitted by using the secret key and then sends it. Upon reception, the receiver decrypts the information with the same key and recovers the plaintext. Secret key techniques can be used if there is confidence in the fact that the key is only known by the sender and recipient and no disputes can arise (e.g., a dispute can arise if the sender of a message denies having ever sent it). Public keys, like secret keys, can provide authenticity of the sender, the recipient, and the message as follows. Each user establishes a pair of keys: The private key is known only to her, and the public key can be known to everybody. A user wishing to send a message to another user encrypts the message by using the public key of the receiver and then sends it. Upon reception, the receiver decrypts the message with his/her private key. Public keys can also be used to provide *nonrepudiation,* meaning the sender of a message cannot deny having sent it. The use of public keys to provide nonrepudiation is based on the concept of *digital signatures,* which, like handwritten signatures, provide a way for a sender to sign the information being transmitted. Digital signatures are essentially encoded information, a function of the message and the key, which are appended to a message. Digital signatures can be enforced through public key technology by having the sender of a message encrypting the message with her private key before transmission. The recipient will retrieve the message by decrypting it with the public key of the sender. Nonrepudiation is provided because only the sender knows her public key and therefore only the sender could have produced the message in question. In the application of secret keys, instead, the sender can claim that the message was forged by the recipient herself, who also knows the key. The two uses of public keys can be combined, thus providing sender, message, and recipient authentication, together with nonrepudiation.

Public key algorithms can do everything that secret key algorithms can do. However, all known public key algorithms are orders of magnitude slower than secret key algorithms. For this reason, often public key techniques are used for things that secret key techniques cannot do. In particular, they may be used at the beginning of a communication for authentication and to establish a secret key with which to encrypt information to be transmitted.

AUTHENTICATION AND ACCESS CONTROL IN OPEN SYSTEMS

Throughout this chapter, we illustrated the services crucial to the protection of data. In this section, we present ongoing work addressing authentication and access control in emerging applications and new scenarios.

Authentication in Open Environments

Today, accessing information on the global Internet has become an essential requirement of the modern economy. Recently, the focus has shifted from access to traditional information stored in WWW sites to access to large *e-services* (34). *Global e-services* are also coming of age, designed as custom applications exploiting single e-services already available on the Internet and integrating the results. In such a context, users normally have to sign-on to several distributed systems where e-services were built up in a way that they act independently as isolated security domains. Therefore, each user who wants to enter one of these domains has to authenticate herself at each of these security domains by the use of separate credentials (see Fig. 9). However, to realize the cooperation of services located in different domains, there is a need to have some authentication mechanism that does not require the user to authenticate and enter her user credentials several times. On corporate networks as well as on the global Net, access to services by *single-sign-on authentication* is now becoming a widespread way to authenticate users. Single-sign-on authentication relying on *digital certificates* (35) is currently the most popular choice for e-business applications. The widespread adoption of advanced *Public-Key Infrastructure* (PKI) technology and PKI-based solutions has provided secure techniques for outsourcing digital certificate management. Basically, any certificate-based authentication language needs a *subject description*, including a set of attributes. Several standardization efforts are aimed at exploiting such descriptions; for example, the Liberty Alliance (www.projectliberty.org) consortium is promoting a set of open standards for developing federated identity management infrastructures. However, the most successful standard is SAML (36), an authentication protocol handling authentication information across transactions between parties. Such an authentication information, which is wrapped into a SAML document, is called an *assertion* and it belongs to an entity (i.e., a human person, a computer, or any other subject as well). According to the specifications, an assertion may carry information about:

- Authentication acts performed by the subject.
- Attributes of the subject.
- Authorization decisions relating to the subject.

SAML assertions are usually transferred from *identity providers* (i.e., authorities that provide the user's identity to the affiliated services) to *service providers* (i.e., entities that simply offer services) and consist of one or more *statements*. Three kinds of statements are allowed: authentication statements, attribute statements, and authorization decision statements. Authentication statements assert to the service provider that the principal did indeed authenticate with the identity provider at a particular time using a particular method of authentication. Other information about the principal may be disclosed in an authentication statement. For instance, in the authentication statement in Fig. 10, the e-mail address of the principal is disclosed to the service provider. Note that the three statement types are not mutually exclusive. For instance, both

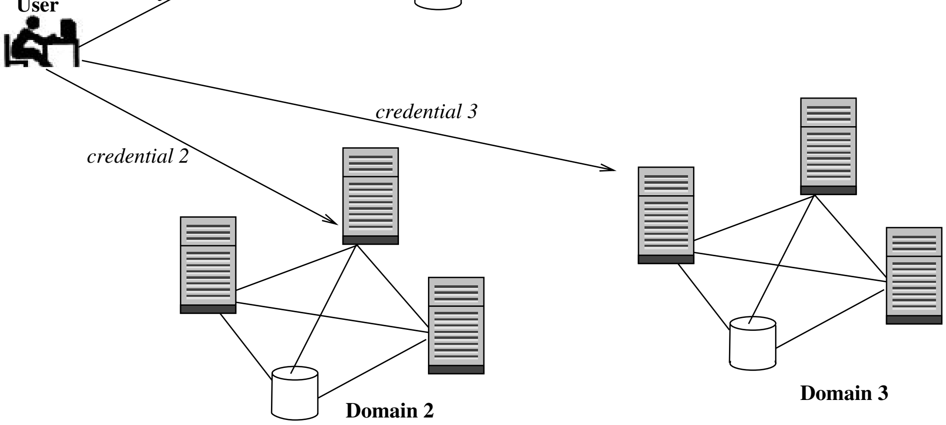


Figure 9. Multiple domains requiring multiple credentials.

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" MajorVersion="1" MinorVersion="1"
AssertionID="..." Issuer="https://MyOrganization.org/saml/" IssueInstant="2005-01-05T17:05:37.795Z">
  <saml:Conditions NotBefore="2005-01-05T17:05:37.795Z" NotOnOrAfter="2005-07-05T17:05:37.795Z"/>
  <saml:AuthenticationStatement AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
  AuthenticationInstant="2005-01-05T17:05:37.700Z">
    <saml:Subject>
      <saml:NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
        user@mo.org
      </saml:NameIdentifier>
      <saml:SubjectConfirmation>
        <saml:ConfirmationMethod>
          urn:oasis:names:tc:SAML:1.0:cm:artifact
        </saml:ConfirmationMethod>
      </saml:SubjectConfirmation>
    </saml:Subject>
  </saml:AuthenticationStatement>
</saml:Assertion>
```

Figure 10. An example of SAML authentication assertion.

authentication statements and attribute statements may be included in a single assertion. Beside the format of the assertion, the SAML specifications include an XML-based request-response protocol for interchanging the assertions. By the use of this protocol, an assertion consuming service can request a specified assertion belonging to a given subject or entity. This protocol can be based on various existing transport protocols such as HTTP or SOAP over HTTP.

Attribute-based Access Control

As said, in open environments, resource/service requesters are not identified by unique names but depend on their *attributes* (usually substantiated by certificates) to gain accesses to resources. Basing authorization on attributes of the the resource/service requester provides flexibility and scalability that is essential in the context of large distributed open systems, where subjects are identified by using a certificate-based authentication language. Attribute-based access control differs from the traditional discretionary access control model by replacing both the *subject* by a set of attributes and the *objects* by descriptions in terms of available properties associated with them. The meaning of a stated attribute may be a granted capability for a service, an identity, or a nonidentifying characteristic of a user (e.g., a skill). Here, the basic idea is that not all access control decisions are identity based. For instance, information about a user's current role (e.g., physician) or a client's ability to pay for a resource access may be more important than the client's identity.

Several proposals have been introduced for access control to distributed heterogeneous resources from multiple sources based on the use of attribute certificates (37–40). In such a context, two relevant access control languages using XML are WS-Policy (41) and XACML (42). Based on the WS-Security (43), WS-Policy provides a grammar for expressing Web service policies. The WS-Policy includes a set of general messaging related assertions defined in WS-Policy Assertions (44) and a set of security policy assertions related to supporting the WS-Security specification defined in WS-SecurityPolicy (45). In addition to the WS-Policy, WS-Policy Attachment (46) defines how to attach these policies to Web services or other subjects (e.g., service locators). The eXtensible Access Control Markup Language (XACML) (42) is the result of a recent OASIS standardization effort proposing an XML-based language to express and interchange access control policies. XACML is designed to express authorization policies in XML against objects that are themselves identified in XML. The language can represent the functionalities of most policy representation mechanisms. The main conceptual difference between XACML and WS-Policy is that although XACML is based on a model that provides a *formal* representation of the access control security policy and its workings, WS-Policy has been developed without taking into consideration this modeling phase. The result is an ambiguous language that is subject to different interpretations and uses. This means that given a set of policies expressed by using the syntax and semantics of WS-Policy, their evaluation may have a different result depending on how the ambiguities of the language have been resolved. This is obviously a serious problem, especially in the access control area, where access decisions have to be deterministic (47,48). In the remainder of this section, we focus on XACML.

XACML Policy Language Model. XACML describes both an access control policy language and a request/response language. The policy language is used to express access control policies (who can do what when). The request/response language expresses queries about whether a particular access (request) should be allowed and describes answers (responses) to those queries (see the next subsection). Figure 11 formally illustrates the *XACML policy language model,* where the main concepts of interests are *rule, policy,* and *policy set.* An XACML policy has as a root element, either a `Policy or a PolicySet`. A `PolicySet` is a collection of `Policy` or `PolicySet`. An XACML policy consists of a *target,* a set of *rules,* an optional set of *obligations,* and a *rule combining algorithm.*

Target. A `Target` basically consists of a simplified set of conditions for the subject, resource, and action that must be satisfied for a policy to be applicable to a given access request: If all conditions of a `Target` are satisfied, its

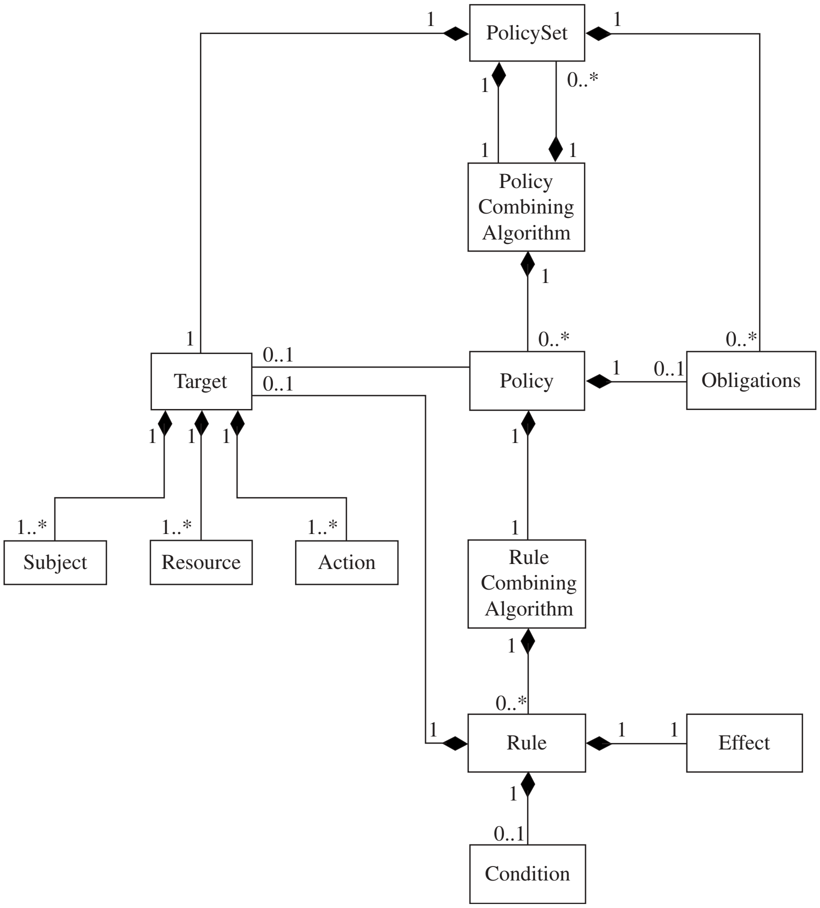


Figure 11. XACML policy language model 42.

associated `Policy` (or `Policyset`) applies to the request. If a policy applies to all subjects, actions, or resources, an empty element, named `AnySubject`, `AnyAction`, and `AnyResource`, respectively, is used.

Rule. A Rule specifies a permission (`permit`) or a denial (`deny`) for a subject to perform an action on an object. The components of a rule are a *target,* an *effect,* and a *condition.* The target defines the set of resources, subjects, and actions to which the rule is intended to apply. The effect of the rule can be `permit` or `deny`. The condition represents a boolean expression that may further refine the applicability of the rule. Note that the `Target` element is an optional element: A rule with no target applies to all possible requests. An important feature of XACML is that a rule is based on the definition of attributes corresponding to specific characteristics of a subject, resource, action, or environment. For instance, a physician at a hospital may have the attribute of being a researcher, a specialist in some field, or many other job roles. According to these attributes, that physician can perform different functions within the hospital. As another example, a particular function may be dependent on the time of the day (e.g., access to the patient records can be limited to the working hours of 8:00 am to 6:00 pm). An access request is mainly composed of attributes that will be compared with attribute values in a policy to make an access decision. Attributes are identified by the `SubjectAttributeDesignator, ResourceAttributeDesignator, ActionAttributeDesignator`, and `EnvironmentAttributeDesignator` elements. These elements use the `AttributeValue` element to define the value of a particular attribute. Alternatively, the `AttributeSelector` element can be used to specify where to retrieve a particular attribute. Note that both the attribute designator and the attribute selector elements can return multiple values. For this reason, XACML provides an attribute type called *bag*. A bag is an unordered collection and it can contain duplicates values for a particular attribute. In addition, XACML defines other standard value types such as string, boolean, integer, and time. Together with these attribute types, XACML also defines operations to be performed on the different types, such as equality operation, comparison operation, and string manipulation.

Obligation. An `Obligation` is an operation that has to be performed in conjunction with the enforcement of an authorization decision. For instance, an obligation can state that all accesses on medical data have to be logged. Note that only policies that are evaluated and have returned a response of `permit` or `deny` can return obligations. More precisely, an obligation is returned only if the effect of the policy matches the value specified in the `FullfillOn` attribute associated with the obligation. This means that if a policy evaluates to `indeterminate` or `not applicable`, then the associated obligations are not returned.

Rule Combining Algorithm. Each `Policy` also defines a rule combining algorithm used for reconciling the decisions each rule makes. The final decision value, called the *authorization decision,* inserted into the XACML context is the value of the policy as defined by the rule combining algorithm. XACML defines different combining algorithms. Examples of these are as follows:

- *Deny overrides.* If a rule exists that evaluates to `deny` or, if all rules evaluate to `not applicable`, then the result is `deny`. If all rules evaluate to `permit`, then the result is `permit`. If some rules evaluate to `permit` and some evaluate to `not applicable`, then the result is `permit`.
- *Permit overrides.* If a rule exists that evaluates to `permit`, then the result is `permit`. If all rules evaluate to `not applicable`, then the result is `deny`. If some rules evaluate to `deny` and some evaluate to `not applicable`, then the result is `deny`.
- *First applicable.* Each rule is evaluated in the order in which it appears in the `Policy`. For each rule, if the target matches and the conditions evaluate to true, then the result is the effect (`permit` or `deny`) of such a rule. Otherwise, the next rule is considered.

```
<Policy PolicyId="Hospital1" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
            http://www.hospital.com/research_report/reports.html
          </AttributeValue>
          <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#anyURI"
          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Rule RuleId="ReadReport" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              read
            </AttributeValue>
            <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="group"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        SeniorResearcher
      </AttributeValue>
    </Condition>
  </Rule>
</Policy>
```

Figure 12. An example of XACML policy.

- *Only-one-applicable.* If more than one rule applies, then the result is `indeterminate`. If no rule applies, then the result is not `applicable`. If only one policy applies, the result coincides with the result of evaluating that rule.

In summary, according to the selected combining algorithm, the authorization decision can be `permit, deny, not applicable` (when no applicable policies or rules could be found), or `indeterminate` (when some errors occurred during the access control process).

The `PolicySet` element consists of a set of *policies,* a *target,* an optional set of *obligations,* and a *policy combining algorithm.* The policy, target, and obligation components are as described above. The policy combining algorithms define how the results of evaluating the policies in the policy set have to be combined when evaluating the policy set. This value is then inserted in the XACML response context. As an example of policy, suppose that there is a hospital where a high-level policy states that:

Any member of the `SeniorResearcher` group can read the research report web page www.hospital.com/research report/reports.html.

Figure 12 shows the XACML policy corresponding to this high-level policy. The policy applies to requests on the web page http://www.hospital.com/research report/reports.html. It has one rule with a target that requires an action of read and a condition that applies only if the subject is a member of the group `SeniorResearcher`.

XACML Request and Response

XACML defines a standard format for expressing requests and responses. More precisely, the original request is translated, through the context handler, in a canonical form and is then evaluated. Such a request contains attributes for the subject, resource, action, and optionally, for the environment. Each request includes exactly one set of attributes for the resource and action and at most one set of environment attributes. There may be multiple sets of subject attributes, each of which is identified by a category URI. Figure 13(a) illustrates the XSD Schema of the request. A response element contains one or more results, each of which correspond to the result of an evaluation. Each result contains three elements, namely `Decision`, `Status`, and `Obligations`. The `Decision` element specifies the authorization decision (i.e., `permit, deny, indeterminate, not applicable`), the `Status` element indicates if some error occurred during the evaluation process, and the optional `Obligations` element states the obligations that must be fulfilled. Figure 13(b) illustrates the XSD Schema of the response.

As an example of request and response, suppose now that a user belonging to group `SeniorResearcher` and with email user1@hospital.com wants to read the www.example.com/forum/private.html web page. The corresponding XACML request is illustrated in Fig. 14(a). This request is compared with the XACML policy in Fig. 12. The result is that the user is allowed to access the requested web page. The corresponding XACML response is illustrated in Fig. 14(b).

```
<xs:element name="Request" type="xacml-context:RequestType">
<xs:complexType name="RequestType">
  <xs:sequence>
    <xs:element ref="xacml-context:Subject" maxOccurs="unbounded">
    <xs:element ref="xacml-context:Resource">
    <xs:element ref="xacml-context:Action">
    <xs:element ref="xacml-context:Environment" minOccurs="0">
  </xs:sequence>
</xs:complexType>
```

(a)

```
<xs:element name="Result" type="xacml-context:ResultType">
<xs:complexType name="ResultType">
  <xs:sequence>
    <xs:element ref="xacml-context:Decision">
    <xs:element ref="xacml-context:Status">
    <xs:element ref="xacml-context:Obligations" minOccurs="0">
  </xs:sequence>
  <xs:attribute name="ResourceId" type="xs:string" use="optional">
</xs:complexType>
```

(b)

Figure 13. (a) XACML request schema and (b) response schema.

CONCLUSIONS

Ensuring protection to information stored in a computer system means safeguarding the information against possible violations to its secrecy, integrity, or availability. This is a requirement that any information system must satisfy and that requires the enforcement of different protection methods and related tools. Authentication, access control, auditing, and encryption are all necessary to this task. As it should be clear from this article, these different measures are not independent but strongly dependent on each other. Access control relies on good authentication, because accesses allowed or denied depend on the identity of the user requesting them. Strong authentication supports good auditing, because users can be held accountable for their actions. Cryptographic techniques are necessary to ensure strong authentication, for example, to securely store or transmit passwords. A weakness in any of these measures may compromise the security of the whole system (a chain is as strong as its weakness link). Their correct and coordinated enforcement is therefore crucial to the protection of the information.

ACKNOWLEDGMENT

This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591 and by the Italian MIUR within the KIWI and MAPS projects.

```
<Request>
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
    DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
      <AttributeValue>user1@hospital.com</AttributeValue>
    </Attribute>
    <Attribute AttributeId="group" DataType="http://www.w3.org/2001/XMLSchema#string"
    Issuer="administrator@hospital.com">
      <AttributeValue>SeniorResearcher</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <AttributeValue>http://www.hospital.com/research_report/reports.html</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
    DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>read</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

(**a**)

```
<Response>
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

(**b**)

Figure 14. (a) An example of XACML request and (b) response.

BIBLIOGRAPHY

1. S. Castano, M.G. Fugini, G. Martella, and P. Samarati, *Database Security*, Reading, MA: Addison-Wesley, 1995.
2. F. Monrose and A. Rubin, Authentication via keystroke dynamics, *Proc. of the ACM Conference on Computer and Communications Security*, Zurich, Switzerland, 1997.
3. T. Fine and S. E. Minear, Assuring distributed trusted mach, *Proc. IEEE Symp. on Security and Privacy*, Oakland, CA, 1993, pp. 206–218.
4. S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian, Flexible support for multiple access control policies, *ACM Transactions on Database Systems*, **26** (2): 214–260, 2001.
5. O.S. Saydjari, S.J. Turner, D.E. Peele, J.F. Farrell, P.A. Loscocco, W. Kutz, and G.L. Bock, Synergy: A distributed, microkernel-based security architecture, Technical report, National Security Agency, Ft. George G. Meade, MD, November 1993.
6. T.Y.C. Woo and S.S. Lam, Authorizations in distributed systems: A new approach, *Journal of Computer Security*, **2** (2,3): 107–136, 1993.
7. R.W. Baldwin, Naming and grouping privileges to simplify security management in large databases, *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, 1990, pp. 61–70.
8. T. Lunt, Access control policies: Some unanswered questions, *IEEE Computer Security Foundations Workshop II,* Franconia, NH, June 1988, pp. 227–245.
9. H. Shen and P. Dewan, Access control for collaborative environments, *Proc. Int. Conf. on Computer Supported Cooperative Work*, November pp. 51–58.
10. E. Bertino, S. Jajodia, and P. Samarati, Supporting multiple access control policies in database systems, *Proc. IEEE Symp. on Security and Privacy*, Oakland, CA, 1996.
11. E. Bertino, S. Jajodia, and P. Samarati, A flexible authorization mechanism for relational data management systems, *ACM Transactions on Information Systems*, **17** (2): 101–140, 1999.
12. F. Rabitti, E. Bertino, W. Kim, and D. Woelk, A model of authorization for next-generation database systems, *ACM Transactions on Database Systems*, **16** (1), 1991.
13. E. Bertino, S. De Capitani di Vimercati, E. Ferrari, and P. Samarati, Exception-based information flow control! in object-oriented systems, *ACM Transactions on Information and System Security,***1** (1): 26–65, 1998.
14. D.E. Denning, A lattice model of secure information flow, *Communications of the ACM*, **19** (5): 236–243, 1976.
15. R. Graubart, On the need for a third form of access control, *NIST-NCSC National Computer Security Conference*, 1989, pp. 296–303.
16. P.A. Karger, Limiting the damage potential of discretionary trojan horses, *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, 1987.
17. C.J. McCollum, J.R. Messing, and L. Notargiacomo, Beyond the pale of mac and dac - defining new forms of access control, *Proc. IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA, 1990, pp. 190–200.
18. J. McLean, Security models and information flow, *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, 1990, pp. 180–187.

19. D.E. Bell and L.J. LaPadula, Secure computer systems: Unified exposition and Multics interpretation, Technical report, The Mitre Corp., March 1976.
20. D.F. Ferraiolo and R. Kuhn, Role-based access controls, *Proc. of the NIST-NCSC National Computer Security Conference*, Baltimore, MD, 1993, pp. 554–563.
21. R. Sandhu, E.J Coyne, H.L. Feinstein, and C.E. Youman, Role-based access control models, *IEEE Computer*, **29** (2): 38–47, 1996.
22. D.J. Thomsen, Role-based application design and enforcement, in S. Jajodia and C.E. Landwehr, (eds.), *Database Security IV: Status and Prospects*, North-Holland, 1991, pp. 151–168.
23. D.F.C Brewer and M.J. Nash, The chinese wall security policy, *Proc. IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA, 1989, pp. 215–228.
24. M.N. Nash and K.R. Poland, Some conundrums concerning separation of duty, *Proc. IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA, 1982, pp. 201–207.
25. R. Sandhu, Transaction control expressions for separation of duties, *Fourth Annual Computer Security Application Conference*, Orlando, FL, 1988, pp. 282–286.
26. D.E. Denning, *Cryptography and Data Security*. Reading, MA: Addison-Wesley, 1982.
27. C.E. Shannon, Communication theory of secrecy systems, *Bell System Technical Journal*, **28** (4): 656–715, October 1949.
28. National Bureau of Standard, Washington, D.C, *Data Encryption Standard,* January 1977. FIPS PUB 46.
29. National Institute of Standards and Technology (NIST), Washington, D.C, *Advanced Encryption Standard (AES),* November 2001. FIPS-197.
30. W. Diffie and M. Hellman, New directions in cryptography, *IEEE Transaction on Information Theory*, **22** (6): 644–654, 1976.
31. T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transaction on Information Theory*, **31** (4): 469–472, 1985.
32. R.L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, **21** (2): 120–126, February 1978.
33. C. Kaufman, R. Perlman, and M. Speciner, *Network Security*, Englewood Cliffs, NJ: Prentice Hall, 1995.
34. S. Feldman, The Changing Face of E-Commerce, *IEEE Internet Computing*, **4** (3): 82–84, 2000.
35. P. Samarati and S. De Capitani di Vimercati, Access control: Policies, models, and mechanisms, in R. Focardi and R. Gorrieri, (eds.), *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag, 2001.
36. OASIS, *Security Assertion Markup Language (SAML) V1.1,*2003. Available: http://www.oasis-open.org/committees/security/.
37. P. Bonatti and P. Samarati, A unified framework for regulating access and information release on the web, *Journal of Computer Security*, **10** (3): 241–272, 2002.
38. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati, Securing SOAP E-services, *International Journal of Information Security (IJIS)*, **1** (2): 100–115, 2002.
39. J.A. Hine, W. Yao, J. Bacon, and K. Moody, An architecture for distributed OASIS services, *Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms nd Open Distributed Processing*, Hudson River Valley, New York, 2000.
40. H. Koshutanski and F. Massacci, An access control framework for business processes for web services, *Proc. of the 2003 ACM workshop on XML security*, Fairfax, VA, 2003.
41. D. Box et al., Web Services Policy Framework (WS-Policy) version 1.1. Available: http://msdn. microsoft.com/library/en-us/dnglobspec/html/ws-policy.asp, May 2003.
42. OASIS eXtensible Access Control Markup Language (XACML) version 1.1. Available: http://www. oasis-open.org/committees/xacml/repository/cs-xacml-specific%ation-1.1.pdf.
43. B. Atkinson and G. Della-Libera et al. Web services security (WS-Security). http://msdn. microsoft.com/library/en-us/dnglobspec/html/ws-security.asp%, April 2002.
44. D. Box et al., Web services policy assertions language (WS-PolicyAssertions) version 1.1. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyassert%ions.asp, May 2003.
45. Web services security policy (WS-SecurityPolicy), Available: http://www-106.ibm.com/developerworks/library/ws-secpol/. December 2002.
46. D. Boxet al. Web Services Policy Attachment (WS-PolicyAttachment) version 1.1. Available: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyattach%ment.asp, May 2003.
47. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, An XPath-based preference language for P3P, *Proc. of the World Wide Web Conference*, Budapest, Hungary, 2003.
48. C. Ardagna and S. De Capitani di Vimercati, A comparison of modeling strategies in defining xml-based access control languages, *Computer Systems Science & Engineering Journal,* **19** (3), 2004.

SABRINA DE CAPITANI DI VIMERCATI
PIERANGELA SAMARATI
Università degli Studi di Milano
Crema, Italy
SUSHIL JAJODIA
George Mason University
Fairfax, Virginia

DATA STRUCTURES AND ALGORITHMS

INTRODUCTION

An *algorithm* is a finite procedure, with a well-defined set of steps, aimed at solving a specific problem. Algorithms are present in any problem-solving task involving the use of computers in numerical or nonnumerical applications. An algorithm is usually required to stop in a finite number of steps, although some useful procedures are meant to be run continuously, such as operating systems and related system software. These procedures are sometimes not regarded as algorithms, but simply as programs. In general, a *program* is a set of machine-readable and executable codes used to implement algorithms.

These definitions make clear the fact that an algorithm is a high-level description, which should be independent of the underlying machine that is used to implement it. This requirement is important to simplify the task of analysis of the properties of an algorithm: We do not need to consider all possible machine details necessary to implement it, but only the properties common to all such implementations.

Algorithms can be classified and studied according to their performance characteristics. Among all algorithms to solve a specific problem, we clearly want the ones that have better performance, according to established performance measurements for an application. Among the performance parameters used for classification purposes, the most important are *time* and *space*. For a specified problem, the time and space taken by the algorithm with best possible performance are called the *time* and *space complexity* measures for the problem.

Space is a simpler measure to analyze, as it is clearly tied to time: An algorithm cannot use more than t units of space, where t is the total time spent by the algorithm. Usually, the space used by an algorithm is clearly identifiable from the data structures used in its operation. For example, if an algorithm operates using only a vector that depends on the size of the input, the space complexity of the algorithm can be bounded by a function of the input size.

On the other hand, time complexity is difficult to characterize in general. In fact, the most fundamental question about an algorithm, whether it stops in finite time or not, is known to be undecidable (i.e., it cannot be answered in general). Although this result is a very negative result, it is still possible to analyze the time complexity of several classes of practical algorithms, which is the main objective of the area of *analysis of algorithms*. This type of analysis is important because time complexity is the main restriction on applying an algorithm in practice. Consider, for example, an algorithm that takes time exponential in the size of the input. Such an algorithm is not practical, unless the size of the inputs for the considered problem is always very small. For general inputs, such algorithm will easily surpass any reasonable time limit.

More formally, the *time complexity* of an algorithm is usually defined as the total number of elementary operations (arithmetic operations, comparisons, branching instructions, etc.) performed. The time complexity is generally quantified as a function of the *input size*. The input size of a problem is the total number of bits required to represent all data necessary for solving the problem. For example, if an algorithm receives as input a vector with n integer values, each value represented in binary notation, then the number of bits required to represent the input is $n[\log K]$, where K is an upper bound on the maximum value stored on any position of the input vector (note that all logarithms discussed in this article are taken with respect to the base 2). It is common to assume that integer numbers used in an algorithm are all limited in size to a given upper bound, unless it is an important parameter (for example, in numerical algorithms such as factorization). Thus, it is usual to say that the input size for a vector of n integers is equal to n.

When analyzing an algorithm, the main concern is in understanding its performance under different conditions. Most common conditions considered are the following:

- *Worst case:* By the worst case of an algorithm, we mean the maximum time necessary to compute its result, for some fixed input size. The worst case analysis is usually considered the best way of analyzing algorithms, because it gives a time complexity bound that is independent of the input. It is also surprisingly easy to compute for most problems.
- *Average case:* The average case analysis of an algorithm is concerned with the expected time necessary for the algorithm to run, for all data inputs with fixed size. Average case analysis generally requires some assumptions about the probabilistic distribution of input data, and therefore the results are dependent on such assumptions, which are not easy to guess in most cases. Moreover, the stochastic analysis of even the simplest algorithms may become a daunting task.
- *Best case:* It is the time complexity of an algorithm under the best conditions for a specific input size. Although simple to compute in most cases, the best case analysis is not very useful to predict the actual performance of an algorithm in real situations.

From the three types of analysis shown above, worst case analysis is the most interesting from the theoretical as well as practical point of view. As an example of the above complexity cases, consider the classic problem of searching a specific value x in a list of n elements. The simplest algorithm consists of comparing x with each element in sequence, from positions 1 to n. The worst case of this algorithm occurs when the element searched is not present, or when present in the last position. The best case occurs when the searched element is in position 1. The average case of the algorithm, however, depends on the distribution

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

of the input. If we assume that each value has the same probability to occur in each position and there are no repeated values, then, on average, $n/2$ comparisons are necessary to find the desired element.

To simplify the study of time and space requirements of algorithms, asymptotic notation is usually employed. This notation allows one to ignore constant factors in the running time, making the analysis independent of different specific details developing from the algorithm implementation. Given the integer functions f and g, we say that $f(n) = \mathcal{O}(g(n))$ (which is read as "$f(n)$ is of the order of $g(n)$") if there are constants $c > 0$ and $N > 0$, such that $f(n) \leq c \cdot g(n)$ for all $n \geq N$. Similarly, given the integer functions f and g, we say that $f(n) = \Omega(n)$ if there are constants $c > 0$ and $N > 0$, such that $f(n) \geq c \cdot g(n)$ for all $n \geq N$. The notations for O and Ω can be also combined by saying that $f(n) = \Theta(n)$ if and only if $f(n) = \mathcal{O}(n)$ and $f(n) = \Omega(n)$.

DATA STRUCTURES

Data structures are a fundamental aspect of algorithms, because they describe how data will be manipulated in the computer's memory. The organization of data can make the difference between effcient and inefficient algorithms. Consider, for example, the problem of inserting a number in the middle of an ordered list with n elements. This operation can be done in constant ($\mathcal{O}(1)$) time if the list is manipulated as a set of linked elements. On the other hand, if it is represented as an array with fixed positions, it is necessary to create space in the array for the required position, which requires at least $n/2$ elements to be moved. Thus, the time complexity in this case is $\mathcal{O}(n)$.

Data structures can be studied from a high-level standpoint, where we consider only the operations performed on data. Data structures described in this way are called *abstract data types* (ADT). ADTs are important because they allow an abstraction of the properties of a data structure, which can have several implementations. Thus, a second way of looking at data structures is through the analysis of the actual implementation of its ADT, as data is organized in the computer's memory.

Queues and Stacks

As an example of ADT, consider the concept of queue. A queue can receive new elements at its end. Elements can be removed only at the beginning of the queue. The concept leads to the formulation of the following natural operations: ADD(ELEMENT): adds ELEMENT to the end of the queue; REMOVE(ELEMENT): removes ELEMENT from the beginning of the queue.

The ADT queue can be implemented using several data structures. The simplest implementation uses an array a with n positions to hold the elements, considering that n is the maximum size of the queue. This implementation has the following properties: The ADD operation can be performed in constant time, as we need only to update the location of the last element (in this implementation, the array is circular, for example, element $a[1]$ comes right after position $a[n]$). Similarly, the REMOVE operation can be implemented just by decreasing the position of the front of the queue, and therefore it can be performed in constant time. A second implementation of queue uses doubly linked lists. This implementation has similar time complexity, because it needs only to add or remove an element and its corresponding links. However, it has the advantage that it is not restricted to the size of a preallocated array.

A second useful ADT is the stack. It has the same operations as a queue, but the semantic of the operations is different: ADD puts a new element on the top of a stack, and REMOVE takes the element that is at its top. Similarly to queues, implementations can be based on arrays or linked lists, operating in constant time.

Queues and stacks are examples of more general data structures that support a set of operations, in this case ADD and REMOVE. This class of data structures include all general *lists*, where insertion and removal can occur in any position of the list.

A well-known class of data structures is the *dictionary*, with the operations INSERT, DELETE, and SEARCH. Still another class is the *priority queue*, where the desired operations are FIND-MIN, DELETE-MIN, and INSERT.

Binary Trees

A binary tree is a recursive structure that can be either empty or have a root node with two children, which are themselves binary trees. Binary trees are important data structures used to implement abstract data types. An example of abstract structure that can be implemented with binary trees are dictionaries.

A special node in the binary tree is the *root* node, which is the top level node. At the lower level of the tree are the *leaves*, defined as subtrees with empty children. The level of a node x is an indicator of distance from x to the root node. The level of the root is defined to be 1, and the level a child node is defined as 1 plus the level of its parent node. The *height* of a tree is the maximum level of any of its nodes.

The main feature of binary trees is that they can be used to segment the domain of elements stored in their nodes into two subdomains at each node, which allows some algorithms running in binary trees to operate on time $\mathcal{O}(\log n)$. Consider a binary tree implementing a dictionary, where the elements are integer numbers. We can store a number x in the tree using the following strategy: Check initially the root node. If it is empty, store x in the root of the subtree. Otherwise, let the value stored there be equal to r. If $x < r$, then check recursively the left child tree; if $x > r$, then check recursively the left child tree. Searching the binary tree (instead of inserting a node) is similar, but we just need to check if $x = r$ and return when this is true. Note that we are partitioning the solution space into two subsets at each step of the algorithm. The resulting data structure is called a *binary search tree*.

As an example of binary tree, consider Fig. 1. We assume that we want to insert the value 36 into the tree. The first comparison with the root node will determine that the position of the value must be on the right subtree. The second comparison with 45 will then result in the selection of the left subtree. Comparing with the value 38, we see that the element must be on the left subtree. Finally, the comparison with the empty subtree shows that it is the position at which the node with value 36 should be inserted.

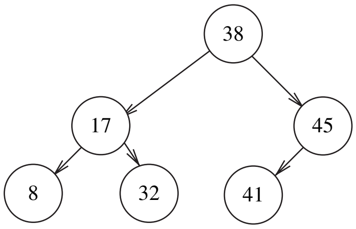


Figure 1. An example of binary tree.

A closer examination of the tree in Fig. 1 reviews that a possible order for insertion of the values in the tree is 38, 17, 45, 8, 32, and 41 when starting from an empty binary tree. Any other ordering of these numbers according to *nondecreasing levels* in the final tree would produce the same result. However, some orderings of these values may produce trees with large height, which is not desirable as we discuss below. For example, the ordering 8, 17, 32, 38, 41, and 45 will generate a tree that has only one branch with height 6.

It is easy to check that operations in binary trees such as Find or Insert have to perform at most h comparisons, where h is the height of the tree. It is also simple to show that the height of a binary tree satisfies $h = \Omega(\log n)$. Therefore, it is an important goal for any algorithm applied to binary trees to keep the height as close as possible to the lower bound of $\log n$. If true, then the algorithms for binary tree operations discussed above will also take time $\Theta(\log n)$.

Several methods have been developed to guarantee that binary trees produced by insertion and searching procedures have the height $\mathcal{O}(\log n)$. These methods, which include *red-black* trees, 2-3 trees, and AVL trees, provide elaborate techniques for rotating an existing configuration of nodes in the tree, such that the final binary tree is balanced (i.e., it has height close to $\mathcal{O}(\log n)$). These ideas can be stated as in the following theorem:

Theorem 1. *A dictionary and a priority queue can be implemented such that the* Find *and* Insert *operations take time* $\Theta(\log n)$.

Graphs

A graph is a data structure that represents a set of nodes (possibly with labels), together with relations between pairs of nodes. Graphs are denoted as $G = (V, E)$, where V is a nonempty set and $E \subseteq V \times V$. They are a fundamental data structure for processing discrete information, because a large number of combinatorial problems can be expressed using graphs. A well-known example is the *traveling salesman problem*. Consider a set of n cities, with a set of roads connecting pairs of cities. This system can be represented as a graph $G = (V, E)$, where V is the set of cities and E is the set of all pairs $(u, v) \in V \times V$ such that there is a road between cities u and v with cost $c(u, v)$. The problem requires finding a tour through all cities with minimum cost.

Graphs are classified into *directed* and *undirected* graphs. In a directed graph, the edge (i, j) has an associated direction. In an undirected graph, however, the direction is not specified, and therefore $\{i, j\}$ and $\{j, i\}$ are notations for the same edge.

A graph is an abstract data type with the following basic operations: InsertEdge, RemoveEdge, Find, Adjacent, and Connected. Other operations are also interesting, but they vary according to the application. Graphs can be represented using different data structures. The simplest representation is called the *node incidence matrix* representation. In this case, the graph $G = (V, E)$ with n nodes is represented by a $n \times n$ matrix $A = (a_{ij})$, such that $a_{ij} = 1$ if $(i, j) \in E$ and $a_{ij} = 0$ otherwise. A graph is called *sparse* when the number of edges is $\mathcal{O}(|V|)$. The node incidence matrix is a convenient representation, but it usually leads to suboptimal algorithm performance, especially when the graph is sparse, which happens, for example, when the algorithm needs to find all edges in a graph, which requires $\Theta(n^2)$ in the matrix representation, as all pairs $(i, j) \in V \times V$ need to be probed for adjacency.

A variation of the first representation for graphs is the *node-edge* incidence matrix. This representation is more useful for directed graphs, although it can also be used with undirected graphs by simultaneously considering both the edges (i, j) and (j, i). In the node-edge incidence matrix representation, a graph $G = (V, E)$ with n nodes and m edges is represented by an $n \times m$ matrix $A = (a_{ie})$, where $a_{ie} = -1$ if $e = (i, j) \in E$, $a_{ie} = 1$ if $e = (j, i) \in E$, and $a_{ie} = 0$ otherwise.

A third representation for graphs, which is more useful when the graph is sparse, is called the *adjacency list* representation. In this representation, an array a with n positions is used, where $a[i]$ keeps a link to a list of nodes that are adjacent to node i. With this data representation, it is possible to iterate over all edges of the graph by looking at each linked list, which takes time $\Omega(|E|)$, instead of $\Omega(|V|^2)$ as in the node incidence matrix representation, which means that all algorithms that need to iterate over the edges of a graph will have improved performance on sparse graphs, when compared with the matrix representations.

ADVANCED DATA STRUCTURES

Problems involving data manipulation can usually benefit from the use of more advanced data structures. Examples can be found in areas such as computational geometry, search and classification of Internet content, and fast access to external memory. Although the complex techniques employed in such applications cannot be fully described here because of space constraints, we provide references for some problems that employ advanced data structures to be solved.

External memory algorithms have application whenever there is a large amount of data that cannot fit in the computer's main memory. Examples of such applications include manipulation of scientific data, such as meteorologic, astrophysical, geological, geographical, and medical databases. The general problem, however, is the same encountered in any computer system with memory hierarchies levels, such as cache memory, main memory, and external disk storage.

A data structure frequently used to access out-of-memory data is the B-tree. A B-tree is a type of balanced tree designed to perform optimally when large parts of the stored data is out of the computer's main memory. The challenge in this kind of application is that accessing secondary storage, such as a hard disk, is many orders of magnitude slower than accessing the main memory. Thus, the number of accesses to secondary storage is the bottleneck that must be addressed. The basic design strategy of B-trees is to reduce the amount of blocks of the hard disk that must be read, in order to locate some specific information in the data structure. This strategy minimizes the time lost in accessing secondary storage. Several improvements to B-trees have been proposed in the last decades, with the objective of making this adequate for new applications. For example, cache-oblivious B-trees (1,2) are extensions of the B-tree model where the algorithm has no information about the sizes of the main memory cache or secondary storage block.

Another example of external memory data structure that tries to reduce the number of disk accesses is the *buffer tree* (3). The basic idea of a buffer tree is that elements of the tree are considered to be blocks of the secondary storage medium, so that access to these blocks is minimized.

An example of application of external memory algorithms to geographical databases is the problem of processing line segments (4). In such databases, large amounts of data must be processed, and questions must be answered fast, with the minimum number of accesses to secondary storage. The type of questions occurring here include how to find if two line segments stored in the database intersect, or how to optimally triangulate a specific area.

Data structures for external memory algorithms also play an important role in the solution of some graph theory problems, occurring in applications such as telecommunications, financial markets, medicine, and so on. An example of large network database is provided by the so-called *call graph*, created from information about calls among users of AT&T (5). The call graph is defined as having nodes representing the users of the telephony system. Edges connect nodes whenever there is a call between the corresponding users during some specified period of time. The resulting database had several million nodes and a similar number of edges. The whole amount of data had to be held on special data structures for fast processing.

Questions that are interesting in this kind of graph are, for example, the number of connected components, the average size of such connected components, and the maximum size of cliques (groups of completely connected users) in the graph. In this application, special data structures had to be created to explore the sparsity of the graph and avoid the access to secondary storage, which resulted in an algorithm with efficient performance that was able to find large sets of completely connected nodes in the call graph.

ALGORITHMS

General Techniques

The development of a new algorithm for a problem is always a creative effort, because there is no easy, general way of doing that (note also, that some problems may have not a finite algorithm (6)). However, some general approaches used for the construction of algorithms have demonstrated their usefulness and applicability for solving different problems of varied natures and origins. Next, we briefly describe some of these techniques.

Divide and conquer is a strategy for development of algorithms in which the problem is divided into smaller parts that are then solved *recursively* (a recursive algorithm is an algorithm that calls itself). This strategy has been successful in several areas, such as sorting, searching, and computational geometry algorithms. In the next section, we discuss examples of this approach. The simplicity of this type of algorithm stems from its recursive nature: We do not have to regard the solution of large problems, because the recursion will break the input data into smaller pieces.

Dynamic programming is a technique that also tries to divide the problem into smaller pieces; however, it does it in the way that solutions of smaller problems can be orderly reused, and therefore they do not need to be computed more than once. An example of such an algorithm is the Floid's method for finding all shortest paths in a graph. The algorithm basically computes solutions where paths can pass only through some subset of nodes. Then, the final solution is built using this information.

Greedy methods have been frequently used for the construction of efficient algorithms. A greedy algorithm always makes decisions that will maximize a short-term goal, and employs the results to build a complete solution for the problem. Using again the shortest path problem as an example, the Dijkstra's algorithm is a type of greedy method. In this case, the function that is minimized at each step is the distance to the set of nodes previously considered by the algorithm.

Finally, we can mention some other important algorithm construction methods such as *backtracking*, *enumeration*, and *randomization*. The use of such techniques has allowed to develop many efficient algorithms, which provides optimal or approximate solutions to a great variety of problems in numerous areas.

Algorithms for Some Basic Problems

In this section, we discuss some basic algorithms for problems such as binary search, sorting, matrix multiplication, and minimum weight spanning tree.

Sorting. Suppose we are given an array of numbers $a_1, \ldots, a_n$. Our task is to sort this array in nondecreasing order such that $a_1 \leq a_2 \leq \ldots \leq a_n$. Probably the most obvious algorithm for this problem is the so-called *selection sort*. Its main idea is the following. First, we find the smallest number of these n elements and interchange it with a_1. The number of comparisons we need is exactly (n−1). Next, we repeat this procedure with the array $a_2, \ldots, a_n$, then with the array $a_3, \ldots, a_n$, and so on. The total number of required comparisons is

$$(n-1) + (n-2) + \ldots + 1 = \Theta(n^2)$$

A better algorithm called the *merge sort* can be developed using the divide and conquer strategy. Initially, we divide the input vector into two parts, say from a_1 to $a_{n/2}$ and from $a_{n/2+1}$ to a_n. Then, we can call the sorting algorithm *recursively* for the smaller vectors $a_1, \ldots, a_{n/2}$ and $a_{n/2+1}, \ldots, a_n$ (the recursion can be solved very easily when the vectors to be sorted have size 1 or 2; therefore, the algorithm will always end with a correctly sorted vector). Finally, we need to *merge* these two sorted arrays into one sorted array. This procedure can be done in $\Theta(n)$ time. If $T(n)$ is the total number of comparisons made by an algorithm for the input array of size n, then for the described algorithm $T(n)$ satisfies $T(n) = 2T(n/2) + \Theta(n)$, which results in $T(n) = \Theta(n \log n)$.

Next, we show that merging two arrays respectively with l and m elements can be done in $\Theta(l + m)$ time. Let $X = x_1, \ldots, x_l$ and $Y = y_1, \ldots, y_m$ be our two input arrays sorted in nondecreasing order and Z be an auxiliary vector of length $l + m$. As X and Y are sorted, if we compare x_1 and y_1, then the smallest of these two numbers is also the smallest number of X and Y combined. Suppose, for example, $x_1 \leq y_1$. Let $z_1 = x_1$ and remove x_1 from X. Repeating the procedure described above with new X and Y, we obtain z_2. Proceed this way until one of the arrays becomes empty. Finally, keeping the order, we can output all the elements of the remaining array to Z. Obviously, the obtained array Z contains the needed result. It is also easy to observe that every time we move an element to Z we remove one of the elements of X or Y and make exactly one comparison. Therefore, the number of comparisons we perform is $\Theta(l + m)$.

Proposition 1. *We can sort an array with n elements in* $\Theta(n \log n)$ *time*.

It can also be shown that any sorting algorithm requires at least $\Omega(n \log n)$ comparisons. Therefore, the merge sort is asymptotically optimal.

Binary Search. In the binary search problem, for an array $a_1, \ldots, a_n$ sorted in nondecreasing order, we need to check if a given element x is present in this array. A divide-and-conquer strategy can be used to design a simple and effective algorithm for solving this problem.

As the first step of the algorithm, we check whether $x = a_{n/2}$. If it is true, then the problem is solved. Otherwise, because the array is sorted in nondecreasing order, if $x > a_{n/2}$, then we conclude that x cannot be in the first part of the array, for example, x cannot be present in $a_1, \ldots, a_{n/2}$. Applying the same argument, if $x < a_{n/2}$, then the second part of the array, which is $a_{n/2}, \ldots, a_n$, can be excluded from our consideration. Next, we repeat the procedure described above with the remaining half of the initial array. At every step, the size of the array reduces by a factor of 2. Therefore, denoting by $T(n)$ the number of comparisons, we obtain that $T(n) = 2T(n/2) + 1$. Solving the recurrence equation, we have $T(n) = \Theta(\log n)$.

Matrix Multiplication. Suppose that, given two $n \times n$ matrices A and B, we need to calculate their product $C = AB$. Let a_{ij}, b_{ij}, and c_{ij} be the elements of matrices A, B, and C, respectively. Then, by definition, for all i and j, we have $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$. Using this formula, every element c_{ij} of C can be calculated in $\Theta(n)$ time. As there are n^2 elements in the matrix C, the matrix C can be computed in $\Theta(n^3)$ time.

Strassen developed an algorithm based on a divide-and-conquer strategy, which requires only $\Theta(n^{\log_2 7})$ time. The main idea of the algorithm is based on the simple observation that two matrices of size 2×2 can be multiplied using only 7 multiplications (instead of 8) and 18 additions (fortunately, the running time of the algorithm asymptotically is not sensible to increasing the number of additions). Let the matrices A and B in the problem be partitioned as follows:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

where the size of matrices A_{ij} and B_{ij} is $(n/2) \times (n/2)$. Then calling the algorithm recursively and applying the Strassen formulas, we obtain the following formula for the running time $T(n)$ of the algorithm:

$$T(n) = 7T(n/2) + \Theta(n^2)$$

The solution of the recursive formula above is $T(n) = \Theta(n^{\log_2 7})$. A more detailed description of the algorithm can be found in textbooks on algorithms, which are listed in the last section of this article.

Minimum Weight Spanning Tree. A *tree* is a connected graph without cycles; that is, it contains a path between every pair of nodes, and this path is unique. Consider an undirected connected graph $G = (V, E)$, where each edge (i, j) has a weight w_{ij}. A spanning tree T_G of the graph G is a subgraph of G that is a tree and spans (i.e., contains) all nodes in V. The *minimum spanning tree* (MST) problem is to find a spanning tree with minimum total sum of its edge weights.

The following algorithm for solving this problem is designed using the greedy approach. Initially, we sort all the edges of the given graph G in nondecreasing order and create a list of edges L. Let T be an empty subgraph of G. Pick an element of L with minimum weight (say e), add e to T, and remove it from L. Proceed with this procedure checking at every step that adding the new edge does not create a cycle in T (if this happens, then we do not add the corresponding edge to T). After $n - 1$ edges are added to T, where n is the number of nodes in G, stop. Denote the obtained subgraph by T_G. The following proposition can be proved:

Proposition 2. *The greedy algorithm described above returns a minimum weight spanning tree* T_G.

Analysis of Algorithms and Complexity Issues

As mentioned above, worst case analysis is probably the most widely used criteria for evaluating algorithms. The basic algorithms discussed (binary search, sorting, selection, matrix multiplication) require only a polynomial number of elementary operations in the size of the input data. In the area of analysis of algorithms and complexity theory, the set of all problems that can be solved in polynomial time (i.e., by a polynomialtime algorithm) is usually denoted by P. Another important class of problems is NP, defined as the collection of all problems for which the correctness of a solution, described using a polynomial-sized encoding, can be verified in polynomial time. Obviously, P is a subset of NP (i.e., $\mathrm{P} \subseteq \mathrm{NP}$). On the other hand, deciding whether $\mathrm{P} \neq \mathrm{NP}$ is probably the most famous open problem in theoretical computer science.

We say that problem II is *polynomial-time reducible* to problem II_1 if given an instance $I(\mathrm{II})$ of problem II, we can, in polynomial time, obtain an instance $I(\mathrm{II}_1)$ of problem II_1 such that by solving $I(\mathrm{II}_1)$ one can compute in polynomial time an optimal solution to $I(\mathrm{II})$. In other words, under this type of reduction, the existence of a polynomial algorithm for solving II_1 will imply the existence of a polynomial algorithm for II. A problem II is called *NP-complete* if $\mathrm{II} \in \mathrm{NP}$, and any problem in NP can be reduced in polynomial time to II, therefore, the class of NP-complete problems consists of the hardest problems in NP. Notice that if any NP-complete problem can be solved in polynomial time, then all problems in NP can be solved in polynomial time (i.e., $\mathrm{P} = \mathrm{NP}$).

In 1971, Steve Cook provided the foundation of NP-completeness theory, proving that the SATISFIABILITY problem is NP-complete (7) (a formal definition of this problem is omitted here). Since that time, the number of NP-complete problems has been significantly increased. Another classic NP-complete problem is the so-called SUBSET SUM problem: Given a set of positive integers $S = \{s_1, s_2, \ldots, s_n\}$ and a positive integer K, does there exist a vector $\mathbf{x} \in \{0,1\}^n$, such that $\sum_{i=1}^{n} s_i x_i = K$? Other examples of NP-complete problems include the TSP (see section Graphs), SET COVER, VERTEX COVER, PARTITION, and so on.

Although it is not known whether $\mathrm{P} \neq \mathrm{NP}$, it is generally assumed that all NP-complete problems are hard to solve. In other words, proving that some problem belongs to the class of NP-complete problems implies that it cannot be solved in polynomial time. The standard procedure of proving that a problem II_1 is NP-complete consists of two steps: (1) proving that $\mathrm{II}_1 \in \mathrm{NP}$ and (2) reducing in polynomial time some known NP-complete problem II to the problem II_1.

Finally, let us also consider the definition of NP-hardness. We say that a problem II belongs to the class of NP-hard problems if there exists an NP-complete problem that can be reduced in polynomial time to II. Therefore, the problem II is at least as hard to solve as any other NP-complete problem, but it is not known whether this problem belongs to NP.

For more detailed information on the theory of NP-completeness and rigorous definitions (including the formal Turing machine model), we refer to the famous book by Garey and Johnson (8) or to the more recent book by Papadimitriou (6).

Approximation Algorithms

The notion of NP-hardness (8) leads researchers to consider alternative strategies to solve problems that are intractable. A possible technique in this case is to solve the problem in an approximate way by not requiring an exact solution, but a solution with an *approximation guarantee*.

An *approximation algorithm* is defined as follows. Let II be a minimization problem, f be the objective function that is minimized by II, and $OPT(x)$ be the optimum solution cost for instance x of II (i.e., $OPT(x)$ is the minimum value (over all feasible solutions s of II) of $f(s)$). Given a minimization problem II and an algorithm A for II, we say that A is an approximation algorithm with approximation guarantee (or performance guarantee, or performance ratio) $\delta > 1$ if, for every instance x of II, the resulting solution s returned by A on instance x has value $f(s) \leq \delta OPT(x)$. A similar definition applies when II is a maximization problem, but in this case $0 < \delta < 1$ and we want $f(s) \geq \delta OPT(x)$.

Approximation algorithms have been studied since the 1970s (9,10). In the last few years, however, the area has received more attention due to great advances in the understanding of approximation complexity. Other recent advances include the use of mathematical programming techniques (linear programming, semidefinite programming) to study the quality of relaxations for some problems and determining approximation guarantees.

A simple example of approximation algorithm was proposed by Christofides (10) for the *traveling salesman problem* (see Graphs section). The TSP is known to be strongly NP-hard, and therefore no polynomial time exact algorithm is known for solving it. It is also known that the problem cannot be approximated in general. However, if the distances considered are Euclidean, therefore obeying the triangular inequality, we can use Christofides' algorithm to find an approximate solution in the following way:

```
1. Algorithm Christofides
2. Input: Graph $G = (V, E)$
3. Find a minimum spanning tree $T$ connecting $V(G)$
4. Double each edge in $T$, resulting in $T'$
5. Find a circuit $C$ in $T'$ starting from any node
6. Shortcut $C$ whenever a node is repeated (i.e., substitute $(a, b, c) \subset C$ for $(a, c)$, whenever $b$ was visited previously)
7. return $C$
```

Theorem 2. *Christofides' algorithm has an approximation guarantee of* 2.

Proof. Just notice that the cost of a minimum spanning tree is a lower bound of the cost of the optimum solution, because, given a solution to TSP, we can find a minimum spanning tree by removing one of its edges. Thus, doubling the edges of T will only multiply the bound by two. Then, the shortcuting operation in line 6 is guaranteed not to increase the cost of the solution, because of the triangular inequality satisfied by G. □

The algorithm above shows that simple strategies may lead to good guarantees of approximation. However, it is known that many problems have no constant approximation guarantee, such as the SET COVER problem, unless $P = NP$ (a large list of impossibility results for approximation is given in Ref. (11)). More elaborate algorithms are based on linear programming (LP) duality and semidefinite programming. These two mathematical programming techniques are frequently useful to provide lower bounds for the solutions obtained by combinatorial algorithms.

Randomized Algorithms

Randomized algorithms correspond to a different way of looking at the task of solving a problem using computers. The usual way of thinking about algorithms is to design a set of steps that will correctly solve the problem whenever the input is given correctly, within a pre-established time, which, however, may not be the simplest or even the more effective way of solving a problem. We might consider algorithms that give the right solution for the problem with some known probability. We can also consider algorithms that always provide the right solution, but with a running time that as known just through its probabilistic distribution. Such algorithms are called *randomized algorithms*.

Randomized algorithms are usually described as algorithms that use “coin tossing” (i.e., a randomizer) or a random number generator. There are two major types of randomized algorithms: *Las Vegas* and *Monte Carlo*.

A *Las Vegas* algorithm is defined as a randomized algorithm, which always provides the correct answer but whose running time is a random variable. A *Monte Carlo* algorithm is a randomized algorithm that always has a predetermined running time (which, of course, may depend on the input size) but whose output is correct with high probability. If n is the input size of the algorithm, then by the *high probability* we imply a probability, which is equal to or greater than $1 - n^{-\alpha}$ for some fixed $\alpha \geq 1$.

Next we describe a simple example of a Monte-Carlo algorithm. Suppose we are given an array $A = a_1, \ldots, a_n$ of n numbers, where all elements are distinct and n is even. The task is to find an element of A greater than the median M of the given array.

In the first step of the algorithm, we randomly choose $\alpha \log n$ elements of the input array A. Let us denote this subset by $\tilde{A}$. Thus, $|\tilde{A}| = \alpha \log n$ and $\tilde{A} \subset A$. Then, we find the largest number from the selected subset. Obviously, we can do it in $\Theta(\log n)$ time. We argue that this element is the correct answer with high probability. The algorithm fails if all elements of the randomly selected subset are less than or equal to the median M. As elements of $\tilde{A}$ are randomly selected, it is easy to observe that any element of $\tilde{A}$ is less than or equal to the median M with probability 1/2. Therefore, the probability of failure is $P_f = (1/2)^{\alpha \log n} = n^{-\alpha}$. In summary, we can conclude that if the size of the subset $\tilde{A}$ satisfies $|\tilde{A}| \geq \alpha \log n$, then the largest element of $\tilde{A}$ is a correct result with probability at least $1 - n^{-\alpha}$. Applying the randomized algorithm, we solve the problem in $\Theta(\log n)$ time with high probability.

Parallel Algorithms

Parallel algorithms provide methods to harness the power of multiple processors working together to solve a problem. The idea of using parallel computing is natural and has been explored in multiple ways. Theoretical models for parallel algorithms have been developed, including the shared memory and message passing (MP) parallel systems.

The PRAM (Parallel Random Access Machine) is a simple model in which multiple processors access a shared memory pool. Each read or write operation is executed asynchronously by the processors. Different processors collaborate to solve a problem by writing information that will be subsequently read by others. The PRAM is a popular model that captures the basic features of parallel systems, and there is a large number of algorithms developed for this theoretical model. However, the PRAM is not a realistic model: Real systems have difficulty in guaranteeing concurrent access to memory shared by several processors.

Variations of the main model have been proposed to overcome the limitations of PRAM. Such models have additional features, including different memory access policies. For example, the EREW (exclusive read, exclusive write) allows only one processor to access the memory for read and write operations. The CREW model (concurrent read, exclusive write) allows that multiple processors read the memory simultaneously, whereas only one processor can write to memory at each time.

MP is another basic model that is extensively used in parallel algorithms. In MP, all information shared by processors is sent via messages; memory is local and accessed by a single processor. Thus, messages are the only mechanism of cooperation in MP. A major result in parallel programming is that the two mechanisms discussed (shared memory and MP) are equivalent in computational power.

The success of a parallel algorithm is measured by the amount of work that it can evenly share among processors. Therefore, we need to quantify the *speedup* of a parallel algorithm, given the size of the problem, the time $T_S(n)$ necessary to solve the problem in a sequential computer, and the time $T_P(n, p)$ needed by the parallel algorithm with p processors. The speedup $\phi(n, p)$ can be computed as $\phi(n, p) = T_S(n)/T_P(n, p)$, and the *asymptotic speedup* $\phi_\infty(p)$ is the value of $\phi(n, p)$ when n approaches infinity (i.e., $\phi_\infty(p) = \lim_{n \to \infty} \phi(n, p)$). The desired situation is that all processors can contribute to evenly reduce the computational time. In this case, $\phi(n, p) = \mathcal{O}(p)$, and we say that the algorithm has *linear speedup*.

We show a simple example of parallel algorithm. In the *prefix computation* problem, a list $L = \{l_1, \ldots, l_n\}$ of numbers is given, and the objective is to compute the sequence of values $\sum_{i=1}^{j} l_i$ for $j \in \{1, \ldots, n\}$. The following algorithm uses n processors to compute the parallel prefix of n numbers in $\mathcal{O}(\log n)$ time.

```
Algorithm Parallel-prefix
Input: $L = \{l_1, \ldots, l_n\}$
for $k \leftarrow 0$ to $\lfloor \log n \rfloor - 1$ do
    Execute the following instructions in parallel:
    for $j \leftarrow 2^k + 1$ to $n$ do
        $l_j \leftarrow l_{j-2^k} + l_j$
    end
end
```

As a sequential algorithm for this problem has complexity $\Theta(n)$, the speedup of using the parallel algorithm is $\phi(n, n) = \Theta(n/\log n)$.

Heuristics and Metaheuristics

One of the most popular techniques for solving computationally hard problems is to apply the so-called *heuristic* approaches. A heuristic is an ad hoc algorithm that applies a set of rules to create or improve solutions to an NP-hard problem. Heuristics give a practical way of finding solutions that can be satisfactory for most purposes. The main distinctive features of these type of algorithms are fast execution time and the goal of finding good-quality solutions without necessarily providing any guarantee of quality. Heuristics are particularly useful for solving in practice hard large-scale problems for which more exact solution methods are not available.

As an example of a simple heuristic, we can consider a *local search* algorithm. Suppose we need to solve a combinatorial optimization problem, where we minimize an objective function $f(x)$, and the decision variable x is a vector of n binary variables (i.e., $x_i \in \{0, 1\}$ for $i \in \{1, \ldots, n\}$). For each problem, we define a *neighborhood* $N(x)$ of a feasible solution x as the set of feasible solutions x' that can be created by a well-defined modification of x. For example, let $x \in \{0, 1\}^n$ be a feasible solution, then $N(x)$ can be defined as the set of solutions x^k such that

$$x_i^k = \begin{cases} 1 - x_i & \text{if } i = k \\ x_i & \text{otherwise} \end{cases}$$

for $k \in \{1, \ldots, n\}$. A point $x \in \{0, 1\}^n$ is called a *local optimum* if it does not have a neighbor whose objective value is strictly better than $f(x)$. In our example, which we suppose is a minimization problem, it means that if x is a local optimum, then $f(x) \leq f(x^k)$ for all $k \in \{1, \ldots, n\}$. In this case, if we can generate a feasible solution x, then we can check if x is locally optimal in time $\mathcal{O}(n)$ (notice that other neighborhoods may lead to different search times). Let $\tilde{x} \in N(x)$, with $f(\tilde{x}) < f(x)$. Assigning $x \leftarrow \tilde{x}$, we can repeat the afore-mentioned procedure that searches for a locally optimal solution while the optimality criterion is reached. For many problems, local search-based approaches similar to the one described above proved to be very successful.

A *metaheuristic* is a family of heuristics that can be applied to find good solutions for difficult problems. The main difference between metaheuristics and heuristics is that, while a heuristic is an ad hoc procedure, a metaheuristic must follow a well-defined sequence of steps and use some specified data structures. However, the exact implementation of some of the steps in a metaheuristic may be customized, according to the target problem. Therefore, a concrete implementation of a metaheuristic provides a heuristic for a specific problem (hence, the use of the prefix "meta"). Elements of heuristics such as local search or greedy methods are usually combined in metaheuristics in order to find solutions of better quality.

Examples of metaheuristic methods include simulated annealing, genetic algorithms, tabu search, GRASP, path relinking, reactive search, ant colony optimization, and variable neighborhood search.

BIBLIOGRAPHIC NOTES

Algorithms and data structures have a vast literature, and we provide only some of the most important references. Popular books in algorithms and data structures include the ones by Cormen et al. (12), Horowitz et al. (13), Sedgewick (14), and Knuth (15). Networks and graph algorithms are covered in depth in Ahuja et al. (16). Books on data structures include the classics by Aho et al. (17) and Tarjan (18).

NP-hard problems have been discussed in several books, but the best-known reference is the compendium by Garey and Johnson (8). A standard reference on randomized algorithms is Ref. (19). For more information on parallel algorithms, and especially PRAM algorithms, see the book by Jaja (20). Good reviews of approximation algorithms are presented in the book edited by Hochbaum (21), and by Vazirani (22). For a more detailed discussion of different metaheuristics and related topics, the reader is referred to Ref. (23).

Information about the complexity of numerical optimization problems is available in the paper by Pardalos and Vavasis (24) and the books listed in Refs. (25) and (26).

BIBLIOGRAPHY

1. E. D. Demaine, Cache-oblivious algorithms and data structures, in *Lecture Notes from the EEF Summer School on Massive Data Sets*, Lecture Notes in Computer Science. New York: Springer-Verlag, 2002.
2. M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, Cache-oblivious algorithms, in *Proc. of 40th Annual Symposium on Foundations of Computer Science*, New York, 1999.
3. L. Arge, The buffer tree: A new technique for optimal I/O algorithms, in *Proceedings of Fourth Workshop on Algorithms and Data Structures (WADS), volume 955 of Lecture Notes in Computer Science*. New York: Springer-Verlag, 1995, pp. 334–345.
4. L. Arge, D. Vengroff, and J. Vitter, External-memory algorithms for processing line segments in geographic information systems, in *Proceedings of the Third European Symposium on Algorithms, volume 979 of Lecture Notes in Computer Science*. New York: Springer-Verlag, 1995, pp. 295–310.
5. J. Abello, P. Pardalos, and M. Resende (eds.), *Handbook of Massive Data Sets*. Dordrecht, the Netherlands: Kluwer Academic Publishers, 2002.
6. C. H. Papadimitriou, *Computational Complexity*. Reading, MA: Addison-Wesley, 1994.
7. S. Cook, The complexity of theorem-proving procedures, in *Proc. 3rd Ann. ACM Symp. on Theory of Computing*. New York: Association for Computing Machinery, 1971, pp. 151–158.
8. M. R. Garey and D. S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Company, 1979.
9. D. Johnson, Approximation algorithms for combinatorial problems, *J. Comp. Sys. Sci.*, **9**(3): 256–278, 1974.
10. N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, in J. F. Traub (ed.), *Symposium*

on New Directions and Recent Results in Algorithms and Complexity. New York: Academic Press, 1976, p. 441.

11. S. Arora and C. Lund, Hardness of approximations, in D. Hochbaum (ed.), *Approximation Algorithms for NP-hard Problems*. Boston, MA: PWS Publishing, 1996.
12. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., Cambridge, MA: MIT Press, 2001.
13. E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms*. New York: Computer Science Press, 1998.
14. R. Sedgewick, *Algorithms*. Reading, MA: Addison-Wesley, 1983.
15. D. Knuth, *The Art of Computer Programming – Fundamental Algorithms*. Reading, MA: Addison-Wesley, 1997.
16. R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
17. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Computer Science and Information Processing. Reading, MA: Addison-Wesley, 1982.
18. R. E. Tarjan, *Data Structures and Network Algorithms*. Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1983.
19. R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge: Cambridge University Press, 1995.
20. J. JaJa, *An Introduction to Parallel Algorithms*. Reading, MA: Addison-Wesley, 1992.
21. D. Hochbaum (ed.), *Approximation Algorithms for NP-hard Problems*. Boston, MA: PWS Publishing, 1996.
22. V. Vazirani, *Approximation Algorithms*. New York: Springer-Verlag, 2001.
23. M. Resende and J. de Sousa (eds.), *Metaheuristics: Computer Decision-Making*. Dordrecht, the Netherlands: Kluwer Academic Publishers, 2004.
24. P. M. Pardalos and S. Vavasis, Complexity issues in numerical optimization, *Math. Program. B*, **57**(2): 1992.
25. P. Pardalos (ed.), *Complexity in Numerical Optimization*. Singapore: World Scientific, 1993.
26. P. Pardalos (ed.), *Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems*. Dordrecht, the Netherlands: Kluwer Academic Publishers, 2000.

CARLOS A.S. OLIVEIRA
Oklahoma State University
Stillwater, Oklahoma

PANOS M. PARDALOS
OLEG A. PROKOPYEV
University of Florida
Gainesville, Florida

DATA WAREHOUSE

INTRODUCTION

For a few decades, the role played by database technology in companies and enterprises has only been that of storing operational data, that is data generated by daily, routine operations carried out within business processes (such as selling, purchasing, and billing). On the other hand, managers need to access quickly and reliably the strategic information that supports decision making. Such information is extracted mainly from the vast amount of operational data stored in corporate databases, through a complex selection and summarization process.

Very quickly, the exponential growth in data volumes made computers the only suitable support for the decisional process run by managers. Thus, starting from the late 1980s, the role of databases began to change, which led to the rise of *decision support systems* that were meant as the suite of tools and techniques capable of extracting relevant information from a set of electronically recorded data. Among decision support systems, data warehousing systems are probably those that captured the most attention from both the industrial and the academic world.

A typical decision-making scenario is that of a large enterprise, with several branches, whose managers wish to quantify and evaluate the contribution given from each of them to the global commercial return of the enterprise. Because elemental commercial data are stored in the enterprise database, the traditional approach taken by the manager consists in asking the database administrators to write an ad hoc query that can aggregate properly the available data to produce the result. Unfortunately, writing such a query is commonly very difficult, because different, heterogeneous data sources will be involved. In addition, the query will probably take a very long time to be executed, because it will involve a huge volume of data, and it will run together with the application queries that are part of the operational workload of the enterprise. Eventually, the manager will get on his desk a report in the form of a either summary table, a diagram, or a spreadsheet, on which he will base his decision.

This approach leads to a useless waste of time and resources, and often it produces poor results. By the way, mixing these ad hoc, analytical queries with the operational ones required by the daily routine causes the system to slow down, which makes all users unhappy. Thus, the core idea of data warehousing is to separate analytical queries, which are commonly called OLAP *(On-Line Analytical Processing)* queries, from the operational ones, called OLTP *(On-Line Transactional Processing)* queries, by building a new information repository that integrates the elemental data coming from different sources, organizes them into an appropriate structure, and makes them available for analyses and evaluations aimed at planning and decision making.

Among the areas where data warehousing technologies are employed successfully, we mention but a few: trade, manufacturing, financial services, telecommunications, and health care. On the other hand, the applications of data warehousing are not restricted to enterprises: They also range from epidemiology to demography, from natural sciences to didactics. The common trait for all these fields is the need for tools that enable the user to obtain summary information easily and quickly out of a vast amount of data, to use it for studying a phenomenon and discovering significant trends and correlations—in short, for acquiring useful knowledge for decision support.

BASIC DEFINITIONS

A *data warehousing system* can be defined as a collection of methods, techniques, and tools that support the so-called *knowledge worker* (one who works primarily with information or develops and uses knowledge in the workplace: for instance, a corporate manager or a data analyst) in decision making by transforming data into information. The main features of data warehousing can be summarized as follows:

- Easy access to nonskilled computer users.
- Data integration based on a model of the enterprise.
- Flexible querying capabilities to take advantage of the information assets.
- Synthesis, to enable targeted and effective analysis.
- Multidimensional representation to give the user an intuitive and handy view of information.
- Correctness, completeness, and freshness of information.

At the core of this process, the *data warehouse* is a repository that responds to the above requirements. According to the classic definition by Bill Inmon (see Further Reading), a data warehouse is a collection of data that exhibits the following characteristics:

1. Subject-oriented, which means that all the data items related to the same business object are connected.
2. Time-variant, which means that the history of business is tracked and recorded to enable temporal reports.
3. Nonvolatile, which means that data are read-only and never updated or deleted.
4. Integrated, which means that data from different enterprise applications are collected and made consistent.

Although operational data commonly span a limited time interval, because most business transactions only involve recent data, the data warehouse must support

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

analyses that cover some years. Thus, the data warehouse is refreshed periodically starting from operational data. According to a common metaphor, we can imagine that photographs of operational data are periodically taken; the sequence of photos is then stored in the data warehouse, where a sort of movie is generated that depicts the history of business up to the current time.

Because in principle data are never deleted, and refreshes are made when the system is offline, a data warehouse can be considered basically as a read-only database. This feature, together with the importance given to achieving good querying performances, has two main consequences. First, the database management systems (DBMSs) used to manage the data warehouse do not need sophisticated techniques for supporting transactions. Second, the design techniques used for data warehouses are completely different from those adopted for operational databases.

As mentioned, another relevant difference between operational databases and data warehouses is related to the types of queries supported. OLTP queries on operational databases typically read and write a relatively small number of records from some tables related by simple relationships (e.g., search for customers' data to insert new orders). Conversely, OLAP queries on data warehouses commonly read a huge number of records to compute a few pieces of summary information. Most importantly, although the OLTP workload is "frozen" within applications and only occasionally ad hoc queries are formulated, the OLAP workload is intrinsically interactive and dynamic.

ARCHITECTURES

To preserve the separation between transactional and analytical processing, most data warehousing architectures are based on at least two data levels: the data sources and the data warehouse.

Data sources are heterogeneous; they may be part of the corporate information system (operational databases, legacy systems, spreadsheets, flat files, etc.). or even reside outside the company (Web databases, streams, etc.). These data are extracted, cleaned, completed, validated, integrated into a single schema, and loaded into the data warehouse by the so-called *ETL (Extraction, Transformation, and Loading)* tools.

The data warehouse is the centralized repository for the integrated information. Here, different from the sources, data are stored in multidimensional form, and their structure is optimized to guarantee good performance for OLAP queries. In practice, most often, the data warehouse is replaced physically by a set of *data marts* that include the portion of information that is relevant to a specific area of business, division of the enterprise, and category of users. Note the presence of a *metadata repository* that contains the "data about data," for example, a description of the logical organization of data within the sources, the data warehouse, and the data marts.

Finally, the information in the data warehouse is accessed by users by means of different types of tools: reporting tools, OLAP tools, data-mining tools, and what-if analysis tools.

Some architectures include an additional level called the *reconciled level* or *operational data-store*. It materializes the operational data obtained by extracting and cleaning source data: Thus, it contains integrated, consistent, correct, detailed, and current data. These reconciled data are then used to feed the data warehouse directly. Although the reconciled level introduces a significant redundancy, it also bears some notable benefits. In fact, it defines a reference data model for the whole company, and at the same time, it introduces a clear separation between the issues related to data extraction, cleaning and integration and those related to data warehouse loading. Remarkably, in some cases, the reconciled level is also used to better accomplish some operational tasks (such as producing daily reports that cannot be prepared satisfactorily using the corporate applications).

In the practice, these ingredients are blended differently to give origin to the five basic architectures commonly recognized in the literature:

- Independent data marts architecture
- Bus architecture
- Hub-and-spoke architecture
- Centralized data warehouse architecture
- Federated architecture

In the *independent data mart architecture*, different data marts are designed separately and built in a nonintegrated fashion (Fig. 1). This architecture, although sometimes initially adopted in the absence of a strong sponsorship toward an enterprise-wide warehousing project or when the organizational divisions that make up the company are coupled loosely, tends to be soon replaced by other architectures that better achieve data integration and cross-reporting.

The *bus architecture* is apparently similar to the previous one, with one important difference: A basic set of conformed dimension and facts, derived by a careful analysis of the main enterprise processes, is adopted and shared as a common design guideline to ensure logical integration of data marts and an enterprise-wide view of information.

In the *hub-and-spoke architecture*, much attention is given to scalability and extensibility and to achieving an enterprise-wide view of information. Atomic, normalized data are stored in a reconciled level that feeds a set of data marts containing summarized data in multidimensional form (Fig. 2). Users mainly access the data marts, but they occasionally may query the reconciled level.

The *centralized architecture* can be viewed as a particular implementation of the hub-and-spoke architecture where the reconciled level and the data marts are collapsed into a single physical repository.

Finally, *the federated architecture* is sometimes adopted in contexts where preexisting data warehouses/data marts are to be integrated noninvasively to provide a single, cross-organization decision support environment (e.g., in the case of mergers and acquisitions). Each data warehouse/data mart is either virtually or physically integrated with the

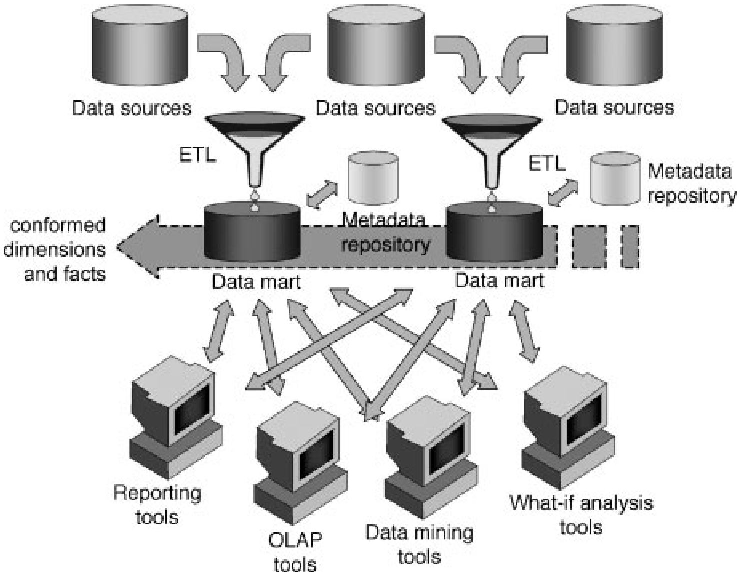


Figure 1. Independent data marts and bus architectures (without and with conformed dimensions and facts).

others by leaning on a variety of advanced techniques such as distributed querying, ontologies, and metadata interoperability.

ACCESSING THE DATA WAREHOUSE

This section discusses how users can exploit information stored in the data warehouse for decision making. In the following subsection, after introducing the particular features of the multidimensional model, we will survey the two main approaches for analyzing information: reporting and OLAP.

The Multidimensional Model

The reasons why the multidimensional model is adopted universally as the paradigm for representing data in data warehouses are its simplicity, its suitability for business analyses, and its intuitiveness for nonskilled computer users, which are also caused by the widespread use of spreadsheets as tools for individual productivity. Unfortunately, although some attempts have been made in the literature to formalize the multidimensional model (e.g., Ref. 1), none of them has emerged as a standard so far.

The multidimensional model originates from the observation that the decisional process is ruled by the *facts* of the business world, such as sales, shipments, bank transactions, and purchases. The occurrences of a fact correspond to *events* that occur dynamically: For example, every sale or shipment made is an event. For each fact, it is important to know the values of a set of *measures* that quantitatively describe the events: the revenue of a sale, the quantity shipped, the amount of a bank transaction, and the discount on a purchase.

The events that happen in the enterprise world are obviously too many to be analyzed one by one. Thus, to make them easily selectable and groupable, we imagine arranging them within an n-dimensional space whose axes, called *dimensions* of analysis, define different perspectives for their identification. Dimensions commonly are discrete, alphanumeric attributes that determine the minimum granularity for analyzing facts. For instance, the sales in a chain of stores can be represented within a three-dimensional space whose dimensions are the products, the stores, and the dates.

The concepts of dimension gave birth to the well-known *cube* metaphor for representing multidimensional data. According to this metaphor, events correspond to cells of a cube whose edges represents the dimensions of analysis. A cell of the cube is determined uniquely by assigning a value to every dimension, and it contains a value for each measure. Figure 3 shows an intuitive graphical representation of a cube centered on the sale fact. The dimensions are **product, store**, and **date**. An event corresponds to the selling of a given product in a given store on a given day, and it is described by two measures: the quantity sold and the revenue. The figure emphasizes that the cube is sparse, i.e., that several events did not happen at all: Obviously, not all products are sold every day in every store.

Normally, each dimension is structured into a *hierarchy* of *dimension levels* (sometimes called *roll-up hierarchy*) that group its values in different ways. For instance, products may be grouped according to their type and their brand, and types may be grouped additionally into categories. Stores are grouped into cities, which in turn are grouped into regions and nations. Dates are grouped into months and years. On top of each hierarchy, a final level exists that groups together all possible values of a hierarchy (all products, all stores, and all dates). Each dimension level may be described even more by one or more *descriptive attributes* (e.g., a product may be described by its name, its color, and its weight).

A brief mention to some alternative terminology used either in the literature or in the commercial tools is useful.

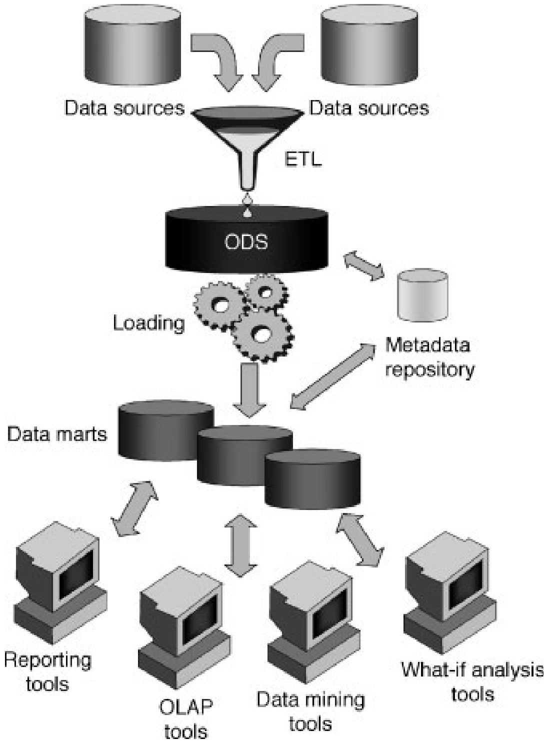


Figure 2. Hub-and-spoke architecture; ODS stands for operational data store.

Although with the term *dimension* we refer to the attribute that determines the minimum fact granularity, sometimes the whole hierarchies are named as dimensions. Measures are sometimes called *variables, metrics, categories, properties*, or *indicators*. Finally, dimension levels are sometimes called *parameters* or *attributes*.

We now observe that the cube cells and the data they contain, although summarizing the elemental data stored within operational sources, are still very difficult to analyze because of their huge number. Two basic techniques are used, possibly together, to reduce the quantity of data and thus obtain useful information: *restriction* and *aggregation*. For both, hierarchies play a fundamental role because they determine how events may be aggregated and selected.

Restricting data means cutting out a portion of the cube to limit the scope of analysis. The simplest form of restriction is *slicing*, where the cube dimensionality is reduced by focusing on one single value for one or more dimensions. For instance, as depicted in Fig. 4, by deciding that only sales of store "S-Mart" are of interest, the decision maker actually cuts a slice of the cube obtaining a two-dimensional subcube. *Dicing* is a generalization of slicing in which a subcube is determined by posing Boolean conditions on hierarchy levels. For instance, the user may be interested in sales of products of type "Hi-Fi" for the stores in Rome during the days of January 2007 (see Fig. 4).

Although restriction is used widely, aggregation plays the most relevant role in analyzing multidimensional data. In fact, most often users are not interested in analyzing events at the maximum level of detail. For instance, it may be interesting to analyze sale events not on a daily basis but by month. In the cube metaphor, this process means grouping, for each product and each store, all cells corresponding to the days of the same month into one macro-cell. In the aggregated cube obtained, each macro-cell represents a synthesis of the data stored in the cells it aggregates: in our example, the total number of items sold in each month and the total monthly revenue, which are calculated by summing the values of **quantity** and **revenue** through the corresponding cells. Eventually, by aggregating along the time hierarchy, an aggregated cube is obtained in which each macro-cell represents the total sales over the whole time period for each product and store. Aggregation can also be operated along two or more hierarchies. For instance, as shown in Fig. 5, sales can be aggregated by month, product type, and city.

Noticeably, not every measure can be aggregated consistently along all dimensions using the sum operator. In some cases, other operators (such as average or minimum) can be used instead, whereas in other cases, aggregation is not possible at all. For details on the two related problems of *additivity* and *summarizability*, the reader is referred to Ref. 2.

Reporting

Reporting is oriented to users who need to access periodically information structured in a fixed way. For instance, a hospital must send monthly reports of the costs of patient stays to a regional office. These reports always have the same form, so the designer can write the query that generates the report and "freeze" it within an application so that it can be executed at the users' needs.

A report is associated with a query and a presentation. The query typically entails selecting and aggregating multidimensional data stored in one or more facts. The presentation can be in tabular or graphical form (a diagram, a histogram, a cake, etc.). Most reporting tools also allow for automatically distributing periodic reports to interested users by e-mail on a subscription basis or for posting reports in the corporate intranet server for downloading.

OLAP

OLAP, which is probably the best known technique for querying data warehouses, enables users to explore interactively and analyze information based on the multidimensional model. While the users of reporting tools essentially play a passive role, OLAP users can define actively a complex analysis session where each step taken follows from the results obtained at previous steps. The impromptu character of OLAP sessions, the deep knowledge of data required, the complexity of the possible queries, and the orientation toward users not skilled in computer science maximize the importance of the employed tool, whose interface necessarily has to exhibit excellent features of flexibility and friendliness.

An OLAP session consists in a "navigation path" that reflects the course of analysis of one or more facts from different points of view and at different detail levels. Such a path is realized into a sequence of queries, with each differentially expressed with reference to the previous

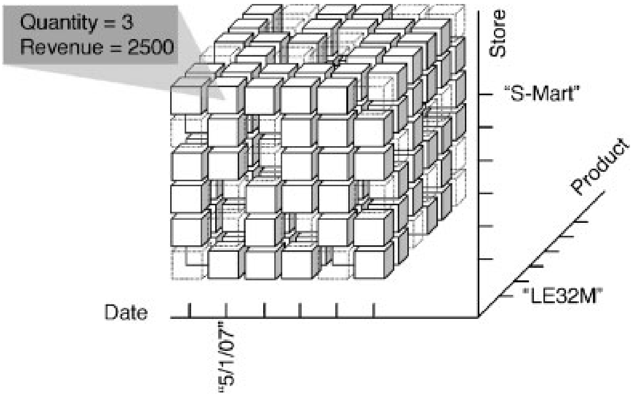


Figure 3. The three-dimensional cube that models the sales in a chain of shops. In the S-Mart store, on 5/1/2007, three LE32M TVs were sold, for a total revenue of $2500.

query. Query results are multidimensional; like for reporting, OLAP tools typically represent data in either tabular or graphical form.

Each step in the analysis session is marked by the application of an *OLAP operator* that transforms the previous query into a new one. The most common OLAP operators are as follows:

- *Roll-up*, which aggregates data even more (e.g., from sales by product, nation, and month to sales by category, nation, and year).
- *Drill-down*, which adds detail to data (e.g., from sales by category, nation, and year to sales by category, city, and year).
- *Slice-and-dice*, which selects data by fixing values or intervals for one or more dimensions (e.g., sales of products of type "Hi-Fi" for stores in Italy).
- *Pivoting*, which changes the way of visualizing the results by rotating the cube (e.g., swaps rows with columns).
- *Drill-across*, which joins two or more correlated cubes to compare their data (e.g., join the sales and the promotions cubes to compare revenues with discounts).

We close this section by observing that, in several applications, much use is made of an intermediate approach commonly called *semistatic reporting*, in which only a reduced set of OLAP navigation paths are enabled to avoid obtaining inconsistent or wrong results by incorrectly using aggregation, while allowing for some flexibility in manipulating data.

IMPLEMENTATIONS OF THE MULTIDIMENSIONAL MODEL

Two main approaches exist for implementing a data warehouse: *ROLAP*, which stands for *relational OLAP*, and *MOLAP*, which stands for *multidimensional OLAP*. Recently a third, intermediate approach has been adopted in some commercial tools: *HOLAP*, that is, *hybrid OLAP*.

Relational OLAP

On a ROLAP platform, the relational technology is employed to store data in multidimensional form. This approach is motivated by the huge research work made on the relational model, by the widespread knowledge of relational databases and their administration, and by the excellent level of performance and flexibility achieved by relational DBMSs. Of course, because the expressiveness of the relational model does not include the basic concepts of the multidimensional model, it is necessary to adopt specific schema structures that allow the multidimensional model to be mapped onto the relational model. Two main such structures are commonly adopted: the *star schema* and the *snowflake schema*.

The star schema is a relational schema composed of a set of relations called *dimension tables* and one relation called a *fact table*. Each dimension table models a hierarchy; it includes a surrogate key (i.e., a unique progressive number generated by the DBMS) and one column for each level and descriptive attribute of the hierarchy. The fact table includes a set of foreign keys, one that references each dimension table, which together define the primary key, plus one column for each measure. Figure 6 shows a star schema for the sales example. Noticeably, dimension tables are denormalized (they are not in the third normal form); this is aimed at reducing the number of relational joins to be computed when executing OLAP queries, so as to improve performance.

A snowflake schema is a star schema in which one or more dimension tables have been partially or totally normalized to reduce redundancy. Thus, a dimension table can be split into one *primary* dimension table (whose surrogate key is references by the fact table) and one or more

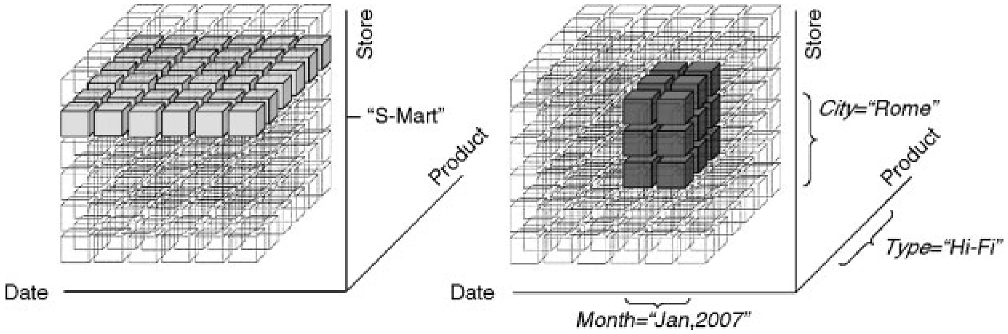


Figure 4. Slicing (left) and dicing (right) on the sales cube.

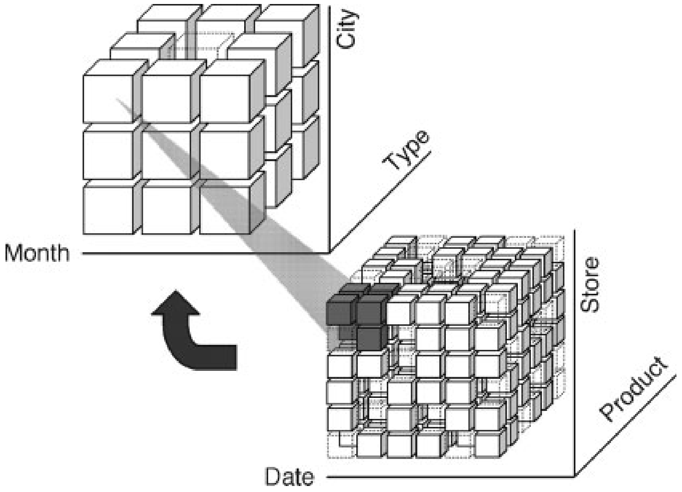


Figure 5. Aggregation on the sales cube.

secondary dimension tables (each including a surrogate key and referencing the key of another dimension table). Figure 7 shows an example for the sale schema, in which the product dimension has been normalized partially.

Multidimensional OLAP

Differently from ROLAP, a MOLAP system is based on a native logical model that directly supports multidimensional data and operations. Data are stored physically into multidimensional arrays, and positional techniques are used to access them.

The great advantage of MOLAP platforms is that OLAP queries can be executed with optimal performances, without resorting to complex and costly join operations. On the other hand, they fall short when dealing with large volumes of data, mainly because of the problem of sparseness: In fact, when a large percentage of the cube cells are empty, a lot of memory space is wasted uselessly unless ad hoc compression techniques are adopted.

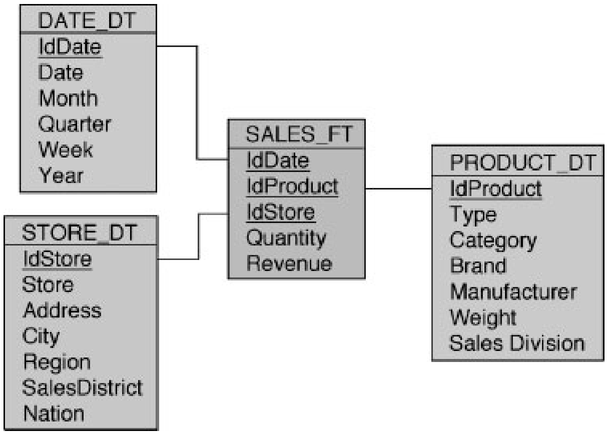


Figure 6. Star schema for the sales example (primary keys are underlined).

Hybrid OLAP

HOLAP can be viewed as an intermediate approach between ROLAP and MOLAP, because it tries to put together their advantages into a single platform. Two basic strategies are pursued in commercial tools to achieve this goal. In the first strategy, detailed data are stored in a relational database, whereas a set of useful preaggregates are stored on proprietary multidimensional structures. In the second strategy, cubes are partitioned into dense and sparse subcubes, with the former being stored in multidimensional form, and the latter in relational form.

DESIGN TECHNIQUES

Despite the basic role played by a well-structured methodological framework in ensuring that the data warehouse designed fully meets the user expectations, a very few *comprehensive* design methods have been devised so far (e.g., Refs. 3 and 4). None of them has emerged as a standard, but all agree on one point: A bottom-up approach is preferable to a top-down approach, because it significantly reduces the risk of project failure. While in a top-down approach the data warehouse is planned and designed initially in its entirety, in a bottom-up approach, it is built incrementally by designing and prototyping one data mart at a time, starting from the one that plays the most strategic business role. In general terms, the macro-phases for designing a data warehouse can be stated as follows:

- *Planning*, based on a feasibility study that assesses the project goals, estimates the system borders and size, evaluates costs and benefits, and analyzes risks and users' expectations.
- *Infrastructure design*, aimed at comparing the different architectural solutions, at surveying the available technologies and tools, and at preparing a draft design of the whole system.
- *Data mart development*, which iteratively designs, develops, tests and deploys each data mart and the related applications.

As concerns the design of each data mart, the methodology proposed in Ref. 3 encompasses eight closely related, but not necessarily strictly sequential, phases:

1. *Data source analysis*. The source schemata are analyzed and integrated to produce a reconciled schema describing the available operational data.
2. *Requirement analysis*. Business users are interviewed to understand and collect their goals and needs, so as to generate requirement glossaries and a preliminary specification of the core workload.
3. *Conceptual design*. Starting from the user requirements and from the reconciled schema, a conceptual schema that describes the data mart in an implementation-independent manner is derived.
4. *Schema validation*. The preliminary workload is better specified and tested against the conceptual schema to validate it.

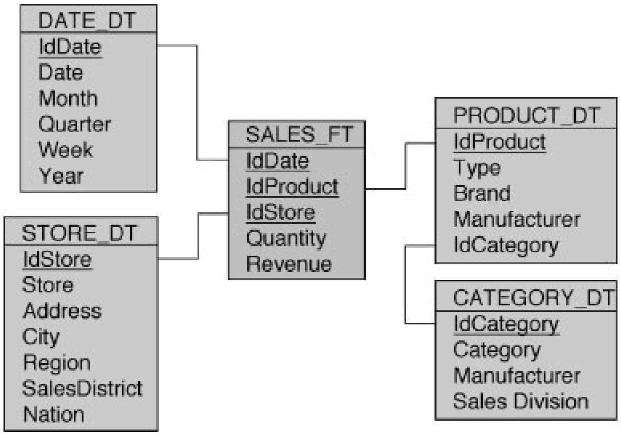


Figure 7. Snowflake schema for the sales example.

5. *Logical design*. The conceptual schema is translated into a logical schema according to the target logical model (relational or multidimensional), considering the expected workload and possible additional constraints related to space occupation and querying performances.
6. *ETL design*. The ETL procedures used to feed the data mart starting from the data sources via the reconciled level are designed.
7. *Physical design*. Physical optimization of the logical schema is done, depending on the specific characteristic of the platform chosen for implementing the data mart.
8. *Implementation*. The physical schema of the data mart is deployed, ETL procedures are implemented, and the applications for data analysis are built and tested.

Several techniques for supporting single phases of design have been proposed in the literature; a brief survey of the most relevant approaches is reported in the following subsections.

Data Source Analysis

A huge literature about schema integration has been accumulating over the last two decades. Integration methodologies have been proposed (e.g., Ref. 5), together with formalisms to code the relevant information (e.g., Ref. 6). However, the integration tools developed so far [such as TSIMMIS (7) and MOMIS (8)] should still be considered research prototypes rather than industrial tools, with the notable exception of Clio (9), which is supported by IBM.

Requirement Analysis

A careful requirement analysis is one of the keys to reduce dramatically the risk of failure for warehousing projects. From this point of view, the approaches to data warehouse design usually are classified in two categories:

- *Supply-driven* (or *data-driven*) approaches design the data warehouse starting from a detailed analysis of the data sources (e.g., Ref. 10). User requirements impact design by allowing the designer to select which chunks of data are relevant for the decision-making process and by determining their structuring according to the multidimensional model.
- *Demand-driven* (or *requirement-driven*) approaches start from determining the information requirements of business users (like in Ref. 11). The problem of mapping these requirements onto the available data sources is faced only *a posteriori*, by designing proper ETL routines, and possibly by accommodating data sources to accomplish the information needs.

A few *mixed* approaches were also devised (12,13), where requirement analysis and source inspection are carried out in parallel, and user requirements are exploited to reduce the complexity of conceptual design.

Conceptual Design

Although no agreement exists on a standard conceptual model for data warehouses, most authors agree on the importance of a conceptual design phase providing a high level of abstraction in describing the multidimensional schema of the data warehouse aimed at achieving independence of implementation issues. To this end, conceptual models typically rely on a graphical notation that facilitates writing, understanding, and managing conceptual schemata by designers and users.

The existing approaches may be framed into three categories: extensions to the entity-relationship model (e.g, Ref. 14), extensions to UML (e.g., Ref. 15), and ad hoc models (e.g., Ref. 16). Although all models have the same core expressivity, in that they all allow the basic concepts of the multidimensional model to be represented graphically, they significantly differ as to the possibility of representing more advanced concepts such as irregular hierarchies, many-to-many associations, and additivity.

Logical Design

The goal of logical design is to translate a conceptual schema into a logical schema for the data mart. Although on MOLAP platforms this task is relatively simple, because the target logical model is multidimensional like the source conceptual one, on ROLAP platforms, two different models (multidimensional and relational) have to be matched. This is probably the area of data warehousing where research has focused the most during the last decade (see, for instance, Ref. 17); in particular, a lot has been written about the so-called view materialization problem.

View materialization is a well-known technique for optimizing the querying performance of data warehouses by physically materializing a set of (redundant) tables, called *views*, that store data at different aggregation levels. In the presence of materialized views, an ad hoc component of the underlying DBMS (often called *aggregate navigator*) is entrusted with the task of choosing, for each query formulated by the user, the view(s) on which the query can be answered most cheaply. Because the number of potential views is exponential in the number of hierarchy levels, materializing all views would be prohibitive. Thus, research has focused mainly on effective techniques for determining the optimal subset of views to be materialized under different kinds of constraints (e.g., Ref. 18).

Another optimization technique that is sometimes adopted to improve the querying performance is *fragmentation* (also called*partitioning* or *striping*). In particular, in *vertical* fragmentation, the fact tables are partitioned into smaller tables that contain the key and a subset of measures that are often accessed together by the workload queries (19).

ETL Design

This topic has earned some research interest only in the last few years. The focus here is to model the ETL process either from the functional, the dynamic, or the static point of view. In particular, besides techniques for conceptual design of ETL (20), some approaches are aimed at automating (21) and optimizing (22) the logical design of ETL. Although the research on ETL modeling is probably less mature than that on multidimensional modeling, it will probably have a very relevant impact on improving the overall reliability of the design process and on reducing its duration.

Physical Design

Physical design is aimed at filling the gap between the logical schema and its implementation on the specific target platform. As such, it is concerned mainly with the problem of choosing which types of indexes should be created on which columns of which tables. Like the problem of view selection, this problem has exponential complexity. A few papers on the topic can be found in the literature: For instance, Ref. 23 proposes a technique that jointly optimizes view and index selection, whereas Ref. 24 selects the optimal set of indexes for a given set of views in the presence of space constraints. The problem is made even more complex by the fact that ROLAP platforms typically offer, besides classic B-trees, other types of indexes, such as star indexes, projection indexes, and bitmap indexes (25).

Note that, although some authors consider both view selection and fragmentation as part of physical design, we prefer to include them into logical design for similarity with the design of operational databases (26).

ADVANCED TOPICS

Several other topics besides those discussed so far have been addressed by the data warehouse literature. Among them we mention:

- *Query processing*. OLAP queries are intrinsically different from OLTP queries: They are read-only queries requiring a very large amount of data, taken from a few tables, to be accessed and aggregated. In addition, DBMSs oriented to data warehousing commonly support different types of specialized indexes besides B-trees. Finally, differently from the OLTP workload, the OLAP workload is very dynamical and subject to change, and very fast response times are needed. For all these reasons, the query processing techniques required by data warehousing systems are significantly different from those traditionally implemented in relational DBMSs.
- *Security*. Among the different aspects of security, confidentiality (i.e., ensuring that users can only access the information they have privileges for) is particularly relevant in data warehousing, because business information is very sensitive. Although the classic security models developed for operational databases are used widely by data warehousing tools, the particularities of OLAP applications ask for more specific models centered on the main concepts of multidimensional modeling—facts, dimensions, and measures.
- *Evolution*. The continuous evolution of the application domains is bringing to the forefront the dynamic aspects related to describing how the information stored in the data warehouse changes over time. As concerns changes in values of hierarchy data (the so-called *slowly changing dimensions*), several approaches have been devised, and some commercial systems allow us to track changes and to query cubes effectively based on different temporal scenarios. Conversely, the problem of managing changes on the schema level has only been explored partially, and no dedicated commercial tools or restructuring methods are available to the designer yet.
- *Quality*. Because of the strategic importance of data warehouses, it is absolutely crucial to guarantee their quality (in terms of data, design, technology, business, etc.) from the early stages of a project. Although some relevant work on the quality of data has been carried out, no agreement still exists on the quality of the design process and its impact on decision making.
- *Interoperability*. The wide variety of tools and software products available on the market has lead to a broad diversity in metadata modeling for data warehouses. In practice, tools with dissimilar metadata are integrated by building complex metadata bridges, but some information is lost when translating from one form of metadata to another. Thus, a need exists for a standard definition of metadata in order to better support data warehouse interoperability and integration, which is particularly relevant in the recurrent case of mergers and acquisitions. Two industry standards developed by multivendor organizations have originated in this context: the Open Information Model (OIM) by the Meta Data Coalition (MDC) and the Common Warehouse Metamodel (CWM) by the OMG.
- *New architectures and applications*. Advanced architectures for business intelligence are emerging to support new kinds of applications, possibly involving new and more complex data types. Here we cite *spatial data warehousing, web warehousing, real-time data warehousing, distributed data warehousing, and scientific data warehousing*. Thus, it becomes necessary to adapt

and specialize the existing design and optimization techniques to cope with these new applications.

See Ref. 26 for an up-to-date survey of open research themes.

BIBLIOGRAPHY

1. H. Lenz and A. Shoshani, Summarizability in OLAP and statistical data bases, *Proc. Int. Conf. on Scientific and Statistical Database Management,* Olympia, WA, 1997, pp. 132–143.
2. R. Agrawal, A. Gupta, and S. Sarawagi, Modeling multidimensional databases, IBM Research Report, IBM Almaden Research Center, 1995.
3. M. Golfarelli and S. Rizzi, A methodological framework for data warehouse design, *Proc. Int. Workshop on Data Warehousing and OLAP,* Washington DC, 1998, pp. 3–9.
4. S. Luján-Mora and J. Trujillo, A comprehensive method for data warehouse design, *Proc. Int. Workshop on Design and Management of Data Warehouses,* Berlin, Germany, 2003, pp. 1.1–1.14.
5. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, A principled approach to data integration and reconciliation in data warehousing, *Proc. Int. Workshop on Design and Management of Data Warehouses,* Heidelberg, Germany, 1999, pp. 16.1–16.11.
6. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, Description logic framework for information integration, *Proc. Int. Conf. on Principles of Knowledge Representation and Reasoning,* Trento, Italy, 1998, pp. 2–13.
7. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, The TSIMMIS project: integration of heterogeneous information sources, *Proc. Meeting of the Inf. Processing Soc. of Japan,*Tokyo, Japan, 1994, pp. 7–18.
8. D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori, and M. Vincini, Information integration: the MOMIS project demonstration, *Proc. Int. Conf. on Very Large Data Bases,* Cairo, Egypt, 2000, pp. 611–614.
9. L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth, Clio grows up: from research prototype to industrial tool, *Proc. SIGMOD Conf.,* Baltimore, MD, 2005, pp. 805–810.
10. M. Golfarelli, D. Maio, and S. Rizzi, The dimensional fact model: a conceptual model for data warehouses, *Int. J. Coope. Informat. Sys.*, **7**(2-3): 215–247, 1998.
11. N. Prakash and A. Gosain, Requirements driven data warehouse development, *CAiSE Short Paper Proc.,* Klagenfurt/ Velden, Austria, 2003, pp. 13–16.
12. A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, and S. Paraboschi, Designing data marts for data warehouses, ACM Trans. Softw. *Engineer. Methodol.*, **10**(4): 452–483, 2001.
13. P. Giorgini, S. Rizzi, and M. Garzetti, Goal-oriented requirement analysis for data warehouse design, *Proc. Int. Workshop on Data Warehousing and OLAP,* Bremen, Germany, 2005, pp. 47–56.
14. C. Sapia, M. Blaschka, G. Höfling, and B. Dinter, Extending the E/R model for the multidimensional paradigm, *Proc. Int. Workshop on Design and Management of Data Warehouses, Singapore, 1998,* pp. 105–116.
15. S. Luján-Mora, J. Trujillo, and I. Song, A UML profile for multidimensional modeling in data warehouses, *Data Know. Engineer.*, **59**(3): 725–769, 2006.
16. S. Rizzi, Conceptual modeling solutions for the data warehouse, in R. Wrembel and C. Koncilia (eds.), *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, IRM Press: 2006, pp. 1–26.
17. J. Lechtenbörger and G. Vossen, Multidimensional normal forms for data warehouse design, *Informat. Sys.*, **28**(5): 415–434, 2003.
18. D. Theodoratos and M. Bouzeghoub, A general framework for the view selection problem for data warehouse design and evolution, *Proc. Int. Workshop on Data Warehousing and OLAP,* Washington DC, 2000, pp. 1–8.
19. M. Golfarelli, V. Maniezzo, and S. Rizzi, Materialization of fragmented views in multidimensional databases, *Data Knowl. Engineer.*, **49**(3): 325–351, 2004.
20. P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, Conceptual modeling for ETL processes, *Proc. Int. Workshop on Data Warehousing and OLAP,* McLean, 2002, pp. 14–21.
21. A. Simitsis, Mapping conceptual to logical models for ETL processes, *Proc. Int. Workshop on Data Warehousing and OLAP,* Bremen, Germany, 2005, pp. 67–76.
22. A. Simitsis, P. Vassiliadis, and T. K. Sellis, Optimizing ETL processes in data warehouses, *Proc. Int. Conf. on Data Engineering, Tokyo, Japan, 2005*, pp. 564–575.
23. H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman, Index selection for OLAP, *Proc. Int. Conf. on Data Engineering,* Birmingham, UK, 1997, pp. 208–219.
24. M. Golfarelli, S. Rizzi, and E. Saltarelli, Index selection for data warehousing, *Proc. Int. Workshop on Design and Management of Data Warehouses,* Toronto, Canada, 2002, pp. 33–42.
25. P. O'Neil and D. Quass, Improved query performance with variant indexes, *Proc. SIGMOD Conf.,* Tucson, AZ, 1997, pp. 38–49.
26. S. Rizzi, A. Abell, J. Lechtenbörger, and J. Trujillo, Research in data warehouse modeling and design: Dead or alive? *Proc. Int. Workshop on Data Warehousing and OLAP,* Arlington, VA, 2006, pp. 3–10.

FURTHER READING

B. Devlin, *Data Warehouse: From Architecture to Implementation*, Reading, MA: Addison-Wesley Longman, 1997.

W. H. Inmon, *Building the Data Warehouse*, 4th ed. New York: John Wiley & Sons, 2005.

M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis, *Fundamentals of Data Warehouse*, New York: Springer, 2000.

R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite, *The Data Warehouse Lifecycle Toolkit*, New York: John Wiley & Sons, 1998.

R. Mattison, *Data Warehousing*, New York: McGraw-Hill, 1996.

STEFANO RIZZI
University of Bologna
Bologna, Italy

DECISION SUPPORT SYSTEMS: FOUNDATIONS AND VARIATIONS

INTRODUCTION

Over the past quarter century, economic and technological forces have produced radical redefinitions of work, the workplace, and the marketplace. They have ushered in the era of knowledge workers, knowledge-based organizations, and the knowledge economy. People have always used the knowledge available to them to make decisions that shape the world in which they live. Decisions of workers, consumers, and organizations range from those affecting the world in some small or fleeting way to those of global and lasting proportions. In recent times, the number of decisions being made per time period and the complexity of factors involved in decision activities have grown dramatically. As the world's supply of knowledge continues to accelerate, the amount of knowledge used in making decisions has exploded. Computer-based systems that help decision makers deal with both the knowledge explosion and the incessant demands for decisions in a fast-paced, complicated world are called decision support systems (DSSs). Such systems have become practically indispensable for high performance, competitiveness, and even organizational survival.

Imagine an organization in which managers and other workers cannot use computers to aid any of their decisional activities. Contrast this fantasy with the vision of an organization whose managers and other knowledge workers routinely employ computers to get at and process knowledge that has a bearing on decisions being made. These DSSs store and process certain kinds of knowledge in much higher volumes and at much higher speeds than the human mind. In addition to such efficiency advantages, they can also be more effective for certain kinds of knowledge handling because they are not subject to such common human conditions as oversight, forgetfulness, miscalculation, bias, and stress. Failure to appreciate or exploit such decision support capabilities puts individuals and organizations at a major disadvantage

As a prelude to considering the characteristics of DSSs, we need to examine a couple of preliminaries: decision making and knowledge. Understanding what it means to make a decision provides a useful basis for exploring decision support possibilities. Understanding salient aspects of knowledge gives a starting point for appreciating ways in which computers can support the use of knowledge during decision making.

DECISION MAKING

General agreement in the management literature exists that a *decision* is a choice. It may be a choice about a "course of action" (1,2), choice of a "strategy for action" (3), or a choice leading to a certain desired objective"(4). Thus, we can think of *decision making* as an activity culminating in the selection of one from among multiple alternative courses of action.

In general, the number of alternatives identified and considered in decision making could be very large. The work involved in becoming aware of alternatives often makes up a major share of a decision-making episode. It is concerned with such questions as "Where do alternatives come from?" "How many alternatives are enough?" "How can large numbers of alternatives be managed so none is forgotten or garbled?" A computer-based system (i.e, a DSS) can help a decision maker cope with such issues.

Ultimately, one alternative is selected. But, which one? This choice depends on a study of the alternatives to understand their various implications as well as on a clear appreciation of what is important to the decision maker. The work involved in selecting one alternative often makes up a major share of a decision-making episode. It is concerned with such questions as: "To what extent should each alternative be studied?" "How reliable is our expectation about an alternative's impacts?" "Are an alternative's expected impacts compatible with the decision maker's purposes?" "What basis should be used to compare alternatives with each other?" "What strategy will be followed in arriving at a choice?" Computer-based systems (i.e., DSSs) can be very beneficial in supporting the study of alternatives. Some systems even recommend the selection of a particular alternative and explain the rationale underlying that advice.

Complementing the classic view of decisions and decision making, there is the *knowledge-based view* that holds that a decision is knowledge that indicates the nature of an action commitment (5). When we regard a decision as a piece of knowledge, making a decision means we are making a new piece of knowledge that did not exist before, manufacturing new knowledge by transforming or assembling existing pieces of knowledge. The manufacturing process may yield additional new knowledge as byproducts (e.g., knowledge derived as evidence to justify the decision, knowledge about alternatives that were not chosen, knowledge about improving the decision manufacturing process itself). Such byproducts can be useful later in making other decisions. A DSS is a computer-based system that aids the manufacturing process, just as machines aid in the manufacturing of material goods.

According to Mintzberg (6), there are four decisional roles: entrepreneur, disturbance handler, resource allocator, and negotiator. When playing the entrepreneur role, a decision maker searches for opportunities to advance in new directions aligned with his/her/its purpose. If such an opportunity is discovered, the decision maker initiates and devises controlled changes in an effort to seize the opportunity. As a disturbance handler, a decision maker initiates and devises corrective actions when facing an unexpected disturbance. As a resource allocator, a decision maker determines where efforts will be expended and how assets

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

will be deployed. This decision can be thought of as determining a strategy for structuring available resources. When playing the negotiator role, a decision maker bargains with others to try to reach a joint decision. Decision support systems are capable of supporting these four roles, although a particular DSS can be oriented more toward one role than the others.

A DSS can also vary to suit other particularities of contexts in which it is to be used. For instance, the context could be strategic decision making (concerned with deciding on purposes to fulfill, objectives to meet, changes in objectives, policies to adopt) decision making to ensure objectives are met and policies are observed, or operational decision making about performing specific tasks. These contexts vary along such dimensions as time horizons for deciding, extent of precision and detailed knowledge needed, narrow to wide-ranging knowledge, rhythm to decision-making activities, and degree of creativity or qualitative judgment required.

As another example of decision context, consider the maturity of the situation in which a decision is being made. Some decisions are made in established situations, whereas others are made in emergent situations. Well-established situations imply considerable experience in previously having made similar kinds of decisions, with a relatively high level of knowledge existing about the current state of affairs and the history of previous decisions of a similar nature. In contrast, emergent situations are characterized not only by some surprising new knowledge, but also often by a scarcity of relevant knowledge as well, often with intense effort required to acquire needed knowledge. The type of support likely to be most useful for established contexts could be quite different than what is valuable in the case of emergent settings.

Simon (2,7) says that decisions comprise a continuum ranging from structured to unstructured. The structuredness of a decision is concerned with how routine and repetitive is the process that produced it. A highly structured decision is one that has been manufactured in an established context. Alternatives from which the choice is made are clear-cut, and each can be readily evaluated in light of purposes and goals. All knowledge required to make the decision is available in a form that makes it straightforward to use. Unstructured decisions tend to be produced in emergent contexts. Issues pertinent to producing a decision are not well understood. Some issues may be entirely unknown to the decision maker; alternatives from which a choice will be made are vague or unknown, are difficult to compare and contrast, or cannot be easily evaluated. In other words, the knowledge required to produce a decision is unavailable, difficult to acquire, incomplete, suspect, or in a form that cannot be readily used by the decision maker. Semistructured decisions lie between the two extremes.

DSSs of varying kinds can be valuable aids in the manufacture of semistructured and unstructured decisions (8), as well as structured decisions (9). For the former, DSSs can be designed to facilitate the exploration of knowledge, help synthesize methods for reaching decisions, catalog and examine the results of brainstorming, provide multiple perspectives on issues, or stimulate a decision maker's creative capabilities. For structured decisions, DSSs automatically carry out some portion of the process used to produce a decision.

Because decisions are not manufactured in a vacuum, an appreciation of decision contexts and types can help us understand what features would be useful to have in a DSS. The same can be said for an appreciation of decision makers and decision processes, which we now consider in turn.

Decision making can involve an individual participant or multiple participants. In the multiparticipant case, the power to decide may be vested in a single participant, with other participants having varying degrees of influence over what the decision will be and how efficiently it will be produced. They do so by specializing in assorted knowledge processing tasks assigned to them during the making of the decision. These supporting participants function as extensions to the deciding participant's own knowledge processing capabilities. At the other extreme of multiparticipant decision making, participants share equal authority over the decision being made, with little formal specialization in knowledge processing tasks. This is referred to as a group decision maker, whereas the other extreme is called an organization decision maker. There are many variations between these extremes. Correspondingly, DSSs for these different kinds of multiparticipant decision makers can be expected to exhibit some different kinds of features. Moreover, a particular DSS may assist a specific participant, some subset of participants, or all participants involved in a multiparticipant decision.

Now, as for the process of decision making, Simon (2) says there are three important phases, which he calls intelligence, design, and choice. Moreover, running through the phases in any decision-making process, a decision maker is concerned with recognizing and solving some problems in some sequence (10). A decision-making process is governed by the decision maker's strategy for reaching a choice (11).

The intelligence phase is a period when the decision maker is alert for occasions to make decisions, preoccupied with collecting knowledge, and concerned with evaluating it in light of a guiding purpose. The design phase is a period wherein the decision maker formulates alternative courses of action, analyzes those alternatives to arrive at expectations about the likely outcomes of choosing each, and evaluates those expectations with respect to a purpose or objective. During the design phase, the decision maker may find that additional knowledge is needed, triggering a return to the intelligence phase to satisfy that need before continuing with the design activity. Evaluations of the alternatives are carried forward into the choice phase of the decision process, where they are compared and one is chosen. This choice is made in the face of internal and external pressures related to the nature of the decision maker and the decision context. It may happen that none of the alternatives are palatable, that several competing alternatives yield very positive evaluations, or that the state of the world has changed significantly since the alternatives were formulated and analyzed. So, the decision maker may return to one of the two earlier phases to collect more up-to-date knowledge, formulate new alternatives, reanalyze alternatives, reevaluate them, and so forth. Any phase is susceptible to computer-based support.

Recognizing and solving problems is the essence of activity within intelligence, design, and choice phases. For structured decisions, the path toward the objective of producing a decision is well charted. Problems to be surmounted are recognized easily, and the means for solving them are readily available. Unstructured decisions take us into uncharted territory. Problems that will be encountered along the way are not known in advance. Even when stumbled upon, they may be difficult to recognize and subsequently solve. Ingenuity and an exploratory attitude are vital for coping with these types of decisions.

Thus, a decision-making process can be thought of as a flow of problem-recognition and problem-solving exercises. In the case of a multiparticipant decision maker, this flow has many tributaries, made up of different participants working on various problems simultaneously, in parallel, or in some necessary sequence. Only if we solve its subproblems can we solve an overall decision problem. DSSs can help decision makers in recognizing and/or solving problems.

A decision-making process, and associated knowledge processing, are strongly colored by the strategy being used to choose an alternative. Well-known decision-making strategies include optimizing, satisficing, elimination-by-aspects, incrementalism, mixed scanning, and the analytic hierarchy process. As a practical matter, each strategy has certain strengths and limitations (9). A DSS designed to support an optimizing strategy may be of little help when a satisficing strategy is being adopted and vice versa.

We close this brief overview of decision making by considering two key questions about decision support: Why does a decision maker need support? What is the nature of the needed support? Computer systems to support decision makers are not free. Not only is there the cost of purchasing or developing a DSS, costs are also associated with learning about, using, and maintaining a DSS. It is only reasonable that the benefits of a DSS should be required to outweigh its costs. Although some DSS benefits can be difficult to measure in precise quantitative terms, all benefits are the result of a decision maker's need for support in overcoming cognitive, economic, or time limits (9).

Cognitive limits refer to limits in the human mind's ability to store and process knowledge. A person does not know everything all the time, and what is known cannot always be recalled in an instantaneous, error-free fashion. Because decision making is a knowledge-intensive activity, cognitive limits substantially restrict an individual's problem-solving efficiency and effectiveness. They may even make it impossible for the individual to reach some decisions. If these limits are relaxed, decision-maker productivity should improve. The main reason multiparticipant decision makers exist is because of this situation. Rather than having an individual find and solve all problems leading to a decision, additional participants serve as extensions to the deciding participant's own knowledge-handling skills, allowing problems to be solved more reliably or rapidly. A DSS can function as a supporting participant in decision making, essentially extending a person's cognitive capabilities.

To relax cognitive limits as much as possible, we could consider forming a very large team of participants. But this can be expensive not only in terms of paying and equipping more people, but also with respect to increased communication and coordination costs. At some point, the benefits of increased cognitive abilities are outweighed by the costs of more people. Decision support systems can soften the effects of economic limits when they are admitted as decision-making participants. If properly conceived and used, added DSSs increase the productivity of human participants and allow the organization decision maker to solve problems more efficiently and effectively.

A decision maker may be blessed with extraordinary cognitive abilities and vast economic resources but very little time. Time limits can put severe pressure on the decision maker, increasing the likelihood of errors and poor-quality decisions. There may not be sufficient time to consider relevant knowledge, to solve relevant problems, or to employ a desirable decision-making strategy. Because computers can process some kinds of knowledge much faster than humans, are not error-prone, work tirelessly, and are immune to stresses from looming deadlines, DSSs can help lessen the impacts of time limits.

To summarize, the support that a DSS offers normally includes at least one of the following:

- Alerts user to a decision-making opportunity or challenge
- Recognizes problems that need to be solved as part of the decision-making process
- Solves problems recognized by itself or by the user
- Facilitates or extends the user's ability to process (e.g., acquire, transform, and explore) knowledge
- Offers advice, expectations, evaluations, facts, analyses, and designs to users
- Stimulates the user's perception, imagination, or creative insight
- Coordinates/facilitates interactions within multiparticipant decision makers

Because knowledge forms the fabric of decision making, all the various kinds of support that a DSS can provide are essentially exercises in knowledge management. Thus, we now take a closer look at the matter of knowledge.

KNOWLEDGE MATTERS

Now, consider the notion of knowledge in more detail. A decision maker possesses a storehouse of knowledge, plus abilities to both alter and draw on the contents of that inventory (12). This characterization holds for all types of decision makers—individuals, groups, and organizations. In the multiparticipant cases, both knowledge and processing abilities are distributed among participants. Knowledge is extracted on an as-needed basis from the inventory and manipulated to produce solutions for the flow of problems that constitutes a decision manufacturing process. When the inventory is inadequate for solving some problem, outgoing messages are used in an effort to acquire the additional knowledge. The solution to each problem arising during the manufacturing process is itself a piece of knowledge. In turn, it may be used to find or solve other problems, whose solutions are knowledge allowing still other problems to be solved, and so forth, until the overall problem of producing a decision is solved (10). Thus, knowledge is the raw material, work-in-process, byproduct, and finished good of decision making.

If a system has and can use a representation of "something (an object, a procedure, . . . whatever), then the system itself can also be said to have knowledge, namely, the knowledge embodied in that representation about that thing" (13). Knowledge is embodied in *usable representations*, where a representation is a pattern of some kind: symbolic, digital, mental, behavioral, audio, visual, etc. To the extent that we can make use of that representation, it embodies knowledge. Of particular interest for DSSs are the representations that a computer can use and the knowledge processing ability corresponding to each knowledge representation approaches permitted in its portion of the knowledge storehouse. A DSS cannot process knowledge that it cannot represent. Conversely, a DSS cannot know what is represented by some pattern that it cannot process.

When designing or encountering a particular DSS, we should examine it in terms of the possibilities it presents for representing and processing knowledge—that is, the knowledge-management abilities it has to supplement human cognitive abilities. Over the years, several computer-based techniques for managing knowledge have been successfully applied to support decision makers, including text/hypertext/document management, database management, data warehousing, solver management spreadsheet analysis, rule management, message management, process management, and so forth (5). Each of these techniques can represent and process one or more of the three basic types of knowledge important for study of DSSs: descriptive, procedural, and reasoning knowledge (12,14).

Knowledge about the state of some world is called descriptive knowledge. It includes descriptions of past, present, future, and hypothetical situations. This knowledge includes data and information. In contrast, procedural knowledge is concerned with step-by-step procedures for accomplishing some task. Reasoning knowledge specifies conclusions that can be drawn when a specified situation exists. Descriptive, procedural, and reasoning knowledge can be used together within a single DSS to support decision making (10). For example, a DSS may derive (e.g., from past data) descriptive knowledge (e.g., a forecast) as the solution to a problem by using procedural knowledge indicating how to derive the new knowledge (e.g., how to calculate a forecast from historical observations). Using reasoning knowledge (e.g., rules) about what procedures are valid under different circumstances, the DSS infers which procedure is appropriate for solving the specific forecasting problem or to infer a valid sequence of existing procedures that, when carried out, would yield a solution.

Aside from knowledge type, knowledge has other attribute dimensions relevant to DSSs (15,16). One of these dimensions is knowledge orientation, which holds that a processor's knowledge can be oriented in the direction of the decision domain, of other related processors with which it interacts, and/or itself. A DSS can thus possess domain knowledge, which is descriptive, procedural, and/or reasoning (DPR) knowledge that allows the DSS to find or solve problems about a domain of interest (e.g., finance). It can possess relational knowledge, which is DPR knowledge that is the basis of a DSS's ability to effectively relate to (e.g., interact with) its user and other processors in the course of decision making. A DSS may also have self-knowledge, which is DPR knowledge about what it knows and what it can do. An adaptive DSS is one for which DPR knowledge for any of these three orientations can change by virtue of the DSS's experiences.

DECISION SUPPORT SYSTEM ROOTS, CHARACTERISTICS, AND BENEFITS

Rooted in an understanding of decision making, appreciating the purposes of DSSs serves as a starting point for identifying possible characteristics and benefits that we might expect a DSS to exhibit. These purposes include:

- Increase a decision maker's efficiency and/or effectiveness
- Help a decision maker successfully deal with the decisional context
- Aid one or more of the three decision-making phases
- Help the flow of problem-solving episodes proceed more smoothly or rapidly
- Relax cognitive, economic, and/or temporal constraints on a decision maker
- Help manage DPR knowledge that is important for reaching decisions

Decision support systems are deeply rooted in the evolution of business computing systems (aka information systems). Another way to appreciate characteristics and benefits of DSSs is to compare/contrast them with traits of their two predecessors in this evolution: data processing systems (DPS) and management information systems (MIS). All three share the traits of being concerned with record keeping; however, they differ in various ways, because each serves a different purpose in managing knowledge resources.

The main purpose of DPS was and is to automate the handling of large numbers of transactions. For example, a bank must deal with large volumes of deposit and withdrawal transactions every day, properly track each transaction's effect on one or more accounts, and maintain a history of all transactions to give a basis for auditing its operations. At the heart of a DPS lies a body of descriptive knowledge—computerized records of what is known as a result of transactions having happened. A data processing system has two major abilities related to the stored data: record keeping and transaction generation. The former keeps the records up-to-date in light of incoming transactions and can cause creation of new records, modification of existing records, deletion of obsolete records, or alteration of relationships among records. The second DPS ability is production of outgoing transactions based on stored descriptive knowledge, and transmitted to such targets as customers, suppliers, employees, or governmental regulators.

Unlike DPS, the central purpose of MIS was and is to provide periodic reports that recap certain predetermined aspects of past operations. They give regular snapshots of what has been happening. For instance, MIS might provide manufacturing managers with daily reports on parts usage and inventory levels, weekly reports on shipments received and parts ordered, a monthly report of production expenditures, and an annual report on individual workers' levels of productivity. Whereas DPS concern is with transforming transactions into records and generating transactions from records, MIS concern with record keeping focuses on using this stored descriptive knowledge as a base for generating recurring standard reports. Of course, an MIS also has facilities for creating and updating the collection of records that it keeps. Thus, an MIS can be regarded as extending the DPS idea to emphasize production of standard reports rather than producing voluminous transactions for customers, suppliers, employees, or regulators.

Information contained in standard MIS reports certainly can be factored into their users' decision-making activities. When this is the case, MIS can be fairly regarded as a kind of DSS. However, the nature of such support is very limited in light of our understanding of decision making. Reports generated by MIS are defined before the system is created. However, the situation surrounding a decision maker can be very dynamic. Except for the most structured kinds of decisions, information needs can arise unexpectedly and change more rapidly than MIS can be built or revised. Even when some needed information exists in a stack of reports accumulated from MIS, it may be buried within other information held by a report, scattered across several reports, and presented in a fashion not suitable for a user. Moreover, relevant information existing in MIS reports may not only be incomplete, difficult to dig out, unfocused, or difficult to grasp, it may also be in need of additional processing. For instance, a series of sales reports may list daily sales levels for various products, when a user actually needs projections of future sales based on data in these reports. Decision making proceeds more efficiently and effectively when a user can easily get complete, fully processed, focused descriptive knowledge (or even procedural and reasoning knowledge) presented in the desired way.

Standard reports generated by MIS are typically issued at set time intervals. But decisions that are not fully structured tend to be required at irregular, unanticipated times. The knowledge needed for manufacturing decisions must be available on an ad hoc, spur-of-the-moment, as-needed basis. Another limit on MIS ability to support decisions stems from their exclusive focus on managing descriptive knowledge. Decision makers frequently need procedural and/or reasoning knowledge as well. While an MIS deals with domain knowledge, decision making can often benefit from relational and self-knowledge possessed by its participants.

Decision support capabilities can be built on top of DPS and MIS functions. For instance, so-called digital dashboards are a good example. A digital dashboard integrates knowledge from multiple sources (e.g., external feeds and departmental DPSs and MISs) and can present various measures of key performance indicators (e.g., sales figures, operations status, balanced scorecards, and competitor actions) as an aid to executives in identifying and formulating problems in the course of decision making. Executives can face decisions, particularly more strategic decisions, that involve multiple inter-related issues involving marketing, strategy, competition, cash flow, financing, outsourcing, human resources, and so forth. In such circumstances, it is important for the knowledge system contents to be sufficiently wide ranging to help address cross-functional decisions.

DSS Characteristics

Ideally, a decision maker should have immediate, focused, clear access to whatever knowledge is needed on the spur-of-the-moment. Pursuit of this ideal separates decision support systems from their DPS and MIS ancestors. It also suggests characteristics we might expect to observe in a DSS:

- A DSS includes a body of knowledge that describes some aspects of the decision domain, that specifies how to accomplish various tasks, and/or that indicates what conclusions are valid in various circumstances.
- A DSS may also possess DPR knowledge of other decision-making participants and itself as well.
- A DSS has an ability to acquire, assimilate, and alter its descriptive, procedural, and/or reasoning knowledge.
- A DSS has an ability to present knowledge on an ad hoc basis in various customized ways as well as in standard reports.
- A DSS has an ability to select any desired subset of stored knowledge for either presentation or deriving new knowledge in the course of problem recognition and/or problem solving.
- DSS can interact directly with a participant in a decision maker in such a way that there is flexibility in choosing and sequencing knowledge manipulation activities.

There are, of course, variations among DSSs with respect to each of these characteristics. For instance, one DSS may possess descriptive and procedural knowledge, another holds only descriptive and reasoning knowledge, and another DSS may store only descriptive knowledge. As another example, there can be wide variations in the nature of users' interactions with DSSs (push versus pull interactions). Regardless of such variations, these characteristics combine to amplify a decision maker's knowledge management capabilities.

The notion of DSSs arose in the early 1970s (17,18). Within a decade, each of the characteristics cited had been identified as an important DSS trait (8,19–21). In that period, various DSSs were proposed or implemented for specific decision-making applications such as those for corporate planning (22), water-quality planning (23), banking (24), and so forth (19). By the late 1970s, new technological developments were emerging that would prove to give tremendous impetus to the DSS field. These developments included the microcomputer, electronic spreadsheets, management science packages, and ad hoc query interfaces (9). Technological advances impacting the DSS field have continued, including progress in artificial intelligence, collaborative technologies, and the Internet.

It is also notable that DSS characteristics are increasingly appearing in software systems not traditionally thought of as providing decision support, such as enterprise systems (25). They also commonly appear in websites, supporting decisions of both users and providers of those sites (26).

DSS Benefits

Benefits of a particular DSS depend not only on its precise characteristics, but also on the nature of the user and on the decision context. A good fit among these three factors must exist if potential benefits of the DSS are to become practical realities. A DSS that one user finds very beneficial may be of little value to another user, even though both are facing similar decisions. Or, a DSS that is beneficial to a user in one decision context may not be so valuable to that user in another context.

Nevertheless, we can identify potential DSS benefits (9,27), one or more of which is exhibited by any specific DSS:

- In a most fundamental sense, a DSS augments a user's own innate knowledge and knowledge manipulation skills by extending the user's capacity for representing and processing knowledge in the course of decision making.
- A DSS may be alert for events that lead to problem recognition, that demand decisions, or that present decisional opportunities and notify users.
- A user can have the DSS solve problems that the user alone would not even attempt or that would consume a great deal of time because of their complexity and magnitude.
- Even for relatively simple problems encountered in decision making, a DSS may be able to reach solutions faster and/or more reliably than the user.
- Even though a DSS may be unable to find or solve a problem facing a user, it may be used to guide the user into a rational view of the problem or otherwise stimulate the user's thoughts about the problem. For instance, the DSS may be used in an exploratory way to browse selectively through stored data or to analyze selectively the implications of ideas related to the problem. The user may ask the DSS for advice about dealing with the problem. Perhaps the user can have the DSS solve a similar problem to trigger insights about the problem actually being faced.
- The very activity of constructing a DSS may reveal new ways of thinking about the decision domain, relational issues among participants, or even partially formalize various aspects of decision making.
- A DSS may provide additional compelling evidence to justify a user's position, helping secure agreement or cooperation of others. Similarly, a DSS may be used by the decision maker to check on or confirm the results of problems solved independently of the DSS.
- Because of the enhanced productivity and/or agility a DSS fosters, it may give users or their organizations competitive advantages or allow them to stay competitive.

Empirical evidence supporting the actual existence of these benefits is examined in Ref. 28. Because no one DSS provides all these benefits to all decision makers in all decision situations, frequently many DSSs within an organization help to manage its knowledge resources. A particular decision maker may also make use of several DSSs within a single decision-making episode or across different decision-making situations.

DECISION SUPPORT SYSTEM ARCHITECTURE

We are now in a good position to examine the architecture of DSSs, which identifies four essential elements of a DSS and explains their interrelationships. A DSS has a language system, presentation system, knowledge system, and problem-processing system (9,10,29). By varying the makeup of these four elements, different DSSs are produced. Special cases of the generic DSS architecture vary in terms of the knowledge representation and processing approaches they employ giving DSS categories such as text-oriented DSSs, database-oriented DSSs, spreadsheet-oriented DSSs, solver-oriented DSSs, rule-oriented DSSs, compound DSSs, and collaborative DSSs (9).

The Generic Architecture

A decision support system can be defined in terms of three systems of representation:

- A language system (LS) consists of all messages the DSS can accept.
- A presentation system (PS) consists of all messages the DSS can emit.
- A knowledge system (KS) consists of all knowledge the DSS has assimilated and stored.

By themselves, these three kinds of systems can do nothing. They simply represent knowledge, either in the sense of messages that can be passed or representations that have been accumulated for possible processing. Yet they are essential elements of a DSS. Each is used by the fourth element: the problem-processing system (PPS). This system is the active part of a DSS, its software engine. As its name suggests, a PPS is what tries to recognize and solve problems (i.e., process problems) during the making of a decision.

Figure 1 illustrates how the four subsystems are related to each other and to a DSS user. The user is typically a human participant in the decision making but could also be the DSS developer, administrator, knowledge-entry person/device, or even another DSS. In any case, a user makes a

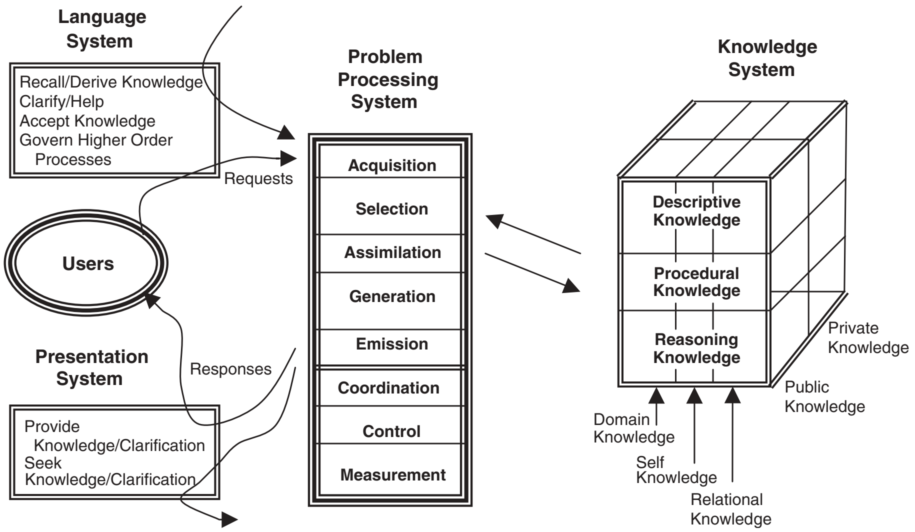


Figure 1. Decision support system architecture.

request by selecting an element of the LS. It could be a request to accept/provide knowledge, clarify previous requests/responses, and find/solve a problem. Once the PPS has been requested to process a particular LS element, it does so. This processing may very well require the PPS to draw on KS contents in order to interpret the request and develop a response to it. The processing may also change the knowledge held in the KS. In either event, the PPS may issue a response to the user. It does so by choosing to present one PS element to the user. The presentation choice is determined by the processing carried out with KS contents in response to the user's request. In some cases, PPS activity is triggered by an event rather than by a user's request, with the processing result being pushed to the user.

This simple architecture captures crucial and fundamental aspects common to all DSSs. To more fully appreciate the nature of a specific DSS, we must know about the requests that make up its LS, the responses that make up its PS, the knowledge representations allowed and existing in its KS, and the knowledge-processing capabilities of its PPS. Developers of DSSs must pay careful attention to all these elements when they design and build DSSs.

Requests that comprise a LS include those seeking:

- Recall or derivation of knowledge (i.e., solving a problem)
- Clarification of prior responses or help in making subsequent responses
- Acceptance of knowledge from the user or other external sources
- To govern a higher order process (e.g., launch a workflow)

Similarly, PS subsets include those that:

- Provide knowledge or clarification
- Seek knowledge or clarification from the user or other external sources

These LS and PS categorizations are based on message semantics. Yet another way of categorizing LS requests and PS responses could be based on distinctions in the styles of messages (e.g., menu, command, text, form, graphic, audio, direct manipulation, video, and animation).

A KS can include DPR knowledge for any of the orientations (domain, relational, and/or self), although the emphasis is commonly on domain-oriented knowledge. Many options are available to DSS developers for representations to employ in a KS, including text, hypertext, forms, datasets, database, spreadsheet, solvers, programs, rules, case bases, grammars, dictionaries, semantic nets, frames, documents, and video/audio records. The key point is that, for any of these representations to convey knowledge, it must be a representation that is usable by the DSS's problem processing system. That is, a PPS must have processing capabilities that correspond the each knowledge representation technique employed in its KS.

Regardless of the particular knowledge processing technique(s) it uses, a PPS tends to have the following knowledge manipulation abilities (9,10,16):

- Acquire knowledge from outside itself (interpreting it along the way)
- Select knowledge from its KS (e.g., for use in acquiring, assimilating, generating, and emitting knowledge)
- Assimilate knowledge into its KS (e.g., incorporate acquired or generated knowledge subject to maintenance of KS quality)
- Generate knowledge (e.g., derive or discover new knowledge)
- Emit knowledge to the outside (packaging it into suitable presentations along the way)

Some PPSs do not have all five abilities (e.g., can select knowledge, but not generate it). In addition, a PPS may have some higher order abilities that guide/govern the patterns of acquisition, selection, generation, assimilation, and emission activities that occur during a decisional episode (9,16):

- Measurement of knowledge and processing (allowing subsequent evaluation and learning)
- Control of knowledge and processing (ensuring integrity, quality, security, and privacy)
- Coordination of knowledge and processing (e.g., routing communications among participants, guiding workflows, promoting incentives to participants, intervening in negotiations, and enforcing processing priorities across concurrent decisional episodes)

The generic DSS architecture gives us a common base and fundamental terms for discussing, comparing, and contrasting specific DSSs and classes of DSSs.

DSS Classes

A useful way to look at KS contents and PPS abilities is in terms of the knowledge management techniques employed by a DSS (5,9). This gives rise to many special cases of the generic DSS architecture, several classes of these being considered here.

A text-oriented DSS supports decision makers by keeping track of textually represented DPR knowledge that could have a bearing on decisions (30,31). It allows documents to be created, revised, and reviewed by decision makers on an as-needed basis. The viewing can be exploratory browsing in search of stimulation or a focused search for some particular piece of knowledge needed in the manufacture of a decision. In either event, there is a problem with traditional text management: It is not convenient to trace a flow of ideas through separate pieces of text in the KS, as there are no explicit relationships among them.

This limitation is remedied in a hypertext-oriented DSS, whose KS is comprised of pieces of text explicitly linked to other pieces of text that are conceptually related to it. The PPS capabilities include creation, deletion, and traversal of nodes and links. The hypertext-oriented DSS supplements a decision maker's own associative capabilities by accurately storing and recalling large volumes of concepts and connections (32,33).

Another special case of the DSS architecture uses the database technique of knowledge management, especially relational and multidimensional approaches. Database-oriented DSSs have been used since the early years of the DSS field (34–36), tending to handle rigidly structured, often extremely voluminous, descriptive knowledge. The PPS is a database control system, plus an interactive query processing system and/or various custom-built processing systems, to satisfy user requests. Data warehouses (37) and data marts belong to this DSS class.

Well known for solving "what-if" problems, spreadsheet-oriented DSSs are in widespread use (38). The KS holds descriptive and procedural knowledge in spreadsheets. Using the spreadsheet technique, a DSS user not only can create, view, and modify procedural knowledge held in the KS but also can tell the PPS to carry out the instructions they contain. This capability gives DSS users much more power in handling procedural knowledge than is typical with either text management or database management. However, it is not nearly as convenient as database management in handling large volumes of descriptive knowledge, or text management in representing and processing unstructured textual passages.

Another class of DSSs is based on the notion of a solver—an executable algorithm that solves any member of a particular class of problems. Solver management is concerned with storage and use of a collection of solvers. Two approaches to solver-oriented DSS are fixed and flexible. In the fixed approach, solvers are part of the PPS, which means that a solver cannot be easily added to, or deleted from, the DSS nor readily modified. With the flexible approach, the PPS is designed to manipulate (e.g., create, delete, update, combine, and coordinate) solver modules held in the KS according to user requests.

The KS for a fixed solver-oriented DSS is typically able to hold datasets (groupings of numbers organized according to conventions required by the solvers). Many solvers can use a dataset, and a given solver can feed on multiple datasets. It is not uncommon for the KS to also hold editable problem statements and report format descriptions. In addition to solver modules, the KS flexible approach also accommodates datasets and perhaps problem statements or report formats. Each module requires certain data to be available for its use before its instructions can be carried out. Some of that data may already exist in KS datasets. The remaining data must either be furnished by the user (i.e., in the problem statement) or produced by executing other modules. In other words, a single module may not be able to solve some problems. Yet they can be solved by executing a certain sequence of modules. The results of carrying out instructions in the first module are used as data inputs in executing the second module, whose results become data for the third or subsequent module executions, and so forth, until a solution is achieved. Thus, the PPS coordinates the executions of modules that combine to make up the solver for a user's problem statement.

Another special case of the generic DSS architecture involves representing and processing rules (i.e., reasoning knowledge), (21,39). The KS of a rule-oriented DSS holds one or more rule sets, each pertaining to reasoning about what recommendation to give a user seeking advice on some subject. In addition to rule sets, it is common for the KS to contain descriptions of the current state. A user can request advice and explanation of the rationale for that advice. The PPS can do logical inference (i.e., to reason) with a set of rules to produce advice sought by a user. The problem processor examines pertinent rules in a rule set, looking for those whose premises are true for the current situation. This situation is defined by current state descriptions and the user's request for advice. When the PPS finds a true premise, it takes the actions specified in that rule's conclusion. This action sheds additional light on the situation, which allows premises of still other rules to be established as true, which causes actions in their conclusions to be taken. Reasoning continues in this way until some action is taken that yields the request advice or the PPS gives up because of insufficient knowledge in its KS. The PPS also has the ability to explain its behavior both during and after conducting the inference.

Rule-based inference is an artificial intelligence technique. A rule-oriented DSS is an example of what is called an intelligent DSS (21,39). Generally, any of the DSS categories can include systems that incorporate artificial intelligence mechanisms to enhance their problem processing capabilities. These mechanisms include natural language processing (for understanding and interpreting natural language), intelligent tutoring features (for offering help to users), machine learning approaches such as genetic algorithms or neural networks (for giving a DSS the ability to adapt its behavior based on its experiences), knowledge representation approaches such as semantic networks (for KS enrichment), search strategies (for knowledge selection and acquisition), and intelligent agent architectures (for event monitoring, collaborative processing, etc).

Each foregoing DSS class emphasizes a single knowledge management technique, supporting users in ways that cannot be easily replicated by DSSs based on different techniques. If a user needs the kinds of support offered by multiple knowledge management techniques, there are two basic options:

- Use multiple DSSs, each oriented toward a particular technique.
- Use a single DSS that encompasses multiple techniques.

The latter is a compound DSS, having a PPS equipped with the knowledge manipulation abilities of two or more techniques. The KS holds knowledge representations associated with all of these techniques. A good example of a compound DSS is evident in the architecture introduced by Sprague and Carlson (40), which combines database management and solver management into a single system, so that solvers (aka models) can operate against full-scale databases instead of datasets. Online analytic processing systems when operating against data warehouses belong to this class of compound DSSs. Software tools such as KnowledgeMan (aka the Knowledge Manager) and Guru have been used as prefabricated PPSs for building compound DSSs, synthesizing many knowledge management techniques in a single system (5,35,39). Such prefabricated PPSs are a realization of the concept of a generalized problem processing system (9,10).

Another important class, multiparticipant decision support systems (MDSSs), involves DSSs specifically designed to support decision-making efforts of a decision maker comprised of multiple participants. An MDSS that supports a group decision maker is called a group decision support system (GDSS). An MDSS that supports other kinds of multiparticipant decision makers, such as hierarchic teams, project teams, firms, agencies, or markets, is called an organizational decision support system (ODSS). Compared with a group, an organization has greater differentiation/specialization of participant roles in the decision making, greater coordination among these roles, greater differentiation in participants' authority over the decision, and more structured message flows among participants (9). It is possible that an MDSS supports negotiations among participants to resolve points of contention. If so, it is a negotiation support system (NSS) as well as being a GDSS or ODSS.

The KS and/or PPS of an MDSS can be distributed across multiple computers, which may be in close physical proximity (e.g., an electronic meeting room) or dispersed worldwide (e.g., as Internet nodes). Participants in the decision may interact at the same time or asynchronously. Although MDSSs can take on any of the characteristics and employ any of the knowledge management techniques discussed above, their hallmark is a focus on strong PPS coordination ability, perhaps with some control and measurement abilities as well. Examples of such ability includes (41-43):

- PPS controls what communication channels are open for use at any given time.
- PPS guides deliberations in such ways as monitoring and adjusting for the current state of participants' work, requiring input from all participants, permitting input to be anonymous, enforcing a particular coordination method (e.g., nominal group technique), and handling/tabulating participant voting.
- PPS continually gathers, organizes, filters, and formats public materials generated by participants during the decision-making process, electronically distributing them to participants periodically or on demand; it permits users to transfer knowledge readily from private to public portions of the KS (and vice versa) and perhaps even from one private store to another.
- PPS continually tracks the status of deliberations as a basis for giving cues to participants (e.g., who has viewed or considered what, where are the greatest disagreements, where is other clarification or analysis needed, when is there a new alternative to be considered, and who has or has not voted).
- PPS regulates the assignment of participants to roles (e.g., furnishing an electronic market in which they bid for the opportunity to fill roles).
- PPS implements an incentive scheme designed to motivate and properly reward participants for their contributions to decisions.
- By tracking what occurred in prior decision-making sessions, along with recording feedback on the results for those sessions (e.g., decision quality, process innovation), a PPS enables the MDSS to learn how to coordinate better or to avoid coordination pitfalls in the future.

For each of these abilities, a PPS may range from offering relatively primitive to relatively sophisticated features.

Thus, the KS of an MDSS typically includes a group/organization memory of what has occurred in decisional episodes. In addition to public knowledge that can be selected by any/all participants, the KS may also accommodate private knowledge spaces for each participant. Similarly, the LS (and PS) may include both a public language (and presentations) comprising messages suited to all participants and private languages (and presentations) available to specific participants. In addition to the kinds of users noted in Fig. 1, an MDSS may also interact with a facilitator(s), who helps the participants (individually and collectively) make effective use of the MDSS.

A GDSS seeks to reduce losses that can result from working as a group, while keeping (or enhancing) the gains that group work can yield (44). DeSanctis and Gallupe (45) identify three levels of GDSSs, differing in terms of features they offer for supporting a group decision maker. A Level-1 GDSS reduces communication barriers that would otherwise occur among participants, stimulating and hastening messages exchanges. A Level-2 GDSS reduces uncertainty and "noise" that can occur in a group's decision process via various systematic knowledge manipulation techniques like those encountered in the DSS classes previously described. A Level-3 GDSS governs timing, content, or patterns of messages exchanged by participants, actively driving or regulating a group's decision process.

Nunamaker et al. (44) draw the following conclusions from their observations of GDSSs in the laboratory and the field:

- Parallel communication encourages greater participation and reduces the likelihood of a few participants dominating the proceedings.
- Anonymity reduces apprehensions about participating and lessens the pressure to conform, allowing for more candid interactions.
- Existence of a group memory makes it easier for participants to pause and ponder the contributions of others during the session, as well as preserving a permanent record of what has occurred.
- Process structuring helps keep the participants focused on making the decision, reducing tendencies toward digression and unproductive behaviors.
- Task support and structuring give participants the ability to select and derive needed knowledge.

The notion of an ODSS has long been recognized, with an early conception viewing an organizational decision maker as a knowledge processor having multiple human and multiple computer components, organized according to roles and relationships that divide their individual labors in alternative ways in the interest of solving a decision problem facing the organization (46). Each component (human or machine) is an intelligent processor capable of solving some class of problems either on its own or by coordinating the efforts of other components—passing messages to them and receiving messages from them. The key ideas in this early framework for ODSS are the notions of distributed problem solving by human and machine knowledge processors, communication among these problem solvers, and coordination of interrelated problem-solving efforts in the interest of solving an overall decision problem. To date, organizational DSSs have not received nearly as much attention as group DSSs.

George (47) identifies three main ODSS themes:

- Involves computer-based technologies and may involve communication technology
- Accommodates users who perform different organizational functions and who occupy different positions in the organization's hierarchical levels
- Is primarily concerned with decisions that cut across organizational units or impact corporate issues

and organizes candidate technologies for ODSS development into several categories:

- Technologies to facilitate communication within the organization and across the organization's boundaries
- Technologies to coordinate use of resources involved in decision making
- Technologies to filter and summarize knowledge (e.g., intelligent agents)
- Technologies to track the status of the organization and its environment
- Technologies to represent and process diverse kinds of knowledge needed in decision making
- Technologies to help the organization and its participants reach decisions

Computer systems designed for specific business processes, such as customer relationship management, product lifecycle management, and supply chain management are evolving from an initial emphasis on transaction handling and reporting, to increasingly offer decision support characteristics. As such, they can be considered to be ODSSs. The most extensive of these systems are enterprise resource planning systems, which seek to integrate traditionally distinct business applications into a single system with a common knowledge store. Although these systems often regarded from the perspectives of data processing and management information systems, research indicates that enterprise systems do have some ODSS features and do provide decision support benefits to organizations (25,28). Their potential as decision support platforms is increasingly reflected in new product offerings of software vendors.

Although some GDSSs and ODSSs have features that can benefit negotiators, these features have not been the central motive or interest of such systems. DSSs designed specifically for supporting negotiation activities are called negotiationsupport systems. Pioneering tools for NSS development include NEGO, a PPS designed to help negotiators change their strategies, form coalitions, and evaluate compromises (48), and NEGOPLAN, an expert system shell that represents negotiation issues and decomposes negotiation goals to help examine consequences of different negotiation scenarios (49). See Refs. 50 and 51 for surveys of NSS software and Ref. 52 for a formal theoretical foundation of NSSs.

CONCLUSION

Decision support systems have major socioeconomic impacts and are so pervasive as to be practically invisible. The study and application of DSSs comprises a major subject area within the information systems discipline.

This extensive area, has unifying principles rooted in knowledge management and the generic architecture shown in Fig. 1; it is also an area rich in diversity, nuances, and potential for additional advances. For a thorough, in-depth appreciation of the DSS area, consult the two-volume *Handbook on Decision Support Systems*(53), the flagship journal *Decision Support Systems*, the DSS applications-intensive *Interfaces*, the *Journal of Decision Systems, Journal of Data Warehousing*, and *Business Intelligence Journal*.

For all their value, we must keep in mind the fact that DSSs have limitations. They cannot make up for a faulty (e.g., irrational) decision maker, and the efficacy of the support they provide is constrained by the extent and quality of their knowledge systems relative to the decision situation being faced (9). Their effectiveness is influenced by such factors as adequacy of a user's problem formulations, capture of relevant variables, and timely and accurate knowledge about the status of these decision parameters. Ultimately, the value of a DSS does not derive simply from its existence, but it depends very much on how it is designed, used, maintained, and evaluated, plus the decision maker's assumptions about the DSS.

BIBLIOGRAPHY

1. T. W. Costello and S. S. Zalkind, *Psychology in Administration: A Research Orientation*, Englewood Cliffs, NJ: Prentice Hall, 1963.
2. H. A. Simon, *The New Science of Management Decision*, New York: Harper & Row, 1960.
3. P. C. Fishburn, *Decision and Value Theory*, New York: John Wiley, 1964.
4. C. W. Churchman, *Challenge to Reason*, New York: McGraw-Hill, 1968.
5. C. W. Holsapple and A. B. Whinston, *The Information Jungle*, Homewood, IL: Dow Jones-Irwin, 1988.
6. H. Mintzberg, *The Nature of Managerial Work*, Englewood Cliffs, NJ: Prentice Hall, (first published in 1973), 1980.
7. H. A. Simon, *Models of Man*, New York: John Wiley, 1957.
8. P. G. W. Keen and M. S. ScottMorton, *Decision Support Systems: An Organizational Perspective*, Reading, MA: Addison-Wesley, 1978.
9. C. W. Holsapple and A. B. Whinston, *Decision Support Systems: A Knowledge-Based Approach*, St. Paul, MN: West, 1996.
10. R. H. Bonczek, C. W. Holsapple, and A. B. Whinston, *Foundations of Decision Support Systems*, New York: Academic Press, 1981.
11. I. L. Janis and I. Mann, *Decision Making: A Psychological Analysis of Conflict, Choice, and Commitment*, New York: The Free Press, 1977.
12. C. W. Holsapple, Knowledge management in decision making and decision support, *Knowledge and Policy: The Internat. J. Knowledge Trans. Utilization*, **8**(1): 1995.
13. A. Newell, The knowledge level, *Artificial Intelli.*, **18**(1): 1982.
14. C. W. Holsapple, The inseparability of modern knowledge management and computer-based technology, *J. Knowledge Management*, **9**(1): 2005.
15. C. W. Holsapple, Knowledge and its attributes, in C. W. Holsapple (ed.), *Handbook on Knowledge Management*, Vol. 1, Berlin: Springer, 2003.
16. C. W. Holsapple and K. D. Joshi, A formal knowledge management ontology: conduct, activities, resources, and influences, *J. Amer. Soc. Infor. Sci. Technol.*, **55**(7): 2004.
17. T. P. Gerrity, Design of man-machine decision systems: an application to portfolio management, *Sloan Management Review*, Winter, 1971.
18. M. S. Scott Morton, *Management Decision Systems: Computer-Based Support for Decision Making*, Cambridge, MA: Division of Research, Harvard University, 1971.
19. S. L. Alter, *Decision Support Systems: Current Practice and Continuing Challenges*, Reading, MA: Addison-Wesley, 1980.
20. R. H. Bonczek, C. W. Holsapple and A. B. Whinston, The evolving roles of models within decision support systems, *Decision Sciences*, April, 1980.
21. R. H. Bonczek, C. W. Holsapple, and A. B. Whinston, Future directions for developing decision support systems, *Decision Sciences*, October, 1980.
22. R. A. Seaberg and C. Seaberg, Computer-based decision systems in Xerox corporate planning, *Management Science*, **20**(4): 1973.
23. C. W. Holsapple and A. B. Whinston, A decision support system for area-wide water quality planning, *Socio-Economic Planning Sciences*, **10**(6): 1976.
24. R. H. Sprague, Jr., and H. J. Watson, A decision support system for banks, *Omega*, **4**(6): 1976.
25. C. W. Holsapple and M. Sena, Decision support characteristics of ERP systems, *Internat. J. Human-Computer Interaction*, **16**(1): 2003.
26. C. W. Holsapple, K. D. Joshi, and M. Singh, in M. Shaw (ed.), Decision support applications in electronic commerce, *Handbook on Electronic Commerce*, Berlin: Springer, 2000.
27. C. W. Holsapple, Adapting demons to knowledge management environments, *Decision Support Systems*, **3**(4): 1987.
28. C. W. Holsapple and M. Sena, ERP plans and decision support benefits, *Decision Support Systems*, **38**(4): 2005.
29. B. Dos Santos and C. W. Holsapple, A framework for designing adaptive DSS interfaces, *Decision Support Systems*, **5**(1): 1989.
30. J. Fedorowicz, Evolving technology for document-based DSS, in R. Sprague, Jr. and H. Watson (eds.), *Decision Support Systems: Putting Theory into Practice*, 2nd ed., Englewood Cliffs, NJ: Prentice-Hall, 1989.
31. P. G. W. Keen, Decision support systems: The next decade, *Decision Support Systems*, **3**(3): 1987.
32. M. Bieber, Automating hypermedia for decision support, *Hypermedia*, **4**(2): 1992.
33. R. P. Minch, Application research areas for hypertext in decision support systems, *J. Managem. Informat. Syst.*, **6**(2): 1989.
34. R. C. Bonczek, C. W. Holsapple, and A. B. Whinston, A decision support system for area-wide water quality planning, *Socio-Economic Planning Sciences*, **10**(6): 1976.
35. J. D. Joyce and N. N. Oliver, Impacts of a relational information system in industrial decisions, *Database*, **8**(3): 1977.
36. R. L. Klaas, A DSS for airline management, *Database*, **8**(3): 1977.
37. P. Gray and H. J. Watson, *Decision Support in the Data Warehouse*, Upper Saddle River, NJ: Prentice-Hall, 1998.
38. P. B. Cragg and M. King, A review and research agenda for spreadsheet based DSS, *International Society for Decision Support Systems Conference*, Ulm, Germany, 1992.
39. C. W. Holsapple and A. B. Whinston, *Manager's Guide to Expert Systems*, Homewood, IL: Dow Jones-Irwin, 1986.

40. R. H. Sprague, Jr. and E. D. Carlson, *Building Effective Decision Support Systems*, Englewood Cliffs, NJ: Prentice Hall, 1982.
41. C. Ching, C. W. Holsapple, and A. B. Whinston, Reputation, learning, and organizational coordination, *Organization Science*, **3**(2): 1992.
42. J. A. Hoffer and J. S. Valacich, Group memory in group support systems: A foundation for design, in L. Jessup and J. Valacich (eds.), *Group Support Systems: New Perspectives*, New York: Macmillan, 1993.
43. M. Turoff, M. S. R. Hiltz, A. N. F. Bahgat, and A. R. Rana, Distributed group support systems, *MIS Quarterly*, **17**(4): 1993.
44. J. F. Nunamaker, Jr.A. R. Dennis, J. S. Valacich, D. R. Vogel, and J. F. George, Group support systems research: Experience from the lab and field, in L. Jessup and J. Valacich (eds.), *Group Support Systems: New Perspectives*, New York: Macmillan, 1993.
45. G. DeSanctis and R. B. Gallupe, A foundation for study of group decision support systems, *Management Science*, **33**(5): 1987.
46. R. H. Bonczek, C. W. Holsapple, and A. B. Whinston, Computer based support of organizational decision making, *Decision Sciences*, April, 1979.
47. J. F. George, The conceptualization and development of organizational decision support systems, *J. Management Inform. Sys.*, **8**(3): 1991.
48. G. Kersten, NEGO—group decision support system, *Inform. and Management*, **8**: 1985.
49. S. Matwin, S. Szpakowicz, E. Koperczak, G. Kersten, and W. Michalowski, Negoplan: An expert system shell for negotiation support, *IEEE Expert*, **4**(1): 1989.
50. R. C. Anson and M. T. Jelassi, A developmental framework for computer-supported conflict resolution, *European J. Operational Res.*, **46**: 1990.
51. M. Jelassi and A. Foroughi, Negotiation support systems: An overview of design issues and existing software, *Decision Support Systems*, **5**(2): 1989.
52. C. W. Holsapple, H. Lai, and A. B. Whinston, A formal basis for negotiation support system research, *Group Decision and Negotiation*, **7**(3): 1995.
53. F. Burstein and C. W. Holsapple, *Handbook on Decision Support Systems*, Berlin: Springer, 2008.

FURTHER READING

R. H. Bonczek, C. W. Holsapple, and A. B. Whinston, Aiding decision makers with a generalized database management system, *Decision Sciences*, April, 1978.

P. B. Osborn and W. H. Zickefoose, Building expert systems from the ground up, *AI Expert*, **5**(5): 1990.

CLYDE W. HOLSAPPLE
University of Kentucky
Lexington, Kentucky

D

DEDUCTIVE DATABASES

INTRODUCTION

The field of deductive databases is based on logic. The objective is to derive new data from facts in the database and rules that are provided with the database. In the Background section, a description is provided of a deductive database, of a *query* and of an *answer* to a query in a deductive database. Also discussed is how deductive databases extend relational databases (see RELATIONAL DATABASES) and form a subset of *logic programming* (see AI LANGUAGES AND PROCESSING). In the Historical Background of Deductive Databases section, we discuss the pre-history, the start of the field, and the major historical developments including the formative years and initial prototype systems. Then we first present Datalog databases with recursion but without negation and also explain semantic query optimization *(SQO)* and cooperative answering; then we introduce default negation for stratified databases; discuss current stratified prototype systems; the introduction of deductive database concepts into the relational database language *SQL:99;* why the deductive database technology has not led to commercial systems; and introduce the concept of nonstratified deductive databases. The Disjunctive Deductive Databases section describes incomplete databases, denoted $Datalog^{\neg}_{disj}$, that permit more expressive knowledge base systems. We discuss the need for disjunction in knowledge base systems; disjunctive deductive databases that do not contain default negation; the extension of disjunctive systems to include default and logical negation; the extension of the answer set semantics to incorporate default negation; and methods to select an appropriate semantics. In the next section, implementations of nonstratified deductive and disjunctive databases are presented and we define a knowledge base system in terms of the extensions to relational databases described in this article. The Applications section has brief descriptions of some of the applications of deductive databases: data integration, handling preferences, updates, AI planning, and handling inconsistencies. The Final section summarizes the work.

BACKGROUND

A deductive database is an extension of a relational database. Formally, a deductive database, (*DDB*) is a triple, < *EDB, IDB, IC* >, where *EDB* is a set of facts, called the *extensional database, IDB* is a set of rules, called the *intensional database,* and *IC* is a set of *integrity constraints*. A *DDB* is based on first-order logic. An atomic formula is a k-place predicate letter whose arguments are constants or variables. Atomic formulas evaluate to *true* or *false*. The *EDB* consists of *ground atomic formulas* or disjunctions of ground atomic formulas. An atomic formula is *ground* if it consists of a predicate with k arguments, where the arguments are constants. Examples of ground atomic formulas are *supplies(acme, shovels),* and *supplies (acme, screws),* whose intended meaning is: "The Acme Corporation supplies shovels and screws." An example of a disjunction is: *supplierloc(acme, boston)* ∨ *supplierloc(acme, washington),* whose intended meaning is: "The Acme Corporation is located either in Boston or in Washington, or in both locations." Corresponding to an atomic formula, there is a *relation* that consists of all tuples whose arguments are in an atomic formula with the same name. For the *supplies* predicate, there is a relation, the *SUPPLIES* relation, that consists of a set of tuples (e.g., {< *acme, shovels* >, < *acme, screws* >}) when the *SUPPLIES* relation consists of the above two facts. In a relational database, the *EDB* consists only of atoms. Throughout the article, predicate letters are written in lower case and arguments of predicates that are constants are also written in lower case, whereas upper-case letters denote variables.

The intensional database consists of a set of rules of the form:

$$L_1, \ldots, L_n \leftarrow M_1, \ldots, M_m,\ not\ M_{m+1}, \ldots,\ not\ M_{m+l} \quad (1)$$

where the L_i and the M_j are atomic formulas and *not* is default negation (discussed below). Intensional rules are universally quantified and are an abbreviation of the formula:

$$\forall X_1 \ldots, X_k (L_1 \vee \ldots \vee L_n \leftarrow M_1 \wedge \ldots \wedge M_m \wedge not\, M_{m+1} \wedge \ldots \wedge not\, M_{m+l})$$

where the $X_1, \ldots, X_k$ lists all free variables.

A rule with $n = 0$ is either a query or an integrity constraint. When $n > 0$, the rule is either an IDB rule used to derive data or it may be an integrity constraint that restricts what tuples may be in the database. Rules for which $n \leq 1$ and $l = 0$ are called *Horn rules*.

DDBs restrict arguments of atomic formulas to constants and variables, whereas in first-order logic atomic formulas may also contain function symbols as arguments, which assures that answers to queries in *DDBs* are finite[1]. Rules may be read either declaratively or procedurally. A declarative reading of Formula (1) is:

L_1 *or* L_2 *or* ... *or* L_n is *true* if M_1 *and* M_2 *and* ... *and* M_m *and not* M_{m+1} *and* ... *and not* M_{m+l} are all *true*.

A procedural reading of Formula (1) is:

[1] When there are function symbols, an infinite number of terms may be generated from the finite number of constants and the function symbols; hence, the answer to a query may be infinite.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

L_1 *or* L_2 *or* ... *or* L_n are solved if M_1 *and* M_2 *and* ... *and* M_m *and not* M_{m+1} *and* ... *and not* M_{m+l} are solved.

The left-hand side of the implication, L_1 *or* ... *or* L_n, is called the *head* of the rule, whereas the right-hand side,

M_1 *and* M_2 *and* ... *and* M_m *and not* M_{m+1} *and* ... *and not* M_{m+l} is called the *body* of the rule.

Queries to a database, $Q(X_1, \ldots, X_r)$, are of the form $\exists X_1 \ldots \exists X_r\,(L_1 \wedge L_2 \ldots \wedge L_s)$, written as $\leftarrow L_1, L_2, \ldots, L_s$, where $s \geq 1$, the L_i are literals and the X_i, $1 \leq i \leq r$, are the free variables in Q. An answer to a query has the form $< a_{11}, \ldots, a_{1r} > + < a_{21}, \ldots, a_{2r} > + \ldots + < a_{k1}, \ldots, a_{kr} >$ such that $Q(a_{11}, \ldots, a_{1r}) \vee Q(a_{21}, \ldots, a_{2r}) \vee \ldots \vee Q(a_{k1}, \ldots, a_{kr})$ is provable from the database, which means that an inference system is used to find answers to queries.

DDBs are closely related to logic programs when the facts are restricted to atomic formulas and the rules have only one atom in the left-hand side of a rule. The main difference is that a logic program query search is for a single answer, proceeding top-down from the query to an answer. In *DDBs,* searches are bottom-up, starting from the facts, to find all answers. A logic program query might ask for an item supplied by a supplier, whereas in a deductive database, a query asks for all items supplied by a supplier. *DDBs* restricted to atoms as facts, and rules that consist of single atoms on the left-hand side of a rule and atoms on the right-hand side of a rule that do not contain the default rule for negation, *not,* are called *Datalog* databases, (i.e., rules in Formula(1), where $n = 1$, $m \geq 0$, and $l = 0$). Rules in *Datalog* databases may be recursive. A traditional relational database is a *DDB* where the *EDB* consists of atoms and *IDB* rules are not recursive.

There are several different concepts of the relationship of integrity constraints to the union of the *EDB* and the *IDB* in the *DDB*. Two such concepts are *consistency* and *theoremhood*. In the consistency approach (proposed by Kowalski), the *IC* must be consistent with $EDB \cup IDB$. In the theoremhood approach (proposed by Reiter and by Lloyd and Topor), each integrity constraint must be a theorem of $EDB \cup IDB$.

To answer queries that consist of conjunctions of positive and default negated atoms in *Datalog* requires that semantics be associated with negation because only positive atoms can be derived from *Datalog* DDBs. How one interprets the semantics of default negation can lead to different answers. Two important semantics for handling default negation are termed the *closed-world assumption* (*CWA*), due to Reiter, and *negation-as-finite-failure* (*NFF*), due to Clark. In the *CW A,* failure to prove the positive atom implies that the negated atom is *true*. In the *NFF,* predicates in the *EDB* and the *IDB* are considered the *if* portion of the database and are closed by effectively reversing the implication to achieve the *only if* part of the database. The two approaches lead to slightly different results. Negation, as applied to disjunctive theories, is discussed later.

Example 1 (Ancestor). *Consider the following database that consists of parents and ancestors. The database consists of two predicates, whose schema are* $p(X, Y)$, *intended to mean that Y is a* parent of *X, and* $a(X, Y)$, *intended to mean that Y is an* ancestor of *X. The database consists of five EDB statements and two IDB rules:*

r1. $p(mike, jack)$
r2. $p(sally, jack)$
r3. $p(katie, mike)$
r4. $p(beverly, mike)$
r5. $p(roger, sally)$
r6. $a(X, Y) \leftarrow p(X, Y)$
r7. $a(X, Y) \leftarrow p(X, Z), a(Z, Y)$

The answer to the question $p(mike, X)$ *is jack. The answer to the question* $a(mike, X)$ *is jack using rule r6. An answer to the query* $a(roger, X)$ *is sally using rule r6. Another answer to the query* $a(roger, X)$*, jack, is found by using rule r7.*

For the query $p(katie, jack)$*, the answer by the* CWA *is* no; *jack is not a* parent of katie. *The reason is that there are only five facts, none of which specify* $p(katie, jack)$*, and there are no rules that can be used to find additional parents.*

More expressive power may be obtained in a *DDB* by allowing negated atoms on the right-hand side of a rule. The semantics associated with such databases depends on how the rule of negation is interpreted, as discussed in the Datalog and Extended Deductive Databases section and the Disjunctive Deductive Databases section.

HISTORICAL BACKGROUND OF DEDUCTIVE DATABASES

The prehistory of *DDBs* is considered to be from 1957 to 1970. The efforts in this period used primarily ad hoc or simple approaches to perform deduction. The period 1970 to 1978 were the formative years, which preceded the start of the field. The period 1979 to 2003 saw the development of a theoretical framework and prototype systems.

Prehistory of Deductive Databases

In 1957, a system called *ACSI-MATIC* was under development to automate work in Army intelligence. An objective was to derive new data based on given information and general rules. Chains of related data were sought and the data contained reliability estimates. A prototype system was implemented to derive new data whose reliability values depended on the reliability of the original data. The deduction used was modus ponens (i.e., from p and $p \rightarrow q$, one concludes q, where p and q are propositions).

Several *DDBs* were developed in the 1960s. Although in 1970 Codd founded the field of *Relational Databases,* relational systems were in use before then. In 1963, using a relational approach, Levien and Maron developed a system, *Relational Data File* (*RDF*), that had an inferential capability, implemented through a language termed *INFEREX*. An *INFEREX* program could be stored in the system (such as in current systems that store views) and re-executed, if necessary. A programmer specified reasoning rules via an *INFEREX* program. The system handled credibility ratings of sentences in forming deductions. Theoretical work by Kuhns on the *RDF* project recognized that there were classes of questions that were, in a sense, not "reasonable." For example, let the database consist of the statement, "Reichenbach wrote *Elements of Symbolic Logic.*" Whereas the question, "What books has Reichenbach written?", is reasonable, the questions, "What books has Reichenbach not written?", or, "Who did not write 'Elements of Symbolic Logic'?", are not reasonable. It is one of the first times that the issue of negation in queries was explored.

In 1964, Raphael, for his Ph.D. thesis at M.I.T., developed a system called *Semantic Information Retrieval (SIR),* which had a limited capability with respect to deduction, using special rules. Green and Raphael subsequently designed and implemented several successors to *SIR:* $QA-1$, a re-implementation of *SIR;* $QA-2$, the first system to incorporate the *Robinson Resolution Principle* developed for automated theorem proving; $QA-3$ that incorporated added heuristics; and $QA-3.5$, which permitted alternative design strategies to be tested within the context of the resolution theorem prover. Green and Raphael were the first to recognize the importance and applicability of the work performed by Robinson in automated theorem proving. They developed the first *DDB* using formal techniques based on the Resolution Principle, which is a generalization of modus ponens to first-order predicate logic. The Robinson Resolution Principle is the standard method used to deduce new data in *DDBs*.

Deductive Databases: The Formative Years 1969–1978

The start of deductive databases is considered to be November 1977, when a workshop, "Logic and Data Bases," was organized in Toulouse, France. The workshop included researchers who had performed work in deduction from 1969 to 1977 and used the Robinson Resolution Principle to perform deduction. The Workshop, organized by Gallaire and Nicolas, in collaboration with Minker, led to the publication of papers from the workshop in the book, "Logic and Data Bases," edited by Gallaire and Minker.

Many significant contributions were described in the book. Nicolas and Gallaire discussed the difference between model theory and proof theory. They demonstrated that the approach taken by the database community was model theoretic (i.e., the database represents the truths of the theory and queries are answered by a bottom-up search). However, in logic programming, answers to a query used a proof theoretic approach, starting from the query, in a top-down search. Reiter contributed two papers. One dealt with compiling axioms. He noted that if the *IDB* contained no recursive axioms, then a theorem prover could be used to generate a new set of axioms where the head of each axiom was defined in terms of relations in a database. Hence, a theorem prover was no longer needed during query operations. His second paper discussed the *CWA*, whereby in a theory, if one cannot prove that an *atomic formula* is *true,* then the negation of the atomic formula is assumed to be *true*. Reiter's paper elucidated three major issues: the definition of a query, an answer to a query, and how one deals with negation. Clark presented an alternative theory of negation. He introduced the concept of *if-and-only-if* conditions that underly the meaning of negation, called negation-as-finite-failure. The Reiter and Clark papers are the first to formally define default negation in logic programs and deductive databases. Several implementations of deductive databases were reported. Chang developed a system called *DEDUCE;* Kellogg, Klahr, and Travis developed a system called *Deductively Augmented Data Management System (DADM);* and Minker described a system called *Maryland Refutation Proof Procedure 3.0 (MRPPS 3.0)*. Kowalski discussed the use of logic for data description. Darvas, Futo, and Szeredi presented applications of *Prolog* to drug data and drug interactions. Nicolas and Yazdanian described the importance of integrity constraints in deductive databases. The book provided, for the first time, a comprehensive description of the interaction between logic and databases.

References to work on the history of the development of the field of deductive databases may be found in Refs. 1 and 2. A brief description of the early systems is contained in Ref. 1.

Deductive Databases: Prototypes, 1979–2004

During the period 1979 through today, a number of prototype systems were developed based on the Robinson Resolution Principle and bottom-up techniques. Most of the prototypes that were developed during this period either no longer exist or are not supported. In this section, we describe several efforts because these systems contributed to developments that were subsequently incorporated into *SQL* as described in the SQL: 1999 section.

These systems include *Validity* developed by Nicolas and Vieille; *Coral,* developed by Ramakrishnan at the University of Wisconsin; and *NAIL!,* developed by Ullman and his group at Stanford University. All of these prototypes introduced new techniques for handling deductive databases.

The *Validity* system's predecessor was developed at the European Computer Research Consortium, was directed by Nicolas, and started in 1984. It led to the study of algorithms and prototypes: deductive query evaluation methods (*QSQ*/*SLD* and others); integrity checking (Soundcheck); hypothetical reasoning and *IC* checking; and aggregation through recursion.

Implementation at Stanford University, directed by Ullman, started in 1985 on *NAIL! (Not Another Implementation of Logic!)*. The effort led to the first paper on recursion using the *magic sets* method. See the Datalog Databases section for a discussion of *magic sets*. Other contributions were aggregation in logical rules and theoretical contributions to negation: stratified negation by Van Gelder, well-founded negation by Van Gelder, Ross and Schlipf, (see the Nonstratified Deductive Databases section for a discussion of stratification and well-foundedness) and modularly stratified negation (3).

Implementation efforts at the University of Wisconsin, directed by Ramakrishnan, on the *Coral* DDBs started in the 1980s. Bottom-up and magic set methods were implemented. The declarative query language supports general Horn clauses augmented with complex terms, set-grouping, aggregation, negation, and relations with tuples that contain universally quantified variables.

DATALOG AND EXTENDED DEDUCTIVE DATABASES

The first generalization of relational databases was to permit function-free recursive Horn rules in a database (i.e., rules in which the head of a rule is an atom and the body of a rule is a conjunction of atoms). So, in Formula (1), $n = 1, m \geq 1$ and $l = 0$. These databases are called *DDBs,* or *Datalog* databases.

Datalog Databases

In 1976, van Emden and Kowalski formalized the semantics of logic programs that consist of Horn rules, where the rules are not necessarily function-free. They recognized that the semantics of Horn theories can be characterized in three distinct ways: by model, fixpoint, or proof theory. These three characterizations lead to the same semantics. When the logic program is function-free, their work provides the semantics for *Datalog* databases.

To better understand model theory we need to introduce the concept of *Herbrand universe*, which is the set of constants of the database. The *Herbrand base* is the set of all atoms that can be constructed from the predicates using only elements from the Herbrand universe for the arguments. A set of atoms from the Herbrand base that satisfies all the rules is called a *Herbrand model*.

Model theory deals with the collection of models that captures the intended meaning of the database. Fixpoint theory deals with a fixpoint operator that constructs the collection of all atoms that can be inferred to be *true* from the database. Proof theory provides a procedure that finds answers to queries with respect to the database. van Emden and Kowalski showed that the intersection of all Herbrand models of a Horn *DDB* is the unique minimal model, which is the same as all of the atoms in the fixpoint and are exactly the atoms provable from the theory.

Example 2 (Example of Semantics). *Consider Example 1. The unique minimal Herbrand model of the database is:*

$M = \{p(mike, jack), p(sally, jack), p(katie, mike), p(beverly, mike), p(roger, sally), a(mike, jack), a(sally, jack), a(katie, mike), a(beverly, mike), a(roger, sally), a(katie, jack), a(beverly, jack), a(roger, jack)\}$.

These atoms are all true, *and when substituted into the rules in Example 1, they make all of the rules* true. *Hence, they form a model. If we were to add another fact to the model M, say, p(jack, sally), it would not contradict any of the rules, and it would also be a model. However, this fact can be eliminated because the original set was a model and is contained in the expanded model. That is, minimal Herbrand models are preferred. It is also easy to see that the atoms in M are the only atoms that can be derived from the rules and the data. In Example 3, below, we show that these atoms are in the fixpoint of the database.*

To find if the negation of a ground atom is *true,* one can subtract from the Herbrand base the minimal Herbrand model. If the atom is contained in this set, then it is assumed *false*. Alternatively, answering queries that consist of negated atoms that are ground may be achieved using negation-as-finite failure as described by Clark.

Initial approaches to answering queries in *DDBs* did not handle recursion and were primarily top-down (or backward reasoning). However, answering queries in relational database systems was bottom-up (or forward reasoning) to find all answers. Several approaches were developed to handle recursion, two of which are called the *Alexander* and *magic set* methods, which make use of constants that appear in a query and perform search by bottom-up reasoning. Rohmer, Lescoeur, and Kerisit introduced the Alexander method. Bancilhon, Maier, Sagiv, and Ullman developed the concept of magic sets. These methods take advantage of constants in the query and effectively compute answers using a combined top-down and bottom-up approach. Bry reconciled the bottom-up and top-down methods to compute recursive queries. He showed that the Alexander and magic set methods based on rewriting and methods based on resolution implement the same top-down evaluation of the original database rules by means of auxiliary rules processed bottom-up. In principle, handling recursion poses no additional problems. One can iterate search (referred to as the *naive method)* until a fixpoint is reached, which can be achieved in a finite set of steps because the database has a finite set of constants and is function-free. However, it is unknown how many steps will be required to obtain the fixpoint. The Alexander and magic set methods improve search time when recursion exists, such as for transitive closure rules.

Example 3 (Fixpoint). *The fixpoint of a database is the set of all atoms that satisfy the EDB and the IDB. The fixpoint may be found in a* naive *manner by iterating until no more atoms can be found. Consider Example 1 again.*

Step$(0) = \phi$. *That is, nothing is in the fixpoint.*

Step$(1) = \{p(mike, jack), p(sally, jack), p(katie, mike), p(beverly, mike), p(roger, sally)\}$.

These are all facts, and satisfy r1, r2, r3, r4, and r5. The atoms in Step$(0) \cup$ *Step*(1) *now constitutes a partial fixpoint.*

Step$(2) = \{a(mike, jack), a(sally, jack), a(katie, mike), a(beverly, mike), a(roger, sally)\}$ *are found by using the results of Step*$(0) \cup$ *Step*(1) *on rules r6 and r7. Only rule r6 provides additional atoms when applied. Step*$(0) \cup$ *Step*$(1) \cup$ *Step*(2) *becomes the revised partial fixpoint.*

Step$(3) = \{a(katie, jack), a(beverly, jack), a(roger, jack)\}$, *which results from the previous partial fixpoint. These were obtained from rule r7, which was the only rule that provided new atoms at this step. The new partial fixpoint is Step*$(0) \cup$ *Step*$(1) \cup$ *Step*$(2) \cup$ *Step*(3).

Step$(4) = \phi$. *No additional atoms can be found that satisfy the EDB* $\cup$ *IDB. Hence, the fixpoint iteration may be terminated, and the fixpoint is Step*$(0) \cup$ *Step*$(1) \cup$ *Step*$(2) \cup$ *Step*(3).

Notice that this result is the same as the minimal model M in Example 2.

Classes of recursive rules exist where it is known how many iterations will be required. These rules lead to what has been called *bounded recursion,* noted first by Minker and Nicolas and extended by Naughton and Sagiv. Example 4 illustrates bounded recursion.

Example 4 (Bounded Recursion). *If a rule is* singular, *then it is bound to terminate in a finite number of steps independent of the state of the database. A recursive rule is* singular *if it is of the form*

$$R \leftarrow F \wedge R_1 \wedge \ldots \wedge R_n$$

where F is a conjunction of possibly empty base relations (i.e., empty EDB*) and R,* $R_1, R_2, \ldots, R_n$ *are atoms that have the same relation name if:*

1. *each variable that occurs in an atom* R_i *and does not occur in R only occurs in* R_i;
2. *each variable in R occurs in the same argument position in any atom* R_i *where it appears, except perhaps in at most one atom* R_1 *that contains all of the variables of R.*

Thus, the rule

$$R(X,Y,Z) \leftarrow R(X,Y',Z), R(X,Y,Z')$$

is singular because (a) Y′ and Z′ appear, respectively, in the first and second atoms in the head of the rule (condition 1), and (b) the variables X, Y, Z always appear in the same argument position (condition 2).

The major use of *ICs* has been to assure that a database update is consistent. Nicolas showed how to improve the speed of update, using techniques from *DDBs*. Reiter showed that *Datalog* databases can be queried with or without *ICs* and the answer to the query is identical, which, however, does not preclude the use of *ICs* in the query process. Although *ICs* do not affect the result of a query, they may affect the efficiency to compute an answer. *ICs* provide *semantic* information about the data in the database. If a query requests a join (see RELATIONAL DATABASES) for which there will never be an answer because of the constraints, this can be used to omit trying to answer the query and return the empty answer set and avoids unnecessary joins on potentially large relational databases, or performing a long deduction in a *DDB*. The use of *ICs* to constrain search is called *semantic query optimization (SQO).* McSkimin and Minker were the first to use *ICs* for *SQO* in *DDBs*. Hammer and Zdonik as well as King first applied *SQO* to relational databases. Chakravarthy, Grant, and Minker formalized *SQO* and developed the *partial subsumption algorithm* and method of *residues*, which provide a general technique applicable to any relational or *DDB*. Godfrey, Gryz, and Minker applied the technique bottom-up. Semantic query optimization is being incorporated into relational databases. In *DB2,* cases are recognized when only one answer is to be found and the search is terminated. In other systems, equalities and other arithmetic constraints are being added to optimize search. One can envision the use of join elimination in *SQO* to be introduced to relational technology. One can now estimate when it will be useful to eliminate a join. The tools and techniques already exist and it is merely a matter of time before users and system implementers have them as part of their database systems.

A topic related to *SQO* is that of *cooperative answering systems*. The objective is to give a user the reason why a particular query succeeded or failed. When a query fails, one generally cannot tell why failure occurred. There may be several reasons: The database currently does not contain information to respond to the user or there will never be an answer to the query. The distinction may be useful. *User constraints (UCs)* are related to *ICs*. A user constraint is a formula that models a user's preferences. It may omit answers to queries in which the user has no interest (e.g., stating that, in developing a route of travel, the user does not want to pass through a particular city) or provide other constraints to restrict search. When *UCs* are identical in form to *ICs,* they can be used for this purpose. Although *ICs* provide the semantics of the entire database, *UCs* provide the semantics of the user. *UCs* may be inconsistent with a database. Thus, a separation of these two semantics is essential. To maintain the consistency of the database, only *ICs* are relevant. A query may then be thought of as the conjunction of the original query and the *UCs*. Hence, a query can be semantically optimized based both on *ICs* and *UCs*.

Other features may be built into a system, such as the ability to relax a query that fails, so that an answer to a related query may be found. This feature has been termed *query relaxation*.

The first article on magic sets may be found in Ref. 4. A description of the magic set method to handle recursion in *DDBs* may be found in Refs. 5 and 6. References to work in bounded recursion may be found in Ref. 2. For work on fixpoint theory of *Datalog,* and the work of van Emden and Kowalski, see the book by Lloyd,(7). A comprehensive survey and references to work in cooperative answering systems is in Ref. 8. References to alternative definitions of *ICs,* semantic query optimization, and the method of partial subsumption may be found in Ref. 2.

Stratified Deductive Databases

Logic programs that use default negation in the body of a clause were first used in 1986. Apt, Blair, and Walker, and Van Gelder introduced the concept of stratification to logic programs in which L_1 and the M_j, $1 \leq j \leq m + l$, in Formula (1) are atomic formulas and there is no recursion through negation. They show that there is a unique preferred minimal model, computed from strata to strata. Przymusinski termed this minimal model the *perfect model*. When a theory is stratified, rules can be placed in different strata, where the definition of a predicate in the head of a rule is in a higher stratum than the definitions of predicates negated in the body of the rule. The definition of a predicate is the collection of rules containing the predicate in their head.

Thus, one can compute positive predicates in a lower stratum and a negated predicate's complement is *true* in the body of the clause if the positive atom has not been computed in the lower stratum. The same semantics is obtained regardless of how the database is stratified. When the theory contains no function symbols, the *DDB* is termed $Datalog^{\neg}$. If a database can be stratified, then there is no recursion through negation, and the database is called $Datalog^{\neg}_{strat}$.

Example 5 (Stratified Program). *The rules,*

$r_1 : p \leftarrow q,\ not\ r$

$r_2 : q \leftarrow p$

$r_3 : q \leftarrow s$

$r_4 : s$

$- - - - - - - -$

$r_5 :\ r \leftarrow t$

comprise a stratified theory in which there are two strata. The rule r_5 *is in the lowest stratum, whereas the other rules are in a higher stratum. The predicate p is in a higher stratum than the stratum for r because it depends negatively on r. q is in the same stratum as p because it depends on p. s is also in the same stratum as q. The meaning of the stratified program is that {s, q, p} are* true, *whereas {t, r} are* false. *t is* false *because there is no defining rule for t. As t is* false, *and there is only one rule for r, r is* false. *s is given as* true, *and hence, q is* true. *As q is* true *and r is* false, *from rule* r$_1$, *p is* true.

Current Prototypes

In this section, we discuss two systems that are currently active: *Aditi* and *LDL++*. In addition, relational databases such as *Oracle* and *IBM DB2* have incorporated deductive features from the language *SQL,* discussed in the next subsection.

Aditi. The *Aditi* system has been under development at the University of Melbourne under the direction of Dr. Ramamohanarao. A beta release of the system took place approximately in December 1997. *Aditi* handles stratified databases and recursion and aggregation in stratified databases. It optimizes recursion with magic sets and semi-naive evaluation. The system interfaces with *Prolog*. *Aditi* continues to be developed. Its programming language is Mercury, which contains Datalog as a subset. *Aditi* can handle transactions and has traditional recovery procedures as normally found in commercial databases. There is currently no security-related implementations in the system, but several hooks are available in the system to add these features, which are contemplated for future releases. However, parallel relational operations have not been implemented. It is unclear if *Aditi* will be developed for commercial use.

LDL++. Implementation efforts at *MCC,* directed by Tsur and Zaniolo, started in 1984 and emphasized bottom-up evaluation methods and query evaluation using such methods as seminaive evaluation, magic sets and counting, semantics for stratified negation and set-grouping, investigation of safety, the finiteness of answer sets, and join order optimization. The *LDL* system was implemented in 1988 and released in the period 1989 to 1991. It was among the first widely available *DDBs* and was distributed to universities and shareholder companies of MCC. This system evolved into *LDL++*. No commercial development is currently planned for the system.

To remedy some of the difficulties discovered in applications of LDL, the system called LDL++ was designed in the early 1990s. This system was finally completed as a research prototype in 2000 at UCLA. LDL++ has many innovative features particularly involving its language constructs for allowing negation and aggregates in recursion; its execution model is designed to support data intensive applications, and its application testbed can be used to evaluate deductive database technology on domains such as middleware and data mining. In this summary, we concentrate on two language features. A thorough overview of the system is given in Ref. 9.

A special construct *choice* is used to enforce a functional dependency integrity constraint. Consider the case with student and professor data in which each student has one advisor who must be in the same department as the student. Suppose we have the following facts:

student(jeff, cs)

professor (grant, cs)

professor(minker, cs)

The rule for eligible advisor is

$elig - adv(S,\ P) \leftarrow student(S,\ Major),$
$\quad professor(P,\ Major)$

Thus, we deduce *elig − adv(jeff, grant)* and *elig − adv(jeff, minker)*. However, a student can have only one advisor. The rule for advisor is

advisor(S, P) ← *student(S, Major), professor(P, Major), choice((S), (P)).*

Thus, *choice* enforces the functional dependency

$advisor : S \rightarrow P$

but the result is nondeterministic. It turns out that the use of *choice* in deductive databases has a well-behaved semantics, works well computationally even in the case of stratified negation, and leads to a simple definition of aggregates, including user-defined aggregates.

In the Stratified Deductive Databases section we discussed stratification. LDL++ introduced the notion of an XY-stratified program. In the Background section we gave an example of the computation of *ancestor*. Here we show how to compute ancestors as well as how to count up the number of generations that separate them from the person, *mike,* in this case.

$$delta - anc(0, mike)$$

$$delta - anc(J+1, Y) \leftarrow delta - anc(J, X),$$
$$\quad parent(Y, X),\ not\, all - anc(J, Y)$$

$$all - anc(J+1, X) \leftarrow all - anc(J, X)$$

$$all - anc(J, X) \leftarrow delta - anc(J, X)$$

Assuming additional facts about parents, the query $\leftarrow all -$ *anc(3, X)* will give all great-grandparents (third-generation ancestors) of Mike. This program is not stratified, but it is XY-stratified, has a unique stable model (see the Nonstratified Deductive Databases section), and allows for efficient computation.

SQL:1999

Many techniques introduced within *DDBs* are finding their way into relational technology. The new SQL standards for relational databases are beginning to adopt many of the powerful features of *DDBs*. The *SQL:1999* standard includes queries involving recursion and hence recursive views (10). The recursion must be linear with at most one invocation of the same recursive item. Negation is stratified by allowing it to be applied only to predicates defined without recursion. The naive algorithm must have a unique fixpoint, and it provides the semantics of the recursion; however, an implementation need not use the naive algorithm.

To illustrate the syntax, we show an example of a recursive query. We assume a relation called *family* with attributes *child* and *parent*. The query asks for all the ancestors of John. We write this query in a way that is more complicated than needed, just for illustration, by creating the relation *ancestor* recursively and then using it to find John's ancestors.

```
With Recursive Ancestor(child,anc) as
(Select child, parent
From Family
Union All
    Select Family.child, Ancestor.anc
    From Family, Ancestor
    Where Family.parent = Ancestor.child)
Select anc
From Ancestor
Where child = 'John';
```

The language also allows for the specification of depth-first or breadth-first traversal. Breadth-first traversal would ensure that all parents are followed by all grandparents, and so on.

Also in SQL:1999, a carryover from SQL-92, is a general class of integrity constraints called *Asserts,* which allow for arbitrary relationships between tables and views to be declared. These constraints exist as separate statements in the database and are not attached to a particular table or view. This extension is powerful enough to express the types of integrity constraints generally associated with *DDBs*.

Linear recursion, in which there is at most one subgoal of any rule that is mutually recursive with the head, is currently a part of the client server of *IBM's DB2* system. They are using the *magic sets* method to perform linear recursion. Indications are that the *ORACLE* database system will support some form of recursion.

Summary of Stratified Deductive Database Implementations

As discussed, the modern era of deductive databases started in 1977 with the workshop "Logic and Data Bases" organized in Toulouse, France, that led to the publication of the book *Logic and Data Bases* edited by Gallaire and Minker (11). As of the writing of this article, no commercial deductive databases are available. Among the prototype systems developed, the only ones remaining are *Aditi* and *LDL++*. There are no current plans to make *LDL++* commercially available. *Aditi* may, in the future, be made commercially available, but it is not yet a commercial product.

Deductive databases have had an influence in commercial relational systems. SQL:1999 adopted an approach to SQL recursion that makes use of stratified negation. Thus, the use of recursive rules and stratified deductive databases can be handled to some extent in relational database systems that follow these rules.

There are two possible reasons why deductive databases have not been made commercially available. The first reason is the prohibitive expense to develop such systems. They are more expensive to implement than were relational databases. The second reason is that relational database systems now incorporate deductive database technology, as discussed above.

As more sophisticated applications are developed that are required for *knowledge base systems* (see section on DDB, DDDB, and EDDB Implementations for Knowledge Base Systems), additional tools will be required to handle them. Some tools required for applications may be able to be added to SQL so that they may be incorporated into extensions of relational database technology. For example, adding a capability to provide cooperative answering may be one such tool (see Datalog Databases section). However, other tools needed will be difficult to incorporate into relational technology. For example, handling inconsistent, incomplete, or disjunctive databases are examples of such tools (see DDB, DDDB, and EDDB Implementations for Knowledge Base Systems section). In the remainder of this article, we discuss developments in extending deductive database technology to be able to handle complicated databases, and we also discuss current prototype systems.

Nonstratified Deductive Databases

The theory of stratified databases was followed by permitting recursion through negation in Formula (1) where the L_1 and M_j are atomic formulas, $n = 1$, $m \geq 0$, $l \geq 0$. In the context of *DDBs*, they are called *normal deductive databases*. Many semantics have been developed for these databases. The most prominent are the *well-founded semantics* of Van Gelder, Ross, and Schlipf and the *stable semantics* of Gelfond and Lifschitz. When the *well-founded semantics* is used, the database is called $Datalog^{\neg}_{norm,wfs}$, and when the stable semantics is used, the database is called $Datalog^{\neg}_{norm,\ stable}$. The *well-founded semantics* leads to a unique three-valued model, whereas the *stable semantics* leads to a (possibly empty) collection of models.

Example 6 [Non-Stratified Database]. *Consider the database given by*:

$r_1 : p(X) \leftarrow not\, q(X)$
$r_2 : q(X) \leftarrow not\ p(X)$
$r_3 : r(a) \leftarrow p(a)$
$r_4 : r(a) \leftarrow q(a)$

Notice that r_1 and r_2 are recursive through negation. Hence, the database is not stratified. According to the well-founded semantics, {*p*(*a*), *q*(*a*), *r*(*a*)} *are assigned* unknown. *However, for the* stable model semantics, *there are two minimal stable models:* {{*p*(*a*), *r*(*a*)}, {*q*(*a*), *r*(*a*)}}. *Hence, one can conclude that r*(*a*) *is* true *and the disjunct, p*(*a*) ∨ *q*(*a*), is also true *in the stable model semantics. The stable model semantics has been renamed the* answer set semantics *and throughout the remainder of this article, we will use that term. Because of the importance of this semantics, we discuss it in some detail below in a more expressive context*.

Extended Deductive Databases

The ability to develop a semantics for databases in which rules have a literal (i.e., an atomic formula or the negation of an atomic formula) in the head and literals with possibly negated-by-default literals in the body of a rule, has significantly expanded the ability to write and understand the semantics of complex applications. Such rules, called *extended clauses,* contain rules in Formula (1) where $n = 1, m \geq 0, l \geq 0$, and the *Ls* and *Ms* are literals. Such databases combine *classical negation* (represented by $\neg$) and *default negation* (represented by *not* immediately preceding a literal), and are called *extended deductive databases*. Combining classical and default negation provides users greater expressive power.

The material on answer set semantics is drawn from Ref. 12. For a comprehensive discussion of *ANS* semantics, see Ref. 13. The *ANS* semantics is important for *Knowledge Base Systems (KBS)* semantics.

By an extended deductive database, Π, is meant a collection of rules of the form (1) where $n = 1$ and the M_i, $1 \leq i \leq m + l$, are literals.

The set of all literals in the language of Π is denoted by *Lit*. The collection of all ground literals formed by the predicate *p* is denoted by *Lit(p)*. The semantics of an extended deductive database assigns to it a collection of its *answer sets*—sets of literals that correspond to beliefs that can be built by a rational reasoner on the basis of Π. A literal $\neg p$ is *true* in an answer set *S* if $\neg p \in S$. We say that *not p* is *true* in *S* if $p \notin S$. We say that Π's answer set to a literal query *q* is *yes* if *q* is *true* in all answer sets of Π, *no* if $\neg q$ is *true* in all answer sets of Π, and *unknown* otherwise.

The *answer set* of Π not containing default negation *not* is the smallest (in the sense of set-theoretic inclusion) subset *S* of *Lit* such that

1. if all the literals in the body of a rule of Π are in *S*, then $L_1 \in S$; and
2. if *S* contains a pair of complementary literals, then $S = Lit$. (That is, the extended deductive database is inconsistent.)

Every deductive database that does not contain default negation has a unique answer set, denoted by $b(\Pi)$. The *answer set* $b(\Pi)$ of an extended deductive database Π that contains default negation without variables, and hence is said to be *ground,* is obtained as follows. Let *S* be a candidate answer set and let Π^S be the program obtained from Π by deleting

1. each rule that has a default negation *not L* in its body with $L \in S$, and
2. all default negations *not L* in the bodies of the remaining rules.

It is clear that Π^S does not contain *not,* so that $b(\Pi^S)$ is already defined. If this answer set coincides with *S,* then we say that *S* is an *answer set* of Π. That is, the answer sets of Π are characterized by the equation $S = b(\Pi^S)$.

As an example, consider the extended deductive database Π_1 that consists of one rule:

$\neg q \leftarrow not\ p.$

The rule intuitively states: "q is *false* if there is no evidence that *p* is *true*." The only answer set of this program is $\neg q$. Indeed, here $S = \{\neg q\}$, and $\Pi^S = \{\neg q \leftarrow\}$. Thus, answers to the queries *p* and *q* are *unknown* and *false*, respectively.

There have been several implementations that incorporate the well-founded and the answer set semantics. These systems can handle large knowledge bases that consist of data facts and rules. In addition, in the following section, we discuss the extension of deductive systems to handle incomplete information and disjunctive information. There have been implementations of such systems. In the DDB, DDDB, and EDDB Implementations for Knowledge Base Systems section, we describe the most important systems that contain the well-founded semantics, the answer set semantics, or the disjunctive semantics.

DISJUNCTIVE DEDUCTIVE DATABASES

The Need for Disjunction

In the databases considered so far, information is definite. However, many applications exist where knowledge of the world is *incomplete*. For example, when a null value appears as an argument of an attribute of a relation, the value of the attribute is unknown. Also, uncertainty in databases may be represented by probabilistic information. Another area of incompleteness occurs when it is unknown which among several facts are *true,* but it is known that one or more are *true*. It is, therefore, necessary to be able to represent and understand the semantics of theories that include incomplete data. The case in which there is disjunctive information is discussed below. A natural extension is to permit disjunctions in the *EDB* and disjunctions in the heads of *IDB* rules. These rules are represented in Formula(1), where $n \geq 1$, $m \geq 0$, and $l \geq 0$, and are called *extended disjunctive rules*. Such databases are called *extended disjunctive deductive databases (EDDDBs),* or $Datalog^{\neg}_{disj,ext}$. Below, we illustrate a knowledge base system that contains disjunctive information, logical negation ($\neg$), and default negation *(not)*.

Example 7 (Knowledge Base (13)). *Consider the database, where p(X,Y) denotes* X is a professor in department Y, *a(X, Y) denotes* individual X has an account on machine Y, *ab(W, Z) denotes* it is abnormal in rule W to be individual Z.

We wish to represent the following information where mike *and* john *are* professors *in the* computer science department*:*

1. *As a rule, professors in the computer science department have* m1 accounts. *This rule is not applicable to Mike, represented by* ab(r4,mike) *(that is, that it is abnormal that in rule r4 we have mike). He may or may not have an account on that machine.*
2. *Every computer science professor has either an* $m1$ or an $m2$ *account, but not both.*

These rules are reflected in the following extended disjunctive database.

r1 $p(mike, cs) \leftarrow$

r2 $p(john, cs) \leftarrow$

r3 $\neg p(X, Y) \leftarrow not\ p(X, Y)$

r4 $a(X, m1) \leftarrow p(X, cs),\ not\, ab(r4, X),\ not \neg a(X, m1)$

r5 $ab(r4, mike) \leftarrow$

r6 $a(X, m1) \vee a(X, m2) \leftarrow p(X, cs), ab(r4, X)$

r7 $\neg a(X, m2) \leftarrow p(X, cs), a(X, m1)$

r8 $\neg a(X, m1) \leftarrow p(X, cs), a(X, m2)$

r9 $a(X, m2) \leftarrow \neg a(X, m1), a(X, cs)$

Rule r3 states that if by default negation p(X, Y) fails, then p(X, Y) is logically false. *The other rules encode the statements listed above. From this formalization, one can deduce that* john *has an* m1 *account, whereas* mike *has either an* m1 *or an* m2 *account, but not both.*

The semantics of *DDDBs* is discussed first, where clauses are given by Formula (1), literals are restricted to atoms, and there is no default negation in the body of a clause. Then the semantics of *EDDDBs,* where there are no restrictions on clauses in Formula (1), is discussed.

Disjunctive Deductive Databases (DDDBs)

The field of *disjunctive deductive databases (DDDBs),* referred to as $Datalog^{\neg}_{disj}$, started in 1982 by Minker who described how to answer both positive and negated queries in such databases. A major difference between the semantics of *DDBs* and *DDDBs* is that *DDBs* usually have a unique minimal model, whereas *DDDBs* generally have multiple minimal models.

To answer positive queries over *DDDBs*, it is sufficient to show that the query is satisfied in every minimal model of the database. Thus, for the *DDDB,* $\{a \vee b\}$, there are two minimal models, {{a}, {b}}. The query a is not satisfied in the model {*b*}, and hence, it cannot be concluded that a is *true*. However, the query $\{a \vee b\}$ is satisfied in both minimal models and hence the answer to the query $\{a \vee b\}$ is *yes*. To answer negated queries, it is not sufficient to use Reiter's *CWA* because, as he noted, from $DB = \{a \vee b\}$, it is not possible to prove a, and it is not possible to prove b. Hence, by the *CWA, not a* and *not b* follow. But, {*a* ∨ *b, not a, not b*} is not consistent. The *Generalized Closed World Assumption (GCWA),* developed by Minker, resolves this problem by specifying that a negated atom is *true* if the atom does not appear in any minimal model of the database, which provides a model theoretic definition of negation. An equivalent proof theoretic definition, also by Minker, is that an atom a is considered *false* if whenever $a \vee C$ is proved *true,* then C can be proven *true,* where C is an arbitrary positive clause.

Answering queries in *DDDBs* has been studied by several individuals. Fernández and Minker developed the concept of a *model tree,* a tree whose nodes consist of atoms. Every branch of the model tree is a model of the database. They show how one can incrementally compute sound and complete answers to queries in *hierarchical DDDBs,* where the database has no recursion. However, one can develop a fixpoint operator over trees to capture the meaning of a *DDDB* that includes recursion. Fernández and Minker compute the model tree of the extensional *DDDB* once. To answer queries, *intensional database* rules may be invoked. However, the models of the extensional disjunctive part of the database do not have to be generated for each query. Their approach to compute answers generalizes to stratified and normal *DDDBs*.

Fernández and Minker also developed a fixpoint characterization of the minimal models of disjunctive and stratified disjunctive deductive databases. They proved that the operator iteratively constructs the perfect models semantics (Przymusinski) of *stratified DDBs*. Given the equivalence between the *perfect model semantics of stratified programs* and *prioritized circumscription* as shown by Przymusinski, their characterization captures the meaning of the corresponding circumscribed theory. They present a bottom-up evaluation algorithm for *stratified DDDBs*. This algorithm uses the *model-tree* data structure to compute answers to queries.

Loveland and his students have developed a top-down approach when the database is *near Horn,* that is, there are few disjunctive statements. They developed a case-based reasoner that uses *Prolog* to perform the reasoning and introduced a relevancy detection algorithm to be used with *SATCHMO,* developed by Manthey and Bry, for automated theorem proving. Their system, termed *SATCHMORE (SATCHMO with RElevancy),* improves on *SATCHMO* by limiting uncontrolled use of forward chaining. There are currently several efforts devoted to implementing disjunctive deductive databases from a bottom-up approach, prominent among these is the system DLV discussed in the next section.

Alternative semantics were developed for nonstratifiable *normal DDDBs* by: Ross (the *strong well-founded semantics);* Baral, Lobo, and Minker (*Generalized Disjunctive Well-Founded Semantics* (*GDWFS*)); Przymusinski (*disjunctive answer set semantics*); Przymusinski (*stationary semantics*); and Brass and Dix (*D-WFS semantics*). Przymusinski described a *semantic framework* for disjunctive logic programs and introduced the *static expansions* of disjunctive programs. The class of static expansions extends both the classes of answer sets, well-founded and stationary models of normal programs, and the class of minimal models of disjunctive programs. Any static expansion of a program P provides the corresponding semantics for P consisting of the set of all sentences logically implied by the expansion. The D-WFS semantics permits a general approach to bottom-up computation in disjunctive programs. The Answer set semantics has been modified to apply to disjunctive as well as extended DDBs. Answer set semantics has become the most used semantics for these types of databases. The semantics encompasses the answer set semantics, and hence the Smodels semantics. We discuss this semantics in the following subsection.

Answer Set Semantics for EDDDBs

A disjunctive deductive database is a collection of rules of the form (1) where the Ls and Ms are literals. When the Ls and Ms are atoms, the program is called a *normal disjunctive program*. When $l = 0$ and the Ls and Ms are atoms, the program is called a *positive disjunctive deductive database*.

An *answer set* of a disjunctive deductive database Π not containing *not* is a smallest (in a sense of set-theoretic inclusion) subset S of *Lit* such that

1. for any rule of the form (1), if $M_1, \ldots, M_m \in S$, then for some i, $0 \leq i \leq n$, $L_i \in S$; and
2. If S contains a pair of complementary literals, then $S = Lit$ (and hence is inconsistent).

The answer sets of a disjunctive deductive database that does not contain *not* is denoted as $\alpha(S)$. A disjunctive deductive database without *not* may have more than one answer set.

A set of literals S is said to be an answer set of a disjunctive deductive database Π if $S \in \alpha(\Pi^S)$, where Π^S is defined in the Extended Deductive Databases section.

Consider the disjunctive deductive database,
$\Pi_0 = p(a) \vee p(b) \leftarrow .$
This deductive database has two answer sets: $\{p(a)\}$ and $\{p(b)\}$.
The disjunctive deductive database,
$\Pi_1 = \Pi_0 \cup \{r(X) \leftarrow not\ p(X)\}$,
has two answer sets: $\{p(a), r(b)\}$ and $\{p(b), r(a)\}$.

Selecting Semantics for EDDBs and EDDDBS

There are a large number of different semantics, in addition to those listed here. A user who wishes to use such a system is faced with the problem of selecting the appropriate semantics for his needs. No guidelines have been developed. However, one way to assess the semantics desired is to consider the complexity of the semantics. Results have been obtained for these semantics by Schlipf and by Eiter and Gottlob.

Ben-Eliahu and Dechter showed that there is an interesting class of disjunctive databases that are tractable. In addition to work on tractable databases, consideration has been given to approximate reasoning where one may give up soundness or completeness of answers. Selman and Kautz developed lower and upper bounds for Horn (Datalog) databases, and Cadoli and del Val developed techniques for approximating and compiling databases.

A second way to determine the semantics to be used is through their properties. Dix proposed criteria that are useful to consider in determining the appropriate semantics to be used. Properties deemed to be useful are: *elimination of tautologies,* where one wants the semantics to remain the same if a tautology is eliminated; *generalized principle of partial evaluation,* where if a rule is replaced by a one-step deduction, the semantics is unchanged; *positive / negative reduction; elimination of nonminimal rules,* where if a subsumed rule is eliminated, the semantics remains the same; *consistency,* where the semantics is not empty for all disjunctive databases; and *independence,* where if a literal l is *true* in a program P and P' is a program whose language is independent of the language of P, then l remains *true* in the program consisting of the union of the two languages.

A semantics may have all the properties that one may desire, and be computationally tractable and yet not provide answers that a user expected. If, for example, the user expected an answer *r(a)* in response to a query *r(X),* and the semantics were, for Example 6, the *well-founded semantics,* the user would receive the answer, *r(a)* is *unknown*. However, if the *answer set semantics* had been used, the answer returned would be *r(a)*. Perhaps the best that can be expected is to provide users with complexity results and criteria by which they may decide which semantics meets the needs of their problems. However, to date, the most important semantics have been the answer set semantics and the well-founded semantics, discussed earlier.

Understanding the semantics of disjunctive theories is related to nonmonotonic reasoning. The field of nonmonotonic reasoning has resulted in several alternative approaches to perform default reasoning. Hence, *DDDBs* may be used to compute answers to queries in such theories. Cadoli and Lenzerini developed complexity results concerning circumscription and closed world reasoning. Przymusinski and Yuan and You describe relationships between autoepistemic circumscription and logic programming. Yuan and You use two different belief constraints to define two semantics the *stable circumscriptive semantics*, and *the well-founded circumscriptive semantics* for autoepistemic theories.

References to work by Fernández and Minker and by Minker and Ruiz may be found in Ref. 2. Work on complexity results appears in Schlipf (15) and in Eiter and Gottlob Refs. (16,17). Relationships between $Datalog^{\neg}_{ext}$ and nonmonotonic theories may be found in Ref. 2. Prototype implementations of extended deductive and extended disjunctive deductive databases are given in the following section.

DDB, DDDB, AND EDDB IMPLEMENTATIONS FOR KNOWLEDGE BASE SYSTEMS

General Considerations

Chen and Warren implemented a top-down approach to answer queries in the well-founded semantics, whereas Leone and Rullo developed a bottom-up method for $Datalog^{\neg}_{norm,wfs}$ databases. Several methods have been developed for computing answers to queries in answer set semantics. Fernández, Lobo, Minker, and Subrahmanian developed a bottom-up approach to compute answers to queries in answer set semantics based on the concept of *model trees*. Bell, Nerode, Ng, and Subrahmanian developed a method based on linear programming.

These notions of default negation have been used as separate ways to interpret and to deduce default information. That is, each application chose one notion of negation and applied it to every piece of data in the domain of the application. Minker and Ruiz defined a more expressive *DDB* that allows several forms of default negation in the same database. Hence, different information in the domain may be treated appropriately. They introduced a new semantics called the *well-founded stable* semantics that characterizes the meaning of *DDBs* that combine well-founded and stable semantics.

Knowledge bases are important for artificial intelligence and expert system developments. A general way to represent knowledge bases is through logic. Work developed for *extended DDBs* concerning semantics and complexity apply directly to knowledge bases. For an example of a knowledge base, see Example 7. *Extended DDBs* permit a wide range of *knowledge bases (KBs)* to be implemented.

Since alternative *extended DDBs* have been implemented, the *KB* expert can focus on writing rules and integrity constraints that characterize the problem, selecting the semantics that meets the needs of the problem, and employing a *DDB* system that uses the required semantics.

Articles on stratified databases by Apt, Blair, and Walker, by Van Gelder, and by Przymusinski may be found in Ref. 18. See Refs. 5 and 6 for a description of computing answers to queries in stratified databases. For an article on the semantics of $Datalog^{\neg}_{wfs}$, see Ref. 19 see Ref. 20 for the answer set semantics; see Ref. 2 for references to work on other semantics for normal extended deductive databases and Schlipf (21) for a comprehensive survey article on complexity results for deductive databases. For results on negation in deductive databases, see the survey article by Shepherdson (22). The development of the semantics and complexity results of extended DDBs that permit a combination of classical negation and multiple default negations in the same DDB are important contributions to database theory. They permit wider classes of applications to be developed.

There have been several implementations of systems for handling extended databases and extended disjunctive databases. There have also been many semantics proposed for these systems. However, of these systems, the most important systems are the *well-founded semantics* (WFS) and the *answer set semantics* (ANS). See Ref. 2 for a discussion of the alternate proposals. There is one major system developed and in use for the WFS, and there are several implementations for the ANS.

Implementation of the Well-Founded Semantics

Warren, Swift, and their associates (23) developed an efficient deductive logic programming system, *XSB,* that computes the well-founded semantics. *XSB* is supported to the extent of answering questions and fixing bugs within the time schedule of the developers. The system extends the full functionality of *Prolog* to the *WFS*. *XSB* forms the core technology of a start-up company, XSB, Inc., whose current focus is application work in data cleaning and mining. In Ref. 24, it is shown how nonmonotonic reasoning may be done within *XSB* and describes mature applications in medical diagnosis, model checking, and parsing. XSB also permits the user to employ Smodels, discussed below. XSB is available on the Internet and is available as open-source. There is no intent by XSB, Inc. to market the program.

Implementation of Answer Set Semantics

Three important implementations of answer set semantics are by Marek and Truszczyński (25), by Niemelä and Simons (26–28), and by Eiter and Leone (29,30). Marek and Truszczyński developed a program, *Default Reasoning System (DeReS),* that implements Reiter's default logic. It computes extensions of default theories. As logic programming with answer set semantics is a special case of Reiter's default logic, *DeReS* also computes the answer sets of logic programs. To test *DeReS,* a system, called *TheoryBase,* was built to generate families of large default theories and logic programs that describe graph problems such as existence of colorings, kernels, and Hamiltonian cycles. No further work is anticipated on the system.

Niemelä and Simons developed a system, *Smodels,* to compute the answer sets of programs in *Datalog* with negation. At present, *Smodels* is considered the most efficient implementation of answer set semantics computation. *Smodels* is based on two important ideas: intelligent grounding of the program, limiting the size of the grounding, and use of the *WFS* computation as a pruning technique. The system is used at many sites throughout the world. New features are continually being added. The system is available for academic use. It is possible to license the system from a company in Finland called Neotide.

Implementation of Disjunctive Deductive Databases

Eiter and Leone developed a system, *DLV (DataLog with Or),* that computes answer sets (in the sense of Gelfond and Lifschitz) for disjunctive deductive databases in the syntax generalizing *Datalog* with negation. As *Smodels, DLV* also uses a very powerful grounding engine and some variants of *WFS* computation as a pruning mechanism. The method used to compute *disjunctive answer sets* is described in Ref. 31. The work is the joint effort between Tech U, Austria and U Calabria, Italy. Many optimization techniques have been added to the system (e.g., magic sets and new heuristics), the system language has been enhanced (e.g., aggregate functions), and new front ends were developed for special applications such as planning.

Definition of a Knowledge Base System

Knowledge bases are important for artificial intelligence and expert system developments. A general way to represent knowledge bases is through logic. All work developed for extended *DDBs* concerning semantics and complexity apply directly to knowledge bases. Baral and Gelfond (14) describe how extended *DDBs* may be used to represent knowledge bases. Many papers devoted to knowledge bases consider them to consist of facts and rules, which is certainly one aspect of a knowledge base, as is the ability to extract proofs. However, integrity constraints supply another aspect of knowledge and differentiate knowledge bases that may have the same rules but different integrity constraints. Since alternative extended deductive databases have been implemented, knowledge base experts can focus on the specification of the rules and integrity constraints and employ the extended deductive databases that have been implemented.

Work on the implementation of semantics related to extended disjunctive deductive databases has been very impressive. These systems can be used for nonmonotonic reasoning. Brewka and Niemela (32) state as follows:

> At the plenary panel session[2], the following major trends were identified in the field: First, serious systems for nonmonotonic reasoning are now available (XSB, SMODELS, DLV). Second, people outside the community are starting to use these systems with encouraging success (for example, in planning). Third, nonmonotonic techniques for reasoning about action are used in highly ambitious long-term projects (for example, the WITAS Project, www.ida.liu.se/ext/witas/eng.html). Fourth, causality is still an important issue; some formal models of causality have surprisingly close connections to standard nonmonotonic techniques. Fifth, the nonmonotonic logics being used most widely are the classical ones: default logic, circumscription, and autoepistemic logic.

APPLICATIONS

There are many applications that cannot be handled by relational database technolgy, but are necessary for advanced knowledge base and artificial intelligence applications (AI). One cannot handle disjunctive information, databases that have default negation that are not stratified, incomplete information, planning for robotics, handling databases with preferences, and other topics. However, all of the above topics can be handled by extensions to databases as described in the previous sections. In this section, we discuss how the following topics can be handled by deductive databases: data integration in which one can combine databases; handling preferences in databases; updating deductive databases; AI planning problems; and handling databases that may be inconsistent. The applications described can be incorporated or implemented on the deductive databases discussed in the previous section. These applications form a representative sample of capabilities that can be implemented with extended deductive database capabilities.

Two specific examples of deductive database applications are briefly discussed here. Abduction is a method of reasoning, which, given a knowledge base and one or more observations, finds possible explanations of the observations in terms of predicates called abducible predicates. The concept of abduction has been used in such applications as law, medicine, diagnosis, and other areas; Refs. 14 and 33. With the vast number of heterogeneous information sources now available, particularly on the world wide web, multiagent systems of information agents have been proposed for solving information retrieval problems, which requires advanced capabilities to address complex tasks, query planning, information merging, and handling incomplete and inconsistent information. For a survey of applications to intelligent information agents see Ref. 34.

Data Integration

Data integration deals with the integration of data from different databases and is a relevant issue when many overlapping databases are in existence. In some cases, various resources exist that are more efficient to access than the actual relations or, in fact, the relations may even be virtual so that all data must be accessed through resources. We review here the approach given in Ref. 35. The basis of this approach is the assumption that the resources are defined by formulas of deductive databases and that integrity constraints can be used, as in SQO, to transform a query from the extensional and intensional predicates to the resources. Grant and Minker show how to handle arbitrary constraints, including the major types of integrity constraints, such as functional and inclusion dependencies. Negation and recursion are also discussed in this framework. We use a simple example to illustrate the basic idea.

Assume that there are three predicates:

$p_1(X, Y, Z), p_2(X, U)$, and $p_3(X, Y)$,

and an integrity constraint:

$p_3(X, Y) \leftarrow p_1(X, Y, Z), Z > 0.$

The resource predicate is defined by the formula

$r(X, Y, Z) \leftarrow p_1(X, Y, Z), p_2(X, U).$

Let the query be:

$p_1(X, Y, Z), p_2(X, U), p_3(X, Y), Z > 1.$

Intuitively one can see from the integrity constraint that p_3 is superfluous in the query, hence is not needed, and therefore the resource predicate can be used to answer the query. Formally, the first step involves reversing the resource rules to define the base predicates, p_1 and p_2 in this case, in terms of the resource predicates. This step is justified by the fact that the resource predicate definition really represents an if-and-only-if definition and is the Clark completion, as mentioned in the Deductive Data-

[2] The plenary panel session was held at the Seventh International Workshop on Nonmonotonic Reasoning as reported in (32).

bases: The Formative Years 1969–1978 section. In this case, the rules are:

$$p_1(X,Y,Z) \leftarrow r(X,Y,Z)$$
$$p_2(X,f(X,Y,Z)) \leftarrow r(X,Y,Z).$$

It is possible then to use resolution theorem proving, starting with the query, the integrity constraint, and these new rules, to obtain a query in terms of r, namely

$$\leftarrow r(X,Y,Z), Z > 1.$$

That is, to answer the original query, a conjunction of three predicates and a select condition, one need only perform a select on the resource predicate.

Handling Preferences

In a standard deductive database, all the clauses are considered to have equal status. But, as was shown, there may be several answer sets for a deductive database. In some cases, one answer set may be preferred over another because there may be preferences among the clauses. Consider the following simple propositional example with two clauses.

r1 : $a \leftarrow not\ b$
r2 : $b \leftarrow not\ a$

There are two answer sets: $\{a\}$ and $\{b\}$. If $r1$ is preferred over $r2$, then the preferred answer set is $\{a\}$.

The general approach to handling preferences has been to use a meta-formalism to obtain preferred answer sets. One way is to generate all answer sets and then select the ones that are preferred. A somewhat different approach is taken in Ref. 36 and is briefly reviewed here.

They start with the original deductive database and using the preferences between clauses transform (compile) them to another deductive database such that the answer sets of the new (tagged) deductive database are exactly the preferred answer sets of the original deductive database. For the simple propositional example with two clauses given above, the new deductive database (actually a simplified version) would be:

$ok(r1) \leftarrow$
$ok(r2) \leftarrow ap(r1)$
$ok(r2) \leftarrow bl(r1)$
$ap(r1) \leftarrow ok(r1),\ not\ b$
$ap(r2) \leftarrow ok(r2),\ not\ a$
$a \leftarrow ap(r1)$
$b \leftarrow ap(r2)$
$bl(r1) \leftarrow b$
$bl(r2) \leftarrow a$

There are three new predicates: ok, ap (for applicable), and bl (for blocked). The first three rules reflect the preference of r1 over r2. The next four rules are obtained from r1 and r2 by adding the ok of the appropriate rules and deducing a (resp. b) in two steps. The last two rules show when a rule is blocked. There is one answer set in this case:

$\{ok(r1),\ ap(r1),\ a,\ bl(r2),\ ok(r2)\}$
whose set of original atoms is $\{a\}$.

This article handles both static rules and dynamic rules, that is, rules that are themselves considered as atoms in rules. The implementation is straightforward and the complexity is not higher than for the original deductive database.

Updates

Although the study of deductive databases started in 1977, as discussed in the Deductive Databases section, for many years it dealt with static databases only. More recently, researchers have tried to incorporate update constructs into deductive databases. We briefly consider here an extension of Datalog, called *DatalogU* (37). This extension has the capability to apply concurrent, disjunctive, and sequential update operations in a direct manner and it has a clear semantics.

We illustrate DatalogU by giving a couple of simple examples. Consider a predicate *employee* with three arguments: name, department, and salary.

To insert a new employee, *joe* into department *5* with salary *45000,* we write *insertemployee(joe,* 5,45000)
To delete all employees from department 7, write
deleteemployee(N, 7, *S)*
Consider now the update that gives every employee in department 3 a 5% raise *deleteemployee(N,* 3, *S*), $S' = S \times 1.05$, *insertemployee(N,* 3, *S'*)
DatalogU allows writing more general rules, such as
$raiseSalary(N,D,P) \leftarrow deleteemployee(N,D,S), S' = S \times (1+P),\ insertemployee(N,D,S')$
that can be used, such as
raiseSalary(N, D, 0.1)
to give everyone a 10% raise.

Another example is the rule for transferring money from one bank account to another. We can use the following rules:

$transfer(BA1,\ BA2,\ Amt) \leftarrow withdraw(BA1,\ Amt),$
$\quad deposit(BA2,\ Amt), BA1 \neq BA2$
$withdraw(BA, Amt) \leftarrow deleteaccount(BA, Bal),$
$\quad insertaccount(BA, Bal'), Bal \geq Amt, Bal' = Bal - Amt$
$deposit(BA, Amt) \leftarrow deleteaccount(BA, Bal),$
$\quad insertaccount(BA, Bal'), Bal' = Bal + Amt$

So now, to transfer $200 from account A0001 to A0002, we write *transfer* (*A*0001, *A*0002, 200).

AI Planning

Planning in AI can be formalized using deductive databases, as shown in Ref. 38. A planning problem can be described in terms of an initial state, a goal state, and a set of possible actions. A successful plan is a sequence of actions that, starting with the initial state and applying the actions in the given sequence, leads to the goal. Each action consists of preconditions before the action can take place as well as the addition and deletion of atoms leading to postconditions that hold after the action has taken place. The following simple example for illustration involves a world of named blocks stacked in columns on a table. Consider the action *pickup (X)* meaning that the robot picks up X. In this case,

$$
\begin{aligned}
&Pre = \{onTable(X), clear(X), handEmpty\}\\
&Post = \{\neg onTable(X),\ \neg handEmpty, holding(X)\}
\end{aligned}
$$

So before the robot can pick up a block, the robot's hand must be empty and the block must be on the table with no block above it. After the robot picks up the block, it is holding it, its hand is not empty, and the block is not on the table.

The idea of the formalization is to write clauses representing postconditions, such as

$$postcond(pickup(X), onTable(X), neg)$$

and use superscripts to indicate the step number in the action, such as in

$$add^J(Cond) \leftarrow\ fired^J(\alpha), postcond(\alpha, Cond, pos)$$

to indicate, for instance, that if an action α is fired at step J, then a particular postcondition is added. A tricky issue is that at each step only one action can be selected among all the firable actions. A nondeterministic choice operator (see also Current Prototype section on LDL++) is used to express this concept:

$$fired^J(\alpha) \leftarrow\ firable^J(\alpha),\ \neg\ firable^J(end), choice^J(\alpha)$$

We have left out many details, but the important issue is that the existence of a solution for a planning problem is equivalent to the existence of a model in the answer set semantics for the planning deductive database and the conversion of the problem to the deductive database can be done in linear time. In fact, each answer set represents a successful plan. These results are also generalized in various ways, including to parallel plans.

The oldest problem in AI planning, the ability of an agent to achieve a specified goal, was defined by McCarthy (39). He devised a logic-based approach to the problem based on the situation calculus that was not entirely satisfactory. One of the problems with the approach was how to handle the *frame problem,* the seeming need for *frame axioms,* to represent changes in the situation calculus. Lifschitz et al. (40) have shown how to solve this problem. They state, "The discovery of the frame problem and the invention of the nonmonotonic formalisms that are capable of solving it may have been the most significant events so far in the history of reasoning about actions." See the paper by Litschitz et al. for their elegant solution to the McCarthy AI planning problem.

Handling Inconsistencies

As explained in The Background section, a database must satisfy its integrity constraints. However, there may be cases where a database becomes inconsistent. For example, a database obtained by integrating existing databases (see Data Integration section) may be inconsistent even though each individual database is consistent. Reference 41 presents a comprehensive approach to dealing with inconsistent databases through giving consistent answers and computing repairs, which is accomplished by transforming the database into extended disjunctive database rules; the answer sets of this database are exactly the repairs of the inconsistent database. We note that the basic definitions for querying and repairing inconsistent databases were introduced in Ref. 47.

We illustrate the technique on a simple example. Let

$$EDB = \{p(a), p(b), q(a), q(c)\} \text{ and } IC = \{q(x) \leftarrow p(x)\}.$$

This database is inconsistent because it contains *p(b)* but not *q(b)*. There are two simple ways of repairing the database: (1) delete *p(b),* (2) insert *q(b)*. However, even though the database is inconsistent, certain queries have the same answers in both repairs. For example, $\leftarrow q(x), not\ p(X)$ has c as its only answer. In this example, the transformation modifies the *IC* to the rule

$$\neg\ p_u(X) \vee q_u(X) \leftarrow p(X), not\, q(X)$$

where p_u and q_u are new "update" predicates. The meaning of this rule is that in case $p(X)$ holds and $q(X)$ does not hold, either delete $p(X)$ or insert $q(X)$. The answer sets of the transformed database are

$M_1 = \{p(a), p(b), q(a), q(c), \neg p_u(p)\}$ and $M_2 = \{p(a), p(b), q(a), q(c), q_u(b)\}$ leading to the two repairs mentioned above.

SUMMARY AND REFERENCES

The article describes how the rule of inference, based upon the Robinson Resolution Principle (developed by J.A. Robinson (42)), started in 1968 with the work of Green and Raphael (43,44), led to a number of systems and culminated in the start of the field of deductive databases in November 1977, with a workshop held in Toulouse, France that resulted in the appearance of a book edited by Gallaire and Minker (11).

The field has progressed rapidly and has led to an understanding of negation, has provided a theoretical framework so that it is well understood what is meant by a query, and an answer to a query. The field of relational databases is encompassed by the work in *DDBs*. As discussed, some concepts introduced in deductive databases have been incorporated into relational technology. As noted, complex knowledge based systems can be implemented using advanced deductive database concepts not contained in relational technology. There are, however, many different kinds of *DDBs* as described in this article. Theoretical results concerning fixpoint theory for *DDBs* may be found in Lloyd (7), while fixpoint theory and theories of negation for disjunctive deductive databases may be found in Lobo, Minker, and Rajasekar (45). Complexity results have not been summarized in this article. The least complex *DDBs* are, in order, *Datalog*, $Datalog^{\neg}_{str}$, $Datalog^{\neg}_{wfs}$, and $Datalog^{\neg}_{stab}$. The first three databases result in unique minimal models. Other databases are more complex and, in addition, there is no current semantics that is uniformly agreed on for $Datalog_{disj}$. However, the answer set semantics discussed in the Extended Deductive Databases section for deductive databases and in the Answer Set Semantics for EDDDBs section for disjunctive deductive databases appears to be the semantics generally favored. As noted in the Selecting Semantics for EDDBs and EDDDBs section, a combination of properties of *DDBs,* developed by Dix and discussed in Ref. 46, (and the complexity of these systems as described in Ref. 15–17), could be used once such systems are developed.

BIBLIOGRAPHY

1. J. Minker, Perspectives in deductive databases, *Journal of Logic Programming*, **5**: 33–60, 1988.
2. J. Minker, Logic and databases: a 20 year retrospective, in D. Pedreschi and C. Zaniolo (eds.), *Logic in Databases*, Proc. Int. Workshop LID'96, San Miniato, Italy, 1996, pp. 3–57.
3. K.A. Ross, Modular stratification and magic sets for datalog programs with negation, *Proc. ACM Symp. on Principles of Database Systems*, 1990.
4. F. Bancilhon, D. Maier, Y. Sagiv, and J. Ullman, Magic sets and other strange ways to implement logic programs, *Proc. ACM Symp. on Principles of Database Systems*, 1986.
5. J.D. Ullman, *Principles of Database and Knowledge-Base Systems I*. Principles of Computer Science Series, Rockville, MD: Computer Science Press, 1988.
6. J.D. Ullman, *Principles of Database and Knowledge-Base Systems II*, Principles of Computer Science Series, Rockville, MD: Computer Science Press, 1988.
7. J.W. Lloyd, *Foundations of Logic Programming*. New York: 2nd ed. Springer-Verlag, 1987.
8. T. Gaasterland, P. Godfrey, and J. Minker, An overview of cooperative answering, *Journal of Intelligent Information Systems*, **1** (2): 123–157, 1992.
9. F. Arni, K. Ong, S. Tsur, H. Wang, and C. Zaniolo, The deductive database system ldl++, *Theory and Practice of Logic Programming*, **3**: 61–94, January 2003.
10. J. Melton and A. R. Simon, *SQL:1999 Understanding Relational Language Components*, San Francisco: Morgan Kaufmann, 2002.
11. H. Gallaire and J. Minker, (eds.), *Logic and Data Bases*, Plenum Press, New York: 1978.
12. T. Gaasterland and J. Lobo, Qualified answers that reflect user needs and preferences, *International Conference on Very Large Databases*, 1994.
13. C. Baral, *Knowledge representation, reasoning and declarative problem solving*, Cambridge, MA: Cambridge University Press, 2003.
14. C. Baral, and M. Gelfond, Logic programming and knowledge representation, *Journal of Logic Programming*, **19**/**20**: 73–148, 1994.
15. J.S. Schlipf, A survey of complexity and undecidability results in logic programming, in H. Blair, V.W. Marek, A. Nerode, and J. Remmel, (eds.), *Informal Proc. of the Workshop on Structural Complexity and Recursion-theoretic Methods in Logic Programming.*, Washington, D.C.1992, pp. 143–164.
16. T. Eiter and G. Gottlob, Complexity aspects of various semantics for disjunctive databases, *Proc. of the Twelfth ACM SIGART–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS-93)*, May 1993, pp. 158–167.
17. T. Eiter and G. Gottlob, Complexity results for disjunctive logic programming and application to nonmonotonic logics, D. Miller, (ed.), *Proc. of the International Logic Programming Symposium ILPS'93*, Vancouver, Canada, 1993, pp. 266–278.
18. J. Minker, (ed.), *Foundations of Deductive Databases and Logic Programming*, New York: Morgan Kaufmann, 1988.
19. A. VanGelder, K. Ross, and J.S. Schlipf, Unfounded Sets and Well-founded Semantics for General Logic Programs, *Proc. 7th Symposium on Principles of Database Systems*, 1988, pp. 221–230 .
20. M. Gelfond and V. Lifschitz, The Stable Model Semantics for Logic Programming, R.A. Kowalski and K.A. Bowen, (eds.), *Proc. 5th International Conference and Symposium on Logic Programming*, Seattle, WA, 1988, pp. 1070–1080.
21. J.S. Schlipf, Complexity and undecidability results for logic programming, *Annals of Mathematics and Artificial Intelligence*, **15** (3–4): 257–288, 1995.
22. J.C. Shepherdson, Negation in Logic Programming, in J. Minker, (ed.), *Foundations of Deductive Databases and Logic Programming*, New York: Morgan Kaufman, 1988, pp. 19–88.
23. P. Rao, K. Sagonas, T. Swift, D.S. Warren, and J. Friere, XSB: A system for efficiently computing well-founded semantics, in J. Dix, U. Ferbach, and A. Nerode, (eds.), *Logic and Nonmonotonic Reasoning - 4th International Conference, LPNMR '97*, Dagstuhl Castle, Germany, 1997, pp. 430–440.
24. T. Swift, Tabling for non-monotonic programming, technical report, SUNY Stony Brook, Stony Brook, NY: 1999.
25. P. Cholewiński, W. Marek, A. Mikitiuk, and M. Truszczyński, Computing with default logic, *Artificial Intelligence*, **112**: 105–146, 1999.
26. I. Niemelä and P. Simons, Efficient implementation of the well-founded and stable model semantics, in I. Niemelä and T. Schaub, (eds.), *Proc. of JICSLP-96*, Cambridge, 1996.
27. I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, in I. Niemelä and T. Schaub, (eds.), *Proc. of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, 1998, pp. 72–79.
28. I. Niemelä and P. Simons, Smodels - an implementation of the stable model and well-founded semantics for normal logic programs, in J. Dix, U. Furbach, and A. Nerode, (eds.), *Logic Programming and Nonmonotonic Reasoning - 4th International Conference, LPNMR '97*, Dagstuhl Castle, Germany, 1997, pp. 420–429.
29. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello, A deductive system for non-monotonic reasoning, Jürgen Dix, Ulrich Furbach, and Anil Nerode, (eds.), *Proc. of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR97)*, number 1265 in LNCS, San Francisco, CA, 1997, pp. 364–375.
30. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello, The kr system dlv: Progress report, comparisons, and benchmarks, A.G. Cohn, L. Schubert, and S.C. Shapiro, (eds.), *Proc. Sixth International Conference on Principles of Knowledge*

Representation and Reasoning (KR-98), San Francisco, CA, 1998, pp. 406–417.

31. N. Leone, P. Rullo, and F. Scarcello, Disjunctive stable models: Unfounded sets, fixpoint semantics and computation, *Information and Computation*, **135**: 69–112, 1997.
32. G. Brewka and I. Niemelä, Report on the Seventh International Workshop on Nonmonotonic Reasoning, *AI Magazine*, **19** (4): 139–139, 1998.
33. A.C. Kakas, A.R. Kowalski, and F. Toni, The role of abduction in logic programming, in D.M. Gabbay, C.J. Hogger, and J.A. Robinson, (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming Volume 5*, Oxford: Oxford University Press, 1998, pp. 235–324.
34. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits, Using methods of declarative logic programming for intelligent information agents, *Theory and Practice of Logic Programming*, **2**: 645–709, November 2002.
35. J. Grant and J. Minker, A logic-based approach to data integration, *Theory and Practice of Logic Programming*, **2**: 323–368, 2002.
36. J. P. Delgrande, T. Schaub, and H. Tompits, A framework for compiling preferences in logic programs, *Theory and Practice of Logic Programming*, **3**: 129–187, March 2003.
37. M. Liu, Extending datalog with declarative updates, *Journal of Intelligent Information Systems*, **20**: 107–129, March 2003.
38. A. Brogi, V. S. Subrahmanian, and C. Zaniolo, A deductive database approach to ai planning, *Journal of Intelligent Information Systems*, **20**: 215–253, May 2003.
39. J. McCarthy, Programs with common sense, *Proc. Teddington Conf. on the Mechanisation of Thought Processes*, London, 1959, pp. 75–91.
40. V. Lifschitz, N. McCain, E. Remolina, and A. Tacchella, Getting to the airport: The oldest planning problem in AI, in Jack Minker, (ed.), *Logic-Based Artificial Intelligence*, Dordrecht: Kluwer Academic Publishers, 2000, pp. 147–165.
41. G. Greco, S. Greco, and E. Zumpano, A logical framework for querying and repairing inconsistent databases, *IEEE Transactions on Knowledge and Data Engineering*, **15**: 1389–1408, Nov/Dec 2003.
42. J.A. Robinson, A Machine-Oriented Logic Based on the Resolution Principle, *J.ACM*, **12** (1), January, 1965.
43. C.C. Green and B. Raphael, Research in intelligent question answering systems, *Proc. ACM 23rd National Conference*, 1968, pp. 169–181.
44. C.C. Green and B. Raphael, The use of theorem-proving techniques in question-answering systems, *Proc. 23rd National Conference ACM*, 1968.
45. J. Lobo, J. Minker, and A. Rajasekar, *Foundations of Disjunctive Logic Programming*, Cambridge MA: The MIT Press, 1992.
46. G. Brewka, J. Dix, and K. Konolige, *Nonmonotonic Reasoning: An Overview*, Center for the Study of Language and Information, Stanford, CA, 1997.
47. M. Arenas, L. Bertossi, and J. Chomicki, Consistent query answers in inconsistent databases, *PODS 1999*, New York: ACM Press, 1999, pp. 68–79.

John Grant
Towson University
Towson, Maryland
Jack Minker
University of Maryland at
College Park
College Park, Maryland

DISCRETE EVENT SYSTEMS: UNTIMED MODELS AND THEIR ANALYSIS

INTRODUCTION

Mathematical system theory is concerned with modeling, analyzing, and controlling *dynamic systems*, that is, systems described by variables that evolve over time. A "system" is an abstract concept to describe an entity that comprises input, state, and output variables. An important class of dynamic systems is that of electromechanical systems. Examples of such dynamic systems abound around us at all scales of human activity including disk drives in computers; heating, ventilation, and air conditioning systems in buildings; powertrains in automobiles; jet engines; power distribution networks, and so on. Their associated variables, such as rotation speed, temperature, torque, and voltage, are continuous in nature, and the laws of electricity and mechanics can be used to obtain a mathematical model of their behavior over time, usually in the form of differential or difference equations.

Another important class of dynamic systems is the class of *discrete event systems*. The distinguishing features of discrete event systems are (*1*) their variables do not evolve according to differential or difference equations, but rather according to the occurrence of asynchronous "events", and (*2*) these variables belong to a discrete (as opposed to continuous) set. The proliferation of technological systems that comprise computers interconnected by networks has led to the development of a mathematical system theory for discrete event systems. A communication protocol between a sender and a receiver in a network is an example of a discrete event system. The "state" of the sender belongs to a discrete set that includes states such as *idle*, *ready-to-send*, *waiting-for-acknowledgment*, and so forth, with similar states for the receiver. The dynamical evolution of the states of the sender and receiver is described by the occurrence of asynchronous events such as *packet-sent*, *packet-received*, *timer-started*, and *timer-expired*. To contrast this kind of behavior with that of a system described by a differential equation, the dynamics are said to be *event-driven* instead of being *time-driven*. Thus, the rules of the protocol lead to a mathematical model of this discrete event system.

Discrete event systems arise in many areas of computer science and engineering, which involve both hardware and software, such as protocols, databases, server systems, and so on. They also are generated in many classes of technological systems that involve automation, such as manufacturing, robotics, transportation, and so forth. The mathematical theory of discrete event systems is multidisciplinary in nature and involves concepts and techniques from computer science theory, dynamic systems, control theory, and operations research. A wide variety of modeling formalisms is used to describe and study the behavior of discrete event systems. Two widely used modeling formalisms for discrete event systems are *automata* and *Petri nets*.

Once a discrete event model of a system has been obtained, it can be used to analyze the behavior of the system in a formal manner. Many analytical questions are often posed in a *verification* context. Namely, along with the model of the system, one is also given a set of *specifications* or *requirements* regarding the desired behavior of the system. Three main categories of specifications are safety, liveness, and diagnosability. *Safety* specifications are concerned with the reachability of illegal states or illegal sequences of events in the behavior of the system. *Liveness* specifications pertain to the ability of the system to complete the tasks it starts. *Diagnosability* specifications arise in partially observed systems and refer to the ability of detecting the occurrence of significant unobservable events based on the observed sequences of events and the model of the system. If the result of the verification phase is that the behavior of the system is unsatisfactory in terms of safety, liveness, or diagnosability, then feedback control can be used to alter its behavior and ensure that all specifications are met. A controller module is adjoined to the system; this module is synthesized in such a way that the behavior of the closed-loop system, that is, the system plus the controller, is guaranteed to meet all specifications. Clearly, controller existence issues are of concern in systems where sensing and actuation limitations restrict the ability of the controller module to observe and affect the behavior of the system.

The remainder of this article will introduce the reader to modeling and analysis of discrete event systems. The discussion will be centered around the automaton modeling formalism.

MODELING AND ANALYSIS USING AUTOMATA

When modeling electromechanical systems in terms of the usual continuous variables of interest, for example, current, voltage, and position, the laws of nature can be used to obtain differential equations that model the behavior of these variables accurately over time. If necessary, techniques such as linearization can be used to obtain a model that is more analytically tractable. When modeling how packets are moved in a digital communications network, or how robots interact with conveyors and automated guided vehicles in moving parts in an automated manufacturing plant, models based on the asynchronous occurrences of a set of discrete events are more appropriate than models based on differential equations.

The first step in modeling a discrete event system is to identify a relevant set of *events* that capture all possible behaviors of the system at the desired level of abstraction. The concept of event is an intuitive one and does not require a formal definition. The level of granularity to consider when defining events depends on the level of abstraction of the verification or control task at hand. Here, one can argue that discrete event modeling is more art than science, in the sense that the knowledge of an expert is needed to

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

choose the right level of abstraction at which to study the system. When modeling the flow of traffic through a signalized intersection, one could define events such as *vehicle arrives*, *vehicle turns left*, *signal changes to red*, and so forth, if the goal is to keep track of how many vehicles are present at each approach. Individual vehicle dynamics, which are based on differential equations, would be too detailed for this kind of analysis.

Events cause state changes in the variables of interest in the system. That is why the dynamics are said to be event driven as opposed to time driven. For instance, the number of customers present in a queueing system will change during an *arrival* event or a *service-starts* event. The variables of interest have values that belong to a discrete set. They could be numerical, (e.g., the number of customers in the queue) or logical (e.g., the state of the server is *idle* or *busy*). The state of the entire system is formed by considering the states of its variables of interest. In the study of a communication protocol between two nodes in a network, the global state would be composed of the states of the sender, channel, and receiver.

Discrete event models of dynamic systems are classified in terms of how they abstract timing information and randomness in the system behavior. Untimed models abstract away precise timing issues by focusing only on the ordering of the events and not on the exact times of their occurrence. Untimed models also abstract away statistical information about the probabilities of the events and consider all possible “sample paths” in the system behavior. Untimed models are often referred to as “logical” models. A large class of logical specifications that involve safety, liveness, and diagnosability can be addressed with untimed models. Timed models enrich untimed models and explicitly include timing information. This information may be provided in a “non-stochastic” manner or in a “stochastic” manner. Event occurrence times can be specified in terms of time intervals for instance, or by means of probability distribution functions. Timed models are necessary to study specifications that involve real time and deadlines. Stochastic models are used for performance analysis under specific statistical assumptions. The process of model refinement from untimed models to [non-]stochastic timed ones is discussed in detail in the textbook by Cassandras and Lafortune (1). The remainder of this article is concerned exclusively with untimed models of discrete event systems.

Languages

Let E be the finite set of events associated with the discrete event system under consideration. This set consists of all events that can possibly be executed by the system. A *string* (or trace) is a finite sequence of events from E. Thus, a string models one possible behavior of the system during its operation. In the case of untimed models of discrete event systems, the exact time of occurrence of the event is abstracted away in the string; only the ordering of the events is retained. The length of a string s, denoted by $|s|$, is a non-negative integer that corresponds to the number of events composing the string, counting multiple occurrences of the same event. The empty string, which is denoted by ε (not to be confused with the generic event $e \in E$), is the string containing no events (i.e., $|\varepsilon| = 0$). The concatenation of two strings s_1 and s_2 is the trace $s_1 s_2$ (i.e., s_1 followed by s_2). Thus, the empty string ε can be interpreted as the identity element for concatenation.

The concept of *language* is central for modeling a discrete event system. An untimed language, or simply language, is a set of strings of events over an event set. The set of all finite strings of elements of E, including the empty string ε, is denoted by E^*; the * operation is called the Kleene closure. For example, if $E = \{a, b, c\}$, then

$$E^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \ldots\}$$

If $t_1 t_2 t_3 = s$ with $t_1, t_2, t_3 \in E^*$, then t_1 is called a *prefix* of s, t_2 a substring of s, and t_3 a *suffix* of s. Both ε and s are prefixes of s by definition.

Clearly, a given system with event set E cannot in general execute all possible strings in E^*. The set of all possible behaviors of a discrete event system is therefore modeled by a *subset* of E^* (i.e., by a language). The language associated with a discrete event system is usually infinite, because the system behavior might include repetitions. It is impractical to represent a language by enumerating all its elements. A *discrete event modeling formalism* is a mechanism for representing in a compact manner the language associated with the system. It can be thought of as a data structure as well as an analytical tool.

Automata

The automaton modeling formalism is the most widely used and studied.

Definition – Automaton. A *deterministic automaton*, denoted by G, is a six-tuple

$$G = (X, E, f, \Gamma, x_0, X_m)$$

where X is the set of *states*, which could be infinite; E is the finite set of *events* associated with the transitions in G; $f : X \times E \rightarrow X$ is the *transition function*; $f(x, e) = y$ means that there is a transition labeled by event e from state x to state y (in general, f is a *partial* function on its domain); $\Gamma : X \rightarrow 2^E$ is the *feasible event function*.[1] $\Gamma(x)$ is the set of all events e for which $f(x, e)$ is defined; x_0 is the *initial* state; $X_m \subseteq X$ is the set of *marked states*.

Proper selection of which states to mark is a modeling issue that depends on the problem of interest. By designating certain states as marked, one may for instance be recording that the system, on entering these states, has completed some operation or task.

When $|X|$ is finite and small, it is convenient to represent automata graphically by means of their *state transition diagrams*. The state transition diagram of an automaton is a directed graph in which nodes represent states, and labeled arcs between nodes are used to represent the transition function f: if $f(x, e) = y$, then an arc labeled by “e” is

[1] Given a set A, the notation 2^A means the power set of A (i.e., the set of all subsets of A).

drawn from x to y. Special notation is used to identify the initial state and marked states when drawing a state transition diagram.

Automata are used to represent and manipulate languages. For the sake of convenience, the transition function f of an automaton is extended from domain $X \times E$ to domain $X \times E^*$ in the following recursive manner:[2]

$$\begin{aligned} f(x,\varepsilon) &:= x \\ f(x,se) &:= f(f(x,s),e) \quad \text{for } s \in E^* \text{ and } e \in E \end{aligned}$$

The *language generated* by G is

$$\mathcal{L}(G) := \{s \in E^* : f(x_0, s) \text{ is defined}\}$$

The *language marked* by G is

$$\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) : f(x_0, s) \in X_m\}$$

The language $\mathcal{L}(G)$ represents all directed paths that can be followed along the state transition diagram of G starting at the initial state; the string corresponding to a path is the concatenation of the event labels of the transitions that compose the path. Therefore, a string s is in $\mathcal{L}(G)$ if and only if it corresponds to an admissible path in the state transition diagram, equivalently, if and only if f is defined at (x_0, s). The second language represented by G, $\mathcal{L}_m(G)$, is the subset of $\mathcal{L}(G)$ that consists only of the strings s for which $f(x_0, s) \in X_m$, that is, these strings correspond to paths that end at marked states in the state transition diagram.

The prefix-closure of a language L, which is denoted by $\overline{L}$, is the language that consists of all prefixes of all the strings in L. In general, $L \subseteq \overline{L}$. L is said to be *prefix-closed* if $L = \overline{L}$. Clearly, $\mathcal{L}(G)$ is prefix-closed by definition, because a path is only possible if all its prefixes are also possible. If f is a total function over its domain, then necessarily $\mathcal{L}(G) = E^*$. However, the language marked by G, $\mathcal{L}_m(G)$, is not prefix-closed in general, unless all the reachable states of G are marked. The language marked is also called the language *recognized* by the automaton, and the given automaton is often referred to as a *recognizer* of the given language.

An automaton G thus represents *two* languages: $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$. In the standard definition of automaton in automata theory, the function f is required to be a total function, and the notion of language generated is not meaningful because it is always equal to E^*. Allowing f to be a partial function is a consequence of the fact that a discrete event system may not produce (or execute) all the strings in E^*.

A language is said to be *regular* if it can be marked by a *finite-state* automaton (i.e., by an automaton where X is a finite set). The class of regular languages is denoted by $\mathcal{R}$. It can be shown that $\mathcal{R}$ is a proper subset of 2^{E^*}, which is the class of all languages over event set E. The class $\mathcal{R}$ is very important because it delimits the languages that possess automata representations that require finite memory when stored in a computer. In other words, automata are a practical means of manipulating regular languages in analysis or controller synthesis problems. A textbook on automata and language theory, such as the book by Sipser (2), can be consulted for additional study of the properties of $\mathcal{R}$. However, automata are not a practical means for representing nonregular languages, because they would require infinite memory. The Petri net modeling formalism has proved useful for modeling many classes of nonregular languages. It will not be discussed more in this article. The interested reader is referred to Chapter 4 in Cassandras and Lafortune (1) and to the book by Hrúz and Zhou (3).

Reachability and Blocking

From the definitions of $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$, all states of G that are not *accessible* or *reachable* from x_0 by some string in $\mathcal{L}(G)$ can be deleted without affecting the languages generated and marked by G. When "deleting" a state, all transitions that are *attached* to that state are also deleted. This operation is denoted by $Ac(G)$, where Ac stands for taking the "accessible" part.

In general, from the definitions of G, $\mathcal{L}(G)$, and $\mathcal{L}_m(G)$, the following set inclusions hold:

$$\mathcal{L}_m(G) \subseteq \overline{\mathcal{L}_m(G)} \subseteq \mathcal{L}(G)$$

The first set inclusion occurs because X_m may be a proper subset of X, whereas the second set inclusion is a consequence of the definition of $\mathcal{L}_m(G)$ and the fact that $\mathcal{L}(G)$ is prefix-closed. It is worth examining this second set inclusion in more detail. An automaton G could reach a state x where $\Gamma(x) = \emptyset$ but $x \notin X_m$. This result is called a *deadlock* because no other event can be executed in a state that is not marked. Another issue to consider is when a set of unmarked states in G form a strongly connected component (i.e., these states are reachable from one another) but with *no transition going out of the set*. When the system enters this set of states, its resulting behavior is a *livelock*. Although the system is "live" in the sense that it can always execute an event, it can never complete the task started because no state in the set is marked, and the system cannot leave this set of states.

Automaton G is said to be *blocking* if $\overline{\mathcal{L}_m(G)} \subset \mathcal{L}(G)$ where the set inclusion is proper,[3] and *nonblocking* when $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$. Thus, if an automaton is blocking, then deadlock and/or livelock can happen. The notion of marked states and the above definitions for language generated, language marked, and blocking, provide an approach for considering deadlock and livelock that is useful in a wide variety of applications.

Composition of Automata

Discrete event models of complex dynamic systems are built rarely in a monolithic manner. Instead, a modular approach is used where models of individual components are built first, followed by the composition of these models to

[2] The Notation ":=" stands for "equal to by definition."

[3] The notation $\subset$ is for "strictly contained in" and $\subseteq$ is for "contained in or equal to."

obtain the model of the overall system. In the automaton modeling formalism, the synchronization, or coupling, between components is captured by the use of *common events*. Namely, if components G_A and G_B share event c, then event c should only occur if both G_A and G_B execute it. The process of composing individual automata (that model interacting system components) in a manner that captures the synchronization constraints imposed by their common events is formalized by the *parallel composition* operation, which is denoted by $||$.

Consider the two automata $G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1})$ and $G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$. The *parallel composition* of G_1 and G_2 is the automaton

$$G_1 \,||\, G_2 := Ac(X_1 \times X_2, E_1 \cup E_2,\ f, \Gamma_{1||2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

where

$$f((x_1, x_2), e) := \begin{cases} (\,f_1(x_1, e),\ f_2(x_2, e)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (\,f_1(x_1, e), x_2) & \text{if } e \in \Gamma_1(x_1) \setminus E_2 \\ (x_1,\ f_2(x_2, e)) & \text{if } e \in \Gamma_2(x_2) \setminus E_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

and thus $\Gamma_{1||2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus E_2] \cup [\Gamma_2(x_2) \setminus E_1]$.

Associativity is used to extend the definition of parallel composition to more than two automata. The accessible operation is used to capture the fact that only the states reachable from (x_{01}, x_{02}) need to be considered. In the parallel composition, a common event (i.e., an event in $E_1 \cap E_2$) can only be executed if the two automata both execute it simultaneously. Thus, the two automata are “synchronized” on their common events. The “private” events, namely, those in $(E_2 \setminus E_1) \cup (E_1 \setminus E_2)$, are not subject to such a constraint and can be executed whenever possible by an automaton without the participation of the other automaton. Thus, the required synchronization on common events provides a modeling strategy to capture interactions among components in a system.

If $E_1 = E_2$, then the parallel composition of G_1 and G_2 results in their lock-step behavior: They must jointly execute every single event. If $E_1 \cap E_2 = \emptyset$, then no synchronized transitions occur, and $G_1||G_2$ is the *concurrent* behavior of G_1 and G_2. This behavior is often termed the *shuffle* of G_1 and G_2.

Analysis

Once a complete system model has been obtained by parallel composition of a set of automata, the resulting monolithic model, which is denoted by G_{all}, can be used to analyze the properties of the system. *Safety properties* are properties that are concerned with the reachability of certain undesirable states (from the initial state or from some other state) in G_{all}, the presence of certain undesirable strings or substrings in $\mathcal{L}(G_{all})$, or more generally the inclusion of $\mathcal{L}(G_{all})$ in a given language, which is referred to as the “legal” or “admissible” language. When G_{all} is finite state, safety properties can be answered by examination and/or manipulation of its transition structure; such tests are usually of polynomial-time complexity in the size of the state set of G_{all}. *Blocking properties* are properties that are concerned with the presence of deadlock and/or livelock in G_{all}. Observe that $G_1||G_2$ may be blocking even if G_1 and G_2 are each nonblocking. Thus, it is necessary to examine the transition structure of G_{all} to answer blocking properties. Again, this process can be done in polynomial-time complexity in the size of the state set.

It should be noted that the size of the state set of G_{all} may in the worst case grow exponentially in the number of automata that are composed by $||$. This process is known as the “curse of dimensionality” in the study of complex systems composed of many interacting components. The presence of common events can curtail this growth significantly by means of the synchronization constraints that it imposes.

Example. Consider the sequential transmission of packets between two nodes, which are denoted by A and B, in a data network. The transmission channel is noisy, and thus packets can “get lost” (namely, they can be corrupted by errors or dropped if memory buffers are full). One class of protocols that control the transmission between A and B is the class of STOP and WAIT protocols. These protocols operate as follows:

1. A sends packet to B then starts timer;
2.a. A receives acknowledgement (ACK) from B: Go back to 1;
2.b. A times out: A resends packet to B and starts timer. Go back to 2.

A little thought shows that node B needs to know whether a packet received is a new packet or a retransmission of a previous packet by node A, because the transmission delay is not known exactly and ACKs can get lost too. A widely used strategy to address this problem is to add one bit to each packet header, which results in “0-packets” and “1-packets.” This header bit is flipped by node A each time A sends a new packet (as opposed to a retransmission). This protocol is called the *alternating bit protocol* (ABP). It is desired to model a simple version of ABP, for transmission from A to B only, using the automaton modeling formalism. Assume that the channel can only contain one packet at a time (half-duplex case). Based on the above discussion, the following list of nine events is defined for this system. [This modeling strategy is inspired by that in the book by Tanenbaum (4, sections 3.3.3 and 3.5.1).]

- *lost*: contents of channel lost
- *new0*: 0-packet received by B; ACK sent by B to A; 0-packet delivered by node B
- *ack + 1*: ACK received by A; 1-packet sent by A to B
- *new1*: 1-packet received by B; ACK sent by B to A; 1-packet delivered by node B
- *ack + 0*: ACK received by A; 0-packet sent by A to B
- *repeat0*: 0-packet received by B; ACK sent by B to A
- *repeat1*: 1-packet received by B; ACK sent by B to A
- *timeout + 0*: timeout at A; 0-packet retransmitted to B
- *timeout + 1*: timeout at A; 1-packet retransmitted to B

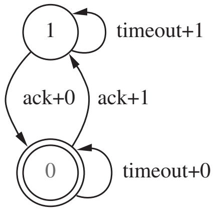


Figure 1. Sender automaton S. The event set is $E_S = \{ack+0, ack+1, timeout+0, timeout+1\}$.

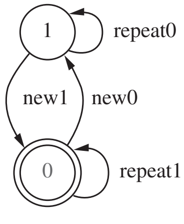


Figure 3. Receiver automaton R.

A packet is "delivered by node B" when B forwards the packet to the application program running at that node.

Three automata are defined as follows: S, the sender automaton, C, the channel automaton, and R, the receiver automaton. They are shown in Figs. 1, 2, and 3, respectively. The software tool DESUMA (5) was used to perform the calculations on automata discussed in this article and to draw the transition diagrams in the figures.[4] The sender has two states: State i models the situation where the sender is attempting to send an i-packet. Similarly, state i of the receiver models the situation where the receiver is expecting an i-packet. The four states of the channel refer to its four possible contents: *EMPTY*, 0- or *1*-packet, and *ACK*. The initial state of the system is chosen to be that where S is sending a 0-packet (state 0 in Fig. 1), C contains a 0-packet (state 0 in Fig. 2), and R is expecting a 0-packet (state 0 in Fig. 3), which is the starting point of the ABP protocol after proper initialization of the communication session between A and B transmission of the first packet. Only the relevant events are included in each automaton. The event set of the sender is $E_S = \{ack + 0, ack + 1, timeout + 0, \quad timeout + 1\}$. The event set of the receiver is $E_R = \{new0, new1, repeat0, repeat1\}$. The event set of the channel is $E_S \cup E_R$ plus the event *lost*. Common events are used to capture the coupling between the sender and the channel and between the receiver and the channel.

The parallel composition $SCR = S||C||R$ is shown in Fig. 4; note that states of SCR are triples formed by the corresponding states of S, C, and R. SCR has 10 reachable states from its initial state (0,0,0), which is less than the 16 possible combinations of states of S, R, and C. Because the only marked states in S, C, and R are their respective initial states, then state (0,0,0) is the only marked state of SCR. Because any state of SCR can eventually reach state (0,0,0), automaton SCR is nonblocking. It can be observed from the transition diagram of SCR that events *new0* and *new1* alternate, and that no *ack+1* events occur between events *new1* and *new0*, and no *ack+0* events occur between events *new0* and *new1*. Thus, it can be concluded that all new packets transmitted by A are eventually delivered by B.

The book by Holzmann (6) may be consulted for further reading on the formal study of communication/computer protocols using discrete-event modeling formalisms such as automata.

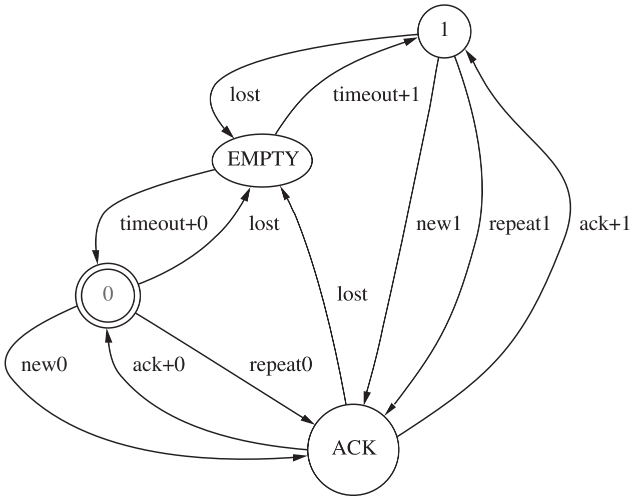


Figure 2. Channel automaton C.

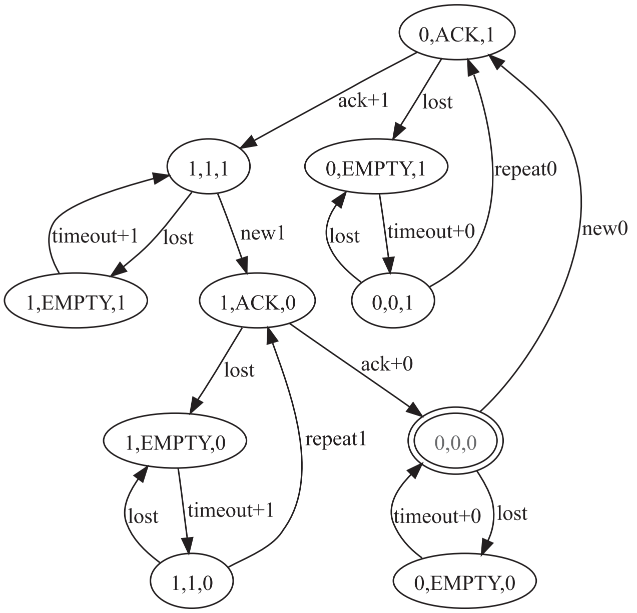


Figure 4. *SCR*: Parallel composition of sender, channel, and receiver.

[4] DESUMA uses the color blue to identify the initial state and draws double circles around marked states. We will indicate in the text the initial state of each automaton.

PARTIALLY OBSERVED SYSTEMS

Automata models of discrete event systems often include descriptive properties for the events, such as observability and controllability, that capture sensing and actuation capabilities. An event is said to be *observable* if it is "seen" by an outside observer of the system behavior. In practice, observable events are those recorded by the sensors attached to the system. Let the set E of events of an automaton G be partitioned into the set of observable events, E_o, and the set of unobservable events, E_{uo}, where $E = E_o \cup E_{uo}$ and $E_o \cap E_{uo} = \emptyset$. From the point of view of the outside observer (whose task may be state estimation, diagnostics, or control), the unobservable transitions of G are not observed; thus, the outside observer may not know the exact state of the system even if it knows the initial state.

To capture the unobservability of events in E_{uo} in the strings of a language, a *projection* operation is introduced. It is denoted by P and it projects strings in E^* to strings in $E_o^* : P : E^* \to E_o^*$ where

$$\begin{aligned} P(\varepsilon) &:= \varepsilon \\ P(e) &:= \begin{cases} e & \text{if } e \in E_o \\ \varepsilon & \text{if } e \in E_{uo} = E \setminus E_o \end{cases} \\ P(se) &:= P(s)P(e) \quad \text{for } s \in E^*, e \in E \end{aligned}$$

(Recall that ε is the symbol for the empty string.) As can be observed from the definition, the projection operation takes a string formed from the event set E and *erases* events in it that are unobservable. The corresponding inverse map $P^{-1} : E_o^* \to 2^{E^*}$ is defined as follows:

$$P^{-1}(t) := \{s \in E^* : P(s) = t\}$$

Given a string of observable events, the inverse projection P^{-1} returns the set of all strings in E^* that project, with P, to the given string. The projection P and its inverse P^{-1} are extended to languages by simply applying them to all the strings in the language. For $L \subseteq E^*$,

$$P(L) := \{t \in E_o^* : (\exists s \in L)\, [P(s) = t]\}$$

and for $L_o \subseteq E_o^*$,

$$P^{-1}(L_o) := \{s \in E^* : (\exists t \in L_o)\, [P(s) = t]\}$$

Observer Automata

Given a language K represented by nonblocking automaton G [i.e., $\mathcal{L}_m(G) = K$ and $\mathcal{L}(G) = \overline{K}$], an automaton representation of language $P(K)$ can be obtained by replacing all occurrences of unobservable events in the transition diagram of G by ε; let us denote the resulting automaton by $P(G)$. The inclusion of ε-transitions makes $P(G)$ a *nondeterministic* automaton. It is desired to obtain a *deterministic* automaton that will represent the language $P(K)$. Such an automaton can be constructed from G by adopting the standard determinization algorithm of automata theory. In the terminology of discrete event systems, the resulting automaton is called the *observer* of G and denoted by $Obs(G)$. The construction of $Obs(G)$ uses the notion of *unobservable reach* of state x, which is the set of states reachable from x by unobservable transitions, including x itself. Formally,

$$UR(x) = \{y \in X : (\exists t \in E_{uo}^*)\, [f(x,t) = y]\}$$

This definition is extended to sets of states $B \subseteq X$ as follows: $UR(B) = \cup_{x \in B} UR(x)$.

Procedure for Building Observer *Obs(G)* of Automaton *G* with Unobservable Events. Let $G = (X, E, f, \Gamma, x_0, X_m)$ be a deterministic automaton and let $E = E_o \cup E_{uo}$.

Then $Obs(G) = (X_{obs}, E_o, f_{obs}, \Gamma_{obs}, x_{0,obs}, X_{m,obs})$ and it is built as follows.

Step 1: Define $x_{0,obs} := UR(x_0)$.
Set $X_{obs} = \{x_{0,obs}\}$.

Step 2: For each $B \in X_{obs}$ and $e \in E_o$, define

$$f_{obs}(B,e) := UR(\{x \in X : (\exists x_e \in B)\, [x = f(x_e, e)]\})$$

whenever $f(x_e, e)$ is defined for some $x_e \in B$. In this case, add the state $f_{obs}(B,e)$ to X_{obs}. If $f(x_e, e)$ is not defined for any $x_e \in B$, then $f_{obs}(B,e)$ is not defined.

Step 3: Repeat Step 2 until the entire *accessible* part of $Obs(G)$ has been constructed.

Step 4: Γ_{obs} is inferred from f_{obs} and $X_{m,obs} := \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$.

By construction, $Obs(G)$ is a deterministic automaton such that $\mathcal{L}(Obs(G)) = P[\mathcal{L}(G)]$ and $\mathcal{L}_m(Obs(G)) = P[\mathcal{L}_m(G)]$. Moreover, let $B(t) \subseteq X$ be the state of $Obs(G)$ that is reached after string $t \in P[\mathcal{L}(G)]$ [i.e., $B(t) = f_{obs}(x_{0,obs}, t)$]. Then $x \in B(t)$ iff X is reachable in G by a string in $P^{-1}(t) \cap \mathcal{L}(G)$. In this sense, state $B(t)$ of $Obs(G)$ is the best possible estimate of the state of G after the observed string of events t.

Observer automata are used to study properties of partially observed discrete event systems. One such property is *diagnosability*, and it is briefly discussed in the next section.

Example. Recall the automaton SCR shown in Fig. 4. Let the set of observable events be $E_o = E_S$, namely, the event set of the sender. The observer automaton of SCR for this set of observable events is given in Fig. 5. The initial state of $Obs(SCR)$ is $\{(0,0,0), (0, EMPTY, 0), (0, ACK, 1), (0, EMPTY, 1)\}$, which is the unobservable reach of state $(0,0,0)$ in Fig. 4.

Diagnoser Automata

When the system model contains unobservable events, one may be interested in determining whether certain unobservable events occurred in the string of events executed by the system. This question is the problem of *event diagnosis*. If these unobservable events of interest model faults of system components, then knowing that one of these events has occurred is very important when monitoring

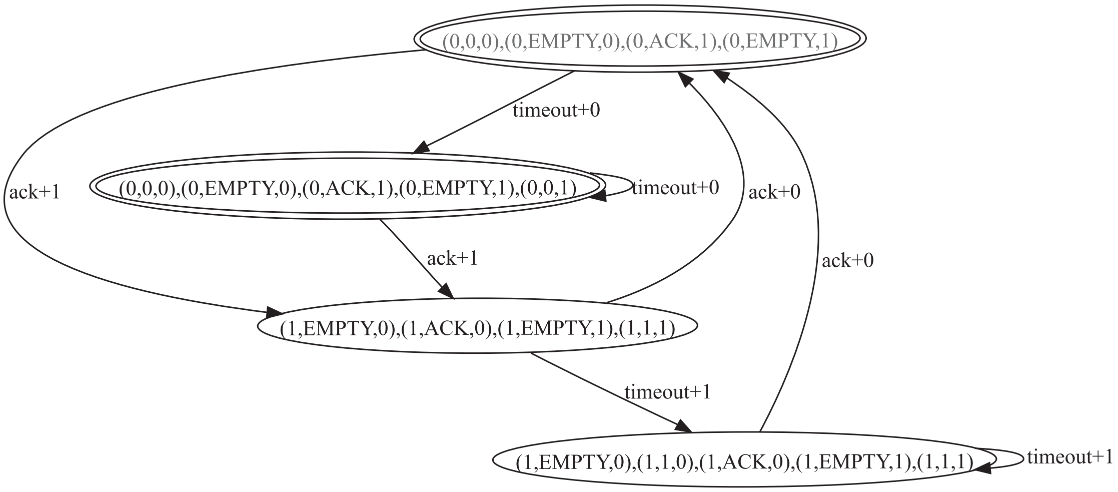

(0,0,0),(0,EMPTY,0),(0,ACK,1),(0,EMPTY,1)
timeout+0
ack+1
(0,0,0),(0,EMPTY,0),(0,ACK,1),(0,EMPTY,1),(0,0,1)
timeout+0
ack+0
ack+1
ack+0
(1,EMPTY,0),(1,ACK,0),(1,EMPTY,1),(1,1,1)
timeout+1
(1,EMPTY,0),(1,1,0),(1,ACK,0),(1,EMPTY,1),(1,1,1)
timeout+1

Figure 5. $Obs(SCR)$: Observer automaton of SCR for the set of observable events E_S.

the performance of the system. The key point here is that as more observations of the system behavior are recorded, uncertainty about prefixes of the string of events executed by the system can be reduced.

Consider for instance automaton SCR of Fig. 4. An observer with knowledge of this model but having only access to the events in E_S at run time would deduce, on the occurrence of *timeout+0* or *timeout+1*, that unobservable event *lost* must have occurred in the string of events executed so far by the system. It is desirable to "automate" this inferencing process using the system model and the string of observable events. The observer $Obs(SCR)$ in Fig. 5 is not sufficient for this purpose; as was discussed earlier, the observer returns the estimate of the state of the system but without accounting for past occurrences of specific unobservable events. Labels can be attached to the states of G in the observer to record execution of specific unobservable events. Assume a single unobservable event denoted by d needs to be diagnosed. Let label N denote that the event has not yet occurred and label Y denote that it has occurred. The label of the initial state x_0 is always N. Once label N has been updated to y to record the occurrence of d, the label will remain Y for all future reachable states. Namely, d causes the label to change from N to Y, after which no additional label change occurs. This label propagation rule can be captured in the form of an automaton. Consider the automaton in Fig. 6. Let us refer to this automaton as the *label automaton* for diagnosing event d and denote it by $A_{label} := (\{N, Y\}, \{d\}, f_{label}, \Gamma_{label}, N, \{N, Y\})$. The state names in A_{label} contain the label information. All the states of A_{label} are marked, and N is the initial state.

An automaton called *diagnoser automaton* and denoted by $Diag(G)$ is built for the purpose of event diagnosis. First, G is composed with A_{label} by parallel composition; note that all states of A_{label} were marked to guarantee that state marking in $G||A_{label}$ is consistent with that in G. Second, the observer of $G||A_{label}$ is constructed for the given set of observable events by applying the construction procedure described earlier. Overall:

$$Diag(G) = Obs(G||A_{label})$$

The parallel composition does not affect the language generated by G but it will result in attaching labels to the states of G as the states of $G||A_{label}$ are of the form $(x, label)$, where $label \in \{N, Y\}$. The label attached to x will depend on the absence (N) or presence (Y) of d in strings that reach x from x_0. State splitting will occur for those states of G that are reachable by strings that contain d and by strings that do not contain d.

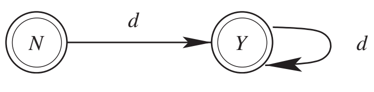


Figure 6. Automaton for building diagnoser.

Example. Recall again the automaton SCR shown in Fig. 4, and let the set of observable events be $E_o = E_S$. Two diagnoser automata are constructed: one for diagnosing unobservable event *lost* the other for diagnosing unobservable event *new0*. The diagnoser for *lost* is shown in Fig. 7; in this case, the label *F1* is used in place of Y[5]. The initial state of this diagnoser is the initial state of the observer in Fig. 5 with label information attached:$\{[(0,0,0), N], [(0, EMPTY, 0), F1], [(0, ACK, 1), N], [(0, EMPTY, 1), F1]$.The diagnoser for *new0* is shown in Fig. 8; in this case, the label *F2* is used in place of Y. Similarly, its initial state is $\{[(0,0,0), N], [(0, EMPTY, 0), N], [(0, ACK, 1), F2], [(0, EMPTY, 1), F2]\}$.

Online (or "on-the-fly") diagnosis of event d is performed by tracking the current diagnoser state in response to the observable events executed by the system G. First, if all states of G in the current state of $Diag(G)$ have label N, then it is certain that event d has not occurred yet. Such a state is called a *negative* state. Second, if all states of G in the current state of $Diag(G)$ have label Y, then it is certain that event d has occurred at

[5] This result is a consequence of the notation employed by the software tool DESUMA.

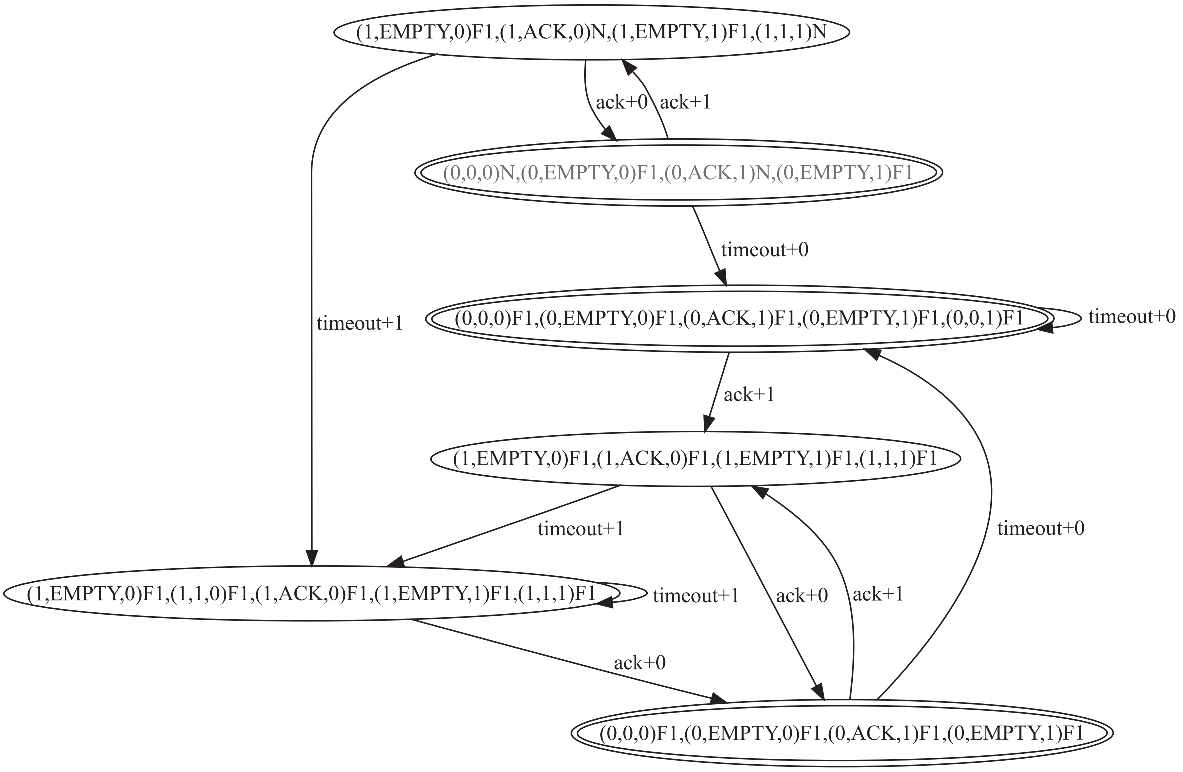

(1,EMPTY,0)F1,(1,ACK,0)N,(1,EMPTY,1)F1,(1,1,1)N
ack+0
ack+1
(0,0,0)N,(0,EMPTY,0)F1,(0,ACK,1)N,(0,EMPTY,1)F1
timeout+0
timeout+1
(0,0,0)F1,(0,EMPTY,0)F1,(0,ACK,1)F1,(0,EMPTY,1)F1,(0,0,1)F1
timeout+0
ack+1
(1,EMPTY,0)F1,(1,ACK,0)F1,(1,EMPTY,1)F1,(1,1,1)F1
timeout+1
timeout+0
(1,EMPTY,0)F1,(1,1,0)F1,(1,ACK,0)F1,(1,EMPTY,1)F1,(1,1,1)F1
timeout+1
ack+0
ack+1
ack+0
(0,0,0)F1,(0,EMPTY,0)F1,(0,ACK,1)F1,(0,EMPTY,1)F1

Figure 7. Diagnoser automaton of *SCR* for event *lost* and observable event set E_S. The label $F1$ is used to record occurrences of *lost*.

some point in the past. Such a state is called a *positive* state. If $t \in P[\mathcal{L}(G)]$ has been observed and $f_d(x_{0,d}, t)$ is positive, where $x_{0,d}$ is the initial state of $Diag(G)$ and f_d its transition function, then all strings in $P^{-1}(t) \cap \mathcal{L}(G)$ must contain d. If not, one state of G in $f_d(x_{0,d}, t)$ would have label N as all the strings in $P^{-1}(t) \cap \mathcal{L}(G)$ lead to state $f_d(x_{0,d}, t)$ of $Diag(G)$. Last, if the current state of $Diag(G)$ contains at least one state of G with label N and at least one state of G with label Y, then event d may or may not have occurred in the past. Such a state is called an *uncertain* state. In this case, two strings exist in $\mathcal{L}(G)$, say s_1 and s_2, such that $P(s_1) = P(s_2)$ (hence, they lead to the same state

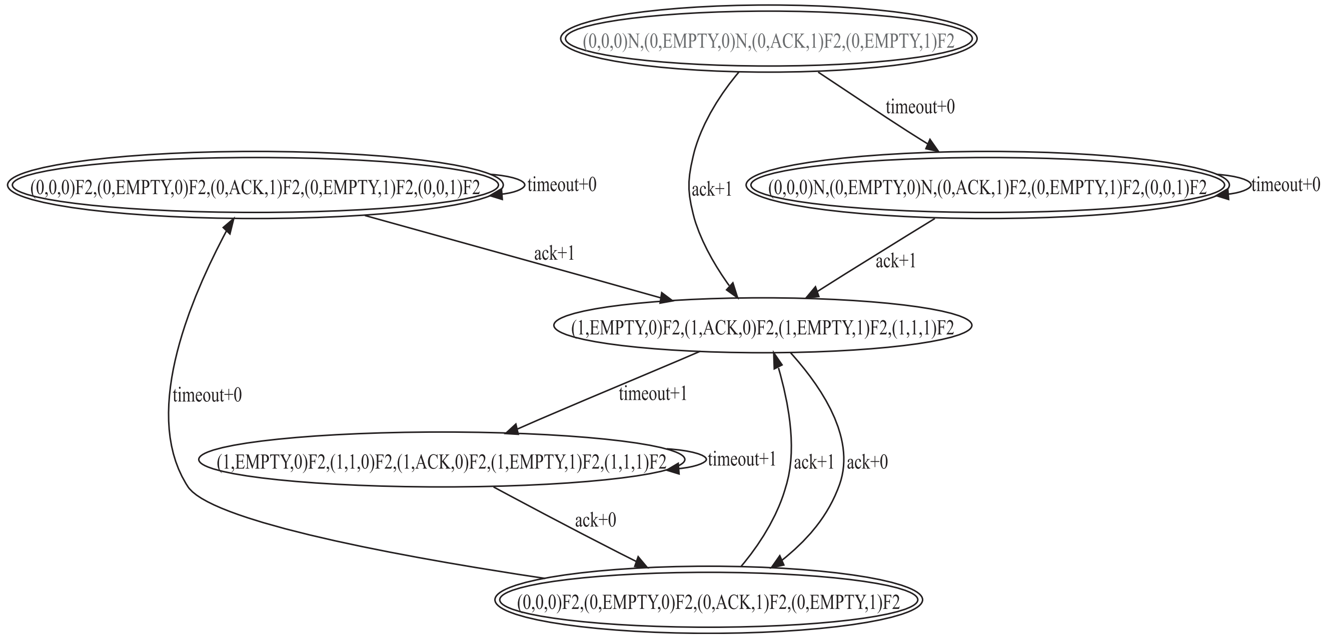

(0,0,0)N,(0,EMPTY,0)N,(0,ACK,1)F2,(0,EMPTY,1)F2
timeout+0
(0,0,0)F2,(0,EMPTY,0)F2,(0,ACK,1)F2,(0,EMPTY,1)F2,(0,0,1)F2
timeout+0
ack+1
(0,0,0)N,(0,EMPTY,0)N,(0,ACK,1)F2,(0,EMPTY,1)F2,(0,0,1)F2
timeout+0
ack+1
ack+1
(1,EMPTY,0)F2,(1,ACK,0)F2,(1,EMPTY,1)F2,(1,1,1)F2
timeout+0
timeout+1
(1,EMPTY,0)F2,(1,1,0)F2,(1,ACK,0)F2,(1,EMPTY,1)F2,(1,1,1)F2
timeout+1
ack+1
ack+0
ack+0
(0,0,0)F2,(0,EMPTY,0)F2,(0,ACK,1)F2,(0,EMPTY,1)F2

Figure 8. Diagnoser automaton of *SCR* for event *new0* and observable event set E_S. The label $F2$ is used to record occurrences of *new0*.

in $Diag(G)$), where s_1 contains d but s_2 does not. Note that state marking information in $Diag(G)$ is not used in event diagnosis.

Returning to automaton G in Fig. 4 and its diagnoser $Diag(G)$ for event $d = new0$ in Fig. 8, let us consider the two following strings in $\mathcal{L}(G)$: $s_N = (lost\ timeout + 0)^m$ and $s_Y = new0\ (lost\ timeout +\ 0\ repeat0)^m$. String s_Y has the following property: Given any $n \in \mathbb{N}$, m always exists such that the suffix of s_Y after event $d = new0$ has length greater than n. In other words, "s_Y is of arbitrarily long length after d." An examination of the transition diagram of G reveals that the suffix of s_Y after d loops in a cycle of G. Clearly, $P(s_N) = P(s_Y)$. Thus, if s_Y is executed by the system, then it is not possible to diagnose with certainty the occurrence of $d = new0$. After the occurrence of d, $P(s_Y) = P(s_N)$ will cycle in uncertain states in $Diag(G)$. In Fig. 8, a self-loop is caused by event *timeout+0* at state $\{[(0,0,0),N],\ [(0,EMPTY,0),N],\ [(0,ACK,1),F2],\ [(0,EMPTY,1),F2],\ [(0,0,1),F2]\}$. When such strings s_Y and s_N exist, the occurrence of d in s_Y is *not diagnosable*.

To formalize the notion of diagnosability, consider for simplicity languages that are *live*, that is, languages that do not contain terminating strings. A language L is said to be live if whenever $s \in L$, then $e \in E$ exists such that $se \in L$. In terms of automata, an automaton that has no deadlock state generates a live language. Unobservable event d is *not diagnosable* in live language $\mathcal{L}(G)$ if two strings s_N and s_Y exist in $\mathcal{L}(G)$ that satisfy the following conditions: (*1*) s_Y contains d and s_N does not, (*2*) s_Y is of arbitrarily long length after d, and (*3*) $P(s_N) = P(s_Y)$. When no such pair of strings exists, d is said to be *diagnosable* in $\mathcal{L}(G)$.

To avoid situations where G keep executing unobservable events after the occurrence of d, which means that no diagnosis can be performed as the state of $Diag(G)$ never gets updated, it is customary to assume that G has no cycles of unobservable events that are reachable after any occurrence of d. When the property of diagnosability is satisfied, if d occurs, then $Diag(G)$ will necessarily enter a positive state in a bounded number of events after the occurrence of d. To see this, observe that (*1*) G does not have a loop of unobservable events after d (by hypothesis), (*2*) $Diag(G)$ cannot loop in a cycle of uncertain states as this would contradict diagnosability, and (*3*) $Diag(G)$ has a finite set of states. An examination of G in Fig. 4 and its diagnoser $Diag(G)$ for event $d = lost$ in Fig. 7 confirms that event *lost* is indeed diagnosable. The cycle formed by events *ack+0* and *ack+1* in $Diag(G)$ and that involves uncertain states does not lead to the existence of a pair of strings s_Y and s_N violating diagnosability; indeed, no s_Y with the above properties exists, and this cycle will be exited by events *timeout+0* or *timeout+1* should event *lost* occur, taking the diagnoser to positive states.

DISCUSSION

This article has introduced modeling and analysis of discrete event systems represented by finite-state automata. An example from communication protocols was used to illustrate key concepts. The interested reader is encouraged to consult recent textbooks in discrete event systems, such as the books by Cassandras and Lafortune (1), Hrúz and Zhou (3), and Zimmermann (7). More advanced treatments of control of discrete event systems can be found in the notes of Wonham (8) as well as in the book of Iordache and Antsaklis (9). Simulation of stochastic discrete event systems is treated in the book by Law (10).

BIBLIOGRAPHY

1. C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York: Springer, 2007.
2. M. Sipser, *Introduction to the Theory of Computation*, 2nd ed. Boston, MA: Thomson Course Technology, 2006.
3. B. Hrúz and M. C. Zhou, *Modeling and Control of Discrete-Event Dynamic Systems*. London: Springer, 2007.
4. A. Tanenbaum, *Computer Networks-3rd ed.*, Upper Saddle River, NJ: Prentice-Hall, 1996.
5. L. Ricker, S. Lafortune, and S. Genc, DESUMA: a tool integrating GIDDES and UMDES, in *Proceedings of the 8th International Workshop on Discrete Event Systems*, Ann Arbor, MI, 2006, pp. 392–393.
6. G. J. Holzmann, *Design and Validation of Computer Protocols*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
7. A. Zimmermann, *Stochastic Discrete Event Systems: Modeling, Evaluation, Applications*. New York: Springer, 2007.
8. W. M. Wonham, Supervisory Control of Discrete-Event Systems. Available: http://www.control.utoronto.ca/DES/wonham.html, updated 1 July 2007.
9. M. V. Iordache and P. J. Antsaklis, *Supervisory Control of Concurrent Systems - A Petri Net Structural Approach*. Boston, MA: Birkhäuser, 2006.
10. A. M. Law, *Simulation Modeling & Analysis*, 4th ed., Boston, MA: McGraw-Hill, 2007.

STÉPHANE LAFORTUNE
University of Michigan
Ann Arbor, Michigan

DISK STORAGE

CONCEPT

Disk storage is a general category of computer storage, in which data are recorded on planar, round, and rotating surfaces (disks, discs, or platters). A disk drive is a peripheral device from which information is collected. Main implementations are hard disks, floppy disks, and optical discs.

Nowadays the term "disk storage" almost exclusively refers to hard disk or magnetic disk, in which a thin spinning disk with magnetic coating is used to hold data. Read/write heads are placed above and/or below the disk so that as the disk spins, each head traverses a circle, called a track, around the disk's upper or lower surface. Since a track can contain more information than we would normally want to manipulate at any one time, each track is divided into arcs called sectors on which information is recorded as a continuous string of bits (1, 2).

The location of tracks and sectors is not a permanent part of a disk's physical structure. Instead, they are marked magnetically through a process called "formatting the disk" (1). This process is usually performed by the disk's manufacturer or an end user, resulting in what are known as formatted disks. In an exemplified Microsoft Windows/MS-DOS formatting operation, some code such as File Allocation Table (FAT) is copied into the boot sector. The boot sector is a special area used by the computer to start the process of booting the computer.

Disk storage is usually managed by a (disk) file system. In the view screen of an application program, a file is a sequence of logic blocks (each block is one or more sectors). A file has a structure that determines how its blocks are organized. The file system also includes software routines used to control access to the storage on a hard disk system. The disk directory knows how big a file is and when it was created. For example, FAT stores the location of the first block for a given filename (Fig. 1) and knows where the other blocks are located.

The physical structure is associated with how a file is actually organized on the storage medium on which it resides. In a Windows environment, FAT is supposed to translate a logic location to a physical location. The FAT contains information about which blocks are assigned to which file. FAT is such an important file on a disk that destroying the FAT makes the disk useless. When a file is written to a hard disk, it is unlikely to have it in a consecutive order. The file (blocks) would be scattered over different sectors. To preserve the proper order, most operating systems maintain a linked list of the blocks on which the file is stored. By means of this list, the operating system can retrieve the sectors in the proper sequence, and therefore, the application software can be written as if the file were stored sequentially, even though the file is actually scattered over various portions of the disk (Fig. 1).

In Unix/Linux environments, indoe (3) is used to preserve a file's order on a disk (Fig. 2). The first 12 of these pointers point to direct blocks; that is, they contain addresses of blocks that contain data of the file. Thus, the data for small (no more than 12 blocks) files do not need a separate index block. The next three pointers point to indirect blocks. The first indirect pointer is the address of a single index block, containing not data, but rather the addresses of blocks that do contain data. This approach could be continued to a second indirect level or third indirect level, depending on the desired maximum file size.

DISK STORAGE HISTORY

In 1956, IBM launched a 305 RAMAC system that consisted of fifty 24-inch diameter disk platters, superseding magnetic cores and magnetic drums. Weighting over a ton and storing 5 MB of data, the 305 RAMAC is the beginning of what we currently call a "disk drive" in which data are stored on magnetic disk platters and accessed by moving disk heads. The random-access, low-density storage of disks also complements the already used sequential-access, high-density storage provided by magnetic tape (4).

Over the past five decades, the disk drive has experienced dramatic development. For example, the recording density has achieved over six orders of magnitude and the performance has been improved over four orders. However, the basic mechanical architecture in disk drive has not been changed too much (4).

The need for large-scale, reliable storage, independent of a particular device, led to the introduction of various disk storage configurations such as redundant array of independent disks (RAID), massive array of idle disks (MAID), network attached storage (NAS), grid-oriented storage (GOS) (5), and storage area network (SAN) that provide efficient, reliable, and remote access to large volumes of data in network and/or grid environments.

DISK STORAGE TECHNIQUES

In disk storage, degradation in performance over time is a serious and common concern. Sometimes, people benchmark their disks when new, and then many months later, they are surprised to find that the disk is getting slower. In fact, the disk most likely has not changed at all, but the second benchmark may have been run on tracks closer to the center of the disk. The tracks on the outside are much longer than the ones on the inside, typically doubling the circumference or more and, therefore containing more data, but the angular velocity of the platters is constant regardless of which track is being read. To counteract the above disk devolution process, an evolutionary disk storage system is constructed.

Modern hard disks employ zoned bit recording (ZBR). Tracks are grouped into zones (6). ZBR stores more sectors

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

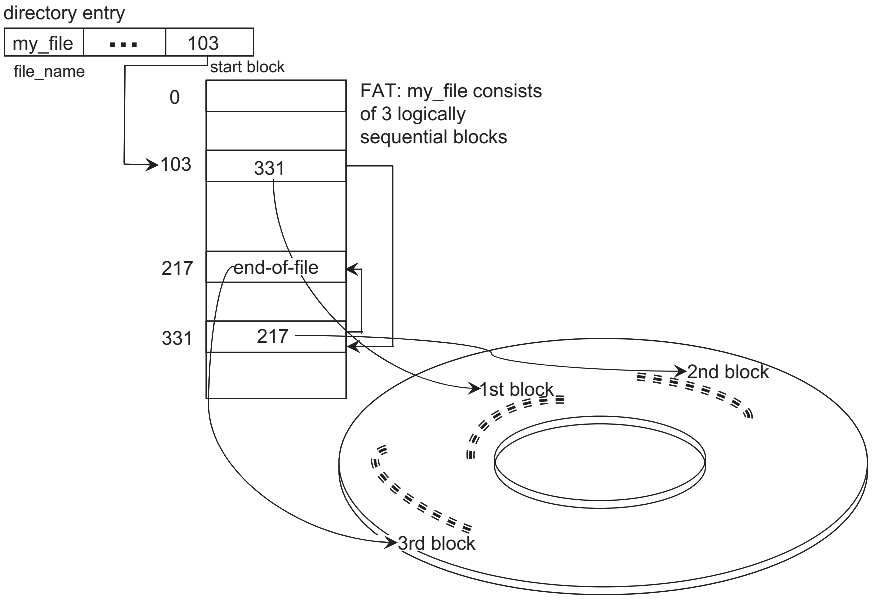


Figure 1. Disk storage replies on a FAT to preserve a file's order in a Microsoft's Windows environment. Each block is one or more sectors.

per track on outer tracks than on inner tracks. The raw data transfer rate of the disk when reading the outside cylinders is much higher than when reading the inside ones.

In the ZBR-based evolutionary disk storage system, the outer 10% of cylinders are grouped into the fast band that absorbs the frequent data, whereas the remaining inner 90% of cylinders, which are grouped into the slow band, contain relatively infrequent data. Object frequency is used to move frequently accessed data to their optimal locations during idle periods, so as not to conflict with system use (7).

The same technique can also be used in a hybrid disk (8). A hybrid hard disk drive was originally concepted by Samsung, which incorporates a flash memory into a magnetic disk. The hybrid disk outplays the above fast band disk at little additional cost on a flash memory.

The combined ultra-high-density benefits of magnetic storage and the low-power and fast read access of NAND technology inspires the construction of redundant arrays of hybrid disks (RAHD; Fig. 3) (9) to offer a possible alternative to today's RAID and/or MAID. Two separable

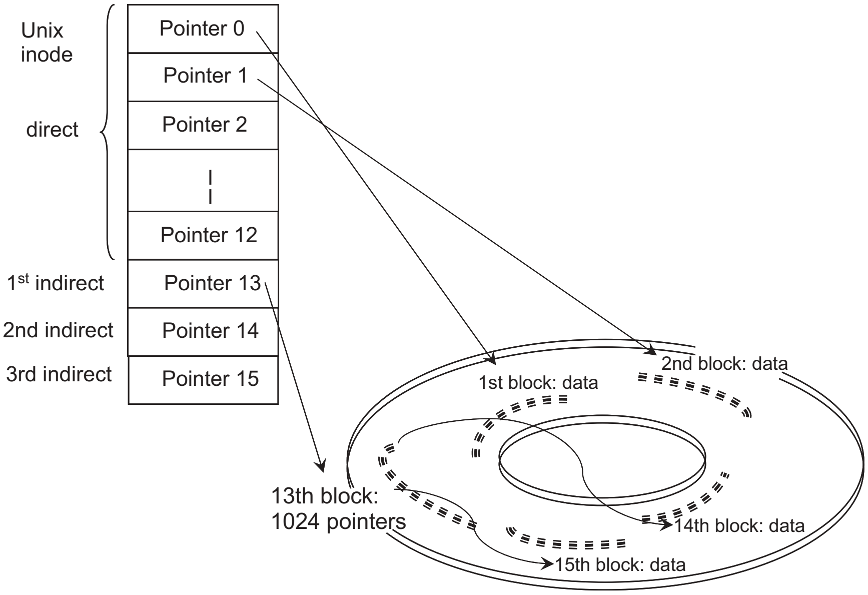


Figure 2. In Unix/Linux environments, indoe is used to preserve a file's order on a disk. The first 12 of these pointers point to direct blocks. The first indirect pointer is the address of a single index block, containing not data, but rather the addresses of blocks that do contain data.

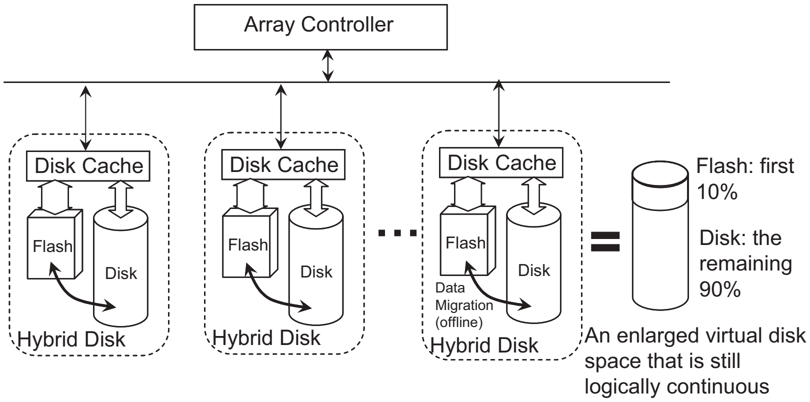


Figure 3. RAHD. Based on the observation that the majority of the disk I/Os are probably going to less than 10% of the total disk space, a flash memory occupies the first 10% of an enlarged disk space that is virtually continuous. Accordingly, the most frequently accessed data blocks are migrated into the flash memory periodically.

working modes have been designed for RAHD arrays: (*1*) high-speed (HS) mode targeting at displaying the full potential speed of hybrid disks; and (*2*) energy-efficient (EE) mode targeting at reducing the energy consumption while maintaining acceptable performance.

In the HS mode, the flash memory always contains frequent data objects. Two schemes have been designed to move the frequent data blocks into the flash memory: (*1*) using the decision tree from data mining to predict the frequencies of data blocks (file attributes of a newly generated file are used to predict its frequency). Blocks with high (predicted) frequency are then written onto the flash memory; (*2*) Tracking the frequencies of data blocks (when the disk is being used) and then migrating the most frequently accessed data blocks into the flash memory during the system idle periods (9). Because of the high skew in common application loads, most requests are likely to be satisfied in the flash memory without bothering the slower disks.

In the EE mode, some “active drives” remain constantly spinning, whereas the remaining “passive drives” are allowed to spin-down after a period of inactivity. A request is directed to the flash in the first instance. If the request is fulfilled by hitting the flash memory, there is no need to awaken the sleeping disk. Otherwise, the request will be forwarded to the disk that is already active or needs to be awakened from sleep (9).

DISK STORAGE APPLICATIONS

Disk storage was originally developed for use with computers. In the twenty-first century, applications for disk storage have expanded beyond computers to include digital video recorders, digital audio players, personal digital assistants, digital cameras, and video game consoles. In 2005, the first mobile phones to include disks were introduced by Samsung and Nokia (10).

Hybrid disks are specially designed for personal and mobile applications, whereas the RAHD can be used more broadly. It is found that a RAHD will provide improvements in performance, power consumption, and scalability. It is a conceptually simple technique for dramatically improving disk storage performance, which is desirable for supercomputers and transaction processing.

BIBLIOGRAPHY

1. J. G. Brookshear, *Computer Science: An Overview*, 6th ed., Reading, MA: Addison-Wesley, 2000
2. http://en.wikipedia.org/wiki/Hard_disk_drive.
3. A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 6th ed., New York: John Wiley, 2002.
4. R. J. T. Morris and B. J. Truskowski. The evolution of storage systems. *IBM Sys. J.*,**42**(2): 2003.
5. F. Wang, S. Wu, N. Helian, A. Parker, Y. Guo, Y. Deng, and V. Khare, Grid-oriented storage: A single-image, cross-domain, high-bandwidth architecture, *IEEE Trans. Comput.*, **56**(4): 2007.
6. http://en.wikipedia.org/wiki/Zone_bit_recording.
7. F. Wang, Y. Deng, N. Helian, S. Wu, V. Khare, C. Liao, and A. Parker, Evolutionary storage: speeding up a magnetic disk by clustering frequent data, *IEEE Trans. Magnetics*, **43**(6):2007.
8. www.samsung.com/, 2006.
9. F. Wang, N. Helian, S. Wu, D. Deng, C. Liao, M. Rashidi, and A. Parker, A case for redundant arrays of hybrid disks (RAHD), *Proc. IEEE INTERMAG Conference*, Madrid, Spain, May 2008.
10. Finally! The Samsung SPH-V5400, world’s first cellphone with a hard drive, Available: engadget.com, September 2004.

Frank Zhigang Wang,
Sining Wu
Derek Deng
Cambridge-Cranfield High Performance
Computing Facility
Cranfield, Bedfordshire, United Kingdom

Na Helian
University of Hertfordshire
Hatfield, Hertfordshire, United Kingdom

E

ENTITY-RELATIONSHIP MODEL

Data modeling is an important phase in the development of a database system. The entity-relationship (ER) model was introduced by Chen (1). This model has been widely used for conceptual data modeling and has become a tool for communication between database designers and end users at the analysis phase of database development. The main constructs of the ER model—entities and relationships and their associated attributes—provide natural modeling concepts of the real world. The ER model has been extended and refined in variuos ways to include additional constructs (2,3) and notations (4).

In this article, we describe the basic concepts in the ER model and some of its extensions. We also discuss how conceptual database design can be carried out using the ER model, the normal form ER diagram, and its properties. Then we present the translation of an ER diagram to the relational database model. Finally, we conclude with several applications of the ER model including view update and schema integration.

BASIC CONCEPTS

The basic concepts in the ER model are entity type and relationship type. An *entity* is an object that exists in our mind and can be distinctly identified. For example, we can have a person entity Ng Hong Kim with IC# 0578936I, or a car entity with licence plate number SBG 3538P. Entities can be classified into different types. An *entity type* contains a set of entities that each satisfy a set of predefined common properties. For example, every person is an instance of the entity type PERSON.

A *relationship* is an association among several entities. For example, we can have a relationship that associates person “Ng Hong Kim” with bank account 50756. A *relationship type* is a set of relationships of the same type that satisfy a set of predefined common properties. Formally, if E1, E2, . . ., En are entity types, then a relationship type R is a subset of the Cartesian product E1 × E2 × . . . × En, that is,

$$\begin{aligned}&\mathrm{R} \subseteq \mathrm{E1} \times \mathrm{E2} \times \ldots \times \mathrm{E}n \qquad \text{or}\\&\mathrm{R} \subseteq \{(\mathrm{e1}, \mathrm{e2}, \ldots, \mathrm{e}n) | \mathrm{ei} \in \mathrm{Ei}, \mathrm{i} = 1, 2, \ldots, n\}\end{aligned}$$

where (e1, e2, . . ., en) is a relationship. For example, we can define the relationship type CUSTACCT to denote the association between customers and the bank accounts that they have, that is,

$$\mathrm{CUSTACCT} \subseteq \mathrm{CUSTOMER} \times \mathrm{ACCT}.$$

The entity types involved in a relationship type are called *participants* of the relationship type. The number of participants in a relationship is called the *degree* of the relationship type. A relationship can be binary, ternary, or even recursive.

An entity type E (or a relationship type R) has attributes that represent the structural (static) properties of E (or R respectively). An *attribute* A is a mapping from E (or R) into a Cartesian Product of n values sets, V1 × V2 × . . . × Vn. If $n \geq 2$, then A is a *composite attribute*. Otherwise, A is a *simple attribute*. For example, Date is a composite attribute with values sets Day, Month, Year. The mapping can be one-to-one (1:1), many-to-one (m:1), one-to-many (1:m), or many-to-many (m:m). If an attribute is a 1:1 or m:1 mapping from E (or R) into the associated value sets, then A is called a *single-valued attribute*, otherwise it is called a *multivalued attribute*.

A minimal set of attributes K of an entity type E that defines a one-to-one mapping from E into the Cartesian product of the associated value sets of K is called a *key* of E. One key of an entity type is designed as the *identifier*. Let K be a set of identifiers of some entity types that participate in a relationship type R. K is called a *key* of the relationship type R if a 1:1 mapping exists from R into the Cartesian product of the associated value sets of K and no proper subset of K has such property. One key of R is designed as the *identifier* of R.

ENTITY-RELATIONSHIP DIAGRAM

The structure (or schema) of a database organized according to the ER approach can be represented graphically using an entity-relationship diagram (ERD). An entity type is represented by the rectangle symbol, whereas a relationship type is represented by the diamond symbol connected by lines to the related entity types. Attributes are denoted by ellipses that are connected by arrowed lines to their owner entity type or relationship type. We use the notation $\leftrightarrow$, $\rightarrow$, $\rightarrow\rightarrow$, $\langle - \rangle\rangle$, to denote that an attribute has a one-to-one (1:1), many-to-one (m:1), many-to-many (m:m), or one-to-many (1:m) mapping, respectively.

Figure 1 shows a simplified ER diagram for a company. The entity type EMP has four attributes: Emp#, Name, Qual, and Child, whereas the entity type PROJ has two attributes, Proj# and Budget. The attributes Emp# and Proj# are the identifiers of the entity types EMP and PROJ, respectively. Hence, they have a mapping of 1:1, as denoted by $\leftrightarrow$. Because each employee has a name and may have many qualifications and each project has a budget, the attributes Name and Budget have a m: 1 mapping, denoted by $\rightarrow$, whereas the attribute Qual has a m:m mapping, denoted by $\rightarrow\rightarrow$. The attribute Child of EMP has a 1:m mapping, which indicates that an employee may have many children but one child can only belong to one employee. The relationship type WORKFOR associates the entity types EMP and PROJ, and it has one attribute Effort. {Emp#, Proj#} is the only key of the relationship type WORKFOR. The cardinalities of EMP and PROJ in WORKFOR are m and m, which indicate that the relationship type WORKFOR between employees and projects is a many-to-many relationship.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

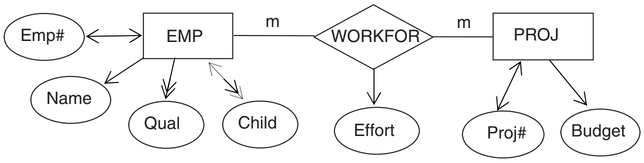


Figure 1. Example ER diagram.

Note that the original ER diagram proposed by Chen (1) does not distinguish between the different mappings for the attributes. As such, one cannot automatically obtain the functional dependencies that involve attributes. The work in Ref. 5 provides a set of heuristics to derive the set of functional dependencies based on the associations among the entity types and their attributes. However, using the arrow notations to specify the mappings for the attributes in the ER diagram as described in Ref. 6, we can directly obtain the functional dependencies as follows:

1. The identifier or 1:1 attributes of an entity type or relationship type functionally determines the single-valued attributes in the entity type or relationship type respectively.
2. The identifier of an entity type or relationship type multidetermines each multivalued attribute in the entity type or relationship type respectively.
3. A 1:m attribute of an entity type or relationship type functionally determines the identifier of the entity type or relationship type.
4. A 1:1 attribute of of an entity type or relationship type functionally determines the identifier of the entity type or relationship type.

For example, in Fig. 1, we can obtain the functional dependencies: Emp# → Name, Emp# →→ Qual, Child → Emp#, Proj# → Budget, {Emp#, Proj#} → Effort.

Figure 2 shows examples of ternary relationship types that involve three participating entity types A, B, and C, as well as their cardinalities. To simplify discussion, we use A, B, and C to denote the identifiers of entity types A, B, and C, respectively. The cardinalities of the participating entity types imply the functional dependencies that hold in the relationship type. Except for the case in which all participating entity types have a cardinality of m (or 1), a participating entity type occurs on the left-hand side of the functional dependency if it has a cardinality of m. Similarly, a participating entity type occurs on the right-hand side of the functional dependency if it has a cardinality of 1. If all the participating entity types have a cardinality of m, as shown in relationship type R1 in Fig. 2(a), then no functional dependency exists in R1. On the other hand, if all the participating entity types have a cardinality of 1, as shown in the relationship type R4 in Fig. 2(d), then the functional dependencies A→ BC, B → AC and C → AB hold in R4. In Fig. 2(b), the functional dependency AB → C in R2, because both A and B have the cardinality m whereas C has the cardinality 1. The functional dependency A → BC holds in R3 [see Fig. 2(c)] because A is the only entity type with the cardinality m.

A relationship type could also have multiple sets of cardinalities for its participating entity types. This finding is illustrated by the relationship type R5 in Fig. 2(e), which has two functional dependencies AB → C and C → A. The first functional dependency AB → C is obtained from the cardinalities m, m, and 1 for A, B, and C, respectively. The second functional dependency C → A is obtained from the cardinalities m, 1 for C and A, respectively. The symbol "/" separates the different sets of cardinalities, whereas the symbol "-" for the participating entity type B in R5 indicates that B is not involved in the second functional dependency. By providing for multiple sets of cardinalities in a relationship type, we can express any number of functional dependencies that involve the participating entity types of the relationship type.

Note that the methods discussed in Refs 5 and 7 cannot express the types of functional dependencies shown in Fig. 2(c–e).

In addition to the specification of key constraints, the ER model also allows *participation constraints* to be specified. The participation constraints of an entity type in a relationship type is denoted by *x:y*, where *x* and *y* denote the minimum and maximum cardinality respectively. These constraints indicate the minimum and maximum number of relationships in which each participating entity can par-

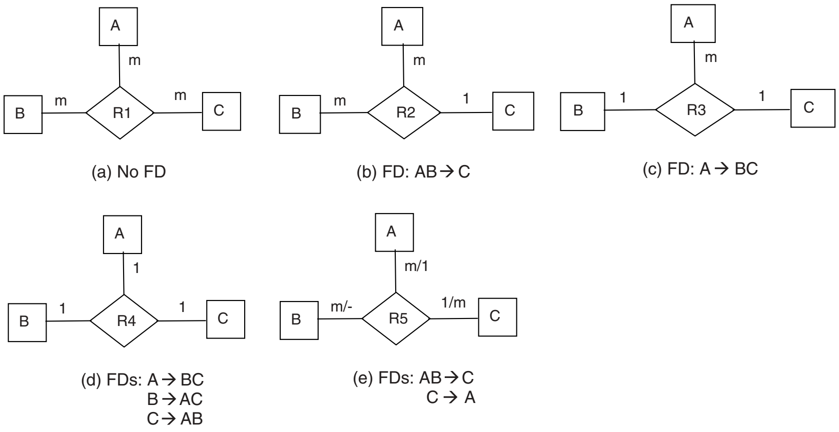


Figure 2. Examples of ternary relationship types.

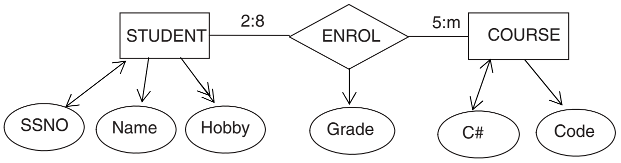


Figure 3. Example to illustrate participation constraints.

ticipate. The values of x and y can be any integer including 0, or m to indicate an unconstrained value. Figure 3 shows the participation constraints of the entity types STUDENT and COURSE in a relationship type ENROL where each student enrolls in a minimum of 2 courses and maximum of 8 courses, and each course should have at least 5 student enrollments with no limit on the class size.

Note that participation constraints cannot be used to express all possible functional dependencies for n-ary relationship types when $n \geq 3$. For example, in Fig. 2(a and b), the participation constraints of all the entity types are either 0:n or 1:n depending on whether the participation is optional or mandatory. Hence, we cannot obtain the functional dependency AB → C for Fig. 2(b) using participation constraints only.

Entity types can be regular or weak. A *weak entity* is one that is existence-dependent on some other entity type, that is, it cannot exist if the other entity does not exist. In Fig. 4(a), the existence of a CHILD entity depends on the existence of an associated EMPLOYEE entity. Thus, if an employee entity is deleted, then its associated child entities are also deleted. The dependent entity type CHILD is a *weak entity type* (represented by a double rectangle), and EMPLOYEE is a *regular entity type*. A relationship type that involves a weak entity type is called *existence-dependency relationship type* (denoted by "EX" together with a relationship type name). Note that an EX relationship type is a one-to-many (1:m) relationship type.

If an entity cannot be identified by the value of its own attributes, but it has to be identified by its relationship with other entities, then such a relationship is called *identifier-dependency relationship type* (denoted by ID). Figure 4(b) shows a typical example of an ID relationship type in which a city can only be uniquely identified together with its country(e.g., the city London can be a city in UK or a city of in the province Ontario in Canada). By definition, an identifier-dependency relationship type is also an existence-dependency relationship type.

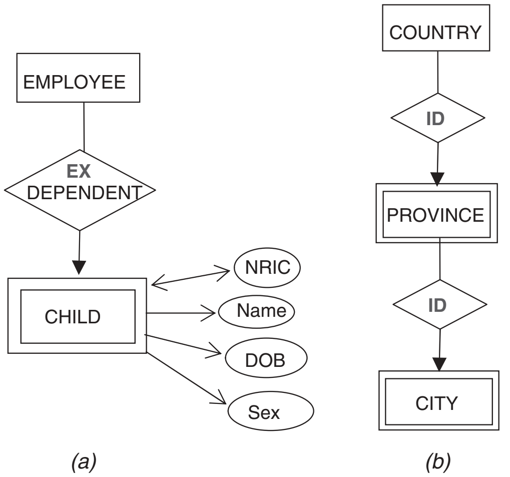


Figure 4. Examples of existence-dependency relationship types and weak entity types.

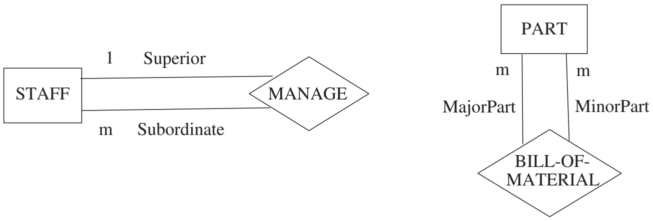


Figure 5. Examples of recursive relationship types.

Relationship types can be *recursive* when they involve an entity type twice. Figure 5 shows the relationship type MANAGE that associates the STAFF entity type twice; one has a *role name* of Superior whereas the other has a role name of Subordinate. Similarly, the relationship type BILL-OF-MATERIAL associates PART entity type twice.

We can also have more than one relationship type between the same set of entity types. Figure 6 shows the two relationship types OWN and LIVE-IN between the entity types PERSON and HOUSE where a person may own many houses but live in one house; a house may be co-owned by several persons and many persons can live in one house.

EXTENDED ER MODEL

The ER model can be extended to include the notions of superclass/subclass relationships between entities as well as specialization and generalization. Set operations such as UNION, INTERSECT, and DECOMPOSE can also be incorporated to link different entity types. We will illustrate these extensions briefly here.

In the real world, entities in an entity type typically can be organized into subgroups that are meaningful and require explicit representation. For instance, in a university database application, the entity type PERSON can be classified even more into STAFF and STUDENT; STAFF can be subclassified into ACAD and ADMIN; STUDENT can be subclassified into UNDERGRAD and GRAD. These entity types can be linked together in an ISA hierarchy as

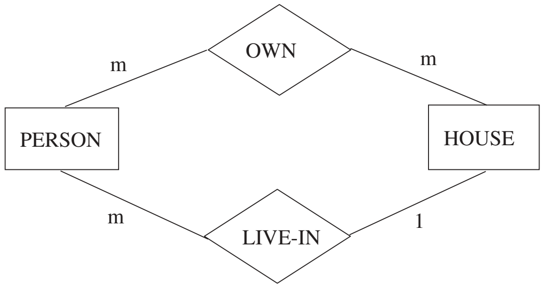


Figure 6. Two relationship types between the same set of entity types.

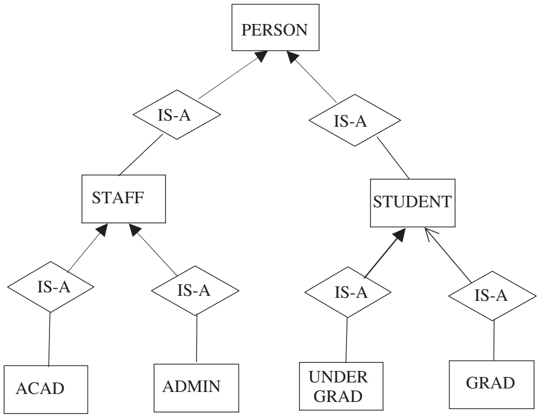


Figure 7. Subtype relationships organized into an ISA hierarchy.

shown in Fig. 7. We call PERSON a superclass of STAFF and STUDENT, whereas STAFF and STUDENT are subclasses of PERSON. The ISA hierarchy is also known as type inheritance. An entity in a subclass represents the same real-world entity in the superclass, and thus it inherits all attributes and relationships of the entity in the superclass in addition to its own specific attributes and relationships.

We can also use set operations to link different entity types together as shown in Fig. 8.

Generalization is the result of taking the union of two or more (lower level) entity types to produce a higher-level entity type. Generalization is the same as UNION, and it is used to emphasize the similarity among lower-level entity types and to hide their differences. *Specialization* is the result of taking a subset of a higher-level entity type to form a lower-level entity type. Specialization is the same as ISA, and its main aim is to highlight the special properties of the lower-level entity types. The attributes of the higher-level entity types are to be inherited by lower-level entity types. Figure 9 shows how an ACCOUNT entity type can be specialized into FIXED-DEP and CHECKING-ACCT entity types. Besides their own specific attributes, that is, FIXED-DEP has InterestRate, StartDate, and MaturityDate, whereas CHECKING-ACCT has OverdraftLimit; these entity types also inherit the attributes Acct#, Name, and Balance from their superclass ACCOUNT. Note that in generalization, every higher-level entity must also be a lower-level entity, whereas specialization does not have this constraint.

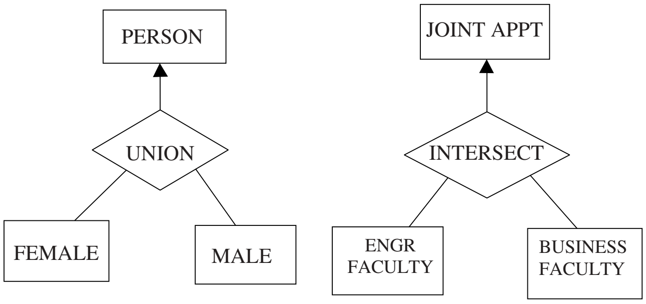


Figure 8. Using set operators to link entity types.

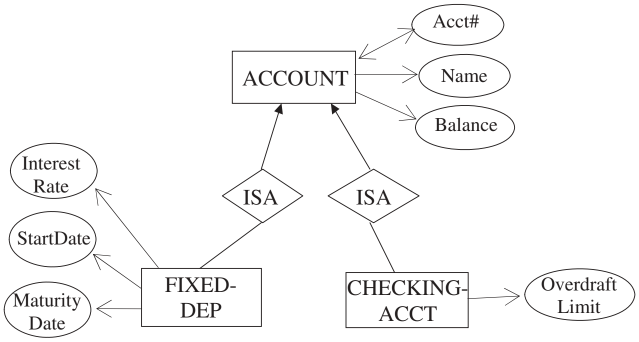


Figure 9. Specialization of ACCOUNT entity type into SAVING-ACCT and CHECKING-ACCT.

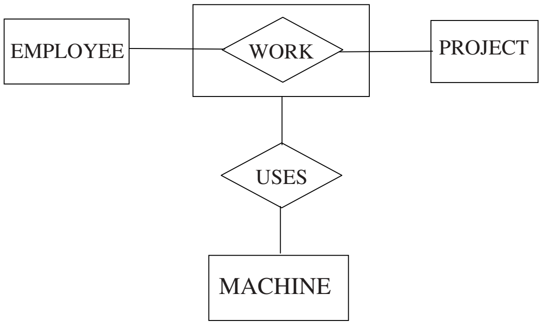


Figure 10. Using aggregations to model relationships among relationships.

Other extensions to the ER model include allowing aggregations and composite constructs to model relationships among relationship types and complex object classes (complex entity types). *Aggregation* is an abstraction through which relationships are treated as higher-level entities. The ER diagram in Fig. 10 treats the relationship type WORK, which is defined between the entity types EMPLOYEE and PROJECT, as a higher-level entity type called WORK. We can then define a many-to-many relationship type between WORK and MACHINE, that is, each work uses many machines and a machine can be used by many works. Note that if the cardinality of a relationship type is not explicitly specified, then by default, it is many-to-many.

We can also use a construct called COMPOSITE to construct complex objects (complex entities) from some other simple and/or complex objects (see Fig. 11). The

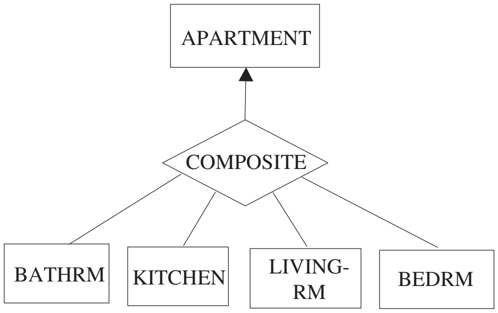


Figure 11. Using the composite construct to model complex objects.

complex object and its component objects are not necessarily type (or object) compatible.

CONCEPTUAL DATABASE DESIGN USING ER MODEL

Using the ER approach for database design has three advantages. First, it is database management system independent. Second, the designer can concentrate on “information requirements” and not on “storage or access efficiency”. Third, the ER diagram is easy to understand by users and database designers. Five main steps to database design exist using the ER approach.

Step 1. Identify the entity types and the attributes that clearly belong to them.

Step 2. Identify the relationship types and their participating entity types as well as the attributes that clearly belong to the relationship types.

Step 3. Identify the remaining attributes, keys, and identifiers of each entity type and relationship type and obtain an ER diagram.

Step 4. Convert the ER diagram to a normal form ER diagram (NF-ER).

Step 5. Translate the NF-ER diagram to a relational database schema (or other data models such as object-oriented data model).

We will cover the first three steps in this section and discuss Steps 4 and 5 separately in the subsequent two sections.

To accomplish steps 1 to 3 properly, the database designer not only has to interview users but also must study the system specification documents that are written in some natural language, such as English. The work in Ref. 8 presented guidelines/rules for translating English sentences into ER diagrams. We list some guidelines below.

R1. A common noun (such as student and employee) in English corresponds to an entity type in an ER diagram.

Note that proper nouns such as John, Singapore, and New York City are entities, not entity type.

R2. A transitive verb in English corresponds to a relationship type in an ER diagram. Note that a transitive verb must have an object.

For example, a person may own one or more cars and a car is owned by only one person. The cardinalities of PERSON and CAR in the OWNS relationship type are 1 and m, respectively, and the participation of PERSON in the OWNS relationship type is optional as some people may not own any car [see Fig. 12(a)].

R3. An adjective in English corresponds to an attribute of an entity type in an ER diagram.

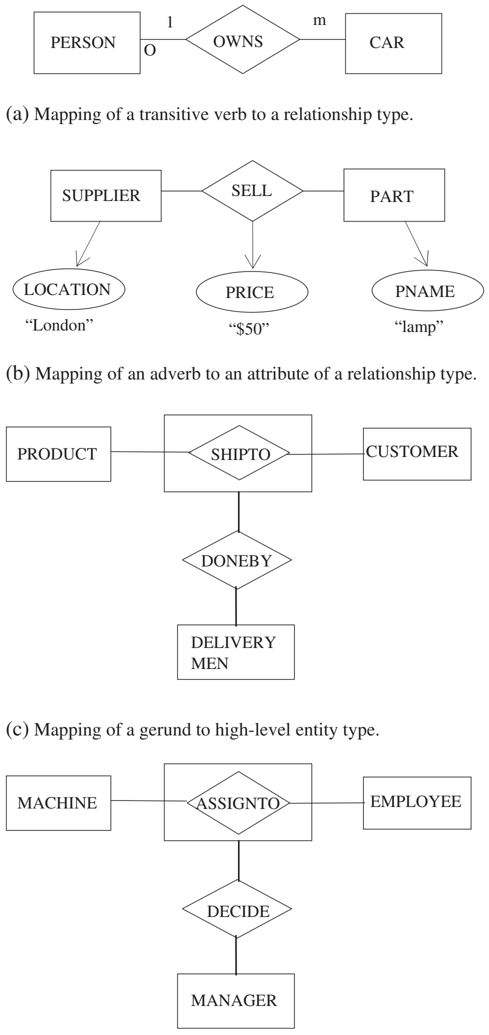
R4. An adverb in English corresponds to an attribute of a relationship type in an ER diagram.

For example, the ER diagram in Fig. 12(b) models a London supplier sells a part with part name lamp for $50.

R5. A gerund in English corresponds to a high-level entity type (or aggregation) converted from a relationship type in an ER diagram. A gerund is a noun in the form of the present participle of a verb.

For example, “shopping” in the sentence “I like shopping.” Figure 12(c) shows the ER diagram that models the situation in which products are shipped to customers, and the shipping is done by delivery men.

R6. A clause in English is a high-level entity type abstracted from a group of interconnected low-level entity and/or relationship types in an ER diagram. A clause is a group of words that contains a subject and a verb, but it is usually only part of a sentence.



(a) Mapping of a transitive verb to a relationship type.

(b) Mapping of an adverb to an attribute of a relationship type.

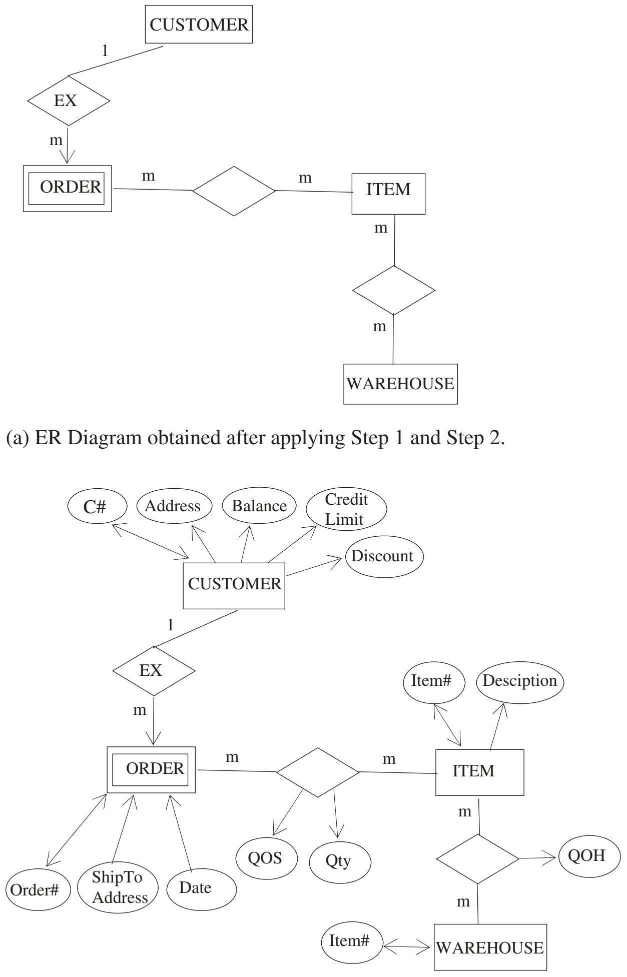
(c) Mapping of a gerund to high-level entity type.

(d) Mapping of a clause to high-level entity type

Figure 12. Examples to illustrate the translation of english sentences into ER diagrams.

For example, Fig. 12(d) models the case in which managers decide which machine is assigned to which employee.

Example 1. We will now work through a case study to illustrate how an ERD can be obtained. Suppose we need to design a database for an order-entry system. The database will capture the following information about customers, items, and orders. For each customer, we record the customer number (unique), valid "ship to" address (several per customer), balance, credit limit, and discount. For each order, we have a header that contains customer number, "ship to" address, date of order, and detail lines (several per order), each giving the item number and quantity ordered. For each item, we record the item number (unique), warehouses, quantity on hand at each warehouse, and item description. For internal processing reasons, a quantity outstanding value is associated with each detail line of each order. Figure 13(a) shows the ERD after performing Step 1 and Step 2. After Step 3, we will obtain the ERD as shown in Fig. 13(b).



(a) ER Diagram obtained after applying Step 1 and Step 2.

(b) ER Diagram obtained after applying Step 3 by adding attributes.

Figure 13. Example to illustrate conceptual database design using the ER diagram.

TRANSLATION OF (NORMAL FORM) ER DIAGRAM TO RELATIONAL MODEL

After obtaining the conceptual design in ER diagram, we can now map it to the relational model. The resulting relational schema may not be in normal form and we can use the normalization theory in relational model to derive a third normal form relational schema. Alternatively, we can first convert the ER diagram to a normal form (6) before mapping it to the relational model, and the relational schema obtained is guaranteed to be in normal form. In this section, we will discuss the steps to translate an ER diagram to the relational schema. The algorithm to obtain a normal form ER diagram will be presented in the next section.

Step 1. Assign role names to certain arcs in a cycle in the ER diagram to conform to the universal relation assumption. Here, a cycle in an ER diagram is defined as a cycle in the corresponding graph of the ER diagram in which all entity types and relationship types are nodes in the graph and arcs that connect entity types and relationship types are edges in the graph, except for cycles formed only by ISA, UNION, INTERSECT, and DECOMPOSE special relationships.

Step 2. Assign identifiers for entity types involved in special relationship such as ISA, UNION, INTERSECT, and DECOMPOSE.

Step 3. Generate relations for each entity type.

a) All the m:1 and 1:1 attributes of an entity type E form a relation. The keys and identifier of E are the keys and primary key of the generated relation.
b) Each m:n attribute and the identifier of E form an all key relation.
c) Each 1:m attribute and the identifier of E form a relation with the 1:m attribute as its key.

Note that all composite attributes are replaced by their components in the generated relations.

Step 4. Generate relations for each regular relationship type R.

a) All the identifiers of the entity types participating in R and all the m:1 and 1:1 attributes of R form a relation. Keys and 1:1 attributes of R are keys of the generated relation, and the identifier of R is the primary key of the generated relation.

   Furthermore, if A → B is a full functional dependency in the generated relation and A is not a key of R, then we record A → B as a constraint of the relation generated.
b) Each m:m attribute of R and the identifier of R form an all key relation.
c) Each 1:m attribute of R and the identifier of R form a relation with the 1:m attribute as the key of the relation.

Example 2. We will use the ER diagram in Fig. 14 to illustrate the translation process. The role names AX, AY,

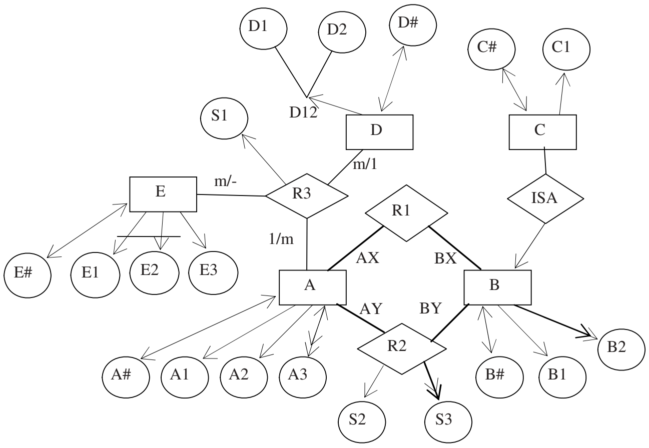


Figure 14. Example to illustrate translation of ER diagram to relations.

BX, and BY are assigned by Step 1. Note that the relationship type R3 has two functional dependencies, namely {E#, D#} → A#, and A# → D#, we will have {E#, D#} and {A#, E#} as the keys of R3, and we designate {E#, D#} as the identifier of R3. The relations generated are as follows:

For entity type A, AE1 A#, A1, A2) and AE3(A3, A#).

For entity type B, BE1(B#, B1) and BE2(B#,B2).

For entity type C, CE1(C#, C1).

Note that C# is generated and we have the constraint C# ISA B# of BE1, that is, CE1[C#] ⊆ BE1[B#].

For entity type D, DE1(D#, D1, D2).

Note that D12 is replaced by D1 and D2.

For entity type E, EE1(E#, E1, E2, E3) where E# is primary key.

For relationship type R1, R1R1(AX, BX).

Note that AX and BX are generated, and we have the constraints AX ISA A# and BX ISA B#.

For relationship type R2, R2R1(AY, BY, S2) and R2R2(AY, BY, S3).

Again, AY and BY are generated and we have the constraints AY ISA A# and BY ISA B#.

For relationship type R3, R3R1(A#, E#, D#, S1).

{D#, E#} is the primary key and we have the constraint A# → D#. Note that R3R1 is in 3NF but not in BCNF. Furthermore no relation is generated for the ISA relationship. Instead, it is translated to C# ISA B#.

Similarly, the ER diagram in Fig. 15 can be translated to the following relations.

Entity relations: Employee (E#, Name, Salary)
Employee_Hobby (E#, Hobby)
Employee_skill (E#, Skill) Project (P#, Pname, Budget)
Relationship relation: Emp_Proj (E#, P#, Progress)

NORMAL FORM ER DIAGRAM

The work in Ref. 6 introduced the notion of a normal form ER diagram. The objectives for defining a normal form for (ERD) are threefold. First, it aims to capture and preserve all the semantics of the real world of a database that can be expressed in terms of functional, multivalued, and join dependencies, by representing them explicitly in the ERD. Second, it ensures that all the relationship types represented in the ERD are nonredundant. Finally, the normal form ERD ensures that all the relations translated from the ERD are in good normal form: either in 3NF or 5NF.

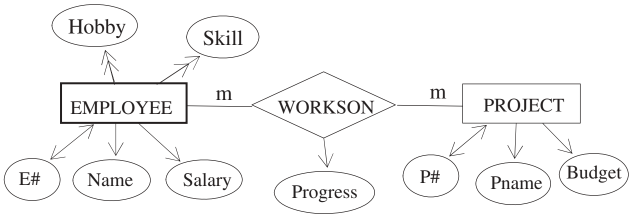


Figure 15. Example ER diagram.

A normal form ERD may consist of composite attributes, multivalued attributes, weak entity types and special relationships such as existence dependent, identifier dependent, ISA, UNION, INTERSECT, and DECOMPOSE relationships. The main steps for deriving a normal form ERD are as follows:

1. Define the set of basic dependencies of an entity type
2. Define the entity normal form (E-NF)
3. Define the set of basic dependencies of a relationship type
4. Define the relationship normal form (R-NF)
5. Define the set of basic dependencies of an ERD
6. Define the normal form ERD (ER-NF)
7. Convert an ERD to a normal form ERD

Definition 1. The set of *basic dependencies of entity type* E with identifier K, denoted by BD(E), is defined as:

(i) For each m:1 attribute A of E, we have K → A ∈ BD(E)
(ii) For each 1:m multivalued attribute A of E, we have A → K ∈ BD(E)
(iii) For each 1:m or m:n multivalued attribute A of E, we have K→→ A ∈ BD(E)
(iv) For each key K1 of E, K1 ≠ K, K → K1 ∈ BD(E) and K1 → K ∈ BD(E)
(v) No other FDs or MVDs are in BD(E).

Definition 2. An entity type E is in E-NF if and only if all the given FDs and MVDs which only involve the attributes of E, can be derived from BD(E).

Informally, BD(E) is the set of functional dependencies and multivalued dependencies involving only the attributes of E, which are explicitly represented in the ERD.

Consider the ERDs in Fig. 16. The entity type EMPLOYEE in Fig. 16(a) has the set of basic dependencies BD(EMPLOYEE) = {E# → SSNO, Name, Gender; SSNO →

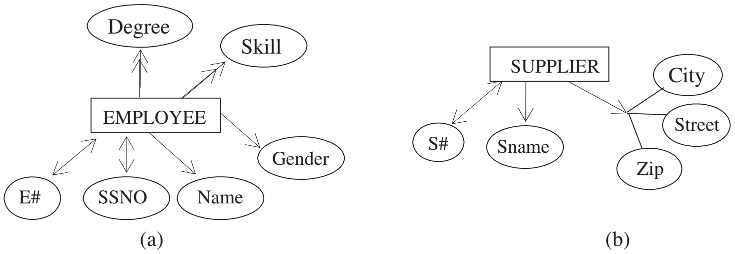


Figure 16. ER diagrams to illustrate basic dependencies in entity types.

E#; E# →→ Skill; E# →→ Degree}. Furthermore, EMPLOYEE is in E-NF. In contrast, the entity type SUPPLIER in Fig. 16(b) is not in E-NF. It is because we have BD(SUPPLIER) = {S# → Sname, S# → {City, Street, Zip}}. However, the two known functional dependencies {City, Street} → Zip, and Zip → City cannot be derived from BD(SUPPLIER).

Definition 3. Let R be a relationship type with identifier K, and $\mathcal{F}$ be the associated set of FDs that only involve the identifiers of the set of entity types participating in R. The set of basic dependencies of R, BD(R) is defined as:

(i) For each 1:1 attribute A of R, we have $K \rightarrow A \in BD(R)$ and $A \rightarrow K \in BD(R)$

(ii) For each m:1 single valued attribute A of R, we have $K \rightarrow A \in BD(R)$

(iii) For each 1:m multivalued attribute A of R, we have $A \rightarrow K \in BD(R)$

(iv) For each 1:m or m:m multivalued attribute A of R, we have $K \rightarrow\rightarrow A \in BD(R)$

(v) Let $A \rightarrow B$ be a full functional dependency in $\mathcal{F}$ such that A is a set of identifiers of entity types participating in R, and B is the identifier of some entity type participating in R. If A is a key of R or B is part of key of R, then we have $A \rightarrow B \in BD(R)$

(vi) No other FDs or MVDs are in BD(R).

In other words, BD(R) is the set of functional dependencies and multivalued dependencies involving only the identifiers of the participating entity types of R and attributes of R that are explicitly represented in the ERD.

Definition 4. An relationship type R of an ERD is in R-NF if and only if all the given FDs and MVDs that only involve the attributes of R and the identifiers of the entity types participating in R are implied by BD(R).

Consider the ERDs in Figure 17. ABD is the identifier of relationship type R1 in Fig. 17(a). BD(R1) = ABD → EG, G → ABD, ABD →→ F, ABD → C} and R1 is in R-NF.

Suppose we have $C \rightarrow D \in \mathcal{F}$, then C → D is also in BD(R1) by definition and hence R1 is still in R-NF. In Fig. 17(b), we have BD(R2) = {AB → CD} and $C \rightarrow D \notin BD(R2)^+$. Hence, R2 is not in R-NF.

Definition 5. The set of basic dependencies of an ERD D, denoted by BD(D), is defined as the union of the sets of basic dependencies of all the entity types of D and the sets of basic dependencies of all relationship types of D.

Definition 6. An ERD D is in normal form (ER-NF) if is satisfies the following conditions:

(1) All attribute names are distinct and of different semantics.

(2) Every entity type in the ERD is in E-NF.

(3) Every relationship type in the ERD is in R-NF.

(4) All given relationships and dependencies are implied by BD(D).

(5) No relationship type (including its attributes) can be derived from other relationship types by using join and projection.

Note that condition 1 in Definition 6 is required in order to conform to the universal relation assumption; condition 2 ensures that all relations generated for all entity types are in 5NF; condition 4 ensures that ERD has captured all relationship types and dependencies of the given database; conditions 3 and 5 ensure that all relations generated for all regular relationship types are either in 3NF or in 5NF, and no relation exists in BCNF but not in 4NF or 5NF, that is, no

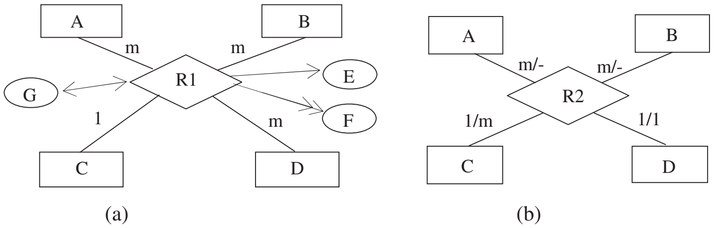


Figure 17. ER Diagram to illustrate basic dependencies in relationship types.

relationship type (together with its attributes) can be derived from other relationship types using join and projection operations.

We have the following important property.

Theorem. The relations generated for each of entity type of a NF-ER diagram are in 5NF. The relations generated for each of the relationship type of a NF-ER diagram are either in 3NF or 5NF.

Based on the above definitions, we can now convert an ERD to a normal form ERD as follows.

Step 1. Ensure that all attribute names are unique and of different semantics to conform to the Universal Relation Assumption.

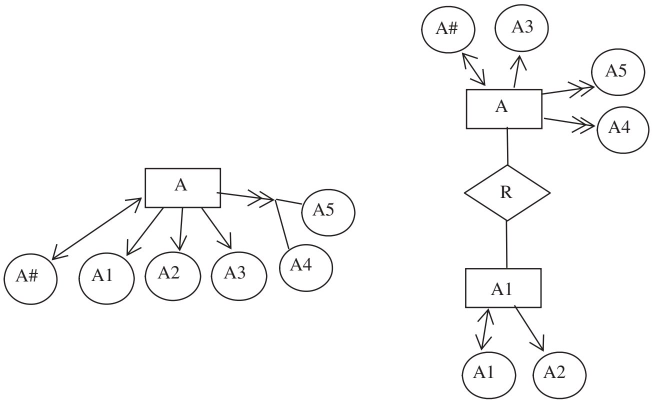
Step 2. Convert any non E-NF entity type to E-NF. Remove all undesirable FDs and/or MVDs by introducing new entity types and relationship types.

Step 3. Convert any non–R-NF relationship type to R-NF. Remove all undesirable FDs, MVDs, and/or JDs by introducing new entity types and relationship types or by splitting the relationship type into smaller ones.

Step 4. Remove relationship types that have no associated attributes and are equal to the join and/or projection of other relationship types as they can be derived.

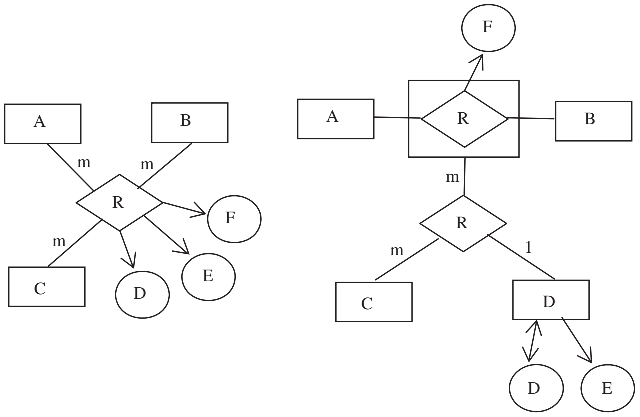
We will now give two examples on how to convert an ERD that is not in NF-ERD to one that is in NF-ERD. Details of the conversion process can be found in Ref. 6.

Example 3. Consider the ERD in Fig. 18(a). If the entity type A has additional dependencies A1 → A2 and A# →→ A4, then A is not in E-NF. To make A into entity normal form because A1 → A2 ∉ $BD(A)^+$, we convert the attribute A1 into an entity type with identifier A1 and a single-valued attribute A2, and create a binary relationship type R between A and A1. Furthermore, because A# →→ A4 ∉ $BD(A)^+$, we split the multivalued composite attribute into two multivalued attributes A4 and A5. The resulting ERD is shown in Fig. 18(b). Note that the two dependencies A1 → A2 and A# →→ A4 are now explicitly represented in Fig. 18(b). The entity types A and A1 are in E-NF.



(a) A is not in E-NF (b) A is now in E-NF

Figure 18. ER diagram to illustrate entity normal form.



(a) R is not in E-NF (b) ERD is now in NF-ERD

Figure 19. Example to illustrate normal form ERD.

Example 4. Next, let us examine the ERD in Fig. 19(a). If the relationship type R has additional dependencies AB → F and D → E, then R is not in R-NF. Because D → E ∉ $BD(R)^+$, we convert D into an entity type with identifier D and attribute E, and connect the entity type D with the relationship type R. Because AB → F ∉ $BD(R)^+$, we create a new binary relationship type R1 between A and B with single-valued attribute F. We make R1 as a high-level entity type (aggregation) and redefine the relationship type R as a relationship type that associates this high-level entity type R1 and entity types C and D. The resulting ERD shown in Fig. 19(b) is a NF-ERD.

CONCLUSION

In this chapter, we have reviewed the basic concepts of the ER model as well as some commonly used extensions. The semantics captured in the ER model provides for the intuitive modeling of real-world concepts. We have discussed how the conceptual design of a database can be carried out using the ER model and the translation to the relational database. We have also shown the properties of a normal-form ERD and how it can be obtained.

Besides conceptual database design, the semantics in the ER model also play an important role in view updates (9) and schema integration (10,11). Views are external schemas. They increase the flexibility of a database by allowing multiple users to observe the data in different ways. Views also provide logical independence by allowing some changes to be made to the conceptual schema without affecting the application programs. For a view to be useful, users must be able to apply retrieval and update operations to it. These update operations on the view are allowed only if they can be translated into the corresponding update operations on the conceptual schema instances.

Reference 12 describes how one can automatically generate the external-to-conceptual mapping and the conceptual-to-internal mapping of an ER based database management system. Using this mapping, retrievals from a view can always be mapped into equivalent retrievals from the conceptual schema. A mapping is also required to translate view updates into the corresponding updates on the conceptual schema. However, such a mapping does not always exist, and even when it does exist, it may not be unique. We can model views using ER diagrams and use the semantics captured to help resolve any ambiguity when translating view updates. Ling and Lee (8) develop a theory within the framework of the ER approach that characterizes the conditions under which mappings exist from view updates into conceptual schema updates. Concepts such as virtual updates and three types of insertions are introduced.

Schema integration is essential to define a global schema that describes all the data in existing databases participating in a distributed or federated database management system. Lee and Ling (10) describe how the semantics in the ER model can facilitate schema integration. For example, the ER model can easily help in the resolution of structural conflicts, that is, when related real world concepts are modeled using different constructs in different schemas. An attribute in one schema can be transformed into an equivalent entity type in another schema without any loss of semantics or functional dependencies which approaches based on the relational model have not considered. Furthermore, from the ER model, we can conclude that an entity type in one schema cannot be equivalent to (or modeled as) a relationship set in another schema, that is, this type of conflict inherently does not exist.

The ER model captures semantics that are essential for view update translation and meaningful schema integration. Many semantics cannot be captured by the relational model, and therefore, view update translation and schema integration using the relational model is limited.

BIBLIOGRAPHY

1. P. P. Chen, The entity-relationship model-toward a unified view of data, *ACM Trans. Database Syst.*, **1**(1): 9–36, 1976.
2. T. J. Teory, D. Yang, and J. P. Fry, A logical design methodology for relational databases using the extended entity-relationship model, *ACM Comput. Surv.*, **18**(2): 197–222, 1986.
3. V. C. Storey, Relational database design based on the entity-relationship model, *IEEE Trans. Data Knowl. Engineer*. **7**: 47–83, 1991.
4. T. H. Jones and I.-Y Song, Binary representations of ternary relationships in ER conceptual modelling, *14th International Conference on Object-Oriented and Entity-Relationship Modelling*, 1995, pp. 216–225.
5. S. Ram, Deriving functional dependencies from the entity-relationship model, *Communi. ACM*, 1995, pp. 95–107.
6. T. W. Ling, A normal form for entity-relationship diagrams, *4th International Conference on Entity-Relationship Approach*, 1985, pp. 24–35.
7. T. J. Teory, D. Yang, and J. P. Fry, A logical design methodology for relational database using the extended entity-relationship model, *ACM Comput. Surv*. **18**(2): 1986.
8. P. P. Chen, English sentence structure and entity-relationship diagrams, *Informat. Sci.* **29**(2–3): 127–149, 1983.
9. T. W. Ling and M. L. Lee, A theory for entity-relationship view updates, *11th International Conference on Entity-Relationship Approach*, 1992, pp. 262–279.
10. M. L. Lee and T. W. Ling, Resolving structural conflicts in the integration of entity-relationship schemas, *14th International Conference on Object-Oriented and Entity-Relationship Modelling*, 1995, pp. 424–433.
11. M. L. Lee and T. W. Ling, Resolving constraint conflicts in the integration of entity-relationship schemas, *16th International Conference on Entity-Relationship Approach*, 1997, pp. 394–407.
12. T. W. Ling and M. L. Lee, A prolog implementation of an ER based DBMS, *10th International Conference on ER Approach*, 1991, pp. 587–605.

TOK WANG LING
National University of
Singapore

MONG LI LEE
National University of
Singapore

MULTIAGENT SYSTEMS

INTRODUCTION

A multiagent system (MAS) is a system composed of several computing entities called "agents." Being a sub-discipline of distributed artificial intelligence (DAI), multiagent systems research represents a new paradigm for conceptualizing, designing, and implementing large-scale and complex systems that require spatially, functionally, or temporally distributed processing (1–3).

Agents in a MAS typically have distributed resource, expertise, intelligence, and processing capabilities, and they need to work collaboratively to solve complex problems that are beyond the capabilities of any individuals. MAS research covers a very broad areas, including multiagent learning, distributed resource planning, coordination, and interagent communications, to mention only a few (1,4,5), and many studies have insights drawn from other disciplines, such as game theories, communications research, and statistical approaches. In addition to the wide applications in industries (e.g., Network-Centric Warfare, air-traffic control, and trading agents), multiagent systems have also been used in simulating, training, and supporting collaborative activities in human teams.

Different perspectives exist on multiagent systems research. From the AI perspective, people may focus on fundamental issues such as coordination algorithms, agent architectures, and reasoning engines. From the engineering perspective, people may concern system building methodologies, property verifications, and agent-oriented programming. Detailed reviews of MAS from several distinct perspectives have been provided in Sycara (3), Stone and Veloso (2), Bond and Gasser (6), O'Hare and Jennings (7), and Huhns and Singh (8). The objective of this article is to briefly present a view of MAS from the perspective of multiagent teamwork. The remainder is organized as follows. The next section introduces the concept of shared mental models that underpin team-based agent systems. Then some generic multiagent architectures in the literature are described, and the issue of interagent coordination is discussed. Communication and helping behaviors are reviewed, and a summary is given in the last section.

SHARED MENTAL MODELS

The notion of shared mental models (SMMs) is a hypothetical construct that has been put forward to explain certain coordinated behaviors of human teams (6,9). An SMM produces a mutual situation awareness, which is the key for supporting many interactions within a team that lead to its effectiveness and efficiency (10). The scope of shared mental models is very broad, which may involve shared ontology (11), common knowledge and/or beliefs (12), joint goals/intentions (13), shared team structures (5), common recipes (14), shared plans (15), and so on.

A shared ontology provides the common vocabulary for a group of agents to communicate directly. Without a shared ontology, agents can communicate only through a "broker" who provides translations between different ontologies. Because of its importance, ontology is covered in KQML (Knowledge Query and Manipulation Language), and many efforts such as DARPA Agent Markup Language (DAML) have been aimed to facilitate sharing ontology on the semantic web (11).

Agents in a MAS typically have a collection of overlapped knowledge/beliefs that serve as a common basis for the agents to understand and respond to each other's behaviors. Such common knowledge/beliefs may be the description of domain tasks (up to certain levels of detail), the communication protocols to be used, the social laws or social normatives to follow, and so on.

Some MASs also allow individual agents to have a partial picture of the organizational structure (5), which may include information regarding membership of a group, sub-group relations, predetermined group leader, roles each member can play, capability requirements on each role, and so forth. Having a well-defined structure does not imply that the structure is static. A system can still have flexibility in changing its structure through dynamically assigning responsibility to the agents in the system. Having a shared team structure enables an agent to develop a higher level abstraction about the capabilities, expertise, and responsibilities of other agents. It is important for an agent to initiate helping behaviors proactively.

Having a shared objective (goal) is a key characteristic of agent teams (13,16–18), a kind of tightly coupled MASs. A common goal can serve as a cohesive force that binds team members together (16). To distinguish team behavior from coordinated individual behavior (individuals' goal happen to be the same), a notion of joint mental attitude (i.e., joint intention) is introduced based on the concept of joint persistent goal (13). A joint intention can be viewed as a shared commitment to perform a collective action to achieve a shared goal. The joint intentions theory requires a team of agents with a joint intention to not only each try to do its part in achieving the shared goal but also commit to informing others when an individual agent detects that the goal has been accomplished, becomes impossible to achieve, or becomes irrelevant. Thus, having a joint intention not only means all the team members have established the same goal, but also it means that they have committed to maintaining the consistency about the dynamic status of the shared goal. The joint intentions theory is important because it not only offers a framework for studying numerous teamwork issues, but also it provides a foundation for implementing multiagent systems (4).

Joint intentions prescribe how agents should behave when certain things go wrong; it thus indicates that robust multiagent systems can be implemented to work in a dynamic environment if agents can monitor joint intentions and rationally react to changes in the environment.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Jennings pointed out that this is desirable but not sufficient (14). He proposed the joint responsibility model with the explicit representation of cooperation using common recipes. A common recipe provides a context for the performance of actions in much the same way as the joint goal guides the objectives of the individuals (14). Mainly focusing on handling unexpected failures, the joint responsibility model refines Cohen and Levesque's work (16,19) on joint intentions and explicitly captures different causes of recipe failures. In particular, it clearly specifies the conditions under which an agent involved in a team activity should reconsider its commitments (three related to joint goal commitment: goal has been attained, goal will never be attained, or goal motivation no longer present; four related to common recipe commitment: desired outcome is available, or recipe becomes invalid, untenable, or violated). The model furthermore describes how the agent should behave both locally and with respect to its fellow team members if any such situations develop. It should drop the commitment and must endeavor to inform all other team members so that futile activities can be stopped at the earliest possible opportunity.

Although Jennings gave a simple distributed planning protocol for forming teams and for achieving agreement on the details (i.e., timing constraints, and actual performers) of common recipe (14), Grosz and Kraus prescribed a more general process for collaboratively evolving partial shared plans into complete shared plans (15,20). A shared plan is characterized in a mental-state view as a particular collection of beliefs and intentions. An agent is said to have a shared plan with others if and only if the agent works toward establishing and maintaining those required mental attitudes, and it believes that the other agents do so likewise. The common recipes in the SharedPlan theory and in the joint responsibility model have different meanings. In the joint responsibility model, each team member will have the same picture of the common recipe being adopted. Whereas in the SharedPlan theory, even though the common recipe is complete from the external, different team members may have different partial views of the common recipe being considered. In pursuing their common goals, it is the shared plans that ensure team members cooperate smoothly rather than prohibit each other's behavior, which may occur otherwise because of the partial views of common recipes.

Jennings admitted that an agent must track the ongoing cooperative activity to fulfill its social roles (14). However, it is insufficient to allow agents to monitor simply the viability of their commitment through continuously monitoring the relevant events that may occur within the system, as is implemented in GRATE*. The monitoring of teamwork progress is also important for agents to gain awareness of the current context so that they can (*1*) better anticipate others' needs (e.g., relevant information needs) and avoid performing irrelevant actions (e.g., communicating irrelevant information), and (*2*) dynamically adapt the team process to external changes. For example, the already executed part of a team process may never be re-invoked again; thus, the associated information-needs become no longer active. A team process may include decision points, each of which can specify several potential ways to proceed based on the current situation. Each team member needs to track the team's choices at these points because they may affect significantly their information-needs in the future. The CAST (5) system implements a richer notion of shared mental models that covers the dynamic status of team processes. Each member of a CAST agent team can monitor dynamically the progress of team activities that are represented internally as Petri nets.

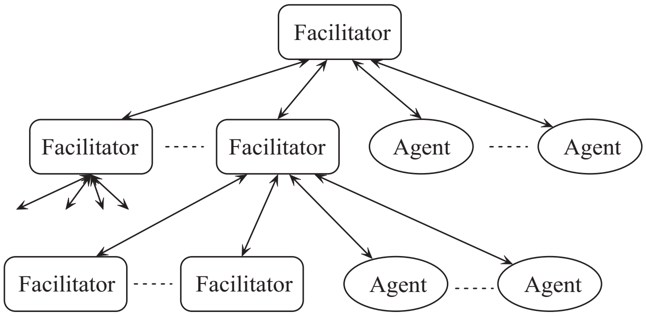


Figure 1. Facilitator-based architecture.

MULTIAGENT ARCHITECTURES

We here focus on multiagent architectures only. Individual agent architectures such as reactive architectures and BDI-style architectures (21), are beyond the scope.

Multiagent architectures can be classified into two categories: facilitator-based architectures and layered architectures. Figures 1 and 2 illustrate a generic facilitator-based architecture and a generic layered architecture, respectively.

In a facilitator-based architecture, facilitator agents play an important role in linking individual agents. Typically, a facilitator maintains knowledge that records the capabilities of a collection of agents or subfacilitators and uses that knowledge to establish connections between service requesters and providers. As well as providing a general notion of transparent delegation, facilitators also offer a mechanism for organizing an agent society in a hierarchical way. OAA (Open Agent Architecture) (22) is a representative of facilitator-based architectures. OAA adopts a blackboard-based framework that allows individual agents to communicate by means of goals posted on blackboard controlled by facilitator agents. Basically, when

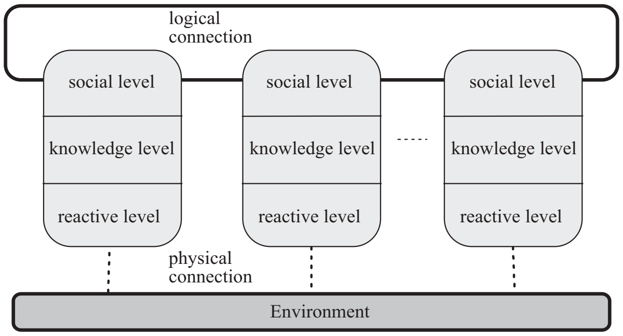


Figure 2. Layered architecture.

a local facilitator agent determines that none of its subordinate agents can achieve the goal posted on its blackboard, it propagates the goal to a higher level blackboard controlled by a higher level facilitator, who in turn can propagate the goal to its superior facilitator if none of its subsidiary blackboards can handle the goal.

Most existing multiagent systems have a layered architecture that can be divided into three levels (see Fig. 2): reactive level, knowledge level, and social level (3). Agent behavior at the reactive level responds to external changes perceived directly from the environment. Deliberative (or reflective) behaviors are often covered at the knowledge level, where an agent achieves situation awareness by translating the external input into internal representations and may project into the future (refer to the Endsley's PCP model of situation awareness in Ref. 23). Social or collaborative behaviors are reflected at the social level, where an agent identifies collaboration needs, adjusts self behavior, and exchanges information to coordinate joint behaviors.

The representatives with such an architecture include TEAMCORE (24), RETSINA (25), JACK (26), and CAST (5).

TEAMCORE is an extension of STEAM (a Shell for TEAMwork) (4), which takes the perspective of team-oriented programming. STEAM is a hybrid teamwork model built on top of the SOAR architecture (27). STEAM borrows insights from both the joint intentions theory and the SharedPlans formalism. It uses joint intentions as a building block to hierarchically build up the mental attitude of individual team members and to ensure that team members pursue a common solution path. STEAM exhibits two valuable features: selective communication and a way of dealing with teamwork failures. In STEAM, communication is driven by commitments embodied in the joint intentions theory as well as by explicit declaration of information-dependency relationships among actions. To make a decision on communication, STEAM agents take into consideration the communication costs, benefits, and the likelihood that some relevant information may be mutually believed already. To handle failures, STEAM uses role-monitoring constraints (AND, OR, dependency) to specify the relationship of a team operator and an individual's or subteam's contributions to it. When an agent is unable to complete actions in its role and the embedding team operator is still achievable, the remaining agents will invoke a repair plan accordingly.

TEAMCORE realized a wrapper agent by combining the domain-independent team expertise initially encoded in STEAM and the domain-specific knowledge at the team level. Hence, a TEAMCORE wrapper agent is a purely social agent that only has core teamwork capabilities. This team-readiness layer hides the details of the coordination behavior, low-level tasking, and replanning (24). To map TEAMCORE to Fig. 2, the TEAMCORE wrapper agent lies at the social level, whereas domain agents implemented using SOAR encompass functionalities required by the knowledge and reactive levels.

RETSINA multiagent infrastructure (25) (RETSINA-MAS) is extended from the RETSINA individual agent architecture (28). A RETSINA agent has four components: Communicator, HTN Planner, Enabled Action Scheduler, and Execution Monitor. RETSINA-MAS agents interact with each other via capability-based and team-oriented coordination as follows. Initially all agents have a commonly agreed partial plan for fulfilling a team task (goal). Each agent then matches his/her capabilities to the requirements of the overall team goal within the constraints of his/her authority and other social parameters. This process will produce a set of candidate roles for the agent, who can select some and communicate to teammates as proposals for his/her role in the team plan. Once the team members have reached a consensus that all plan requirements are covered by the role proposals without any conflicts, they can commit to executing the team plan. The team-oriented coordination implemented in RETSINA-MAS lies at the social level, whereas the RETSINA architecture covers reactive and deliberative behaviors.

JACK Teams (26) is an extension to JACK Intelligent Agents that provides a team-oriented modeling framework. JACK agents are BDI-style agents each with beliefs, desires, and intentions. A JACK team is an individual reasoning entity that is characterized by the roles it performs and the roles it requires others to perform. To form a team is to set up the declared role obligation structure by identifying particular subteams capable of performing the roles to be filled. JACK Teams has constructs particularly for specifying team-oriented behaviors. For instance, *Teamdata* is a concept that allows propagation of beliefs from teams to subteams and vice versa. Statements *@team_achieve* and *@parallel* are used in JACK for handling team goals. An *@parallel* allows several branches of activity in a team plan to progress in parallel. An *@parallel* statement can specify success condition, termination condition, how termination is notified, and whether to monitor and control the parallel execution. JACK Teams can also be viewed as a team wrapper that provides programmable team-level intelligence.

CAST (Collaborative Agents for Simulating Teamwork) (5) is a team-oriented agent architecture that supports teamwork using a shared mental model among teammates. The structure of the team (roles, agents, subteams, etc.) as well as team processes (plans to achieve various team tasks) are described explicitly in a declarative language called MALLET (29). Statements in MALLET are translated into PrT nets (specialized Petri-Nets), which use predicate evaluation at decision points. CAST supports predicate evaluation using a knowledge base with a Java-based backward-chaining reasoning engine called JARE. The main distinguishing feature of CAST is proactive team behavior enabled by the fact that agents within a CAST architecture share the same declarative specification of team structure and team process. Therefore, every agent can reason about what other teammates are working on, what the preconditions of the teammate's actions are, whether the teammate can observe the information required to evaluate a precondition, and hence what information might be potentially useful to the teammate. As such, agents can figure out what information to deliver proactively to teammates and can use a decision-theoretic cost/benefit analysis of the proactive information delivery before actually communicating. Compared with the architectures mentioned above, the layered behavior in CAST is flexible: It depends more on the input language than on the architecture. Both individual reactive behaviors and planned team activities can be encoded in MALLET, whereas the CAST kernel dynamically determines the appropriate actions based on the current situation awareness and on the anticipation of teammates' collaboration needs.

COORDINATION

Agents in MAS possess different expertise and capabilities. A key issue to a multiagent system is how it can maintain global coherence among distributed agents without explicit global control (13). This process requires agent coordination.

Distributed agents need to coordinate with one another when they pursue a joint goal, but the achievement of the goal is beyond the capability, knowledge, or capacity of any individuals. Team tasks in multiagent systems can be classified into three categories: atomic team actions, coordinated tasks, and planned team activities.

Atomic team actions refer to those atomic actions that cannot be done by a single agent and must involve at least two agents to do it. For instance, lifting a heavy object is a team operator. Before doing a team operator, the associated preconditions should be satisfied by all agents involved, and the agents should synchronize when performing the action. The team-oriented programming paradigm (hereafter TOP) (30) and CAST support atomic team operators. In CAST, the number of agents required by a team operator can be specified as constraints using the keyword *num*. For example, the following MALLET code specifies a team operator called *co_fire* that requires at least three agents firing at a given coordinate simultaneously:

```
(toper co_fire (?x ?y) (num ge 3)...).
```

By coordinated tasks, we refer to those short-term (compared with long-term) activities involving multiple agents. Executing a coordinated task often requires the involved agents to establish joint and individual commitments to the task or subtasks, to monitor the execution of the task, to broadcast task failures or task irrelevance whenever they occur, and to replan doing the task if necessary. A coordinated task is composed typically of a collection of temporally or functionally related subtasks, the assigned doers of which have to synchronize their activities at the right time and be ready to backup others proactively. STEAM (4) uses role-constraints (a role is an abstract specification of a set of activities in service of a team's overall activity) to specify the relationship between subtasks of a coordinated task. An AND-combination is used when the success of the task as a whole depends on the success of all subtasks; An OR-combination is used when any one subtask can bring success to the whole task; and role-dependency can be used when the execution of one subtask depends on another. Complex joint team activities can be specified by using these role-constraints combinatively and hierarchically. Similarly, in CAST (5), the Joint-Do construct provides a means for describing multiple synchronous processes to be performed by the identified agents or teams in accordance with the specified share type. A share type is either AND, OR, or XOR. For an AND share type, all specified subprocesses must be executed. For an XOR, exactly one subprocess must be executed, and for an OR, one or more subprocesses must be executed. A Joint-Do statement is not executed until all involved team members have reached this point in their plans. Furthermore, the statement after a Joint-Do statement in the team process does not begin until all involved team members have completed their part of the Joint-Do.

Planned team activities refer to common recipes that govern the collaboration behaviors of teammates in solving complex problems. A planned team activity is a long-term process that often involves team formation, points of synchronization, task allocation, execution constraints, and temporal ordering of embedded subactivities. GRATE (14) has a recipe language, where trigger conditions and structure of suboperations can be specified for a recipe. STEAM (4) uses the notion of team operator to prescribe the decomposition of task structures. RETSINA-MAS (25) also uses the concept of shared plans to coordinate individual behaviors, but it lacks an explicit team plan encoding language. Instead of providing a higher level planning encoding language, JACK Teams (26) tried to extend a traditional programming language (i.e., Java) with special statements for programming team activities. In JACK, team-oriented behaviors are specified in terms of roles using a construct called *teamplan*. TOP (30) uses *social structures* to govern team formation, and it is assumed that each agent participating in the execution of a joint plan knows the details of the whole plan.

In MALLET, plans are decomposable higher level actions, which are built on lower level actions or atomic operators hierarchically. A plan specifies which agents (variables), under what preconditions, can achieve which effects by following which processes, and optionally under which conditions the execution of the plan can be terminated. The process component of a plan plays an essential role in supporting coordination among team members. A process can be specified using constructs such as sequential (SEQ), parallel (PAR), iterative (WHILE, FOREACH, FORALL), conditional (IF), and choice (CHOICE).

MALLET has a powerful mechanism for dynamically binding agents with tasks. The *AgentBind* construct introduces flexibility to a teamwork process in the sense that agent selection can be performed dynamically based on the evaluation of certain teamwork constraints (e.g., finding an agent with specific capabilities). For example,

```
(AgentBind (?f)
  (constraints (playsRole ?f fighter)
  (closestToFire ?f ?fireid)))
```

states that the agent variable ?f needs to be instantiated with an agent who can play the role of fighter and is the closest to the fire ?fireid (?fireid already has a value from the preceding context). The selected agent is then responsible for performing later steps (operators, subplans, or processes) associated with ?f. An agent-bind statement becomes eligible for execution at the point when progress of the embedding plan has reached it, as opposed to being executed when the plan is entered. The scope for the binding to an agent variable extends to either the end of the plan in which the variable appears or the beginning of the next agent-bind statement that binds the same variable, whichever comes first. AgentBind statements can be anywhere in a plan as long as agent variables are instantiated before they are used. External semantics can be associated with the constraints described in an AgentBind statement. For instance, a collection of constraints can be ordered increasingly in terms of their priorities. The priority of a constraint represents its degree of importance compared with others. In case not all constraints can be satisfied, the constraints with the least priority will be relaxed first.

Agents also need to coordinate when to make choices on the next course of actions. The *Choice* construct in MALLET can be used to specify explicit choice points in a complex team process. For example, suppose a fire-fighting team is assigned to extinguish a fire caused by an explosion at a chemical plant. After collecting enough information (e.g., nearby chemicals or dangerous facilities), the team needs to decide how to put out the fire. They have to select a suitable plan among several options. The *Choice* construct is composed of a list of branches, each of which invokes a plan (a course of actions) and is associated with preference conditions and priority information. The preference conditions of a branch describe the situation in which the branch is preferred to others. If the preference conditions of more than one branch are satisfied, the one with the highest priority is chosen. In implementation (31), some specific agents can be designated as decision makers at the team level to simplify the coordination process. For instance, at a choice point, each agent can check whether it is the designated decision maker. If it is, the agent evaluates the preference conditions of the potential alternatives based on the information available currently. If no branch exists whose preference condition can be satisfied, the agent simply waits and reevaluates when more information becomes available. If more than one selectable branch exists, the agent can choose one randomly from those branches with the highest priority. The agent then informs others of the chosen branch before performing it. For those agents who are not the designated decision maker, they have to wait until they are informed of the choice from the decision maker. However, while waiting, they still could help in delivering information proactively to the decision maker to make better decisions.

COMMUNICATION

Communication is essential to an effective functional team. For instance, communication plays an important role in dynamic team formation, in implementing team-oriented agent architectures, and more theoretically, in the forming, evolving, and terminating of both joint intentions (16) and SharedPlans (15). Interagent communications can be classified into two categories: reactive communications and proactive communications.

Reactive communications (i.e., ask/reply) are used prevalently in existing distributed systems. Although the ask/reply approach is useful and necessary in many cases, it exposes several limitations. First, an information consumer may not realize certain information it has is already out of date. If this agent needs to verify the validity of every piece of information before they are used (e.g., for decision-making), the team can be overwhelmed easily by the amount of communications entailed by these verification messages. Second, an agent may not realize it needs certain information because of its limited knowledge (e.g., distributed expertise). For instance, a piece of information may be obtained only through a chain of inferences (e.g., being fused according to certain domain-related rules). If the agent does not have all the knowledge needed to make such a chain of inferences, it will not be able to know it needs the information, not to mention request for it.

Proactive information delivery means “providing relevant information without being asked.” As far as the above-mentioned issues are concerned, proactive information delivery by the information source agents offers an alternative, and it shifts the burden of updating information from the information consumer to the information provider, who has direct knowledge about the changes of information. Proactive information delivery also allows teammates to assist the agent who cannot realize it needs certain information because of its limited knowledge.

In fact, to overcome the above-mentioned limitations of “ask,” many human teams incorporate proactive information delivery in their planning. In particular, psychological studies about human teamwork have shown that members of an effective team can often anticipate the needs of other teammates and choose to assist them proactively based on a shared mental model (32).

Interagent communication has been studied extensively (38). For instance, many researchers have been studying agent communication languages (ACLs), by which agents in distributed computing environments can share information. KQML (34) and FIPA’s ACL (35) are two attempts toward a standardized ACL. A complete ACL often covers various categories of communicative acts, such as assertives, directives, commissives, permissives, prohibitives, declaratives, and expressives. Some reseachers even argued for the inclusion of proactives (i.e., proactive performatives) (36).

The mental-state semantics of ACL is one of the most developed areas in agent communication, where most efforts are based on Cohen and Levesque’s work (19). For instance, the semantics of FIPA’s performatives are given in terms of *Attempt*, which is defined within Cohen and Levesque’s framework (35). The semantics of proactive performatives are also treated as attempts but within an extended SharedPlans framework (36).

To understand fully the ties between the semantics of communicative acts and the patterns of these acts, conversation policies or protocols have been studied heavily in the ACL field (33). More recently, social agency is emphasized as a complement to mental agency because communication is inherently public (37), which requires the social construction of communication to be treated as a first-class notion rather than as a derivative of the mentalist concepts. For instance, in Ref. 37, speech acts are defined as social commitments, which are obligations relativized to both the beneficiary agent and the whole team as the social context.

Implemented systems often apply the joint intentions theory (13) in deriving interagent communications. The joint intentions theory requires that all agents involved in a joint persistent goal (JPG) take it as an obligation to inform other agents regarding the achievement or impossibility of the goal. Communication in STEAM (4) is driven by commitments embodied in the joint intentions theory. STEAM also integrated decision-theoretic communication selectivity: Agents deliberate on communication necessities vis-à-vis incoherency, considering communication costs and benefits as well as the likelihood that some relevant information may be mutually believed already.

GRATE*(14), which was built on top of the joint responsibility model, relies even more on communications. As we mentioned, the joint responsibility model clearly specifies the conditions under which an agent involved in a team activity should reconsider its commitments. For instance, as well as communications for dropping a joint intention, an agent should also endeavor to inform all other team members whenever it detects that either one of the following happens to the common recipe on which the group are working: The desired outcome is available already, the recipe becomes invalid, the recipe becomes untenable, or the recipe is violated.

The strong requirement on communication among teammates by the joint intentions theory is necessary to model coherent teamwork, but in a real case, it is too strong to achieve effective teamwork; enforced communication is not necessary and even impossible in time-stress domains. Rather than forcing agents to communicate, CAST (5) allows agents to anticipate others' information needs, which depending on the actual situations, may or may not result in communicative actions. The proactive information delivery behavior is realized in the DIARG (Dynamic Inter-Agent Rule Generator) algorithm. Communication needs are inferred dynamically through reasoning about the current progress of the shared team process. The following criteria are adopted in Ref. 38. First, a team process may include choice (decision) points, each of which can specify several branches (potential ways) to achieving the goal associated with the choice point. Intuitively, those information needs that emerge from the branches should not be activated until a specific branch is selected. Second, a team or an agent may dynamically select goals to pursue. The information needed to pursue one goal may be very different from those needed in another. Third, the already executed part of a team process may never be reinvoked again; thus, the associated information needs become no longer "active." DIARG is designed to generate interagent communication based on the identified information needs and on the speaker's model of others mental models. For instance, an agent will not send a piece of information if the possibility of the information being observed by the potential receiver is high enough. A decision-theoretic approach is also employed in CAST to evaluate the benefits and cost of communications.

HELPING BEHAVIORS

Pearce and Amato (39) have developed an empirically-derived taxonomy of helping behaviors. The model has a threefold structure of helping: (*1*) doing what one can (*direct help*) versus giving what one has (*indirect help*), (*2*) spontaneous help (informal) versus planned help (formal), and (*3*) serious versus nonserious help. These three dimensions correspond to the type of help offered, the social setting where help is offered, and the degree of need of the recipient.

Helping behavior in MASs can be defined as helping other team members perform their roles under conditions of poor workload distributions or asymmetric resource allocations or as helping team members when they encounter unexpected obstacles. As indicated by the above taxonomy, helping behaviors can come in many different forms, such as emergency aids and philanthropic acts. However, *planned direct help* and *spontaneous indirect help* are the two predominant forms of helping behaviors in MASs. For example, agents for an operational cell in simulated battlefields often have contingency plans in responding to environmental uncertainties. Collaborative agents also tend to deliver relevant information only to information consumers by first filtering irrelevant information away.

Helping behaviors can also be classified as *reactive* helping and *proactive* helping. Much helping that occurs in MASs is reactive: in response to specific requests for help. For instance, in the RoboRescue domain, fire fighters ask police agents to clear a blocked area when detecting such a place. A fire-fighting team asks another one for help when the fire is getting out-of-control.

Proactive helping refers to the helping that is not initiated by requests from recipients but by the anticipation of others' needs from shared mental models—even if those needs are not expressed directly (40). Proactive helping often occurs in highly effective human teams. For instance, providing assistance to others who need it has been identified as a key characteristic of human teamwork (41).

The joint intentions theory and the SharedPlans theory are two widely accepted formalisms for modeling teamwork; each has been applied successfully in guiding the design and the implementation of multiagent systems, such as GRATE*(14), STEAM (4), and CAST (5). Both theories allow agents to derive helping behaviors. In particular, the joint intentions theory implies that an agent will intend to help if it is mutually known that one team member requires the assistance of the agent (13).

Grosz and Kraus even proposed axioms for deriving helpful behaviors (15,20). For instance, axiom A5 and A6 in (15) state that an agent will form a potential intention to do ALL the actions it thinks might be helpful. whereas Axiom 2 in Ref. 20 states that if an agent intends that a property *p* hold and some alternative actions exist the agent can take that would lead to *p* holding, then the agent must be in one of three potential states: (*1*) The agent holds a potential intention to do some of these actions, (*2*) the agent holds an intention to do some of these actions; or (*3*) the agent has reconciled all possible actions it could take and has determined they each conflict in some way with other intentions. These two treatments are actually consistent; they characterize "intending-that" from two perspectives. The former shows how potential intentions are triggered from intentions. The latter reflects the process of means-ends reasoning: An agent first adopts a potential intention, then reconciles the potential intention with existing intentions—either adopts it as an actual intention or drops it and considers other options, and then it abandons—if all potential intentions serving the same ends have been tried but none can be reconciled into actual intentions. For instance, suppose an agent A has an intention to make the property p true. A will adopt a potential intention to do action α if the performance of α will enable another agent B to perform some action β, which would directly make p true. Ask/reply is an instance of using this axiom. Enabling other's physical actions also falls into this category. For example, a logistics person supplies ammunition to a fighter in a joint mission.

However, the axioms in the SharedPlans theory are still not rich enough to cover helping behaviors involving three or more parties. Such behaviors occur predominantly in large hierarchical teams with subteams. For instance, as a team scales up in size, the team is often organized into subteams, each of which may be divided even more into smaller subteams, and so on. In such cases, team knowledge might be distributed among several subteams. Hence, agents in one subteam might not be able to anticipate the information needs of agents in other subteams because they may not share the resources for doing so, such as the subteam process, the plans, and the task assignments. To enable information sharing among subteams, some agents in a subteam are often designated as the point of contacts with other subteams. For example, an agent who simultaneously participates in the activities of two subteams can be designated as the broker agent of the two subteams. These broker agents play a key role in informing agents outside the subteam about the information needs of agents in the subteam. Such helping behaviors involving more than two parties can be accounted for by the axiom given in Ref. 42, which establishes a basis for third-party communicative acts.

SUMMARY

Multiagent systems offer a new way of analyzing, designing, and implementing large-scale, complex systems, and they have been applied in an increasing range of software applications. Here, we discussed the concept of shared mental models, multiagent architectures, coordination, communications, and helping behaviors, which are critical in developing team-based multiagent systems. This article certainly is not an exhaustive summary of MAS research; it simply covers a limited view of the MAS research from teamwork perspective. Readers are encouraged to refer to the key conferences in MAS area (such as AAMAS, KIMAS, IAT), which have been attracting a growing number of researchers to present, demonstrate, and share their systems and ideas.

BIBLIOGRAPHY

1. V. R. Lesser, Multiagent systems: an emerging subdiscipline of AI source, *ACM Comp. Surv.*, **27**(3): 340–342, 1995.
2. P. Stone and M. Veloso, Multiagent systems: a survey from a machine learning perspective, *Autonomous Robots*, **8**(3): 345–383, 2000.
3. K. Sycara, Multi agent systems, *AI Magazine***19**(2): 1998.
4. M. Tambe, Towards flexible teamwork, *J. Artificial Intell. Res.*, **7**: 83–124, 1997.
5. J. Yen, J. Yin, T. Ioerger, M. Miller, D. Xu and R. Volz, CAST: collaborative agents for simulating teamwork, *In Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001, pp. 1135–1142.
6. A. H. Bond and L. Gasser, Readings in Distributed Artificial Intelligence. San Francisco, CA: Morgan Kaufmann, 1988.
7. G. O'Hare and N. Jennings, *Foundations of Distributed Artificial Intelligence*. New York: Wiley, 1996.
8. M. Huhns and M. Singh, *Readings in Agents*, San Francisco, CA: Morgan Kaufmann, 1997.
9. K. Sycara and C. M. Lewis, Forming shared mental models, *Proceedings of the Thirteenth Annual Meeting of the Cognitive Science Society*, Chicago, IL: 1991, pp. 400–405.
10. J. Orasanu, Shared mental models and crew performance, *Proceedings of the 34 Annual Meeting of the Human Factors Society*, Orlando, FL, 1990.
11. D. Fensel, I. Horrocks, F. V. Harmelen, D. L. McGuinness and Peter F. Patel-Schneider, oil: an ontology infrastructure for the semantic web, *IEEE Intell. Sys.*, **16**(2): 38–45, 2001.
12. R. Fagin, J. Y. Halpern, Y. Moses and M. Y. Vardi, *Reasoning About Knowledge*, Cambridge, MA: MIT Press, 1995.
13. P. R. Cohen and H. J. Levesque, Teamwork, *Nous*, **25**(4): 487–512, 1991.
14. N. R. Jennings, Controlling cooperative problem solving in industrial multi-agent systems using joint intentions, *Artificial Intelligence*, **75**(2): 195–240, 1995.
15. B. Grosz and S. Kraus, Collaborative plans for complex group actions, *Artificial Intelligence*, 269–358, 1996.
16. P. R. Cohen and H. J. Levesque, On team formation, in J. Hintikka and R. Tuomela (eds.), *Contemporary Action Theory*, 1997.
17. J. Searle, Collective intentions and actions, in P. R. Cohen, J. Morgan and M. E. Pollack, eds., *Intentions in Communication*. Cambridge, MA: MIT Press, 1990, pp. 401–416.
18. R. Tuomela and K. Miller, We-intentions. *Philos. Stud.* **53**: 367–389, 1988.
19. P. R. Cohen and H. J. Levesque, Rational interaction as a basis for communication, in *Intentions in Communication*, MIT Press, 1990, pp. 221–225.
20. B. Grosz and S. Kraus, The evolution of sharedplans, in A. Rao and M. Wooldridge (eds.), *Foundations and Theories of Rational Agencies*, 1998, pp. 227–262.
21. A. S. Rao and M. P. Georef, BDI-agents: from theory to practice, *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, CA, 1995.
22. D. Martin, A. Cheyer and D. Moran, The Open agent architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, **13**(1-2): 91–128, 1999.
23. M. R. Endsley, Towards a theory of situation awareness in dynamic systems, *Human Factors*, **37**: 32–64, 1995.

24. D. V. Pynadath, M. Tambe, N. Chauvat and L. Cavedon, Toward team-oriented programming, in *Agent Theories, Architectures, and Languages*, 1999, pp. 233–247.

25. J. A. Giampapa and K. Sycara, Team-Oriented Agent Coordination in the RETSINA Multi-Agent System, tech. report CMU-RI-TR-02-34, Robotics Institute, Carnegie Mellon University, 2002.

26. JACK Teams manual. Available: http://www.agentsoftware.com/shared/ demosNdocs/JACK-Teams-Manual.pdf, 2003.

27. J. Laird, A. Newell and P. Rosenbloom, SOAR: an architecture for general intelligence, *Artificial Intelligence*, **33**(1): 1–64, 1987.

28. K. Sycara, K. Decker, A. Pannu, M. Williamson and D. Zeng, Distributed intelligent agents, *IEEE Expert, Intelli. Syst. Applicat.* **11**(6): 36–45, 1996.

29. X. Fan, J. Yen, M. Miller and R. Volz, The Semantics of MALLET—an agent teamwork encoding language, *2004 AAMAS Workshop on Declarative Agent Languages and Technologies*, 2004.

30. G. Tidhar, Team oriented programming: preliminary report. Technical Report 41, AAII, Australia, 1993.

31. J. Yen, X. Fan, S. Sun, T. Hanratty and J. Dumer, Agents with shared mental models for enhancing team decision-makings, *Decision Support Sys.,* Special issue on Intelligence and Security Informatics, 2004.

32. J. A. Cannon-Bowers, E. Salas and S. A. Converse, Cognitive psychology and team training: training shared mental models and complex systems, *Human Factors Soc. Bull.*, **33**: 1–4, 1990.

33. F. Dignum and M. Greaves, Issues in agent communication, *LNAI* 1916, Springer-Verlag, Berlin, 2000.

34. Y. Labrou and T. Finin, Semantics for an agent communication language, in M. Wooldridge, M. Singh, and A. Rao (eds), *Intelligent Agents IV: Agent Theories, Architectures and Languages* (LNCS 1365), 1998.

35. FIPA: Agent Communication Language Specification. Available: http://www.fipa.org/, 2002.

36. J. Yen, X. Fan and R. A. Volz, *Proactive communications in agent teamwork,* (F. in Dignum, ed., Advances in Agent Communication LNAI-2922), Springer, 2004, pp. 271–290.

37. M. P. Singh, Agent communication languages: Rethinking the principles, *IEEE Computer*, **31**(12): 40–47, 1998.

38. X. Fan, J. Yen, R. Wang, S. Sun and R. A. Volz, Context-centric proactive information delivery, *Proceedings* of the 2004 IEEE/WIC *Intelligent Agent Technology Conference*, 2004.

39. P. L. Pearce and P. R. Amato, A taxonomy of helping: a multidimensional scaling analysis, *Social Psychology Quart*, **43**(4): 363–371, 1980.

40. M. A. Marks, S. J. Zaccaro and J. E. Mathieu, Performance implications of leader briefings and team interaction training for team adaptation to novel environments, *J. Applied Psychol.*, **85**: 971–986, 2000.

41. T. L. Dickinson and R. M. McIntyre, A conceptual framework for teamwork measurement, in M. T. Brannick, E. Salas, and C. Prince (Eds), *Team Performance Assessment and Measurement: Theory, Methods and Applications*, 1997, pp. 19–44.

42. X. Fan, J. Yen and R. A. Volz, A theoretical framework on proactive information exchange in agent teamwork, *Artificial Intell.*, **169**: 23–97, 2005.

FURTHER READING

B. Grosz and C. Sidner, Plans for discourse, in P. Cohen, J. Morgan and M. Pollack (eds.), *Intentions in Communication*, Cambridge, MA: MIT Press, 1990, pp. 417–444.

M. N. Huhns, L. M. Stephens, Multiagent systems and societies of agents, G. Weiss (ed.), In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Cambridge, MA: MIT Press, 1999, pp. 79–120.

M. Wooldridge, *An Introduction to MultiAgent Systems*, New York: John Wiley & Sons, 2002.

M. Wooldridge and N. Jennings, Intelligent agents: theory and practice. *Knowledge Engine. Rev.*, **10**(2): 115–152, 1995.

M. J. Wooldridge and N. R. Jennings, Pitfalls of Agent-Oriented Development, *Proc. of the 2nd Int. Conf. on Autonomous Agents*, Minneapolis MN, 1998, pp. 385–391.

XIAOCONG FAN
The Pennsylvania State University
Erie, Pennsylvania

OBJECT-ORIENTED DATABASES

Traditional database management systems (DBMSs), which are based on the relational data model, are adequate for business and administrative applications and are characterized by data with a very simple structure and by the concurrent execution of several not-so-complex queries and transactions. The rapid technologic evolution raised, since the early 1980s, new application requirements for which the relational model demonstrated as inadequate. The relational model, indeed, is not suitable to handle data typical of complex applications, such as design and manufacturing systems, scientific and medical databases, geographical information systems, and multimedia databases. Those applications have requirements and characteristics different from those typical of traditional database applications for business and administration. They are characterized by highly structured data, long transactions, data types for storing images and texts, and nonstandard, application-specific operations. To meet the requirements imposed by those applications, new data models and DBMSs have been investigated, which allow the representation of complex data and the integrated specification of domain-specific operations.

Getting a closer look at the large variety of applications DBMSs are mainly used by, we can distinguish different types of applications, with each characterized by different requirements toward data handling. The most relevant application types include *business applications*, which are characterized by large amounts of data, with a simple structure, on which more or less complex queries and updates are executed, which must be accessed concurrently by several applications and require functionalities for data management, like access control; *complex navigational applications*, such as CAD and telecommunications, that need to manipulate data whose structures and relationships are complex and to efficiently traverse such relationships; *multimedia applications*, requiring storage and retrieval of images, texts and spatial data, in addition to data representable in tables, that require the definition of application-specific operations, and the integration of data and operations from different domains. Relational DBMSs handle and manipulate simple data; they support a query language (SQL) well suited to model most business applications, and they offer good performance, multiuser support, access control, and reliability. They have demonstrated extremely successfully for the first kind of applications, but they have strong limitations with respect to the others.

The object-oriented approach, which was becoming powerful in the programming language and software engineering areas in the early 1980s, seemed a natural candidate, because it provides the required flexibility not being constrained by the data types and query languages available in traditional database systems and specifies both the structures of complex objects and the operations to manipulate these structures. The basic principle of the object-oriented approach in programming is indeed to consider the program consisting of independent objects, grouped in classes, communicating among each other through messages. Classes have an interface, which specifies the operations that can be invoked on objects belonging to the class, and an implementation, which specifies the code implementing the operations in the class interface. The encapsulation of class implementation allows for hiding data representation and operation implementation. Inheritance allows a class to be defined starting from the definitions of existing classes, called superclasses. An object can use operations defined in its base class as well as in its superclasses. Inheritance is thus a powerful mechanism for code reuse. Polymorphism (overloading) allows for defining operations with the same name for different object types; together with overriding and late binding, this functionality allows an operation to behave differently on objects of different classes. The great popularity of the object-oriented approach in software development is mainly caused by the increased productivity: The development time is reduced because of specification and implementation reuse; the maintenance cost is reduced as well because of the locality of modifications. Another advantage of object orientation is represented by the uniqueness of the paradigm: All phases of the software lifecycle (analysis, design, programming, etc.) rely on the same model, and thus, the transition from one phase to another is smooth and natural. Moreover, the object-oriented paradigm represents a fundamental shift with respect to how the software is produced: The software is no longer organized according to the computer execution model (in a procedural way); rather, it is organized according to the human way of thinking. Objects encapsulate operations together with the data these operations modify, which thus provides a *data-oriented* approach to program development. Finally, the object-oriented paradigm, because of encapsulation, is well suited for heterogeneous system integration, which is required in many applications.

In an object-oriented programming language, objects exist only during program execution. In a database, by contrast, objects can be created that *persist* and can be shared by several programs. Thus, *object databases* must store persistent objects in secondary memory and must support object sharing among different applications. This process requires the integration of the object-oriented paradigm with typical DBMS mechanisms, such as indexing mechanisms, concurrency control, and transaction management mechanisms. The efforts toward this integration led to the definition and development of object-oriented database management systems (OODBMSs) (1–4) and, later on, of object relational database management systems (5–7). Object-oriented databases are based on an object data model, which is completely different from the traditional relational model of data, whereas object relational databases rely on extensions of the relational data

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

model with the most distinguishing features of the object paradigm. Thanks to their "evolutive" nature with respect to relational DBMSs, object relational systems are succeeding in the marketplace and the data model of the SQL standard has been an object relational data model since SQL: 1999 (7).

OODBMSs have been proposed as an alternative technology to relational DBMSs. They mainly result from the introduction of typical DBMS functionality in an object-oriented programming environment (8). OODBMSs allow for directly representing complex objects and efficiently supporting navigational applications. Because the underlying data model is an object model, they allow you to *"store your data in the same format you want to use your data"* (9), and thus, they overcome the *impedance mismatch* between the programming language and the DBMS (10). Research in the area of object-oriented databases has been characterized by a strong experimental work and the development of several prototype systems, whereas only later theoretical foundations have been investigated and standards have been developed. The ODMG (Object Data Management Group) standard for OODBMSs was indeed first proposed in 1993, several years later than the first OODBMSs were on the market. Moreover, because OODBMSs mainly originated from object-oriented programming languages, systems originating from different languages rely on different data models. Even nowadays, despite the advent of Java, that *de facto* standardized the object model in programming, sensible differences persit among different systems, even those participating in the ODMG consortium. This lack of an initial common reference model and the low degree of standardization, together with the limited support for powerful declarative, high-level query languages, and functionalities not comparable with those of relational DBMSs for what concerned data security, concurrency control, and recovery, are among the main reasons because OODBMSs could not impose themselves in the marketplace.

Object relational DBMSs, by contrast, which were born as an extension of the relational technology with the characteristics of the object model, needed to support a wider range of applications. They are motivated by the need to provide relational DBMS functionalities in "traditional" data handling while extending the data model so that complex data can be handled. The choice is to introduce the distinguishing features of the object paradigm in the relational model without changing the reference model. As relational DBMSs, indeed, object relational DBMSs handle rows of tables, provide a declarative query language (SQL), and inherit the strength of relational systems in terms of efficiency and reliability in data management.

In this article, we discuss the application of the object-oriented paradigm to the database context, with a focus on data model and query language aspects. We first introduce the main notions of object-oriented data models and query languages, and then present the ODMG standard. Existing OODBMSs are then surveyed briefly, and object-relational databases are introduced. We conclude by mentioning some issues in object databases that are not dealt with in the article.

OBJECT-ORIENTED DATA MODELS

In this section, we introduce the main concepts of object-oriented data models, namely, objects, classes, and inheritance. In the next section, we will present the data model of the ODMG standard.

Objects

An object-oriented database is a collection of objects. In object-oriented systems, each real-world entity is represented by an object. Each object has an identity, a state, and a behavior. The identity is different from the identity of any other object and is immutable during the object lifetime; the state consists of the values of the object attributes; the behavior is specified by the methods that act on the object state, which is invoked by the corresponding operations.

Many OODBMs actually do not require each entity to be represented as an object; rather, they distinguish between objects and values. The differences between values and objects are as follows (11)

- Values are universally known abstractions, and they have the same meaning for each user; objects, by contrast, correspond to abstractions whose meaning is specified in the context of the application.
- Values are *built-in* in the system and do not need to be defined; objects, by contrast, must be introduced in the system through a definition.
- The information represented by a value is the value itself, whereas the meaningful information represented by an object is given by the relationships it has with other objects and values; values are therefore used to describe other entities, whereas objects are the entities being described.

Thus, values are elements of built-in domains, whereas objects are elements of uninterpreted domains. Typical examples of values are integers, reals, and strings. Each object is assigned an immutable identifier, whereas a value has no identifier; rather it is identified by itself.

Object Identity. Each object is identified uniquely by an object identifier (OID), which provides it with an identity independent from its value. The OID is unique within the system, and it is immutable; that is, it does not depend on the state of the object. Object identifiers are usually not visible and accessible directly by the database users; rather they are used internally by the system to identify objects and to support object references through object attribute values. Objects can thus be interconnected and can share components. The semantics of *object sharing* is illustrated in Fig. 1. The figure shows two objects that, in case (b), share a component, whereas in case (a), they do not share any object and simply have the same value for the attribute `date`. Although in case (a) a change in the publication date of `Article[i]` from `March 1997` to `April 1997` does not affect the publication date of `Article[j]`, in case (b), the change is also reflected on `Article[j]`.

The notion of object identifier is different from the notion of *key* used in the relational model to identify uniquely each

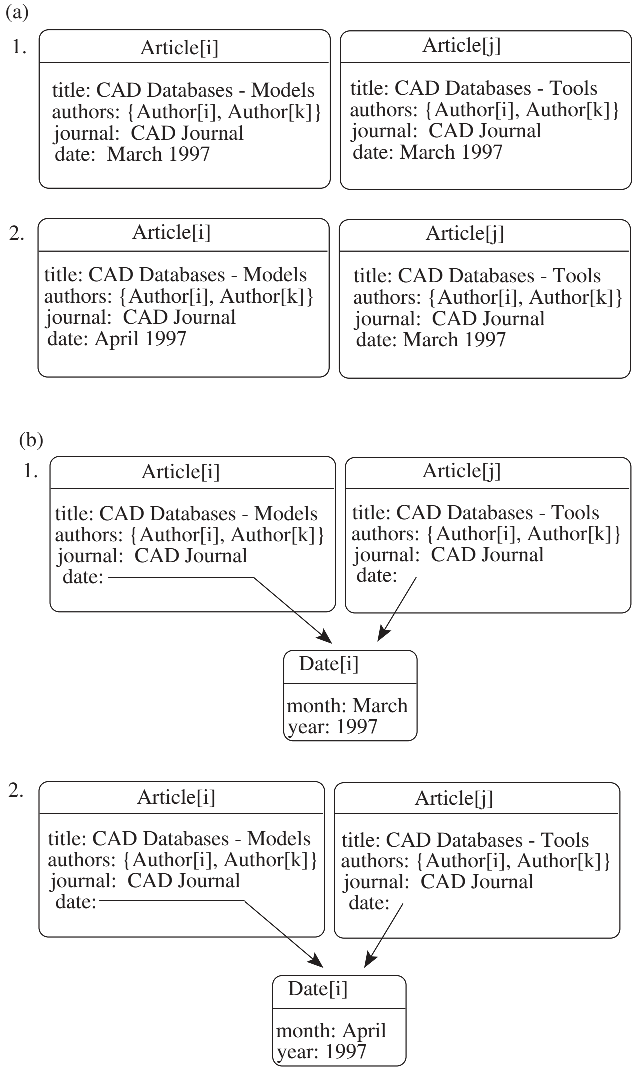

(a)
1.
Article[i]
title: CAD Databases - Models
authors: {Author[i], Author[k]}
journal: CAD Journal
date: March 1997
Article[j]
title: CAD Databases - Tools
authors: {Author[i], Author[k]}
journal: CAD Journal
date: March 1997
2.
Article[i]
title: CAD Databases - Models
authors: {Author[i], Author[k]}
journal: CAD Journal
date: April 1997
Article[j]
title: CAD Databases - Tools
authors: {Author[i], Author[k]}
journal: CAD Journal
date: March 1997
(b)
1.
Article[i]
title: CAD Databases - Models
authors: {Author[i], Author[k]}
journal: CAD Journal
date:
Article[j]
title: CAD Databases - Tools
authors: {Author[i], Author[k]}
journal: CAD Journal
date:
Date[i]
month: March
year: 1997
2.
Article[i]
title: CAD Databases - Models
authors: {Author[i], Author[k]}
journal: CAD Journal
date:
Article[j]
title: CAD Databases - Tools
authors: {Author[i], Author[k]}
journal: CAD Journal
date:
Date[i]
month: April
year: 1997

Figure 1. Object-sharing semantics.

tuple in a relation. A key is defined as the value of one or more attributes, and it can be modified, whereas an OID is independent from the value of an object state. Specifically, two different objects have different OIDs even when all their attributes have the same values. Moreover, a key is unique with respect to a relation, whereas an OID is unique within the entire database. The use of OIDs, as an identification mechanism, has several advantages with respect to the use of keys. First, because OIDs are implemented by the system, the application programmer does not have to select the appropriate keys for the various sets of objects. Moreover, because OIDs are implemented at a low level by the system, better performance is achieved. A disadvantage in the use of OIDs with respect to keys could be the fact that no semantic meaning is associated with them. Note, however, that very often in relational systems, for efficiency reasons, users adopt semantically meaningless codes as keys, especially when foreign keys need to be used.

The notion of object identity introduces at least two different notions of object equality:

- **Equality by identity:** Two objects are *identical* if they are the same object, that is, if they have the same identifier.
- **Equality by value:** Two objects are *equal* if the values for their attributes are *equal* recursively.

Obviously, two identical objects are also equal, whereas the converse does not hold. Some object-oriented data models also provide a third kind of equality, which is known as *shallow value equality* by which two objects are equal, although not being identical, if they share all attributes.

Object State. In an object-oriented database, the value associated with an object (that is, its state) is a complex value that can be built starting from other objects and values, using some type constructors. Complex (or structured) values are obtained by applying those constructors to simpler objects and values. Examples of primitive values are integers, characters, strings, booleans, and reals. The minimal set of constructors that a system should provide include sets, lists, and tuples. In particular, sets are crucial because they are a natural way to represent real-world collections and multivalued attributes; the tuple constructor is important because it provides a natural way to represent the properties of an entity; lists and arrays are similar to sets, but they impose an order on the elements of the collection and are needed in many scientific applications. Those constructors can be nested arbitrarily. A complex value can contain as components (references to) objects.

Many OODBMSs, moreover, support storage and retrieval of nonstructured values of large size, such as character strings or bit strings. Those values are passed as they are, that is, without being interpreted, to the application program for the interpretation. Those values, which are known as **BLOBs** *(binary large objects)*, are big-sized values like image *bitmaps* or long text strings. Those values are not structured in that the DBMS does not know their structure; rather the application using them knows how to interpret them. For example, the application may contain some functions to display an image or to search for some keywords in a text.

Object Behavior. Objects in an object-oriented database are manipulated through methods. A method definition consists of two components: a *signature* and an *implementation*. The *signature* specifies the method name, the names and types of method arguments, and the type of result, for methods returning a result value. Thus, the signature is a specification of the operation implemented by the method. Some OODBMSs do not require the specification of argument types; however, this specification is required in systems performing static-type checking. The method implementation consists of a set of instructions expressed in a programming language. Various OODBMSs exploited different languages. For instance, ORION exploited Lisp; GemStone a Smalltalk extension, namely OPAL; and O_2 a C extension, namely CO_2; other systems, among which are ObjectStore, POET, and Ode, exploited C++ or Java.

The use of a *general-purpose*, computationally complete programming language to code methods allows the whole application to be expressed in terms of objects. Thus no need exists, which is typical of relational DBMSs, to embed the query language (e.g., SQL) in a programming language.

Encapsulation. In a relational DBMS, queries and application programs that act on relations are expressed in an imperative language incorporating statements of the data manipulation language (DML), and they are stored in a traditional file system rather than in the database. In such an approach, therefore, a sharp distinction is made between programs and data and between query language and programming language. In an object-oriented database, data and operations manipulating them are *encapsulated* in a single structure: the object. Data and operations are thus designed together, and they are both stored in the same system.

This notion of encapsulation in programming languages derives from the concept of abstract data type. In this view, an object consists of an interface and an implementation. The interface is the specification of the operations that can be executed on the object, and it is the only part of the object that can be observed from outside. Implementation, by contrast, contains data, that is, the representation or state of the object, and methods specifying the implementation of each operation. This principle, in the database context, is reflected in that an object contains both programs and data, with a variation: In the database context, it is not clear whether the structure that defines the type of object is part of the interface. In the programming language context, the data structure is part of the implementation and, thus, not visible. For example, in a programming language, the data `type list` should be independent from the fact that lists are implemented as arrays or as dynamic structures; thus, this information is hidden correctly. By contrast, in the database context, the knowledge of an object attributes, and references made through them to other objects, is often useful.

Some OODBMSs, like ORION, allow us to read and write the object attribute values, which thus violates encapsulation. The reason is to simplify the development of applications that simply access and modify object attributes. Obviously, those applications are very common in the database context. Strict encapsulation would require writing many trivial methods. Other systems, like O_2, allow for specifying which methods and attributes are visible in the object interface and thus can be invoked from outside the object. Those attributes and methods are called *public*, whereas those that cannot be observed from outside the object are called *private*. Finally, some other systems, including GemStone, force strict encapsulation.

Classes

Instantiation is the mechanism that offers the possibility of exploiting the same definition to generate objects with the same structure and behavior. Object-oriented languages provide the notion of class as a basis for instantiation. In this respect, a class acts as a *template*, by specifying a structure, that is, the set of instance attributes, which is a set of methods that define the instance interface (method signatures) and implement the instance behavior (method implementations). Given a class, the *new* operation generates objects answering to all messages defined for the class. Obviously, the attribute values must be stored separately for each object; however, no need exists to replicate method definitions, which are associated with the class.

However, some class features cannot be observed as attributes of its instances, such as the number of class instances present in each moment in the database or the average value of an attribute. An example of an operation that is invoked on classes rather than on objects is the *new* operation for creating new instances. Some object-oriented data models, like those of GemStone and ORION, allow the definition of attributes and methods that characterize the class as an object, which are, thus, not inherited by the class instances.

Aggregation Hierarchy and Relationships. In almost all object-oriented data models, each attribute has a *domain*, that specifies the class of possible objects that can be assigned as values to the attribute. If an attribute of a class C has a class C' as domain, each C instance takes as value for the attribute an instance of C' or of a subclass of its. Moreover, an **aggregation** relationship is established between the two classes. An aggregation relationship between the class C and the class C' specifies that C is defined in terms of C'. Because C' can be in turn defined in terms of other classes, the set of classes in the schema is organized into an *aggregation hierarchy*. Actually, it is not a hierarchy in a strict sense, because class definitions can be recursive.

An important concept that exists in many semantic models and in models for the conceptual design of databases (12) is the *relationship*. A relationship is a link between entities in applications. A relationship between a person and his employer (*) is one example; another (classic) example is the relationship among a product, a customer, and a supplier (**), which indicates that a given product is supplied to a given customer by a given supplier. Associations are characterized by a *degree*, which indicates the number of entities that participate in the relationship, and by some *cardinality constraints*, which indicate the minimum and maximum number of relationships in which an entity can participate. For example, relationship (*) has degree 2; that is, it is binary, and its cardinality constraints are (0,1) for person and (1,n) for employer. This example reflects the fact that a person can have at most one employer, whereas an employer can have more than one employee. Referring to a maximum cardinality constraint, relationships are partitioned in one-to-one, one-to-many, and many-to-many relationships. Finally, relationships can have their own attributes; for example, relationship (**) can have attributes `quantity` and `unit price`, which indicate, respectively, the quantity of the product supplied and the unit price quoted. In most object-oriented data models, relationships are represented through object references. This approach, however, imposes a directionality on the relationship. Some models, by contrast, allow the specification of binary relationships, without, however, proper attributes.

Extent and Persistence Mechanisms. Besides being a *template* for defining objects, in some systems, the class also denotes the collection of its instances; that is, the class has also the notion of **extent**. The extent of a class is the collection of all instances generated from this class. This aspect is important because the class is the basis on which queries are formulated: Queries are meaningful only when they are applied to object collections. In systems in which classes do not have the extensional function, the extent of each class must be maintained by the applications through the use of constructors such as the set constructor. Different sets can contain instances of the same class. Queries are thus formulated against such sets and not against classes. The automatic association of an extent with each class (like in the ORION system) has the advantage of simplifying the management of classes and their instances. By contrast, systems (like O_2 and GemStone) in which classes define only specification and implementation of objects and queries are issued against *collections* managed by the applications, which provide a greater flexibility at the price of an increased complexity in managing class extents.

An important issue concerns the *persistence* of class instances, that is, the modalities by which objects are made persistent (that is, inserted in the database) and are deleted eventually (that is, removed from the database). In relational databases, explicit statements (like INSERT and DELETE in SQL) are provided to insert and delete data from the database. In object-oriented databases, two different approaches can be adopted with respect to object persistence:

- Persistence is an implicit property of all class instances; the creation (through the *new* operation) of an instance also has the effect of inserting the instance in the database; thus, the creation of an instance automatically implies its persistence. This approach usually is adopted in systems in which classes also have an extensional function. Some systems provide two different *new* operations: one for creating persistent objects of a class and the other one for creating temporary objects of that class.
- Persistence is an orthogonal properties of objects; the creation of an instance does not have the effect of inserting the instance in the database. Rather, if an instance has to survive the program that created it, it must be made persistent, for example, by assigning it a name or by inserting it into a persistent collection of objects. This approach usually is adopted in systems in which classes do not have the extensional function.

With respect to object *deletion*, two different approaches are possible:

- The system provides an explicit delete operation. The possibility of explicitly deleting objects poses the problem of referential integrity; if an object is deleted and other objects refer to it, references are not any longer valid (such references are called as *dangling references).* The explicit deletion approach was adopted by the ORION and Iris systems.
- The system does not provide an explicit delete operation. A persistent object is deleted only if all references to it have been removed (a periodic *garbage collection* is performed). This approach, which was adopted by the GemStone and O_2 systems, ensures referential integrity.

Migration. Because objects represent real-world entities, they must be able to reflect the evolution in time of those entities. A typical example is that of a person who is first of all a student, then an employee, and then a retired employee. This situation can be modeled only if an object can become an instance of a class different from the one from which it has been created. This evolution, known as object *migration*, allows an object to modify its features, that is, attributes and operations, by retaining its identity. Object migration among classes introduces, however, semantic integrity problems. If the value for an attribute A of an object O is another object O', an instance of the class domain of A, and O' changes class, if the new class of O' is no more compatible with the class domain of A, the migration of O' will result in O containing an illegal value for A. For this reason, migration currently is not supported in most existing systems.

Inheritance

Inheritance allows a class, called a *subclass*, to be defined starting from the definition of another class, called a *superclass*. The subclass inherits attributes and methods of its superclass; a subclass may in addition have some specific, noninherited features. Inheritance is a powerful reuse mechanism. By using such a mechanism, when defining two classes their common properties, if any, can be identified and factorized in a common superclass. The definitions of the two classes will, by contrast, specify only the distinguishing specific properties of these classes. This approach not only reduces the amount of code to be written, but it also has the advantage of giving a more precise, concise, and rich description of the world being represented.

Some systems allow a class to have several direct superclasses; in this case, we talk of *multiple inheritance*. Other systems impose the restriction to a single superclass; in this case, we talk of *single inheritance*. The possibility of defining a class starting from several superclasses simplifies the task of the class definition. However, conflicts may develop. Such conflicts may be solved by imposing an ordering on superclasses or through an explicit qualification mechanism.

In different computer science areas and in various object-oriented languages, different inheritance notions exist. In the knowledge representation context, for instance, inheritance has a different meaning from the one it has in object-oriented programming languages. In the former context, a subclass defines a *specialization* with respect to features and behaviors of the superclass, whereas in the latter, the emphasis is on attribute and method reuse. Different inheritance hierarchies can then be distinguished: *subtype hierarchy*, which focuses on consistency among type specifications; *classification hierarchy*, which expresses inclusion relationships among object collections; and *implementation hierarchy*, which allows code sharing among classes. Each hierarchy refers to different properties of the type/class system; those hierarchies, however, generally are merged into a single inheritance mechanism.

Overriding, Overloading, and Late Binding. The notion of *overloading* is related to the notion of inheritance. In many cases, it is very useful to adopt the same name for different operations, and this possibility is extremely useful in the object-oriented context. Consider as an example (9) a **display** operation receiving as input an object and displaying it. Depending on the object type, different display mechanisms are exploited: If the object is a figure, it should appear on the screen; if the object is a person, its data should be printed in some way; if the object is a graph, a graphical representation of it should be produced. In an application developed in a conventional system, three different operations `display_graph`, `display_person`, and `display_figure` would be defined. This process requires the programmer to be aware of all possible object types and all associated display operations and to use them properly.

In an object-oriented system, by contrast, the **display** operation can be defined in a more general class in the class hierarchy. Thus, the operation has a single name and can be used indifferently on various objects. The operation implementation is *redefined* for each class; this redefinition is known as *overriding*. As a result, a single name denotes different programs, and the system takes care of selecting the appropriate one at each time during execution. The resulting code is simpler and easier to maintain, because the introduction of a new class does not require modification of the applications. At any moment, objects of other classes, for example, information on some products, can be added to the application and can be displayed by simply defining a class, for example, `product`, to provide a proper (re)definition of the `display` operation. The application code would not require any modification.

To support this functionality, however, the system can no longer bind operation names to corresponding code at compile time; rather, it must perform such binding at run time: This late translation is known as *late binding*. Thus, the notion of *overriding* refers to the possibility for a class of redefining attributes and methods it inherits from its superclasses; thus, the inheritance mechanism allows for specializing a class through additions and substitutions. Overriding implies *overloading*, because an operation shared along an inheritance hierarchy can have different implementations in the classes belonging to this hierarchy; therefore, the same operation name denotes different implementations.

An Example

Figure 2 illustrates an example of object-oriented database schema. In the figure, each node represents a class. Each node contains names and domains of the attributes of the class it represents. For the sake of simplicity, we have included in the figure neither operations nor class features. Moreover, only attributes and no relationships have been included in classes. Nodes are connected by two different kinds of arcs. The node representing a class C can be linked to the node representing class C' through:

1. A thin arc, which denotes that C' is the domain of an attribute A of C (aggregation hierarchy).
2. A bold arc, which denotes that C' is superclass of C (inheritance hierarchy).

QUERY LANGUAGES

Query languages are an important functionality of any DBMS. A query language allows users to retrieve data by simply specifying some conditions on the content of those data. In relational DBMSs, query languages are the only

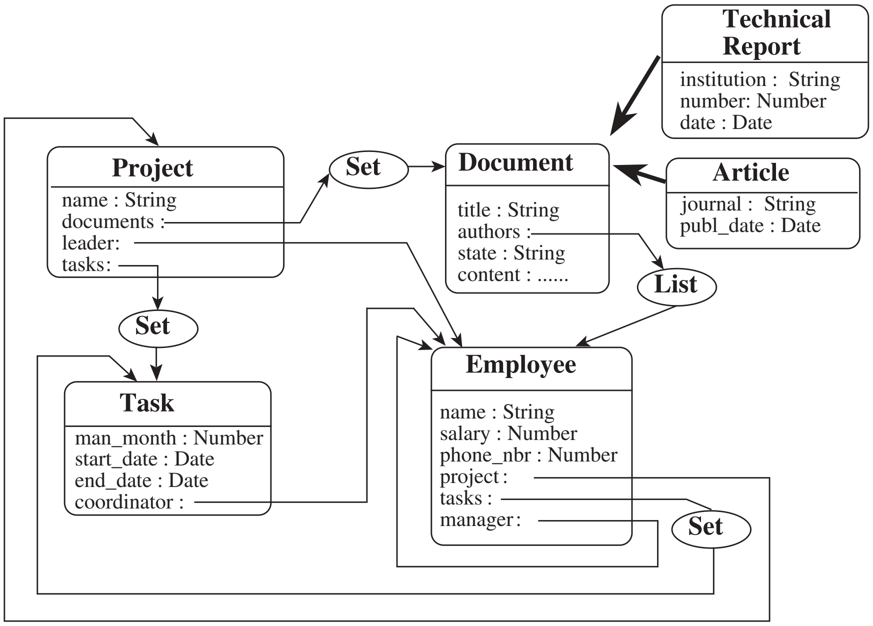


Figure 2. An example of object-oriented database schema.

way to access data, whereas OODBMSs usually provide two different modalities to access data. The first one is called *navigational* and is based on object identifiers and on the aggregation hierarchies into which objects are organized. Given a certain OID, the system can access directly and efficiently the object referred by it and can navigate through objects referred by the components of this object. The second access modality is called *associative*, and it is based on SQL-like query languages. These two different access modalities are used in a complementary way: A query is evaluated to select a set of objects that are then accessed and manipulated by applications through the navigational mechanism. Navigational access is crucial in many applications, such as graph traversal. Such type of access is inefficient in relational systems because it requires the execution of a large number of join operations. Associative access, by contrast, has the advantage of supporting the expression of declarative queries, which reduces thus application development time. Relational DBMSs are successful mostly because of their declarative query languages.

A first feature of object-oriented query languages is the possibility they offer of imposing conditions on nested attributes of an object aggregation hierarchy, through *path expressions*, which allows for expressing joins to retrieve the values of the attributes of an object components. In object-oriented query languages, therefore, two different kinds of join can be distinguished: *implicit join*, which derives from the hierarchical structure of objects, and *explicit join*, which, as in relational query languages, explicitly compares two objects. Other important aspects are related to inheritance hierarchies and methods. First, a query can be issued against a class or against a class and all its subclasses. Most existing languages support both of these possibilities. Methods can be used as *derived attributes* or as *predicate methods*. A method used as a derived attribute is similar to an attribute; however, whereas the attribute stores a value, the method computes a value starting from data values stored in the database. A predicate method is similar, but it returns the Boolean constants *true* or *false*. A predicate method evaluates some conditions on objects and can thus be part of the Boolean expressions that determine which objects satisfy the query.

Moreover, object-oriented query languages often provides constructs for expressing recursive queries, although recursion is not a peculiar feature of the object-oriented paradigm and it has been proposed already for the relational data model. It is, however, important that some kind of recursion can be expressed, because objects relevant for many applications are modeled naturally through recursion. The equality notion also influences query semantics. The adopted equality notion determines the semantics and the execution strategy of operations like union, difference, intersection, and duplicate elimination. Finally, note that *external names* that some object-oriented data models allow for associating with objects provides some semantically meaningful handlers that can be used in queries.

A relevant issue for object-oriented query languages is related to the language *closure*. One of the most remarkable characteristics of relational query languages is that the results of a query are in turn relations. Queries can then be composed; that is, the result of a query can be used as an operand in another query. Ensuring the closure property in an object-oriented query language is by contrast more difficult: The result of a query often is a set of objects, whose class does not exist in the database schema and that is defined by the query. The definition of a new class "on-the-fly" as a result of a query poses many difficulties, including where to position the new class in the inheritance hierarchy and which methods should be defined for such class. Moreover, the issue of generating OIDs for the new objects, which are the results of the query and instances of the new class, must be addressed.

To ensure the closure property, an approach is to impose restrictions on the projections that can be executed on classes. A common restriction is that either all the object attributes are returned by the query or only a single attribute is returned. Moreover, no explicit joins are allowed. In this way the result of a query is always a set of already existing objects, which are instances of an already existing class; the class can be a primitive class (such as the class of integers, string, and so forth) or a user-defined class. If one wants to support more general queries with arbitrary projections and explicit joins, a first approach to ensure closure is to consider the results of a query as instances of a general class, accepting all objects and whose methods only allow for printing or displaying objects. This solution, however, does not allow objects to be reused for other manipulations and, therefore, limits the nesting of queries, which is the main motivation for ensuring the closure property. Another possible approach is to consider the result of a query as a collection of objects, instances of a new class, which is generated by the execution of the query. The class implicitly defined by the query has no methods; however, methods for reading and writing attributes are supposed to be available, as system methods. The result of a query is thus similar to a set of tuples. An alternative solution (11) is, finally, that of including relations in the data model and of defining the result of a query as a relation.

THE ODMG STANDARD

ODMG is an OODBMS standard, which consists of a data model and a language, whose first version was proposed in 1993 by a consortium of major companies producing OODBMSs (covering about 90% of the market). This consortium included as voting members Object Design, Objectivity, O_2 Technology, and Versant Technology and as nonvoting members HP, Servio Logics, Itasca, and Texas Instruments. The ODMG standard consists of the following components:

- A data model (ODMG Object Model)
- A data definition language (ODL)
- A query language (OQL)
- Interfaces for the object-oriented programming languages C++ Java, and Smalltalk and data manipulation languages for those languages

The ODMG Java binding is the basis on which the *Java Data Objects* specification (13) has been developed, which provides the reference data model for persistent Java applications. In this section, we briefly introduce the main features of the ODMG 3.0 standard data model and of its query language OQL (14).

Data Definition in ODMG

ODMG supports both the notion of object and the notion of value (literal in the ODMG terminology). Literals can belong to atomic types like long, short, float, double, Boolean, char, and string; to types obtained through the set, bag, list, and array constructors; to enumeration types (enum); and to the structured types date, interval, time, and timestamp.

A schema in the ODMG data model consists of a set of object types related by inheritance relationships. The model provides two different constructs to define the external specification of an object type. An *interface* definition only defines the abstract behavior of an object type, whereas a *class* definition defines the abstract state and behavior of an object type. The main difference between class and interface types is that classes are types that are instantiable directly, whereas interfaces are types that cannot be instantiated directly. Moreover, an *extent* and one or more *keys* can be associated optionally with a class declaration. The extent of a type is the set of all instances of the class.

Objects have a state and a behavior. The object state consists of a certain number of properties, which can be either *attributes* or *relationships*. An attribute is related to a class, whereas a relationship is defined between two classes. The ODMG model only supports binary relationships, that is, relationships between two classes. One-to-one, one-to-many, and many-to-many relationships are supported. A relationship is defined implicitly through the specification of a pair of *traversal paths*, which enable applications to use the logical connection between objects participating in the relationship. Traversal paths are declared in pairs, one for each traversal direction of the binary relationship. The inverse clause of the traversal path definition specifies that two traversal paths refer to the same relationship. The DBMS is responsible for ensuring value consistency and referential integrity for relationships, which means that, for example, if an object participating in a relationship is deleted, any traversal path leading to it is also deleted.

Like several object models, the ODMG object model includes inheritance-based type–subtype relationships. More precisely, ODMG supports two inheritance relationships: the *ISA* relationship and the *EXTENDS* relationships. Subtyping through the ISA relationship pertains to the inheritance of behavior only; thus, interfaces may inherit from other interfaces and classes may also inherit from interfaces. Subtyping through the EXTENDS relationship pertains to the inheritance of both state and behavior; thus, this relationship relates only to classes. Multiple inheritance is allowed for the ISA, whereas it is not allowed for the EXTENDS relationship.

The ODMG class definition statement has the following format:

```
class ClassName[:SuperInterface List]
[EXTENDS SuperClass]
[(extent Extent Name[key[s] Attribute List])]
{ Attribute List
  Relationship List
  Method List }
```

In the above statement:

- The : clause specifies the interfaces by which the class inherits through *ISA*.
- The EXTENDS clause specifies the superclass by which the class inherits through *EXTENDS*.
- The extent clause specifies that the extent of the class must be handled by the OODBMS.
- The key [s] clause, which can appear only if the extent clause is present, specifies a list of attributes for which two different objects belonging to the extent cannot have the same values.
- Each attribute in the list is specified as

```
attribute Domain Name;
```

- Each relationship in the list is specified as

```
relationship Domain Name
  inverse Class:: Inverse Name
```

where *Domain* can be either *Class*, in the case of unary relationships, or a collection of *Class* elements, and *Inverse Name* is the name of the inverse traversal path.

- Each method in the list is specified as

```
Type Name(Parameter List) [ raises Exception
List]
```

where *Parameter List* is a list of parameters specified as

```
in | out | inout Parameter Name
```

and the raises clause allows for specifying the exceptions that the method execution can introduce.

The following ODL definition defines the following classes: Employee, Document, Article, Project, and Task of the database schema of Fig. 2, in which some relationships between projects and employees (rather than the leader attribute of class Project), and between employees and tasks (rather than the tasks attribute of class Employee), have been introduced. The main difference in representing a link between objects as a relationship rather than as a reference (that is, attribute value) is in the nondirectionality of the relationship. If, however, only one direction of the link is interesting, the link can be represented as an attribute.

```
class Employee (extent Employees key name)
{ attribute string name;
attribute unsigned short salary;
attribute unsigned short phone_nbr[ 4] ;
attribute Employee manager;
attribute Project project;
relationship Project leads
   inverse Project::leader;
```

```
  relationship Set<Task> tasks
     inverse Task:participants;
}

class Document (extent Documents key title)
{ attribute string title;
  attribute List<Employee> authors;
  attribute string state;
  attribute string content;
}

class Article EXTENDS Document (extent Articles)
{ attribute string journal;
  attribute date publ_date;
}

class Project (extent Projects key name)
{ attribute string name;
  attribute Set<Document> documents;
  attribute Set<Task> tasks;
  relationship Employee leader
    inverse Employee::leads;
}

class Task (extent Tasks)
{ attribute unsigned short man_month;
  attribute date start_date;
  attribute date end_date;
  attribute Employee coordinator;
  relationship Set<Employee> participants
    inverse Employee::tasks;
}
```

Data Manipulation in ODMG

ODMG does not support a single DML; rather, three different DMLs are provided, which are related to C++, Java, and Smalltalk, respectively. These OMLs are based on different persistence policies, which correspond to different object-handling approaches in the languages. For example, C++ OML supports an explicit delete operation (`delete_object`), whereas Java and Smalltalk OMLs do not support explicit delete operations; rather, they are based on a garbage collection mechanism.

ODMG, by contrast, supports an SQL-like query language (OQL), which is based on queries of the `select from where` form that has been influenced strongly by the O_2 query language (15). The query returning all tasks with a manpower greater than 20 months, whose coordinator earns more than $20,000, is expressed in OQL as follows:

```
select t
from Tasks t
where t.man_month > 20 and
  t.coordinator.salary > 20000
```

OQL is a functional language in which operators can be composed freely as a consequence of the fact that query results have a type that belongs to the ODMG type system. Thus, queries can be nested. As a stand-alone language, OQL allows for querying objects denotable through their names. A name can denote an object of any type (atomic, collection, structure, literal). The query result is an object whose type is inferred from the operators in the query expression. The result of the query "retrieve the starting date of tasks with a man power greater than 20 months," which is expressed in OQL as

```
select distinct t.start_date
from Tasks t
where t.man_month > 20
```

is a literal of type `Set < date >`.

The result of the query "retrieve the starting and ending dates of tasks with a man power greater than 20 months," which is expressed in OQL as

```
select distinct struct(sd: t.start_date,
  ed: t.end_date)
from Tasks t
where t.man_month > 20
```

is a literal of type `Set < struct(sd : date, ed : date) >`.

A query can return structured objects having objects as components, as it can combine attributes of different objects. Consider as an example the following queries. The query "retrieve the starting date and the coordinator of tasks with a man power greater than 20 months," which is expressed in OQL as

```
select distinct struct(st: t.start_date,
  c: coordinator)
from Tasks t
where t.man_month > 20
```

produces as a result a literal with type `Set < struct(st : date, c : Employee) >`. The query "retrieve the starting date, the names of the coordinator and of participants of tasks with a man power greater than 20 months," which is expressed in OQL as

```
select distinct struct(sd: t.start_date,
  cn: coordinator.name,
  pn: (select p.name
    from t.participants p))
from Tasks t
where t.man_month > 20
```

produces as a result a literal with type `Set < struct(st : date, cn : string, pn : bag < string >) >`.

OQL is a very rich query language. In particular it allows for expressing, in addition to path expressions and projections on arbitrary sets of attributes, which are illustrated by the above examples, explicit joins and queries containing method invocations. The query "retrieve the technical reports having the same title of an article", is expressed in OQL as

```
select tr
from Technical_Reports tr, Articles a
where tr.title = a title
```

The query “retrieve the name and the bonus of employees having a salary greater than 20000 and a bonus greater than 5000” is expressed in OQL as

```
select distinct struct(n: e.name, b: e.bonus)
from Employees e
where e.salary > 20000 and e.bonus > 5000
```

OQL finally supports the aggregate functions `min`, `max`, `count`, `sum`, and `avg`. As an example, the query “retrieve the maximum salary of coordinators of tasks of the CAD project” can be expressed in OQL as

```
select max(select t.coordinator.salary
  from p.tasks t)
from Projects p
where p.name = ' CAD'
```

OBJECT-ORIENTED DBMSs

As we have discussed, the area of object-oriented databases has been characterized by the development of several systems in the early stages, followed only later by the development of a standard. Table 1 compares some of the most influential systems along several of dimensions. In the comparison, we distinguish systems in which classes have an extensional function, that is, in which with a class the set of its instances is associated automatically, from those in which object collections are defined and handled by the application. We point out, moreover, the adopted persistence mechanism, which distinguishes among systems in which all objects are created automatically as persistent, systems in which persistence is ensured by linking an object to a persistence root (usually an external name), and systems supporting two different creation operations, one for creating temporary objects and the other one for creating persistent objects. The different policies with respect to encapsulation are also shown, which distinguishes among systems forcing strict encapsulation, systems supporting direct accesses to attribute values, and systems distinguishing between private and public features. Finally, the O_2 system allows the specification of exceptional instances, that is, of objects that can have additional features and/or redefine (under certain compatibility restrictions) features of the class of which they are instances.

Most OODBMSs in Table 1, although they deeply influenced the existing OODBMSs and the ODMG standard, are no more available in the marketplace. Table 2 lists most popular commercial and open-source OODBMSs available in 2007 (www.odbms.org). Although most of their producers are ODMG members, the system still exhibits different levels of ODMG compliance.

OBJECT RELATIONAL DATABASES

As discussed at the beginning of this article, object relational databases rely on extensions of the relational data model with the most distinguishing features of the object paradigm. One of the first object relational DBMS is UniSQL (24), and nowadays object relational systems include, among others, DB2 (25,26), Oracle (27), Microsoft SQL Server (28), Illustra/Informix (29), and Sybase (30). All of these systems extend a relational DBMS with object-oriented modeling features. In all those DBMSs, the type system has been extended in some way and the possibility has been introduced of defining methods to model user-defined operations on types. The SQL standard, since its SQL:1999 version (7), has been based on an object-relational data model. In what follows, we discuss briefly the most relevant type system extensions according to the most recent version of the SQL standard, namely, SQL:2003 (31).

Table 1. Comparison among data models of most influential OODBMSs

	Gem-Stone	Iris	O2	Orion	Object-Store	Ode	ODMG
Reference	(16)	(17)	(18), (19)	(20), (21)	(22)	(23)	(14)
Class extent	NO	YES	NO	YES	NO	YES	YES[a]
Persistence	R	A	R	A	R	2op	A[b]
Explicit deletion	NO	YES	NO	YES	YES	YES	YES[b]
Direct access to attributes	NO	YES	P	YES	P	P	YES
Domain specification for attributes	O	M	M	M	M	M	M
Class attributes and methods	YES	NO	NO	YES	NO	NO	NO
Relationships	NO	YES	NO	NO	YES	NO	YES
Composite objects	NO	NO	NO	YES	NO	NO	NO
Referential integrity	YES	NO	YES	NO	YES[c]	NO	YES[c]
Multiple inheritance	NO	YES	YES	YES	YES	YES	YES
Migration	L	YES	NO	NO	NO	NO	NO
Exceptional instances	NO	NO	YES	NO	NO	NO	NO

R = root persistence, A = automatic, 2op = two different *new* operations.
P = only for public attributes.
O = optional, M = mandatory.
L = in limited form.
[a] For those classes in which definition of an extent clause is specified.
[b] In C++ OML, created objects are automatically persistent and explicit deletion is supported; in Smalltalk OML, persistence is by root and no explicit delete operation exists.
[c] Referential integrity is ensured for relationships but not for attributes.

Table 2. OODBMS scenario in 2007 www.odbms.org

Commercial systems		
Company	System(s)	Web reference
db4objects	db4o	www.db4o.com
Objectivity	Objectivity/DB	www.objectivity.com
Progress	ObjectStore, PSE Pro	www.progress.com
Versant	Versant Object Database, FastObjects	www.versant.com
InterSystems	Cache	www.intersystems.com
GemStone	GemStone/S, Facets	www.facetsodb.com
Matisse	Matisse	www.matisse.com
ObjectDB	ObjectDB	www.objectdb.com
W3apps	Jeevan	www.w3apps.com

Open source systems	
System	Web reference
db4o	www.db4o.com
EyeDB	www.eyedb.com
Ozone	www.ozone-db.com
Perst	www.mcobject.com/perst/
Zope (ZODB)	www.zope.org

Primitive Type Extensions

Most DBMSs support predefined types like integers, floating points, strings, and dates. Object relational DBMSs support the definition of new primitive types starting from predefined primitive types and the definition of user-defined operations for these new primitive types. Operations on predefined types are inherited by the user-defined type, unless they are redefined explicitly. Consider as an example a `yen` type, which corresponds to the Japanese currency. In a relational DBMS, this type is represented as a numeric type with a certain scale and precision, for example `DECIMAL(8,2)`. The predefined operations of the DECIMAL type can be used on values of this type, but no other operations are available. Thus, any additional semantics, for instance, as to convert yens to dollars, must be handled by the application, as the display in an appropriate format of values of that type. In an object relational DBMS, by contrast, a type `yen` can be defined as follows:

```
CREATE TYPE yen AS Decimal(8,2);
```

and the proper functions can be associated with it.

Complex Types

A complex, or structured, type includes one or more attributes. This notion corresponds to the notion of struct of the C language or to the notion of record of the Pascal language. Complex types are called *structured types* in SQL:2003 (31). As an example, consider the type `t_Address`, defined as follows:

```
CREATE TYPE t_Address AS (street VARCHAR(50),
                          number INTEGER,
                          city CHAR(20),
                          country CHAR(2),
                          zip INTEGER),
```

Relations can contain attributes whose type is a complex type, as shown by the following example:

```
CREATE TABLE Employees(name CHAR(20),
                       emp# INTEGER,
                       curriculum CLOB,
                        salary INTEGER,
                        address t_Address);
```

This relation can be defined as equivalently :

```
CREATE TYPE t_Employee AS (name CHAR(20),
                      emp# INTEGER,

             curriculum CLOB,
             salary INTEGER,
             address t_Address);
            CREATE TABLE Employees OF t_Employee;
```

Note that for what concerns the structure of the tuples in the relation, the above declarations are equivalent to the following one, which makes use of an anonymous *row type* for specifying the structure of addresses:

```
CREATE TABLE Employees (name CHAR(20),
                             emp# INTEGER,
                             curriculum CLOB,
                             salary INTEGER,
        address ROW (street VARCHAR(50),
                        number INTEGER,
                        city CHAR(20),
                        country CHAR(2),
                        zip INTEGER));
```

Components of attributes, whose domain is a complex type, are accessed by means of the nested *dot notation*. For example, the zipcode of the address of an employee is accessed as `Employees.address.zip`.

Methods. Methods can be defined on simple and complex types, as part of the type definition. Each method has a signature and an implementation that can be specified in SQL/PSM (31) or in different programming languages. The method body can refer to the instance on which the method is invoked through the `SELF` keyword.

The definition of the type `t_Employee` can, for example, be extended with the definition of some methods as follows:

```
CREATE TYPE t_Employee (...)
INSTANCE METHOD double_salary()
   RETURNS BOOLEAN;
INSTANCE METHOD yearly_salary()
   RETURNS INTEGER;
INSTANCE METHOD add_telephone(n CHAR(15))
   RETURNS BOOLEAN;
```

With each complex type, a *constructor* method, which is denoted by `NEW` and the name of the type, is associated. This method creates an instance of the type, given its attribute values. As an example, the invocation

```
 NEW t_address('via pisa', 36, 'genova',
'italy', 16146)
```

returns a new instance of the `t_address` type.

For each attribute `A` in a complex type, an *accessor* method `A`, returning the attribute value, and a *mutator* method A, taking as input a value *v* and setting the attribute value to *v*, are associated implicitly with the type.

Referring to type `t_Employee` and to relation `Employees` above, the statement:

```
SELECT address.city()
FROM Employees;
```

contains an invocation of the accessor method for attribute `city`. Because the method has no parameters, the brackets can be omitted (i.e., `address.city`). By contrast, the statement:

```
UPDATE Employees
SET address = address.city('genova')
WHERE empl# = 777;
```

contains an invocation of the mutator method for attribute `city`.

Collection Types

Object relational DBMSs support constructors for grouping several instances of a given type, which thus models *collections* of type instances. Specifically, SQL:2003 supports `ARRAY` and `MULTISET` collections. Referring to the `Employees` relation above, suppose we want to add a `tel_nbrs` attribute as a collection of telephone numbers (strings). An attribute declared as

```
tel_nbrs CHAR(15) ARRAY[3]
```

allows for representing a maximum of three telephone numbers, ordered by importance. By contrast, an attribute declared as

```
tel_nbrs CHAR(15) MULTISET
```

allows for representing an unlimited number of telephone numbers, without specifying an order among them. Note that duplicates are allowed because the collection is a multiset.

Elements of the collections are denoted by indexes in case of arrays (for example, `tel_nbrs[2]` accesses the second number in the array), whereas multisets can be converted to relations through the `UNNEST` function and then they can be iterated over through an SQL query as any other relation. The following SQL statement:

```
SELECT e.name, T.N
FROM Employees e, UNNEST(e.tel_nbrs) T(N)
WHERE e.emp# = 777;
```

returns the employee name and the set of its telephone numbers. In the previous statement, `T` is the name of a virtual table and `N` is the name of its single column.

Reference Types

As we have seen, a structured type can be used in the definition of types and tables. Its instances are complex values. Structured types can be used as well for defining objects, as tuples with an associated identifier. These tuples are contained in *typed tables*, which are characterized by an additional identifier field, which is specified as `REF IS`. Note that the uniqueness of identifiers is not ensured across relations. The `Employees` relation above can be specified as a typed table as follows (provided the definition of type `t_Employee`:

```
CREATE TABLE Employees OF t_Employee
   (REF IS idE);
```

The values for the `idE` additional field are system-generated.

Reference types allow a column to refer a tuple contained in a typed table through its identifier. Thus, those types allow a tuple in a relation to refer to a tuple in another relation. The reference type for a type `T` is the type of the identifier of instances of `T` and is denoted by `REF(T)`. Given the following declaration and the above declarations of the `Employees` relation:

```
CREATE TYPE t_Department (name CHAR(10),
                dept# INTEGER,
                chair REF(t_Employee),
                dependents REF(t_Employee)
                MULTISET);
CREATE TABLE Departments
OF t_Department
(REF IS idD);
```

the values of the `chair` and `dependents` attributes are identifiers of instances of type `t_Employee`. This definition, however, does not provide information about the relation containing the instances of the type (SQL:2003 calls these *unconstrained* references). To ensure referential integrity, a `SCOPE` clause, which requires the reference to belong to the extent associated with a typed table, can be used. With the following declaration of the `Departments` relation:

```
CREATE TABLE Departments OF t_Department
(REF IS idD,
chair WITH OPTIONS SCOPE Employees,
dependents WITH OPTIONS SCOPE Employees);
```

for each tuple in the `Departments` relation, the `chair` column is guaranteed to refer to a tuple of the `Employees` relation (corresponding to the department chair) and the values in the multiset in the `dependent` column are guaranteed to refer to tuples of the `Employees` relation.

To manipulate reference-type instances, SQL provides the dereferencing function `DEREF`, returning the tuple referenced by a reference, and the reference function `->`, returning the value of a specific attribute of the tuple referenced by a reference. The attributes of a referred instance can be accessed by means of the dot notation. For example, referring to the example above, the name of a department chair can be denoted either as `Departments.chair->name` or `Departments.DEREF(chair).name`.

Inheritance

Inheritance specifies subtype/supertype relationships among types. Subtypes inherit attributes and methods of their supertypes. Object relational DBMSs allow for specifying inheritance links both among types and among relations. The following declarations specify types `t_ Student` and `t_Teacher` as subtypes of the `t_Person` type:

```
 CREATE TYPE t_Person AS (name      CHAR(20),
                          ssn       INTEGER,
                          b_date    DATE,
                          address   t_Address);
CREATE TYPE t_Teacher UNDER t_Person AS
                         (salary  DECIMAL(8,2),
                          dept     REF t_Department,
                         teaches  REF(t_Course) MULTISET);
CREATE TYPE t_Student UNDER t_Person AS
                       (avg_grade  FLOAT,
                       attends    REF(t_Course) MULTISET);
```

The following declarations, by contrast, specify inheritance relationships among relations:

```
 CREATE TABLE Persons OF t_Person;
 CREATE TABLE Teachers OF t_Teacher
 UNDER Persons;
 CREATE TABLE Students OF t_Student
 UNDER Persons;
```

At the data level, those two declarations imply that instances of Teachers and Students relations are also instances of the Persons relation (inheritance among relations) and that instances of those relations have `name`, `ssn`, `b_date`, and `address` as attributes (inheritance among types). The following query:

```
 SELECT name, address
 FROM Teachers
 WHERE salary > 2000
```

can thus be expressed.

Inheritance among types also implies method inheritance and method overloading. Overriding and late binding are supported.

LOBs

Object relational DBMSs, finally, provide LOB types to support the storage of multimedia objects, such as documents, images, and audio messages. LOBs are stored semantically as a column of the relation. Physically, however, they are stored outside the relations, typically in an external file. Usually, for efficiency reasons, those external files are not manipulated under transactional control (or, at least, logging is disabled). LOBs can be either CLOBs (characters) or BLOBs (binaries). Ad hoc indexing mechanisms are exploited to handle LOBs efficiently.

CONCLUDING REMARKS

In this article, we have focused on the modeling aspects and query and data manipulation languages of OODBMs and object relational DBMSs. The effective support of object-oriented data models and languages requires revisiting and possibly extending techniques and data structures used in DBMS architectures. In this section, we briefly discuss some of those architectural issues and point out relevant references. We mention moreover some relevant issues in OODBMSs not dealt with in the article.

An important aspect is related to the indexing techniques used to speed up query executions. Three object-oriented concepts have an impact on the evaluation of object-oriented queries as well as on the indexing support required: the inheritance hierarchy, the aggregation hierarchy, and the methods. For what concerns inheritance, a query on a class C has two possible interpretations. In a *single-class query*, objects are retrieved from only the queried class C itself, whereas in a *class-hierarchy query*, objects are retrieved from all the classes in the inheritance hierarchy rooted at C. To facilitate the evaluation of such types of queries, a *class-hierarchy index* needs to support efficient retrieval of objects from a single class as well as from all the classes in the class hierarchy. A class-hierarchy index is characterized by two parameters: the hierarchy of classes to be indexed and the index attribute of the indexed hierarchy.

Two approaches to class-hierarchy indexing exist: The *class-dimension-based approach*(32,33) partitions the data space primarily on the class of an object, and the *attribute-dimension-based approach*(32) partitions the data space primarily on the indexed attribute of an object. Although the class-dimension-based approach supports single-class queries efficiently, it is not effective for class-hierarchy queries because of the need to traverse multiple single-class indexes. On the other hand, the attribute-dimension-based approach generally provides efficient support for class-hierarchy queries on the root class (i.e., retrieving objects of all indexed classes), but it is inefficient for single-class queries or class-hierarchy queries on a subhierarchy of the indexed class hierarchy, as it may need to access many irrelevant leaf nodes of the single index structure. To support both types of queries efficiently, the index must support both ways of data partitioning (34). However, this is not a simple or direct application of multi-dimensional indexes, because totally ordering of classes is not possible and, hence, partitioning along the class dimension is problematic.

A second important issue in indexing techniques is related to aggregation hierarchies and to navigational accesses along these hierarchies. Navigational access is based on traversing object references; a typical example is represented by graph traversal. Navigations from one object in a class to objects in other classes in a class aggregation hierarchy are essentially expensive pointer-chasing operations. To support navigations efficiently, indexing structures that enable fast path instantiation have been developed, including the the multi-index technique, the nested index, the path index, and the join hierarchy index. In practice, many of these structures are based on precomputing traversals along aggregation hierarchies. The major problem of many of such indexing techniques is related to update operations that may require access to several objects to determine the index entries that need updating. To reduce update overhead and yet maintain the efficiency of path indexing structures, paths can be broken into subpaths that are then indexed separately (35,36). The proper splitting and allocation is highly dependent on the query and update patterns and frequencies. Therefore, adequate index allocation tools should be developed to support the optimal index allocation.

Finally, a last issue to discuss is related to the use of user-defined methods into queries. The execution of a query involving a method may require the execution of such method for a large number of instances. Because a method can be a general program, the query execution costs may become prohibitive. Possible solutions are based on method precomputation; such approaches, however, make object updates rather expensive. We refer the reader to Ref. 37, for an extensive discussion on indexing techniques for OODBMSs.

Another important issue, which is related to performance, is query optimization. Because most object-oriented queries only require implicit joins through aggregation hierarchies, the efficient support of such a join is important. Therefore, proposed query execution strategies have focused on efficient traversal of aggregation hierarchies. Because aggregation hierarchies can be represented as graphs, and a query can be observed as a visit of a portion of such a graph, traversal strategies can be formalized as strategies for visiting nodes in a graph. The main methods proposed for such visits include forward traversal, reverse traversal, and mixed traversal. They differ with respect to the order according to which the nodes involved in a given query are visited. A second dimension in query processing strategies concerns how instances from the visited class are retrieved. The two main strategies are the nested-loop and the sort-domain. Each of those strategies can be combined with each node traversal strategy, which results in a wide spectrum of strategies. We refer the reader to Ref. 1 for an extensive discussion on query execution strategies and related cost models.

Other relevant issues that we do not discuss here, but are dealt with in Ref. 1, include access control mechanisms, versioning models, schema evolutions, benchmarks, concurrency control, and transaction management mechanisms. Another aspect concerns integrity constraint and trigger support (38).

A final topic we would like to mention is related to the modeling and the management of *spatiotemporal* and *moving* objects. A large percentage of data managed in a variety of different application domains has spatiotemporal characteristics. For what concerns the spatial characteristics of data, for instance, an object may have a geographical location. Specifically, geographic objects, such as a land parcel, a car, and a person, do have a location. The location of a geographic object is a spatial attribute value, whose data type can be, for instance, a polygon or a point. Moreover, attribute values of geographic objects may be space dependent, which is different from spatial attribute values. For instance, the soil type of a land parcel applies to the entire spatial extent (i.e., the location of the land parcel) and it is not a "normal" attribute of a land parcel, but it is inherited from the underlying geographical space via the object location. This means that the attribute can be modeled as a function from the spatial domain. For what concerns the temporal characteristics of objects, both the time when some property holds *(valid time)* and the time when something is believed in/recorded as current in the database *(transaction time)* can be of interest. Several kinds of applications can be devised in which both spatial and temporal aspects of objects are important, among which at least three different types can be distinguished: *cadastral applications*, in which the spatial aspects are modeled primarily as regions and points and changes occur discretely across time; *transportation applications*, in which the spatial aspects are modeled primarily as linear features, graphs, and polylines and changes occur discretely across time; and *environmental applications*, or "location-based services," characterized by continuously changing spatial aspects. Several proposals that provide an integrated approach for the management of spatial and temporal information have been presented in the recent past (39,40). A growing interest has been devised also in the area of moving and geometric objects, mainly involving abstract modeling. Recently, spatiotemporal extensions of SQL:1999 and the ODMG model also have been proposed (41–43).

BIBLIOGRAPHY

1. E. Bertino and L. D. Martino, *Object-Oriented Database Systems - Concepts and Architecture*, Reading, MA: Addison-Wesley, 1993.
2. R. Cattel, *Object Data Management - Object-Oriented and Extended Relational Database Systems*, Reading, MA: Addison-Wesley, 1991.
3. A. Kemper and G. Moerkotte, *Object-Oriented Database Management: Applications in Engineering and Computer Science*, Englewood Coiffs, NJ: Prentice-Hall, 1994.
4. W. Kim and F. H. Lochovsky, *Object-Oriented Concepts, Databases, and Applications*, Reading, MA: Addison-Wesley, 1989.
5. M. Stonebraker and D. Moore, *Object-Relational DBMSs: The Next Great Wave*, San Francisco, CA: Morgan Kaufmann, 1996.
6. M. Carey, D. Chamberlin, S. Narayanan, B. Vance, D. Doole, S. Rielau, R. Swagerman, and N. Mattos, O-O, What's happening to DB2? *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, 1999*, pp. 511–512.
7. J. Melton and A.R. Simon, *SQL:1999 - Understanding Relational Language Components*, San Francisco, CA: Morgan-Kaufmann, 2001.
8. A.B. Chaudhri and R. Zicari, *Succeeding with Object Databases*, New York: John Wiley & Sons, 2001.
9. W. Cook, et al., *Objects and Databases: State of the Union in 2006*. Panel at OOPSLA 2006, New York: ACM Press, 2006.

10. M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, The object-oriented database system manifesto. in W. Kim, et al., (eds.), *Proc. First Int'l Conf. on Deductive and Object-Oriented Databases, 1989*, pp. 40–57.
11. C. Beeri, Formal models for object-oriented databases, in W. Kim, et al., (eds.), *Proc. First Int'l Conf. on Deductive and pp. Object-Oriented Databases, 1989*, pp. 370–395.
12. P. Chen, The entity-relationship model - towards a unified view of data, *ACM Trans. Database Sys.*, **1**(1): 9–36, 1976.
13. Sun Microsystems, Java Data Objects Version 1.0.1. 2003. Available: http://java.sun.com/products/jdo.
14. R. Cattel, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russel, O. Schadow, T. Stanienda, and F. Velez, *The Object Database Standard: ODMG 3.0*, San Francisco, CA: Morgan-Kaufmann, 1999.
15. S. Cluet, Designing OQL: allowing objects to be queried, *Informat. Sys.*, **23**(5): 279–305, 1998.
16. R. Breitl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, and M. Williams, The GemStone data management system, in W.F. Kim and F.H. Lochovsky (eds.), Ref. 4. pp. 283–308.
17. D. H. Fishman et al. Overview of the Iris DBMS. In Ref. 4, pp. 219-250.
18. F. Bancilhon, C. Delobel, and P. Kanellakis, *Building an Object-Oriented Database System: The Story of O_2*, San Francisco, CA: Morgan-Kaufmann, 1992.
19. O. Deux, The Story of O_2, *IEEE Trans. Knowl, Data Engineer.*, **2**(1): 91–108, 1990.
20. W. Kim, et al., Features of the ORION object-oriented database system. In Ref. 4, pages. 251-282.
21. W. Kim, *Introduction to Object-Oriented Databases*, Cambridge, MA: The MIT Press, 1990.
22. Object Design. ObjectStore Java API User Guide (ObjectStore 6.0). 1998. Available at http://www.odi.com.
23. R. Agrawal and N. Gehani, ODE (Object Database and Environment): the language and the data model, in *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, 1989*, pp. 36–45.
24. W. Kim, UniSQL/X unified relational and object-oriented database system, *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, 1994*, p. 481.
25. D. Chamberlin, *Using the New DB2 - IBM's Object-Relational Database System*, San Franciso, CA: Morgan-Kaufmann, 1996.
26. *IBM DB2 Universal Database SQL Reference, Volume 1.* IBM Corp., 2004.
27. *Oracle Database. Application Developer's Guide - Object-Relational Features.* Oracle Corp., 2005.
28. Microsoft Corporation. *Microsoft SQL Server*, Version 7.0, 1999.
29. Illustra Information Technologies, Oakland, California. *Illustra User's Guide.* Release 2.1.
30. SYBASE Inc., Berkley, California. *Transact-SQL User's Guide for Sybase.* Release 10.0.
31. A. Eisenberg, J. Melton, K. Kulkharni, J.E. Michels, and F. Zemke, SQL:2003 has been published. *SIGMOD Record*, **33**(1): 119–126, 2004.
32. W. Kim, K.C. Kim, and A. Dale, Indexing techniques for object-oriented databases, In [Ref. 4], pages. 371-394.
33. C. C. Low, B. C. Ooi, and H. Lu, H-trees: a dynamic associative search index for OODB. *Proc. 1992 ACM SIGMOD International Conference on Management of Data*, 1992, pp. 134–143.
34. C.Y. Chan, C.H. Goh and B. C. Ooi, Indexing OODB instances based on access proximity, *Proc. 13th International Conference on Data Engineering, 1997*, pp. 14–21.
35. E. Bertino, On indexing configuration in object-oriented databases, *VLDB J.*, **3**(3): 355–399, 1994.
36. Z. Xie and J. Han, Join index hierarchy for supporting efficient navigation in object-oriented databases, *Proc. 20th International Conference on Very Large Data Bases, 1994*, pp. 522–533.
37. E. Bertino, R. Sacks-Davis, B. C. Ooi, K. L. Tan, J. Zobel, B. Shidlovsky, and B. Catania, *Indexing Techniques for Advanced Database Systems*, Dordrecht: Kluwer, 1997.
38. E. Bertino, G. Guerrini, I. Merlo, Extending the ODMG object model with triggers, *IEEE Trans. Know. Data Engin.*, **16**(2): 170–188, 2004.
39. G. Langran, *Time in Geographic Information Systems*, Oxford: Taylor & Francis, 1992.
40. M. Worboys, A unified model for spatial and temporal information, *Computer J.*, **37**(1): 26–34, 1994.
41. C. Chen and C. Zaniolo, SQLST: a spatio-temporal data model and query language, *Proc. of Int'l Conference on Conceptual Modeling / the Entity Relational Approach, 2000*.
42. T. Griffiths, A. Fernandes, N. Paton, K. Mason, B. Huang, M. Worboys, C. Johnsonon, and J.G. Stell, Tripod: a comprehensive system for the management of spatial and aspatial historical objects, *Proc. of 9th ACM Symposium on Advances in Geographic Information Systems, 2001*.
43. B. Huang and C. Claramunt, STOQL: an ODMG-based spatio-temporal object model and query language, *Proc. of 10th Int'l Symposium on Spatial Data Handling, 2002*.

ELISA BERTINO
Purdue University
West Lafayette, Indiana
GIOVANNA GUERRINI
Università degli Studi di Genova
Genova, Italy

PROCESS-AWARE INFORMATION SYSTEMS: DESIGN, ENACTMENT, AND ANALYSIS

INTRODUCTION

Information technology has changed business processes within and between enterprises. More and more work processes are being conducted under the supervision of information systems that are driven by process models. Examples are workflow management systems such as FileNet P8, Staffware, WebSphere, FLOWer, and YAWL and enterprise resource planning (ERP) systems such as SAP and Oracle. Moreover, many domain-specific systems have components driven by (process) models. It is hard to imagine enterprise information systems that are unaware of the processes taking place. Although the topic of business process management using information technology has been addressed by consultants and software developers in depth, more fundamental approaches toward such *process-aware information systems* (PAISs) have been rare (1). Only since the 1990s have researchers started to work on the foundations of PAISs.

The goal of this article is to (a) provide an overview of PAISs and put these systems in a historical context, (b) to show their relevance and potential to improve business processes, dramatically, and (c) to discuss some more advanced topics to provide insights in current challenges and possible inhibitors. Before going into more detail, we first provide some definitions and give an overview of the different types of PAISs.

PAISs play an important role in *business process management* (BPM). Many definitions of BPM exist. Here we will use the following definition: "Business process management (BPM) is a field of knowledge that combines knowledge from information technology and knowledge from management sciences and applies this to operational business processes." BPM can be viewed as an extension of *workflow management* (WFM), which primarily focuses on the automation of business processes.

Figure 1 shows the relation among PAISs, BPM, and WFM. Note that the term "PAIS" refers to software, whereas the terms "BPM" and "WFM" refer to fields of knowledge in which PAISs can be used. Workflow management systems (WFMSs) can be seen as a particular kind of PAISs where the emphasis is on process automation rather than on redesign and analysis. A definition of WFMS could be "a generic software system that is driven by explicit process designs to enact and manage operational business processes." Clearly, a WFMS should be process-aware and generic in the sense that it is possible to modify the processes it supports. Note that the process designs automated by a WFMS are often graphical and that the focus is on structured processes that need to handle many cases.

Although a WFMS can be seen as a prototypical example of a PAIS, not all PAISs can be classified as pure WFMSs. As shown in Fig. 1, WFMSs are considered to be a subclass of all PAISs. Many examples of systems are process-aware, but they do not provide a generic approach to the modeling and enactment of operational business processes. For example, there may be systems where processes are hard-coded and cannot be modified. For example, many processes supported by an ERP system (e.g., SAP R/3) are hard-coded in software and can only be modified through explicit configuration parameters; that is, the set of possible variation points is predefined, and no notion of a process model can be modified freely. Many organizations have developed software to support processes without using a WFMS; for example, many banks, hospitals, electronic shops, insurance companies, and municipalities have created custom-made software to support processes. These systems are process-aware but are developed without using a WFMS. Another difference between PAISs and WFMSs is the fact that process automation is just one aspect of BPM. Process analysis and diagnosing existing processes clearly extend the scope beyond pure process automation.

Figure 1 also shows some more terms that are relevant in this context (BPR, SOA, BAM, etc.). Business process redesign (BPR) is concerned with finding better process designs. BPR efforts can be supported and enabled by PAISs. Business activity monitoring (BAM) uses information about running processes extracted from PAISs. Process mining techniques can be used to analyze this information and to come up with ideas to improve processes. Recently, the so-called *service-oriented architecture* (SOA) has been proposed as a platform for realizing PAISs. SOA is an architectural style whose goal is to achieve loose coupling among interacting parties. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by different pieces of software. The provider and consumer may reside in the same organization but also within different organizations. By using the SOA, it becomes easier to compose and maintain PAISs, because application functionality can be wrapped into servers that are invoked using a BPEL engine. The Business Process Execution Language (BPEL) (2) is the de facto standard for process execution in a SOA environment. In BPEL one can specify processes and enact them using the process engines of systems such as IBM's WebSphere or Oracle BPEL.

BPEL is a textual XML-based language, and its constructs are close to programming (2). People talk about "programming in the large" illustrating that it is not easy for nonprogrammers to model processes using BPEL. Therefore, languages such as BPMN (Business Process Modeling Notation) (3) have been proposed. Note that many modeling tools support languages similar to BPMN (e.g., ARIS and Protos). Figure 1 shows that the emphasis of execution languages like BPEL is on enactment, whereas languages like BPMN, EPCs, and Protos focus more on (re)design. Note that BPMN is not executable and has no formal semantics. However, in many cases, it is possible to generate some BPEL template code (3,4).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

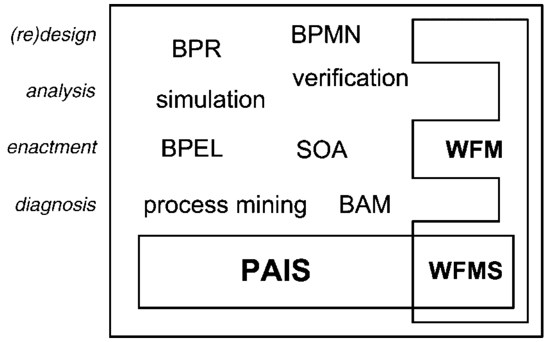


Figure 1. Relating PAISs to other approaches and tools in BPM.

A PAIS requires the modeling of different perspectives (e.g., control flow, information, and organization/ resources). This article will mention the different perspectives, but it will focus primarily on the control-flow perspective. Moreover, we use a particular technique to model this perspective: Petri nets (5).

In the remainder of this article, we will first put BPM and related PAIS technology in their historical context (see the next Section 2). Then, we discuss models for process design. As PAIS are typically driven by explicit models, it is important to use the right techniques. Therefore, we discuss techniques for the analysis of process models. We will argue that it is vital to have techniques to assert the correctness of workflow designs. Based on this argument, we focus on systems for process enactment (i.e., systems that actually make the "work flow" based on a model of the processes and organizations involved). Finally, we focus on two more advanced topics: process flexibility and process mining.

BUSINESS PROCESS MANAGEMENT FROM A HISTORICAL PERSPECTIVE

To show the relevance of PAISs, it is interesting to put them in a historical perspective (6). Consider Fig. 2, which shows some ongoing trends in information systems. This figure shows that today's information systems consist of several layers. The center is formed by the operating system (i.e., the software that makes the hardware work). The second layer consists of generic applications that can be used in a wide range of enterprises. Moreover, these applications are typically used within multiple departments within the same enterprise. Examples of such generic applications are a database management system, a text editor, and a spreadsheet program. The third layer consists of domain-specific applications. These applications are only used within specific types of enterprises and departments. Examples are decision support systems for vehicle routing, call center software, and human resource management software. The fourth layer consists of tailor-made applications. These applications are developed for specific organizations.

In the 1960s, the second and third layer were missing. Information systems were built on top of a small operating system with limited functionality. As no generic nor domain-specific software was available, these systems mainly consisted of tailor-made applications. Since then, the second and third layer have developed and the ongoing trend is that the four circles are increasing in size (i.e., they are moving to the outside while absorbing new functionality). Today's operating systems offer much more functionality. Database management systems that reside in the second layer offer functionality that used to be in tailor-made applications. As a result of this trend, the emphasis has shifted from programming to assembling of complex software systems. The challenge no longer is the coding of individual modules but the orchestrating and gluing together of pieces of software from each of the four layers.

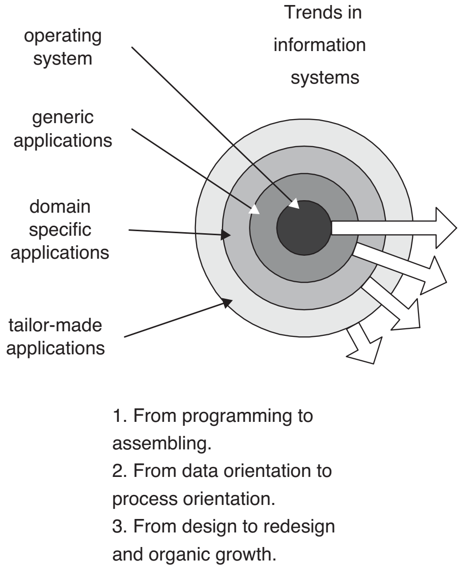

Trends in information systems
operating system
generic applications
domain specific applications
tailor-made applications
1. From programming to assembling.
2. From data orientation to process orientation.
3. From design to redesign and organic growth.

Figure 2. Trends relevant for business process management.

Another trend is the shift from data to processes. The 1970s and 1980s were dominated by data-driven approaches. The focus of information technology was on storing and retrieving information, and as a result, data modeling was the starting point for building an information system. The modeling of business processes was often neglected, and processes had to adapt to information technology. Management trends such as business process reengineering illustrate the increased emphasis on processes. As a result, system engineers are resorting to a more process-driven approach.

The last trend we would like to mention is the shift from carefully planned designs to redesign and organic growth. Because of to the omnipresence of the Internet and its standards, information systems change on the fly. Few systems are built from scratch. In most cases, existing applications are partly used in the new system. As a result, software development is much more dynamic.

The trends shown in Fig. 2 provide a historical context for PAISs. PAISs are either separate applications residing in the second layer or are integrated components in the domain-specific applications (i.e., the third layer). Notable examples of PAISs residing in the second layer are WFMSs (7–9) such as Staffware, FileNet P8, and COSA, and case handling systems such as FLOWer. Middleware platforms such as IBM's WebSphere provide a workflow engine (typically based on BPEL (2). Moreover, many open-source WFMSs exist (cf. ActiveBPEL, Enhydra-Shark, jBPM, and YAWL). Note that leading ERP systems populating the third layer also offer a workflow management module. The workflow engines of SAP, Baan, PeopleSoft, Oracle, and JD Edwards can be considered integrated PAISs. The idea to isolate the management of business processes in a separate component is consistent with the three trends identified. PAISs can be used to avoid hard-coding the work processes into tailor-made applications and thus support the shift from programming to assembling. Moreover, process orientation, redesign, and organic growth are supported. For example, today's workflow management systems can be used to integrate existing applications and to support process change by merely changing the workflow diagram. Given these observations, we hope to have demonstrated the practical relevance of PAISs. In the remainder of this article, we will focus more on the scientific importance of these systems. Moreover, for clarity, we will often restrict the discussion to clear-cut business process management systems such as WFMSs.

An interesting starting point from a scientific perspective is the early work on office information systems. In the 1970s, people like Skip Ellis (10), Anatol Holt (11), and Michael Zisman (12) already worked on so-called office information systems, which were driven by explicit process models. It is interesting to see that the three pioneers in this area independently used Petri net variants to model office procedures. During the 1970s and 1980s, there was great optimism about the applicability of office information systems. Unfortunately, few applications succeeded. As a result of these experiences, both the application of this technology and the research almost stopped for a decade. Consequently, hardly any advances were made in the 1980s. In the 1990s, there again was a huge interest in these systems. The number of WFMSs developed in the past decade and the many papers on workflow technology illustrate the revival of office information systems. Today WFMSs are readily available. However, their application is still limited to specific industries such as banking and insurance. As indicated by Skip Ellis, it is important to learn from these ups and downs. The failures in the 1980s can be explained by both technical and conceptual problems. In the 1980s, networks were slow or not present at all, there were no suitable graphical interfaces, and proper development software was missing. However, more fundamental problems also existed: A unified way of modeling processes was missing, and the systems were too rigid to be used by people in the workplace. Most of the technical problems have been resolved by now. However, the more conceptual problems remain. Good standards for business process modeling are still missing, and even today's WFMSs are too rigid.

One of the great challenges of PAISs is to offer both support and flexibility. Today's systems typically are too rigid, thus, forcing people to work around the system. One of the problems is that software developers and computer scientists are typically inspired by processes inside a computer system rather than by processes outside a computer. As a result, these engineers think in terms of control systems rather than in terms of support systems, which explains why few of the existing WFMSs allow for the so-called implicit choice (i.e., a choice resolved by the environment rather than by the system).

To summarize we would like to state that, although the relevance of PAISs is undisputed, many fundamental problems remain to be solved. In the remainder of this article, we will try to shed light on some of these problems.

MODELS FOR PROCESS DESIGN

PAISs are driven by models of processes and organizations (1). By changing these models, the behavior of the system adapts to its environment and changing requirements. These models cover different perspectives. Figure 3 shows some of the perspectives relevant for PAISs (9). The process perspective describes the control flow (i.e., the ordering of tasks). The information perspective describes the data that are used. The resource perspective describes the structure of the organization and identifies resources, roles, and groups. The task perspective describes the content of individual steps in the processes. Each perspective is relevant. However, the process perspective is dominant for the type of systems addressed in this article.

Many techniques have been proposed to model the process perspective. Some of these techniques are informal in the sense that the diagrams used have no formally defined semantics. These models are typically very intuitive, and the interpretation shifts depending on the modeler, application domain, and characteristics of the business processes at hand. Examples of informal techniques are ISAC, DFD, SADT, and IDEF. These techniques may serve well for discussing work processes. However, they are inadequate for directly driving information systems because they are incomplete and subject to multiple interpretations. Therefore, more precise ways of modeling are required.

Figure 4 shows an example of an order handling process modeled in terms of a so-called workflow net (13). Workflow nets are based on the classic Petri net model invented by Carl Adam Petri in the early 1960s (5). The squares are

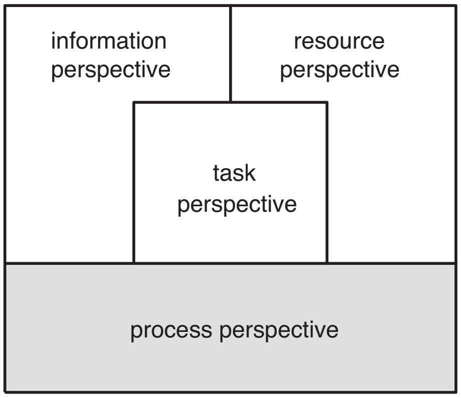


Figure 3. Perspectives of models driving PAISs.

the active parts of the model and correspond to tasks. The circles are the passive parts of the model and are used to represent states. In the classic Petri net, the squares are named transitions and the circles places. A workflow net models the lifecycle of one case. Examples of cases are insurance claims, tax declarations, and traffic violations. Cases are represented by tokens, and in this case, the token in *start* corresponds to an order. Task *register* is a so-called AND-split and is enabled in the state shown. The arrow indicates that this task requires human intervention. If a person executes this task, the token is removed from place *start* and two tokens are produced: one for *c1* and one for *c2*. Then, in parallel, two tasks are enabled: *check_availability* and *send_bill*. Depending on the eagerness of the workers executing these two tasks, either *check_available* or *send_bill* is executed first. Suppose *check_availability* is executed first. If the ordered goods are available, they can be shipped by executing task *ship_goods*. If they are not available, either a replenishment order is issued or not. Note that *check_availability* is an OR-split and produces one token for *c3*, *c4*, or *c5*. Suppose that not all ordered goods are available but that the appropriate replenishment orders were already issued. A token is produced for *c3*, and task *update* becomes enabled. Suppose that at this point in time task *send_bill* is executed, resulting in the state with a token in *c3* and *c6*. The token in *c6* is input for two tasks. However, only one of these tasks can be executed, and in this state, only *receive_payment* is enabled. Task *receive_payment* can be executed the moment the payment is received. Task *reminder* is an AND-join/AND-split and is blocked until the bill is sent and the goods have been shipped. Note that the reminder is sent after a specified period as indicated by the clock symbol. However, it is only possible to send a remainder if the goods have been actually shipped. Assume that in the state with a token in *c3* and *c6,* task *update* is executed. This task does not require human involvement and is triggered by a message of the warehouse indicating that relevant goods have arrived. Again *check_availability* is enabled. Suppose that this task is executed and that the result is positive. In the resulting state, *ship_goods* can be executed. Now there is a token in *c6* and *c7*, thus enabling task *reminder*. Executing task *reminder* again enables the task *send_bill*. A new copy of the bill is sent with the appropriate text. It is possible to send several reminders by alternating *reminder* and *send_bill*. However, let us assume that after the first loop, the customer pays resulting in a state with a token in *c7* and *c8*. In this state, the AND-join *archive* is enabled and executing this task results in the final state with a token in *end*.

This very simple workflow net shows some of the routing constructs relevant for business process modeling. Sequential, parallel, conditional, and iterative routing are present in this model. There also are more advanced constructs such as the choice between *receive_payment* and *reminder*, which is a so-called *implicit choice* (14) because it is not resolved by the system but by the environment of the system. The moment the bill is sent, it is undetermined whether *receive_payment* or *reminder* will be the next step in the process. Another advanced construct is the fact that task *reminder* is blocked until the goods have been shipped. The latter construct is a so-called *milestone* (14). The reason that we point out both constructs is that many systems have problems supporting these fundamental process patterns (14).

Workflow nets have clear semantics. The fact that one can play the so-called "token game" using a minimal set of rules shows the fact that these models are executable. None of the informal informal techniques mentioned before (i.e., ISAC, DFD, SADT, and IDEF) have formal semantics. Besides these informal techniques, many formal techniques also exist. Examples are the many variants of process algebra and statecharts. The reason we prefer to use a variant of Petri nets is threefold (13):

- Petri nets are graphical and yet precise.
- Petri nets offer an abundance of analysis techniques.
- Petri nets treat states as first-class citizens.

The latter point deserves some more explanation. Many techniques for business process modeling focus exclusively on the active parts of the process (i.e., the tasks), which is very strange because in many administrative processes, the actual processing time is measured in minutes and the flow time is measured in days. This process for measuring means that most of the time cases are in between two subsequent tasks. Therefore, it is vital to model these states explicitly.

At the beginning of this section, we mentioned that there are informal techniques (without formal semantics) and rigorous formal methods such as Petri nets. Over the last two decades, many semiformal methods have been proposed (i.e., in between the two extreme classes mentioned earlier). These methods are informal, however, because the models that need to be transformed into executable code for more rigorous interpretations are added afterward. The UML (Unified Modeling Language) (15) in an example of such a language. It has become the de facto standard for software development. UML has four diagrams for process modeling. UML supports variants of statecharts, and its activity diagrams are inspired by Petri nets (i.e., a token-based semantics is used). Many notations exist that are at the same level as UML activity diagrams. BPMN (3) diagrams and event-driven process chains (EPCs) (16) are examples of such languages. Many researchers are trying to provide solid semantics for UML, EPCs, BPMN, BPEL, and so on. For subsets of these languages, there are formalizations in terms of Petri nets and transition systems. These formalizations typically reveal ambiguous constructs in the corresponding language.

Note that the goal of this article is not to advocate Petri nets as an end-user modeling language. UML, EPCs, and BPMN provide useful constructs that support the workflow designer. However, Petri nets serve as an important foundation for PAIS technology. Without such foundations it is impossible to reason about semantics, correctness, completeness, and so on. A nice illustration is the OR-join in EPC and BPMN models that have semantics leading to paradoxes such as the "vicious circle" (16). Moreover, a solid foundation can be used for analysis as will be shown next.

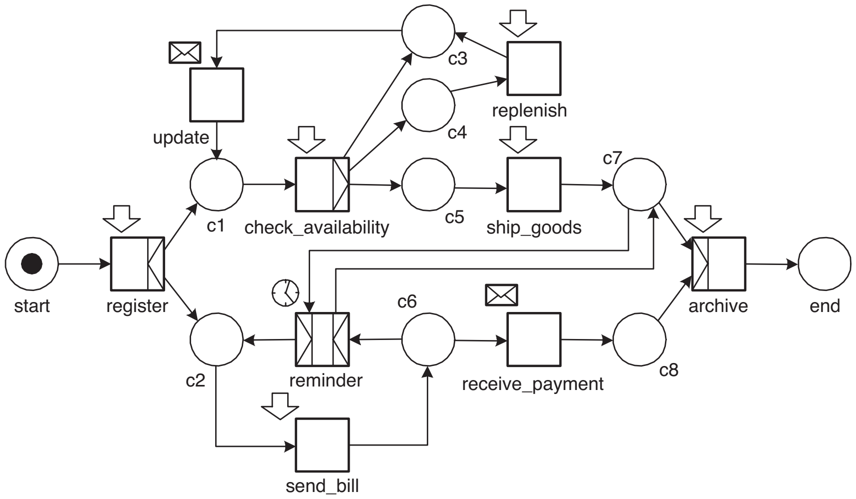


Figure 4. A WF net modeling the handling of orders. The top part models the logistical part of the process, whereas the bottom part models the financial part.

TECHNIQUES FOR PROCESS ANALYSIS

Many PAISs allow organizations to change their processes by merely changing the models. The models are typically graphical and can be changed ability to change easily. This provides more flexibility than conventional information systems. However, by reducing the threshold for change, errors are introduced more easily. Therefore, it is important to develop suitable analysis techniques. However, it is not sufficient to just develop these techniques. It is as least as important to look at methods and tools to make them applicable in a practical context.

Traditionally, most techniques used for the analysis of business processes originate from operations research. All students taking courses in operations management will learn to apply techniques such as simulation, queuing theory, and Markovian analysis. The focus mainly is on *performance analysis,* and less attention is paid to the correctness of models. *Verification* and *validation* are often neglected. As a result, systems fail by not providing the right support or even break down. Verification is needed to check whether the resulting system is free of logical errors. Many process designs suffer from deadlocks and livelocks that could have been detected using verification techniques. Validation is needed to check whether the system actually behaves as expected. Note that validation is context dependent, whereas verification is not. A system that deadlocks is not correct in any situation. Therefore, verifying whether a system exhibits deadlocks is context independent. Validation is context dependent and can only be done with knowledge of the intended business process.

To illustrate the relevance of validation and verification and to demonstrate some of the techniques available, we return to the workflow net shown in Fig. 4. This workflow process allows for the situation where a replenishment is issued before any payment is received. Suppose that we want to change the design such that replenishments are delayed until receiving payment. An obvious way to model this change is to connect task *receive_payment* with *replenish* using an additional place *c9* as shown in Fig. 5. Although this extension seems to be correct at first glance, the resulting workflow net exhibits several errors. The workflow will deadlock if a second replenishment is needed and something is left behind in the process if no replenishments

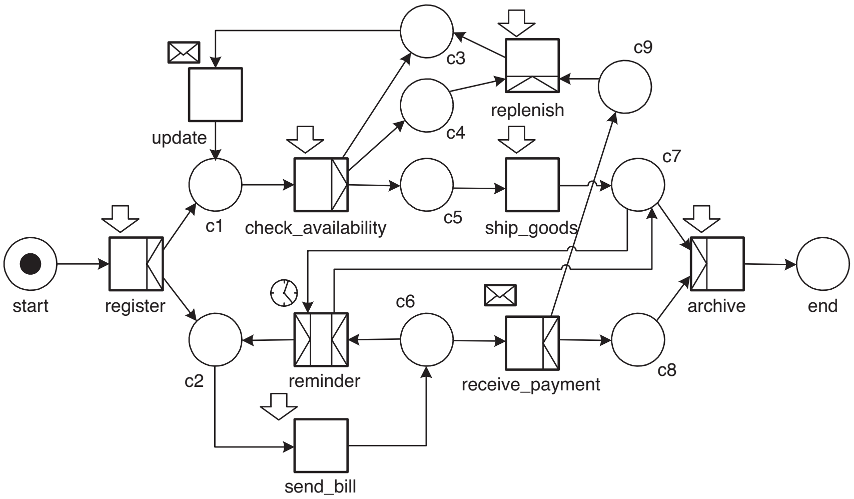


Figure 5. An incorrect WF net. Place *c9* has been added to model that a replenishment order can only be placed if the customer has paid. However, because of this addition, the process can deadlock and a token is left behind in the process if no replenishments are needed.

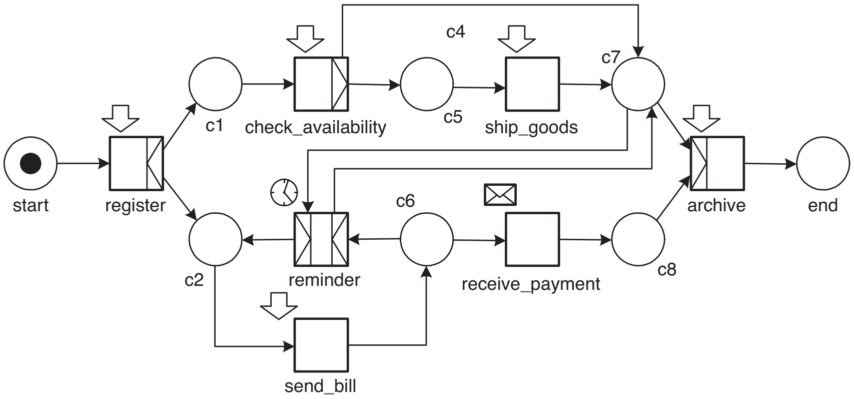


Figure 6. A sound but incorrect WF net. The shipping of goods is no longer guaranteed.

are needed. These logical errors can be detected without any knowledge of the order handling process. For verification, application-independent notions of correctness are needed. One of these notions is the so-called *soundness property*(13). A workflow net is sound if an only if the workflow contains no dead parts (i.e., tasks that can never be executed), from any reachable state it is always possible to terminate, and the moment the workflow terminates all places except the sink place are empty. Note that soundness rules out logical errors such as deadlocks and livelocks. The notion of soundness is applicable to any workflow language. An interesting observation is that soundness corresponds to liveness and boundedness of the short-circuited net (13). The latter properties have been studied extensively (17). As a result, powerful analysis techniques and tools can be applied to verify the correctness of a workflow design. Practical experience shows that many errors can be detected by verifying the soundness property. Moreover, Petri net theory can also be applied to guide the designer toward the error.

Soundness does not guarantee that the workflow net behaves as intended. Consider, for example, the workflow net shown in Fig. 6. Compared with the original model, the shipment of goods is skipped if some goods are not available. Again this idea may seem like a good one at first glance. However, customers are expected to pay even if the goods are never delivered. In other words, task *receive_payment* needs to be executed although task *ship_goods* may never be executed. The latter error can only be detected using knowledge about the context. Based on this context, one may decide whether this is acceptable. Few analysis techniques exist to support this kind of validation automatically. The only means of validation offered by today's WFMSs is gaming and simulation.

An interesting technique to support validation is inheritance of dynamic behavior. Inheritance can be used as a technique to compare processes. Inheritance relates subclasses with superclasses (18). A workflow net is a subclass of a superclass workflow net if certain dynamic properties are preserved. A subclass typically contains more tasks. If by hiding and/or blocking tasks in the subclass one obtains the superclass, the subclass inherits the dynamics of the superclass.[1] The superclass can be used to specify the minimal properties the workflow design should satisfy. By merely checking whether the actual design is a subclass of the superclass, one can validate the essential properties. Consider, for example, Fig. 7. This workflow net describes the minimal requirements the order handling process should satisfy. The tasks *register*, *ship_goods*, *receive_payment*, and *archive* are mandatory. Tasks *ship_goods* and *receive_payment* may be executed in parallel but should be preceded by *register* and followed by *archive*. The original order handling process shown in Fig. 4 is a subclass of this superclass. Therefore, the minimal requirements are satisfied. However, the order handling process shown in Fig. 6 is not a subclass. The fact that task *ship_goods* can be skipped demonstrates that not all properties are preserved.

Inheritance of dynamic behavior is a very powerful concept that has many applications. Inheritance-preserving transformation rules and transfer rules offer support at design time and at run time (19). Subclass–superclass relationships also can be used to enforce correct processes in an e-commerce setting. If business partners only execute subclass processes of some common contract process, then the overall workflow will be executed as agreed. It should be noted that workflows crossing the borders of organizations are particularly challenging from a verification and validation point of view. Errors resulting from miscommunication between business partners are highly disruptive and costly. Therefore, it is important to develop techniques and tools for the verification and validation of these processes. For example, in the context of SOA-based processes (e.g., BPEL processes), the so-called open WF nets (OWF-nets) (20,21) are used to study notions such as controllability and accordance.

Few mature tools aiming at the verification of workflow processes exist. Woflan (22) is one of the notable exceptions. Figure 8 shows a screenshot of Woflan. Woflan combines state-of-the-art scientific results with practical applications (22). Woflan can interface with WFMSs such as Staffware, Websphere, Oracle BPEL, COSA, and YAWL. It can also interface with BPR tools such as Protos and process mining tools such as ProM (23). Workflow processes

[1] We have identified four notions of inheritance. In this article, we only refer to life-cycle inheritance.

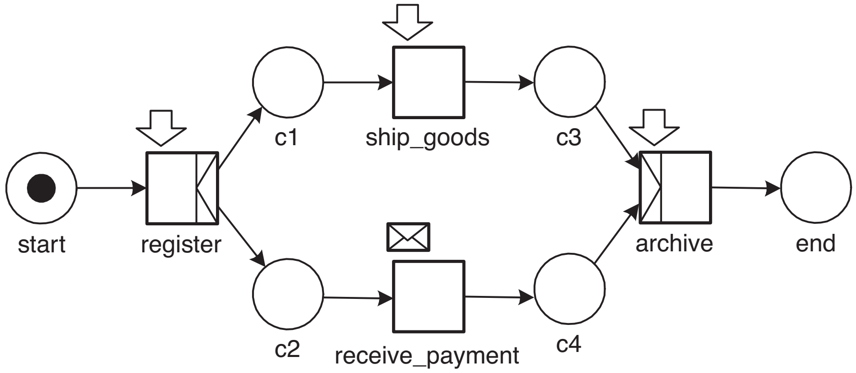
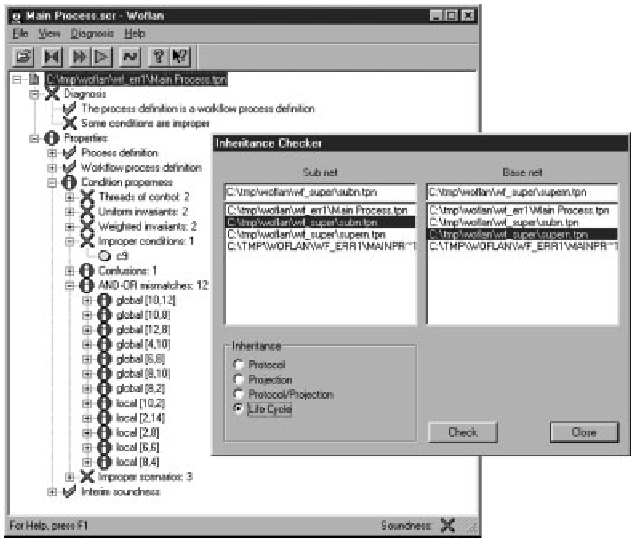


Figure 7. A superclass WF net.

designed using any of these tools can be verified for correctness. It turns out that the challenge is not to decide whether the design is sound. The real challenge is to provide diagnostic information that guides the designer to the error. Woflan also supports the inheritance notions mentioned before. Given two workflow designs, Woflan can decide whether one is a subclass of the other. Tools such as Woflan illustrate the benefits of a more fundamental approach. Recently, several tools have been developed for the analysis of BPEL processes. The Tools4BPEL toolset consisting of Fiona, LoLa, and BPEL2oWFN is an example of a state-of-the-art BPEL analyzer (21).

To conclude this section, we would like to refer to a study reported in Ref. 24. This study shows that of the 604 process models in the SAP R/3 Reference Model, 20% contain errors that can easily be discovered using verification. Since the middle of the 1990s, the SAP R/3 Reference Model has been available in different versions to support the implementation and configuration of the SAP system. The reference model is not only included in the SAP system, but also it is shipped with the business process modeling tools ARIS of IDS Scheer or NetProcess of IntelliCorp. The reference model covers several modeling perspectives such as data and organization structure, but the main emphasis is on 604 nontrivial business processes represented as EPCs. More than 20% of these EPCs contain errors stemming from incorrect combinations of connector elements such as deadlocks and livelocks. A deadlock describes a situation in a process model where a customer order remains waiting for an activity to complete that can never be executed. A simple pattern leading to a deadlock is an XOR split is joined with an AND. A livelock is an infinite loop (i.e., it is impossible to move beyond a certain point and terminate). The many errors in the SAP R/3 Reference Model illustrate the need for rigorous analysis techniques.



```
C:\tmp\woflan\wf_err1\Main Process.tpn
  Diagnosis
    The process definition is a workflow process definition
    Some conditions are improper
  Properties
    Process definition
    Workflow process definition
    Condition properness
      Threads of control: 2
      Uniform invariants: 2
      Weighted invariants: 2
      Improper conditions: 1
        c9
      Confusions: 1
      AND-OR mismatches: 12
        global [10,12]
        global [10,8]
        global [12,8]
        global [4,10]
        global [6,8]
        global [8,10]
        global [8,2]
        local [10,2]
        local [2,14]
        local [2,8]
        local [6,6]
        local [8,4]
      Improper scenarios: 3
    Interim soundness
For Help, press F1                                Soundness: ✗
```

Inheritance Checker

Sub net	Base net
C:\tmp\woflan\wf_super\subn.tpn	C:\tmp\woflan\wf_super\supern.tpn
C:\tmp\woflan\wf_err1\Main Process.tpn	C:\tmp\woflan\wf_err1\Main Process.tpn
C:\tmp\woflan\wf_super\subn.tpn	C:\tmp\woflan\wf_super\subn.tpn
C:\tmp\woflan\wf_super\supern.tpn	C:\tmp\woflan\wf_super\supern.tpn
C:\TMP\WOFLAN\WF_ERR1\MAINPR~1	C:\TMP\WOFLAN\WF_ERR1\MAINPR~1

Inheritance
○ Protocol
○ Projection
○ Protocol/Projection
● Life Cycle

Figure 8. A screenshot showing the verification and validation capabilities of Woflan.

SYSTEMS FOR PROCESS ENACTMENT

Progress in computer hardware has been incredible. In 1964 Gordon Moore predicted that the number of elements on a produced chip would double every 18 months.[2] Up until now, Moore's law still applies. Information technology has also resulted in a spectacular growth of the information being gathered. The commonly used term "information overload" illustrates this growth. Already in 2003, it is estimated that for each individual (i.e., child, man, and woman), 800 megabytes of data are gathered each year (25). The Internet and the World Wide Web have made an abundance of information available at low costs. However, despite the apparent progress in computer hardware and information processing, many information systems leave much to be desired. One problem is that process logic is mixed with application logic. As a result, it is difficult to change a system and people need to "work around the system" rather than getting adequate support. To improve flexibility and reliability, process logic should be separated from application logic. These observations justify the use of solid models and analysis techniques, as discussed before.

Thus far, the focus of this article has been on the design and analysis of work processes. Now it is time to focus on the systems to enact these work processes. Fig. 9 shows the typical architecture of a business process management system. The designer uses the design tools to create models describing the processes and the structure of the organization. The manager uses management tools to monitor the flow of work and act if necessary. The worker interacts with the enactment service. The enactment service can offer work to workers, and workers can search, select, and per-

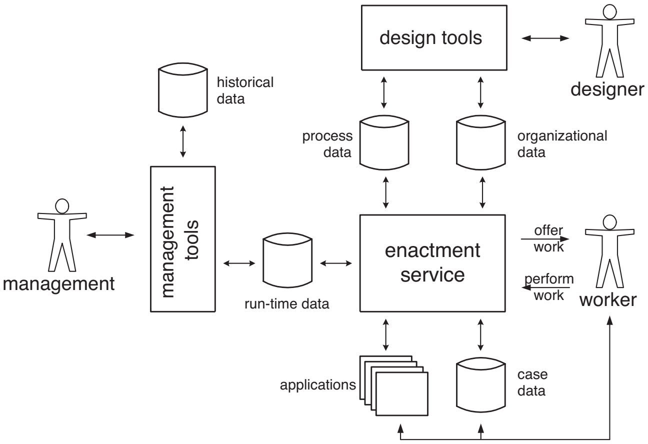


Figure 9. The architecture of a PAIS.

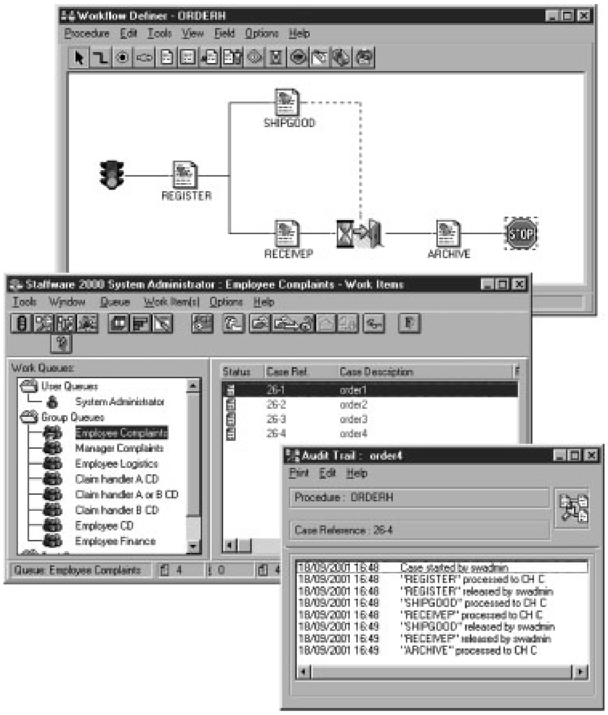
form work. To support the execution of tasks, the enactment service may launch various kinds of applications. Note that the enactment service is the core of the system deciding on "what," "how," "when," and "by whom." Clearly, the enactment service is driven by models of the processes and the organizations. By merely changing these models, the system evolves and adapts, which is the ultimate promise of PAISs.

However, PAIS systems are not the "silver bullet" that solves all problems (i.e., "there is no such thing as a free lunch"), and rigorous modeling is needed to capture processes adequately. Moreover, existing WFMSs still have problems supporting flexibility.

Today's WFMSs have an architecture consistent with Fig. 9. Consider, for example, the screenshots of Staffware shown in Fig. 10. Staffware is one of the leading WFMSs. The top window shows the design tool of Staffware while defining a simple workflow process. Work is offered through so-called work queues. One worker can have multiple work queues, and one work queue can be shared among multiple workers. The window in the middle shows the set of available work queues (left) and the content of one of these work queues (right). The bottom window shows an audit trail of a case. The three windows show only some of the capabilities offered by contemporary workflow management systems. It is fairly straightforward to map these windows onto the architecture. In other processes-aware information systems such as, for example, enterprise resource planning systems, one will find the architecture shown in Fig. 9 embedded in a larger architecture.

The architecture shown in Fig. 9 assumes a centralized enactment service. Inside a single organization such an assumption may be realistic. However, in a cross-organizational setting, this is not the case. Fortunately, most vendors now support the SOA, mentioned earlier. In a SOA, tasks are subcontracted to other parties (i.e., what is one task for the service consumer may be a complex process for a service consumer). The Web services stack using standards such as WSDL and BPEL facilitates the development of cross-organizational workflows.

Despite the acceptance of PAISs, the current generation of products leaves much to be desired. To illustrate,



Status	Case Ref	Case Description
	26-1	order1
	26-2	order2
	26-3	order3
	26-4	order4

Procedure : ORDERH

Case Reference : 26-4

```
18/09/2001 16:48    Case started by swadmin
18/09/2001 16:48    'REGISTER' processed to CH C
18/09/2001 16:48    'REGISTER' released by swadmin
18/09/2001 16:48    'SHIPGOOD' processed to CH C
18/09/2001 16:48    'RECEIVEP' processed to CH C
18/09/2001 16:49    'SHIPGOOD' released by swadmin
18/09/2001 16:49    'RECEIVEP' released by swadmin
18/09/2001 16:49    'ARCHIVE' processed to CH C
```

Figure 10. The Graphical Workflow Definer, Work Queue, and Audit Trail of Staffware.

we focus on the current generation of WFMSs. We will use Fig. 9 to identify five problems.

First of all, there is a lack of good standards for workflow management. There is, for example, not a good standard for exchanging process models. Existing formats have no clearly defined semantics and fail to capture many routing constructs. Current standards for workflow management are incomplete, inconsistent, at the wrong abstraction level, and mainly driven by the commercial interests of workflow vendors. The Workflow Management Coalition (WfMC) has been trying to standardize workflow processes since the early 1990s. This effort resulted in the Workflow Process Definition Language (WPDL) and the XML Process Definition Language (XPDL). Only a few vendors actively supported these standards. The standards had no clearly defined semantics and encouraged vendors to make product-specific extensions. The BPEL (2) emerged later and is currently the de facto standard for process execution in a SOA environment. However, this language also has no clearly defined semantics and is at a technical level (26). BPEL has the look and feel of a programming language rather than a high-level modeling language.

Second, the expressive power (i.e., the ability to represent complex work processes) of the current generation of WFMSs is insufficient. Several evaluations revealed that the classic WFMSs support less than half of the desirable workflow patterns (14). As an example, consider the workflow process shown in Fig. 4. Few systems can handle the implicit choice and milestone construct identified before. Fortunately, modern systems (e.g., based on BPEL) support more patterns.

A third problem is the lack of understanding of how people actually work. Work processes are more than the ordering of tasks. Work is embedded in a social context. A better understanding of this context is needed to make systems socially aware as well. Modeling processes as if people are "machines" is a too limited view of reality. It is vital to empower workers and to provide more flexibility.

The fourth problem is the limited support for workflow analysis. As indicated before, there are powerful techniques for workflow analysis. However, few systems embed advanced analysis techniques. Besides model-based verification, validation, and performance analysis, new types of analysis are possible. The combination of historical and run-time data, on the one hand, and workflow designs, on the other, offers breathtaking possibilities. Historical data can be used to obtain stochastic data about routing and timing. Using run-time data to reconstruct the current state in a simulation model allows for *on-the-fly simulation*. Simulation based on the current state, historical data, and a good model offers high-quality information about potential problems in the near future. Historical data can also be used for *process mining*. The goal of workflow mining is to derive process models from transaction logs.

Finally, many technical problems remain. Some of these problems can be resolved using Internet-based technology and standards. However, many problems related to the integration of components and long-lived transactions remain unsolved. Since the early 1990s (8) many database researchers have been focusing on transactional aspects of workflows. Note that an instance of a workflow can be viewed as a long running transaction [e.g., some cases (such as a mortgage or insurance) may run dozens of years] (27).

In the remainder, we would like to focus on two particular challenges: *process flexibility* and *process mining*.

CHALLENGE: FLEXIBILITY

Adaptability has become one of the major research topics in the area of workflow management (28). Today's WFMSs and many other PAISs have problems supporting flexibility. As a result, these systems are not used to support dynamically changing business processes or the processes are supported in a rigid manner (i.e., changes are not allowed or handled outside of the system). These problems have been described and addressed extensively in the literature (29–36). Nevertheless, many problems related to flexibility remain unsolved.

In this section, we provide a taxonomy of flexibility because it is probably the biggest challenge today's PAISs are facing. To clarify things, we focus on WFMSs rather than on the broader class of PAISs.

To start, let us identify the different *phases* of a process (instance) in the context of a WFMS:

- *Design time*. At design time, a generic process model is created. This model cannot be enacted because it is not connected to some organizational setting.
- *Configuration time*. At configuration time, a generic model is made more specific and connected to some organizational context that allows it to be instantiated.
- *Instantiation time*. At instantiation time, a process instance is created to handle a particular case (e.g., a customer order or travel request).
- *Run time*. At run time, the process instance is executed according to the configured model. The different activities are being enacted as specified.
- *Auditing time*. At auditing time, the process instance has completed; however, its audit trail is still available and can be inspected and analyzed.

Flexibility plays a role in most phases. At design time, some modeling decisions can be postponed to run time. At run time, one can decide to deviate from the model, and at instantiation time, one can change the process model used for the particular instance. When it comes to flexibility, we identify three *flexibility mechanisms*:

- *Defer, i.e., decide to decide later*. This flexibility mechanism deliberately leaves freedom to maneuver at a later phase. Examples are the use of a declarative process modeling language that allows for the "under specification" of processes and the use of late binding (i.e., the process model has a "hole" that needs to be filled in a later phase).
- *Change, i.e., decide to change model*. Most researchers have addressed flexibility issues by allowing for change. Decisions made at an earlier phase may be revisited. For example, for premium customers, the process may be adapted in an ad hoc manner. The change may refer to the model for a single instance (adhoc change) or to all future cases (evolutionary change). In both cases, a change can create inconsistencies. For evolutionary change, cases may also need to be migrated.
- *Deviate, i.e., decide to ignore model*. The third mechanism is to deviate from the model (e.g., tasks are skipped even if the model does not allow for this to happen). In many environments, it is desirable that people are in control (i.e., the system can only suggest activities but not force them to happen).

Figure 11 relates the two dimensions just mentioned. Based on the different phases and the three mechanisms, different types of flexibility are classified. Note that we did not mention any examples of flexibility at auditing time. After the process instance completes, it is not possible to defer, change, or violate things, because this would imply fraud. Figure 11 can be used to characterize the support for flexibility of a concrete WFMS. Unfortunately, today's systems support only a few forms of flexibility, thus limiting the applicability of PAISs.

It is impossible to provide a complete overview of the work done on flexibility in workflows. The reader is referred to Ref. 28 for another taxonomy. Many authors have focused on the problems related to change (31–34) (cf. the cell "change at run time" in Fig. 11). The problem of changing a process while instances are running was first

	defer (decide to decide later)	change (decide to change model)	deviate (decide to ignore model)
design time	e.g., defer to run-time by using late binding or declarative modeling	N/A	N/A
configuration time	e.g., defer configuration decisions	e.g., remodel parts of the process at configuration time	e.g., violate a configuration constraint
instantiation time	e.g., defer the selection of parameters or process fragments	e.g., modify model for a particular customer	N/A
run time	N/A	e.g., change model for running instance or migrate instance to new model	e.g., skip or redo a task while this is not specified
auditing time	N/A	N/A	N/A

Figure 11. Classification of the different types of flexibility based on the phase and mechanism.

mentioned in Ref. 31. In the context of ADEPT (32,34), many problems have been addressed. See Ref. 34 for an excellent overview of problems related to dynamic change (the cell "change at run time" in Fig. 11). Other authors approach the problem by avoiding change, for example, either by using a more declarative language (33) or by late binding (30,35,36) (also referred to as worklets, pockets of flexibility, or process fragments). These approaches fit into the column "defer" in Fig. 11. Another interesting approach is provided by the system FLOWer of Pallas Athena. This system uses the so-called "case handling" concept to provide more flexibility (29). Most of the ideas of case handling relate to the "defer" and "deviate" columns in Fig. 11.

CHALLENGE: PROCESS MINING

Process mining has emerged as a way to analyze systems and their actual use based on the event logs they produce (37,38). Process mining always starts with *event logs*. Events logs may originate from all kinds of PAISs. Examples are classic WFMSs, ERP systems (e.g., SAP), case handling systems (e.g., FLOWer), PDM systems (e.g., Windchill), CRM systems (e.g., Microsoft Dynamics CRM), middleware (e.g., IBM's WebSphere), hospital information systems (e.g., Chipsoft), and so on. These systems provide very detailed information about the activities that have been executed.

The goal of process mining is to extract information (e.g., process models) from these logs (i.e., process mining describes a family of *a posteriori* analysis techniques exploiting the information recorded in the event logs). Typically, these approaches assume that it is possible to record events sequentially such that each event refers to an activity (i.e., a well-defined step in the process) and is related to a particular case (i.e., a process instance). Furthermore, some mining techniques use additional information such as the performer or originator of the event (i.e., the person/resource executing or initiating the activity), the timestamp of the event, or data elements recorded with the event (e.g., the size of an order).

Process mining addresses the problem that most "process/system owners" have limited information about what is actually happening. In practice, there is often a significant gap between what is prescribed or supposed to happen and what *actually* happens. Only a concise assessment of reality, which process mining strives to deliver, can help in verifying process models and ultimately be used in system or process redesign efforts.

The idea of process mining is to discover, monitor, and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs. We consider three basic types of process mining (Fig. 12):

- *Discovery*: No a priori model exists (i.e., based on an event log, some model is constructed). For example, using the α-algorithm (37), a process model can be discovered based on low-level events.
- *Conformance*: An a priori model exists. This model is used to check whether reality conforms to the model. For example, a process model may indicate that purchase orders of more than one million Euro require two checks. Another example is the checking of the

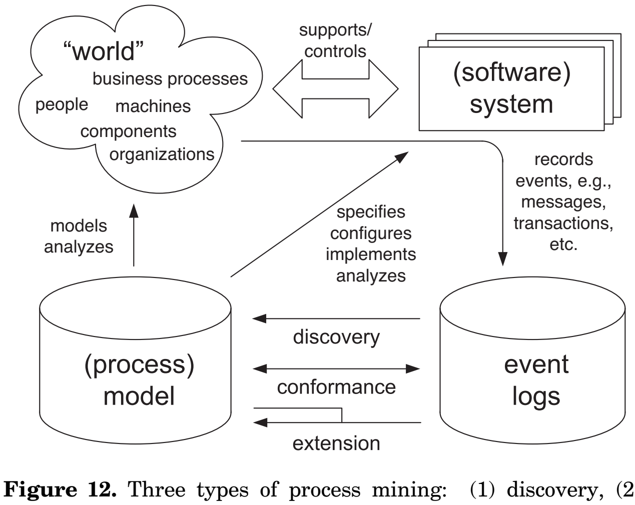


Figure 12. Three types of process mining: (1) discovery, (2) conformance, and (3) extension.

four-eyes principle. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations.

- *Extension*: An a priori model exists. This model is extended with a new aspect or perspective (i.e., the goal is not to check conformance but to enrich the model with the data in the event log). An example is the extension of a process model with performance data (i.e. some a priori process model is used on which bottlenecks are projected).

Traditionally, process mining has been focusing on *discovery* (i.e., deriving information about the original process model, the organizational context, and execution properties from enactment logs). An example of a technique addressing the control flow perspective is the α-algorithm, which constructs a Petri net model (17) describing the behavior observed in the event log. However, process mining is not limited to process models (i.e., control flow) and recent process mining techniques are more and more focusing on other perspectives (e.g., the organizational perspective or the data perspective). For example, there are approaches to extract social networks from event logs and analyze them using social network analysis. This allows organizations to monitor how people, groups, or software/system components are working together.

Conformance checking compares an a priori model with the observed behavior as recorded in the log. In Ref. 39, it is shown how a process model (e.g., a Petri net) can be evaluated in the context of a log using metrics such as "fitness" (Is the observed behavior possible according to the model?) and "appropriateness" (Is the model "typical" for the observed behavior?). However, it is also possible to check conformance based on organizational models, predefined business rules, temporal formulas, quality of service (QoS) definitions, and so on.

There are different ways to *extend* a given process model with additional perspectives based on event logs (e.g., decision mining, performance analysis, and user profiling). Decision mining, also referred to as decision point analysis, aims at the detection of data dependencies that affect the routing of a case. Starting from a process model, one can analyze how data attributes influence the choices made in the process based on past process executions. Classic data mining techniques such as decision trees can be leveraged for this purpose. Similarly, the process model can be extended with timing information (e.g., bottleneck analysis).

At this point in time there are mature tools such as the ProM framework (23), featuring an extensive set of analysis techniques that can be applied to real-life logs while supporting the whole spectrum depicted in Fig. 12.

Although flexibility requirements may form an inhibitor for the application of PAISs, process mining techniques may in fact increase the value of a PAIS. The structured analysis of the event logs of PAISs provides an added value over information systems that are not aware of the processes these support.

Process mining is strongly related to classic data mining approaches (40). However, the focus is not on data but on process-related information (e.g., the ordering of activities). Process mining is also related to monitoring and business intelligence (41).

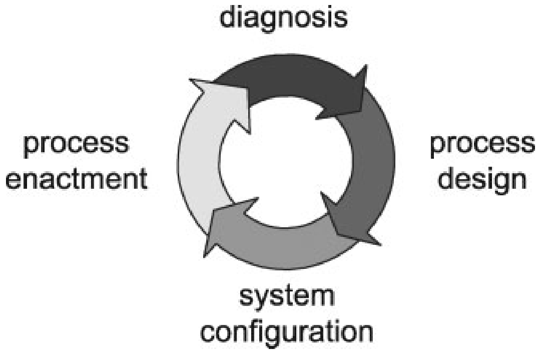


Figure 13. PAIS lifecycle.

CONCLUSION

Process-aware information systems (PAISs) follow a characteristic *lifecycle*. Figure 13 shows the four phases of such a lifecycle (7). In the *design phase*, the processes are (re)designed. In the *configuration phase*, designs are implemented by configuring a PAIS (e.g., a WFMS). After configuration, the *enactment phase* starts where the operational business processes are executed using the system configured. In the *diagnosis phase*, the operational processes are analyzed to identify problems and to find things that can be improved. The focus of traditional workflow management (systems) is on the lower half of the lifecycle. As a result, there is little support for the diagnosis phase. Moreover, support in the design phase is limited to providing a graphical editor while analysis and real design support are missing.

In this article, we showed that PAISs support operational business processes by combining advances in information technology with recent insights from management science. We started by reviewing the history of such systems and then focused on process design. From the many diagramming techniques available, we chose one particular technique (Petri nets) to show the basics. We also emphasized the relevance of process analysis, for example, by pointing out that 20% of the more than 600 process models in the SAP reference model are flawed (24). We also discussed the systems that enact such process designs (e.g., the workflow engines embedded in various systems) and concluded by elaborating on two challenges: flexibility and process mining. More flexibility is needed to widen the scope of PAISs. Today's systems tend to restrict people in their actions, even if this is not desired. Process mining is concerned with extracting knowledge from event logs. This is relatively easy in the context of PAISs and offers many opportunities to improve the performance of the underlying business processes. Moreover, process mining is an essential factor in closing the PAIS lifecycle shown in Fig. 13.

BIBLIOGRAPHY

1. M. Dumas, W. M. P. van derAalst, and A. H. M. ter Hofstede, *Process-Aware Information Systems: Bridging People and Software through, Process Technology*, New York: Wiley & Sons, 2005.
2. A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C. K. Liu, R. Khalaf, Dieter Koenig, M. Marin, V. Mehta, S. Thatte, D. Rijn, P. Yendluri, and A. Yiu, Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.
3. S. A. White et al., Business Process Modeling Notation Specification (Version 1.0, OMG Final Adopted Specification), 2006.
4. C. Ouyang, M. Dumas, A. H. M. terHofstede, and W. M. P. van der Aalst, Pattern-Based Translation of BPMN Process Models to BPEL Web Services, *Int. J. Web Serv. Res.*, **5**(1): 42–62, 2007.
5. W. Reisig and G. Rozenberg, ed., *Lectures on Petri Nets I: Basic Models,* volume 1491 *of Lecture Notes in Computer Science*, Berlin: Springer-Verlag, 1998.
6. W. M. P. van der Aalst, Making, work flow: On the application of Petri nets to business process management, in J. Esparza and C. Lakos (eds.), *Application and Theory of Petri Nets 2002*, Vol. 2360 *of Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2002, pp. 1–22.
7. W. M. P. van der Aalst and K. M. van Hee, *Workflow Management: Models, Methods, and Systems*, Cambridge, MA: MIT Press, 2004.
8. D. Georgakopoulos, M. Hornick, and A. Sheth, An overview of workflow management: From process modeling to workflow automation infrastructure, *Dist. Parallel Databases*, **3**: 119–153, 1995.
9. S. Jablonski, and C. Bussler, *Workflow Management: Modeling Concepts, Architecture, and Implementation*, London: International Thomson Computer Press, 1996.
10. C. A. Ellis, Information control nets: A mathematical model of office information flow, *Proc. Conference on Simulation, Measurement and Modeling of Computer Systems*, ACM Press: Boulder, Colorado, 1979, pp. 225–240.
11. A. W. Holt, Coordination technology and Petri nets, in G. Rozenberg (ed.), *Advances in Petri Nets 1985*, Vol. 222 *of Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 1985, pp. 278–296.
12. M. D. Zisman, *Representation, Specification and Automation of Office Procedures*. PhD thesis, University of Pennsylvania, Warton School of Business, 1977.
13. W. M. P. van der Aalst, The application of Petri nets to workflow management, *J. Circuits, Sys. Comp.*, **8**(1): 21–66, 1998.
14. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, Workflow patterns, *Dist. Parallel Databases*, **14**(1): 5–51, 2003.
15. Object Management Group, *OMG Unified Modeling Language 2.0*. OMG, Available: http://www.omg.com/uml/, 2005.
16. E. Kindler. On the semantics of EPCs: A framework for resolving the vicious circle, *Data Know. Eng.*, **56**(1): 23–40, 2006.
17. J. Desel and J. Esparza, *Free Choice Petri Nets*, Vol. 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge UK: Cambridge University Press, 1995.
18. T. Basten and W. M. P. van der Aalst, Inheritance of behavior, *J. Logic Algebraic Programming*, **47**(2): 47–145, 2001.
19. W. M. P. van der Aalst and T. Basten, Inheritance of workflows: An approach to tackling problems related to change, *Theor. Comp. Sci.*, **270**(1–2): 125–203, 2002.
20. W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. From public views to private views: Correctness-by-design for services, in M. Dumas and H. Heckel (eds.), *Informal Proc. of the 4th International Workshop on Web Services and Formal Methods (WS-FM 2007)*; QUT, Brisbane Australia, 2007, 119–134.
21. N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, Analyzing interacting BPEL processes, in S. Dustdar, J. L. Faideiro, and A. Sheth (eds.), *International Conference on Business Process Management (BPM 2006),* Vol. 4102 of *Lecture Notes in Computer Science*, Berlin: Springer-Verlag, 2006, 17–32.
22. H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst, Diagnosing workflow, processes using woflan, *Comp. J.*, **44**(4): 246–279, 2001.
23. W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, R. S. Mans, A. K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H. M. W. Verbeek, and A. J. M. M. Weijters. ProM 4.0: Comprehensive support for real process analysis, in J. Kleijn and A. Yakovlev (eds.), *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, Vol. 4546 *of Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2007, pp. 484–494.
24. J. Mendling, G. Neumann, and W. M. P. van der Aalst. Understanding the occurrence of errors in process models based on metrics, in F. Curbera, F. Leymann, and M. Weske (eds.), *Proc. OTM Conference on Cooperative information Systems (CoopIS 2007),* Vol. 4803 of *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2007, pp. 113–130.
25. P. Lyman and H. Varian, How Much Information. Available: http://www.sims.berkeley.edu/how-much-info.
26. W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed, Life after BPEL?, in M. Bravetti, L. Kloul, and G. Zavattaro (eds.), *WS-FM 2005,* Vol. 3670 of *Lecture Notes in Computer Science*, Berlin: Springer-Verlag, 2005, 35–50.
27. G. Weikum and G. Vossen, *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*, San Francisco, CA: Morgan Kaufmann Publishers, 2002.
28. W. M. P. van der Aalst and S. Jablonski, Dealing with workflow change: identification of issues and solutions, *Int. J. Comp. Sys., Sci., Eng.*, **15**(5): 267–276, 2000.
29. W. M. P. van der Aalst, M. Weske, and D. Grünbauer. Case handling: A new paradigm for business process support. *Data Knowl. Eng.*, **53**(2): 129–162, 2005.
30. M. Adams, A. H. M. ter Hofstede, W. M. P. van der Aalst, and D. Edmond, Dynamic, extensible and context-aware exception handling for workflows, in F. Curbera, F. Leymann, and M. Weske (eds.), *Proc. OTM Conference on Cooperative information Systems (CoopIS 2007),* Vol. 4803 of *Lecture Notes in Computer Science*, Berlin: Springer-Verlag, 2007, 95–112.
31. C. A. Ellis, K. Keddara, and G. Rozenberg, Dynamic change within workflow systems, in N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan (eds.), *Proc. Conference on Organizational Computing Systems*; pages 10–21, Milpitas, California, 1995.
32. M. Reichert and P. Dadam, ADEPTflex: Supporting dynamic changes of workflow without loosing control, *J. Intell. Inf. Sys.*, **10**(2): 93–129, 1998.
33. M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst. Constraint-based workflow models: Change made easy,

in F. Curbera, F. Leymann, and M. Weske (eds.), *Proc. OTM Conference on Cooperative information Systems (CoopIS 2007),* Vol. 4803 of *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2007, pp. 77–94.

34. S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems: A survey, *Data Know. Eng.*, **50**(1): 9–34, 2004.
35. S. Sadiq, W. Sadiq, and M. Orlowska, Pockets of flexibility in workflow specification, in *Proc. 20th International Conference on Conceptual Modeling (ER 2001),* Vol. 2224 *of Lecture Notes in Computer Science*, Berlin: Springer-Verlag, 2001, pp. 513–526.
36. M. Weske, Formal foundation and conceptual design of dynamic adaptations in a workflow management system, in R. Sprague (ed.), *Proc. Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-34)*. Los Alamitos, CA: IEEE Computer Society Press, 2001.
37. W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster, Workflow mining: Discovering process models from event logs, *IEEE Trans. Knowl. Data Eng.*, **16**(9): 1128–1142, 2004.
38. W. M. P. van der Aalst, H. A. Reijers, A. J. M. M. Weijters, B. F. vanDongen, A. K. Alves de Medeiros, M. Song, and H. M. W. Verbeek, Business process mining: An industrial application, *Inf. Sys.*, **32**(5): 713–732, 2007.
39. A. Rozinat and W. M. P. van der Aalst, *Conformance testing: Measuring the fit and appropriateness of event logs and process models*, in C. Bussler et al. (ed.), *BPM 2005 Workshops (Workshop on Business Process Intelligence),* volume 3812 of *Lecture Notes in Computer Science*, Berlin: Springer-Verlag, 2006, 163–176.
40. I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques* (2nd edition). San Francisco, CA: Morgan Kaufmann, 2005.
41. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M. C. Shan, Business process intelligence, *Comp. Ind.*, **53**(3): 321–343, 2004.

Wil M.P. van der Aalst
Eindhoven University of Technology
Eindhoven, The Netherlands

REAL TIME DATABASE SYSTEMS

INTRODUCTION

A real-time database system (RTDBS) is a database system providing all the features of a traditional database system such as data independence and concurrency control, while also enforcing real-time constraints that applications may have (1). Like a traditional database system, a RTDBS functions as a repository of data, provides efficient storage, and performs retrieval and manipulation of information. However, as a part of a real-time system, tasks have time constraints; therefore, an RTDBS has the added requirement of ensuring some degree of confidence in meeting the system's timing requirements (2). A real-time database is a database in which transactions have deadlines or timing constraints (3). Real-time databases are commonly used in real-time computing applications that require timely access to data. And, usually, the definition of timeliness is not quantified; for some applications, it is milliseconds, and for others, it is minutes (4).

Traditional database systems differ from an RTDBS in many aspects. Most importantly, RTDBSs have a different performance goal, correctness criteria, and assumptions about the applications. Unlike a traditional database system, an RTDBS may be evaluated based on how often transactions miss they deadlines, the average lateness or tardiness of late transactions, the cost incurred in transactions missing their deadlines, data external consistency, and data temporal consistency.

For example, a stock market changes very rapidly and is dynamic. The graphs of the different markets appear to be very unstable, and yet, a database has to keep track of the current values for all of the markets of the stock exchange.

Numerous real-word applications contain time-constrained access to data as well as access to data that has temporal validity (5). Consider, for example, a telephone switching system, network management, navigation systems, stock trading, and command and control systems. Moreover, consider the following tasks within these environments: looking up the "800 directory," obstacle detection and avoidance, radar tracking, and recognition of objects. All of these tasks entail gathering data from the environment, processing information in the context of information obtained in the past, and contributing a timely response. Another characteristic of these examples is that they entail processing both temporal data, which lose their validity after certain time intervals, as well as historical data.

Traditional databases, hereafter referred to as databases, deal with persistent data. Transactions access these data while preserving their consistency. The goal of transaction and query processing approaches chosen in databases is to get a good throughput or response time. In contrast, real-time systems can also deal with temporal data, i.e., data that becomes outdated after a certain time. Because of the temporal character of the data and the response-time requirements forced by the environment, tasks in real-time systems have time constraints, e.g., periods or deadlines. The important difference is that the goal of real-time systems is to meet the time constraints of the tasks.

One of the most important points to remember here is that real time does not just mean fast (4). Furthermore, real time does not mean timing constraints that are in nanoseconds or microseconds. Real time means the need to manage *explicit* time constraints in a predictable fashion, that is, to use time-cognizant methods to deal with deadlines or periodicity constraints associated with tasks. Databases are useful in real-time applications because they combine several features that facilitate (*1*) the description of data, (*2*) the maintenance of correctness and integrity of the data, (*3*) efficient access to the data, and (*4*) the correct executions of query and transaction execution despite concurrency and failures (6).

Previous work on real-time databases in general has been based on simulation. However, several prototypes of general-purpose, real-time databases has been introduced. One of the first real-time database implementations was the disk-based transaction processing testbed, RT-CARAT (7). Some of the early prototype projects are the REACH (Real-time Active and Heterogeneous) mediator system project (8) and the STRIP (Stanford Real-time Information Processor) project (9).

Kim and Son (10) have presented a StarBase real-time database architecture. This architecture has been developed over a real-time microkernel operating system, and it is based on relational model. Wolfe et al. (11) have implemented a prototype of the object-oriented, real-time database architecture RTSORAC. Their architecture is based on open OODB architecture with real-time extensions. The database is implemented over a thread-based POSIX-compliant operating system. Additionally, the DeeDS project at the University of Skövde (12) and the BeeHive project at the University of Virginia (13) are examples of more recent RTDBS prototype projects.

Another object-oriented architecture is presented by Cha et al. (14). Their M2RTSS-architecture is a main-memory database system. It provides classes that implement the core functionality of storage manager, real-time transaction scheduling, and recovery. Real-Time Object-Oriented Database Architecture for Intelligent Networks (RODAIN) (15) is an architecture for a real-time, object-oriented, and fault-tolerant database management system. The RODAIN prototype system is a main-memory database, which uses priority- and criticality-based scheduling and optimistic concurrency control.

At the same time, commercial "real-time" database system products have started to appear on the marked, such as Eaglespeed-RTDB (16), Clustra (17), Timesten (18), Empress (19), eXtremeDB (20), and SolidDB (21). Although these products may not be considered true RTDBS from the viewpoint of many researchers in the RTDB community

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

since most of them only have very limited real-time features, they represent a significant step forward in the success of RTDB. Most of these products use main-memory database techniques to achieve a better real-time performance. Additionally, some of them include features for real-time transaction management.

Several research angles emerge from real-time databases: real-time concurrency control (e.g., Refs. 22–24), buffer management (e.g., Refs. 25), disk scheduling (e.g., Refs. 26 and 27), system failure and recovery (e.g., Refs. 28 and 29), overload management (e.g., Refs. 30–32), sensor data (e.g., Ref. 32, 33), security (e.g., Refs. 34–36), and distributed real-time database systems (e.g., Refs. 37–44).

While developing RTDB systems that provide the required timeliness of the data and transactions, several issues must be considered. Below is a list of some of the issues that have been the subject of research in this field (45).

- *Data, transactions, and system characteristics:* An RTDB must maintain not only the logical consistency of the data and transactions, but it must also maintain transaction timing properties as well as the temporal consistency of the data. These issues will be presented in more detail in the next section.

- *Scheduling and transaction processing:* Scheduling and transaction processing issues that consider data and transaction features have been a major part of the research that has been performed in the field of RTDB. These issues will be discussed in more detail in the third section.

- *Managing I/O and buffers:* Although the scheduling of CPU and data resources have been studied extensively, study of other resources like disk I/O and buffers has begun only recently (25, 45). Work presented in Refs. 46–48, 27, 49, and 26 has shown that transaction priorities must be considered in the buffer management and disk I/O.

- *Concurrency control*: Concurrency control has been one of the main research areas in RTDBSs. This issue will be discussed in the fourth section.

- *Distribution*: Many applications that require RTDB are not located on a single computer. Instead, they are distributed and may require that real-time data be distributed as well. Issues involved with distributing data include deadline assignment (41, 44, 33, 50), distributed database architectures (12), distributed resource management (51) data replication (43), replication consistency (52) distributed transaction processing (53, 54, 40, 38), and distributed - concurrency control (55, 51).

- *Quality of service and quality of data:* Maintaining logical consistency of the database and the temporal consistency of the data is hard (56). Therefore, there must be a trade-off made to decide which is more important (57, 58).

REAL-TIME DATABASE MODEL

A real-time system consists of a *controlling system* and a *controlled system* (3). The controlled system is the environment with which the computer and its software interacts. The controlling system interacts with its environment based on the data read from various sensors, e.g., distance and speed sensors. It is essential that the state of the environment is consistent with the actual state of the environment to a high degree of accuracy. Otherwise, the actions of the controlling systems may be disastrous. Hence, timely monitoring of the environment as well as timely processing of the information from the environment is necessary. In many cases, the read data are processed to derive new data (59).

This section discusses the characteristics of data and the characteristics of transactions in real-time database systems.

Data and Consistency

In addition to the timing constraints that originate from the need to continuously track the environment, the timing correctness requirements in a real-time database system also surface because of the need to make data available to the controlling system for its decision-making activities (60). The need to maintain consistency between the actual state of the environment and the state as reflected by the contents of the database leads to the notion of *temporal consistency*. Temporal consistency has two components (61):

- *Absolute consistency:* Data are only valid between absolute points in time. This is due to the need to keep the database consistent with the environment.

- *Relative consistency:* Different data items that are used to derive new data must be temporally consistent with each other. This requires that a set of data items used to derive a new data item form a *relative consistency set R*.

Data Item d is *temporally consistent* if and only if d is absolutely consistent and relatively consistent (3). Every data item in the real-time database consists of the current state of the object (i.e., current value stored in that data item) and two timestamps. These timestamps represent the time when this data item was last accessed by the committed transaction. These timestamps are used in the concurrency control method to ensure that the transaction reads only from committed transactions and writes after the latest committed write. Formally,

Definition 2.1. *A data item in the real-time database is denoted by d: (value, RTS, WTS, avi), where* d_{value} *denotes the current state of d;* d_{RTS} *denotes when the last committed* *transaction has read the current state of d;* d_{WTS} *denotes when the last committed transaction has written d, i.e., when the observation relating to d was made; and* d_{avi} *denotes d's absolute validity interval, i.e., the length of the time interval following* R_{WTS} *during which d is considered to have absolute validity.*

A set of data items used to derive a new data item form a relative consistency set R. Each such set R is associated with a *relative validity interval*. Assume that $d \in R$. d has a correct state if and only if (3):

1. d_{value} is logically consistent, i.e., satisfies all integrity constraints.
2. d is temporally consistent:

   - Data item $d \in R$ is absolutely consistent if and only if $(current\_time - d_{observationtime}) \leq \mathrm{d}_{absolutevalidityinterval}$.
   - Data items are relatively consistent if and only if $\forall d' \in R |d'_{timestamp} - d'_{timestamp}| \leq R_{relativevalidityinterval}$.

In this section we do not consider temporal data or temporal constraints. A good book on temporal databases can be found in Ref. 62. A discussion on integration of temporal and real-time database systems can be found from (63). Finally, temporal consistency maintenance is discussed in Refs. 64 and 65.

Transactions in Real-Time Database System

In this section, transactions are characterized along three dimensions: the manner in which data is used by transactions, the nature of time constraints, and the significance of executing a transaction by its deadline, or more precisely, the consequence of missing specified time constraints (66).

To reason about transactions and about the correctness of the management algorithms, it is necessary to define the concept formally. For the simplicity of the exposition, it is assumed that each transaction reads and writes a data item at most once. From now on, the abbreviations r, w, a, and c are used for the read, write, abort, and commit operations, respectively.

Definition 2.2. *A transaction* T_i *is partial order with an ordering relation* $\prec_i$*, where (67):*

1. $T_i \subseteq \{r_i[x], w_i[x] | x \textit{ is a data item}\} \cup \{a_i, c_i\}$;
2. $a_i \in T_i$ *if and only if* $c_i \notin T_i$;
3. *if* t *is* c_i *or* a_i*, for any other operation* $p \in T_i$, $p \prec_i t$; *and*
4. *if* $r_i[x]$, $w_i[x] \in T_i$, *then*
   either $r_i[x] \prec_i w_i[x]$ *or* $w_i[x] \prec_i r_i[x]$.

Informally, (*1*) a transaction is a subset of read, write, and abort or commit operations. (2) If the transaction executes an abort operation, then the transaction is not executing a commit operation. (3) If a certain operation t is abort or commit, then the ordering relation defines that for all other operations precede operation t in the execution of the transaction. (4) If both read and write operation are executed to the same data item, then the ordering relation defines the order between these operations.

A *real-time transaction* is a transaction with additional real-time attributes. We have added additional attributes for a *real-time transaction*. These attributes are used by the real-time scheduling algorithm and concurrency control method. Additional attributes are as follows (2):

- Timing constraints—For example, the deadline is a timing constraint associated with the transaction.
- Criticalness—It measures how critical it is that a transaction meets its timing constraints. Different transactions have different criticalness. Furthermore, criticalness is a different concept from deadline because a transaction may have a very tight deadline, but missing it may not cause great harm to the system.
- Value function—The value function is related to a transaction's criticalness. It measures how valuable it is to complete the transaction at some point in time after the transaction arrives.
- Resource requirements—Indicates the number of I/O operations to be executed, expected CPU usage, and so on.
- Expected execution time—Generally very hard to predict but can be based on an estimate or experimentally measured value of worst-case execution time.
- Data requirements—Read sets and write sets of transactions.
- Periodicity—If a transaction is periodic, what its period is.
- Time of occurrence of events—At which point in time a transaction issues a read or write request.
- Other semantics—Transaction type (read-only, write-only, etc.).

Based on the values of the above attributes, the availability of the information, and other semantics of the transactions, a real-time transaction can be characterized as follows (68):

- Implication of missing deadline: *hard, critical*, or *soft (firm)* real time.
- Arrival pattern: *periodic, sporadic*, or *aperiodic*.
- Data access pattern: predefined (*read-only, write-only*, or *update*) or *random*.
- Data requirement: *known* or *unknown*.
- Runtime requirement, i.e., pure processor or data access time: *known* or *unknown*.
- Accessed data type: *continuous, discrete*, or both.

The real-time database system applies all three types of transactions discussed in the database literature (69):

- *Write-only transactions* obtain the state of the environment and write into the database.
- *Update transactions* derive a new data item and store it in the database.
- *Read-only transactions* read data from the database and transmit that data or derived actions based on that data to the controlling system.

The above classification can be used to tailor the appropriate concurrency control methods (2). Some transaction-time constraints come from temporal consistency requirements, and some come from requirements imposed on system reaction time. The former typically take the form of periodicity requirements. Transactions can also be distinguished based on the effect of missing a transaction's deadline.

Transaction processing and concurrency control in a real-time database system should be based on priority and the criticalness of the transactions (70). Traditional methods for transaction processing and concurrency control used in a real-time environment would cause some unwanted behavior. Below the four typified problems are characterized and priority is used to denote either scheduling priority or criticality of the transaction:

- *wasted restart:* A wasted restart occurs if a higher priority transaction aborts a lower priority transaction, and later, the higher priority transaction is discarded when it misses its deadline.
- *wasted wait:* A wasted wait occurs if a lower priority transaction waits for the commit of a higher priority transaction, and later, the higher priority transaction is discarded when it misses its deadline.
- *wasted execution:* A wasted execution occurs when a lower priority transaction in the validation phase is restarted due to a conflicting higher priority transaction that has not finished yet.
- *unnecessary restart:* An unnecessary restart occurs when a transaction in the validation phase is restarted even when history would be serializable.

Traditional two-phase locking methods suffer from the problem of wasted restart and wasted wait. Optimistic methods suffer the problems of wasted execution and unnecessary restart (71).

TRANSACTION PROCESSING IN THE REAL-TIME DATABASE SYSTEM

This section presents several characteristics of transaction and query processing. Transactions and queries have time constraints attached to them, and there are different effects of not satisfying those constraints (3). A key issue in transaction processing is *predictability* (72). If a real-time transaction misses its deadline, it can have catastrophic consequences. Therefore, it is necessary to able to *predict* that such transactions will complete before their deadlines. This prediction will be possible only if the worst-case execution time of a transaction and the data and resource needs of the transaction is known.

In a database system, several sources of unpredictability exist (3):

- Dependence of the transaction's execution sequence on data values.
- Data and resource conflicts.
- Dynamic paging and I/O.
- Transactions abort and the resulting rollbacks and restarts.
- Communication delays and site failures on distributed databases.

Because a transaction's execution path can depend on the values of the data items it accessed, it may not be possible to predict the worst-case execution time of the transaction. Similarly, it is better to avoid using unbounded loops and recursive or dynamically created data structures in real-time transactions. Dynamic paging and I/O unpredictability can be solved by using main memory databases (73). Additionally, I/O unpredictability can be decreased using deadlines and priority-driven I/O controllers (e.g. 48, 46).

Transaction rollbacks also reduce predictability. Therefore, it is advisable to allow a transaction to write only to its own memory area and after the transaction is guaranteed to commit write the transaction's changes to the database (71).

Scheduling Real-Time Transactions

A *transaction scheduling policy* defines how priorities are assigned to individual transactions (66). The goal of transaction scheduling is that as many transactions as possible will meet their deadlines. Numerous transaction scheduling policies are denned in the literature. Only a few examples are quoted here.

Transactions in a real-time database can often be expressed as *tasks* in a realtime system (66). Scheduling involves the allocation of resources and time to tasks in such a way that certain performance requirements are met. A typical real-time system consists of several tasks, which must be executed concurrently. Each task has a *value*, which is gained to the system if a computation finishes in a specific time. Each task also has a *deadline*, which indicates a time limit, when a result of the computing becomes useless.

In this section, the terms *hard, soft*, and *firm* are used to categorize the transactions (66). This categorization tells the *value* imparted to the system when a transaction meets its deadline. In systems that use priority-driven scheduling algorithms, value and deadline are used to derive the priority (74, 75).

A characteristic of most real-time scheduling algorithms is the use of priority based scheduling (66). Here transactions are assigned "priorities" which are implicit or explicit functions of their deadlines or *criticality* or both. The criticality of a transaction is an indication of its level of importance. However, these two requirements sometimes conflict with each other. That is, transactions with very short deadlines might not be very critical, and vice versa (76). Therefore, the criticality of the transactions is used in place of the deadline in choosing the appropriate value to priority. This avoids the dilemma of priority scheduling, yet integrates criticality and deadline so that not only the more critical transactions meet their deadlines. The overall goal is to maximize the net worth of the executed transactions to the system.

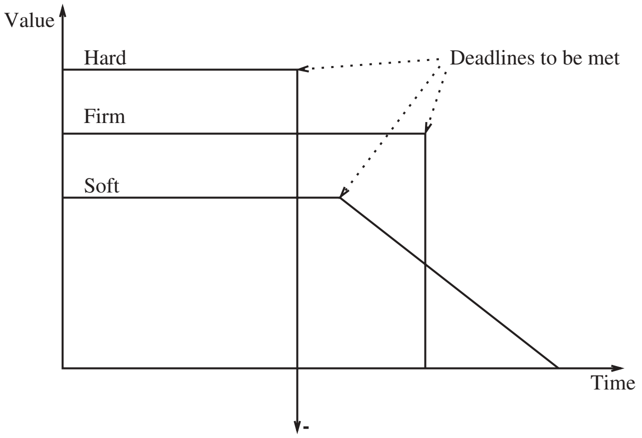


Figure 1. The deadline types.

Whereas arbitrary types of value functions can be associated with transactions (77–79), the following simple functions occur more often (see also Fig. 1):

- *Hard* deadline transactions are those that may result in a catastrophe if the deadline is missed. One can say that a large negative *value* is imparted to the system if a hard deadline is missed. These are typically safety-critical activities, such as those that respond to life or environment-threatening emergency situations (80, 81).
- *Soft* deadline transactions have some value even after their deadlines. Typically, the value drops to zero at a certain point past the deadline (82, 54).
- *Firm* deadline transactions impart no value to the system once their dead lines expire; i.e., the value drops to zero at the deadline (30, 78).

Scheduling Paradigms

Several scheduling paradigms emerge, depending on (*1*) whether a system performs a schedulability analysis; (*2*) if it does, whether it is done statically or dynamically; and (*3*) whether the result of the analysis itself produces a schedule or plan according to which tasks are dispatched at run time. Based on this, the following classes of algorithms can be identified (83):

- Static table-driven approaches: These perform a static schedulability analysis, and the resulting schedule is used at run time to decide when a task must begin execution.
- Static priority-driven preemptive approaches: These perform a static schedu lability analysis, but unlike the previous approach, no explicit schedule is constructed. At run time, tasks are executed using a highest priority first policy.
- Dynamic planning-based approaches: The feasibility is checked at run time; i.e., a dynamically arriving task is accepted for execution only if it is found feasible.
- Dynamic best effort approaches: The system tries to do its best to meet deadlines.

In the *earliest deadline first* (EDF) (80) policy, the transaction with the earliest deadline has the highest priority. Other transactions will receive their priorities in descending deadline order. In the *least slack first* (LSF) (66) policy, the transaction with the shortest slack time is executed first. The slack time is an estimate of how long the execution of a transaction can be delayed and still meet its deadline. In the *highest value* (HV) (74) policy, transaction priorities are assigned according to the transaction value attribute. A survey of transaction scheduling policies can be found in Ref. 66.

Priority Inversion

In a real-time database environment, resource control may interfere with CPU scheduling (84). When blocking is used to resolve a resource allocation such as in 2PL (85), a *priority inversion* (73) event can occur if a higher priority transaction gets blocked by a lower priority transaction.

Figure 2 illustrates an execution sequence, where a priority inversion occurs. A task T_3 executes and reserves a resource. A higher priority task T_1 preempts task T_3 and tries to allocate a resource reserved by task T_3. Then, task T_2 becomes eligible and blocks T_3. Because T_3 cannot be executed, the resource remains reserved suppressing T_1. Thus, T_1 misses its deadline due to the resource conflict.

In Ref. 86, a *priority inheritance* approach was proposed to address this problem. The basic idea of priority inheritance protocols is that when a task blocks one or more higher priority tasks, the lower priority transaction inherits the highest priority among conflicting transactions.

Figure 3 illustrates how a priority inversion problem presented in Fig. 2 can be solved with the priority inheritance protocol. Again, task T_3 executes and reserves a resource, and a higher priority task T_1 tries to allocate the same resource. In the priority inheritance protocol, task T_3 inherits the priority of T_1 and executes. Thus, task T_2 cannot preempt task T_3. When T_3 releases the resource, the priority of T_3 returns to the original level. Now T_1 can acquire the resource and complete before its deadline.

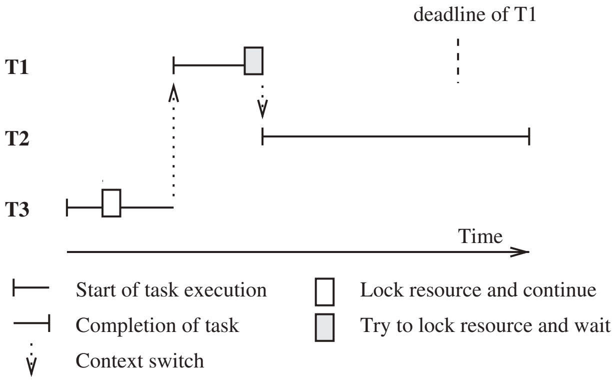


Figure 2. Priority inversion example.

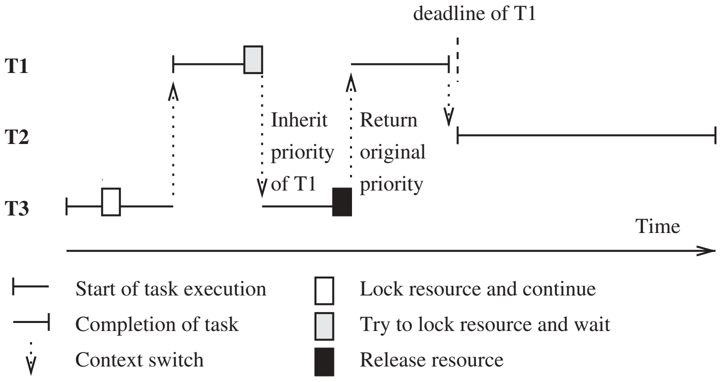


Figure 3. Priority inheritance example.

CONCURRENCY CONTROL IN REAL-TIME DATABASES

A Real-Time Database System (RTDBS) processes transactions with timing constraints such as deadlines (3). Its primary performance criterion is timeliness, not average response time or throughput. The scheduling of transactions is driven by priority order. Given these challenges, considerable research has recently been devoted to designing concurrency control methods for RTDBSs and to evaluating their performance (7, 73, 87, 88, 89, 90). Most of these methods are based on one of the two basic concurrency control mechanisms: *locking* (85) or *optimistic concurrency control* (OCC) (91).

In real-time database systems, transactions are scheduled according to their timing constraints. Task scheduler assigns a priority for a task based on its timing constraints or criticality or both (3). Therefore, higher priority transactions are executed before lower priority transactions. This is true only if a high-priority transaction has some database operation ready for execution. If no operation from a higher priority transaction is ready for execution, then an operation from a lower priority transaction is allowed to execute its database operation. Therefore, the operation of the higher priority transaction may conflict with the already executed operation of the lower priority transaction. In non-preemptive methods, a higher priority transaction must wait for the release of the resource. This is the priority inversion problem presented earlier. Therefore, data conflicts in concurrency control should also be based on transaction priorities or criticalness or both. Hence, numerous traditional concurrency control methods have been extended to the real-time database systems.

There are many ways in which the schedulers can be classified (67). One obvious classification criterion is the mode of database distribution. Some schedulers that have been proposed require a fully replicated database, whereas others can operate on partially replicated or partitioned databases. The schedulers can also be classified according to network topology. The most common classification criterion, however, is the synchronization primitive, i.e., those methods that are based on mutually exclusive access to shared data and those that attempt to order the execution of the transactions according to a set of rules (92). There are two possible views: the pessimistic view that many transactions will conflict with each other, or the optimistic view that not too many transactions will conflict with each other. Pessimistic methods synchronize the concurrent execution of transactions early in their execution, and optimistic methods delay the synchronization of transactions until their terminations (93). Therefore, the basic classification is as follows:

- Pessimistic Methods
  - Timestamp Ordering Methods (94, 95)
  - Serialization Graph Testing (67)
  - Locking Methods (19, 39, 40, 96–98)
- Optimistic Methods
  - Backward Validation Methods (99)
  - Forward Validation Methods (90, 100–103)
  - Serialization Graph Testing (67, 94, 104)
  - Hybrid Methods (105)
- Hybrid Methods (88, 89, 106–108)

In the locking-based methods, the synchronization of transactions is acquired by using physical or logical locks on some portion or granule of the database. The timestamp ordering method involves organizing the execution order of transactions so that they maintain mutual and internal consistency. This ordering is maintained by assigning timestamps to both the transactions and the data that are stored in the database (92).

The state of a conventional non-versioning database represents the state of a system at a single moment of time. Although the contents of the database change as new information is added, these changes are viewed as modification to the state. The current contents of the database may be viewed as a snapshot of the system. Additionally, conventional DBSs provide no guarantee of transaction completion times.

In the following sections, recent related work on locking and optimistic methods for real-time databases are presented.

Locking Methods in Real-Time Databases

In this section, we present some well-known pessimistic concurrency control methods. Most of these methods are based on 2PL.

2PL High Priority. In the 2PL-HP (2PL High Priority) concurrency control method (73, 109, 97), conflicts are resolved in favor of the higher priority transactions. If the priority of the lock requester is higher than the priority of the lock holder, the lock holder is aborted, rolled back, and restarted. The lock is granted to this requester, and the requester can continue its execution. If the priority of the lock requester is lower than that of the lock holder, the requesting transaction blocks to wait for the lock holder to finish and release its locks. High Priority concurrency control may lead to the cascading blocking problem, a deadlock situation, and priority inversion.

2PL Wait Promote. In 2PL-WP (2PL Wait Promote) (84, 97), the analysis of the concurrency control method is developed from Ref. 73. The mechanism presented uses shared and exclusive locks. Shared locks permit multiple concurrent readers. A new definition is made—the priority of a data object, which is defined to be the highest priority of all the transactions holding a lock on the data object. If the data object is not locked, its priority is undefined.

A transaction can join in the read group of an object if and only if its priority is higher than the maximum priority of all transactions in the write group of an object. Thus, conflicts originate from the incompatibility of locking modes as usual. Particular attention is given to conflicts that lead to priority inversions. A priority inversion occurs when a transaction of high priority requests and blocks for an object that has lesser priority. This means that all the lock holders have less priority than the requesting transaction. This same method is also called 2PL-PI (2PL Priority Inheritance) (97).

2PL Conditional Priority Inheritance. Sometimes high priority may be too strict a policy. If the lock holding transaction T_h can finish in the time that the lock requesting transaction T_r can afford to wait, that is within the slack time of T_r, and let T_h proceed to execution and T_r wait for the completion of T_h. This policy is called 2PL-CR (2PL Conditional Restart) or 2PL-CPI (2PL Conditional Priority Inheritance) (97).

Priority Ceiling Protocol (55, 86). The focus is to minimize the duration of blocking to at most one lower priority task and to prevent the formation of deadlocks. A real-time database can often be decomposed into sets of database objects that can be modeled as atomic data sets. For example, two radar stations track an aircraft representing the local view in data objects O_1 and O_2. These objects might include, e.g., the current location, velocity, and so on. Each of these objects forms an atomic data set, because the consistency constraints can be checked and validated locally. The notion of atomic data sets is especially useful for tracking multiple targets.

A simple locking method for elementary transactions is the two-phase locking method; a transaction cannot release any lock on any atomic data set unless it has obtained all the locks on that atomic data set. Once it has released its locks, it cannot obtain new locks on the same atomic data set; however, it can obtain new locks on different data sets. The theory of modular concurrency control permits an elementary transaction to hold locks across atomic data sets. It increases the duration of locking and decreases preemptibility. In this study, transactions do not hold locks across atomic data sets.

Priority Ceiling Protocol minimizes the duration of blocking to at most one elementary lower priority task and prevents the formation of deadlocks (55, 86). The idea is that when a new higher priority transaction preempts a running transaction, its priority must exceed the priorities of all preempted transactions, taking the priority inheritance protocol into consideration. If this condition cannot be met, the new transaction is suspended and the blocking transaction inherits the priority of the highest transaction it blocks.

The priority ceiling of a data object is the priority of the highest priority transaction that may lock this object (55, 86). A new transaction can preempt a lock-holding transaction if and only if its priority is higher than the priority ceilings of all the data objects locked by the lock-holding transaction. If this condition is not satisfied, the new transaction will wait and the lock-holding transaction inherits the priority of the highest transaction that it blocks. The lock-holder continues its execution, and when it releases the locks, its original priority is resumed. All blocked transactions are awakened, and the one with the highest priority will start its execution.

The fact that the priority of the new lock-requesting transaction must be higher than the priority ceiling of all the data objects that it accesses prevents the formation of a potential deadlock. The fact that the lock-requesting transaction is blocked only at most the execution time of one lower priority transaction guarantees, the formation of blocking chains is not possible (55, 86).

Read/Write Priority Ceiling. The Priority Ceiling Protocol is further advanced in Ref. 96, where the Read/Write Priority Ceiling Protocol is introduced. It contains two basic ideas. The first idea is the notion of priority inheritance. The second idea is a total priority ordering of active transactions. A transaction is said to be active if it has started but not completed its execution. Thus, a transaction can execute or wait caused by a preemption in the middle of its execution. Total priority ordering requires that each active transaction execute at a higher priority level than the active lower priority transaction, taking priority inheritance and read/write semantics into consideration.

OPTIMISTIC METHODS IN REAL-TIME DATABASES

Optimistic Concurrency Control (OCC) (99, 91), is based on the assumption that conflict is rare, and that it is more efficient to allow transactions to proceed without delays. When a transaction desires to commit, a check is performed to determine whether a conflict has occurred. Therefore, there are three phases to an optimistic concurrency control method:

- *Read phase:* The transaction reads the values of all data items it needs from the database and stores them in local variables. Concurrency control scheduler stores identity of these data items to a read set. However, writes are applied only to local copies of the data items kept in the transaction workspace. Concurrency control scheduler stores identity of all written data items to a write set.
- *Validation phase:* The validation phase ensures that all the committed transactions have executed in a serializable fashion. For a read-only transaction, this consists of checking that the data values read are still the current values for the corresponding data items. For a transaction that has writes, the validation consists of determining whether the current transaction has executed in a serializable way.
- *Write phase:* This follows the successful validation phase for transactions, including write operations. During the write phase, all changes made by the transaction are permanently stored into the database.

In the following discussion, we introduce some well-known optimistic methods for realtime database systems.

Broadcast Commit. For RTDBSs, a variant of the classic concurrency control method is needed. In Broadcast Commit, OPT-BC (87), when a transaction commits, it notifies other running transactions that conflict with it. These transactions are restarted immediately. There is no need to check a conflict with committed transactions since the committing transaction would have been restarted in the event of a conflict. Therefore, a validating transaction is always guaranteed to commit. The broadcast commit method detects the conflicts earlier than the conventional concurrency control mechanism, resulting in earlier restarts, which increases the possibility of meeting the transaction deadlines (87).

The main reason for the good performance of locking in a conventional DBMS is that the blocking-based conflict resolution policy results in conservation of resources, whereas the optimistic method with its restart-based conflict resolution policy wastes more resources (87). But in an RTDBS environment, where conflict resolution is based on transaction priorities, the OPT-BC policy effectively prevents the execution of a lower priority transaction that conflicts with a higher priority transaction, thus decreasing the possibility of further conflicts and the waste of resources is reduced. Conversely, 2PL-HP loses some of the basic 2PL blocking factor because of the partially restart-based nature of the High Priority scheme.

The delayed conflict resolution of optimistic methods aids in making better decisions since more information about the conflicting transactions is available at this stage (87). Compared with 2PL-HP, a transaction could be restarted by, or wait for, another transaction that is later discarded. Such restarts or waits are useless and cause performance degradation. OPT-BC guarantees the commit, and thus the completion of, each transaction that reaches the validation stage. Only validating transactions can cause the restart of other transactions; thus, all restarts generated by the OPT-BC method are useful.

First of all, OPT-BC has a bias against long transactions (i.e., long transactions are more likely to be aborted if there are conflicts), like in the conventional optimistic methods (87). Second, as the priority information is not used in the conflict resolution, a committing lower priority transaction can restart a higher priority transaction very close to its validation stage, which will cause missing the deadline of the restarted higher priority transaction (100).

Opt-Sacrifice. In the OPT-SACRIFICE (100) method, when a transaction reaches its validation phase, it checks for conflicts with other concurrently running transactions. If conflicts are detected and at least one of the conflicting transactions has a higher priority, then the validating transaction is restarted, i.e., sacrificed in favor of the higher priority transaction. Although this method prefers high-priority transactions, it has two potential problems. First, if a higher priority transaction causes a lower priority transaction to be restarted, but fails in meeting its deadline, the restart was useless. This degrades the performance. Second, if priority fluctuations are allowed, there may be the mutual restarts problem between a pair of transactions (i.e., both transactions are aborted). These two drawbacks are analogous to those in the 2PL-HP method (100).

Opt-Wait and Wait-X. When a transaction reaches its validation phase, it checks whether any of the concurrently running other transactions have a higher priority. In the OPT-WAIT (100) case, the validating transaction is made to wait, giving the higher priority transactions a chance to make their deadlines first. While a transaction is waiting, it is possible that it will be restarted because of the commit of one of the higher priority transactions. Note that the waiting transaction does not necessarily have to be restarted. Under the broadcast commit scheme, a validating transaction is said to conflict with another transaction, if the intersection of the write set of the validating transaction and the read set of the conflicting transaction is not empty. This result does not imply that the intersection of the write set of the conflicting transaction and the read set of the validating transaction is not empty either (100).

The WAIT-50 (100) method is an extension of the OPT-WAIT—it contains the priority wait mechanism from the OPT-WAIT method and a wait control mechanism. This mechanism monitors transaction conflict states and dynamically decides when and for how long a lower priority transaction should be made to wait for the higher priority transactions. In WAIT-50, a simple 50% rule is used—a validating transaction is made to wait while half or more of its conflict set is composed of transactions with higher priority. The aim of the wait control mechanism is to detect when the beneficial effects of waiting are outweighed by its drawbacks (100).

We can view OPT-BC, OPT-WAIT, and WAIT-50 as being special cases of a general WAIT-X method, where X is the cutoff percentage of the conflict set composed of higher priority transactions. For these methods X takes the values infinite, 0, and 50, respectively.

Validation Methods

The validation phase ensures that all the committed transactions have executed in a serializable fashion (91). Most validation methods use the following principles to ensure serializability. If a transaction Ti is before transaction T_j in the serialization graph (i.e., $T_i \prec T_j$), the following two conditions must be satisfied (71):

1. No overwriting. The writes of T_i should not overwrite the writes of T_j.
2. No read dependency. The writes of T_j should not affect the read phase of T_i.

Generally, condition 1 is automatically ensured in most optimistic methods because I/O operations in the write phase are required to be done sequentially in the critical section (71). Thus, most validation schemes consider only condition 2. During the write phase, all changes made by the transaction are permanently installed into the database. To design an efficient real-time optimistic concurrency control method, three issues have to be considered (71):

1. which validation scheme should be used to detect conflicts between transactions;
2. how to minimize the number of transaction restarts; and
3. how to select a transaction or transactions to restart when a nonserializable execution is detected.

In *Backward Validation* (99), the validating transaction is checked for conflicts against (recently) committed transactions. Conflicts are detected by comparing the read set of the validating transaction and the write sets of the committed transactions. If the validating transaction has a data conflict with any committed transactions, it will be restarted. The classic optimistic method in Ref. 91 is based on this validation process.

In *Forward Validation* (99), the validating transaction is checked for conflicts against other active transactions. Data conflicts are detected by comparing the write set of the validating transaction and the read set of the active transactions. If an active transaction has read an object that has been concurrently written by the validating transaction, the values of the object used by the transactions are not consistent. Such a data conflict can be resolved by restarting either the validating transaction or the conflicting transactions in the read phase. Optimistic methods based on this validation process are studied in Ref. 99. Most of the proposed optimistic methods are based on Forward Validation.

Forward Validation is preferable for the real-time database systems because Forward Validation provides flexibility for conflict resolution (99). Either the validating transaction or the conflicting active transactions may be chosen to restart. In addition to this flexibility, Forward Validation has the advantage of early detection and resolution of data conflicts. In recent years, the use of optimistic methods for concurrency control in real-time databases has received more and more attention. Different real-time optimistic methods have been proposed.

Forward Validation (OCC-FV) (99) is based on the assumption that the serialization order of transactions is determined by the arriving order of the transactions at the validation phase. Thus, the validating transaction, if not restarted, always precedes concurrently running active transactions in the serialization order. A validation process based on this assumption can cause restarts that are not necessary to ensure data consistency. These restarts should be avoided.

The major performance problem with optimistic concurrency control methods is the late restart (71). Sometimes the validation process using the read sets and write sets erroneously concludes that a nonserializable execution has occurred, even though it has not done so in actual execution (106) (see Example 1). Therefore, one important mechanism to improve the performance of an optimistic concurrency control method is to reduce the number of restarted transactions.

Example 1. *Consider the following transactions T_1, T_2 and history H_1:*

$$T_1\text{: } r_1[x]c_1$$
$$T_2\text{: } w_2[x]c_2$$
$$H_1\text{: } r_1[x]w_2[x]c_2$$

Based on the OCC-FV method (99), T_1 has to be restarted. However, this is not necessary, because when T_1 is allowed to commit such as:

$$H_2 : r_1[x]w_2[x]c_2c_1,$$

then the schedule of H_2 is equivalent to the serialization order $T_1 \rightarrow T_2$ as the actual write of T_2 is performed after its validation and after the read of T_1. There is no cycle in their serialization graph and H_2 is serializable.

One way to reduce the number of transaction restarts is to adjust dynamically the serialization order of the conflicting transactions (71). Such methods are called *dynamic adjustment of the serialization order* (71). When data conflicts between the validating transaction and active transactions are detected in the validation phase, there is no need to restart conflicting active transactions immediately. Instead, a serialization order of these transactions can be dynamically defined.

Definition 4.1. *Suppose there is a validating transaction T_v and a set of active transactions $T_j (j = 1, 2, \ldots, n)$. Three possible types of data conflicts can cause a serialization order between T_v and T_j* (71,110,106):

1. *$RS(T_v) \cap WS(T_j) \neq \varnothing$ (read–write conflict)*
   A read–write conflict between T_v and T_j can be resolved by adjusting the serialization order between T_v and T_j as $T_v \rightarrow T_j$ so that the read of T_v cannot be affected by T_j's write. This type of serialization adjustment is called forward ordering or forward adjustment.

2. $WS(T_v) \cap RS(T_j) \neq \varnothing$ *(write–read conflict)*
A write–read conflict between T_v and T_j can be resolved by adjusting the serialization order between T_v and T_j as $T_j \rightarrow T_v$. It means that the read phase of T_j is placed before the write of T_v. This type of serialization adjustment is called backward ordering or backward adjustment.
3. $WS(T_v) \cap WS(T_j) \neq \varnothing$ *(write–write conflict)*
A write–write conflict between T_v and T_j can be resolved by adjusting the serialization order between T_v and T_j as $T_v \rightarrow T_j$ such that the write of T_v cannot overwrite T_j's write (forward ordering).

OCC-TI. The OCC-TI (71,111) method resolves conflicts using the timestamp intervals of the transactions. Every transaction must be executed within a specific time slot. When an access conflict occurs, it is resolved using the read and write sets of the transaction together with the allocated time slot. Time slots are adjusted when a transaction commits.

OCC-DA. OCC-DA (90) is based on the Forward Validation scheme (99). The number of transaction restarts is reduced by using dynamic adjustment of the serialization order. This is supported with the use of a dynamic timestamp assignment scheme. Conflict checking is performed at the validation phase of a transaction. No adjustment of the timestamps is necessary in case of data conflicts in the read phase. In OCC-DA, the serialization order of committed transactions may be different from their commit order.

OCC-DATI. Optimistic Concurrency Control with Dynamic Adjustment using Timestamp Intervals (OCC-DATI) (112). OCC-DATI is based on forward validation. The number of transaction restarts is reduced by dynamic adjustment of the serialization order that is supported by similar timestamp intervals as in OCC-TI. Unlike the OCC-TI method, all checking is performed at the validation phase of each transaction. There is no need to check for conflicts while a transaction is still in its read phase. As the conflict resolution between the transactions in OCC-DATI is delayed until a transaction is near completion, there will be more information available for making the choice in resolving the conflict. OCC-DATI also has a new final timestamp selection method compared with OCC-TI.

SUMMARY

The field of real-time database research has evolved greatly over the relatively short time of its existence. The future of RTDB research is dependent of continues progress of this evolution. Research on this field should continue to pursue state-of-the-art applications and to apply both existing techniques to them as well as develop new ones when needed.

BIBLIOGRAPHY

1. A. Buchmann, *Real Time Database Systems*, Idea Group, 2002.
2. B. Kao and H. Garcia-Molina, An overview of real-time database systems, in S. H. Son, (ed.), *Advances in Real-Time Systems*, Englewood Cliffs, NJ: Prentice Hall, 1995, pp. 463–486.
3. K. Ramamritham, Real-time databases. *Distributed Parallel Databases*, **1**: 199–226, 1993.
4. J. A. Stankovic, S. H. Son, and J. Hansson, Misconceptions about real-time databases, *IEEE Computer*, **32** (6): 29–36, 1999.
5. D. Locke, Applications and system characteristics, in *Real-Time Database Systems - Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 2001, pp. 17–26.
6. B. Purimetla, R. M. Sivasankaran, K. Ramamritham, and J. A. Stankovic, Real-time databases: Issues and applications, in S. H. Son, (ed.), *Advances in Real-Time Systems*, Englewood Cliffs, NJ: Prentice Hall, 1996, pp. 487–507.
7. J. Huang, J. A. Stankovic, D. Towsley, and K. Ramamritham, Experimental evaluation of real-time transaction processing, *Proc of the 10th IEEE Real-Time Systems Symposium*, Santa Monica, California, USA, 1989, pp. 144–153.
8. A. P. Buchmann, H. Branding, T. Kudrass, and J. Zimmermann, Reach: a real-time, active and heterogeneous mediator system, *IEEE Data Eng. Bull.*, **15** (1–4): 44–47, 1992.
9. B. Adelberg, B. Kao, and H. Garcia-Molina, Overview of the stanford real-time information processor, *SIGMOD Record*, **25** (1): 34–37, 1996.
10. Y. K. Kim and S. H. Son, Developing a real-time database: The starbase experience, in A. Bestavros, K. Lin, and S. Son (eds.), *Real-Time Database Systems: Issues and Applications*, Boston, MA: Kluwer, 1997, pp. 305–324.
11. V. Wolfe, L. DiPippo, J. Prichard, J. Peckham, and P. Fortier, The design of real-time extensions to the open object-oriented database system, Technical report TR-94-236, University of Rhode Island, 1994.
12. S. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring, Deeds towards a distributed and active real-time database system, *SIGMOD Record*, **25** (1): 38–40, 1996.
13. John A. Stankovic, Sang H. Son, and Jörg Liebeherr, Beehive: Global multimedia database support for dependable, real-time applications, *RTDB*, 1997, pp. 409–422.
14. S. Cha, B. Park, S. Lee, S. Song, J. Park, J. Lee, S. Park, D. Hur, and G. Kim, Object-oriented design of main-memory DBMS for real-time applications, *2nd Int. Workshop on Real-Time Computing Systems and Applications*, Tokyo, Japan, 1995, pp. 109–115.
15. J. Kiviniemi, T. Niklander, P. Porkka, and K. Raatikainen, Transaction processing in the RODAIN real-time database system, in A. Bestavros and V. Fay-Wolfe, (eds.), *Real-Time Database and Information Systems*, London: Kluwer Academic Publishers, 1997, pp. 355–375.
16. Lockheed Martin Corporation, Available: http://www.lockheedmartin.com/products/eaglespeedrealtimedatabasemanager/index.html.
17. S. O. Hvasshovd, Ø. Torbjørnsen, S. E. Bratsberg, and P. Holager, The ClustRa telecom database: High availability, high throughput, and real-time response, *Proc of the 21st VLDB Conference*, San Mateo, California, 1995. pp. 469–477.
18. Oracle Corp, Available: http://www.oracle.com/database/timesten.html.
19. Empress Software Inc., Available http://www.empress.com.
20. McObject LLC, Available: http://www.mcobject.com/.

21. Solid Information Technology Ltd, Available: http://www.solidtech.com/en/products/relationaldatabasemanagement-software/embed.asp.

22. T.-W. Kuo and K.-Y. Lam, Conservative and optimistic protocols, in *Real-Time Database Systems Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 2001, pp. 29–44.

23. A. Datta, S. H. Son, and V. Kumar, Is a bird in the hand worth more than two in the bush? Limitations of priority cognizance in conflict resolution for firm real-time database systems, *IEEE Trans. Comput.*, **49** (5): 482–502, 2000.

24. J. Lee and S. H. Son, An optimistic concurrency control protocol for real-time database systems, in *3rd International Conference on Database Systems for Advanced Applications (DASFAA)*, 1993, pp. 387–394.

25. A. Datta and S. Mukherjee, Buffer management in real-time active database systems, *Real-Time Database Systems - Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 2001, pp. 77–96.

26. B. Kao and R. Cheng, Disk scheduling, in *Real-Time Database Systems-Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 2001, pp. 97–108.

27. S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley, Performance evaluation of two new disk scheduling algorithms for real-time systems, *J. Real-Time Systems*, **3** (3): 307–336, 1991.

28. M. Sivasankaram, R. and J. Ramamritham, K. an Stankovic, System failure and recovery, in *Real-Time Database Systems –Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 2001, pp. 109–124.

29. LihChjun. Shu, J. A. Stankovic, and Sang. H. Son, Achieving bounded and predictable recovery using real-time logging, *Comput. J.*, **47** (3): 373–394, 2004.

30. A. Datta, S. Mukherjee, P. Konana, I. Viguier, and A. Bajaj, Multiclass transaction scheduling and overload management in firm real-time database systems, *Information Sys.*, **21** (1): 29–54, 1996.

31. J. Hansson and S. H. Son, Overload management in RTDBS, In *Real-Time Database Systems - Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 2001, pp. 125–140.

32. T. Gustafsson and J. Hansson, Data freshness and overload handling in embedded systems, *RTCSA*, 2006, pp. 173–182.

33. K. D. Kang, S. H. Son, and J. A. Stankovic, Managing deadline miss ratio and sensor data freshness in real-time databases, *IEEE Trans. Knowl. Data Eng*, **16** (10): 1200–1216, 2004.

34. B.-S. Jeong, D. Kim, and S. Lee, Optimistic secure real-time concurrency control using multiple data version, in *Lecture Notes in Computer Science*, volume 1985, 2001.

35. H. Han, S. Park, and C. Park, A concurrency control protocol for read-only transactions in real-time secure database systems, *Proc of the 7th International Conference on Real-Time Computing Systems and Applications*, 2000, pp. 458–462.

36. S. H. Son, D. Rasikan, and B. Thuraisingham, Improving timeliness in real-time secure database systems, *SIGMOD Record*, **25** (1): 25–33, 1996.

37. J. R. Haritsa, K. Ramamritham, and R. Gupta, Real-time commit processing, in *Real-Time Database Systems-Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 2001, pp. 227–244.

38. K. Y. Lam and T. W. Kuo, Mobile distributed real-time database systems, in *Real-Time Database Systems-Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 2001, pp. 245–258.

39. K. Y. Lam, S. L. Hung, and S. H. Son, On using real-time static locking protocols for distributed real-time databases, *J. Real-Time Sys.*, **13** (2): 141–166, 1997.

40. K. Y. Lam and S. L. Hung, Concurrency control for time-constrained transactions in distributed databases systems, *Comput. J.*, **38** (9): 704–716, 1995.

41. V. C. S. Lee, K.-Y. Lam, and S.-L. Hung, Virtual deadline assignment in distributed real-time database systems, *Second International Workshop on Real-Time Computing Systems and Applications*, 1995.

42. U. Halici and A. Dogac, An optimistic locking technique for concurrency control in distributed databases, *IEEE Trans. Software Eng.*, **17** (7): 712–724, 1991.

43. K.-J. Lin and M.-J. Lin, Enhancing availability in distributed real-time databases, *ACM SIGMOD Record*, **17** (1): 34–43, 1988.

44. Yuan. Wei, Sang. H. Son, and John. A. Stankovic, Maintaining data freshness in distributed real-time databases, in *ECRTS*, 2004, pp. 251–260.

45. Krithi. Ramamritham, S. H. Son, and Lisa. Cingiser. DiPippo, Real-time databases and data services, *Real-Time Sys.*, **28** (2–3): 179–215, 2004.

46. B. Sprunt, D. Kirj, and L. Sha, Priority-driven, preemptive i/o controllers for real-time systems, *Proc of the International Symposium on Computer Architecture*, Honolulu, Hawaii, USA, 1988, pp. 152–159.

47. M. J. Carey, R. Jauhari, and M. Livny, Priority in DBMS resource scheduling, *VLDB*, 1989, pp. 397–410.

48. R. Abbott and H. Garcia-Molina, Scheduling I/O requests with deadlines: A performance evaluation, *Proc of the 11th IEEE Real-Time Systems Symposium*, 1990, pp. 113–124.

49. W. Kim and J. Srivastava, Enhancing real-time DBMS performance with multiversion data and priority based disk scheduling, *Proc of the 12th IEEE Real-Time Systems Symposium*, Los Alamitos, California, 1991, pp. 222–231.

50. Ming. Xiong and Krithi. Ramamritham, Deriving deadlines and periods for real-time update transactions, *IEEE Trans. Computers*, **53** (5): 567–583, 2004.

51. T. He, J. A. Stankovic, M. Marley, C. Lu, Y. Lu, T. F. Abdelzaher, S. H. Son, and G. Tao, Feedback control-based dynamic resource management in distributed real-time systems, *J. Sys. Software*, **80** (7): 997–1004, 2007.

52. Ming. Xiong, Krithi. Ramamritham, Jonh. R. Haritsa, and Jayant. A. Stankovic, Mirror: a state-conscious concurrency control protocol for replicated real-time databases, *Inf. Syst.*, **27** (4): 277–297, 2002.

53. Ö. Ulusoy and G. G. Belford. A simulation model for distributed real-time database systems, *Proc of the 25th Annual Simulation Symposium*, Los Alamitos, Calif., 1992, pp. 232–240.

54. B. Kao and H. Garcia-Molina, Deadline assigment in a distributed soft real-time system, *Proc of the 13th International Conference on Distributed Computing Systems*, Pittsburgh, PA, USA, 1993, pp. 428–437.

55. L. Sha, R. Rajkumar, and J. P. Lehoczky, Concurrency control for distributed real-time databases, *ACM SIGMOD Record*, **17** (1): 82–98, 1988.

56. A. K. Jha, M. Xiong, and K. Ramamritham, Mutual consistency in real-time databases, *RTSS*, 2006, pp. 335–343.

57. M. Amirijoo, J. Hansson, and S. H. Son, Specification and management of qos in real-time databases supporting

imprecise computations, *IEEE Trans. Comput.*, **55** (3): 304–319, 2006.

58. Y. Wei, V. Prasad, S. H. Son, and J. A. Stankovic, Prediction-based qos management for real-time data streams, in *RTSS*, 2006, pp. 344–358.
59. M. H. Graham, Issues in real-time data management, *J. Real-Time Sys.*, **4**: 185–202, 1992.
60. M. H. Graham, How to get serializability for real-time transactions without having to pay for it, *Proc of the 14th IEEE Real-time Systems Symposium*, 1993, pp. 56–65.
61. R. Snodgrass and I. Ahn, Temporal databases, *IEEE Comput.*, **19** (9): 35–42, 1986.
62. A. Tansel, J. Clifford, S. Jojodia, A. Segev, and R. Snodgrass, *Temporal Databases: Theory, Design, and Implementation*, Redwood City, CA: Benjamin/Cummings, 1994.
63. R. Ramamritham, R. M. Sivasankaran, J. A. Stankovic, D. F. Towsley, and M. Xiong, Integrating temporal, real-time, and active databases, *SIGMOD Record*, **25** (1): 8–12, 1996.
64. M. Xiong, J. A. Stankovic, R. Ramamritham, D. F. Towsley, and R. M. Sivasankaran, Maintaining temporal consistency: Issues and algorithms, In *RTDB*, 1996, pp. 1–6.
65. Ming. Xiong, Krithi. Ramamritham, John. A. Stankovic, D. F. Towsley, and R. M. Sivasankaran, Scheduling transactions with temporal constraints: Exploiting data semantics, *IEEE Trans. Knowl. Data Eng.*, **14** (5): 1155–1166, 2002.
66. R. Abbott and H. Garcia-Molina, Scheduling real-time transactions, *ACM SIGMOD Record*, **17** (1): 71–81, 1988.
67. P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Reading, MA: Addison-Wesley, 1987.
68. Y.-K. Kim and S. H. Son, Predictability and consistency in real-time database systems, in *Real-Time Database Systems-Architecture and Techniques*, Norwell, MA: Kluwer Academic Publishers, 1995, pp. 509–531.
69. I. Ahn, Database issues in telecommunications network management, *ACM SIGMOD Record*, **23** (2): 37–43, 1994.
70. J. A. Stankovic and W. Zhao, On real-time transactions, *ACM SIGMOD Record*, **17** (1): 4–18, 1988.
71. J. Lee and S. H. Son, Using dynamic adjustment of serialization order for real-time database systems, *Proc of the 14th IEEE Real-Time Systems Symposium*, Raleigh-Durham, NC, USA, 1993, pp. 66–75.
72. J. Stankovic and K. Ramamritham, What is predictability for real-time systems? *J. Real-Time Sys.*, **2**: 247–254, 1990.
73. R. Abbott and H. Garcia-Molina, Scheduling real-time transactions: A performance evaluation, *Proc of the 14th VLDB Conference*, Los Angeles, California, 1988, pp. 1–12.
74. J. Haritsa, M. Carey, and M. Livny, Value-based scheduling in real-time database systems, Tech. Rep. CS-TR-91–1024, Madison, WI: University of Winconsin, 1991.
75. S.-M. Tseng, Y. H. Chin, and W.-P. Yang, Scheduling real-time transactions with dynamic values: A performance evaluation, *Proc of the Second International Workshop on Real-Time Computing Systems and Applications*, Tokio, Japan, 1995.
76. S. R. Biyabani, J. A. Stankovic, and K. Ramamritham, The integration of deadline and criticalness in hard real-time scheduling, *Proc of the 8th IEEE Real-Time Systems Symposium*, Huntsville, Alabama, 1988, pp. 487–507.
77. A. P. Buchmann, D. R. McCarthy, M. Hsu, and U. Dayal, Time-critical database scheduling: A framework for integrating real-time scheduling and concurrency control, *Proc of the 5th International Conference on Data Engineering*, Los Angeles, California, USA, 1989, pp. 470–480.
78. J. R. Haritsa, M. J. Carey, and M. Livny, Data access scheduling in firm real-time database systems, *J. Real-Time Systems*, **4** (2): 203–241, 1992.
79. E. D. Jensen, C. D. Locke, and H. Tokuda, A time-driven scheduling model for real-time systems, *Proc of the 5th IEEE Real-Time Systems Symposium*, San Diego, California, USA, 1985, pp. 112–122.
80. C. L. Liu and J. W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, *J. ACM*, **20** (1): 46–61, 1973.
81. K. W. Lam, V. Lee, S. L. Hung, and K. Y. Lam, An augmented priority ceiling protocol for hard real-time systems, *J. Computing Information, Special Issue: Proc of Eighth International Conference Comput. Information*, **2** (1): 849–866, 1996.
82. J. Huang, J. A. Stankovic, K. Ramamritham, D. Towsley, and B. Purimetla, Priority inheritance in soft real-time databases, *J. Real-Time Sys.*, **4** (2): 243–268, 1992.
83. K. Ramamritham and J. Stankovic, Scheduling algorithms and operating systems support for real-time systems, *Proc IEEE*, **82** (1): 55–67, 1994.
84. R. Abbott and H. Garcia-Molina, Scheduling real-time transactions with disk resident data, *Proc of the 15th VLDB Conference*, Amsterdam, 1989, pp. 385–396.
85. K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, The notions of consistency and predicate locks in a database system, *Communications ACM*, **19** (11): 624–633, 1976.
86. L. Sha, R. Rajkumar, and J. P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, *IEEE Trans. on Comput.*, **39** (9): 1175–1185, 1990.
87. J. R. Haritsa, M. J. Carey, and M. Livny, On being optimistic about real-time constraints, *Proc of the 9th ACM Symposium on Principles of Database Systems*, 1990, pp. 331–343.
88. J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley, Experimental evaluation of real-time optimistic concurrency control schemes, *Proc of the 17th VLDB Conference*, Barcelona, Catalonia, Spain, September 1991, pp. 35–46.
89. K. W. Lam, S. H. Son, and S. Hung, A priority ceiling protocol with dynamic adjustment of serialization order, *Preceeding of the 13th IEEE Conference on Data Engineering*, Birmingham, UK, 1997.
90. K. W. Lam, K. Y. Lam, and S. Hung, An efficient real-time optimistic concurrency control protocol, *Proc of the First International Workshop on Active and Real-Time Database Systems*, New York: Springer, 1995, pp. 209–225.
91. H. T. Kung and J. T. Robinson, On optimistic methods for concurrency control, *ACM Trans. Database Sys.*, **6** (2): 213–226, 1981.
92. M. T. Özsu and P. Valduriez, *Principles of Distributed Database System*, Englewood Cliffs, NJ: Prentice Hall, 1999.
93. G. Weikum and G. Vossen, *Transactional Information Systems: Theory, algorithms, and the practice of concurrency control and recovery*, San Mateo, CA: Morgan Kaufmann, 2002.
94. K. Marzullo, Concurrency control for transactions with priorities, tech. report TR 89-996, Cornell University, Ithaca, NY, 1989.
95. Ö. Ulusoy and G. Belford, Real-time transaction scheduling in database systems, *Information Sys.*, **18** (6): 559–580, 1993.
96. L. Sha, R. Rajkumar, S. H. Son, and C.-H. Chang, A real-time locking protocol, *IEEE Trans. Comput.*, **40** (7): 793–800, 1991.

97. J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley, On using priority inheritance in real-time databases, *Proc of the 12th IEEE Real-Time Systems Symposium*, San Antonio, Texas, USA, 1991, pp. 210–221.

98. D. Agrawal, A. E. Abbadi, and R. Jeffers, Using delayed commitment in locking protocols for real-time databases, *Proc of the 1992 ACM SIGMOD International Conference on Management of Data*, 1992, pp. 104–113.

99. T. Härder, Observations on optimistic concurrency control schemes, *Information Sys.*, **9** (2): 111–120, 1984.

100. J. R. Haritsa, M. J. Carey, and M. Livny, Dynamic real-time optimistic concurrency control, *Proc of the 11th IEEE Real-Time Systems Symposium*, 1990, pp. 94–103.

101. K. W. Lam, K. Y. Lam, and S. Hung, Real-time optimistic concurrency control protocol with dynamic adjustment of serialization order, *Proc of the IEEE Real-Time Technology and Application Symposium*, 1995, pp. 174–179.

102. I. Yoon and S. Park, Enhancement of alternative version concurrency control using dynamic adjustment of serialization order, *Proc of the Second International Workshop on Real-Time Databases: Issues and Applications*, Burlington, Vermont, USA, 1997.

103. U. Lee and B. Hwang, Optimistic concurrency control based on timestamp interval for broadcast environment, *Proc. Advances in Databases and Information Systems, 6th East European Conference, ADBIS 2002, Bratislava, Slovakia*, Vol. 2435 of *Lecture Notes in Computer Science*, 2002.

104. V. C. S. Lee and K.-W. Lam, Conflict free transaction scheduling using serialization graph for real-time databases, *J. Sys. Software*, **55** (1): 57–65, 2000.

105. V. C. S. Lee and K.-W. Lam, Optimistic concurrency control in broadcast environments: Looking forward at the server and backward at the clients, in H. V. Leong, W.-C. Lee, B. Li, and L. Yin, (eds.), *First International Conference on Mobile Data Access,* Lecture Notes in Computer Science, **1748,** Verlag: Springer, 1999, pp. 97–106.

106. S. H. Son, J. Lee, and Y. Lin, Hybrid protocols using dynamic adjustment of serialization order for real-time concurrency control, *J. Real-Time Sys.*, **4** (2): 269–276, 1992.

107. P. S. Yu and D. M. Dias, Analysis of hybrid concurrency control for a high data contention environment, *IEEE Trans. on Software Eng.*, **SE-18** (2): 118–129, 1992.

108. P. Graham and K. Barker, Effective optimistic concurrency control in multiversion object bases, *Lecture Notes Computer Science*, **858**: 313–323, 1994.

109. R. Abbott and H. Garcia-Molina, Scheduling real-time transactions: A performance evaluation, *ACM Transactions Database Sys.*, **17** (3): 513–560, 1992.

110. Y. Lin and S. H. Son, Concurrency control in real-time databases by dynamic adjustment of serialization order, *Proc of the 11th IEEE Real-Time Systems Symposium*, Los Alamitos, California, 1990, pp. 104–112.

111. J. Lee, *Concurrency Control Algorithms for Real-Time Database Systems*. PhD Thesis, Charolotte suille, VA: University of Virginia, 1994.

112. J. Lindström and K. Raatikainen, Dynamic adjustment of serialization order using timestamp intervals in real-time databases, *Proc of the 6th International Conference on Real-Time Computing Systems and Applications*, 1999, pp. 13–20.

Jan Lindström
IBM Corporation
Helsinki, Finland

RELATIONAL DATABASES

To manage a large amount of persistent data with computers requires storing and retrieving these data in files. However, it was found in the early 1960s that files are not sufficient for the design and use of more and more sophisticated applications. As a consequence, database systems have become a very important tool for many applications over the past 30 years. Database management systems (DBMSs) aim to provide users with an efficient tool for good modeling and for easy and efficient manipulation of data. It is important to note that concurrency control, data confidentiality, and recovery from failure also are important services that a DBMS should offer. The very first DBMSs, known as hierarchical and then as network systems, were based on a hierarchical and then network-like conceptual data organization, which actually reflects the physical organization of the underlying files. Thus, these systems do not distinguish clearly between the physical and the conceptual levels of data organization. Therefore, these systems, although efficient, have some important drawbacks, among which we mention data redundancies (which should be avoided) and a procedural way of data manipulation, which is considered not easy enough to use.

The relational model, proposed by Codd in 1970 (1), avoids the drawbacks mentioned above by distinguishing explicitly between the physical and the conceptual levels of data organization. This basic property of the relational model is a consequence of the fact that, in this model, users see the data as tables and do not have to be aware of how these tables are stored physically. The tables of a relational database are accessed and manipulated as a whole, contrary to languages based on hierarchical or network models, according to which data are manipulated on a record-by-record basis. As a consequence, data manipulation languages for relational databases are set-oriented, and so, they fall into the category of declarative languages, in which there is no need of control structures, such as conditional or iterative statements. On the other hand, because relations are a well-known mathematical concept, the relational model stimulated a lot of theoretical research, which led to successful implementations. As an example of a relational database, Fig. 1 shows the two tables, called EMP and DEPT, of a sample database for a business application.

The main results obtained so far are summarized as follows:

1. The expressional power of relational data manipulation languages is almost that of first-order logic without function symbols. Moreover, relational languages have large capabilities of optimization. This point is of particular importance, because it guarantees that data are efficiently retrieved, independently of the way the query is issued by the user.
2. Integrity constraints, whose role is to account for properties of data, are considered within the model. The most important and familiar are the functional dependencies. Research on this topic led to theoretical criteria for what is meant by a “good” conceptual data organization for a given application.
3. A theory of concurrency control and transaction management has been proposed to account for the dynamic aspects of data manipulation with integrity constraints. Research in this area led to actual methods and algorithms that guarantee that, in the presence of multiple updates in a multiuser environment, the modified database still satisfies the integrity constraints imposed on it.

These fundamental aspects led to actual relational systems that rapidly acquired their position in the software market and still continue to do so today. Relational DBMSs are currently the key piece of software in most business applications running on various types of computers, ranging from mainframe systems to personal computers (PCs). Among the relational systems available on the marketplace, we mention DB2 (IBM), INGRES (developed at the University of California, Berkeley), ORACLE (Oracle Corp.), and SQLServer (Microsoft Corp.), all of which implement the relational model of databases together with tools for developing applications.

In the remainder of this article, we focus on the theory of the relational model and on basic aspects of dependency theory. Then, we deal with problems related to updates and transaction management, and we briefly describe the structure of relational systems and the associated reference language called SQL. We conclude with a brief discussion on several extensions of the relational model.

THEORETICAL BACKGROUND OF RELATIONAL DATABASES

The theory of the relational model of databases is based on relations. Although relations are well known in mathematics, their use in the field of databases requires definitions that slightly differ from those used in mathematics. Based on these definitions, basic operations on relations constitute the relational algebra, which is related closely to first-order logic. Indeed, relational algebra has the same expressional power as a first-order logic language, called relational calculus, and this relationship constitutes the basis of the definition of actual data manipulation

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

EMP	*empno*	*ename*	*sal*	*deptno*
	123	john	23,000	1
	234	julia	50,000	1
	345	peter	7,500	2
	456	laura	12,000	2
	567	paul	8,000	1

DEPT	*deptno*	*dname*	*mgr*
	1	sales	234
	2	staff	345

Figure 1. A sample relational database D.

languages, among which the language called SQL is now the reference.

Basic Definitions and Notations

The formal definition of relational databases starts with a finite set, called the *universe*, whose elements are called *attributes*. If U denotes a universe, each attribute A of U is associated with a nonempty and possibly infinite set of values (or constants), called the *domain of* A and denoted by *dom*(A). Every nonempty subset of U is called a *relation scheme* and is denoted by the juxtaposition of its elements. For example, in the database of Fig. 1, the universe U contains the attributes *empno, ename, sal, deptno, dname*, and *mgr* standing, respectively, for numbers of employees, names of employees, salaries of employees, numbers of departments, names of departments and numbers of managers. Moreover, we consider here that *empno*, *deptno*, and *mgr* have the same domain, namely the set of all positive integers, whereas the domain of the attributes *ename* and *dname* is the set of strings of alphabetic characters of length at most 10.

Given a relation scheme R′, a *tuple t over* R is a mapping from R to the union of the domains of the attributes in R, so that, for every attribute A in R, *t*(A) is an element of *dom*(A). Moreover, if R′ is a nonempty subset of R the *restriction of t to* R, being the restriction of a mapping, is also a tuple, denoted by *t*.R′. As a notational convenience, tuples are denoted by the juxtaposition of their values, assuming that the order in which values are written corresponds to the order in which attributes in R are considered.

Given a universe U and a relation scheme R, a *relation over* R is a *finite* set of tuples over R, and a *database over* U is a set of relations over relations schemes obtained from U.

Relational Algebra

From a theoretical point of view, querying a database consists of computing a relation (which in practice is displayed as the answer to the query) based on the relations in the database. The relation to be computed can be expressed in two different languages: relational algebra, which explicitly manipulates relations, and relational calculus, which is based on first-order logic. Roughly speaking, relational calculus is the *declarative* counterpart of relational algebra, which is observed as a *procedural* language.

The six fundamental operations of the relational algebra are union, difference, projection, selection, join, and renaming (note that replacing the join operation by the Cartesian product is another popular choice discussed in Refs. (2) and (3)). The formal definitions of these operations are as follows: Let *r* and *s* be two relations over relation schemes R and S, respectively. Then

1. *Union*. If R = S, then $r \cup s$ is a relation defined over R, such that $r \cup s = \{t | t \in r \text{ or } t \in s\}$. Otherwise $r \cup s$ is undefined.
2. *Difference*. If R = S, then $r - s$ is a relation defined over R, such that $r - s = \{t | t \in r \text{ and } t \notin s\}$. Otherwise $r - s$ is undefined.
3. *Projection*. Let Y be a relation scheme. If $\mathrm{Y} \subseteq \mathrm{R}$, then $\pi_{\mathrm{Y}}(r)$ is a relation defined over Y, such that $\pi_{\mathrm{Y}}(r) = \{t | \exists u \in r \text{ such that } u.\mathrm{Y} = t\}$. Otherwise $\pi_{\mathrm{Y}}(r)$ is undefined.
4. *Selection* of *r* with respect to a condition *C*: $\sigma_c(r)$ is a relation defined over R, such that $\sigma_c(r) = \{t | t \in r \text{ and } t \text{ satisfies } C\}$. Selection conditions are either atomic conditions or conditions obtained by combination of atomic conditions, using the logical connectives $\vee$ (or), $\wedge$ (and), or $\neg$ (not). An atomic condition is an expression of the form $\mathrm{A}\Theta\mathrm{A}'$ or $\mathrm{A}\Theta a$ where A and A′ are attributes in R whose domains are "compatible" [i.e., it makes sense to compare a value in *dom*(A) with a value in *dom*(A′)], *a* is a constant in *dom*(A), and Θis an operator of comparison, such as $<$, $>$, $\leq$, $\geq$, or $=$.
5. *Join*. $r \bowtie s$ is a relation defined over $R \cup S$, such that $r \bowtie s = \{t | t.\mathrm{R} \in r \text{ and } t.\mathrm{S} \in s\}$.
6. *Renaming*. If A is an attribute in R and B is an attribute not in R, such that *dom*(A) = *dom*(B), then $\rho_{\mathrm{B} \leftarrow \mathrm{A}}(r)$ is a relation defined over $(\mathrm{R} - \{\mathrm{A}\}) \cup \{\mathrm{B}\}$ whose tuples are the same as those in *r*.

For example, in the database of Fig. 1, the following expression computes the numbers and names of all departments having an employee whose salary is less than \$10,000:

$$\mathrm{E} : \pi_{\mathrm{deptno\ dname}}[\sigma_{\mathrm{sal} < 10{,}000}(\mathrm{EMP} \bowtie \mathrm{DEPT})]$$

Figure 2 shows the steps for evaluating this expression against the database of Fig. 1. As an example of using renaming, the following expression computes the numbers of employees working in at least two different departments:

$$\mathrm{E}_1 : \pi_{\mathrm{empno}}[\sigma_{\mathrm{deptno} \neq \mathrm{dnumber}}(\mathrm{EMP}) \bowtie \rho_{\mathrm{dnumber} \leftarrow \mathrm{deptno}} [\pi_{\mathrm{deptno\ empno}}(\mathrm{EMP})]]$$

The operations introduced previously enjoy properties, such as commutativity, associativity, and distributivity (see Ref. (3) for full details). The properties of the relational operators allow for syntactic transformations according to which the same result is obtained, but through a more efficient computation. For instance, instead of evaluating

(a) EMP $\bowtie$ DEPT

empno	*ename*	*sal*	*deptno*	*dname*	*mgr*
123	john	23,000	1	sales	234
234	julia	50,000	1	sales	234
345	peter	7,500	2	staff	345
456	laura	12,000	2	staff	345
578	paul	8,000	1	sales	234

(b) $\sigma_{\text{sal}<10{,}000}$ (EMP $\bowtie$ DEPT)

empno	*ename*	*sal*	*deptno*	*dname*	*mgr*
345	peter	7,500	2	staff	345
578	paul	8,000	1	sales	234

(c) $\pi_{\text{deptno dname}}[\sigma_{\text{sal}<10{,}000}(\text{EMP} \bowtie \text{DEPT})]$

deptno	*dname*
2	staff
1	sales

Figure 2. The intermediate relations in the computation of expression **E** applied to the database D of Fig. 1. (a) The computation of the join, (b) the computation of the selection, and (c) the computation of the projection.

the previous expression **E**, it is more efficient to consider the following expression:

$$\text{E}' : \pi_{\text{deptno dname}}[\sigma_{\text{sal} < 10{,}000}(\text{EMP}) \bowtie \pi_{\text{deptno dname}}(\text{DEPT})]$$

Indeed, the intermediate relations computed for this expression are "smaller" than those of Fig. 2 in the number of rows and the number of columns.

Such a transformation is known as query optimization. To optimize an expression of the relational algebra, the expression is represented as a tree in which the internal nodes are labeled by operators and the leaves are labeled by the names of the relations of the database. Optimizing an expression consists of applying properties of relational operators to transform the associated tree into another tree for which the evaluation is more efficient. For instance, one of the most frequent transformations consists of pushing down selections in the tree to reduce the number of rows of intermediate relations. We refer to Ref. (2) for a complete discussion of query optimization techniques. Although efficient in practice, query optimization techniques are not optimal, because, as Kanellakis notices in Ref. (4), the problem of deciding whether two expressions of the relational algebra always yield the same result is impossible to solve.

Relational Calculus

The existence of different ways to express a given query in the relational algebra stresses the fact that it can be seen as a procedural language. Fortunately, relational algebra has a declarative counterpart, namely the relational calculus. This result comes from the observation that, if r is a relation defined over a relation scheme R containing n distinct attributes, then membership of a given tuple t in r is equivalently expressed by first-order formalism if we regard r as an n-ary predicate, and t as an n-ary vector of constants, and if we state that the atomic formula $r(t)$ is true. More formally, the correspondence between relational algebra and calculus is as follows: Given a database $D = \{r_1, r_2, \ldots, r_n\}$ over a universe U and with schema $\{R_1, R_2, \ldots, R_n\}$, we consider a first-order alphabet with the usual connectives $(\wedge, \vee, \neg)$ and quantifiers $(\exists, \forall)$ where

1. the set of constant symbols is the union of all domains of the attributes in U;
2. the set of predicate symbols is $\{r_1, r_2, \ldots, r_n\}$, where each r_i is a predicate symbol whose arity is the cardinality of R_i; and
3. the variable symbols may range over tuples, in which case, the language is called *tuple* calculus, or over domain elements, in which case, the language is called *domain* calculus.

One should notice that no function symbols are considered in relational calculus. Based on such an alphabet, formulas of interest are built up as usual in logic, but with some syntactic restrictions explained later. Now we recall that without loss of generality, a well-formed formula has the form $\psi = (Q_1)(Q_2)\ldots(Q_k)[\varphi(x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_1)]$, where $x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_1$ are the only variable symbols occurring in φ, where (Q_i) stands for $(\exists x_i)$ or $(\forall x_i)$, and where φ is a quantifier-free formula built up from connectives and atomic formulas (atomic formulas have the form $r(t_1, t_2, \ldots, t_n)$, where r is an n-ary predicate symbol and t_j is either a variable or a constant symbol). Moreover, in the formula ψ, the variables x_i are bound (or quantified) and the variables y_j are free (or not quantified). See Ref. (5) for full details on this topic.

In the formalism of tuple calculus, the relational expression **E** is written as

$$\begin{aligned}\{z \mid (\exists x)(\exists y)(\text{EMP}(x) \;&\wedge\; \text{DEPT}(y) \wedge y.deptno = z.deptno\\ &\wedge\; y.dname = z.dname\\ &\wedge\; x.deptno = y.deptno \wedge x.sal\\ &\quad < 10{,}000)\}\end{aligned}$$

One should note that, in this formula, variables stand for tuples, whose components are denoted as restrictions in the relational algebra. Considering domain calculus, the previous formula is written as follows:

$$\{(z_1,z_2)|(\exists x_1)(\exists x_2)(\exists x_3)(\exists x_4)(\exists y_1)(\exists y_2)(\exists y_3)\\ (\mathbf{EMP}(x_1,x_2,x_3,x_4)\wedge \mathbf{DEPT}(y_1,y_2,y_3)\wedge z_1\\ = y_1 \wedge z_2 = y_2 \wedge x_4 = y_1 \wedge x_1 < \mathbf{10,000})\}$$

The satisfaction of a formula ψ in a database D is defined in a standard way, as in first-order logic. In the context of databases, however, some well-formed formulas must be discarded because relations are assumed to be finite and, thus, so must be the set of tuples satisfying a given formula in a database. For instance, the domain calculus formula $(\exists x)[\neg r(x,y)]$ must be discarded, because in any database, the set of constants a satisfying the formula $\neg r(r_0,a)$ for some appropriate x_0 may be infinite (remember that domains may be infinite). The notion of safeness is based on what is called the domain of a formula ψ, denoted by DOM(ψ). DOM(ψ) is defined as the set of all constant symbols occurring in ψ, together with all constant symbols of tuples in relations occurring in ψ as predicate symbols. Hence, DOM(ψ) is a finite set of constants and ψ is called *safe* if all tuples satisfying it in D contain only constants of DOM(ψ). To illustrate the notion of safeness, again consider the formula $\psi = (\exists x)[\neg r(x,y)]$. Here DOM(ψ) = {α|α occurs in a tuple of *r*}, and so, ψ may be satisfied in D by values β not in DOM(ψ). Therefore, ψ is a nonsafe formula. On the other hand, the formula $\psi' = (\exists x)[\neg r(x,y) \wedge s(x,y)]$ is safe, because every β satisfying ψ′ in D occurs in DOM(ψ′).

It is important to note that tuple and domain calculus are equivalent languages that have resulted in the emergence of actual languages for relational systems. A formal proof of the equivalence between relational calculus and relational algebra was given by Codd in Ref. (6).

DATA DEPENDENCIES

The theory of data dependencies has been motivated by problems of particular practical importance, because in all applications, data stored in a database must be restricted so as to satisfy some required properties or constraints. For instance, in the database of Fig. 1, two such properties could be (*1*) two departments with distinct names cannot have the same number and (*2*) a department has only one manager, so that the relation DEPT cannot contain two distinct tuples with the same *deptno* value. Investigations on constraints in databases have been carried out in the context of the relational model to provide sound methods for the design of database schemas. The impact of constraints on schema design is exemplified through properties (*1*) and (*2*). Indeed, assume that the database consists of only one relation defined over the full universe. Then clearly, information about a given department is stored as many times as the number of its employees, which is redundant. This problem has been solved by the introducing normal forms in the case of particular dependencies called functional dependencies. On the other hand, another problem that arises in the context of our example is the following: Assuming that a database D satisfies the constraints (*1*) and (*2*), does D satisfy other constraints? Clearly, this problem, called the implication problem, has to be solved to make sure that all constraints are considered at the design phase just mentioned. Again, the implication problem has been solved in the context of functional dependencies. In what follows, we focus on functional dependencies, and then, we outline other kinds of dependencies that have also been the subject of research.

The Theory of Functional Dependencies

Let *r* be a relation over a relation scheme R, and let X and Y be two subschemes of R. The functional dependency from X to Y, denoted by X → Y, is satisfied by *r* if, for all tuples *t* and *t*′ in *r*, the following holds: $t.\mathrm{X} = t'.\mathrm{X} \Rightarrow t.\mathrm{Y} = t'.\mathrm{Y}$. Then, given a set F of functional dependencies and a dependency X → Y, F implies X → Y if every relation satisfying the dependencies in F also satisfies the dependency X → Y. For instance, for R = ABC and F = {A → B, AB → C}, F implies A → C. However, this definition of the implication of functional dependencies is not effective from a computational point of view. An axiomatization of this problem, proposed in Ref. (7), consists of the following rules, where X, Y, and Z are relation schemes:

1. $\mathrm{Y} \subseteq \mathrm{X} \Rightarrow \mathrm{X} \rightarrow \mathrm{Y}$
2. $\mathrm{X} \rightarrow \mathrm{Y} \Rightarrow \mathrm{XZ} \rightarrow \mathrm{YZ}$
3. $\mathrm{X} \rightarrow \mathrm{Y}, \mathrm{Y} \rightarrow \mathrm{Z} \Rightarrow \mathrm{X} \rightarrow \mathrm{Z}$

A derivation using these axioms is defined as follows: F derives X → Y if either X → Y is in F or X → Y can be generated from F using repeatedly the axioms above. Then, the soundness and completeness of these axioms is expressed as follows: F implies X → Y if and only if F derives X → Y, thus providing an effective way for solving the implication problem in this case.

An important aspect of functional dependencies is that they allow for the definition of normal forms that characterize suitable database schemas. Normal forms are based on the notion of key defined as follows: If R is a relation scheme with functional dependencies F, then K is a key of (R, F) if K is a minimal relation scheme with respect to set inclusion such that F implies (or derives) K → R. Four normal forms can be defined, among which we mention here only three of them:

1. The first normal form (1NF) stipulates that attributes are atomic in the relational model, which is implicit in the definitions of relational databases but restricts the range of applications that can be taken easily into account. It explains, in particular, the emergence of object-oriented models of databases.
2. The third normal form (3NF) stipulates that attributes participating in no keys depend fully and exclusively on keys. The formal definition is as follows: (R, F) is in 3NF if, for every derived dependency X → A from F, such that A is an attribute not in X and appearing in no keys of (R, F), X contains a key of (R, F).

3. The Boyce–Codd normal form (BCNF) is defined as the previous form, except that the attribute A may now appear in a key of (R, F). Thus, the formal definition is the following: (R, F) is in BCNF if, for every derived dependency X → A from F, such that A is an attribute not in X, X contains a key of (R, F).

It turns out that every scheme (R, F) in BCNF is in 3NF, whereas the contrary is false in general. Moreover, 3NF and BCNF characterize those schemes recognized as suitable in practice. If a scheme (R, F) is neither 3NF nor BCNF, then it is always possible to decompose (R, F) into subschemes that are at least 3NF. More precisely, by schema decomposition, we mean the replacement of (R, F) by schemes (R_1, F_1), (R_2, F_2), …, (R_k, F_k), where

1. each R_i is a subset of R and R in the union of the R_is;
2. each F_i is the set of all dependencies X → Y derivable from F, such that $XY \subseteq R_i$; and
3. each (R_i, F_i) is in 3NF or in BCNF.

Furthermore, this replacement must ensure that data and dependencies are preserved in the following sense:

1. Data preservation: starting with a relation r that satisfies F, the relations r_i are the projections of r over R_i, and their join must be equal to r.
2. Dependency preservation: the set F and the union of the sets F_i must derive exactly the same functional dependencies.

In the context of functional dependencies, data preservation is characterized as follows, in the case where k = 2: The decomposition of (R, F) into (R_1, F_1), (R_2, F_2) preserves the data if F derives at least one of the two functional dependencies $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$. If k is greater than 2, then the previous result can be generalized using properties of the join operator. Unfortunately, no such easy-to-check property is known for dependency preservation. What has to be done in practice is to make sure that every dependency of F can be derived from the union of the F_is.

It has been shown that it is always possible to decompose a scheme (U, F) so that data and dependencies are preserved and the schemes (R_i, F_i) are all at least in 3NF. But it should be noticed that BCNF is not guaranteed when decomposing a relation scheme. Two kinds of algorithms have been implemented for schema decomposition: the synthesis algorithms (which generate the schemes based on a canonical form of the dependencies of F) and the decomposition algorithms (which repeatedly split the universe U into two subschemes). Synthesis algorithms ensure data and dependency preservation together with schemes in 3NF (at least), whereas decomposition algorithms ensure data preservation together with schemes in BCNF, but at the cost of a possible loss of dependencies.

More on Data Dependencies

Dependencies other than functional dependencies have been widely studied in the past. In particular, multivalued dependencies and their interaction with functional dependencies have motivated much research. The intuitive idea behind multivalued dependencies is that, in a relation over R, a value over X is associated with a set of values over Y, and is independent of the values over R − XY. An example of multivalued dependencies is the following: assume that we have R = {*empno*, *childname*, *car*}, to store the names of the children and the cars of employees. Clearly, every *empno* value is associated with a fixed set of names (of children), independent of the associated *car* values. Multivalued dependencies and functional dependencies have been axiomatized soundly and completely, which has led to an additional normal form, called the fourth normal form, and defined similarly to BCNF.

Other dependencies of practical interest which have been studied are inclusion dependencies. For example, in the database of Fig. 1, stating that every manager must be an employee is expressed by the following inclusion dependency: $\pi_{mgr}(DEPT) \subseteq \pi_{empo}(EMP)$. In general, an inclusion dependency is an expression of the form $\pi_X(r) \subseteq \pi_Y(s)$ where r and s are relations of the database and where X and Y are relation schemes, such that the projections and the inclusion are defined. Although it has been shown that the implication problem for inclusion dependencies in the presence of functional dependencies is not decidable (see Ref. (2)), a restricted case of practical significance is decidable in polynomial time: The restriction is roughly that the relations in inclusion dependencies are all unary.

DATABASE UPDATES

Although updates are an important issue in databases, this area has received less attention from the research community than the topics just addressed. Roughly speaking, updates are basic insert, delete, or modify operations defined on relations seen as *physical* structures, and no theoretical background similar to that discussed for queries is available for updates. As a consequence, no declarative way of considering updates has been proposed so far, although there is much effort in this direction. Actually, current relational systems handle sophisticated updates procedurally, based on the notion of *transactions*, which are programs containing update statements. An important point is that, to maintain data consistency, these programs must be considered as units, in the sense that either all or none of their statements are executed. For instance, if a failure occurs during the execution of a transaction, all updates performed before the failure must be undone before rerunning the whole program. In what follows, we first discuss the relation between updates and data dependencies, and then, we give a short introduction to transaction execution.

Updates and Data Dependencies

There are two main ways to maintain the database consistent with respect to constraints in the presence of updates: (*1*) reject all updates contradicting a constraint and (*2*) take appropriate actions to restore consistency with respect to constraints. To illustrate these two ways of treating updates, let us consider again the database of Fig. 1 and let us assume that the relation DEPT must satisfy the functional dependency *deptno* → *dname mgr*. According to (*1*) previous, the insertion in DEPT of the tuple 1 *toy* 456 is rejected, whereas it is accepted according to (*2*) previous, if, in addition, the tuple 1 *sales* 234 is removed from DEPT. Actually, it turns out that (*1*) gives priority to “old” knowledge over “new” knowledge, whereas (2) does the opposite. Clearly, updating a database according to (*1*) or (*2*) depends on the application. In practice, policy (*1*) is implemented as such for keys and policy (*2*) is specified by transactions.

Before we come to problems related to transaction execution, we would like to mention that an important issue related to policy (*2*) is that of *active rules*. This concept is considered the declarative counterpart of transactions, and thus, is meant as an efficient tool to specify how the database should react to updates, or, more generally, to events.

Active rules are rules of the form: ***on*** ⟨event⟩ ***if*** ⟨condition⟩ ***then*** ⟨action⟩, and provide a declarative formalism for ensuring that data dependencies remain satisfied in the presence of updates. For example, if we consider the database of Fig. 1 and the inclusion dependency $\pi_{\mathrm{mgr}}(\mathrm{DEPT}) \subseteq \pi_{\mathrm{empno}}(\mathrm{EMP})$, the insertion of a new department respects this constraint if we consider the following active rule:

$$\begin{array}{l}\textbf{on}\ \mathrm{insert(n, d, m)\ into\ DEPT}\\ \textit{if}\ \mathrm{m} \notin \pi_{\mathrm{empno}}(\mathrm{EMP})\\ \textbf{then}\ \mathrm{call\ insert\_EMP(m, d)}\end{array}$$

where insert_EMP is an interactive program asking for a name and a salary for the new manager, so that the corresponding tuple can be inserted in the relation EMP.

Another important feature of active rules is their ability to express *dynamic* dependencies. The particularity of dynamic dependencies is that they refer to more than one database state (as opposed to static dependencies that refer to only one database state). A typical dynamic dependency, in the context of the database of Fig. 1, is to state that salaries must never decrease, which corresponds to the following active rule:

$$\begin{array}{l}\textbf{on}\ \mathrm{update\_sal}(ne, new\text{-}sal)\mathrm{in\ EMP}\\ \textit{if}\ new\text{-}sal > \pi_{\mathrm{sal}}(\sigma_{\mathrm{empno=ne}}(\mathrm{EMP}))\\ \textbf{then}\ \mathrm{set}\ sal = new\text{-}sal\ \mathrm{where}\ empno = ne\end{array}$$

where update_sal is the update meant to assign the salary of the employee number *ne* to the value *new-sal* and where the set instruction actually performs the modification.

Although active rules are an elegant and powerful way to specify various dynamic aspects of databases, they raise important questions concerning their execution. Indeed, as the execution of an active rule fires other active rules in its action, the main problem is to decide how these rules are fired. Three main execution modes have been proposed so far in the literature: the immediate mode, the deferred mode, and the concurrent mode. According to the immediate mode, the rule is fired as soon as its event occurs while the condition is true. According to the deferred mode, the actions are executed only after the last event occurs and the last condition is evaluated. In the concurrent mode, no policy of action execution is considered, but a separate process is spawned for each action and is executed concurrently with other processes. It turns out that executing the same active rules according to each of these modes generally gives different results and the choice of one mode over the others depends heavily on the application.

Transaction Management

Contrary to what has been discussed before, the problem of transaction management concerns the physical level of DBMSs and not the conceptual level. Although transaction execution is independent of the conceptual model of databases being used (relational or not), this research area has been investigated in the context of relational databases. The problem is that, in a multiuser environment, several transactions may have to access the same data simultaneously, and then, in this case the execution of these transactions may leave the database inconsistent, whereas each transaction executed alone leaves the database in a consistent state (an example of such a situation will be given shortly). Additionally, modifications of data performed by transactions must survive possible hardware or software failures.

To cope with these difficulties, the following two problems have to be considered: (*1*) the concurrency control problem (that is, how to provide synchronization mechanisms which allow for efficient and correct access of multiple transactions in a shared database) and (*2*) the recovery problem (that is, how to provide mechanisms that react to failures in an automated way). To achieve these goals, the most prominent computational model for transactions is known as the *read-write* model, which considers transactions as sequences of read and write operations operating on the tuples of the database. The operation read(t) indicates that t is retrieved from the secondary memory and entered in the main memory, whereas the operation write(t) does the opposite: The current value of t in the main memory is saved in the secondary memory, and thus survives execution of the transaction. Moreover, two additional operations are considered, modeling, respectively, successful or failed executions: the *commit* operation (which indicates that changes in data must be preserved), and the *abort* operation (which indicates that changes in data performed by the transaction must be undone, so that the aborted transaction is simply ignored). For example, call t the first tuple of the relation EMP of Fig. 1, and assume that two transactions T_1 and T_2 increase John’s salary of 500 and 1,000, respectively. In the read-write model, both T_1 and T_2 have the form: read(t); write(t'); commit, where $t'.sal = t.sal + 500$ for T_1 and where $t'.sal = t.sal + 1{,}000$ for T_2.

Based on these operations, the criterion for correctness of transaction execution is known as *serializability* of schedules. A schedule is a sequence of interleaved operations originating from various transactions, and a schedule built up from transactions $T_1, T_2, \ldots, T_k$ is said to be serializable if its execution leaves the database in the same state as the sequential execution of transactions T_i’s, in some order would do. In the previous example, let us consider the following schedule:

$$\mathrm{read}_1(t); \mathrm{read}_2(t); \mathrm{write}_1(t_1); \mathrm{commit}_1; \mathrm{write}_2(t_2); \mathrm{commit}_2$$

where the subscripts correspond to the transaction where the instructions occur. This schedule is not serializable, because its execution corresponds neither to T_1 followed by T_2 nor to T_2 followed by T_1. Indeed, transactions T_1 and T_2 both read the initial value of t and the effects of T_1 on tuple t are lost, as T_2 commits its changes after T_1.

To characterize serializable schedules, one can design execution protocols. Here again many techniques have been introduced, and we focus on the most frequent of them in actual systems, known as the *two-phase locking protocol*. The system associates every read or write operation on the same object to a lock, respectively, a read-lock or a write-lock, and once a lock is granted for a transaction, other transactions cannot access the corresponding object. Additionally, no lock can be granted to a transaction that has already released a lock. It is easy to see that, in the previous example, such a protocol prevents the execution of the schedule we considered, because T_2 cannot read t unless T_1 has released its write-lock.

Although efficient and easy to implement, this protocol has its shortcomings. For example, it is not free of deadlocks, that is, the execution may never terminate because two transactions are waiting for the same locks at the same time. For instance, transaction T_1 may ask for a lock on object o_1, currently owned by transaction T_2 which in turn asks for a lock on object o_2, currently owned by transaction T_1. In such a situation, the only way to restart execution is to abort one of the two transactions. Detecting deadlocks is performed by the detection of cycles in a graph whose nodes are the transactions in the schedule and in which an edge from transaction T to transaction T′ means that T is waiting for a lock owned by T′.

RELATIONAL DATABASE SYSTEMS AND SQL

In this section, we describe the general architecture of relational DBMSs, and we give an overview of the language SQL which has become a reference for relational systems.

The Architecture of Relational Systems

According to a proposal by the ANSI/SPARC normalization group in 1975, every database system is structured in three main levels:

1. the *internal (or physical) level* which is concerned with the actual storage of data and by the management of transactions;
2. the *conceptual level* which allows describing a given application in terms of the DBMSs used, that is, in terms of relations in the case of a relational DBMSs; and
3. the *external level* which is in charge of taking user's requirements into account.

Based on this three-level general architecture, all relational DBMSs are structured according to the same general schema that is seen as two interfaces, the external interface and the storage interface.

The external interface, which is in charge of the communication between user's programs and the database, contains five main modules: (*1*) precompilers allowing for the use of SQL statements in programs written in procedural languages such as COBOL, C, PASCAL, or JAVA, (*2*) an interactive interface for a real-time use of databases, (*3*) an analyzer which is in charge of the treatment of SQL statements issued either from a user's program or directly by a user via the interactive interface, (*4*) an optimizer based on the techniques discussed previously, and (*5*) a catalog, where information about users and about all databases that can be used, is stored. It is important to note that this catalog, which is a basic component for the management of databases, is itself organized as a relational database, usually called the metadatabase, or data dictionary.

The storage interface, which is in charge of the communications between database and the file management system, also contains five main modules: (*1*) a journal, where all transactions on the database are stored so that the system restarts safely in case of failures, (*2*) the transaction manager which generally works under the two-phase locking protocol discussed previously, (*3*) the index manager (indexes are created to speed up the access to data), (*4*) the space disk manager which is charge of defining the actual location of data on disks, and (*5*) the buffer manager which is in charge of transferring data between the main memory and the disk. The efficiency of this last module is crucial in practice because accesses on disks are very long operations that must be optimized. It is important to note that this general architecture is the basis for organizing relational system that also integrate network and distributed aspects in a client-server configuration or distributed database systems.

An Overview of SQL

Many languages have been proposed to implement relational calculus. For instance, the language QBE (Query By Example) is based on domain calculus, whereas the language QUEL (implemented in the system INGRES) is based on tuple calculus. These languages are described in Ref. (2). We focus here on language SQL which is now implemented in all relational systems.

SQL is based on domain calculus but also refers to the tuple calculus in some of its aspects. The basic structure of an SQL query expression is the following:

```
SELECT ⟨list of attributes⟩
FROM ⟨list of relations⟩
WHERE ⟨condition⟩
```

which roughly corresponds to a relational expression containing projections, selections, and joins. For example, in the database of Fig. 1, the query **E** is expressed in SQL as follows:

```
SELECT EMP.deptno, dname
FROM EMP, DEPT
WHERE sal < 10,000 AND EMP.deptno =
   DEPT.deptno
```

We draw attention to the fact that the condition part reflects not only the selection condition from **E**, but also that, to join tuples from the relations EMP and DEPT, their deptno values must be equal. This last equality must be explicit in SQL, whereas, in the relational algebra, it is a consequence of the definition of the join operator. We also note that terms such as EMP.deptno or DEPT.deptno can be seen as terms from tuple calculus, whereas terms such as deptno or dname, refer to domain calculus. In general, prefixing an attribute name by the corresponding relation name is required if this attribute occurs in more than one relation in the FROM part of the query.

The algebraic renaming operator is implemented in SQL, but concerns relations, rather than attributes as in relational algebra. For example, the algebraic expression $\mathbf{E}_1$ (which computes the number of employees working in at least two distinct departments) is written in SQL as follows:

```
SELECT EMP.empno
FROM EMP,EMP EMPLOYEES
WHERE EMP.deptno != EMPLOYEES.deptno AND
EMP.empno = EMPLOYEES.empno
```

Set theoretic operators union, intersection, and difference are expressed as such in SQL, by the keywords UNION, INTERSECT, and MINUS (or EXCEPT), respectively. Thus, every expression of relational algebra can be written as a SQL statement, and this basic result is known as the *completeness* of the language SQL. An important point in this respect is that SQL expresses more queries than relational algebra as a consequence of introducing functions (whereas function symbols are not considered in relational calculus) and "grouping" instructions in SQL. First, because relations are restricted to the first normal form, it is impossible to consider structured attributes, such as dates or strings. SQL overcomes this problem by providing usual functions for manipulating dates or strings, and additionally, arithmetic functions for counting or for computing minimum, maximum, average, and sum are available in SQL. Moreover, SQL offers the possibility of grouping tuples of relations, through the GROUP BY instruction. As an example of these features, the numbers of departments together with the associated numbers of employees are obtained in the database of Fig. 1 with the following SQL query (in which no WHERE statement occurs, because no selection has to be performed):

```
SELECT deptno, COUNT(empno)
FROM EMP
GROUP BY deptno
```

On the other hand, a database system must incorporate many other basic features concerning the physical storage of tuples, constraints, updates, transactions, and confidentiality. In SQL, relations are created with the CREATE TABLE instruction, where the name of the relation together with the names and types of the attributes are specified. It is important to note that this instruction allows specifying constraints and information about the physical storage of the tuples. Moreover, other physical aspects are taken into account in SQL by creating indexes or clusters to speed up data retrieval.

Update instructions in SQL are either insertion, deletion, or modification instructions in which WHERE statements are incorporated to specify which tuples are affected by the update. For example, in the database of Fig. 1, increasing the salaries of 10% of all employees working in department number 1 is achieved as follows:

```
UPDATE EMP
SET sal = sal * 1.1
WHERE deptno = 1
```

Transactions are managed in SQL by the two-phase locking protocol, using different kinds of locks, allowing only read data or allowing read and write data. Moreover, activeness in databases is taken into account in SQL through the notion of *triggers*, which are executed according to the immediate mode.

Data confidentiality is another very important issue, closely related to data security, but has received very little attention at the theoretical level. Nevertheless, this problem is addressed in SQL in two different ways: (*1*) by restricting the access to data to specified users and (*2*) by allowing users to query only the part of the database they have permission to query. Restricting access to data by other users is achieved through the GRANT instruction, that is specified by the owner either on a relation or on attributes of a relation. A GRANT instruction may concern queries and/or updates, so that, for example, a user is allowed to query for salaries of employees, while forbidding the user to modify them. On the other hand, a different way to ensure data confidentiality consists in defining derived relations called *views*. For instance, to prevent users from seeing the salaries of employees, one can define a view from the relation EMP of Fig. 1 defined as the projection of this relation over attributes *empno*, *ename*, and *deptno*. A view is a query, whose SQL code is stored in the metadatabase, but whose result is not stored in the database. The concept of views is a very efficient tool for data confidentiality, thanks to the high expressional power of queries in SQL. However, the difficulty with views is that they are not updatable, except in very restricted cases. Indeed, because views are derived relations, updates on views must be translated into updates on the relations of the database, and this translation, when it exists, is generally not unique. This problem, known as the nondeterminism of view updating, is the subject of many research efforts, but has not yet been satisfactorily solved.

We conclude by mentioning that relational systems are successful in providing powerful database systems for many applications, essentially for business applications. However, these systems are not adapted to many new applications, such as geographical information systems or knowledge-based management because of two kinds of limitations on the relational model:

1. Relations are flat structures which prevent easily managing data requiring sophisticated structures. This remark led to the emergence of object-oriented database systems that are currently the subject of important research efforts, most of them originating from concepts of object-oriented languages, and also from concepts of relational databases. As

another research direction in this area, we mention the emergence of object-relational data models that extend the relational model by providing a richer type system including object orientation, and that add constructs to relational languages (such as SQL) to deal with the added data types. An introductory discussion on object-oriented databases and object-relational data models is given in Ref. (8), whereas a complete and formal description of these models can be found in Refs. (2) and (9).

2. Relational algebra does not allow for recursivity (see Ref. (3)), and thus, queries, such as the computation of the transitive closure of a graph cannot be expressed. This remark has stimulated research in the field of deductive databases, a topic closely related to logic programming but which also integrates techniques and concepts from relational databases. The basic concepts of deductive databases and their connections with relational databases are presented in Refs. (2) and (9) and studied in full detail in Ref. (10).

We finally mention several new and important fields of investigation that have emerged during the last decade. These fields are *data mining, data warehousing*, and *semistructured data*. Indeed, extracting abstracted information from many huge and heterogeneous databases is now a crucial issue in practice. As a consequence, many research efforts are still currently devoted to the study of efficient tools for knowledge discovery in databases (KDD or data mining), as well as for data integration in a data warehouse. It is important to note that these new fields rely heavily on the concept of relational databases, because the relational model is the basic database model under consideration. Data mining and data warehousing are briefly discussed in Ref. (8) and are introduced in more details in Refs. (2) and (11). On the other hand, the Web is causing a revolution in how we represent, retrieve, and process information. In this respect, the language XML is recognized as the reference for data exchange on the Web, and the field of semistructured data aims to study how XML documents can be managed. Here again, relational database theory is the basic reference for the storage and the manipulation of XML documents. An in-depth and up-to-date look at this new topic can be found in Ref. (12).

We note that the latest versions of DBMSs now available on the marketplace propose valuable and efficient tools for dealing with data mining, data warehousing, and semistructured data.

BIBLIOGRAPHY

1. E. F. Codd, A relational model of data for large shared data banks, *Commun. ACM*, **13**: 377–387, 1970.
2. H. Garcia-Molina, J. D. Ullman, and J. D. Widom, *Database Systems: The Complete Book*, Englewood Cliffs, NJ: Prentice-Hall, 2001.
3. D. Maier, *The Theory of Relational Databases*, Rockville, MD: Computer Science Press, 1983.
4. P. C. Kanellakis,Elements of relational database theory, in J. VanLeuwen (ed.), *Handbook of Theoretical Computer Science*, Vol. B: *Formal and Semantics*. Amsterdam: North Holland, 1990, pp. 1073–1156.
5. J. W. Lloyd, *Foundations of Logic Programming*, 2nd ed., Berlin: Springer-Verlag, 1987.
6. E. F. Codd, Relational completeness of data base sublanguages, in R. Rustin (ed.), *Data Base Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1972, pp. 65–98.
7. W. W. Armstrong, Dependency structures of database relations. *Proc. IFIP Congress*, Amsterdam: North Holland, 1974, pp. 580–583.
8. A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 3rd ed., New York: McGraw-Hill series in Computer Science, 1996.
9. S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Reading, MA: Addison-Wesley, 1995.
10. S. Ceri, G. Gottlob, and L. Tanca, *Logic Programming and Databases*, Berlin: Springer-Verlag, 1990.
11. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, San Francisco, CA: Morgan Kaufman, 2006.
12. S. Abiteboul, P. Buneman, and D. Suciu, *Data on the Web: From relations to Semistructured Data and Xml*, San Francisco, CA: Morgan Kaufman, 1999.

FURTHER READING

P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Reading, MA: Addison-Wesley, 1987. A good introduction and a fine reference source for the topic of transaction management.

C. J. Date, *Introduction to Database Systems*, 8th ed., Reading, MA: Addison-Wesley, 2003. One of the reference textbooks on relational databases.

R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 2nd ed., Redwood City, CA: Benjamin Cummings, 1994. One of the most widely used database textbooks.

M. Levene and G. Loizou, *A Guided Tour of Relational Databases and Beyond*, Berlin: Springer-Verlag, 1999. A complete textbook that addresses theoretical and practical aspects of relational databases.

C. H. Papadimitriou, *The Theory of Database Concurrency Control*, Rockville, MD: Computer Science Press, 1986. A reference source for the theoretical foundations of concurrency control.

J. D. Ullman, *Principles of Database and Knowledge Base Systems*, Vol. I, II, Rockville MD: Computer Science Press, 1988. One of the first complete and reference textbooks on databases.

M. Y. Vardi, Fundamentals of dependency theory, in E. Borger (ed.), *Trends in Theoretical Computer Science*, Rockville, MD: Computer Science Press, 1987, pp. 171–224. A complete introduction to theoretical aspects of dependency theory.

G. Vossen, *Data Models, Database Languages, and Database Management Systems*, Workingham, UK: Addison-Wesley, 1991. This book is a fine introduction to the theory of databases.

Dominique Laurent
University of Cergy-Pontoise
Cergy-Pontoise, France

SPATIAL DATABASES

INTRODUCTION

Spatial database management systems (1–6) aim at the effective and efficient management of data related to

- space in the physical world (geography, urban planning, astronomy, human anatomy, fluid flow, or an electromagnetic field),
- biometrics (fingerprints, palm measurements, and facial patterns),
- engineering design (very large-scale integrated circuits, layout of a building, or the molecular structure of a pharmaceutical drug), and
- conceptual information space (virtual reality environments and multidimensional decision-support systems).

A spatial database management system (SDBMS) can be characterized as follows:

- A SDBMS is a software module that can work with an underlying database management system, for example, an object-relational database management system or object-oriented database management system.
- SDBMSs support multiple spatial data models, commensurate spatial abstract data types (ADTs), and a query language from which these ADTs are callable.
- SDBMSs support spatial indexing, efficient algorithms for spatial operations, and domain-specific rules for query optimization.

Spatial database research has been an active area for several decades. The results of this research are being used several areas. To cite a few examples, the filter-and-refine technique used in spatial query processing has been applied to subsequence mining; multidimensional-index structures such as R-tree and Quad-tree used in accessing spatial data are applied in the field of computer graphics and image processing; and space-filling curves used in spatial query processing and data storage are applied in dimension-reduction problems. The field of spatial databases can be defined by its accomplishments; current research is aimed at improving its functionality, extensibility, and performance. The impetus for improving functionality comes from the needs of existing applications such as geographic information systems (GIS), location-based services (LBS) (7), sensor networks (8), ecology and environmental management (9), public safety, transportation (10), earth science, epidemiology (11), crime analysis (12), and climatology.

Commercial examples of spatial database management include ESRI's ArcGIS Geodatabase (13), Oracle Spatial (14), IBM's DB2 Spatial Extender and Spatial Datablade, and future systems such as Microsoft's SQL Server 2008 (code-named Katmai) (15). Spatial databases have played a major role in the commercial industry such as Google Earth (16) and Microsoft's Virtual Earth (17). Research prototype examples of spatial database management systems include spatial datablades with PostGIS (18), MySQL's Spatial Extensions (19), Sky Server (20), and spatial extensions. The functionalities provided by these systems include a set of spatial data types such as a points, line segments and polygons, and a set of spatial operations such as inside, intersection, and distance. The spatial types and operations may be made a part of a query language such as SQL, which allows spatial querying when combined with an object-relational database management system (21,22). The performance enhancement provided by these systems includes a multidimensional spatial index and algorithms for spatial database modeling such as OGC (23) and 3-D topological modeling; spatial query processing including point, regional, range, and nearest-neighbor queries; and spatial data methods that use a variety of indexes such as quad trees and grid cells.

Related Work and Our Contributions

Published work related to spatial databases can be classified broadly as follows:

- Textbooks (3,4,6,24), which explain in detail various topics in spatial databases such as logical data models for spatial data, algorithms for spatial operations, and spatial data access methods. Recent textbooks (6,25) deal with research trends in spatial databases such as spatio–temporal databases and moving objects databases.
- Reference books (26,27), which are useful for studying areas related to spatial databases, for example, multidimensional data structures and geographic information systems (GIS).
- Journals and conference proceedings (28–37), which are a source of in-depth technical knowledge of specific problem areas in spatial databases.
- Research surveys (1,38,39), which summarize key accomplishments and identify research needs in various areas of spatial databases at that time.

Spatial database research has continued to advance greatly since the last survey papers in this area were published (1,38,39). Our contribution in this chapter is to summarize the most recent accomplishments in spatial database research, a number of which were identified as research needs in earlier surveys. For instance, bulk loading techniques and spatial join strategies are rereferenced here as well as other advances in spatial data mining and conceptual modeling of spatial data. In addition, this chapter provides an extensive updated list of research needs in

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

such areas as management of 3-D spatial data, visibility queries, and many others. The bibliography section at the end of this chapter contains a list of over 100 references, updated with the latest achievements in spatial databases.

Scope and Outline

The goal of this chapter is to provide the reader with a broad introduction to spatial database systems. Spatial databases are discussed in the context of object-relational databases (21,22,40), which provide extensibility to many components of traditional databases to support the spatial domain. Three major areas that receive attention in the database context—conceptual, logical, and physical data models—are discussed (see Table 1). In addition, applications of spatial data for spatial data mining are also explored.

Emerging needs for spatial database systems include the handling of 3-D spatial data, spatial data with temporal dimension, and effective visualization of spatial data. The emergence of hardware technology such as storage area networks and the availability of multicore processors are two additional fields likely to have an impact on spatial databases. Such topics of research interest are introduced at the end of each section. References are provided for more exploration. Because of the size constraints of this chapter, several other overlapping research needs such as spatio–temporal databases and uncertainty are not included in this chapter.

The rest of this chapter is organized as follows: Fundamental concepts helpful to understand spatial databases are presented. Spatial database modeling is described at the conceptual and logical levels; techniques for spatial query processing are discussed; file organizations and index data structures are presented; and spatial data mining patterns and techniques are explored.

MATHEMATICAL FRAMEWORK

Accomplishments

Spatial data are relatively more complex compared with traditional business data. Specific features of spatial data include: (1) rich data types (e.g., extended spatial objects), (2) implicit spatial relationships among the variables, (3) observations that are not independent, and (4) spatial autocorrelation among the features.

Spatial data can be considered to have two types of attributes: nonspatial attributes and spatial attributes. nonspatial attributes are used to characterize nonspatial features of objects, such as name, population, and unemployment rate for a city. Spatial attributes are used to define the spatial location and extent of spatial objects (41). The spatial attributes of a spatial object most often include information related to spatial locations, for example, longitude, latitude, elevation, and shape. Relationships among nonspatial objects are explicit in data inputs, e.g., arithmetic relation, ordering, instance of, subclass of, and membership of. In contrast, relationships among spatial objects are often implicit, such as overlap, intersect, and behind.

Space is a framework to formalize specific relationships among a set of objects. Depending on the relationships of interest, different models of space such as set-based space, topological space, Euclidean space, metric space, and network space can be used (6). Set-based space uses the basic notion of elements, element-equality, sets, and membership to formalize the set relationships such as set-equality, subset, union, cardinality, relation, function, and convexity. Relational and object-relational databases use this model of space.

Topological space uses the basic notion of a neighborhood and points to formalize the extended object relations such as boundary, interior, open, closed, within, connected, and overlaps, which are invariant under elastic deformation. Combinatorial topological space formalizes relationships such as Euler's formula (number of faces + number of vertices − number of edges = 2 for planar configuration). Network space is a form of topological space in which the connectivity property among nodes formalizes graph properties such as connectivity, isomorphism, shortest-path, and planarity.

Euclidean coordinatized space uses the notion of a coordinate system to transform spatial properties and relationships to properties of tuples of real numbers. Metric spaces formalize the distance relationships using positive symmetric functions that obey the triangle inequality. Many multidimensional applications use Euclidean coordinatized space with metrics such as distance.

Research Needs

Many spatial applications manipulate continuous spaces of different scales and with different levels of discretization. A sequence of operations on discretized data can lead to growing errors similar to the ones introduced by finite-precision arithmetic on numbers. Preliminary results (1) are available on the use of discrete basis and bounding errors with peg-board semantics. Another related problem concerns interpolation to estimate the continuous field from a discretization. Negative spatial autocorrelation makes interpolation error-prone. More work is needed on a framework to formalize the discretization process and its associated errors, and on interpolation.

Table 1. Spatial database topics

Mathematical Framework	
Conceptual Data Model	
Logical Data Model	Trends: Spatial Data Mining
Query Languages	
Query Processing	
File Organizations and Indices	

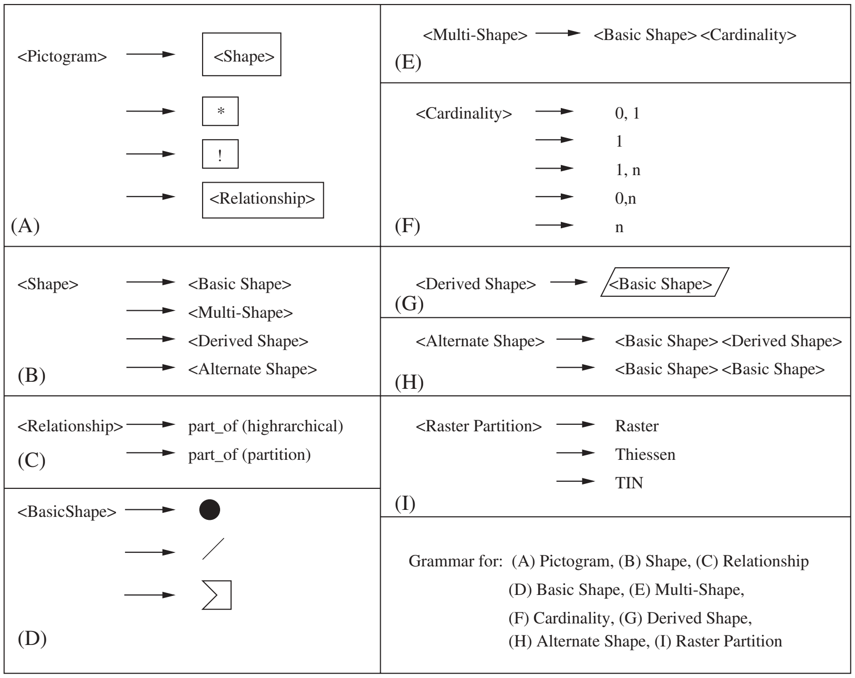


Figure 1. Pictograms.

SPATIAL-DATABASE CONCEPTUAL MODELING

Accomplishments

Entity relationship (ER) diagrams are commonly used in designing the conceptual model of a database. Many extensions (42) have been proposed to extend ER to make the conceptual modeling of spatial applications easier and more intuitive. One such extension is the use of pictograms (43). A pictogram is a graphical icon that can represent a spatial entity or a spatial relationship between spatial entities. The idea is to provide constructs to capture the semantics of spatial applications and at the same time to keep the graphical representation simple. Figure 2 provides different types of pictograms for spatial entities and relationships. In the following text we define pictograms to represent spatial entities and relationships and their grammar in graphical form.

Pictogram: A pictogram is a representation of the object inserted inside of a box. These iconic representations are used to extend ER diagrams and are inserted at appropriate places inside the entity boxes. An entity pictogram can be of a basic shape or a user-defined shape.

Shape: Shape is the basic graphical element of a pictogram that represents the geometric types in the spatial data model. It can be a basic shape, a multishape, a derived shape, or an alternate shape. Most objects have simple (basic) shapes [Fig. 1 (b)].

Basic Shape: In a vector model, the basic elements are point, line, and polygon. In a forestry example, the user may want to represent a facility as a point (0-D), a river or road network as lines (1-D), and forest areas as polygons (2-D) [Fig. 1(d)].

Multishape: To deal with objects that cannot be represented by the basic shapes, we can use a set of aggregate shapes. Cardinality is used to quantify multishapes. For example, a river network that is represented as a line pictogram scale will have cardinality 0 [Fig. 1(b) and (e)].

Derived shape: If the shape of an object is derived from the shapes of other objects, its pictogram is italicized. For example, we can derive a forest boundary (polygon) from its “forest-type” boundaries (polygon), or a country boundary from the constituent-state boundaries [Fig. 1(c) and (g)].

Alternate shape: Alternate shapes can be used for the same object depending on certain conditions; for example, objects of size less than x units are represented as points, whereas those greater than x units are represented as polygons. Alternate shapes are represented as a concatenation of possible pictograms. Similarly, multiple shapes are needed to represent objects at different scales; for example, at higher scales lakes may be represented as points and at lower scales as polygons [Fig. 1(d) and (h)].

Any possible shape: A combination of shapes is represented by a wild card * symbol inside a box, which implies that any geometry is possible (Fig. 1(e)).

User-defined shape: Apart from the basic shapes of point, line, and polygon, user-defined shapes are possible. User-defined shapes are represented by an exclamation symbol (!) inside a box [Fig. 1(a)].

Relationship pictograms: Relationship pictograms are used to model the relationship between entities. For example, *part_of* is used to model the relationship between a route and a network, or it can be used to model the partition of a forest into forest stands [Fig. 1(c)].

The popularity of object-oriented languages such as C++ and Java has encouraged the growth of object-oriented database systems (OODBMS). The motivation behind this growth in OODBMS is that the direct mapping of the conceptual database schema into an object-oriented language leads to a reduction of impedance mismatch encountered when a model on one level is converted into a model on another level.

UML is one of the standards for conceptual-level modeling for object-oriented software design. It may also be applied to an OODBMS to capture the design of the system conceptually. A UML design consists of the following building blocks:

Class: A class is the encapsulation of all objects that share common properties in the context of the application. It is the equivalent of the entity in the ER model. The class diagrams in a UML design can be further extended by adding pictograms. In a forestry example, classes can be forest, facility, forest stand, and so forth.

Attributes: Attributes characterize the objects of the class. The difference between an attribute ER and a UML model design is that no notion of a key attribute exists in UML, in an object-oriented system, each object has an implicit system-generated unique identification. In UML, attributes also have a scope that restricts the attribute's access by other classes. Three levels of scope exist, and each has a special symbol: + Public: This symbol allows the attribute to be accessed and manipulated from any class. - Private: Only the class that owns the attribute is allowed to access the attribute. # Protected: Other than the class that owns the attribute, classes derived from the class that owns can access the attibute.

Methods: Methods are functions and a part of class definition. They are responsible for modifying the behavior or state of the class. The state of the class is embodied in the current values of the attributes. In object-oriented design, attributes should only be accessed through methods.

Relationships: Relationships relate one class to another or to itself. This concept is similar to the concept of relationship in the ER model. Three important categories of relationships are as follows:

- *Aggregation*: This category is a specific construct to capture the part–whole relationship. For instance, a group of forest–stand classes may be aggregated into a forest class.
- *Generalization*: This category describes a relationship in which a child class can be generalized to a parent class. For example, classes such as point, line, and polygon can be generalized to a geometry class.
- *Association*: This category shows how objects of different classes are related. An association is binary if it connects two classes or ternary if it connects three classes. An example of a binary association is *supplieswater to* between the classes river and facility.

Figures 2 and 3 provide an example for modeling a *State-Park* using ER and UML with pictograms, respectively.

Research Needs

Conceptual modeling for spatio–temporal and moving-object data needs to be researched. Pictograms as introduced in this section may be extended to handle such data.

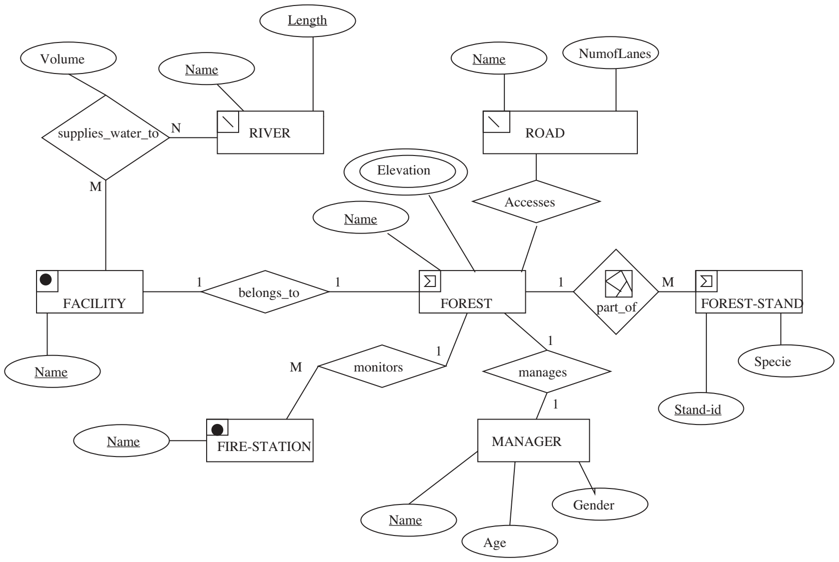


Figure 2. Example ER diagram with pictograms.

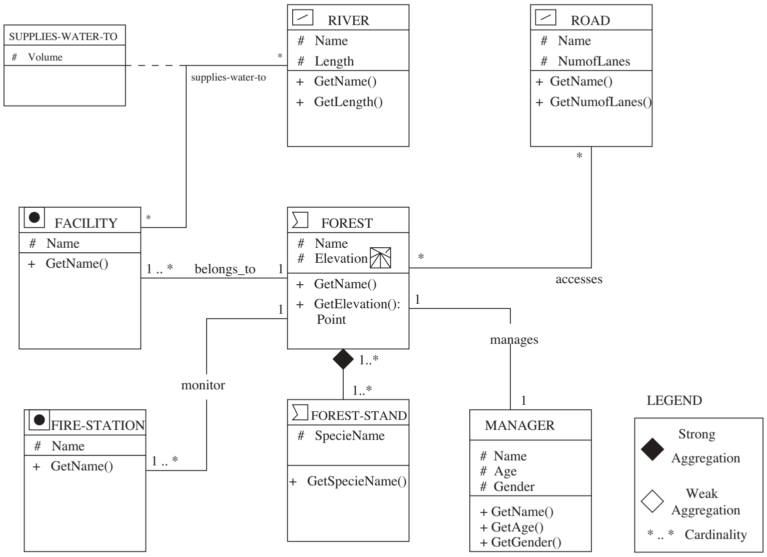


Figure 3. Example UML class diagram with pictograms.

Models used in the spatial representation of data can be extended to conside the time dimension. For instance, the nine-intersection matrix used to represent topology can be differentiated to consider the change in topology over a period of time. Similarly, other spatial properties such as position, orientation, and shape can be differentiated to consider effects over time such as motion, rotation, and deformation of a spatial object. Similarly, series of points can be accumulated to represent time-varying spatial data and properties.

Another area of research is the use of ontology for knowledge management. An ontology defines a common vocabulary that allows knowledge to be shared and reused across different applications. Ontologies provide a shared and common understanding of some domain that can be communicated across people and computers.

Geospatial ontology (44) is specific to the geospatial domain. Research in geospatial ontology is needed to provide interoperability between geospatial data and software. Developing geospatial ontologies is one of the long-term research challenges for the University Consortium for Geographic Information Systems (UCGIS) (37). Research in this area is also being carried out by companies such as CYC for geospatial ontology.

Geospatial ontology can be extended to include the temporal dimension. The ontology of time has been researched in the domain of artificial intelligence as situation calculus. OWL-Time (45) is an ontology developed to represent time.

Semantic web (46) is widely known as an efficient way to represent data on the Web. The wealth of geographic information currently available on the Web has prompted research in the area of GeoSpatial Semantic Web (47,48). In this context, it becomes necessary to create representations of the geographic information resources. This necessity must lead to a framework for information retrieval based on the semantics of spatial ontologies. Developing the geospatial ontology that is required in a geo-spatial semantic web is challenging because the defining properties of geographic entities are very closely related to space (i.e., multidimensional space). In addition, each entity may have several subentities resulting in a complex object (48). One popular data model used in representing semantic web is the resource description framework (RDF) (49).

RDF is being extended (GeoRDF) (50) to include spatial dimensions and hence to provide the necessary support for geographica data on the web.

SPATIAL DATA MODELS AND QUERY LANGUAGES

Accomplishments

Data Models. A spatial data model provides the data abstraction necessary to hide the details of data storage. The two commonly used models are the field-based model and the object-based model. Whereas the field-based model adopts a functional viewpoint, the object-based model treats the information space as a collection of discrete, identifiable, spatially referenced entities. Based on the type of data model used, the spatial operations may change. Table 2 lists the operations specific to the field-based and object-based models. In the context of object-relational databases, a spatial data model is implemented using a set of spatial data types and operations. Over the last two decades, an enormous amount of work has been done in the design and development of spatial abstract data types and their embedding in a query language. Serious efforts are being made to arrive at a consensus on standards through the OGC (51).

OGC proposed the general feature model (51) where features are considered to occur at two levels, namely, feature instances and feature types. A geographic feature is represented as a discrete phenomenon characterized by its geographic and temporal coordinates at the instance

Table 2. Data model and operations

Data Model	Operator Group	Operation
Vector Object	Set-Oriented	equals, is a member of, is empty, is a subset of, is disjoint from, intersection, union, difference, cardinality
	Topological	boundary, interior, closure, meets, overlaps, is inside, covers, connected, components, extremes, is within
	metric	distance, bearing/angle, length, area, perimeter
	Direction	east, north, left, above, between
	Network	successors, ancestors, connected, shortest-path
	Dynamic	translate, rotate, scale, shear, split, merge
Raster Field	Local	point-wise sums, differences, maximumms, means, etc.
	Focal	slop, aspect, weighted average of neighborhood
	Zonal	sum or mean or maximum of field values in each zone

level, and the instances with common characteristics are grouped into classes called feature types. Direction is another important feature used in spatial applications. A direction feature can be modeled as a spatial object (52). Research has also been done to efficiently compute the cardinal direction relations between regions that are composed of sets of spatial objects (53).

Query Languages. When it comes to database sytems, spatial database researchers prefer object-based models because the data types provided by object-based database systems can be extended to spatial data types by creating abstract data types (ADT). OGC provides a framework for object-based models. Figure 4 shows the OpenGIS approach to modeling geographic features. This framework provides conceptual schemas to define abstract feature types and provides facilities to develop application schemas that can capture data about feature instances. Geographic phenomena fall into two broad categories, discrete and continuous. Discrete phenomena are objects that have well-defined boundaries or spatial extent, for example, buildings and streams. Continuous phenomena vary over space and have no specific extent (e.g., temperature and elevation). A continuous phenomenon is described in terms of its value at a specific position in space (and possibly time). OGC represents discrete phenomena (also called vector data) by a set of one or more geometric primitives (points, curves, surfaces, or solids). A continuous phenomenon is represented through a set of values, each associated with one of the elements in an array of points. OGC uses the term “coverage” to refer to any data representation that assigns values directly to spatial position. A coverage is a function from a spatio–temporal domain to an attribute domain. OGC provides standardized representations for spatial characteristics through geometry and topology. Geometry provides the means for the quantitative description of the spatial characteristics including dimension, position, size, shape, and orientation. Topology deals with the characteristics of geometric figures that remain invariant if the space is deformed elastically and continuously. Figure 5 shows the hierarchy of geometry data types. Objects under primitive (e.g., points and curves) will be open (i.e., they will not contain their boundary points), and the objects under complex (e.g., disjoint objects) will be closed.

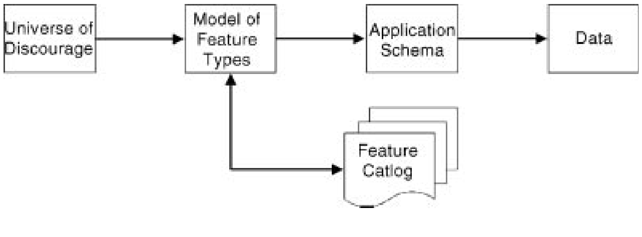


Figure 4. Modeling geographic information [source: (51)].

In addition to defining the spatial data types, OGC also defines spatial operations. Table 3 lists basic operations operative on all spatial data types. The topological operations are based on the ubiquitous nine-intersection model. Using the OGC specification, common spatial queries can be posed intuitively in SQL. For example, the query *Find all lakes which have an area greater than 20 sq. km. and are within 50 km. from the campgrounds* can be posed as shown in Table 4 and Fig. 6. Other GIS and LBS example queries are provided in Table 5. The OGC specification is confined to topological and metric operations on vector data types. Also, several spatio–temporal query languages have been studied that are trigger-based for relational-oriented models (54), moving objects (55), future temporal languages (56), and constraint-based query languages (57).

For spatial networks, commonly used spatial data types include objects such as node, edge, and graph. They may be constructed as an ADT in a database system. Query languages based on relational algebra are unable to express certain important graph queries without making certain assumptions about the graphs. For example, the transitive closure of a graph may not be determined using relational algebra. In the SQL3, a recursion operation RECURSIVE has been proposed to handle the transitive closure operation.

Research Needs

Map Algebra. Map Algebra (58) is a framework for raster analysis that has now evolved to become a preeminent language for dealing with field-based models. Multiple operations can be performed that take multiple data layers that are overlayed upon each other to create a new layer. Some common groups of operations include local, focal, and zonal. However, research is needed to account for the generalization of temporal or higher dimensional data sets (e.g., 3-D data).

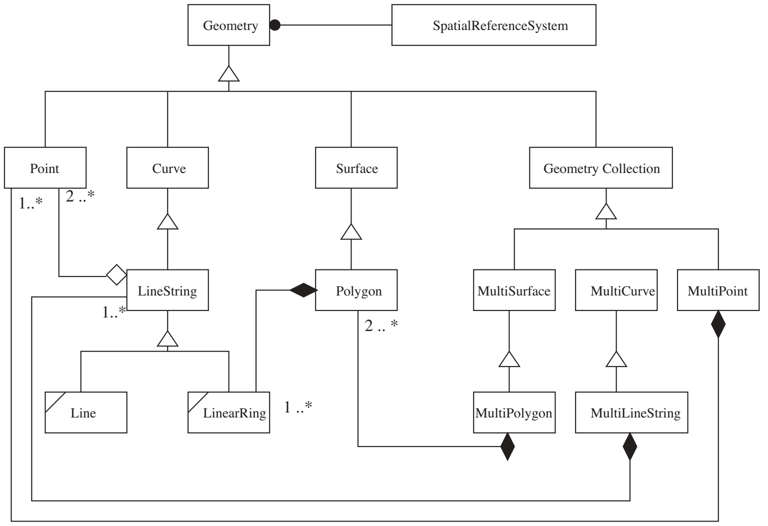


Figure 5. Hierarchy of data types.

Modeling 3-D Data. The representation of volumetric data is another field to be researched. Geographic attributes such as clouds, emissions, vegetation, and so forth. are best described as point fields on volumetric bounds. Sensor data from sensor technologies such as LADAR (Laser Detection and Ranging), 3-D SAR (Synthtic Arperture Radar), and EM collect data volumetrically. Because volumetric data is huge, current convention is to translate the data into lower-dimensional representations such as B-reps, Point clouds, NURBS, and so on. This action results in loss of intrinsic 3-D information. Efforts (59) have been made to develop 3-D data models that emphasize the

Table 3. A sample of operations listed in the OGC standard for SQL

	Basic Functions
SpatialReference()	Returns the underlying coordinate system of the geometry
Envelope()	Returns the minimum orthogonal bounding rectangle of the geometry
Export ()	Returns the geometry in a different representation
IsEmpty()	Returns true if the geometry is an empty set.
IsSimple()	Returns true if the geometry is simple (no self-intersection)
Boundary ()	Returns the boundary of the geometry
	Topological/ Set Operators
Equal	Returns true if the interior and boundary of the two geometries are spatially equal
Disjoint	Returns true if the boundaries and interior do not intersect.
Intersect	Returns true if the interiors of the geometries intersect
Touch	Returns true if the boundaries intersect but the interiors do not.
Cross	Returns true if the interior of the geometries intersect but the boundaries do not
Within	Returns true if the interior of the given geometry does not intersect with the exterior of another geometry.
Contains	Tests if the given geometry contains another given geometry
Overlap	Returns true if the interiors of two geometries have non-empty intersection
	Spatial Analysis
Distance	Returns the shortest distance between two geometries
Buffer	Returns a geometry that consists of all points whose distance from the given geometry is less than or equal to the specified distance
ConvexHull	Returns the smallest convex set enclosing the geometry
Intersection	Returns the geometric intersection of two geometries
Union	Returns the geometric union of two geometries
Difference	Returns the portion of a geometry which does not intersect with another given geometry
SymmDiff	Returns the portions of two geometries which do not intersect with each other

Table 4. SQL query with spatial operators

```
SELECT L.name
FROM Lake L, Facilities Fa
WHERE Area(L.Geometry) > 20 AND
Fa.name = ’campground’ AND
Distance(Fa.Geometry, L.Geometry) < 50
```

significance of the volumetric shapes of physical world objects. This topological 3-D data model relies on Poincare algebra. The internal structure is based on a network of simplexes, and the internal data structure used is a tetrahedronized irregular network (TIN) (59,60), which is the three-dimensional variant of the well-known triangulated irregular network (TIN).

Modeling Spatial Temporal Networks. Graphs have been used extensively to represent spatial networks. Considering the time-dependence of the network parameters and their topology, it has become critically important to incorporate the temporal nature of these networks into their models to make them more accurate and effective. For example, in a transportation network, the travel times on road segments are often dependent on the time of the day, and there can be intervals when certain road segments are not available for service. In such, time-dependent networks modeling the time variance becomes very important. Time-expanded graphs (61) and time-aggregated graphs (62) have been used to model time-varying spatial networks. In the time-expanded representation, a copy of the entire network is maintained for every time instant, whereas the time-aggregated graphs maintain a time series of attributes, associated to every node and edge.

Network modeling can be extended to consider 3-D spatial data. Standard road network features do not represent 3-D structure and material properties. For instance, while modeling a road tunnel, we might want to represent its overpass clearance as a spatial property. Such properties will help take spatial constraints into account while selecting routes.

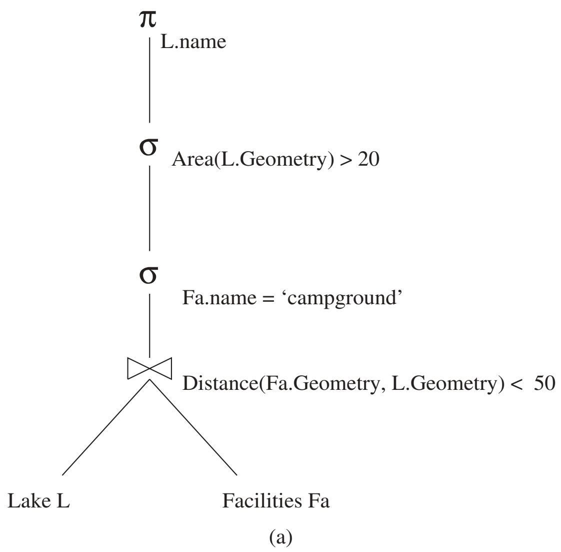


Figure 6. SQL query tree.

Modeling Moving Objects. A moving object database is considered to be a spatio–temporal database in which the spatial objects may change their position and extent over a period of time. To cite a few examples, the movement of taxi cabs, the path of a hurricane over a period of time, and the geographic profiling of serial criminals are a few examples in which a moving-objects database may be considered. Referances 25 and 63 have provided a data model to support the design of such databases.

Markup Languages. The goals of markup languages, such as geography markup language (GML) (64), are to provide a standard for modeling language and data exchange formats for geographic data. GML is an XML-based markup language to represent geographic entities and the relationships between them. Entities associated with geospatial data such as geometry, coordinate systems, attributes, and

Table 5. Typical spatial queries from GIS and LBS

GIS Queries	
Grouping	Recode all land with silty soil to silt-loadm soil
Isolate	Select all land owned by Steve Steiner
Classify	If the population density is less than 100 people / sq. mi., land is acceptable
Scale	Change all measurement’s’ to the metric system
Rank	If the road is an Interstate, assign it code 1; if the road is a state or US highway, assign it code 2; otherwise assign it code 3
Evaluate	If the road code is 1, then assign it Interstate; if the road code is 2, then assign it Main Artery; if the road code is 3, assign it Local Road
Rescale	Apply a function to the population density
Attribute Join	Join the Forest layer with the layer containing forest-cover codes
Zonal	Produce a new map showing state populations given county population
Registration	Align two layers to a common grid reference
Spatial Join	Overlay the land-use and vegetation layers to produce a new layer
LBS Queries	
Nearest Neighbor	List the nearest gas stations
Directions	Display directions from a source to a destation (e.g. Google Maps, Map Quest)
Local Search	Search for restaurants in the neighborhood (e.g. Microsoft Live Local, Google Local)

so forth, can be represented in a standard way using GML. Several computational challenges exist with GML, such as spatial query processing and indexing (65). CityGML (66) is a subclass of GML useful for representing 3-D urban objects, such as buildings, bridges, tunnels, and so on. CityGML allows modeling of spatial data at different levels of detail regarding both geometry and thematic differentiation. It can be used to model 2.5-D data (e.g., digital terrain model), and 3-D data (walkable architecture model). Keyhole markup language (KML) (67) is another XML-based markup language popular with commercial spatial software from Google. Based on a structure similar to GML, KML allows representation of points, polygons, 3-D objects, attributes, and so forth.

SPATIAL QUERY PROCESSING

Accomplishments

The efficient processing of spatial queries requires both efficient representation and efficient algorithms. Common representations of spatial data in an object model include spaghetti, the node-arc-node (NAA) model, the doubly connected-edge-list (DCEL), and boundary representation, some of which are shown in Fig. 7 using entity-relationship diagrams. The NAA model differentiates between the topological concepts (node, arc, and areas) and the embedding space (points, lines, and areas). The spaghetti-ring and DCEL focus on the topological concepts. The representation of the field data model includes a regular tessellation (triangular, square, and hexagonal grid) and triangular irregular networks (TIN).

Query processing in spatial databases differs from that of relational databases because of the following three major issues:

- Unlike relational databases, spatial databases have no fixed set of operators that serve as building blocks for query evaluation.
- Spatial databases deal with extremely large volumes of complex objects. These objects have spatial extensions and cannot be naturally sorted in a one-dimensional array.
- Computationally expensive algorithms are required to test for spatial predicates, and the assumption that I/O costs dominate processing costs in the CPU is no longer valid.

In this section, we describe the processing techniques for evaluating queries on spatial databases and discuss open problems in spatial query processing and query optimization.

Spatial Query Operations. Spatial query operations can be classified into four groups (68).

- **Update Operations:** These include standard database operations such as modify, create, and delete.
- **Spatial Selection:** These can be of two types:
  - **Point Query:** Given a query point, find all spatial objects that contain it. An example is the following query, “Find all river flood-plains which contain the SHRINE.”
  - **Regional Query:** Given a query polygon, find all spatial objects that intersect the query polygon. When the query polygon is a rectangle, this query is called a window query. These queries are sometimes also referred to as range queries. An example query could be “Identify the names of all forest stands that intersect a given window.”
  - **Spatial Join:** Like the join operator in relational databases, the spatial join is one of the more important operators. When two tables are joined on a spatial attribute, the join is called a spatial join. A variant of the spatial join and an important operator in GIS is the *map overlay.* This operation combines two sets of spatial objects to form new ones. The “boundaries” of a set of these new objects are determined by the nonspatial attributes assigned by the overlay operation. For example, if the operation assigns the same value of the nonspatial attribute

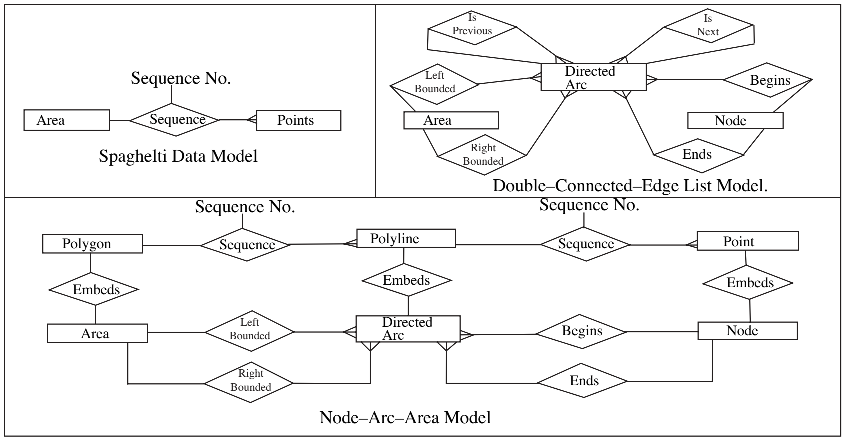


Figure 7. Entity relationship diagrams for common representations of spatial data.

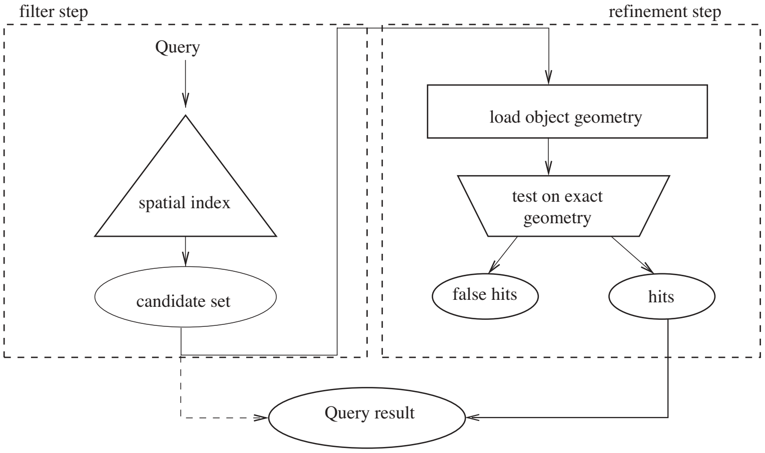


Figure 8. Two-step processing.

to two neighboring objects, then the objects are "merged." Some examples of spatial join predicates are *intersect, contains, is_enclosed_by, distance, northwest, adjacent, meets,* and *overlap*. A query example of a spatial join is "Find all forest-stands and river flood-plains which overlap."

– **Spatial Aggregate:** An example of a spatial aggregate is "Find the river closest to a campground." Spatial aggregates are usually variants of the *Nearest Neighbor* (69–71) search problem: Given a query object, find the object having minimum distance from the query object. A *Reverse Nearest Neighbor* (RNN) (72–76) query is another example of a spatial aggregate. Given a query object, a RNN query finds objects for which the query object is the nearest neighbor. Applications of RNN include army strategic planning where a medical unit, *A*, in the battlefield is always in search of a wounded soldier for whom *A* is the nearest medical unit.

Visibility Queries. Visibility has been widely studied in computer graphics. Visibility may be defined as the parts of objects and the environment that are visible from a point in space. A visibility query can be thought of as a query that returns the objects and part of the environment visible at the querying point. For example, within a city, if the coverage area of a wireless antenna is considered to be the visible area, then the union of coverage areas of all the antennas in the city will provide an idea about the area that is not covered. Such information may be used to place a new antenna strategically at an optimal location. In a visibility query, if the point in space moves, then the area of visibility changes. Such a query may be called a continuous visibility query. For example, security for the president's motorcade involves cordoning off the buildings that have route visibility. In such a case, the visibility query may be thought of as a query that returns the buildings visible at different points on the route.

Visual Queryings. Many spatial applications present results visually, in the form of maps that consist of graphic images, 3-D displays, and animations. These applications allow users to query the visual representation by pointing to the visual representation using pointing devices such as a mouse or a pen. Such graphical interfaces are needed to query spatial data without the need by users to write any SQL statements. In recent years, map services, such as Google Earth and Microsoft Earth, have become very popular. more work is needed to explore the impact of querying by pointing and visual presentation of results on database performance.

Two-Step Query Processing of Spatial Operations. Because spatial query processing involves complex data types, a lake boundary might need a thousand vertices for exact representation. Spatial operations typically follow a two-step algorithm *(filter* and *refinement)* as shown in Fig. 8 to process complex spatial objects efficiently (77). Approximate geometry, such as the minimal orthogonal bounding rectangle of an extended spatial object, is first used to filter out many irrelevant objects quickly. Exact geometry then is used for the remaining spatial objects to complete the processing.

- **Filter step:** In this step, the spatial objects are represented by simpler approximations like the minimum bounding rectangle (MBR). For example, consider the following point query, "Find all rivers whose flood-plains overlap the SHRINE." In SQL this query will be:

```
SELECT   river. name
FROM     river
WHERE    overlap  (river.  flood-plain,  :
SHRINE)
```

  If we approximate the flood-plains of all rivers with MBRs, then it is less expensive to determine whether the point is in a MBR than to check whether a point is in an irregular polygon, that is, in the exact shape of the flood-plain. The answer from this approximate test is a superset of the real answer set. This superset is sometimes called the candidate set. Even the spatial predicate may be replaced by an approximation to simplify a query optimizer. For example, touch (river.flood-plain, :SHRINE) may be replaced by overlap(MBR(river.flood-plain, :SHRINE), and MBR(:SHRINE)) in the filter step. Many spatial operators, for example, inside, north-of, and buffer, can be approximated using the overlap relationship among corresponding MBRs. Such a transformation guarantees that no tuple from the final answer using exact geometry is eliminated in the filter step.

- **Refinement step:** Here, the exact geometry of each element from the candidate set and the exact spatial predicate is examined. This examination usually requires the use of a CPU-intensive algorithm. This step may sometimes be processed outside the spatial database in an application program such as GIS, using the candidate set produced by the spatial database in the filter step.

Techniques for Spatial Operations. This section presents several common operations between spatial objects: selection, spatial join, aggregates, and bulk loading.

Selection Operation. Similar to traditional database systems, the selection operation can be performed on indexed or non indexed spatial data. The difference is in the technique used to evaluate the predicate and the type of index. As discussed in the previous section, a two-step approach, where the geometry of a spatial object is approximated by a rectangle, is commonly used to evaluate a predicate. Popular indexing techniques for spatial data are R-tree, and space-filling curves. An R-tree is a height-balanced tree that is a natural extension of a B-tree for k-dimensions. It allows a point search to be processed in *O(log n)* time. Hash filling curves provide one-to-one continuous mappings that map points of multidimensional space into one-dimensional space. This mapping allows the user to impose order on higher-dimensional spaces. Common examples of space-filling curves are row-order Peano, Z-order, and Hilbert curves. Once the data has been ordered by a space-filling curve, a B-tree index can be imposed on the ordered entries to enhance the search. Point search operations can be performed in *O(log n)* time.

Spatial Join Operation. Conceptually, a join is defined as a cross product followed by a selection condition. In practice, this viewpoint can be very expensive because it involves materializing the cross product before applying the selection criterion. This finding is especially true for spatial databases. Many ingenious algorithms have been proposed to preempt the need to perform the cross product. The two-step query-processing technique described in the previous section is the most commonly used. With such methods, the spatial join operation can be reduced to a rectangle–rectangle intersection, the cost of which is relatively modest compared with the I/O cost of retrieving pages from secondary memory for processing.

A number of strategies have been proposed for processing spatial joins. Interested readers are encouraged to refer to Refs. 78 through 82.

Aggregate Operation: Nearest Neighbor, Reverse Nearest Neighbor. Nearest Neighbor queries are common in many applications. For example, a person driving on the road may want to find the nearest gas station from his current location. Various algorithms exist for nearest neighbor queries (69–71,83,84). Techniques based on Voronoi diagrams, Quad-tree indexing, and Kd-trees have been discussed in Ref. 27.

Reverse Nearest Neighbor queries were introduced in Ref. 72 in the context of decision support systems. For example, a RNN query can be used to find a set of customers who can be influenced by the opening of a new store-outlet location.

Bulk Loading. Bulk operations affect potentially a large set of tuples, unlike other database operations, such as insert into a relation, which affects possibly one tuple at a time. Bulk loading refers to the creation of an index from scratch on a potentially large set of data. Bulk loading has its advantages because the properties of the data set may be known in advance. These properties may be used to design efficiently the space-partitioning index structures commonly used for spatial data. An evaluation of generic bulk loading techniques is provided in Ref. 85.

Parallel GIS. A high-performance geographic information system (HPGIS) is a central component of many interactive applications like real-time terrain visualization, situation assessment, and spatial decision-making. The geographic information system (GIS) often contains large amounts of geometric and feature data (e.g., location, elevation, and soil type) represented as large sets of points, chains of line segments, and polygons. This data is often accessed via range queries. The existing sequential methods for supporting GIS operations do not meet the real-time requirements imposed by many interactive applications.

Hence, parallelization of GIS is essential for meeting the high performance requirements of several realtime applications. A GIS operation can be parallelized either by function partitioning (86–88) or by data partitioning (89–97). Function-partitioning uses specialized data structures (e.g., distributed data structures) and algorithms that may be different from their sequential counterparts. Data partitioning techniques divide the data among different processors and independently execute the sequential algorithm on each processor. Data partitioning in turn is achieved by declustering (98,99) the spatial data. If the static declustering methods fail to distribute the load equally among different processors, the load balance may be improved by redistributing parts of the data to idle processors using dynamic load-balancing (DLB) techniques.

Research Needs

This section presents the research needs for spatial query processing and query optimization.

Query Processing. Many open research areas exist at the logical level of query processing, including query-cost modeling and queries related to fields and networks. Cost

Table 6. Difficult spatial queries from GIS

Voronoize	Classify households as to which supermarket they are closest to
Network	Find the shortest path from the warehouse to all delivery stops
Timedependentnetwork	Find the shortest path where the road network is dynamic
Allocation	Where is the best place to build a new restaurant
Transformation	Triangulate a layer based on elevation
BulkLoad	Load a spatial data file into the database
Raster ↔ Vector	Convert between raster and vector representations
Visibility	Find all points of objects and environment visible from a point
EvacuationRoute	Find evacuation routes based on capacity and availability constraints
PredictLocation	Predict the location of a mobile person based on personal route patterns

models are used to rank and select the promising processing strategies, given a spatial query and a spatial data set. However, traditional cost models may not be accurate in estimating the cost of strategies for spatial operations, because of the distance metric and the semantic gap between relational operators and spatial operation. Comparison of the execution costs of such strategies required that new cost models be developed to estimate the selectivity of spatial search and join operations. Preliminary work in the context of the R-tree, tree-matching join, and fractal-model is promising (100,101), but more work is needed.

Many processing strategies using the overlap predicate have been developed for range queries and spatial join queries. However, a need exists to develop and evaluate strategies for many other frequent queries such as those listed in Table 6. These include queries on objects using predicates other than overlap, queries on fields such as slope analysis, and queries on networks such as the shortest path to a set of destinations.

Depending on the type of spatial data and the nature of the query, other research areas also need to be investigated. A *moving objects query* involves spatial objects that are mobile. Examples of such queries include "Which is the nearest taxi cab to the customer?", "Where is the hurricane expected to hit next?", and "What is a possible location of a serial criminal?" With the increasing availability of streaming data from GPS devices, continuous queries has become an active area of research. Several techniques (25,102,103) have been proposed to execute such queries.

A skyline query (104) is a query to retrieve a set of interesting points (records) from a potentially huge collection of points (records) based on certain attributes. For example, considering a set of hotels to be points, the skyline query may return a set of interesting hotels based on a user's preferences. The set of hotels returned for a user who prefers a cheap hotel may be different from the set of hotels returned for a user who prefers hotels that are closer to the coast. Research needed for skyline query operation includes computation of algorithms and processing for higher dimensions (attributes). Other query processing techniques in which research is required are querying on *3-D spatial data* and *spatio-temporal data.*

Query Optimization. The *query optimizer,* a module in database software, generates different evaluation plans and determines the appropriate execution strategy. Before the query optimizer can operate on the query, the high-level declarative statement must be scanned through a *parser.* The parser checks the syntax and transforms the statement into a *query tree.* In traditional databases, the data types and functions are fixed and the parser is relatively simple. Spatial databases are examples of an extensible database system and have provisions for user-defined types and methods. Therefore, compared with traditional databases, the parser for spatial databases has to be considerably more sophisticated to identify and manage user-defined data types and map them into syntactically correct query trees. In the query tree, the leaf nodes correspond to the relations involved and the internal nodes correspond to the basic operations that constitute the query. Query processing starts at the leaf nodes and proceeds up the tree until the operation at the root node has been performed.

Consider the query, "Find all lakes which have an area greater than 20 sq. km. and are within 50 km. from the campground." Let us assume that the *Area()* function is not precomputed and that its value is computed afresh every time it is invoked. A query tree generated for the query is shown in Fig. 9(a). In the classic situation, the rule "select before join" would dictate that the *Area* function be computed before the join predicate function, *Distance()* [Fig. 9(b)], the underlying assumption being that the computational cost of executing the select and join predicate is equivalent and negligible compared with the I/O cost of the operations. In the spatial situation, the relative cost per tuple of *Area()* and *Distance()* is an important factor in deciding the order of the operations (105). Depending the implementation of these two functions, the optimal strategy may be to process the join before the select operation [Fig. 9(c)]. This approach thus violates the main heuristic rule for relational databases, which states "Apply select and project before the join and binary operations" are no longer unconditional. A cost-based optimization technique exists to determine the optimal execution strategy from a set of execution plans. A quantitative analysis of spatial index structures is used to calculate the expected number of disk accesses that are required to perform a spatial query (106). Nevertheless, in spite of these advances, query optimization techniques for spatial data need more study.

SPATIAL FILE ORGANIZATION AND INDICES

Accomplishments

Space-Filling Curves. The physical design of a spatial database optimizes the instructions to storage devices for

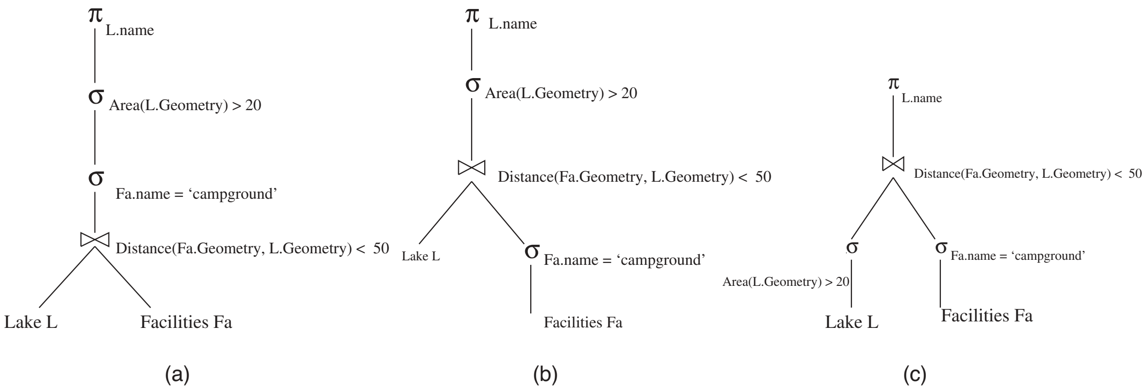


Figure 9. (a) Query tree, (b) "pushing down": select operation, and (c) "pushing down" may not help.

performing common operations on spatial data files. File designs for secondary storage include clustering methods and spatial hashing methods. Spatial clustering techniques are more difficult to design than traditional clustering techniques because no natural order exista in multidimensional space where spatial data resides. This situation is only complicated by the fact that the storage disk is a logical one-dimensional device. Thus, what is needed is a mapping from a higher-dimensional space to a one-dimensional space that is distance-preserving: This mapping ensures that elements that are close in space are mapped onto nearby points on the line and that no two points in the space are mapped onto the same point on the line (107). Several mappings, none of them ideal, have been proposed to accomplish this feat. The most prominent ones include row-order, Z-order, and the Hilbert-curve (Fig. 10).

Metric clustering techniques use the notion of distance to group nearest neighbors together in a metric space. Topological clustering methods like connectivity-clustered access methods (108) use the min-cut partitioning of a graph representation to support graph traversal operations efficiently. The physical organization of files can be supplemented with indices, which are data structures to improve the performance of search operations.

Classical one-dimensional indices such as the B^+-tree can be used for spatial data by linearizing a multidimensional space using a space-filling curve such as the Z-order. Many spatial indices (27) have been explored for multidimensional Euclidean space. Representative indices for point objects include grid files, multidimensional grid files (109), Point-Quad-Trees, and Kd-trees. Representative indices for extended objects include the R-tree family, the Field-tree, Cell-tree, BSP-tree, and Balanced and Nested grid files.

Grid Files. Grid files were introduced by Nievergelt (110). A grid file divides the space into n-dimensional spaces that can fit into equal-size buckets. The structures are not hierarchical and can be used to index static uniformly distributed data. However, because of its structure, the directory of a grid file can be so sparse and large that a large main memory is required. Several variations of grid files, exists to index data efficiently and to overcome these limitations (111,112). An overview of grid files is given in Ref. 27.

Tree indexes. *R-tree* aims to index objects in a hierarchical index structure (113). The R-tree is a height-balanced tree that is the natural extension of the B-tree for k-dimensions. Spatial objects are represented in the R-tree by their minimum bounding rectangle (MBR). Figure 11 illustrates spatial objects organized as an R-tree index. R-trees can be used to process both point and range queries.

Several variants of R-trees exist for better performance of queries and storage use. The R^+-tree (114) is used to store objects by avoiding overlaps among the MBRs, which increases the performance of the searching. R^*-trees (115) rely on the combined optimization of the area, margin, and overlap of each MBR in the intermediate nodes of the tree, which results in better storage use.

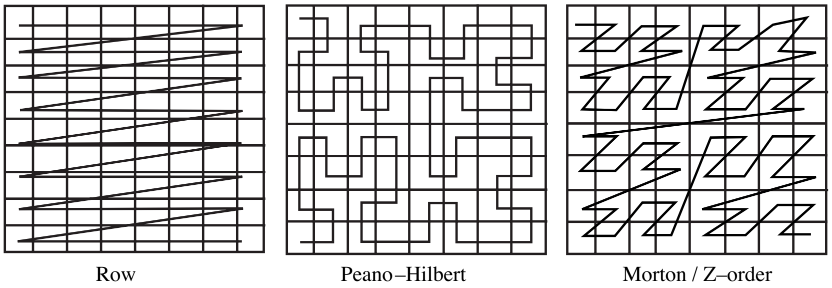


Figure 10. Space-filling curves to linearize a multidimensional space.

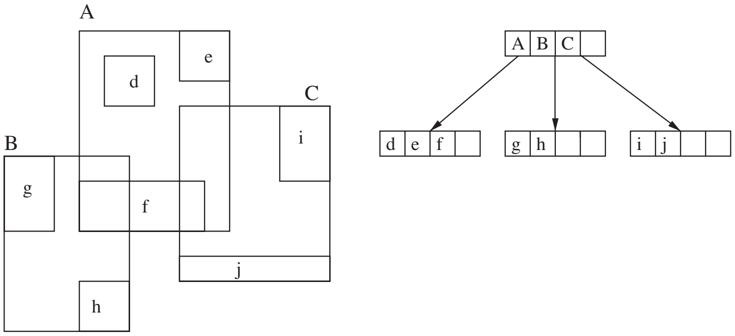


Figure 11. Spatial objects (d, e, f, g, h, and i) arranged in an R-tree hierarchy.

Many R-tree-based index structures (116–119,120,121) have been proposed to index spatio–temporal objects. A survey of spatio–temporal access methods has been provided in Ref. 122.

Quad tree(123) is a space-partitioning index structure in which the space is divided recursively into quads. This recursive process is implemented until each quad is homogeneous. Several variations of quad trees are available to store point data, raster data, and object data. Also, other quad tree structures exist to index spatio–temporal data sets, such as overlapping linear quad trees (24) and multiple overlapping features (MOF) trees (125).

The *Generalized Search Tree* (*GiST*) (126) provides a framework to build almost any kind of tree index on any kind of data. Tree index structures, such as B$^+$-tree and *R*-tree, can be built using GiST. A spatial-partitioning generalized search tree (SP-GiST) (127) is an extensible index structure for space-partitioning trees. Index trees such as quad tree and *k*d-tree can be built using SP-GiST.

Graph Indexes. Most spatial access methods provide methods and operators for point and range queries over collections of spatial points, line segments, and polygons. However, it is not clear if spatial access methods can efficiently support network computations that traverse line segments in a spatial network based on connectivity rather than geographic proximity. *A connectivity-clustered access method for spatial network* (*CCAM*) is proposed to index spatial networks based on graph partitioning (108) by supporting network operations. An auxiliary secondary index, such as B$^+$-tree, *R*-tree, and Grid File, is used to support network operations such as *Find*(), *get-a-Successor*(), and *get-Successors*().

Research Needs

Concurrency Control. The R-link tree (128) is among the few approaches available for concurrency control on the R-tree. New approaches for concurrency-control techniques are needed for other spatial indices. Concurrency is provided during operations such as search, insert, and delete. The R-link tree is also recoverable in a write-ahead logging environment. Reference 129 provides general algorithms for concurrency control for GiST that can also be applied to tree-based indexes. Research is required for concurrency control on other useful spatial data structures.

TRENDS: SPATIAL DATA MINING

Accomplishments

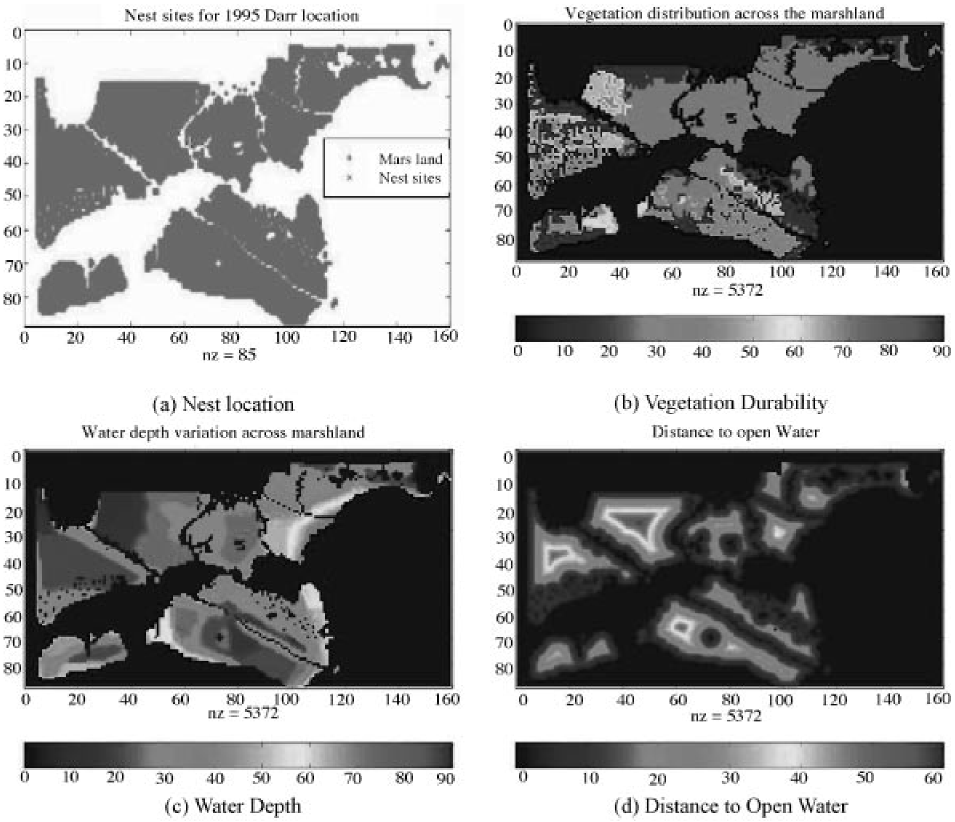
The explosive growth of spatial data and widespread use of spatial databases emphasize the need for the automated discovery of spatial knowledge. Spatial data mining is the process of discovering interesting and previously unknown, but potentially useful, patterns from spatial databases. Some applications are location-based services, studying the effects of climate, land-use classification, predicting the spread of disease, creating high-resolution three-dimensional maps from satellite imagery, finding crime hot spots, and detecting local instability in traffic. A detailed review of spatial data mining can be found in Ref. 130.

The requirements of mining spatial databases are different from those of mining classic relational databases. The difference between classic and spatial data mining parallels the difference between classic and spatial statistics. One fundamental assumption that guides statistical analysis is that the data samples are generated independently, as with successive tosses of a coin or the rolling of a die. When it comes to the analysis of spatial data, the assumption about the independence of samples is generally false. In fact, spatial data tends to be highly self-correlated. For example, changes in natural resources, wildlife, and temperature vary gradually over space. The notion of *spatial autocorrelation,* the idea that similar objects tend to cluster in geographic space, is unique to spatial data mining.

For detailed discussion of spatial analysis, readers are encouraged to refer to Refs. 131 and 132.

Spatial Patterns. This section presents several spatial patterns, specifically those related to location prediction, Markhov random fields, spatial clustering, spatial outliers, and spatial colocation.

Location Prediction. Location prediction is concerned with the discovery of a model to infer locations of a spatial phenomenon from the maps of other spatial features. For example, ecologists build models to predict habitats for endangered species using maps of vegetation, water bodies, climate, and other related species. Figure 12 shows the learning data set used in building a location-prediction


Nest sites for 1995 Darr location
Mars land
Nest sites
nz = 85
Vegetation distribution across the marshland
nz = 5372
Water depth variation across marshland
nz = 5372
Distance to open Water
nz = 5372
0
10
20
30
40
50
60
70
80
90
100
120
140
160

(a) Nest location

(b) Vegetation Durability

(c) Water Depth

(d) Distance to Open Water

Figure 12. (a) Learning data set: The geometry of the Darr wetland and the locations of the nests, (b) the spatial distribution of *vegetation durability* over the marshland, (c) the spatial distribution of *water depth*, and (d) the spatial distribution of *distance to open water*.

model for red-winged blackbirds in the Darr and Stubble wetlands on the shores of Lake Erie in Ohio. The data set consists of nest location, vegetation durability, distance to open water, and water depth maps. Spatial data mining techniques that capture the spatial autocorrelation (133,134) of nest location such as the spatial autoregression model (SAR) and markov random fields (MRF) are used for location-prediction modeling.

Spatial Autoregression Model. Linear regression models are used to estimate the conditional expected value of a dependent variable y given the values of other variables X. Such a model assumes that the variables are independent. The spatial autoregression model (131,135–137) is an extension of the linear regression model that takes spatial autocorrelation into consideration. If the dependent values y and X are related to each other, then the regression equation (138) can be modified as

$$y = \rho \mathrm{W} y + \mathrm{X}\beta + \varepsilon \tag{1}$$

Here, W is the neighborhood relationship contiguity matrix, and ? is a parameter that reflects the strength of the spatial dependencies between the elements of the dependent variable. Notice that when $\rho = 0$, this equation collapses to the linear regression model. If the spatial autocorrelation coefficient is statistically significant, then SAR will quantify the presence of spatial autocorrelation. In such a case, the spatial autocorrelation coefficient will indicate the extent to which variations in the dependent variable *(y)* are explained by the average of neighboring observation values.

Markov Random Field. Markov random field-based (139) Bayesian classifiers estimate the classification model, $\hat{f}c$, using MRF and Bayes' rule. A set of random variables whose interdependency relationship is represented by an undirected graph (i.e., a symmetric neighborhood matrix) is called a Markov random field. The Markov property specifies that a variable depends only on its neighbors and is independent of all other variables. The location prediction problem can be modeled in this framework by assuming that the class label, $l_i = f_C(s_i)$, of different locations, s_i, constitutes an MRF. In other words, random variable l_i is independent of l_i if $W(s_i, s_j) = 0$.

The Bayesian rule can be used to predict l_i from feature value vector X and neighborhood class label vector L_i as follows:

$$Pr(l_i|X, L_i) = \frac{Pr(X|l_i, L_i)Pr(l_i|L_i)}{Pr(X)} \tag{2}$$

The solution procedure can estimate $Pr(l_i \mid L_i)$ from the training data, where L_i denotes a set of labels in the neighborhood of s_i excluding the label at s_i. It makes this estimate by examining the ratios of the frequencies of class labels to the total number of locations in the spatial framework. $Pr(X \mid l_i, L_i)$ can be estimated using kernel functions from the observed values in the training data set.

A more detailed theoretical and experimental comparison of these methods can be found in Ref. 140. Although MRF and SAR classification have different formulations, they share a common goal of estimating the posterior probability distribution. However, the posterior probability for the two models is computed differently with different assumptions. For MRF, the posterior is computed using Bayes' rule, whereas in SAR, the posterior distribution is fitted directly to the data.

Spatial Clustering. Spatial clustering is a process of grouping a set of spatial objects into clusters so that objects within a cluster have high similarity in comparison with one another but are dissimilar to objects in other clusters.

For example, clustering is used to determine the "hot spots" in crime analysis and disease tracking. Many criminal justice agencies are exploring the benefits provided by computer technologies to identify crime hot spots to take preventive strategies such as deploying saturation patrols in hot-spot areas.

Spatial clustering can be applied to group similar spatial objects together; the implicit assumption is that patterns in space tend to be grouped rather than randomly located. However, the statistical significance of spatial clusters should be measured by testing the assumption in the data. One method to compute this measure is based on quadrats (i.e., well-defined areas, often rectangular in shape). Usually quadrats of random location and orientations in the quadrats are counted, and statistics derived from the counters are computed. Another type of statistics is based on distances between patterns; one such type is Ripley's K-function (141). After the verification of the statistical significance of the spatial clustering, classic clustering algorithms (142) can be used to discover interesting clusters.

Spatial Outliers. A spatial outlier (143) is a spatially referenced object whose nonspatial attribute values differ significantly from those of other spatially referenced objects in its spatial neighborhood. Figure 13 gives an example of detecting spatial outliers in traffic measurements for sensors on highway I-35W (North bound) for a 24-hour time period. Station 9 seems to be a spatial outlier as it exhibits inconsistent traffic flow as compared with its neighboring stations. The reason could be that the sensor at station 9 is malfunctioning. Detecting spatial outliers is useful in many applications of geographic information systems and spatial databases, including transportation, ecology, public safety, public health, climatology, and location-based services.

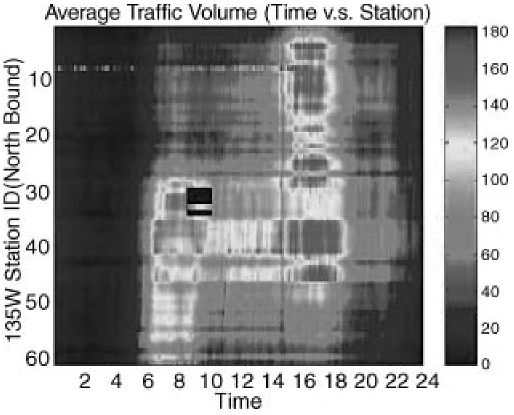


Figure 13. Spatial outlier (station ID 9) in traffic volume data.

Spatial attributes are used to characterize location, neighborhood, and distance. Nonspatial attribute dimensions are used to compare a spatially referenced object with its neighbors. Spatial statistics literature provides two kinds of bipartite multidimensional tests, namely graphical tests and quantitative tests. Graphical tests, which are based on the visualization of spatial data, highlight spatial outliers, for example, variogram clouds (141) and Moran scatterplots (144). Quantitative methods provide a precise test to distinguish spatial outliers from the remainder of data. A unified approach to detect spatial outliers efficiently is discussed in Ref. 145., Referance 146 provides algorithms for multiple spatial-outlier detection.

Spatial Colocation The colocation pattern discovery process finds frequently colocated subsets of spatial event types given a map of their locations. For example, the analysis of the habitats of animals and plants may identify the colocations of predator–prey species, symbiotic species, or fire events with fuel, ignition sources and so forth. Figure 14 gives an example of the colocation between roads and rivers in a geographic region.

Approaches to discovering colocation rules can be categorized into two classes, namely spatial statistics and datamining approaches. Spatial statistics-based approaches use measures of spatial correlation to characterize the relationship between different types of spatial features. Measures of spatial correlation include the cross K-function with Monte Carlo simulation, mean nearest-neighbor distance, and spatial regression models.

Data-mining approaches can be further divided into transaction-based approaches and distance-based approaches. Transaction-based approaches focus on defining transactions over space so that an a priori-like algorithm can be used. Transactions over space can be defined by a reference-feature centric model. Under this model, transactions are created around instances of one user-specified spatial feature. The association rules are derived using the a priori (147) algorithm. The rules formed are related to the reference feature. However, it is nontrivial to generalize the paradigm of forming rules related to a reference feature to the case in which no reference feature is specified. Also, defining transactions around locations of instances of all features may yield duplicate counts for many candidate associations.

In a distance-based approach (148–150), instances of objects are grouped together based on their Euclidean distance from each other. This approach can be considered to be an event-centric model that finds subsets of spatial

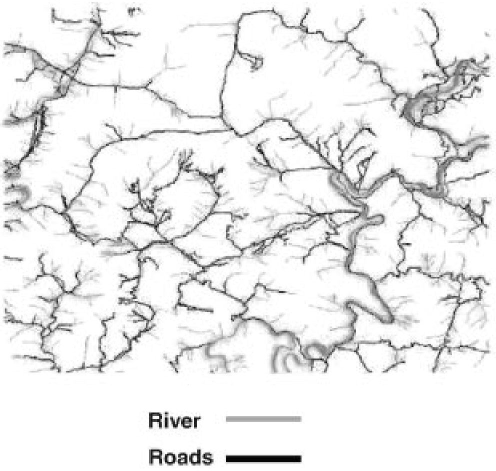


Figure 14. Colocation between roads and rivers in a hilly terrain (Courtesy: Architecture Technology Corporation).

features likely to occur in a neighborhood around instances of given subsets of event types.

Research Needs

This section presents several research needs in the area of spatio–temporal data mining and spatial–temporal network mining.

Spatio–Temporal Data Mining. Spatio–temporal (ST) data mining aims to develop models and objective functions and to discover patterns that are more suited to Spatio–temporal databases and their unique properties (15). An extensive survey of Spatio–temporal databases, models, languages, and access methods can be found in Ref. 152. A bibliography of Spatio–temporal data mining can be found in Ref. 153.

Spatio–temporal pattern mining focuses on discovering knowledge that frequently is located together in space and time. Referances 154, 155, and 156 defined the problems of discovering mixed-drove and sustained emerging Spatio–temporal co-occurrence patterns and proposed interest measures and algorithms to mine such patterns. Other research needs include conflation, in which a single feature is obtained from several sources or representations. The goal is to determine the optimal or best representation based on a set of rules. Problems tend to occur during maintenance operations and cases of vertical obstruction.

In several application domains, such as sensor networks, mobile networks, moving object analysis, and image analysis, the need for spatio–temporal data mining is increasing drastically. It is vital to develop new models and techniques, to define new spatio–temporal patterns, and to formulize monotonic interest measures to mine these patterns (157).

Spatio–Temporal Network Mining. In the post-9/11 world of asymmetric warfare in urban area, many human activities are centered about ST infrastructure networks, such as transportation, oil/gas-pipelines, and utilities (e.g., water, electricity, and telephone). Thus, activity reports, for example, crime/insurgency reports, may often use network-based location references, for exmple, street address such as “200 Quiet Street, Scaryville, RQ 91101.” In addition, spatial interaction among activities at nearby locations may be constrained by network connectivity and network distances (e.g., shortest path along roads or train networks) rather than geometric distances (e.g., Euclidean or Manhattan distances) used in traditional spatial analysis. Crime prevention may focus on identifying subsets of ST networks with high activity levels, understanding underlying causes in terms of ST-network properties, and designing ST-network-control policies.

Existing spatial analysis methods face several challenges (e.g., see Ref. 158). First, these methods do not model the effect of explanatory variables to determine the locations of network hot spots. Second, existing methods for network pattern analysis are computationally expensive. Third, these methods do not consider the temporal aspects of the activity in the discovery of network patterns. For example, the routes used by criminals during the day and night may differ. The periodicity of bus/train schedules can have an impact on the routes traveled. Incorporating the time-dependency of transportation networks can improve the accuracy of the patterns.

SUMMARY

In this chapter we presented the major research accomplishments and techniques that have emerged from the area of spatial databases in the past decade. These accomplishments and techniques include spatial database modeling, spatial query processing, and spatial access methods. We have also identified areas in which more research is needed, such as spatio–temporal databases, spatial data mining, and spatial networks.

Figure 15 provides a summary of topics that continue to drive the research needs of spatial database systems. Increasingly available spatial data in the form of digitized maps, remotely sensed images, Spatio–temporal data (for example, from videos), and streaming data from sensors have to be managed and processed efficiently. New ways of querying techniques to visualize spatial data in more than one dimension are needed. Several advances have been made in computer hardware over the last few years, but many have yet to be fully exploited, including increases in main memory, more effective storage using storage area networks, greater availability of multicore processors, and powerful graphic processors. A huge impetus for these advances has been spatial data applications such as land navigation systems and location-based services. To measure the quality of spatial database systems, new benchmarks have to be established. Some benchmarks (159,160) established earlier have become dated. Newer benchmarks are needed to characterize the spatial data management needs of other systems and applications such as Spatio–temporal databases, moving-objects databases, and location-based services.

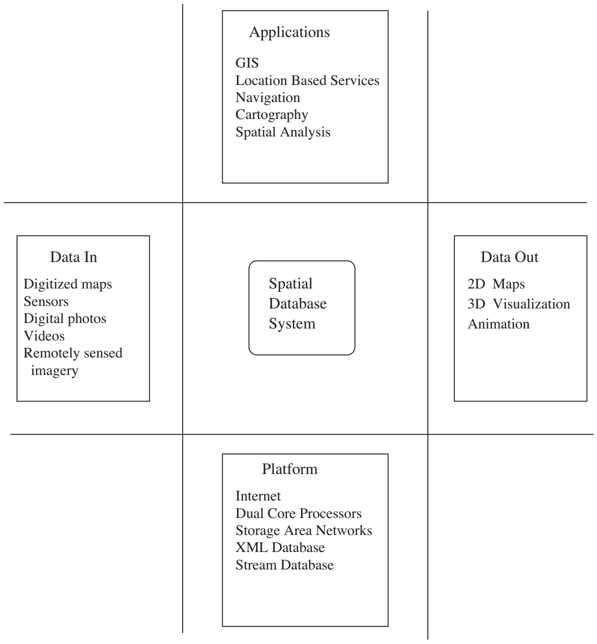


Figure 15. Topics driving future research needs in spatial database systems.

ACKNOWLEDGMENTS

We thank the professional organizations that have funded the research on spatial databases, in particular, the National Science Foundation, Army Research Laboratory, Topographic Engineering Center, Oak Ridge National Laboratory, Minnesota Department of Transportation, and Microsoft Corporation. We thank members of the spatial database and spatial data mining research group at the University of Minnesota for refining the content of this chapter. We also thank Kim Koffolt for improving the readability of this chapter.

BIBLIOGRAPHY

1. R. H. Guting, An introduction to spatial database systems, *VLDB Journal, Special Issue on Spatial Database Systems*, **3**(4): 357–399, 1994.
2. W. Kim, J. Garza, and A. Kesin, Spatial data management in database systems, in *Advances in Spatial Databases, 3rd International Symposium, SSD'93*, Vol. **652**. Springer, 1993.
3. Y. Manolopoulos, A. N. Papadopoulos, and M. G. Vassilakopoulos, *Spatial Databases: Technologies, Techniques and Trends*. Idea Group Publishing, 2004.
4. S. Shekhar and S. Chawla, *Spatial Databases: A Tour*. Prentice Hall, 2002.
5. Wikipedia. Available: http://en.wikipedia.org/wiki/Spatial_Database, 2007.
6. M. Worboys and M. Duckham, *GIS: A Computing Perspective*, 2nd ed. CRC, 2004.
7. J. Schiller, *Location-Based Services*. Morgan Kaufmann, 2004.
8. A. Stefanidis and S. Nittel, *GeoSensor Networks*. CRC, 2004.
9. R. Scally, *GIS for Environmental Management*. ESRI Press, 2006.
10. L. Lang, *Transportation GIS*, ESRI Press, 1999.
11. P. Elliott, J. C. Wakefield, N. G. Best, and D. J. Briggs, *Spatial Epidemiology: Methods and Applications*. Oxford University Press, 2000.
12. M. R. Leipnik and D. P. Albert, *GIS in Law Enforcement: Implementation Issues and Case Studies*, CRC, 2002.
13. D. K. Arctur and M. Zeiler, *Designing Geodatabases*, ESRI Press, 2004.
14. E. Beinat, A. Godfrind, and R. V. Kothuri, *Pro Oracle Spatial*, Apress, 2004.
15. SQL Server 2008 (Code-name Katmai). Available: http://www.microsoft.com/sql/prodinfo/futureversion/default.mspx, 2007.
16. Google Earth. Available: http://earth.google.com, 2006.
17. Microsoft Virtual Earth. Available: http://www.microsoft.com/virtualearth, 2006.
18. PostGIS. Available: http://postgis.refractions.net/, 2007.
19. MySQL Spatial Extensions. Available: http://dev.mysql.com/doc/refman/5.0/en/spatial-extensions.html, 2007.
20. Sky Server, 2007. Available: http://skyserver.sdss.org/.
21. D. Chamberlin, Using the New DB2: IBM's Object Relational System, *Morgan Kaufmann*, 1997.
22. M. Stonebraker and D. Moore, Object Relational DBMSs: The Next Great Wave. Morgan Kaufmann, 1997.
23. OGC. Available: http://www.opengeospatial.org/standards, 2007.
24. P. Rigaux, M. Scholl, and A. Voisard, *Spatial Databases: With Application to GIS*, Morgan Kaufmann Series in Data Management Systems, 2000.
25. R. H. Guting and M. Schneider, *Moving Objects Databases (The Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2005.
26. S. Shekhar and H. Xiong, *Encyclopedia of GIS*, Springer, 2008, forthcoming.
27. H. Samet, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann Publishers, 2006.
28. ACM Geographical Information Science Conference. Available: http://www.acm.org.
29. ACM Special Interest Group on Management Of Data. Available: http://www.sigmod.org/.
30. Geographic Information Science Center Summer and Winter Assembly. Available: http://www.gisc.berkeley.edu/.
31. Geoinformatica. Available: http://www.springerlink.com/content/100268/.
32. IEEE International Conference on Data Engineering. Available: http://www.icde2007.org/icde/.
33. IEEE Transactions on Knowledge and Data Engineering (TKDE). Available: http://www.computer.org/tkde/.
34. International Journal of Geographical Information Science. Available: http://www.tandf.co.uk/journals/tf/13658816.html.
35. International Symposium on Spatial and Temporal Databases. Available: http://www.cs.ust.hk/ sstd07/.
36. Very Large Data Bases Conference. Available: http://www.vldb2007.org/.
37. UCGIS, 1998.
38. S. Shekhar, R. R. Vatsavai, S. Chawla, and T. E. Burke, Spatial Pictogram Enhanced Conceptual Data Models and Their Translations to Logical Data Models, *Integrated*

Spatial Databases: Digital Images and GIS, Lecture Notes in Computer Science, **1737**: 77–104, 1999.

39. N. R. Adam and A. Gangopadhyay, *Database Issues in Geographic Information Systems*, Norwell, MA: Kluwer Academic Publishers, 1997.
40. M. Stonebraker and G. Kemnitz, The Postgres Next Generation Database Management System, *Commun. ACM*, **34**(10): 78–92, 1991.
41. P. Bolstad, *GIS Fundamentals: A First Text on Geographic Information Systems*, 2nd ed. Eider Press, 2005.
42. T. Hadzilacos and N. Tryfona, An Extended Entity-Relationship Model for Geographic Applications, *ACM SIGMOD Record*, **26**(3): 24–29, 1997.
43. S. Shekhar, R. R. Vatsavai, S. Chawla, and T. E. Burk, Spatial pictogram enhanced conceptual data models and their translation to logical data models, *Lecture Notes in Computer Science*, **1737**: 77–104, 2000.
44. F. T. Fonseca and M. J. Egenhofer, Ontology-driven geographic information systems, in Claudia Bauzer Medeiros (ed.), *ACM-GIS '99, Proc. of the 7th International Symposium on Advances in Geographic Information Systems*, *Kansas City*, US 1999, pp. 14–19. ACM, 1999.
45. Time ontology in owl, *Electronic*, September 2005.
46. H. J. Berners-Lee, T. Lassila, and O. Lassila, The semantic web, *The Scientific American*, 2001, pp. 34–43.
47. M. J. Egenhofer, Toward the semantic geospatial web, *Proc. Tenth ACM International Symposium on Advances in Geographic Information Systems*, 2002.
48. F. Fonseca and M. A. Rodriguez, From geo-pragmatics to derivation ontologies: New directions for the geospatial semantic web, *Transactions in GIS*, **11**(3), 2007.
49. W3C. Resource Description Framework. Available: *http://www.w3.org/RDF/*,2004.
50. G. Subbiah, A. Alam, L. Khan, B. Thuraisingham, An integrated platform for secure geospatial information exchange through the semantic web, *Proc. ACM Workshop on Secure Web Services (SWS)*, 2006.
51. Open Geospatial Consortium Inc. OpenGIS Reference Model. Available: *http://orm.opengeospatial.org/*, 2006.
52. S. Shekhar and X. Liu, Direction as a Spatial Object: A Summary of Results, in R. Laurini, K. Makki, and N. Pissinou, (eds.), *ACM-GIS '98, Proc. 6th international symposium on Advances in Geographic Information Systems*. ACM, 1998, pp. 69–75.
53. S. Skiadopoulos, C. Giannoukos, N. Sarkas, P. Vassiliadis, T. Sellis, and M. Koubarakis, Computing and managing cardinal direction relations, *IEEE Trans. Knowledge and Data Engineering*, **17**(12): 1610–1623, 2005.
54. *A Spatiotemporal Model and Language for Moving Objects on Road Networks*, 2001.
55. *Modeling and querying moving objects in networks*, volume **15**, 2006.
56. *Modeling and Querying Moving Objects*, 1997.
57. *On Moving Object Queries*, 2002.
58. K. K. L. Chan and C. D. Tomlin, Map Algebra as a Spatial Language, in D. M. Mark and A. U. Frank, (eds.), *Cognitive and Linguistic Aspects of Geographic Space*. Dordrecht, Netherlands: Kluwer Academic Publishers, 1991, pp. 351–360.
59. W. Kainz, A. Riedl, and G. Elmes, eds, *A Tetrahedronized Irregular Network Based DBMS Approach for 3D Topographic Data*, Springer Berlin Heidelberg, September 2006.
60. C. Arens, J. Stoter, and P. Oosterom, Modeling 3D Spatial Objects in a Geo-DBMS Using a 3D Primitive, *Computers and Geosciences*, **31**(2): 165–177, act 2005.
61. E. Köhler, K. Langkau, and M. Skutella, Time-expanded graphs for flow-dependent transit times, in *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*. London, UK: Springer-Verlag, 2002, pp. 599–611.
62. B. George and S. Shekhar, Time-aggregated graphs for modeling spatio–temporal networks, in *ER (Workshops)*, 2006, pp. 85–99.,
63. K. Eickhorst, P. Agouris, and A. Stefanidis, Modeling and Comparing Spatiotemporal Events, in *Proc. 2004 annual national conference on Digital government research*. Digital Government Research Center, 2004, pp. 1–10.
64. Geographic Markup Language. Available: http://www.opengis.net/gml/, 2007.
65. S. Shekhar, R. R. Vatsavai, N. Sahay, T. E. Burk, and S. Lime, WMS and GML based Interoperable Web Mapping System, in *9th ACM International Symposium on Advances in Geographic Information Systems, ACMGIS01*. ACM, November 2001.
66. CityGML, 2007. Available: http://www.citygml.org/.
67. Keyhole Markup Language. Available: http://code.google.-com/apis/kml/documentation/, 2007.
68. V. Gaede and O. Gunther, Multidimensional access methods, *ACM Computing Surveys*, **30**, 1998.
69. G. R. Hjaltason and H. Samet, Ranking in spatial data–bases, in *Symposium on Large Spatial Databases*, 1995, pp. 83–95.
70. N. Roussopoulos, S. Kelley, and F. Vincent, Nearest neighbor queries, in *SIGMOD '95: Proc. 1995 ACM SIGMOD international conference on Management of data*, New York: ACM Press, 1995, pp. 71–79.
71. D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, Aggregate nearest neighbor queries in spatial databases, *ACM Trans. Database Systems*, **30**(2): 529–576, 2005.
72. F. Korn and S. Muthukrishnan, Influence Sets Based on Reverse Nearest Neighbor Queries, in *Proc. ACM International Conference on Management of Data, SIGMOD*, 2000, pp. 201–212.
73. J. M. Kang, M. Mokbel, S. Shekhar, T. Xia, and D. Zhang, Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors, in *Proc. IEEE 23rd International Conference on Data Engineering (ICDE)*, 2007.
74. I. Stanoi, D. Agrawal, and A. ElAbbadi, Reverse Nearest Neighbor Queries for Dynamic Databases, in *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000, pp. 44–53.
75. A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos, C2P: Clustering based on Closest Pairs, in *Proc. International Conference on Very Large Data Bases, VLDB*, 2001, pp. 331–340.
76. T. Xia and D. Zhang, Continuous Reverse Nearest Neighbor Monitoring, in *Proc. International Conference on Data Engineering, ICDE*, 2006.
77. T. Brinkoff, H.-P. Kriegel, R. Schneider, and B. Seeger, Multistep processing of spatial joins, In *Proc. ACM International Conference on Management of Data, SIGMOD*, 1994, pp. 197–208.
78. L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J.S. Vitter, Scalable sweeping-based spatial join, in *Proc. Very Large Data Bases (VLDB)*, 1998, pp. 570–581.

79. N. Mamoulis and D. Papadias, Slot index spatial join, *IEEE Trans. Knowledge and Data Engineering*, **15**(1): 211–231, 2003.

80. M.-J. Lee, K.-Y. Whang, W.-S. Han, and I.-Y. Song, Transform-space view: Performing spatial join in the transform space using original-space indexes, *IEEE Trans. Knowledge and Data Engineering*, **18**(2): 245–260, 2006.

81. S. Shekhar, C.-T. Lu, S. Chawla, and S. Ravada, Efficient join-index-based spatial-join processing: A clustering approach, in *IEEE Trans. Knowledge and Data Engineering*, **14**(6): 1400–1421, 2002.

82. M. Zhu, D. Papadias, J. Zhang, and D. L. Lee, Top-k spatial joins, *IEEE Trans. Knowledge and Data Engineering*, **17**(4): 567–579, 2005.

83. H. Hu and D. L. Lee, Range nearest-neighbor query, *IEEE Trans. Knowledge and Data Engineering*, **18**(1): 78–91, 2006.

84. M. L. Yiu, N. Mamoulis, and D. Papadias, Aggregate nearest neighbor queries in road networks, *IEEE Trans. Knowledge and Data Engineering*, **17**(6): 820–833, 2005.

85. J. van den Bercken and B. Seeger, An evaluation of generic bulk loading techniques, in *VLDB '01: Proc. 27th International Conference on Very Large Data Bases*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2001, pp. 461–470.

86. A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, and C. Yap, Parallel computational geometry. *Proc. 25th IEEE Symposium on Foundations of Computer Science*, 1985, pp. 468–477.

87. S. G. Akl and K. A. Lyons, *Parallel Computational Geometry*, Englewood Cliffs, NJ: Prentice Hall, 1993.

88. R. Sridhar, S. S. Iyengar, and S. Rajanarayanan, Range search in parallel using distributed data structures, *International Conference on Databases, Parallel Architectures, and Their Applications*, 1990, pp. 14–19.

89. M. P. Armstrong, C. E. Pavlik, and R. Marciano, Experiments in the measurement of spatial association using a parallel supercomputer, *Geographical Systems*, **1**: 267–288, 1994.

90. G. Brunetti, A. Clematis, B. Falcidieno, A. Sanguineti, and M. Spagnuolo, Parallel processing of spatial data for terrain characterization, *Proc. ACM Geographic Information Systems*, 1994.

91. W. R. Franklin, C. Narayanaswami, M. Kankanahalli, D. Sun, M. Zhou, and P. Y. F. Wu, Uniform grids: A technique for intersection detection on serial and parallel machines, *Proc. 9th Automated Cartography*, 1989, pp. 100–109.

92. E. G. Hoel and H. Samet, Data Parallel RTree Algorithms, *Proc. International Conference on Parallel Processing*, 1993.

93. E. G. Hoel and H. Samet, Performance of dataparallel spatial operations, *Proc. of the 20th International Conference on Very Large Data Bases*, 1994, pp. 156–167.

94. V. Kumar, A. Grama, and V. N. Rao, Scalable load balancing techniques for parallel computers, *J. Parallel and Distributed Computing*, **22**(1): 60–69, July 1994.

95. F. Wang, A parallel intersection algorithm for vector polygon overlay, *IEEE Computer Graphics and Applications*, **13**(2): 74–81, 1993.

96. Y. Zhou, S. Shekhar, and M. Coyle, Disk allocation methods for parallelizing grid files, *Proc. of the Tenth International Conference on Data Engineering, IEEE*, 1994, pp. 243–252.

97. S. Shekhar, S. Ravada, V. Kumar, D. Chubband, and G. Turner, Declustering and load-balancing methods for parallelizing spatial databases, *IEEE Trans. Knowledge and Data Engineering*, **10**(4): 632–655, 1998.

98. M. T. Fang, R. C. T. Lee, and C. C. Chang, The idea of de-clustering and its applications, *Proc. of the International Conference on Very Large Data Bases*, 1986, pp. 181–188.

99. D. R. Liu and S. Shekhar, A similarity graph-based approach to declustering problem and its applications, *Proc. of the Eleventh International Conference on Data Engineering, IEEE*, 1995.

100. A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension, in *Proc. 21st International Conference on Very Large Data Bases, VLDB*, 1995, pp. 299–310.

101. Y. Theodoridis, E. Stefanakis, and T. Sellis, Cost models for join queries in spatial databases, in *Proceedings of the IEEE 14th International Conference on Data Engineering*, 1998, pp. 476–483.

102. M. Erwig, R. Hartmut Guting, M. Schneider, and M. Vazirgiannis, Spatio–temporal data types: An approach to modeling and querying moving objects in databases, *GeoInformatica*, **3**(3): 269–296, 1999.

103. R. H. Girting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis, A foundation for representing and querying moving objects, *ACM Transactions on Database Systems*, **25**(1): 1–42, 2000.

104. S. Borzsonyi, D. Kossmann, and K. Stocker, The skyline operator, In *Proc. the International Conference on Data Engineering*, Heidelberg, Germany, 2001, pp. 421–430.

105. J. M. Hellerstein and M. Stonebraker, Predicate migration: Optimizing queries with expensive predicates, In *Proc. ACM-SIGMOD International Conference on Management of Data*, 1993, pp. 267–276.

106. Y. Theodoridis and T. Sellis, A model for the prediction of r-tree performance, in *Proceedings of the 15th ACM Symposium on Principles of Database Systems PODS Symposium*, ACM, 1996, pp. 161–171.

107. T. Asano, D. Ranjan, T. Roos, E. Wiezl, and P. Widmayer, Space-filling curves and their use in the design of geometric data structures, *Theoretical Computer Science*, **181**(1): 3–15, July 1997.

108. S. Shekhar and D.R. Liu, A connectivity-clustered access method for networks and network computation, *IEEE Trans. Knowledge and Data Engineering*, **9**(1): 102–119, 1997.

109. J. Lee, Y. Lee, K. Whang, and I. Song, A physical database design method for multidimensional file organization, *Information Sciences*, **120**(1): 31–65(35), November 1997.

110. J. Nievergelt, H. Hinterberger, and K. C. Sevcik, The grid file: An adaptable, symmetric multikey file structure, *ACM Trancsactions on Database Systems*, **9**(1): 38–71, 1984.

111. M. Ouksel, The interpolation-based grid file, *Proc. of Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1985, pp. 20–27.

112. K. Y. Whang and R. Krishnamurthy, *Multilevel grid files, IBM Research Laboratory Yorktown, Heights, NY*, 1985.

113. A. Guttman, R-trees: A Dynamic Index Structure for Spatial Searching, *Proc. of SIGMOD International Conference on Management of Data*, 1984, pp. 47–57.

114. T. Sellis, N. Roussopoulos, and C. Faloutsos, The R$^+$-tree: A dynamic index for multidimensional objects, *Proc. 13th International Conference on Very Large Data Bases*, September 1987, pp. 507–518.

115. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, *Proc. ACM SIGMOD International Conference on Management of Data*, 1990, pp. 322–331.

116. Y. Theodoridis, M. Vazirgiannis, and T. Sellis, Spatio–temporal indexing for large multimedia applications, *International Conference on Multimedia Computing and Systems*, 1996, pp. 441–448.

117. S. Saltenis and C.S. Jensen, R-tree based indexing of general Spatio–temporal data, *Technical Report TR-45 and Chorochronos CH-99-18, TimeCenter*, 1999.

118. M. Vazirgiannis, Y. Theodoridis, and T. Sellis, Spatio–temporal composition and indexing large multimedia applications, *Multimedia Systems*, **6**(4): 284–298, 1998.

119. M. Nascimiento, R. Jefferson, J. Silva, and Y. Theodoridis, Evaluation of access structures for discretely moving points, *Proc. International Workshop on Spatio–temporal Database Management*, 1999, pp. 171–188.

120. S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, Indexing the positions of continuously moving objects, in *SIGMOD Conference*, 2000, pp. 331–342.

121. Y. Tao, D. Papadias, and J. Sun, The TPR*-Tree: An Optimized Spatio–temporal Access Method for Predictive Queries, in *VLDB*, 2003, pp. 790–801.

122. M. F. Mokbel, T. M. Ghanem, and W. G. Aref, Spatio-temporal access methods, *IEEE Data Engineering Bulletin*, **26**(2): 40–49, 2003.

123. R. A. Finkel and J. L. Bentley, Quad trees: A data structure for retrieval on composite keys, *Acta Informatica*, **4**:1–9, 1974.

124. T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos, Overlapping linear quadtrees: A Spatio–temporal access method, *ACM-Geographic Information Systems*, 1998, pp. 1–7.

125. Y. Manolopoulos, E. Nardelli, A. Papadopoulos, and G. Proietti, MOF-Tree: A Spatial Access Method to Manipulate Multiple Overlapping Features, *Information Systems*, **22**(9): 465–481, 1997.

126. J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, Generalized Search Trees for Database System, *Proc. 21th International Conference on Very Large Database Systems*, September 11–15 1995.

127. W. G. Aref and I. F. Ilyas. SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees, *J. Intell. Inf. Sys.*, **17**(2–3): 215–240, 2001.

128. M. Kornacker and D. Banks, High-Concurrency Locking in R-Trees, 1995.

129. M. Kornacker, C. Mohan, and Joseph M. Hellerstein, Concurrency and recovery in generalized search trees, in *SIGMOD '97: Proc. 1997 ACM SIGMOD international conference on Management of data*. New York, ACM Press, 1997, pp. 62–72.

130. S. Shekhar, P. Zhang, Y. Huang, and R. R. Vatsavai, Spatial data mining, in Hillol Kargupta and A. Joshi, (eds.), *Book Chapter in Data Mining: Next Generation Challenges and Future Directions*.

131. N. A. C. Cressie, *Statistics for Spatial Data*. New York: Wiley-Interscience, 1993.

132. R. Haining, *Spatial Data Analysis : Theory and Practice*, Cambridge University Press, 2003.

133. Y. Jhung and P. H. Swain, Bayesian Contextual Classification Based on Modified M-Estimates and Markov Random Fields, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **34**(1): 67–75, 1996.

134. A. H. Solberg, T. Taxt, and A. K. Jain, A Markov Random Field Model for Classification of Multisource Satellite Imagery, *IEEE Trans. Geoscience and Remote Sensing*, **34**(1): 100–113, 1996.

135. D.A. Griffith, *Advanced Spatial Statistics*. Kluwer Academic Publishers, 1998.

136. J. LeSage, *Spatial Econometrics*. Available: http://www.spatial-econometrics.com/, 1998.

137. S. Shekhar, P. Schrater, R. Raju, and W. Wu, Spatial contextual classification and prediction models for mining geospatial data, *IEEE Trans. Multimedia*, **4**(2): 174–188, 2002.

138. L. Anselin, *Spatial Econometrics: methods and models*, Dordrecht, Netherlands: Kluwer, 1988.

139. S.Z. Li, A Markov Random Field Modeling, *Computer Vision*. Springer Verlag, 1995.

140. S. Shekhar et al. Spatial Contextual Classification and Prediction Models for Mining Geospatial Data, *IEEE Transaction on Multimedia*, **4**(2), 2002.

141. N. A. Cressie, *Statistics for Spatial Data (revised edition)*. New York: Wiley, 1993.

142. J. Han, M. Kamber, and A. Tung, *Spatial Clustering Methods in Data Mining: A Survey, Geographic Data Mining and Knowledge Discovery*. Taylor and Francis, 2001.

143. V. Barnett and T. Lewis, *Outliers in Statistical Data*, 3rd ed. New York: John Wiley, 1994.

144. A. Luc, Local Indicators of Spatial Association: LISA, *Geographical Analysis*, **27**(2): 93–115, 1995.

145. S. Shekhar, C.-T. Lu, and P. Zhang, A unified approach to detecting spatial outliers, *GeoInformatica*, **7**(2), 2003.

146. C.-T. Lu, D. Chen, and Y. Kou, Algorithms for Spatial Outlier Detection, *IEEE International Conference on Data Mining*, 2003.

147. R. Agrawal and R. Srikant, Fast algorithms for mining association rules, in J. B. Bocca, M. Jarke, and C. Zaniolo (eds.), *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, Morgan Kaufmann, 1994, pp. 487–499.

148. Y. Morimoto, Mining Frequent Neighboring Class Sets in Spatial Databases, in *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.

149. S. Shekhar and Y. Huang, Co-location Rules Mining: A Summary of Results, *Proc. Symposium on Spatial and Spatio–temporal Databases*, 2001.

150. Y. Huang, S. Shekhar, and H. Xiong, Discovering Co-location Patterns from Spatial Datasets: A General Approach, *IEEE Trans. Knowledge and Data Engineering*, **16**(12): 1472–1485, December 2005.

151. J. F. Roddick and B. G. Lees, *Paradigms for spatial and Spatio–temporal data mining*. Taylor and Frances, 2001.

152. M. Koubarakis, T. K. Sellis, A. U. Frank, S. Grumbach, R. Hartmut Guting, C. S. Jensen, N. A. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, B. Theodoulidis, and N. Tryfona, eds, *Spatio–temporal Databases: The CHOROCHRONOS Approach*, Vol. **2520** of *Lecture Notes in Computer Science*. Springer, 2003.

153. J. F. Roddick, K. Hornsby, and M. Spiliopoulou, An updated bibliography of temporal, spatial, and Spatio–temporal data mining research, *Proc. First International Workshop on Temporal, Spatial and Spatio–temporal Data Mining*, 2001, pp. 147–164.

154. M. Celik, S. Shekhar, J. P. Rogers, and J. A. Shine, Sustained emerging Spatio–temporal co-occurrence pattern mining: A summary of results, *18th IEEE International Conference on Tools with Artificial Intelligence*, 2006, pp. 106–115.

155. M. Celik, S. Shekhar, J. P. Rogers, J. A. Shine, and J. S. Yoo, Mixed-drove Spatio–temporal co-occurrence pattern mining: A summary of results, *Sixth International Conference on Data Mining, IEEE*, 2006, pp. 119–128.

156. M. Celik, S. Shekhar, J. P. Rogers, J. A. Shine, and J. M. Kang, Mining at most top-k mixed-drove Spatio–temporal co-occurrence patterns: A summary of results, in *Proc. of the Workshop on Spatio–temporal Data Mining (In conjunction with ICDE 2007*), 2008, forthcoming.

157. J. F. Roddick, E. Hoel, M. J. Egenhofer, D. Papadias, and B. Salzberg, Spatial, temporal and Spatio–temporal databases - hot issues and directions for phd research, *SIGMOD record*, **33**(2), 2004.

158. O. Schabenberger and C. A. Gotway, *Statistical Methods for Spatial Data Analysis*. Chapman & Hall/CRC, 2004.

159. M. Stonebraker, J. Frew, K. Gardels, and J. Meredith, The Sequoia 2000 Benchmark, in Peter Buneman and Sushil Jajodia (eds.), *Proc. 1993 ACM SIGMOD International Conference on Management of Data*. ACM Press, 1993. pp. 2–11.

160. J. M. Patel, J.-B. Yu, N. Kabra, K. Tufte, B. Nag, J. Burger, N. E. Hall, K. Ramasamy, R. Lueder, C. Ellmann, J. Kupsch, S. Guo, D. J. DeWitt, and J. F. Naughton, Building a scaleable geo-spatial dbms: Technology, implementation, and evaluation, in *ACM SIGMOD Conference*, 1997, pp. 336–347.

Vijay Gandhi
James M. Kang
Shashi Shekhar
University of Minnesota
Minneapolis, Minnesota

STATISTICAL DATABASES

INTRODUCTION

Statistical databases are databases that contain statistical information. Such databases normally are released by national statistical institutes, but on occasion they can also be released by health-care authorities (epidemiology) or by private organizations (e.g., consumer surveys). Statistical databases typically come in three formats:

- *Tabular data*, that is, tables with counts or magnitudes, which are the classic output of official statistics.
- *Queryable databases*, that is, online databases to which the user can submit statistical queries (sums, averages, etc.).
- *Microdata*, that is, files in which each record contains information on an individual (a citizen or a company).

The peculiarity of statistical databases is that they should provide useful *statistical* information, but they should not reveal private information on the individuals to whom they refer (respondents). Indeed, supplying data to national statistical institutes is compulsory in most countries, but in return those institutes commit to preserving the privacy of respondents. Inference control in statistical databases, also known as statistical disclosure control (SDC), is a discipline that seeks to protect data in statistical databases so that they can be published without revealing confidential information that can be linked to specific individuals among those to whom the data correspond. SDC is applied to protect *respondent privacy* in areas such as official statistics, health statistics, and e-commerce (sharing of consumer data). Because data protection ultimately means data modification, the challenge for SDC is to achieve protection with minimum loss of accuracy sought by database users.

In Ref. 1, a distinction is made between SDC and other technologies for database privacy, like privacy-preserving data mining (PPDM) or private information retrieval (PIR): What makes the difference between those technologies is whose privacy they seek. Although SDC is aimed at respondent privacy, the primary goal of PPDM is to protect owner privacy when several database owners wish to cooperate in joint analyses across their databases without giving away their original data to each other. On its side, the primary goal of PIR is user privacy, that is, to allow the user of a database to retrieve some information item without the database exactly knowing which item was recovered.

The literature on SDC started in the 1970s, with the seminal contribution by Dalenius (2) in the statistical community and the works by Schlörer (3) and Denning et al. (4) in the database community. The 1980s saw moderate activity in this field. An excellent survey of the state of the art at the end of the 1980s is in Ref. 5. In the 1990s, renewed interest occurred in the statistical community and the discipline was developed even more under the names of statistical disclosure control in Europe and statistical disclosure limitation in America. Subsequent evolution has resulted in at least three clearly differentiated subdisciplines:

- *Tabular data protection*. The goal here is to publish *static* aggregate information, i.e., tables, in such a way that no confidential information on specific individuals among those to which the table refers can be inferred. See Ref. 6 for a conceptual survey.
- *Queryable databases*. The aggregate information obtained by a user as a result of successive queries should not allow him to infer information on specific individuals. Since the late 1970s, this has been known to be a difficult problem, which is subject to the tracker attack (4,7). SDC strategies here include perturbation, query restriction, and camouflage (providing interval answers rather than exact answers).
- *Microdata protection*. It is only recently that data collectors (statistical agencies and the like) have been persuaded to publish microdata. Therefore, microdata protection is the youngest subdiscipline, and it has experienced continuous evolution in the last few years. Its purpose is to mask the original microdata so that the masked microdata still are useful analytically but cannot be linked to the original respondents.

Several areas of application of SDC techniques exist, which include but are not limited to the following:

- *Official statistics*. Most countries have legislation that compels national statistical agencies to guarantee statistical confidentiality when they release data collected from citizens or companies. It justifies the research on SDC undertaken by several countries, among them the European Union (e.g., the CASC project) and the United States.
- *Health information*. This area is one of the most sensitive regarding privacy. For example, in the United States, the Privacy Rule of the Health Insurance Portability and Accountability Act (HIPAA) requires the strict regulation of protected health information for use in medical research. In most Western countries, the situation is similar.
- *E-commerce*. Electronic commerce results in the automated collection of large amounts of consumer data. This wealth of information is very useful to companies, which are often interested in sharing it with their subsidiaries or partners. Such consumer information transfer should not result in public profiling of individuals and is subject to strict regulation.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

THEORY

Formal Definition of Data Formats

A *microdata* file **X** with s respondents and t attributes is an $s \times t$ matrix, where X_{ij} is the value of attribute j for respondent i. Attributes can be numerical (e.g., age or salary) or categorical (e.g., gender or job).

The attributes in a microdata set can be classified in four categories that are not necessarily disjoint:

- *Identifiers.* These attributes *unambiguously* identify the respondent. Examples are the passport number, social security number, and name-surname.
- *Quasi-identifiers or key attributes.* These attributes identify the respondent with some degree of ambiguity. (Nonetheless, a combination of key attributes may provide unambiguous identification.) Examples are address, gender, age, and telephone number.
- *Confidential outcome attributes.* These attributes contain sensitive information on the respondent. Examples are salary, religion, political affiliation, and health condition.
- *Nonconfidential outcome attributes.* These attributes contain nonsensitive information on the respondent.

From microdata, *tabular data* can be generated by crossing one or more categorical attributes. Formally, a table is a function

$$T : D(X_{i1}) \times D(X_{i2}) \times ... \times D(X_{il}) \rightarrow \mathbb{R} \quad \text{or} \quad \mathbb{N}$$

where $l \leq t$ is the number of crossed categorical attributes and $D(X_{ij})$ is the domain where attribute X_{ij} takes its values.

Two kinds of tables exist: *frequency tables* that display the count of respondents at the crossing of the categorical attributes (in $\mathbb{N}$) and *magnitude tables* that display information on a numeric attribute at the crossing of the categorical attributes (in $\mathbb{R}$). For example, given some census microdata containing attributes "Job" and "Town," one can generate a *frequency table* that displays the count of respondents doing each job type in each town. If the census microdata also contain the "Salary" attribute, one can generate a magnitude table that displays the average salary for each job type in each town. The number n of cells in a table is normally much less than the number s of respondent records in a microdata file. However, tables must satisfy several linear constraints: marginal row and column totals. Additionally, a set of tables is called *linked* if they share some of the crossed categorical attributes: For example, "Job" × "Town" is linked to "Job" × "Gender."

Overview of Methods

Statistical disclosure control will be first reviewed for tabular data, then for queryable databases, and finally for microdata.

Methods for Tabular Data

Even if tables display aggregate information, a risk of disclosure exists in tabular data release. Several attacks are conceivable:

- *External attack.* For example, let a frequency table "Job" × "Town" be released where a single respondent exists for job J_i and town T_j. Then if a magnitude table is released with the average salary for each job type and each town, the exact salary of the only respondent with job J_i working in town T_j is disclosed publicly.
- *Internal attack.* Even if two respondents for job J_i and town T_j exist, the salary of each of them is disclosed to each other.
- *Dominance attack.* If one (or a few) respondents dominate in the contribution to a cell of a magnitude table, the dominant respondent(s) can upper-bound the contributions of the rest (e.g., if the table displays the total salary for each job type and town and one individual contributes 90% of that salary, he knows that his colleagues in the town are not doing very well).

SDC methods for tables fall into two classes: nonperturbative and perturbative. *Nonperturbative methods* do not modify the values in the tables; the best known method in this class is *cell suppression* (CS). *Perturbative methods* output a table with some modified values; well-known methods in this class include *controlled rounding* (CR) and the recent *controlled tabular adjustment* (CTA).

The idea of CS is to suppress those cells that are identified as sensitive, i.e., from which the above attacks can extract sensitive information, by the so-called sensitivity rules (e.g., the dominance rule, which identifies a cell as sensitive if it is vulnerable to a dominance attack). Sensitive cells are the primary suppressions. Then additional suppressions (secondary suppressions) are performed to prevent primary suppressions from being computed or even inferred within a prescribed protection interval using the row and column constraints (marginal row and column totals). Usually, one attempts to minimize either the number of secondary suppressions or their pooled magnitude, which results in complex optimization problems. Most optimization methods used are heuristic, based on mixed integer linear programming or network flows (8), and most of them are implemented in the τ-Argus free software package (9). CR rounds values in the table to multiples of a rounding base, which may entail rounding the marginal totals as well. On its side, CTA modifies the values in the table to prevent the inference of values of sensitive cells within a prescribed protection interval. The idea of CTA is to find the *closest* table to the original one that ensures such a protection for all sensitive cells. It requires optimization methods, which are typically based on mixed linear integer programming. Usually CTA entails less information loss than does CS.

Methods for Queryable Databases

In SDC of queryable databases, three main approaches exist to protect a confidential vector of numeric data from disclosure through answers to user queries:

- *Data perturbation.* Perturbing the data is a simple and effective approach whenever the users do not require deterministically correct answers to queries that are functions of the confidential vector. Perturbation can be applied to the records on which queries are computed (input perturbation) or to the query result after computing it on the original data (output perturbation). Perturbation methods can be found in Refs. 10–12.
- *Query restriction.* This approach is right if the user does require deterministically correct answers and these answers have to be exact (i.e., a number). Because exact answers to queries provide the user with very powerful information, it may become necessary to refuse to answer certain queries at some stage to avoid disclosure of a confidential datum. Several criteria decide whether a query can be answered; one of them is query set size control, that is, to refuse answers to queries that affect a set of records that is too small. An example of the query restriction approach can be found in Ref. 13.
- *Camouflage.* If deterministically correct, nonexact answers (i.e., small-interval answers) suffice, confidentiality via camouflage [CVC, (14)] is a good option. With this approach, unlimited answers to any conceivable query types are allowed. The idea of CVC is to "camouflage" the confidential vector a by making it part of the relative interior of a compact set Π of vectors. Then each query $q = f(a)$ is answered with an inverval $[q^-, q^+]$ that contains $[f^-, f^+]$, where f^- and f^+ are, respectively, the minimum and the maximum of f over Π.

Methods for Microdata

Microdata protection methods can generate the protected microdata set $\mathbf{X}'$

- Either by *masking original data*, i.e., generating a modified version $\mathbf{X}'$ of the original microdata set $\mathbf{X}$.
- Or by *generating synthetic data* $\mathbf{X}'$ that preserve some statistical properties of the original data $\mathbf{X}$.

Masking methods can in turn be divided in two categories depending on their effect on the original data (6):

- *Perturbative.* The microdata set is distorted before publication. In this way, unique combinations of scores in the original dataset may disappear, and new, unique combinations may appear in the perturbed dataset; such confusion is beneficial for preserving statistical confidentiality. The perturbation method used should be such that statistics computed on the perturbed dataset do not differ significantly from the statistics that would be obtained on the original dataset. *Noise addition, microaggregation, data/rank swapping, microdata rounding, resampling, and PRAM* are examples of perturbative masking methods (see Ref. 8 for details).
- *Nonperturbative.* Nonperturbative methods do not alter data; rather, they produce partial suppressions or reductions of detail in the original dataset. Sampling, global recoding, top and bottom coding, and local suppression are examples of nonperturbative masking methods.

Although a reasonable overview of the methods for protecting tables or queryable databases has been given above, microdata protection methods are more diverse, so that a description of some of them is needed.

Some Microdata Protection Methods

Additive Noise. Additive noise is a family of perturbative masking methods. The noise addition algorithms in the literature are as follows:

- *Masking by uncorrelated noise addition.* The vector of observations x_j for the jth attribute of the original dataset X_j is replaced by a vector

$$z_j = x_j + \epsilon_j$$

where ϵ_j is a vector of normally distributed errors drawn from a random variable $\varepsilon_j \sim N(0, \sigma^2_{\varepsilon_j})$, such that $Cov(\varepsilon_t, \varepsilon_l) = 0$ for all $t \neq l$. This algorithm neither preserves variances nor correlations.

- *Masking by correlated noise addition.* Correlated noise addition also preserves means and additionally allows preservation of correlation coefficients. The difference with the previous method is that the covariance matrix of the errors is now proportional to the covariance matrix of the original data, i.e., $\varepsilon \sim N(0, \Sigma_\varepsilon)$, where $\Sigma_\varepsilon = \alpha\Sigma$, with Σ being the covariance matrix of the original data.
- *Masking by noise addition and linear transformation.* In Ref. 15, a method is proposed that ensures by additional transformations that the sample covariance matrix of the masked attributes is an unbiased estimator for the covariance matrix of the original attributes.
- *Masking by noise addition and nonlinear transformation.* Combining simple additive noise and nonlinear transformation has also been proposed in such a way that application to discrete attributes is possible and univariate distributions are preserved. Unfortunately, the application of this method is very time-consuming and requires expert knowledge on the dataset and the algorithm. See Ref. 8 for more details.

In practice, only simple noise addition (two first variants) or noise addition with linear transformation are used. When using linear transformations, a decision has to be made whether to reveal them to the data user to allow for bias adjustment in the case of subpopulations.

In general, additive noise is not suitable to protect categorical data. On the other hand, it is well suited for continuous data for the following reasons:

- It makes no assumptions on the range of possible values for X_i (which may be infinite).
- The noise being added typically is continuous and with mean zero, which suits well continuous original data.

Microaggregation. Microaggregation is a family of SDC techniques for continous microdata. The rationale behind microaggregation is that confidentiality rules in use allow publication of microdata sets if records correspond to groups of k or more individuals, where no individual dominates (i.e., contributes too much to) the group and k is a threshold value. Strict application of such confidentiality rules leads to replacing individual values with values computed on small aggregates (microaggregates) before publication. It is the basic principle of microaggregation.

To obtain microaggregates in a microdata set with n records, these are combined to form g groups of size at least k. For each attribute, the average value over each group is computed and is used to replace each of the original averaged values. Groups are formed using a criterion of maximal similarity. Once the procedure has been completed, the resulting (modified) records can be published.

The optimal k-partition (from the information loss point of view) is defined to be the one that maximizes within-group homogeneity; the higher the within-group homogeneity, the lower the information loss, because microaggregation replaces values in a group by the group centroid. The sum of squares criterion is common to measure homogeneity in clustering. The within-groups sum of squares SSE is defined as

$$\mathrm{SSE} = \sum_{i=1}^{g}\sum_{j=1}^{n_i}(x_{ij}-\bar{x}_i)'(x_{ij}-\bar{x}_i)$$

The lower the SSE, the higher the within-group homogeneity. Thus, in terms of sums of squares, the optimal k-partition is the one that minimizes SSE.

Given a microdata set that consists of p attributes, these can be microaggregated together or partitioned into several groups of attributes. Also, the way to form groups may vary. Several taxonomies are possible to classify the microaggregation algorithms in the literature: *(1)* fixed group size (16–18) versus variable group size (19–21), *(2)* exact optimal [only for the univariate case, (22)] versus heuristic microaggregation (the rest of the microaggregation literature), and *(3)* categorical (18,23) versus continuous (the rest of references cited in this paragraph).

To illustrate, we next give a heuristic algorithm called MDAV [maximum distance to average vector, (18,24)] for multivariate fixed-group-size microaggregation on unprojected continuous data. We designed and implemented MDAV for the μ-Argus package (17).

1. Compute the average record $\bar{x}$ of all records in the dataset. Consider the most distant record x_r to the average record $\bar{x}$ (using the squared Euclidean distance).
2. Find the most distant record x_s from the record x_r considered in the previous step.
3. Form two groups around x_r and x_s, respectively. One group contains x_r and the $k-1$ records closest to x_r. The other group contains x_s and the $k-1$ records closest to x_s.
4. If at least $3k$ records do not belong to any of the two groups formed in Step 3, go to Step 1, taking as a new dataset the previous dataset minus the groups formed in the last instance of Step 3.
5. If between $3k-1$ and $2k$ records do not belong to any of the two groups formed in Step 3: a) compute the average record $\bar{x}$ of the remaining records, b) find the most distant record x_r from $\bar{x}$, c) form a group containing x_r and the $k-1$ records closest to x_r, and d) form another group containing the rest of records. Exit the Algorithm.
6. If less than $2k$ records do not belong to the groups formed in Step 3, form a new group with those records and exit the Algorithm.

The above algorithm can be applied independently to each group of attributes that results from partitioning the set of attributes in the dataset.

Data Swapping and Rank Swapping. Data swapping originally was presented as a perturbative SDC method for databases that contain only categorical attributes. The basic idea behind the method is to transform a database by exchanging values of confidential attributes among individual records. Records are exchanged in such a way that low-order frequency counts or marginals are maintained.

Even though the original procedure was not used often in practice, its basic idea had a clear influence in subsequent methods. A variant of data swapping for microdata is *rank swapping*, which will be described next in some detail.

Although originally described only for ordinal attributes (25), rank swapping can also be used for any numeric attribute. First, values of an attribute X_i are ranked in ascending order; then each ranked value of X_i is swapped with another ranked value randomly chosen within a restricted range (e.g., the rank of two swapped values cannot differ by more than $p\%$ of the total number of records, where p is an input parameter). This algorithm independently is used on each original attribute in the original dataset.

It is reasonable to expect that multivariate statistics computed from data swapped with this algorithm will be less distorted than those computed after an unconstrained swap.

PRAM. The post-randomization method [PRAM, (26)] is a probabilistic, perturbative method for disclosure protection of categorical attributes in microdata files. In the masked file, the scores on some categorical attributes for certain records in the original file are changed to a different score according to a prescribed probability mechanism, namely a Markov matrix called the PRAM matrix. The Markov approach makes PRAM very general because it encompasses noise addition, data suppression, and data recoding.

Because the PRAM matrix must contain a row for each possible value of each attribute to be protected, PRAM cannot be used for continuous data.

Sampling. This approach is a nonperturbative masking method. Instead of publishing the original microdata file, what is published is a sample S of the original set of records (6).

Sampling methods are suitable for categorical microdata, but for continuous microdata they probably should be combined with other masking methods. The reason is that sampling alone leaves a continuous attribute X_i unperturbed for all records in S. Thus, if attribute X_i is present in an external administrative public file, unique matches with the published sample are very likely: Indeed, given a continuous attribute X_i and two respondents o_1 and o_2, it is highly unlikely that X_i will take the same value for both o_1 and o_2 unless $o_1 = o_2$ (this is true even if X_i has been truncated to represent it digitally).

If, for a continuous identifying attribute, the score of a respondent is only approximately known by an attacker, it might still make sense to use sampling methods to protect that attribute. However, assumptions on restricted attacker resources are perilous and may prove definitely too optimistic if good-quality external administrative files are at hand.

Global Recoding. This approach is a nonperturbative masking method, also known sometimes as generalization. For a categorical attribute X_i, several categories are combined to form new (less specific) categories, which thus result in a new X_i' with $|D(X_i')| < |D(X_i)|$, where $| \cdot |$ is the cardinality operator. For a continuous attribute, global recoding means replacing X_i by another attribute X_i', which is a discretized version of X_i. In other words, a potentially infinite range $D(X_i)$ is mapped onto a finite range $D(X_i')$. This technique is used in the μ-Argus SDC package (17).

This technique is more appropriate for categorical microdata, where it helps disguise records with strange combinations of categorical attributes. Global recoding is used heavily by statistical offices.

Example. If a record exists with "Marital status = Widow/er" and "Age = 17," global recoding could be applied to "Marital status" to create a broader category "Widow/er or divorced" so that the probability of the above record being unique would diminish. □

Global recoding can also be used on a continuous attribute, but the inherent discretization leads very often to an unaffordable loss of information. Also, arithmetical operations that were straightforward on the original X_i are no longer easy or intuitive on the discretized X_i'.

Top and Bottom Coding. Top and bottom coding are special cases of global recoding that can be used on attributes that can be ranked, that is, continuous or categorical ordinal. The idea is that top values (those above a certain threshold) are lumped together to form a new category. The same is done for bottom values (those below a certain threshold). See Ref. 17.

Local Suppression. This approach is a nonperturbative masking method in which certain values of individual attributes are suppressed with the aim of increasing the set of records agreeing on a combination of key values. Ways to combine local suppression and global recoding are implemented in the μ-Argus SDC package (17).

If a continuous attribute X_i is part of a set of key attributes, then each combination of key values probably is unique. Because it does not make sense to suppress systematically the values of X_i, we conclude that local suppression is oriented to categorical attributes.

Synthetic Microdata Generation. Publication of synthetic —i.e.,simulated— data was proposed long ago as a way to guard against statistical disclosure. The idea is to generate data randomly with the constraint that certain statistics or internal relationships of the original dataset should be preserved.

More than ten years ago, Rubin suggested in Ref. 27 to create an entirely synthetic dataset based on the original survey data and multiple imputation. A simulation study of this approach was given in Ref. 28.

We next sketch the operation of the original proposal by Rubin. Consider an original microdata set X of size n records drawn from a much larger population of N individuals, where background attributes A, nonconfidential attributes B, and confidential attributes C exist. Background attributes are observed and available for all N individuals in the population, whereas B and C are only available for the n records in the sample X. The first step is to construct from X a multiply-imputed population of N individuals. This population consists of the n records in X and M (the number of multiple imputations, typically between 3 and 10) matrices of (B, C) data for the $N - n$ nonsampled individuals. The variability in the imputed values ensures, theoretically, that valid inferences can be obtained on the multiply-imputed population. A model for predicting (B, C) from A is used to multiply-impute (B, C) in the population. The choice of the model is a nontrivial matter. Once the multiply-imputed population is available, a sample Z of n' records can be drawn from it whose structure looks like the one of a sample of n' records drawn from the original population. This process can be done M times to create M replicates of (B, C) values. The result are M multiply-imputed synthetic datasets. To make sure no original data are in the synthetic datasets, it is wise to draw the samples from the multiply-imputed population excluding the n original records from it.

Synthetic data are appealing in that, at a first glance, they seem to circumvent the reidentification problem: Because published records are invented and do not derive from any original record, it might be concluded that no individual can complain from having been reidentified. At a closer look this advantage is less clear. If, by chance, a published synthetic record matches a particular citizen's nonconfidential attributes (age, marital status, place of residence, etc.) and confidential attributes (salary, mortgage, etc.), reidentification using the nonconfidential attributes is easy and that citizen may feel that his confidential attributes have been unduly revealed. In that case, the citizen is unlikely to be happy with or even understand the explanation that the record was generated synthetically.

On the other hand, limited data utility is another problem of synthetic data. Only the statistical properties explicitly captured by the model used by the data protector are preserved. A logical question at this point is why not directly publish the statistics one wants to preserve rather than release a synthetic microdata set.

One possible justification for synthetic microdata would be whether valid analyses could be obtained on several subdomains; i.e., similar results were obtained in several subsets of the original dataset and the corresponding subsets of the synthetic dataset. Partially synthetic or hybrid microdata are more likely to succeed in staying useful for subdomain analysis. However, when using partially synthetic or hybrid microdata, we lose the attractive feature of purely synthetic data that the number of records in the protected (synthetic) dataset is independent from the number of records in the original dataset.

EVALUATION

Evaluation of SDC methods must be carried out in terms of data utility and disclosure risk.

Measuring Data Utility

Defining what a generic utility loss measure is can be a tricky issue. Roughly speaking, such a definition should capture the amount of information loss for a reasonable range of data uses.

We will attempt a definition on the data with maximum granularity, that is, microdata. Similar definitions apply to rounded tabular data; for tables with cell suppressions, utility normally is measured as the reciprocal of the number of suppressed cells or their pooled magnitude. As to queryable databases, they can be viewed logically as tables as far as data utility is concerned: A denied query answer is equivalent to a cell suppression, and a perturbed answer is equivalent to a perturbed cell.

We will say little information loss occurs if the protected dataset is analytically valid and interesting according to the following definitions by Winkler (29):

- A protected microdata set is *analytically valid* if it approximately preserves the following with respect to the original data (some conditions apply only to continuous attributes):

  1. Means and covariances on a small set of subdomains (subsets of records and/or attributes)
  2. Marginal values for a few tabulations of the data
  3. At least one distributional characteristic

- A microdata set is *analytically interesting* if six attributes on important subdomains are provided that can be validly analyzed.

More precise conditions of analytical validity and analytical interest cannot be stated without taking specific data uses into account. As imprecise as they may be, the above definitions suggest some possible measures:

- Compare raw records in the original and the protected dataset. The more similar the SDC method to the identity function, the less the impact (but the higher the disclosure risk!). This process requires pairing records in the original dataset and records in the protected dataset. For masking methods, each record in the protected dataset is paired naturally to the record in the original dataset from which it originates. For synthetic protected datasets, pairing is less obvious.
- Compare some statistics computed on the original and the protected datasets. The above definitions list some statistics that should be preserved as much as possible by an SDC method.

A strict evaluation of information loss must be based on the data uses to be supported by the protected data. The greater the differences between the results obtained on original and protected data for those uses, the higher the loss of information. However, very often microdata protection cannot be performed in a data use-specific manner, for the following reasons:

- Potential data uses are very diverse, and it may be even hard to identify them all at the moment of data release by the data protector.
- Even if all data uses could be identified, releasing several versions of the same original dataset so that the ith version has an information loss optimized for the ith data use may result in unexpected disclosure.

Because data often must be protected with no specific data use in mind, generic information loss measures are desirable to guide the data protector in assessing how much harm is being inflicted to the data by a particular SDC technique.

Information loss measures for numeric data. Assume a microdata set with n individuals (records) $I_1, I_2, \ldots, I_n$ and p continuous attributes $Z_1, Z_2, \ldots, Z_p$. Let X be the matrix that represents the original microdata set (rows are records and columns are attributes). Let X' be the matrix that represents the protected microdata set. The following tools are useful to characterize the information contained in the dataset:

- Covariance matrices V (on X) and V' (on X').
- Correlation matrices R and R'.
- Correlation matrices RF and RF' between the p attributes and the p factors $PC_1, \ldots, PC_p$ obtained through principal components analysis.
- Communality between each p attribute and the first principal component PC_1 (or other principal components PC_i's). Communality is the percent of each attribute that is explained by PC_1 (or PC_i). Let C be the vector of communalities for X and C' be the corresponding vector for X'.
- Factor score coefficient matrices F and F'. Matrix F contains the factors that should multiply each

	Mean square error	Mean absolute error	Mean variation
$X - X'$	$\frac{\sum_{j=1}^{p}\sum_{i=1}^{n}(x_{ij}-x'_{ij})^2}{n\,p}$	$\frac{\sum_{j=1}^{p}\sum_{i=1}^{n}\lvert x_{ij}-x'_{ij}\rvert}{n\,p}$	$\frac{\sum_{j=1}^{p}\sum_{i=1}^{n}\frac{\lvert x_{ij}-x'_{ij}\rvert}{\lvert x_{ij}\rvert}}{n\,p}$
$V - V'$	$\frac{\sum_{j=1}^{p}\sum_{1\le i\le j}(v_{ij}-v'_{ij})^2}{\frac{p(p+1)}{2}}$	$\frac{\sum_{j=1}^{p}\sum_{1\le i\le j}\lvert v_{ij}-v'_{ij}\rvert}{\frac{p(p+1)}{2}}$	$\frac{\sum_{j=1}^{p}\sum_{1\le i\le j}\frac{\lvert v_{ij}-v'_{ij}\rvert}{\lvert v_{ij}\rvert}}{\frac{p(p+1)}{2}}$
$R - R'$	$\frac{\sum_{j=1}^{p}\sum_{1\le i<j}(r_{ij}-r'_{ij})^2}{\frac{p(p-1)}{2}}$	$\frac{\sum_{j=1}^{p}\sum_{1\le i<j}\lvert r_{ij}-r'_{ij}\rvert}{\frac{p(p-1)}{2}}$	$\frac{\sum_{j=1}^{p}\sum_{1\le i<j}\frac{\lvert r_{ij}-r'_{ij}\rvert}{\lvert r_{ij}\rvert}}{\frac{p(p-1)}{2}}$
$RF - RF'$	$\frac{\sum_{j=1}^{p}w_j\sum_{i=1}^{p}(rf_{ij}-rf'_{ij})^2}{p^2}$	$\frac{\sum_{j=1}^{p}w_j\sum_{i=1}^{p}\lvert rf_{ij}-rf'_{ij}\rvert}{p^2}$	$\frac{\sum_{j=1}^{p}w_j\sum_{i=1}^{p}\frac{\lvert rf_{ij}-rf'_{ij}\rvert}{\lvert rf_{ij}\rvert}}{p^2}$
$C - C'$	$\frac{\sum_{i=1}^{p}(c_i-c'_i)^2}{p}$	$\frac{\sum_{i=1}^{p}\lvert c_i-c'_i\rvert}{p}$	$\frac{\sum_{i=1}^{p}\frac{\lvert c_i-c'_i\rvert}{\lvert c_i\rvert}}{p}$
$F - F'$	$\frac{\sum_{j=1}^{p}w_j\sum_{i=1}^{p}(f_{ij}-f'_{ij})^2}{p^2}$	$\frac{\sum_{j=1}^{p}w_j\sum_{i=1}^{p}\lvert f_{ij}-f'_{ij}\rvert}{p^2}$	$\frac{\sum_{j=1}^{p}w_j\sum_{i=1}^{p}\frac{\lvert f_{ij}-f'_{ij}\rvert}{\lvert f_{ij}\rvert}}{p^2}$

attribute in X to obtain its projection on each principal component. F' is the corresponding matrix for X'.

A single quantitative measure does not seem to reflect completely those structural differences. Therefore, we proposed in Refs. 30 and 31 to measure information loss through the discrepancies between matrices X, V, R, RF, C, and F obtained on the original data and the corresponding X', V', R', RF', C', and F' obtained on the protected dataset. In particular, discrepancy between correlations is related to the information loss for data uses such as regressions and cross tabulations.

Matrix discrepancy can be measured in at least three ways:

Mean square error. Sum of squared component-wise differences between pairs of matrices, divided by the number of cells in either matrix.

Mean absolute error. Sum of absolute component-wise differences between pairs of matrices, divided by the number of cells in either matrix.

Mean variation. Sum of absolute percent variation of components in the matrix computed on protected data with respect to components in the matrix computed on original data, divided by the number of cells in either matrix. This approach has the advantage of not being affected by scale changes of attributes.

The following table summarizes the measures proposed in the above references. In this table, p is the number of attributes, n is the number of records, and the components of matrices are represented by the corresponding lowercase letters (e.g., x_{ij} is a component of matrix X). Regarding $X - X'$ measures, it also makes sense to compute those on the averages of attributes rather than on all data (call this variant $\overline{X} - \overline{X}'$). Similarly, for $V - V'$ measures, it would also be sensible to use them to compare only the variances of the attributes, i.e., to compare the diagonals of the covariance matrices rather than the whole matrices (call this variant $S - S'$).

Information loss measures for categorical data. These measures can be based on a direct comparison of categorical values, or on a comparison of contingency tables or on Shannon's entropy. See Ref. 30 for more details.

Bounded information loss measures. The information loss measures discussed above are unbounded, (i.e., they do not take values in a predefined interval) On the other hand, as discussed below, disclosure risk measures are bounded naturally (the risk of disclosure is bounded naturally between 0 and 1). Defining bounded information loss measures may be convenient to enable the data protector to trade off information loss against disclosure risk. In Ref. 32, probabilistic information loss measures bounded between 0 and 1 are proposed for continuous data.

Measuring Disclosure Risk

In the context of statistical disclosure control, disclosure risk can be defined as the risk that a user or an intruder can use the protected dataset $\mathbf{X}'$ to derive confidential information on an individual among those in the original dataset $\mathbf{X}$.

Disclosure risk can be regarded from two different perspectives:

1. **Attribute disclosure.** This approach to disclosure is defined as follows. Disclosure takes place when an attribute of an individual can be determined more accurately with access to the released statistic than it is possible without access to that statistic.
2. **Identity disclosure.** Attribute disclosure does not imply a disclosure of the identity of any individual.

Identity disclosure takes place when a record in the protected dataset can be linked with a respondent's identity. Two main approaches usually are employed for measuring identity disclosure risk: uniqueness and reidentification.

2.1 Uniqueness. Roughly speaking, the risk of identity disclosure is measured as the probability that rare combinations of attribute values in the released protected data are indeed rare in the original population from which the data come. This approach is used typically with nonperturbative statistical disclosure control methods and, more specifically, with sampling. The reason that uniqueness is not used with perturbative methods is that when protected attribute values are perturbed versions of original attribute values, it does not make sense to investigate the probability that a rare combination of protected values is rare in the original dataset, because *that* combination is most probably *not found* in the original dataset.

2.2 Reidentification. This empirical approach evaluates the risk of disclosure. In this case, record linkage software is constructed to estimate the number of reidentifications that might be obtained by a specialized intruder. Reidentification through record linkage provides a more unified approach than do uniqueness methods because the former can be applied to any kind of masking and not just to nonperturbative masking. Moreover, reidentification can also be applied to synthetic data.

Trading off Information Loss and Disclosure Risk

The mission of SDC to modify data in such a way that sufficient protection is provided at minimum information loss suggests that a good SDC method is one achieving a good tradeoff between disclosure risk and information loss. Several approaches have been proposed to handle this tradeoff. We discuss *SDC scores, R-U maps*, and *k-anonymity*.

Score Construction. Following this idea, Domingo-Ferrer and Torra (30) proposed a score for method performance rating based on the average of information loss and disclosure risk measures. For each method M and parameterization P, the following score is computed:

$$\mathrm{Score}(\mathbf{X},\mathbf{X}') = \frac{\mathrm{IL}(\mathbf{X},\mathbf{X}') + \mathrm{DR}'(\mathbf{X},\mathbf{X}')}{2}$$

where IL is an information loss measure, DR is a disclosure risk measure, and $\mathbf{X}'$ is the protected dataset obtained after applying method M with parameterization P to an original dataset $\mathbf{X}$.

In Ref. 30, *IL* and *DR* were computed using a weighted combination of several information loss and disclosure risk measures. With the resulting score, a ranking of masking methods (and their parameterizations) was obtained. To illustrate how a score can be constructed, we next describe the particular score used in Ref. 30.

Example. Let X and X' be matrices that represent original and protected datasets, respectively, where all attributes are numeric. Let V and R be the covariance matrix and the correlation matrix of X, respectively; let $\overline{X}$ be the vector of attribute averages for X, and let S be the diagonal of V. Define V', R', $\overline{X}'$, and S' analogously from X'. The Information Loss (IL) is computed by averaging the mean variations of $X - X'$, $\overline{X} - \overline{X}'$, $V - V'$, $S - S'$, and the mean absolute error of $R - R'$ and by multiplying the resulting average by 100. Thus, we obtain the following expression for information loss:

$$\mathrm{IL} = \frac{100}{5}\left(\frac{\sum_{j=1}^{p}\sum_{i=1}^{n}\frac{|x_{ij}-x'_{ij}|}{|x_{ij}|}}{np} + \frac{\sum_{j=1}^{p}\frac{|\overline{n}\,^1_j - \overline{x}'_j|}{|\overline{n}\,^1{}_j|}}{p} \right.$$
$$+\frac{\sum_{j=1}^{p}\sum_{1\le i\le j}\frac{|v_{ij}-v'_{ij}|}{|v_{ij}|}}{\frac{p(p+1)}{2}} + \frac{\sum_{j=1}^{p}\frac{|v_{jj}-v'_{jj}|}{|v_{jj}|}}{p}$$
$$\left. + \frac{\sum_{j=1}^{p}\sum_{1\le i\le j}|r_{ij}-r'_{ij}|}{\frac{p(p-1)}{2}} \right)$$

The expression of the overall score is obtained by combining information loss and information risk as follows:

$$\mathrm{Score} = \frac{\mathrm{IL} + \frac{(0.5DLD+0.5PLD)+ID}{2}}{2}$$

Here, DLD (distance linkage disclosure risk) is the percentage of correctly linked records using distance-based record linkage (30), PLD (probabilistic linkage record disclosure risk) is the percentage of correctly linked records using probabilistic linkage (33), ID (interval disclosure) is the percentage of original records falling in the intervals around their corresponding masked values, and IL is the information loss measure defined above.

Based on the above score, Domingo-Ferrer and Torra (30) found that, for the benchmark datasets and the intruder's external information they used, two good performers among the set of methods and parameterizations they tried were as follows: *(1)* rankswapping with parameter p around 15 (see the description above) and *(2)* multivariate microaggregation on unprojected data taking groups of three attributes at a time (Algorithm MDAV above with partitioning of the set of attributes). □

Using a score permits regarding the selection of a masking method and its parameters as an optimization problem. A masking method can be applied to the original data file, and then a postmasking optimization procedure can be applied to decrease the score obtained.

On the negative side, no specific score weighting can do justice to all methods. Thus, when ranking methods, the values of all measures of information loss and disclosure risk should be supplied along with the overall score.

R-U maps. A tool that may be enlightening when trying to construct a score or, more generally, to optimize the tradeoff between information loss and disclosure risk is a graphical representation of pairs of measures (disclosure risk, information loss) or their equivalents (disclosure risk, data utility). Such maps are called R-U confidentiality maps (34). Here, R stands for disclosure risk and U stands for data utility. In its most basic form, an R-U confidentiality map is the set of paired values (R,U), of disclosure risk and data utility that correspond to various strategies for data release (e.g., variations on a parameter). Such (R,U) pairs typically are plotted in a two-dimensional graph, so that the user can grasp easily the influence of a particular method and/or parameter choice.

***k*-Anonymity.** A different approach to facing the conflict between information loss and disclosure risk is suggested by Samarati and Sweeney (35). A protected dataset is said to satisfy k-anonymity for $k > 1$ if, for each combination of key attribute values (e.g., address, age, and gender), at least k records exist in the dataset sharing that combination. Now if, for a given k, k-anonymity is assumed to be enough protection, one can concentrate on minimizing information loss with the only constraint that k-anonymity should be satisfied. This method is a clean way of solving the tension between data protection and data utility. Because k-anonymity usually is achieved via generalization (equivalent to global recoding, as said above) and local suppression, minimizing information loss usually translates to reducing the number and/or the magnitude of suppressions.

k-Anonymity bears some resemblance to the underlying principle of multivariate microaggregation and is a useful concept because key attributes usually are categorical or can be categorized, [i.e., they take values in a finite (and ideally reduced) range] However, reidentification is not based on necessarily categorical key attributes: Sometimes, numeric outcome attributes, which are continuous and often cannot be categorized, give enough clues for reidentification. Microaggregation was suggested in Ref. 18 as a possible way to achieve k-anonymity for numeric, ordinal, and nominal attributes: The idea is to use multivariate microaggregation on the key attributes of the dataset.

Future

Many open issues in SDC exist, some of which hopefully can be solved with additional research and some of which are likely to stay open because of the inherent nature of SDC. We first list some of the issues that probably could and should be settled in the near future:

- Identifying a comprehensive listing of data uses (e.g., regression models, and association rules) that would allow the definition of data use-specific information loss measures broadly accepted by the community; those new measures could complement and/or replace the generic measures currently used. Work in this line was started in Europe in 2006 under the CENEX SDC project sponsored by Eurostat.
- Devising disclosure risk assessment procedures that are as universally applicable as record linkage while being less greedy in computational terms.
- Identifying, for each domain of application, which are the external data sources that intruders typically can access to attempt reidentification. This capability would help data protectors to figure out in more realistic terms which are the disclosure scenarios they should protect data against.
- Creating one or several benchmarks to assess the performance of SDC methods. Benchmark creation currently is hampered by the confidentiality of the original datasets to be protected. Data protectors should agree on a collection of nonconfidential, original-looking datasets (financial datasets, population datasets, etc.), which can be used by anybody to compare the performance of SDC methods. The benchmark should also incorporate state-of-the-art disclosure risk assessment methods, which requires continuous update and maintenance.

Other issues exist whose solution seems less likely in the near future, because of the very nature of SDC methods. If an intruder knows the SDC algorithm used to create a protected data set, he can mount algorithm-specific reidentification attacks that can disclose more confidential information than conventional data mining attacks. Keeping secret the SDC algorithm used would seem a solution, but in many cases, the protected dataset gives some clues on the SDC algorithm used to produce it. Such is the case for a rounded, microaggregated, or partially suppressed microdata set. Thus, it is unclear to what extent the SDC algorithm used can be kept secret.

CROSS-REFERENCES

statistical disclosure control. See Statistical Databases.

statistical disclosure limitation. See Statistical Databases.

inference control. See Statistical Databases.

privacy in statistical databases. See Statistical Databases.

BIBLIOGRAPHY

1. J. Domingo-Ferrer, A three-dimensional conceptual framework for database privacy, in *Secure Data Management-4th VLDB Workshop SDM'2007, Lecture Notes in Computer Science*, vol. **4721**. Berlin: Springer-Verlag, 2007, pp. 193–202.
2. T. Dalenius, The invasion of privacy problem and statistics production. An overview, *Statistik Tidskrift*, **12**: 213–225, 1974.
3. J. Schlörer, Identification and retrieval of personal records from a statistical data bank. *Methods Inform. Med.*, **14**(1): 7–13, 1975.
4. D. E. Denning, P. J. Denning, and M. D. Schwartz, The tracker: a threat to statistical database security, *ACM Trans. Database Syst.*, **4**(1): 76–96, 1979.

5. N. R. Adam and J. C. Wortmann, Security-control for statistical databases: a comparative study, *ACM Comput. Surv.*, **21**(4): 515–556, 1989.
6. L. Willenborg and T. DeWaal, *Elements of Statistical Disclosure Control*, New York: Springer-Verlag, 2001.
7. J. Schlörer, Disclosure from statistical databases: quantitative aspects of trackers, *ACM Trans. Database Syst.*, **5**: 467–492, 1980.
8. A. Hundepool, J. Domingo-Ferrer, L. Franconi, S. Giessing, R. Lenz, J. Longhurst, E. Schulte-Nordholt, G. Seri, and P.-P. DeWolf, *Handbook on Statistical Disclosure Control (version 1.0)*. Eurostat (CENEX SDC Project Deliverable), 2006. Available: http://neon.vb.cbs.nl/CENEX/.
9. A. Hundepool, A. van deWetering, R. Ramaswamy, P.-P. deWolf, S. Giessing, M. Fischetti, J.-J. Salazar, J. Castro, and P. Lowthian, *τ-ARGUS v. 3.2 Software and User's Manual*, CENEX SDC Project Deliverable, 2007. Available: http://neon.vb.cbs.nl/casc/TAU.html.
10. G. T. Duncan and S. Mukherjee, Optimal disclosure limitation strategy in statistical databases: deterring tracker attacks through additive noise, *J. Am. Statist. Assoc.*, **45**: 720–729, 2000.
11. K. Muralidhar, D. Batra, and P. J. Kirs, Accessibility, security and accuracy in statistical databases: the case for the multiplicative fixed data perturbation approach, *Management Sci.*, **41**: 1549–1564, 1995.
12. J. F. Traub, Y. Yemini, and H. Wozniakowski, The statistical security of a statistical database, *ACM Trans. Database Syst.*, **9**: 672–679, 1984.
13. F. Y. Chin and G. Ozsoyoglu, Auditing and inference control in statistical databases, *IEEE Trans. Software Engin.*, SE-**8**: 574–582, 1982.
14. R. Gopal, R. Garfinkel, and P. Goes, Confidentiality via camouflage: The CVC approach to disclosure limitation when answering queries to databases, *Operations Res.*, **50**: 501–516, 2002.
15. J. J. Kim, A method for limiting disclosure in microdata based on random noise and transformation, *Proceedings of the Section on Survey Research Methods*, Alexandria, VA, 1986, pp. 303–308.
16. D. Defays and P. Nanopoulos, Panels of enterprises and confidentiality: the small aggregates method, *Proc. of 92 Symposium on Design and Analysis of Longitudinal Surveys*, Ottawa: 1993, pp. 195–204. Statistics Canada.
17. A. Hundepool, A. Van deWetering, R. Ramaswamy, L. Franconi, A. Capobianchi, P.-P. DeWolf, J. Domingo-Ferrer, V. Torra, R. Brand, and S. Giessing, *μ-ARGUS version 4.0 Software and User's Manual*, Statistics Netherlands, Voorburg NL, 2005. Available: http://neon.vb.cbs.nl/casc.
18. J. Domingo-Ferrer and V. Torra, Ordinal, continuous and heterogenerous k-anonymity through microaggregation, *Data Mining Knowl. Discov.*, **11**(2): 195–212, 2005.
19. J. Domingo-Ferrer and J. M. Mateo-Sanz, Practical data-oriented microaggregation for statistical disclosure control, *IEEE Trans. Knowledge Data Engin.*, **14**(1): 189–201, 2002.
20. M. Laszlo and S. Mukherjee, Minimum spanning tree partitioning algorithm for microaggregation, *IEEE Trans. Knowledge Data Engineer.*, **17**(7): 902–911, 2005.
21. J. Domingo-Ferrer, F. Sebé, and A. Solanas, A polynomial-time approximation to optimal multivariate microaggregation, *Computers Mathemat. Applicat.*, **55**(4): 714–732, 2008.
22. S. L. Hansen and S. Mukherjee, A polynomial algorithm for optimal univariate microaggregation, *IEEE Trans. Knowl. Data Engineer.*, **15**(4): 1043–1044, 2003.
23. V. Torra, Microaggregation for categorical variables: a median based approach, in *Privacy in Statistical Databases-PSD 2004, Lecture Notes in Computer Science*, vol. **3050**. Berlin: Springer-Verlag, 2004, pp. 162–174.
24. J. Domingo-Ferrer, A. Martínez-Ballesté, J. M. Mateo-Sanz, and F. Sebé, Efficient multivariate data-oriented microaggregation, *VLDB Journal*, **15**: 355–369, 2006.
25. B. Greenberg, Rank swapping for ordinal data, 1987, Washington, DC: U. S. Bureau of the Census (unpublished manuscript).
26. J. M. Gouweleeuw, P. Kooiman, L. C. R. J. Willenborg, and P.-P. DeWolf, Post randomisation for statistical disclosure control: theory and implementation, 1997. Research Paper no. 9731. Voorburg: Statistics Netherlands.
27. D. B. Rubin, Discussion of statistical disclosure limitation, *J. Official Stat.*, **9**(2): 461–468, 1993.
28. J. P. Reiter, Satisfying disclosure restrictions with synthetic data sets, *J. Official Stat.*, **18**(4): 531–544, 2002.
29. W. E. Winkler, Re-identification methods for evaluating the confidentiality of analytically valid microdata, *Rese. in Official Stat.*, **1**(2): 50–69, 1998.
30. J. Domingo-Ferrer and V. Torra, A quantitative comparison of disclosure control methods for microdata, in P. Doyle, J. I. Lane, J. J. M. Theeuwes, and L. Zayatz (eds.), *Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies*, Amsterdam: North-Holland, 2001, pp. 111–134.
31. F. Sebé, J. Domingo-Ferrer, J. M. Mateo-Sanz, and V. Torra, Post-masking optimization of the tradeoff between information loss and disclosure risk in masked microdata sets, in *Inference Control in Statistical Databases, Lecture Notes in Computer Science*, vol. **2316**. Springer-Verlag, 2002, pp. 163–171.
32. J. M. Mateo-Sanz, J. Domingo-Ferrer, and F. Sebé, Probabilistic information loss measures in confidentiality protection of continuous microdata, *Data Min. Knowl.* Disc., **11**(2): 181–193, 2005.
33. I. P. Fellegi and A. B. Sunter, A theory for record linkage, *J. Am. Stat. Associat.*, **64**(328): 1183–1210, 1969.
34. G. T. Duncan, S. E. Fienberg, R. Krishnan, R. Padman, and S. F. Roehrig, Disclosure limitation methods and information loss for tabular data, in P. Doyle, J. I. Lane, J. J. Theeuwes, and L. V. Zayatz (eds.), *Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies*, Amsterdam: North-Holland, 2001, pp. 135–166.
35. P. Samarati and L. Sweeney, Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression, Technical Report, SRI International, 1998.

FURTHER READING

A. Hundepool, J. Domingo-Ferrer, L. Franconi, S. Giessing, R. Lenz, J. Longhurst, E. Schulte-Nordholt, G. Seri, and P.-P. DeWolf, *Handbook on Statistical Disclosure Control (version 1.0)*, Eurostat (CENEX SDC Project Deliverable), 2006. Available: http://neon.vb.cbs.nl/CENEX/.

L. Willenborg and T. DeWaal, *Elements of Statistical Disclosure Control*, New York: Springer-Verlag, 2001.

JOSEP DOMINGO-FERRER
Rovira i Virgili University
Tarragona, Catalonia, Spain

SYSTEM MONITORING

The term *system* refers to a computer system that is composed of hardware and software for data processing. *System monitoring* collects information about the behavior of a computer system while the system is running. What is of interest here is run-time information that cannot be obtained by static analysis of programs. All collected information is essentially about system correctness or performance. Such information is vital for understanding how a system works. It can be used for dynamic safety checking and failure detection, program testing and debugging, dynamic task scheduling and resource allocation, performance evaluation and tuning, system selection and design, and so on.

COMPONENTS AND TECHNIQUES FOR SYSTEM MONITORING

System monitoring has three components. First, the jobs to be run and the items to be measured are determined. Then, the system to be monitored is modified to run the jobs and take the measurements. This is the major component. Monitoring is accomplished in two operations: triggering and recording (1). *Triggering,* also called *activation,* is the observation and detection of specified events during system execution. *Recording* is the collection and storage of data pertinent to those events. Finally, the recorded data are analyzed and displayed.

The selection and characterization of the jobs to be run for monitoring is important, because it is the basis for interpreting the monitoring results and guaranteeing that the experiments are repeatable. A collection of jobs to be run is called a *test workload* (2–4); for performance monitoring, this refers mainly to the *load* rather than the *work,* or *job.* A workload can be real or synthetic. A *real workload* consists of jobs that are actually performed by the users of the system to be monitored. A *synthetic workload,* usually called a *benchmark,* consists of batch programs or interactive scripts that are designed to represent the actual jobs of interest. Whether a workload is real or synthetic does not affect the monitoring techniques.

Items to be measured are determined by the applications. They can be about the entire system or about different levels of the system, from user-level application programs to operating systems to low-level hardware circuits. For the entire system, one may need to know whether jobs are completed normally and performance indices such as job completion time, called *turnaround time* in batch systems and *response time* in interactive systems, or the number of jobs completed per unit of time, called *throughput* (3). For application programs, one may be interested in how often a piece of code is executed, whether a variable is read between two updates, or how many messages are sent by a process. For operating systems, one may need to know whether the CPU is busy at certain times, how often paging occurs, or how long an I/O operation takes. For hardware circuits, one may need to know how often a cache element is replaced, or whether a network wire is busy.

Monitoring can use either event-driven or sampling techniques (3). *Event-driven monitoring* is based on observing changes of system state, either in software programs or hardware circuits, that are caused by events of interest, such as the transition of the CPU from busy to idle. It is often implemented as special instructions for interrupt–intercept that are inserted into the system to be monitored. *Sampling monitoring* is based on probing at selected time intervals, into either software programs or hardware circuits, to obtain data of interest, such as what kinds of processes occupy the CPU. It is often implemented as timer interrupts during which the state is recorded. Note that the behavior of the system under a given workload can be simulated by a simulation tool. Thus monitoring that should be performed on the real system may be carried out on the simulation tool. Monitoring simulation tools is useful, or necessary, for understanding the behavior of models of systems still under design.

Monitoring can be implemented using software, hardware, or both (1,3–5). *Software monitors* are programs that are inserted into the system to be monitored. They are triggered upon appropriate interrupts or by executing the inserted code. Data are recorded in buffers in the working memory of the monitored system and, when necessary, written to secondary storage. *Hardware monitors* are electronic devices that are connected to specific system points. They are triggered upon detecting signals of interest. Data are recorded in separate memory independent of the monitored system. *Hybrid monitors* combine techniques from software and hardware. Often, the triggering is carried out using software, and the data recording is carried out using hardware.

The data collected by a monitor must be analyzed and displayed. Based on the way in which results are analyzed and displayed, a monitor is classified as an on-line monitor or a batch monitor. *On-line monitors* analyze and display the collected data in real-time, either continuously or at frequent intervals, while the system is still being monitored. This is also called *continuous monitoring* (6). *Batch monitors* collect data first and analyze and display them later using a batch program. In either case, the analyzed data can be presented using many kinds of graphic charts, as well as text and tables.

ISSUES IN SYSTEM MONITORING

Major issues of concern in monitoring are at what levels we can obtain information of interest, what modifications to the system are needed to perform the monitoring, the disturbance of such modifications to the system behavior, and the cost of implementing such modifications. There are also special concerns for monitoring real-time systems, parallel architectures, and distributed systems.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Activities and data structures visible to the user process can be monitored at the application-program level. These include function and procedure calls and returns, assignments to variables, loopings and branchings, inputs and outputs, as well as synchronizations. Activities and data structures visible to the kernel can be monitored at the operating-system level; these include system state transitions, external interrupts, system calls, as well as data structures such as process control blocks. At the hardware level, various patterns of signals on the buses can be monitored. Obviously, certain high-level information cannot be obtained by monitoring at a lower level, and vice versa. It is worth noting that more often high-level information can be used to infer low-level information if one knows enough about all the involved components, such as the compilers, but the converse is not true, simply because more often multiple high-level activities are mapped to the same low-level activity.

In general, the software and hardware of a system are not purposely designed to be monitored. This often restricts what can be monitored in a system. To overcome these restrictions, modifications to the system, called *instrumentation,* are often required, for example, inserting interrupt instructions or attaching hardware devices. The information obtainable with a monitor and the cost of measurements determine the *measurability* of a computer system (3,7). At one extreme, every system component can be monitored at the desired level of detail, while at the other extreme, only the external behavior of the system as a whole can be monitored. When a low degree of detail is required, a *macroscopic* analysis, which requires measurement of global indices such as turnaround time and response time, is sufficient. When a high degree of detail is needed, a *microscopic* analysis, which requires, say, the time of executing each instruction or loading each individual page, must be performed.

Monitoring often interferes with the system behavior, since it may consume system resources, due to the time of performing monitoring activities and the space of storing collected data, which are collectively called the *overhead* of monitoring. A major issue in monitoring is to reduce the perturbation. It is easy to see that a macroscopic analysis incurs less interference than a microscopic analysis. Usually, sampling monitoring causes less interference than event-driven monitoring. In terms of implementation, software monitors always interfere and sometimes interfere greatly with the system to be monitored, but hardware monitors cause little or no interference.

Implementing monitoring usually has a cost, since it requires modification to the system to be monitored. Therefore, an important concern is to reduce the cost. Software monitors are simply programs, so they are usually less costly to develop and easier to change. In contrast, hardware monitors require separate hardware devices and thus are usually more difficult to build and modify.

Finally, special methods and techniques are necessary for monitoring real-time systems, parallel architectures, and distributed systems. Real-time systems have real-time constraints, so interference becomes much more critical. For parallel architectures, monitoring needs to handle issues arising from interprocessor communication and scheduling, cache behavior, and shared memory behavior. For distributed systems, monitoring must take into account ordering of distributed events, message passing, synchronization, as well as various kinds of failures.

MONITORING PRINCIPLES

A set of principles is necessary to address all the issues involved in monitoring. The major task is to determine the monitoring techniques needed based on the applications and the trade-offs. Methods and tools that facilitate monitoring are also needed.

Consider the major task. Given the desired information, one first needs to determine all levels that can be monitored to obtain the information. For each possibility, one determines all modifications of the system that are needed to perform the monitoring. Then one needs to assess the perturbation that the monitoring could cause. Finally, one must estimate the cost of the implementations. Clearly, unacceptable perturbation or cost helps reduce the possibilities. Then, one needs to evaluate all possibilities based on the following trade-offs.

First, monitoring at a higher level generally requires less modification to the system and has smaller implementation cost, but it may have larger interference with the system behavior. Thus one principle is to monitor at the highest level whose interference is acceptable. This implies that, if a software monitor has acceptable interference, one should avoid using a hardware monitor. Furthermore, to reduce implementation cost, for a system being designed or that is difficult to measure, one can use simulation tools instead of the real system if credibility can be established.

Second, macroscopic analysis generally causes less perturbation to the system behavior than microscopic analysis, and it often requires less modification to the system and has smaller cost. Therefore, a second principle is to use macroscopic analysis instead of microscopic analysis if possible. While sampling is a statistical technique that records data only at sampled times, event detection is usually used to record all potentially interesting events and construct the execution trace. Thus one should avoid using tracing if the desired information can be obtained by sampling.

Additionally, one should consider workload selection and data analysis. Using benchmarks instead of real workload makes the experiments repeatable and facilitates comparison of monitoring results. It can also reduce the cost, since running real jobs could be expensive or impossible. Thus using benchmarks is preferred, but a number of common mistakes need to be carefully avoided (4). Data analysis involves a separate trade-off: the on-line method adds time overhead but can reduce the space overhead. Thus even when monitoring results do not need to be presented in an on-line fashion, on-line analysis can be used to reduce the space overhead and, when needed, separate processors can be used to reduce also the time overhead.

Finally, special applications determine special monitoring principles. For example, for monitoring real-time systems, perturbation is usually not tolerable, but a full trace is often needed to understand system behavior. To address this problem, one may perform microscopic monitoring based on event detection and implement monitoring in hardware so as to sense signals on buses at high speed and with low overhead. If monitoring results are needed in an on-line fashion, separate resources for data analysis must be used. Of course, all these come at a cost.

To facilitate monitoring, one needs methods and tools for instrumenting the system, efficient data structures and algorithms for storing and manipulating data, and techniques for relating monitoring results to the source program to identify problematic code sections. Instrumentation of programs can be done via program transformation, by augmenting the source code, the target code, the run-time environment, the operating system, or the hardware. Often, combinations of these techniques are used. Efficient data structures and algorithms are needed to handle records of various execution information, by organizing them in certain forms of tables and linked structures. They are critical for reducing monitoring overhead. Additional information from the compiler and other involved components can be used to relate monitoring results with points in source programs. Monitoring results can also help select candidate jobs for further monitoring.

In summary, a number of trade-offs are involved in determining the monitoring techniques adopted for a particular application. Tools should be developed and used to help instrument the system, reduce the overhead, and interpret the monitoring results.

WORKLOAD SELECTION

To understand how a complex system works, one first needs to determine what to observe. Thus before determining how to monitor a system, one must determine what to monitor and why it is important to monitor them. This enables one to determine the feasibility of the monitoring, based on the perturbation and the cost, and then allows repeating and justifying the experiments.

Selecting candidate jobs to be run and measurements to be taken depends on the objectives of monitoring. For monitoring that is aimed at performance behavior, such as system tuning or task scheduling, one needs to select the representative load of work. For monitoring that is aimed at functional correctness, such as for debugging and fault-tolerance analysis, one needs to isolate the “buggy” or faulty parts.

A real workload best reflects system behavior under actual usage, but it is usually unnecessarily expensive, complicated, or even impossible to use as a test workload. Furthermore, the test results are not easily repeated and are not good for comparison. Therefore, a synthetic workload is normally used. For monitoring the functional correctness of a system, a *test suite* normally consists of data that exercise various parts of the system, and monitoring at those parts is set up accordingly. For performance monitoring, the load of work, rather than the actual jobs, is the major concern, and the approaches below have been used for obtaining test workloads (3,4,8).

Addition instruction was used to measure early computers, which had mainly a few kinds of instructions. *Instruction mixes,* each specifying various instructions together with their usage frequencies, were used when the varieties of instructions grew. Then, when pipelining, instruction caching, and address translation mechanisms made computer instruction times highly variable, *kernels,* which are higher-level functions, such as matrix inversion and Ackermann’s function, which represent services provided by the processor, came into use. Later on, as input and output became an important part of real workload, *synthetic programs,* which are composed of exerciser loops that make a specified number of service calls or I/O requests, came into use. For domain-specific kinds of applications, such as banking or airline reservation, *application benchmarks,* representative subsets of the functions in the application that make use of all resources in the system, are used. Kernels, synthetic programs, and application benchmarks are all called benchmarks. Popular benchmarks include the sieve kernel, the LINPACK benchmarks, the debit–credit benchmark, and the SPEC benchmark suite (4).

Consider monitoring the functional behavior of a system. For general testing, the test suite should have complete coverage, that is, all components of the system should be exercised. For debugging, one needs to select jobs that isolate the problematic parts. This normally involves repeatedly selecting more specialized jobs and more focused monitoring points based on monitoring results. For correctness checking at given points, one needs to select jobs that lead to different possible results at those points and monitor at those points. Special methods are used for special classes of applications; for example, for testing fault-tolerance in distributed systems, message losses or process failures can be included in the test suite.

For system performance monitoring, selection should consider the services exercised as well as the level of detail and representativeness (4). The starting point is to consider the system as a service provider and select the workload and metrics that reflect the performance of services provided at the system level and not at the component level. The amount of detail in recording user requests should be determined. Possible choices include the most frequent request, the frequency of request types, the sequence of requests with time stamps, and the average resource demand. The test workload should also be representative of the real application. Representativeness is reflected at different levels (3) at the physical level, the consumptions of hardware and software resources should be representative; at the virtual level, the logical resources that are closer to the user’s point of view, such as virtual memory space, should be representative; at the functional level, the test workload should include the applications that perform the same functions as the real workload.

Workload characterization is the quantitative description of a workload (3,4). It is usually done in terms of workload parameters that can affect system behavior. These parameters are about service requests, such as arrival rate and duration of request, or about measured quantities, such as CPU time, memory space, amount of read and write, or amount of communication, for which system independent parameters are preferred. In addition, various techniques have been used to obtain statistical quantities, such as frequencies of instruction types, mean time for executing certain I/O operations, and probabilities of accessing certain devices. These techniques include averaging, histograms, Markov models, and clustering. *Markov models* specify the dependency among requests using a transition diagram. *Clustering* groups similar components in a workload in order to reduce the large number of parameters for these components.

TRIGGERING MECHANISM

Monitoring can use either event-driven or sampling techniques for triggering and data recording (3). Event-driven techniques can lead to more detailed and accurate information, while sampling techniques are easier to implement and have smaller overhead. These two techniques are not mutually exclusive; they can coexist in a single tool.

Event-Driven Monitoring

An *event* in a computer system is any change of the system's state, such as the transition of a CPU from busy to idle, the change of content in a memory location, or the occurrence of a pattern of signals on the memory bus. Therefore, a way of collecting data about system activities is to capture all associated events and record them in the order they occur. A software event is an event associated with a program's function, such as the change of content in a memory location or the start of an I/O operation. A hardware event is a combination of signals in the circuit of a system, such as a pattern of signals on the memory bus or signals sent to the disk drive.

Event-driven monitoring using software is done by inserting a special trap code or hook in specific places of the application program or the operating system. When an event to be captured occurs, the inserted code causes control to be transferred to an appropriate routine. The routine records the occurrence of the event and stores relevant data in a buffer area, which is to be written to secondary storage and/or analyzed, possibly at a later time. Then the control is transferred back. The recorded events and data form an *event trace*. It can provide more information than any other method on certain aspects of a system's behavior.

Producing full event traces using software has high overhead, since it can consume a great deal of CPU time by collecting and analyzing a large amount of data. Therefore, event tracing in software should be selective, since intercepting too many events may slow down the normal execution of the system to an unacceptable degree. Also, to keep buffer space limited, buffer content must be written to secondary storage with some frequency, which also consumes time; the system may decide to either wait for the completion of the buffer transfer or continue normally with some data loss.

In most cases, event-driven monitoring using software is difficult to implement, since it requires that the application program or the operating system be modified. It may also introduce errors. To modify the system, one must understand its structure and function and identify safe places for the modifications. In some cases, instrumentation is not possible when the source code of the system is not available.

Event-driven monitoring in hardware uses the same techniques as in software, conceptually and in practice, for handling events. However, since hardware uses separate devices for trigger and recording, the monitoring overhead is small or zero. Some systems are even equipped with hardware that makes event tracing easier. Such hardware can help evaluate the performance of a system as well as test and debug the hardware or software. Many hardware events can also be detected via software.

Sampling Monitoring

Sampling is a statistical technique that can be used when monitoring all the data about a set of events is unnecessary, impossible, or too expensive. Instead of monitoring the entire set, one can monitor a part of it, called a *sample*. From this sample, it is then possible to estimate, often with a high degree of accuracy, some parameters that characterize the entire set. For example, one can estimate the proportion of time spent in different code segments by sampling program counters instead of recording the event sequence and the exact event count; samples can also be taken to estimate how much time different kinds of processes occupy CPU, how much memory is used, or how often a printer is busy during certain runs.

In general, sampling monitoring can be used for measuring the fractions of a given time interval each system component spends in its various states. It is easy to implement using periodic interrupts generated by a timer. During an interrupt, control is transferred to a data-collection routine, where relevant data in the state are recorded. The data collected during the monitored interval are later analyzed to determine what happened during the interval, in what ratios the various events occurred, and how different types of activities were related to each other. Besides timer interrupts, most modern architectures also include hardware performance counters, which can be used for generating periodic interrupts (9). This helps reduce the need for additional hardware monitoring.

The accuracy of the results is determined by how representative a sample is. When one has no knowledge of the monitored system, random sampling can ensure representativeness if the sample is sufficiently large. It should be noted that, since the sampled quantities are functions of time, the workload must be stationary to guarantee validity of the results. In practice, operating-system workload is rarely stationary during long periods of time, but relatively stationary situations can usually be obtained by dividing the monitoring interval into short periods of, say, a minute and grouping homogeneous blocks of data together.

Sampling monitoring has two major advantages. First, the monitored program need not be modified. Therefore, knowledge of the structure and function of the monitored program, and often the source code, is not needed for sampling monitoring. Second, sampling allows the system to spend much less time in collecting and analyzing a much smaller amount of data, and the overhead can be kept less than 5% (3,9,10). Furthermore, the frequency of the interrupts can easily be adjusted to obtain appropriate sample size and appropriate overhead. In particular, the overhead can also be estimated easily. All these make sampling monitoring particularly good for performance monitoring and dynamic system resource allocation.

IMPLEMENTATION

System monitoring can be implemented using software or hardware. Software monitors are easier to build and modify and are capable of capturing high-level events and relating them to the source code, while hardware monitors can capture rapid events at circuit level and have lower overhead.

Software Monitoring

Software monitors are used to monitor application programs and operating systems. They consist solely of instrumentation code inserted into the system to be monitored. Therefore, they are easier to build and modify. At each activation, the inserted code is executed and relevant data are recorded, using the CPU and memory of the monitored system. Thus software monitors affect the performance and possibly the correctness of the monitored system and are not appropriate for monitoring rapid events. For example, if the monitor executes 100 instructions at each activation, and each instruction takes 1 μs, then each activation takes 0.1 ms; to limit the time overhead to 1%, the monitor must be activated at intervals of 10 ms or more, that is, less than 100 monitored events should occur per second.

Software monitors can use both event-driven and sampling techniques. Obviously, a major issue is how to reduce the monitoring overhead while obtaining sufficient information. When designing monitors, there may first be a tendency to collect as much data as possible by tracing or sampling many activities. It may even be necessary to add a considerable amount of load to the system or to slow down the program execution. After analyzing the initial results, it will be possible to focus the experiments on specific activities in more detail. In this way, the overhead can usually be kept within reasonable limits. Additionally, the amount of the data collected may be kept to a minimum by using efficient data structures and algorithms for storage and analysis. For example, instead of recording the state at each activation, one may only need to maintain a counter for the number of times each particular state has occurred, and these counters may be maintained in a hash table (9).

Inserting code into the monitored system can be done in three ways: (*1*) adding a program, (*2*) modifying the application program, or (*3*) modifying the operating system (3). Adding a program is simplest and is generally preferred to the other two, since the added program can easily be removed or added again. Also, it maintains the integrity of the monitored program and the operating system. It is adequate for detecting the activity of a system or a program as a whole. For example, adding a program that reads the system clock before and after execution of a program can be used to measure the execution time.

Modifying the application program is usually used for event-driven monitoring, which can produce an execution trace or an exact profile for the application. It is based on the use of *software probes,* which are groups of instructions inserted at critical points in the program to be monitored. Each probe detects the arrival of the flow of control at the point it is placed, allowing the execution path and the number of times these paths are executed to be known. Also, relevant data in registers and in memory may be examined when these paths are executed. It is possible to perform sampling monitoring by using the kernel interrupt service from within an application program, but it can be performed more efficiently by modifying the kernel.

Modifying the kernel is usually used for monitoring the system as a service provider. For example, instructions can be inserted to read the system clock before and after a service is provided in order to calculate the turnaround time or response time; this interval cannot be obtained from within the application program. Sampling monitoring can be performed efficiently by letting an interrupt handler directly record relevant data. The recorded data can be analyzed to obtain information about the kernel as well as the application programs.

Software monitoring, especially event-driven monitoring in the application programs, makes it easy to obtain descriptive data, such as the name of the procedure that is called last in the application program or the name of the file that is accessed most frequently. This makes it easy to correlate the monitoring results with the source program, to interpret them, and to use them.

There are two special software monitors. One keeps *system accounting logs* (4,6) and is usually built into the operating system to keep track of resource usage; thus additional monitoring might not be needed. The other one is *program execution monitor* (4,11), used often for finding the performance bottlenecks of application programs. It typically produces an execution profile, based on event detection or statistical sampling. For event-driven precise profiling, efficient algorithms have been developed to keep the overhead to a minimum (12). For sampling profiling, optimizations have been implemented to yield an overhead of 1% to 3%, so the profiling can be employed continuously (9).

Hardware Monitoring

With *hardware monitoring,* the monitor uses hardware to interface to the system to be monitored (5,13–16). The hardware passively detects events of interest by snooping on electric signals in the monitored system. The monitored system is not instrumented, and the monitor does not share any of the resources of the monitored system. The main advantage of hardware monitoring is that the monitor does not interfere with the normal functioning of the monitored system and rapid events can be captured. The disadvantage of hardware monitoring is its cost and that it is usually machine dependent or at least processor dependent. The snooping device and the signal interpretation are bus and processor dependent.

In general, hardware monitoring is used to monitor the run-time behavior of either hardware devices or software modules. Hardware devices are generally monitored to examine issues such as cache accesses, cache misses, memory access times, total CPU times, total execution times, I/O requests, I/O grants, and I/O busy times. Software modules are generally monitored to debug the modules or to examine issues such as the bottlenecks of a program, the deadlocks, or the degree of parallelism.

A hardware monitor generally consists of a probe, an event filter, a recorder, and a real-time clock. The probe is high-impedance detectors that interface with the buses of the system to be monitored to latch the signals on the buses. The signals collected by the probe are manipulated by the event filter to detect events of interest. The data relevant to the detected event along with the value of the real-time clock are saved by the recorder. Based on the implementation of the event filter, hardware tools can be classified as fixed hardware tools, wired program hardware tools, and stored program hardware tools (5,13).

With *fixed hardware tools,* the event filtering mechanism is completely hard-wired. The user can select neither the events to be detected nor the actions to be performed upon detection of an event. Such tools are generally designed to measure specific parameters and are often incorporated into a system at design time. Examples of fixed hardware tools are timing meters and counting meters. *Timing meters* or *timers* measure the duration of an activity or execution time, and *counting meters* or *counters* count occurrences of events, for example, references to a memory location. When a certain value is reached in a timer (or a counter), an electronic pulse is generated as an output of the timer (or the counter), which may be used to activate certain operations, for instance, to generate an interrupt to the monitored system.

Wired-program hardware tools allow the user to detect different events by setting the event filtering logic. The event filter of a wired-program hardware tool consists of a set of logic elements of combinational and sequential circuits. The interconnection between these elements can be selected and manually manipulated by the user so as to match different signal patterns and sequences for different events. Thus wired-program tools are more flexible than fixed hardware tools.

With *stored-program hardware tools,* filtering functions can be configured and set up by software. Generally, a stored-program hardware tool has its own processor, that is, its own computer. The computer executes programs to set up filtering functions, to define actions in response to detected events, and to process and display collected data. Their ability to control filtering makes stored-program tools more flexible and easier to use. *Logical state analyzers* are typical examples of stored-program hardware tools. With a logical state analyzer, one can specify states to be traced, define triggering sequences, and specify actions to be taken when certain events are detected. In newer logical state analyzers, all of this can be accomplished through a graphical user interface, making them very user-friendly.

Hybrid Monitoring

One of the drawbacks of the hardware monitoring approach is that as integrated circuit techniques advance, more functions are built on-chip. Thus desired signals might not be accessible, and the accessible information might not be sufficient to determine the behavior inside the chip. For example, with increasingly sophisticated caching algorithms implemented for on-chip caches, the information collected from external buses may be insufficient to determine what data need to be stored. Prefetched instructions and data might not be used by the processor, and some events can only be identified by a sequence of signal patterns rather than by a single address or instruction. Therefore passively snooping on the bus might not be effective. Hybrid monitoring is an attractive compromise between intrusive software monitoring and expensive nonintrusive hardware monitoring.

Hybrid monitoring uses both software and hardware to perform monitoring activities (5,16–18). In hybrid monitoring, triggering is accomplished by instrumented software and recording is performed by hardware. The instrumented program writes the selected data to a hardware interface. The hardware device records the data at the hardware interface along with other data such as the current time. Perturbation to the monitored system is reduced by using hardware to store the collected data into a separate storage device.

Current hybrid monitoring techniques use two different triggering approaches. One has a set of selected memory addresses to trigger data recording. When a selected address is detected on the system address bus, the monitoring device records the address and the data on the system data bus. This approach is called *memory-mapped monitoring*. The other approach uses the coprocessor instructions to trigger event recording. The recording unit acts as a coprocessor that executes the coprocessor instructions. This is called *coprocessor monitoring.*

With memory-mapped monitoring, the recording part of the monitor acts like a memory-mapped output device with a range of the computer's address space allocated to it (5,16,17). The processor can write to the locations in that range in the same way as to the rest of the memory. The system or program to be monitored is instrumented to write to the memory locations representing different events. The recording section of the monitor generally contains a comparator, a clock and timer, an overflow control, and an event buffer. The clock and timer provide the time reference for events. The resolution of the clock guarantees that no two successive events have the same time stamp. The comparator is responsible for checking the monitored system's address bus for designated events. Once such an address is detected, the matched address, the time, and the data on the monitored system's data bus are stored in the event buffer. The overflow control is used to detect events lost due to buffer overflow.

With coprocessor monitoring, the recording part is attached to the monitored processor through a coprocessor interface, like a floating-point coprocessor (18). The recorder contains a set of data registers, which can be accessed directly by the monitored processor through coprocessor instructions. The system to be monitored is instrumented using two types of coprocessor instructions: data instructions and event instructions. Data instructions are used to send event-related information to the data registers of the recorder. Event instructions are used to inform the recorder of the occurrence of an event. When an event instruction is received by the recorder, the recorder saves its data registers, the event type, and a time stamp.

DATA ANALYSIS AND PRESENTATION

The collected data are voluminous and are usually not in a form readable or directly usable, especially low-level data collected in hardware. Presenting these data requires automated analyses, which may be simple or complicated, depending on the applications. When monitoring results are not needed in an on-line fashion, one can store all collected data, at the expense of the storage space, and analyze them off-line; this reduces the time overhead of monitoring caused by the analysis. For monitoring that requires on-line data analysis, efficient on-line algorithms are needed to incrementally process the collected data, but such algorithms are sometimes difficult to design.

The collected data can be of various forms (4). First, they can be either qualitative or quantitative. *Qualitative data* form a finite category, classification, or set, such as the set {busy, idle} or the set of weekdays. The elements can be ordered or unordered. *Quantitative data* are expressed numerically, for example, using integers or floating-point numbers. They can be discrete or continuous. It is easy to see that each kind of data can be represented in a high-level programming language and can be directly displayed as text or numbers.

These data can be organized into various data structures during data analysis, as well as during data collection, and presented as tables or diagrams. Tables and diagrams such as line charts, bar charts, pie charts, and histograms are commonly used for all kinds of data presentation, not just for monitoring. The goal is to make the most important information the most obvious, and concentrate on one theme in each table or graph; for example, concentrate on CPU utilization over time, or on the proportion of time various resources are used. With the advancement of multimedia technology, monitored data are now frequently animated. Visualization helps greatly in interpreting the measured data. Monitored data may also be presented using hypertext or hypermedia, allowing details of the data to be revealed in a step-by-step fashion.

A number of graphic charts have been developed specially for computer system performance analysis. These include Gantt charts and Kiviat graphs (4).

Gantt charts are used for showing system resource utilization, in particular, the relative duration of a number of Boolean conditions, each denoting whether a resource is busy or idle. Figure 1 is a sample Gantt chart. It shows the utilization of three resources: CPU, I/O channel, and network. The relative sizes and positions of the segments are arranged to show the relative overlap. For example, the CPU utilization is 60%, I/O 50%, and network 65%. The overlap between CPU and I/O is 30%, all three are used during 20% of the time, and the network is used alone 15% of the time.

A Kiviat graph is a circle with unit radius and in which different radial axes represent different performance metrics. Each axis represents a fraction of the total time during which the condition associated with the axis is true. The points corresponding to the values on the axis can be connected by straight-line segments, thereby defining a polygon. Figure 2 is a sample Kiviat graph. It shows the utilization of CPU and I/O channel. For example, the CPU unitization is 60%, I/O 50%, and overlap 30%. Various typical shapes of Kiviat graphs indicate how loaded and balanced a system is. Most often, an even number of metrics are used, and metrics for which high is good and for which low is good alternate in the graph.

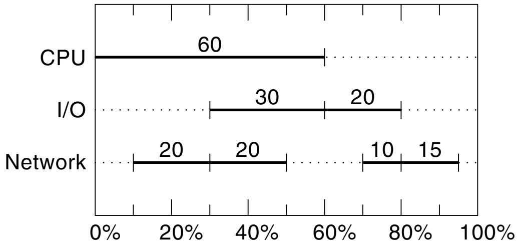


Figure 1. A sample Gantt chart for utilization profile.

APPLICATIONS

From the perspective of application versus system, monitoring can be classified into two categories: that required by the user of a system and that required by the system itself. For example, for performance monitoring, the former concerns the utilization of resources, including evaluating performance, controlling usage, and planning additional resources, and the latter concerns the management of the system itself, so as to allow the system to adapt itself dynamically to various factors (3).

From a user point of view, applications of monitoring can be divided into two classes: (*1*) testing and debugging, and (*2*) performance analysis and tuning. Dynamic system management is an additional class that can use techniques from both classes.

Testing and Debugging

Testing and debugging are aimed primarily at system correctness. Testing checks whether a system conforms to its requirements, while debugging looks for sources of bugs. They are two major activities of all software development.

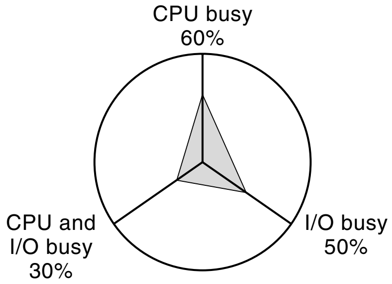


Figure 2. A sample Kiviat graph for utilization profile.

Systems are becoming increasingly complex, and static methods, such as program verification, have not caught up. As a result, it is essential to look for potential problems by monitoring dynamic executions.

Testing involves monitoring system behavior closely while it runs a test suite and comparing the monitoring results with the expected results. The most general strategy for testing is bottom-up: unit test, integration test, and system test. Starting by running and monitoring the functionality of each component separately helps reduce the total amount of monitoring needed. If any difference between the monitoring results and the expected results is found, then debugging is needed.

Debugging is the process of locating, analyzing, and correcting suspected errors. Two main monitoring techniques are used: single stepping and tracing. In single-step mode, an interrupt is generated after each instruction is executed, and any data in the state can be selected and displayed. The user then issues a command to let the system take another step. In trace mode, the user selects the data to be displayed after each instruction is executed and starts the execution at a specified location. Execution continues until a specified condition on the data holds. Tracing slows down the execution of the program, so special hardware devices are needed to monitor real-time operations.

Performance Evaluation and Tuning

A most important application of monitoring is performance evaluation and tuning (3,4,8,13). All engineered systems are subject to performance evaluation. Monitoring is the first and key step in this process. It is used to measure performance indices, such as turnaround time, response time, throughput, and so forth.

Monitoring results can be used for performance evaluation and tuning in as least the following six ways (4,6). First, monitoring results help identify heavily used segments of code and optimize their performance. They can also lead to the discovery of inefficient data structures that cause excessive amount of memory access. Second, monitoring can be used to measure system resource utilization and find performance bottleneck. This is the most popular use of computer system monitoring (6). Third, monitoring results can be used to tune system performance by balancing resource utilization and favoring interactive jobs. One can repeatedly adjust system parameters and measure the results. Fourth, monitoring results can be used for workload characterization and capacity planning; the latter requires ensuring that sufficient computer resources will be available to run future workloads with satisfactory performance. Fifth, monitoring can be used to compare machine performance for selection evaluation. Monitoring on simulation tools can also be used in evaluating the design of a new system. Finally, monitoring results can be used to obtain parameters of models of systems and to validate models. They can also be used to validate models, that is, to verify the representativeness of a model. This is done by comparing measurements taken on the real system and on the model.

Dynamic System Management

For a system to manage itself dynamically, typically monitoring is performed continuously, and data are analyzed in an on-line fashion to provide dynamic feedback. Such feedback can be used for managing both the correctness and the performance of the system.

An important class of applications is dynamic safety checking and failure detection. It is becoming increasingly important as computers take over more complicated and safety-critical tasks, and it has wide applications in distributed systems, in particular. Monitoring system state, checking whether it is in an acceptable range, and notifying appropriate agents of any anomalies are essential for the correctness of the system. Techniques for testing and debugging can be used for such monitoring and checking.

Another important class of applications is dynamic task scheduling and resource allocation. It is particularly important for real-time systems and service providers, both of which are becoming increasingly widely used. For example, monitoring enables periodic review of program priorities on the basis of their CPU utilization and analysis of page usage so that more frequently used pages can replace less frequently used pages. Methods and techniques for performance monitoring and tuning can be used for these purposes. They have low overhead and therefore allow the system to maintain a satisfactory level of performance.

MONITORING REAL-TIME, PARALLEL, AND DISTRIBUTED SYSTEMS

In a sequential system, the execution of a process is deterministic, that is, the process generates the same output in every execution in which the process is given the same input. This is not true in parallel systems. In a parallel system, the execution behavior of a parallel program in response to a fixed input is indeterminate, that is, the results may be different in different executions, depending on the race conditions present among processes and synchronization sequences exercised by processes (1). Monitoring interference may cause the program to face different sets of race conditions and exercise different synchronization sequences. Thus instrumentation may change the behavior of the system. The converse is also true: removing instrumentation code from a monitored system may cause the system to behave differently.

Testing and debugging parallel programs are very difficult because an execution of a parallel program cannot easily be repeated, unlike sequential programs. One challenge in monitoring parallel programs for testing and debugging is to collect enough information with minimum interference so the execution of the program can be repeated or replayed. The execution behavior of a parallel program is bound by the input, the race conditions, and synchronization sequences exercised in that execution. Thus data related to the input, race conditions, and synchronization sequences need to be collected. Those events are identified as process-level events (1). To eliminate the behavior change caused by removing instrumentation code, instrumentation code for process-level events may be kept in the monitored system permanently. The performance penalty can be compensated for by using faster hardware.

To monitor a parallel or distributed system, all the three approaches—software, hardware, and hybrid—may be employed. All the techniques described above are applicable. However, there are some issues special to parallel, distributed, and real-time systems. These are discussed below.

To monitor single-processor systems, only one event-detection mechanism is needed because only one event of interest may occur at a time. In a multiprocessor system, several events may occur at the same time. With hardware and hybrid monitoring, detection devices may be used for each local memory bus and the bus for the shared memory and I/O. The data collected can be stored in a common storage device. To monitor distributed systems, each node of the system needs to be monitored. Such a node is a single processor or multiprocessor computer in its own right. Thus each node should be monitored accordingly as if it were an independent computer.

Events generally need to be recorded with the times at which they occurred, so that the order of events can be determined and the elapsed time between events can be measured. The time can be obtained from the system being monitored. In single-processor or tightly coupled multiprocessor systems, there is only one system clock, so it is guaranteed that an event with an earlier time stamp occurred before an event with a later time stamp. In other words, events are totally ordered by their time stamps. However, in distributed systems, each node has its own clock, which may have a different reading from the clocks on other nodes. There is no guarantee that an event with an earlier time stamp occurred before an event with a later time stamp in distributed systems (1).

In distributed systems, monitoring is distributed to each node of the monitored system by attaching a monitor to each node. The monitor detects events and records the data on that node. In order to understand the behavior of the system as a whole, the global state of the monitored system at certain times needs to be constructed. To do this, the data collected at each individual node must be transferred to a central location where the global state can be built. Also, the recorded times for the events on different nodes must have a common reference to order them. There are two options for transferring data to the central location. One option is to let the monitor use the network of the monitored system. This approach can cause interference to the communication of the monitored system. To avoid such interference, an independent network for the monitor can be used, allowing it to have a different topology and different transmission speed than the network of the monitored system. For the common time reference, each node has a local clock and a synchronizer. The clock is synchronized with the clocks on other nodes by the synchronizer.

The recorded event data on each node can be transmitted immediately to a central collector or temporarily stored locally and transferred later to the central location. Which method is appropriate depends on how the collected data will be used. If the data are used in an on-line fashion for dynamic display or for monitoring system safety constraints, the data should be transferred immediately. This may require a high-speed network to reduce the latency between the system state and the display of that state. If the data are transferred immediately with a high-speed network, little local storage is needed. If the data are used in an off-line fashion, they can be transferred at any time. The data can be transferred after the monitoring is done. In this case, each node should have mass storage to store its local data. There is a disadvantage with this approach. If the amount of recorded data on nodes is not evenly distributed, too much data could be stored at one node. Building a sufficiently large data storage for every node can be very expensive.

In monitoring real-time systems, a major challenge is how to reduce the interference caused by the monitoring. Real-time systems are those whose correctness depends not only on the logical computation but also on the times at which the results are generated. Real-time systems must meet their timing constraints to avoid disastrous consequences. Monitoring interference is unacceptable in most real-time systems (1,14), since it may change not only the logical behavior but also the timing behavior of the monitored system. Software monitoring generally is unacceptable for real-time monitoring unless monitoring is designed as part of the system (19). Hardware monitoring has minimal interference to the monitored system, so it is the best approach for monitoring real-time systems. However, it is very expensive to build, and sometimes it might not provide the needed information. Thus hybrid monitoring may be employed as a compromise.

CONCLUSION

Monitoring is an important technique for studying the dynamic behavior of computer systems. Using collected run-time information, users or engineers can analyze, understand, and improve the reliability and performance of complex systems. This article discussed basic concepts and major issues in monitoring, techniques for event-driven monitoring and sampling monitoring, and their implementation in software monitors, hardware monitors, and hybrid monitors. With the rapid growth of computing power, the use of larger and more complex computer systems has increased dramatically, which poses larger challenges to system monitoring (20–22). Possible topics for future study include:

- New hardware and software architectures are being developed for emerging applications. New techniques for both hardware and software systems are needed to monitor the emerging applications.
- The amount of data collected during monitoring will be enormous. It is important to determine an appropriate level for monitoring and to represent this information with abstractions and hierarchical structures.
- Important applications of monitoring include using monitoring techniques and results to improve the adaptability and reliability of complex software systems and using them to support the evolution of these systems.

- Advanced languages and tools for providing more user-friendly interfaces for system monitoring need to be studied and developed.

BIBLIOGRAPHY

1. J. J. P. Tsai et al. *Distributed Real-Time Systems: Monitoring, Visualization, Debugging and Analysis*, New York: Wiley, 1996.
2. D. Ferrari, Workload characterization and selection in computer performance measurement, *IEEE Comput.*, **5**(7): 18–24, 1972.
3. D. Ferrari, G. Serazzi, and A. Zeigner, *Measurement and Tuning of Computer Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1983.
4. R. Jain, *The Art of Computer Systems Performance Analysis*, New York: Wiley, 1991.
5. P. McKerrow, *Performance Measurement of Computer Systems*, Reading, MA: Addison-Wesley, 1987.
6. G. J. Nutt, Tutorial: Computer system monitors, *IEEE Comput.*, **8**(11): 51–61, 1975.
7. L. Svobodova, *Computer Performance Measurement and Evaluation Methods: Analysis and Applications*, New York: Elsevier, 1976.
8. H. C. Lucas, Performance evaluation and monitoring, *ACM Comput. Surv.*, **3**(3): 79–91, 1971.
9. J. M. Anderson et al., Continuous profiling: Where have all the cycles gone, *Proc. 16th ACM Symp. Operating Syst. Principles*, New York: ACM, 1997.
10. C. H. Sauer and K. M. Chandy, *Computer Systems Performance Modelling*, Englewood Cliffs, NJ: Prentice-Hall, 1981.
11. B. Plattner and J. Nievergelt, Monitoring program execution: A survey, *IEEE Comput.*, **14**(11): 76–93, 1981.
12. T. Ball and J. R. Larus, Optimally profiling and tracing programs, *ACM Trans. Program. Lang. Syst.*, **16**: 1319–1360, 1994.
13. D. Ferrari, *Computer Systems Performance Evaluation*, Englewood Cliffs, NJ: Prentice-Hall, 1978.
14. B. Plattner, Real-time execution monitoring, *IEEE Trans. Softw. Eng.*, **SE-10**: 756–764, 1984.
15. B. Lazzerini, C. A. Prete, and L. Lopriore, A programmable debugging aid for real-time software development, *IEEE Micro*, **6**(3): 34–42, 1986.
16. K. Kant and M. Srinivasan, *Introduction to Computer System Performance Evaluation*, New York: McGraw-Hill, 1992.
17. D. Haban and D. Wybranietz, Real-time execution monitoring, *IEEE Trans. Softw. Eng.*, **SE-16**: 197–211, 1990.
18. M. M. Gorlick, The flight recorder: An architectural aid for system monitoring, *Proc. ACM/ONR Workshop Parallel Distributed Debugging*, New York: ACM, May 1991, pp. 175–183.
19. S. E. Chodrow, F. Jahanian, and M. Donner, Run-time monitoring of real-time systems, in R. Werner (ed.), *Proc. 12th IEEE Real-Time Syst. Symp.*, Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 74–83.
20. R. A. Uhlig and T. N. MudgeTrace-driven memory simulation: A survey, *ACM Comput. Surg.*, **29**(2): 128–170, 1997.
21. M. Rosenblum et al., Using the SimOS machine simulator to study complex computer systems, *ACM Trans. Modeling Comput. Simulation*, **7**: 78–103, 1997.
22. D. R. Kaeli et al., Performance analysis on a CC-PUMA prototype, *IBM J. Res. Develop.*, **41**: 205–214, 1997.

Yanhong A. Liu
Indiana University
Bloomington, Indiana

Jeffrey J. P. Tsai
University of Illinois
Chicago, Illinois

T

TEMPORAL DATABASES

A temporal database keeps time-varying data (past, present, and/or future) and the capabilities to query the stored data in contrast to a conventional database that stores only the current and recent data. The maintenance of temporal data is required in many application domains, such as document management, medicine, spatial data, insurance, financial data, and others.

Temporal extensions to common data models are available: relational, entity relationship, and object oriented data models. Relational databases, because of their widespread use and availability, are a good vehicle for demonstrating the subtle issues in modeling and querying temporal data. It also provides a base for handling temporal data.

An obvious approach for handling temporal data within the relational databases is to add one or more time columns to a relation. Although this solution seems to be simple, it does not address many subtle issues peculiar to temporal data (i.e., comparing database states at two different time instances or periods, capturing the periods for concurrent events and accessing the times beyond these periods, handling attributes with multiple values at a time instance, grouping and restructuring temporal data, etc). Addressing these issues and their implications for modeling and querying temporal data is important for the management of temporal data.

Research in modeling and querying temporal data started in the early 1980s and gained momentum from then on. Several conferences were dedicated to the temporal databases such as Temporal Aspects of Information Systems (1987), Time Conferences (1994 and on), workshops on an Infrastructure on Temporal Databases (1993), and Spatiotemporal Databases (1999–2006), as well as the Dagstuhl Seminar on Temporal Databases (1997). In the mean time, several books on temporal databases were published.

REPRESENTING TEMPORAL DATA

Time

Time is continuous, dense, linearly increasing, and a total order under $\leq$. So, time is the set of time points (instances) $T = \{\ldots t_0 \ldots t_i, t_i + 1, t_i + n, Now \ldots \infty\}$ where the symbol *Now* represents the current time and its value advances as the clock ticks. Time points beyond *Now* represent the future until infinity (∞). Figure 1 gives the time line. The difference between two consecutive time points (t_i and $t_i + 1$) is one unit depending on the time granularity used, such as day, month, year, and so on. Moreover, the inside of a time unit is invisible regardless of its duration unless a finer time granularity is used. It is also customary to designate a relative time origin such as t_0 depending on the application domain.

Any subset of *T* is a *temporal set*. Time between two time points, t_i and $t_i + n$, is represented either as a *closed interval* $[t_i, t_i + n]$ or as a *half-open interval* $[t_i, t_i + n + 1)$. An example for the temporal set is {2/07, 3/07, 4/07, 7/07, 8/07, 11/07} and [2/07, 4/07], [7/07, 9/07), and [11/07 12/07) are the corresponding intervals. A *temporal element* (1) is a finite union of intervals and it allows different representations for the same set of time points depending on the intervals used. A noteworthy representation for a temporal element is the one that contains the maximal intervals that are not adjacent or do not overlap. One possible temporal element, the maximal one for the temporal set given earlier is {[2/07, 5/07) ∪ [7/07, 9/07) ∪ [11/07 12/07)}. Temporal elements are closed under set-theoretic operations whereas intervals are not. Any interval, temporal element, or temporal set that includes *Now* expands as the time advances. Moreover, *Now* can be replaced by the special symbols, "*Until changed*" or infinity (∞).

Various aspects of objects and their attribute values exist with respect to time (2). The time at which an attribute value becomes effective is called *valid time*. However, *transaction time* refers to the time a data value is recorded in the database. Other times of interest are available depending on the application domain, such as birth date, delivery date, or decision date, that can be stored as the *user defined time,* which is an ordinary attribute whose data type is "date". The rest of this entry focuses on the valid time and transaction time aspects of temporal data.

Temporal Data

Events (transactions) take place concurrently or in increasing time order and may cause a change in the database state in the form of insertions, deletions, or modifications. In Fig. 1, related events e_1, e_2, and e_3 take place at time instances t_2, t_5, and t_6, respectively. The effect of event e_1 starts at time instant t_2 and continues until time instant t_5 at which time a new value is created by the event e_2. At time instance t_6 there is another value created by the event e_3 whose effect continues until another event changes it.

Attribute values may have different behaviors. An attribute value, once assigned, may never change anytime afterwards (i.e., it stays constant). The attribute value may only be valid at that time instance only. Still another possibility is that it may change continuously with time. Finally, the attribute value is valid for a while and then changes to different values at other times.

The set *U* is all of the atomic values such as reals, integers, character strings, and the value null. Time-varying data are commonly represented by time-stamping values in *U*. A fact has a time and a value component. It can be represented as a temporal atom (3), which has three forms. A *bitemporal atom* is a triplet, $<Tt, Vt, v>$ where Tt and Vt are timestamps $Tt \subseteq T$, $Vt \subseteq T$, and $v \in U$. The symbol Tt is the transaction time, Vt is the valid time, and v is a value. A bitemporal atom asserts that the value v is valid during Vt and is recorded in the database at time Tt. One

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

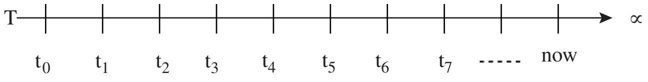


Figure 1. Time Points.

timestamp in a bitemporal atom may be omitted. $<Tt, v>$ is a *transaction time atom* and asserts that the value v is recorded in the data base at time Tt. Similarly, $<Vt, v>$ is a *valid time atom* that asserts that the value v is valid during Vt. In a temporal atom, the timestamps can be time points, time intervals, temporal sets or temporal elements. As an example, consider the projects and their budgets. Figure 2 depicts the budget history of a project as temporal atoms [Fig. 2(b)] and the sequence of events that created that history [Fig. 2(a)]. Note that the valid time atoms give the history after the most recent changes [Fig. 2(d)]. However, transaction time atoms [Fig. 2(c)] reflect the changes on the most recent values but not changes in the past values. In other words, event e_6 can not be stored in a transaction time database. Note that Fig. 2(b) is a complete bitemporal history of the events given in Fig. 2(a).

Types of Temporal Databases

Transaction time and valid time are the two time dimensions commonly used in a temporal database (2). A traditional database is a database of snapshot relations because it does not contain any time dimensions and keeps the most recent data [Fig. 3(a)]. A database that has relations with only valid time dimension is called a *Valid Time* Database or a *Historical* Database [Fig. 3(b)]. It gives a correct state of the history as we know it as of *Now*. Considering the data given in Fig. 2(d), events e_5 and e_6 replace the past values. Any correction in an attribute value replaces it, and there is no trace of the correction in the database. A *Transaction Time* database stores relations with only the transaction time and it is append-only [Fig. 3(c)]. Valid time and transaction time relations have the same form, but their content evolves differently; the former allows replacement of stored (past) data, and the latter does not. Transactions append attribute values that are new values or corrections for the most recent attribute value stored in the database. Past values may not be corrected as in the case of event e_5 in Fig. 2(c), which can not be recorded in the database. It is possible to roll back a transaction time database to any of its states in the past. A *bitemporal* database supports both the transaction time and the valid time [Fig. 3(d)]. It allows retroactive and postactive changes and keeps a complete history of these changes. A bitemporal database can be rolled back to any of its states in the past. The term temporal database is used loosely in the literature to mean a bitemporal, transaction time, or valid time database.

REPRESENTING TEMPORAL DATA

A fundamental issue in modeling temporal data is the representation of temporal atoms in the relational data model or in other data models. Representing a temporal atom involves adding timestamps to the relations, tuples, or attributes. The choice has implications for both modeling—the type of relations used and querying the temporal data. It also determines the expressive power of a temporal data model and its query languages (4).

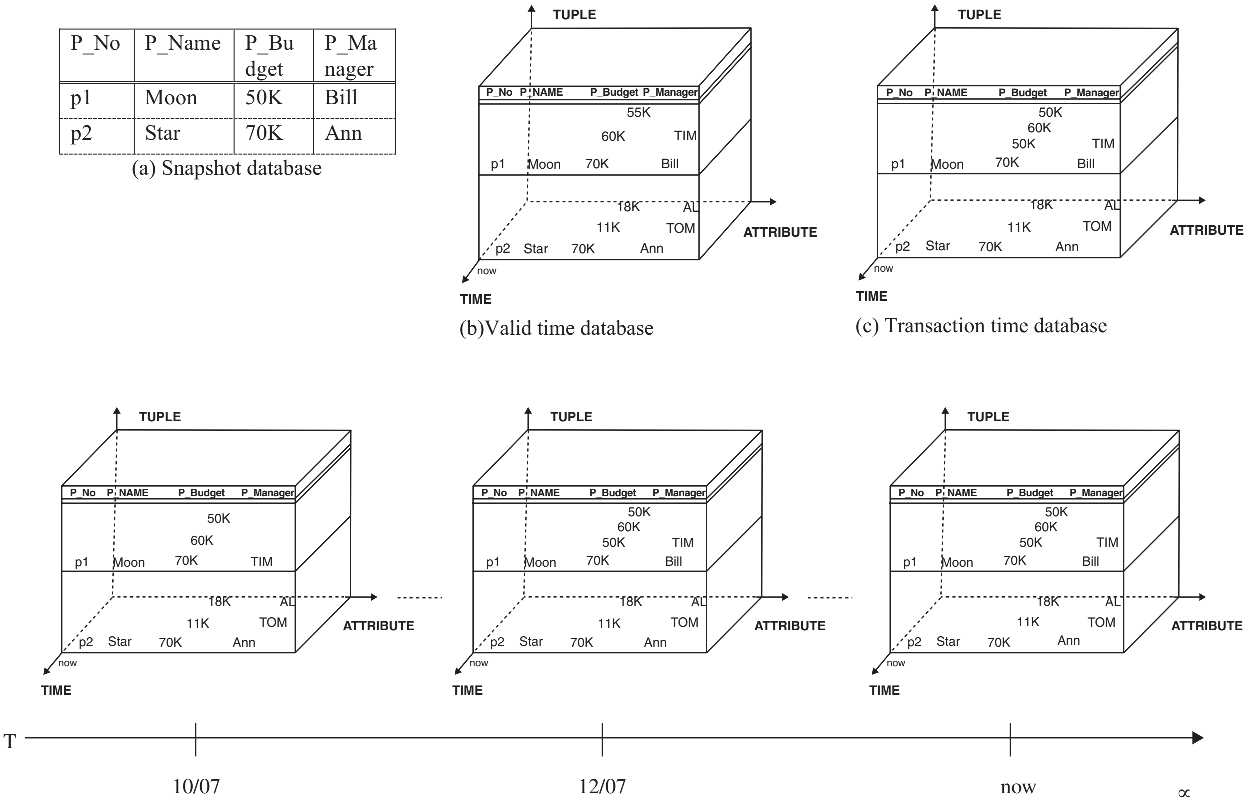
For the sake of simplicity, valid time relations are considered to demonstrate temporal issues. The discussion extends to the transaction-time databases in a straightforward manner. The same applies to bitemporal databases, although it is more complicated.

In tuple time stamping (TTS), temporal atoms are broken into their time and value components. The relation scheme is augmented with two additional columns to represent the end points of time intervals (periods). It is also possible to store a time interval, temporal element, or a temporal set in one column as a composite data type. Figure 4 gives the project data in TTS: Project_N(P_No, P_Name), Project_B(P_No, P_Budget), and Project_M(P_No, P_Manager) relations. P_No, P_Name, P_Budget, and P_Manager denote project number, project name, project budget, and project manager, respectively. Obviously, budget and manager are temporal attributes, whereas project number and project name

Posting Time	**(a)** Events	**(b)** Bitemporal atoms
2/07	e_1: The project starts with a budget of 50K at 2/2007.	<[2/07, 5/07), [2/07, 5/07), 50K>
5/07	e_2: Project's budget increases to 60K at 5/2007.	<[5/07, 7/07), [5/07, 7/07), 60K>
7/07	e_3: Project is suspended at 7/2007.	
10/07	e_4: Project resumes with a budget of 50K at 10/2007.	<[10/07, 11/07), [10/07, 11/07), 50K>
12/07	e_5: At 11/2007, an error is discovered. The project's last budget is 70K.	<[12/07, *Now*], [10/07, *Now*], 70K>
2/08	e_6: At 12/2007 an error is discovered. The project's budget at 2/2007 was 55K, not 50K.	<[2/08, *Now*], [2/07, 5/07), 55K>

(c) Transaction time atoms	**(d)** Valid time atoms
<[2/07, 5/07), 50K>	<[2/07, 5/07), 55K>
<[5/07, 7/07), 60K>	<[5/07, 7/07), 60K>
<[10/07, 11/07), 50K>	<[10/07, *Now*], 70K>
<[12/07, *Now*], 70K>	

Figure 2. Different types of temporal atoms.



P_No	P_Name	P_Budget	P_Manager
p1	Moon	50K	Bill
p2	Star	70K	Ann

(a) Snapshot database

(b)Valid time database

(c) Transaction time database

(d) Bitemporal database

Figure 3. Types of temporal databases.

are not. Project_B and Project_M relations are augmented with two additional attributes, *Start* and *End*. These columns can be explicit (i.e., like any other ordinary attribute) or implicit (i.e., special time attributes maintained by the system) (2). This, causes limitations as explained below.

In attribute time stamping (ATS), attribute values are temporal atoms, and the entire history of an attribute is stored as a set of temporal atoms that requires N1NF relations that allow relations as attribute values in contrast to 1NF (flat) relations. Moreover, it is possible to attach a timestamp to any part of a N1NF relation. In this case, the value part of a temporal atom becomes a N1NF relation which is beyond the focus of this entry. Figure 5 gives the project data in attribute time stamping. Three separate relations in tuple timestamping are combined into one single relation, namely Project in ATS: Project(P_No, P_Name, P_Budget, P_Manager). Temporal attributes P_Budget and P_Manager are sets of temporal atoms, whereas P_No and P_Name are constant and atomic. These attributes can also be represented as temporal atoms where the timestamp is the life of the database (i.e., <$[t_0,\infty)$, p1> where t_0 is the time origin) (5). The choice depends on the type of temporal query language designed.

Time stamping a relation visualizes a temporal relation as a sequence of snapshot relation instances at each time point (i.e., indexed by time).

In case of tuple and attribute timestamping, time points cannot capture the full extent of the temporal reality. Time points have limitations as a timestamp because extrapolating the attribute values beyond the time points creates undue complications. These complications occur because the starting time of a value is not sufficient to indicate the whole period over which that value is valid, as the end of this period is indicated by the starting time of the value in another tuple. If the history is discontinuous, then it requires special null values to determine the validity period of a data value correctly. However, handling these null values in a query language is complicated (6). In the special case when an attribute is valid over just one time point, time points may be used as a timestamp. In the literature, this special case is called an *Event* relation (2).

One noteworthy aspect of data presented in Fig. 5 is that the timestamps are glued to values in temporal atoms (7). In forming new relations, these timestamps stay with the attribute values. However, in implicit tuple time stamping, a timestamp is glued to a tuple. Therefore, it restricts the time of a new tuple created from two constituent tuples, that is, each tuple may not keep its own timestamp, and a new timestamp needs to be assigned to the resulting tuple. It is possible to consider unglued timestamps (i.e., explicit timestamp attributes) and allow multiple timestamps in a tuple. In this case, two tuples may be combined to form a new tuple, each of which carries its own time reference. However, it would be difficult to provide support for this in a query language, and the user would need to keep track of these separate time references.

P_No	P_Name
p1	Moon
p2	Star

(a) Project_N Relation

P_No	P_Budget	Start	End
p1	55K	2/07	5/07
p1	60K	5/07	7/07
P1	70K	10/07	*Now*
p2	70K	4/07	*Now*

(b) Project_B Relation

P_No	P_Manager	Start	End
p1	Tom	2/07	6/07
p1	Bill	6/07	*Now*
p2	Ann	4/07	*Now*

(c) Project_M Relation

Figure 4. Project data in tuple time stamping.

Note that in tuple time stamping, a relation may only contain attributes that change at the same time (i.e., attributes that change at different times require different relations).

Temporal Grouping

In both tuple and attribute timestamping, there is a unique representation of temporal data (5, 7, 8). The Project relation of Fig. 5 is a unique and compact representation of the projects' data in which each tuple contains the entire history of a project. This represantation is called *temporally grouped* (1). The same data are broken into several tuples in Fig. 4. Tuples that have the same project number are related and belong to the same object (entity) or relationship. This table is also a unique representation of temporal data in tuple timestamping. Hence, tuple time stamping is *temporally ungrouped* (1).

The project number is a *temporal grouping identifier* because it is unique. It does not change with time and identifies the tuples that belong to the same object. In case of attribute timestamping, only one tuple exists for each project number value. Temporal grouping identifier also serves as a temporal primary key, which is the attribute(s) that serves as a primary key for each snapshot of a temporal relation. Unique representations in TTS and ATS, as given in Figs. 4 and 5 are suitable for most applications. However, other temporal relation forms in between may be of theoretical significance or suitable for certain type of applications.

In a conventional database, the primary key values are expected to be constant. However, in a temporal database the value of a temporal grouping (object) identifier may change. In this case, a temporal grouping (object) identifier becomes a set of values, and a N1NF temporal relation can store it (8). However, using 1NF relations requires a separate tuple for each change in the temporal grouping (object) identifier value. The fact that these tuples are related and belong to the same object (temporal group) is lost, and this information can only be recovered by adding an artificial, nonchanging object identifier to each tuple.

P_No	P_Name	P_Budget	P_Manager
p1	Moon	<[2/07, 5/07), 55K> <[5/07, 7/07), 60K> <[10/07, *Now*], 70K>	<[2/07, 6/07), Tom> <[6/07, *Now*], Bill>
p2	Star	<[4/07, *Now*], 70K>	<[4/07, *Now*], Ann>

Figure 5. Project relation in attribute time stamping.

Weak Temporal Relations. For the temporal data in unique representation, many other possible representations can be obtained by taking subsets of timestamps and creating several tuples for the same unique tuple. In other words, one tuple in the unique representations is broken into several tuples whose timestamps are the subsets of the timestamp(s) of the original tuple. These relations are called *weak relations* (4, 9) because they contain the same data but are not identical to a relation in its unique representation. For instance, a weak relation may contain tuples (p1, 55K, 2/07, 4/07) and (p1, 55K, 3/07, 5/7) in place of the first tuple of the relation Project_B in Fig. 4. These two tuples are *value equivalent* because their values are identical, but their timestamps are different. Value equivalent tuples can be combined (collapsed) back to a unique tuple (6).

Homogenous Temporal Relations. A homogenous tuple has the same time reference in its attributes (5). A temporal relation is homogenous if all its tuples are homogeneous. This restriction simplifies the model and allows modeling and querying the homogeneous part of the real world unambiguously. If an object's attributes have values over different periods of time, then the attributes of that object can only be represented over the common time periods of its attributes. Moreover, Cartesian product operation can only be defined for the common time period of the operand tuples. Let t_1 and t_2 be tuples from homogenous temporal relations and τ_1 and τ_2 be their timestamps, respectively. Cartesian product of these two tuples can only be defined over $\tau_1 \cap \tau_2$. Parts of τ_1 and τ_2 outside of their intersection are not accessible (i.e., $\tau_1 - \tau_2$ or $\tau_1 - \tau_2$. If τ_1 and τ_2 do not intersect, then it is not even possible to extract information from these two tuples at all. Temporal relations in TTS are homogenous by definition, whereas temporal relations in ATS may or may not be homogenous (8).

Entities and Relationships

In tuple timestamping, 1NF relations are sufficient to represent both the entity types and their relationships. Each temporal attribute generally requires a separate relation. However, in attribute timestamping, N1NF relations are needed, and one level of nesting can model an entity type where a temporal attribute is a set of temporal atoms.

One or two levels of nesting can represent the history of a relationship. In modeling a many-to-many relationship, identifiers of the related objects form a grouping identifier for the relationship. Consider a relationship relation between projects and the tools they use, Project_Tool(P_No, T_No, Budget) where Budget denotes the allocation of a tool to a project. The attribute combination (P_No, T_No) is a temporal grouping identifier for the relationship between projects, and Tools and Budget will be a set of temporal atoms. However, this relationship can be embedded into Project or Tool relations, which requires two levels of nesting: one level for the history of the relationship and another level for the history of budget.

TEMPORAL QUERY LANGUAGES

Many Temporal Relational Algebra (TRA) and Temporal Relational Calculus (TRC) languages are available for temporal databases. The languages depend on how the temporal data are represented (i.e., how the temporal atoms are stored in relations, and the type of timestamps and where they are attached). This in turn determines how temporal data are processed and possible evaluation (semantics) of temporal query languages. Two commonly adopted approaches are as follows: Snapshot (point or sequenced) evaluation that manipulates the snapshot relation at each time point, like temporal logic (1, 5, 6) or traditional (nonsequenced) evaluation that manipulates the entire temporal relation much like the traditional relational languages (7). It is also possible to mix these approaches. Snapshot evaluation requires homogeneous tuples; otherwise, it may have null values that leads to complications (5).

If temporal relations are in 1NF and temporal atoms are already broken into their components (i.e., TTS), then these components are referred directly in TRA and TRC. Moreover, for reaching time points, interval or temporal elements can be expanded to time points or vice versa. In case N1NF relations and temporal atoms are used (i.e., ATS), operations are needed to flatten (unnest) them to 1NF relations or transforming (nesting) 1NF relations back to N1NF relations. Operations for breaking and forming new temporal atoms are also required.

Temporal query languages are defined independent of time granularities. However, if operand relations are defined on different time granularities, then a granularity conversion is required as part of processing the operation.

Temporal Relational Algebras

The syntax of temporal algebras closely follows the syntax of traditional relational algebra. The five basic Temporal Algebra operations, $\cup^t$, $-^t$, π^t, σ^t, and x^t in snapshot evaluation are given below (1, 5). Let Q, R, and S be relations that may be defined in ATS or TTS. Assume Q, R, and S in ATS are Project, $\pi_{\text{P\_No, Buget}}$(Project_Tool), and $\pi_{\text{P\_No, Buget}}$(Project), respectively.

- $R \cup^t S_t$ is $R_t \cup S_t$ for all t in T
- $R -^t S_t$ is R_t-S_t for all t in T
- $\pi^t{}_{A1,A2,\ldots,An}$ (R) is $\pi_{A1,A2,\ldots,An}$ (R_t) for all t in T
- $\sigma^t_F(R)$ is $\sigma_F(R_t)$ for all t in T; Formula F includes traditional predicates and temporal predicates like *Before*, *After*, *Overlaps*, etc
- $R \times^t Q$ is $R_t \times Q_t$ for all t in T

In temporal algebras that use traditional evaluation, $\cup^t$, $-^t$, π^t, σ^t, and x^t may be defined exactly the same as the relational algebra operations or they may include temporal semantics. The temporal set-theoretic operations may be defined by considering the value equivalent tuples. Let {(p1, <[2/07,11/07), 55K>)} and {(p1, <[6/07,8/07), 55K>)} be tuples in R and S, respectively. These two tuples are value equivalent. Set union operation, $R \cup^t S$ combines value equivalent tuples into one single tuple and carries the remaining tuples directly into the result. Considering the former tuples the result is {(p1,<[2/07,11/07), 55K>)}. Set difference however, $R -^t S$ removes the common portion of the timestamps for the value equivalent tuples. For the above tuples the result is {p1, <[2/07,6/07), 55K>), (p1, <[8/07,11/07), 55K>)}. If the temporal semantics is not included in operations, the temporal coalescing operation combines value equivalent tuples into a single tuple.

The definition of temporal projection (π^t) is straightforward. However, it may generate value-equivalent tuples much like the traditional projection operation creates duplicate tuples, and it is costly to eliminate them.

The formula F in the Selection operation [$\sigma^t_F(Q)$] may include time points, the end points of intervals, or temporal elements as well as temporal predicates like *Before*, *After*, *Overlaps*, and so on. It is possible to simulate the temporal predicates by conditions referring to time points or end points of intervals.

Other temporal algebra operations, such as temporal set intersection or temporal join, are defined similarly. Many different versions of temporal join are available. Intersection join is computed over the common time of operand relations. For instance, if {(p1, Moon, <[1/07, 5/07), 80K>, <[1/07, 4/07), Tom>)} is a tuple in Q, then the natural join ($Q \bowtie R$) contains the tuple {(p1, Moon, <[1/07, 5/07), 80K>, <[1/07, 4/07), Tom>, <[2/07,11/07), 55K>)}. If this were an intersection natural join, times of the attributes in this tuple would be restricted to their common time period [2/07, 4/07). It is also possible to define temporal outer joins. Temporal aggregates may be calculated at each time point or cumulatively over a period(s) of time. Temporal query optimization is based on temporal algebraic identities similar to traditional query optimization.

Time Slice operation in valid time databases slices (cuts) the timestamp of a temporal attribute by a specified constant timestamp or by the timestamp of another temporal attribute (7). Time slicing every attribute in a temporal relation by a time point (or timestamp) generates a snapshot (or temporal relation) at that time point (or timestamp). The time slice operation conveniently implements the "when" predicate in natural languages. The symmetric operation in transaction time databases is *Rollback* (10). Applied on a temporal attribute, this operation rolls it back to the designated time point (or timestamp). Applying it on all the attributes of a temporal relation rolls it back to designated time point (or timestamp). In a bitemporal algebra, both operations are available.

Temporal Relational Calculus

A temporal tuple (domain) relational calculus expression is $\{t \mid \psi(t)\}$, where t is a free tuple (domain) variable and $\psi(t)$ is a well-formed formula that contains traditional relational calculus terms and formulas in addition to their temporal counterparts. They provide references to the end points of intervals or time points in temporal elements or temporal sets. In case of N1NF temporal relations, a set membership operator ($\in$) is needed for reaching the elements in a set. Additionally, set formation is achieved by assigning a well-formed formula to a component of a tuple variable or a domain variable, much like the aggregation operation in relational calculus. However, the calculus language with the set membership operator and the set formation formula is more expressive than the relational algebra that has nest and unnest operators (8). The capability to refer to the time reference of an attribute or a formula is useful in temporal relational calculus and algebra expressions

Temporal Structured Query Language (SQL)

SQL2 includes a diverse set of temporal data types: date, time, timestamp, and interval. However, temporal support in SQL2 is limited and left mostly for the applications programming. Temporal atoms cannot be implemented directly, and only tuple timestamping is available. However, SQL3 allows direct implementation of temporal atoms and therefore supports both attribute and tuple time stamping. Temporal atoms can be defined easily as new data types. Thus, current database systems that contain object relational features and adhere to SQL3 standard can be used to implement any of the relations given in Figs. 4 and 5. Moreover, SQL2 and SQL3 include a period data type whose values are time durations in contrast to time instants. Other than some basic operations on temporal types, the details of temporal support are again left for the applications.

Several temporal extension proposals to SQL have been suggested. One noteworthy example is TSQL2 (8). These extensions augment SQL statements with "*when*" clauses for valid time or bitemporal atoms formation and/or to implement temporal conditions. Another addition is the "*as of*" clause that accomplishes rollback to a time in the past in case of transaction time or bitemporal relations. Time-by-example temporal query language proposal follows the syntax and semantics of Query by Example (QBE) and QBE-like languages (11).

REQUIRMENTS FOR TEMPORAL DATA MODELS

Many properties are desirable for temporal databases (4). Let DB_t denote the database state at time t, where t may be a time point, time interval, temporal element, or temporal set):

1. The data model should be capable of modeling and querying the database at any instance of time (i.e., D_t). The data model should at least provide the modeling and querying power of a 1NF relational data model. Note that when t is *now*, D_t corresponds to traditional database.
2. The data model should be capable of modeling and querying the database at two different times (i.e., D_t and D'_t, where $t \neq t'$). This model should be the case for the different types of timestamps.
3. The data model should allow different periods of existence in attributes within a tuple (i.e., nonhomogenous (heterogeneous) tuples should be allowed).
4. The data model should allow multivalued attributes at any time point (i.e., in D_t).
5. A temporal query language should have the capability to return the same type of objects it operates on.
6. A temporal query language should have the capability to regroup the temporal data according to a different temporal attribute. Homogenous relations simplify the regrouping operation.
7. The data model should be capable of expressing set-theoretic operations, as well as set comparison tests, on the timestamps, be it time points, time intervals, temporal sets, or temporal elements.

TEMPORAL INTEGRITY CONSTRAINTS

Each object, such as a project, exists in a certain period of time, which is a subset of $[0, \infty)$. Call this period as the object's *life*, denoted as $l(o)$ for the object o. Any temporal atom that is the value of an attribute of object o contains a timestamp that is a subset of $l(o)$. Constant attributes of an object o do not change during $l(o)$, whereas time-dependent attributes assume different values during this period. They may even be null in parts or all of $l(o)$. Moreover, a constant attribute, such as a project name, can be represented with no timestamp, in which its time reference is implied as $l(o)$ or a temporal atom whose time reference is $l(o)$.

Existence and referential integrity also have their temporal counterparts. Temporal grouping identifier or temporal key in a temporal relation may not be null at any time instance. Also, an object may only participate in a relationship only during its life. Let p and t be the tuples in Project and Project_Tool relations, respectively. Referential integrity requires that $l(p) \subseteq l(t)$ in addition to the restrictions on the respective attribute components of p and t.

DESIGNING TEMPORAL RELATIONS

Traditional relational database design into BCNF and 4NF, and the design of N1NF relations are based on the functional and multivalued dependencies. The same principles can also be applied on the design of temporal databases. A functional dependency in a snapshot database turns into a multivalued dependency in a temporal database. A multivalued dependency in a snapshot database is still a multivalued dependency in a temporal database.

In attribute time stamping, temporal relations that are in nested normal form (10) do not suffer from insertion, deletion, and update anomalies. The Project relation in Fig. 5 is in nested normal form.

In tuple timestamping, 4NF decomposition places each temporal attribute into a separate relation because the temporal grouping identifier functionally determines constant attributes (i.e., P_Name) and multidetermines each temporal attribute (i.e., P_Budget, and P_Manager). Decomposing Project relation of Fig. 5 gives the Project_N, Project_B, and Project_M relations of Fig. 4. Project_N is in BCNF, and Project_B and Project_M are in 4NF.

CURRENT STATE OF TEMPORAL DATABASES

Many index structures have been proposed for accessing temporal data. Modeling and index structures for spatiotemporal data are also studied. Research in temporal query processing and optimization has been limited and needs more investigation.

Proof-of-concept temporal prototypes have been implemented on top of commercial database management systems. However, no commercial temporal database product or a temporal add-on is available in the market yet. Developing temporal database applications or temporal database add-ons are within the reach of application developers through the use of recent object relational database management systems that are available in the market.

BIBLIOGRAPHY

1. J. Clifford, A. Croker, and A. Tuzhilin, On completeness of historical data models, *ACM Trans. Database Sys.*, **19**(1): 64–116, 1994.
2. R. Snodgrass, The temporal query language Tquel, *ACM Trans. Database Sys.*, **12**(2): 247–298, 1987.
3. A. U. Tansel, Temporal relational data model, *IEEE Trans. Knowledge Database Eng.*, **9**(3): 464–479, 1997.
4. A. U. Tansel and E. Tin, Expressive power of temporal relational query languages, *IEEE Transactions on Knowledge and Database Engineering,* **9**(1): 120–134, 1997.
5. S. K. Gadia, A homogeneous relational model and query languages for temporal databases, *ACM Trans. Database Sys.*, **13**(4): 418–448, 1988.
6. M. H. Böhlen, C. S. Jensen, R. T. Snodgrass, Temporal statement modifiers. *ACM Trans. Database Sys.*, **25**(4): 407–456, 2000.
7. J. Clifford, and A. U. Tansel, On an algebra for historical relational databases: Two views, in *Proc. of ACM SIGMOD International Conference on Management of Data*, 247–265, 1985.
8. R. T. Snodgrass (ed.), *The TSQL2 Temporal Query Language*, Norwell, MA: Kluwer 1995.
9. G. Bhargava, S. K. Gadia, Relational database systems with zero information loss. *IEEE Trans. Knowledge & Data Engineering*, **5**(1): 76–87, 1993.
10. M. Z. Ozsoyoglu, L-Y Yuan, A new normal form for nested relations, *ACM Trans. Database Sys.*, **12**(1): 111–136, 1987.
11. A. U. Tansel, M. E. Arkun, and G. Ozsoyoglu, Time-by-example query language for historical databases, *IEEE Trans. Software Eng.***15**(4): 464–478, 1989.

FURTHER READING

C. Betteni, S. Jajodia, S. Wang, *Time granularities in databases, data mining and temporal reasoning*, Springer Verlag, 1998.

O. Etzion, S. Jajodia, S. Sripada (eds.), *Temporal Databases: Research and Practice*, New York. Springer Verlag, 1998.

C. S. Jensen, et al., *TDB Glossary, Availabble:* http://www.cs.aau.dk/~csj/Glossary/index.html, 1994.

R. T. Snodgrass, Developing time oriented Applications in SQL, New York Morgan Kaufmann, 2000.

A. U. Tansel, et al. (eds.), *Temporal Databases: Theory, Design and Implementation*, San Eranciscon, Benjamin/Cummings, 1993.

ABDULLAH UZ TANSEL
Baruch College and the
Graduate Center
City University of New York
New York, New York

T

TRANSACTION PROCESSING IN MOBILE, HETEROGENEOUS DATABASE SYSTEMS

INTRODUCTION

The proliferation of mobile devices has brought about the realization of ubiquitous access to information through the wide breadth of devices with different memory, storage, network, power, and display capabilities. Additionally, with the explosion of data available via the Internet and in private networks, the diversity of information that is accessible to a user at any given time is expanding rapidly.

Current multi-database systems (MDBS) are designed to allow timely and reliable access to large amounts of heterogeneous data from different data sources. Within the scope of these systems, multi-database researchers have addressed issues such as autonomy, heterogeneity, transaction management, concurrency control, transparency, and query resolution (1). These solutions were based on fixed clients and servers connected over a reliable network infrastructure. However, the concept of *mobility*, where a user accesses data through a remote connection with a portable device, has introduced additional complexities and restrictions (2). These include (*1*) limited network connections, (*2*) processing and resource constraints, and (*3*) the problem of effectively locating and accessing information from a multitude of sources. An MDBS with such additional restrictions is called a mobile data access system (MDAS). Within the scope of this infrastructure, two types of services are available to the user: on demand-based services and broadcast-based services (3).

Broadcast-Based Services

Many applications are directed toward public information (i.e., news, weather information, traffic information, and flight information) that are characterized by (*1*) the massive number of users, (*2*) the similarity and simplicity in the requests solicited by the users, and (*3*) the fact that data are modified by a few. The reduced bandwidth attributed to the wireless environment places limitations on the rate and amount of communication. Broadcasting is a potential solution to this limitation. In broadcasting, information is generated and broadcast to all users on the air channels. Mobile users are capable of searching the air channels and pulling the information they want. The main advantage of broadcasting is that it scales up as the number of users increases. In addition, the broadcast channel can be considered as an additional storage available over the air for the mobile clients. Finally, it is shown that pulling information from the air channel consumes less power than pushing information to the air channel. Broadcasting is an attractive solution, because of the limited storage, processing capability, and power sources of the mobile unit. Further discussion about the broadcasting is beyond the scope of this article, and the interested reader is referred to ref. (3).

On-Demand-Based Services

Private data (personal schedules, phone numbers, etc.) and shared data (i.e., a group data, replicated, data or fragmented data of a database) are the subject of these services in which users obtain answers to requests through a two-way communication with the database server; the user request is pushed to the system, data sources are accessed, query operations are performed, partial results are collected and integrated, and generated information is communicated back to the user. This requires a suitable solution that addresses issues such as security and access control, isolation, semantic heterogeneity, autonomy, query processing and query optimization, transaction processing and concurrency control, data integration, browsing, distribution and location transparency, and limited resources (3).

Among these issues, this article concentrates on transaction processing and concurrency control. Traditionally, in a distributed environment, to achieve high performance and throughput, transactions are interleaved and executed concurrently. Concurrent execution of transactions should be coordinated such that there is no interference among them. In an MDAS environment, the concurrent execution of transactions is a more difficult task to control than in distributed database systems, due to the conflicts among global transactions, conflicts among global and local transactions, local autonomy of each site, and frequent network disconnections. Furthermore, some form of data replication should be used at the mobile unit to provide additional availability in case of a weak connection or disconnection.

Researchers have extensively studied caching and replication schemes that may be used to address the constraints of a wireless link. Current distributed database replication and caching schemes are not suitable for an MDAS environment because consistency cannot be effectively maintained due to local autonomy requirements and communication constraints. In addition, because of the autonomy requirements of the local DBMS, the local information about the validity of a page or file is not available globally. Accordingly, any type of invalidation, polling, or timestamp-based method would be too impractical and inefficient to use and, in many cases, impossible.

The use of a hierarchical concurrency control algorithm reduces the required communication overhead in an MDAS and offers higher overall throughput and faster response times. The concurrency control for global transactions is performed at the global level in a hierarchical, distributed manner. The application of the hierarchical structure to enforce concurrency control offers higher performance and reliability.

The limited bandwidth and local autonomy restrictions of an MDAS can be addressed by using an *automated queued queriys* (AQ^2) and caching of data in the form of a *bundled query* (BUNQ). An AQ^2 is a form of prefetching that preloads data onto the mobile unit. A bundled query is an object that consists of a query and its associated data. Read-only queries are cached as a bundled query and are

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

validated using a simple parity-checking scheme. Guaranteeing write consistency while disconnected is extremely difficult or impossible due to local autonomy requirements in an MDAS. Consequently, any transactions containing write operations are directly submitted to the MDAS system or are queued during disconnection and submitted during reconnection. The caching and prefetching policies reduce the access latency and provide timely access to data, notwithstanding the limited bandwidth restrictions imposed by an MDAS environment.

BACKGROUND

Accessing a large amount of data over a limited capability network connection involves two general aspects: the mobile networking environment and mobility issues. The mobile environment includes the physical network architecture and access devices. Mobility issues include adaptability to a mobile environment, autonomy, and heterogeneity. A mobile application must be able to adapt to changing conditions including the network environment and resources available to the application. A resource-scarce mobile system is better served by relying on a server. However, frequent network disconnections, limited network bandwidth, and power restrictions argue for some degree of autonomy.

Multi-databases

An (MDBS) provides a logical integrated view and method to access multiple preexisting local database systems, providing the hardware and software transparencies and maintaining the local autonomy. In addition to autonomy, heterogeneity is also an important aspect of a multi-database system. Support for heterogeneity is a tradeoff between developing and making changes in both hardware and software and limiting participation (1). Consequently, as the number of systems and the degree of heterogeneity among these systems increases, the cost of integration into the global MDBS increases. Access to the local DBMS through a much more diverse and restrictive communication and access device is the natural extension to a traditional MDBS environment, i.e., an MDAS environment (5).

The Summary Schemas Model for Multidatabase Systems. The summary schemas model (SSM) has been proposed as an efficient means to access data in a heterogeneous multi-database environment (4). The identification of the terms that are semantically similar is one key concept in SSM. SSM uses the taxonomy of the English language that contains at least hypernym/hyponym and synonym links among terms to build a hierarchical meta-data. This hierarchical meta-structure provides an incrementally concise view of the data in the form of summary schemas. The SSM hierarchy consists of leaf-node schemas and summary schemas. A leaf-node schema represents an actual database, whereas a summary schema gives an abstract view of the information available at the schemas of its children. The hypernyms of terms in the children of a particular SSM node form the summary schema of that node. As hypernyms are more general or abstract than their hyponyms, many terms could map into a common hypernym. This reduces the overall memory requirements of the SSM meta-data as compared with the global schema approach. The semantic distance metric between the hypernym and their respective hyponyms are relatively small; hence, even though each summary schema does not contain exact information of its children but only an abstracted version, it preserves the semantic contents of its children. The ability to browse/view the global data and to perform an imprecise query, and the small size of the meta-data of the SSM provide several benefits to the traditional multi-database systems that can be directly applied to an MDAS.

SSM can also play an interesting role in the arena of the semantic web. As Tim Berners-Lee pointed out in his 1998 draft “Semantic Web Roadmap” (5), the rationale behind the semantic web is to express information in a machine-understandable form. To this end, a set of standards and tools of the eXtensible Markup Language (XML) (6) is used to create structured web pages: the XML Schema (6), the Resource Description Framework (RDF) (7), the RDF Schema (7), and the Web Ontology Language (OWL) (8). SSM provides a semi-automated solution for millions of existing web pages, which are only human-understandable, to enter the semantic web world. In this application, existing web pages act as local databases and SSM as the portal for information exchange between traditional web pages and the semantic web space.

The MDAS Environment. Overall, the main differentiating feature between an MDAS and an MDBS is the connection of servers and/or clients through a wireless environment and the devices used to access the data. However, both environments are intended to provide timely and reliable access to the globally shared data. Due to the similarities in the objectives of effectively accessing data in a multi-database and a wireless-mobile computing environment, a wireless-mobile computing environment can be easily superimposed on an MDBS. The resulting system is called an MDAS. By superimposing an MDBS onto a mobile computing environment, solutions from one environment are easily mapped to another.

Transaction Management and Concurrency Control

Data access in an MDBS is accomplished through *transactions*. Concurrency control involves coordinating the operations of multiple transactions that operate in parallel and access shared data. By interleaving the operations in such a manner, the potential of interference between transactions arises. The concurrent execution of transactions is considered correct when the ACID properties (atomicity, consistency, isolation, and durability) hold for each transaction (9). The autonomy requirement of local databases in an MDAS introduces additional complexities in maintaining serializable histories because the local transactions are not visible at the global level. Consequently, the operations in a transaction can be subjected to large delays, frequent or unnecessary aborts, inconsistency, and deadlock. Two types of conflicts may arise due to the concurrent execution of transactions—*direct* and *indirect* conflicts (10).

Definition 1. A direct conflict between two transactions T_a and T_b exists if and only if an operation of T_a on data item x [(denoted $o(T_a(x))$)] is followed by $o(T_b(x))$, where T_a does not commit or abort before $o(T_b(x))$, and either $o(T_a(x))$ or $o(T_b(x))$ is a write operation.

Definition 2. An indirect conflict between two transactions T_a and T_b exists if, and only if, a sequence of transactions T_1, T_2, ... T_n exists such that T_a is in direct conflict with T_1, T_1 is in direct conflict with T_2, ..., and T_n is in direct with T_b.

An MDAS should maintain a globally serializable history for correct execution of concurrent transactions, which means that the global history should be conflict free while preserving as much local autonomy as possible. Although the MDBS is responsible for producing a globally serializable history, it is assumed that the local concurrency control system will produce a locally serializable history as well. It is important to note that the MDBS needs to address both direct and indirect conflicts between global transactions. For more information about the concurrency control algorithms that have been advanced in the literature, the reader is referred to ref. 11.

Data Replication for Weak Connections and Disconnection. The communication limitations of an MDAS environment may require that a portion of the data in some form be made readily available to the mobile unit. A large amount of related work has been reported in the area of distributed file systems, distributed replication, and distributed/web caches. The local autonomy requirement of an MDAS environment does not lend itself to the direct application of these works. Data consistency and access time are two main objectives in maintaining replicated data in a mobile or distributed environment.

Replication schemes differ slightly from caching schemes in that the replicated data are accessible by other systems outside of the system on which the data resides. Two types of general replication schemes exist: primary/secondary copy (PSC) replication and voting-based replication schemes. PSC replication does not work in an MDAS environment because write operations to replicated data do not reach the primary copy when disconnected. Therefore, write consistency is not guaranteed. In a distributed system, data consistency is guaranteed with voting-based replications; however, they tend to be more expensive and require much more communication. In an MDAS environment, the local replica, while disconnected, cannot participate in the decision/voting process, and any changes to the local data may result in an inconsistency.

Caching is an effective means of data duplication that is used to reduce the latency of read and write operations on data. Early research on the disconnected operation was done with file system-based projects. When a disconnection occurs, a cache manager services all file system requests from the cache contents. As with replication, the invalidation of data is not possible when disconnected, and thus, consistency cannot be guaranteed. Web-based caching (12) offers the two most common forms of cache consistency mechanisms, i.e., time-to-live (TTL) and client polling (13). The disconnected operation of a mobile system and the local autonomy requirements of the local systems make the application of these schemes impractical in an MDAS environment. Some schemes try to use the concept of compensating transactions (or operations) to keep replicated data consistent. However, compensating transactions are difficult to implement, and in some cases, a compensating transaction cannot semantically undo the transaction (or operation) (10).

CONCURRENCY CONTROL AND DATA REPLICATION FOR MDAS

The proposed concurrency control algorithm is defined in an environment where information sources are stationary and user requests are aimed at shared data sources. Under these conditions, the v-locking algorithm is intended to reduce the amount of communication and hence to reduce the effect of frequent disconnections in wireless environment.

The V-Locking Concurrency Control Scheme

The proposed v-locking algorithm uses a global locking scheme (GLS) to serialize conflicting operations of global transactions. Global locking tables are used to lock data items involved in a global transaction in accordance to the two-phase locking (2PL) rules. In typical multi-database systems, maintaining a global locking table would require communication of information from the local site to the global transaction manager (GTM). In an MDAS environment, this is impractical due to the delay, amount of communication overhead, and frequent disconnections involved. In our underlying infrastructure, software is distributed in a hierarchical structure similar to the hierarchical structure of the SSM. Subsequently, transaction management is performed at the global level in a hierarchical, distributed manner. A global transaction is submitted at any node in the hierarchy—either at a local node or at a summary schema node. The transaction is resolved and mapped into subtransactions, and its global transaction coordinator is determined by the SSM structure (4).

The v-locking algorithm is based on the following assumptions:

1. There is no distinction between local and global transactions at the local level.
2. A local site is completely isolated from other local sites.
3. Each local system ensures local serializability and freedom from local deadlocks.
4. A local database may abort any transaction at any time within the constraints of a distributed atomic commit protocol.
5. Information pertaining to the type of concurrency control used at the local site will be available. For systems to provide robust concurrency and consistency, in most systems, a *strict* history is produced

> through the use of a strict 2PL scheme. Therefore, most local sites will use a strict 2PL scheme for local concurrency control.

Consequently, the MDAS coordinates the execution of global transactions without the knowledge of any control information from local DBMS. The only information (loss of local autonomy) required by the algorithm is the type of concurrency control protocol performed at the local sites. The semantic information contained in the summary schemas is used to maintain global locking tables. As a result, the "data" item being locked is reflected either exactly or as a hypernym term in the summary schema of the transaction's GTM. The locking tables can be used in an aggressive manner where the information is used only to detect potential global deadlocks. A more conservative approach can be used where the operations in a transaction are actually delayed at the GTM until a global lock request is granted. Higher throughput at the expense of lower reliability is the direct consequence of the application of semantic contents rather than exact contents for an aggressive approach. In either case, the global locking table is used to create a global wait-for-graph, which is subsequently used to detect and resolve potential global deadlocks. In the proposed v-locking algorithm, due to the hierarchical nature of the summary schemas model, the global wait-for-graphs are maintained in hierarchical fashion within the summary schemas nodes. In addition, edges in the wait-for-graphs as discussed later are labeled as exact or imprecise.

During the course of operations, the wait-for-graph is constructed based on the available communication. Three cases are considered: (*1*) Each operation in the transaction is individually acknowledged; (*2*) write operations are only acknowledged; and (*3*) only the commit or abort of the transaction is acknowledged. For the first case, based on the semantic contents of the summary schema node, an edge inserted into the wait-for-graph is marked as being an exact or imprecise data item. For each acknowledgment signal received, the corresponding edge in the graph is marked as exact. In the second case, where each write operation generates an acknowledgment signal, for each signal only the edges preceding the last known acknowledgment are marked as being exact. Other edges that have been submitted but that have not been acknowledged are marked as pending. As in the previous two cases, in the third case, the edges are marked as representing exact or imprecise data. However, all edges are marked as pending until the commit or abort signal is received. Keeping the information about the data and status of the acknowledgment signals enables one to detect cycles in the wait-for-graph. The wait-for-graph is checked for cycles after a time threshold for each transaction. For all transactions involved in a cycle, if the exact data items are known and all acknowledgments have been received, then a deadlock is precisely detected and broken. When imprecise data items are present within a cycle, the algorithm will consider the cycle a deadlock only after a longer time threshold has passed. Similarly, a pending acknowledgment of a transaction is only used to break a deadlock in a cycle after an even longer time threshold has passed. The time thresholds can be selected and adjusted dynamically to prevent as many false deadlocks as possible.

Handling Unknown Local Data Sources

The v-locking algorithm has been extended to handle the local "black box" site in which the global level knows nothing about the local concurrency control. As nearly every commercial database system uses some form of 2PL, this case will only comprise a small percentage of local systems. The algorithm merely executes global transactions at such a site in a serial order. This is done by requiring any transaction involving the "black-box" to obtain a site lock before executing any operations in the transaction. These types of locks will be managed by escalating any lock request to these sites to the highest level (site lock).

Data Replication Protocol

Communication Protocol. Maintaining the ACID properties of a transaction with replicated data in an MDAS environment is very difficult. The proposed scheme considers three levels of connectivity in which a mobile unit operates. During a *strong connection*, the mobile unit sends/receives all transactions and returns data directly to/from land-based, fixed sites for processing. When the communication link degrades to a *weak connection*, transactions are queued at the mobile unit and passed through the system according to the availability of bandwidth. Returned data are also queued at the fixed site. The queuing of operations during a weak connection allows a mobile unit to continue processing at the expense of increased latency.

In the disconnected state, a user may perform read-only queries on any cached data. For this case, the consistency of the data is not guaranteed, i.e., the user may receive stale data. Any transaction that contains a write operation is queued and submitted to the system when the connection is reestablished. Naturally, if a read-only access does not find the data locally, the query is queued and submitted later when the connection is established.

Cached Data—Bundled Queries (BUNQ). Data that are cached on the mobile unit consist of a query and its associated data. Several reasons exist for using a BUNQ instead of page-based or file-based data. In a tightly coupled distributed system, it is possible to cache data at the local unit. However, in a multi-database system, the structure of the data at the local databases will vary (*structural differences*—the information may be stored as structured data, unstructured data, web pages, files, objects, etc.). This variation of data at each local site makes it difficult to cache the data in a uniform manner. In addition, the *autonomy requirement* of the local database imposes further restrictions for caching data. It may not be possible to determine the underlying structure of the data at the local site without violating local autonomy requirements. Consequently, instead of caching individual data items from each local source, the data set associated with a particular transaction is cached—a bundled query. By caching a BUNQ, the resolution of the structural differences is done at the

MDAS level, while maintaining the local autonomy requirements. The primary advantage is the ease and simplicity of implementation, which comes at the expense of retaining data in the cache at a very coarse-grained level.

Prefetching and Replacement/Invalidation Policy

Prefetching. The prefetching algorithms can be developed based on user profiles and usage histories. The limited power, storage, processing capability, and bandwidth of a mobile unit make the incorrect prefetch of data extremely expensive. The idea is to prefetch enough data such that the user can still operate during a disconnection (albeit with relaxed consistency requirements) while minimizing the use of additional energy and bandwidth.

Allowing the user to specify a particular read-only transaction as an automated queued query prefetches the data. An AQ^2 has a relaxed requirement for consistency, which is defined by the user. The user sets a valid time threshold for each AQ^2 when defining such a transaction. The mobile unit automatically submits the transaction to the MDAS when the threshold has expired, i.e., prefetches the data. The results are stored as a BUNQ. If the user requests the data in the AQ^2 before the BUNQ is invalidated, the query is serviced from the local cache.

Replacement. The data in the cache consists of both automated queued queries and other user-submitted read-only queries. The data in the cache are replaced based on the least recently used (LRU) policy. The LRU policy has its advantages in that it is well understood and easy to implement. Moreover, other than some web-based caching algorithms, the LRU policy is the most widely used replacement policy in DBMS caches (13).

Invalidation. To maintain consistency between the copies of data residing on the fixed and mobile units, the data in the cache must be correctly invalidated when the main copy changes. A parity-based signature could be used to accomplish this task for each BUNQ in the cache (p-caching). When the user submits a transaction, if a corresponding BUNQ is present in the cache, the transaction (along with the parity code) is sent to the fixed node. The fixed node then performs the query and delays the transmission of the information back to the mobile unit until it generates and compares the two parity codes. If they are identical, only an acknowledgment is sent back to the mobile unit and the data are read locally from the cache. Otherwise, the resultant data, along with its new parity sequence, is returned and replaced in the cache. The old copy of the BUNQ is invalidated according to the LRU rules.

PERFORMANCE EVALUATION

Simulator Design

The performance of the proposed v-locking and p-caching algorithms was evaluated through a simulator written in C^{++} using CSIM. The simulator measures performance in terms of global transaction throughput, response time, and CPU, disk I/O, and network utilization. In addition, the simulator was extended to compare and contrast the behavior of the v-lock algorithm against the site-graph, potential conflict graph, and the forced conflict algorithms. This includes an evaluation with and without the cache. It should be noted that the simulator is designed to be versatile to evaluate the v-locking algorithm based on various system configurations. At each moment in time, a fixed number of active global and local transactions is present in the system. Each operation of the transaction is scheduled and is communicated to the local system based on the available bandwidth. The global scheduler acquires the necessary global virtual locks and processes the operation. The operation(s) then uses the CPU and I/O resources and is communicated to the local system based on the available bandwidth. When acknowledgments or commit/abort signals are received from the local site, the algorithm determines whether the transaction should proceed, commit, or abort. For read-only transactions after a global commit, a parity code is generated for the resultant data and compared with the parity code of the BUNQ. For a matching code, only an acknowldgment signal is sent back to the mobile unit. Otherwise, the data and the new parity code are sent back to the mobile unit. Transactions containing a write operation are placed directly in the ready queue. If a deadlock is detected, or an abort message is received from a local site, the transaction is aborted at all sites and the global transaction is placed in the restart queue. After a specified time elapses, the transaction is again placed on the active queue.

System Parameters

The underlying global information-sharing process is composed of 10 local sites. The size of the local databases at each site can be varied and has a direct effect on the overall performance of the system. The simulation is run for 5000 time units. The global workload consists of randomly generated global queries, spanning over a random number of sites. Each operation of a subtransaction (read, write, commit, or abort) may require data and/or acknowledgments to be sent from the local DBMS. The frequency of messages depends on the quality of the network link. To determine the effectiveness of the proposed algorithm, several parameters are varied for different simulation runs.

A collection of mobile units submits global queries (selected from a pool of 500 queries) to the MDAS. The connection between the mobile unit and the MDAS has a 50% probability of having a strong connection and a 30% probability of having a weak connection. There is a 20% probability of being disconnected. A strong connection has a communication service time of 0.1 to 0.3 seconds, whereas a weak connection has a service time range of 0.3 to 3 seconds. When a disconnection occurs, the mobile unit is disconnected for 30 to 120 seconds. Initially, one third of the pool consists of read-only queries, which are locally cached as a BUNQ. Additionally, 10% of the read-only queries are designated as an AQ^2. For read-only queries, the local cache is first checked for an existing BUNQ. If present, the query is submitted along with the associated parity sequence. The MDAS returns either the

Table 1. Simulation Parameters

Parameters	Default Value
Global System Parameters	
The number of local sites in the system.	10
The number of data items per local site.	100
The maximum number of global transactions in the system. This number represents the global multi-programming level.	10
The maximum number of operations that a global transaction contains.	8
The minimum number of operations that a global transaction contains.	1
The service time for the CPU queue.	0.005 s
The service time for the IO queue.	0.010 s
The service time for each communicated message to the local site.	0.100 s
The number of messages per operation (read/write).	2
Mobile Unit Parameters	
The maximum number of mobile units in the system. This number represents the global multi-programming level.	10
The maximum number of operations that a global transaction contains.	8
The minimum number of operations that a global transaction contains.	1
The service time for the CPU queue.	0.005 s
The service time for the IO queue.	0.010 s
The service time for each communicated message to the global system for a strong connection, randomly selected.	0.100–0.300 s
The service time for each communicated message to the global system for a weak connection, randomly selected.	0.300–3 s
The service time for a disconnection, randomly selected	30–120 s
The probability of a strong connection.	0.50
The probability of a weak connection.	0.30
The probability of a disconnection.	0.20
The number of messages per operation (read/write).	2
The average size of each message.	1,024 bytes
The size of the mobile unit cache.	1 Megabyte
The probability that a query is read-only.	1/3
The probability that a read-only query is submitted as an AQ^2.	0.1
Local System Parameters	
The maximum number of local transactions per site. This number represents the local multi-programming level.	10
The maximum number of write operations per transaction.	8
The maximum number of read operations per transaction.	8
The minimum number of write operations per transaction.	1
The minimum number of read operations per transaction.	1
The service time for the local CPU queue.	0.005
The service time for the local IO queue.	0.010
The service time for each communicated message to the MDAS.	0.100
The number of messages per operation (read/write).	2

data or an acknowledgment signal for a matching BUNQ. Subsequently, if signatures do not match, the mobile unit updates the cache with the new data according to the LRU scheme. Upon termination of a transaction, a new query is selected and submitted to the MDAS.

The local systems perform two types of transactions—local and global. Global subtransactions are submitted to the local DBMS and appear as local transactions. Local transactions generated at the local sites consist of a random number of read/write operations. The only difference between the two transactions is that a global subtransaction will communicate with the global system, whereas the local transaction terminates upon a commit or abort. The local system may abort a transaction, global or local, at any time. If a global subtransaction is aborted locally, it is communicated to the global system and the global transaction is aborted at all sites. Table 1 summarizes all parameters used in the simulation. It should be noted that Table 1 shows the default values; however, the simulator is flexible enough to simulate the behavior of the v-locking algorithm for various system parameters.

Simulation Results and Analysis

The v-locking algorithm has been simulated and compared against some other concurrency control algorithms, i.e., potential conflict graph (PCG), forced-conflict, and site-graph algorithms with and without the proposed p-caching scheme. Figures 1 and 2 show the results. As expected, the v-locking algorithm offers the highest throughput. This result is consistent with the fact that the v-locking algorithm is better able to detect global conflicts and thus

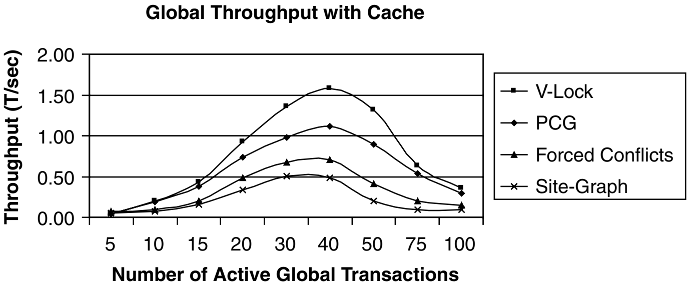


Figure 1. Global throughput with P-Caching.

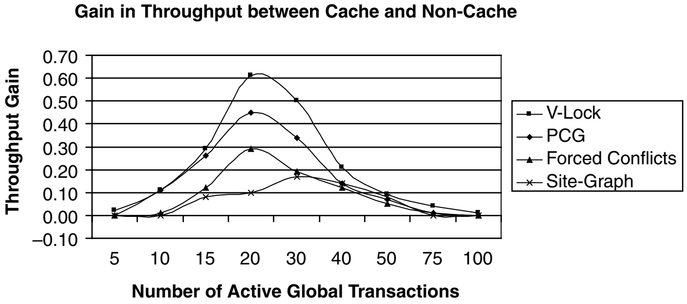


Figure 3. Comparison of the sensitivity of the P-Caching algorithm.

achieves higher concurrency than the other algorithms. As can be abserved, the maximum throughput occurs at a multi-programming level approximately equal to 40. As the number of concurrent global transactions increases, the number of completed global transactions decreases due to the increase in the number of conflicts. The peak throughput for each concurrency control scheme is slightly higher with the use of caching. Furthermore, for all simulated algorithms, the caching scheme allows a higher number of active global transactions and, hence, higher throughput. This characteristic is attributed to the cache hits on the read-only data. For the non-cache case, the throughput is low until the active number of global transactions reaches about 30, with a rapid increase of the throughput from 30 to 40 active transactions. This occurs because of the weak connections and disconnections. With the p-caching algorithm, the rate of the increase in throughput is more gradual because the local cache can service the read-only queries under weak connectivity or disconnection.

The gain in the throughput, sensitivity, of the p-caching algorithm for different concurrency control schemes, is shown in Fig. 3. The v-locking scheme shows the greatest sensitivity to the caching algorithm. At 20 active global transactions, there is an improvement in the throughput of approximately 0.6 when using a cache. At the peak throughput of 40 simultaneous transactions, the throughput is increased by 0.2. The PCG, site-graph, and forced conflict algorithms show similar characteristics to the v-locking algorithm; however, the sensitivity of these algorithms is less. The caching becomes ineffective for all schemes when the number of active global transactions is greater than 75.

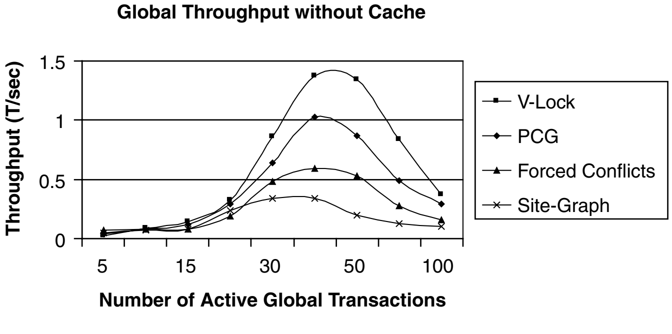


Figure 2. Global throughput without P-Caching.

The simulator also measured the percentage of completed transactions for the various concurrency control algorithms. In general, for all schemes, the number of completed transactions decreased as the number of concurrent transactions increased, due to more conflicts among the transactions. However, the performance of both the forced conflict and the site-graph algorithms decreased at a faster rate, which is due to the increase in the number of false aborts detected by these algorithms. The v-locking algorithm more accurately detects deadlocks by differentiating between global and indirect conflicts and, therefore, performs better than the PCG algorithm.

The simulator also measured the communication utilization. It was found that the communication utilization decreases with the use of the cache. At 20 active global transactions, the v-locking algorithm uses 40% of the communication bandwidth versus 69% utilization without the cache. Similarly, with 30 active global transactions, there is a 75% versus 91% communication utilization with and without the cache, respectively. This result is attributed to the reduction in transferred data from parity acknowledgments and local accesses to the cache. At peak throughput, both locking algorithms (v-locking and PCG) were using nearly 100% of the communication channel. The communication utilization was about 100% at peak throughput and decreased slightly as the number of concurrent transactions increased. It is easy to determine from this result that the communication requirements for the v-locking algorithm represent the bottleneck of the system for both the caching and the non-caching case.

FUTURE RESEARCH DIRECTIONS

The requirements of an "anytime, anywhere" computing environment motivate new concepts that effectively allow a user to access information in a timely and reliable manner. In such an environment, a potentially large number of users may simultaneously access a rapidly increasing amount of aggregate, distributed data. This usage motivates the need for a proper concurrency control algorithm that offers higher throughput in the face of the limitations imposed by technology. A distributed, hierarchically organized concurrency control algorithm was presented and evaluated that satisfies these limitations. The semantic information contained within the SSM was used to maintain the global locking tables to serialize conflicting operations of global transactions and to detect and break deadlocked transactions. Data duplication in the form of replication and caches was also used to lessen the effects of weak communications or disconnection. The duplicated data at the mobile node allow the user to continue to work in case of a weak connection or disconnection. Automated queued queries and bundled queries could be used to address the limited bandwidth and local autonomy restrictions in an MDAS environment.

The work presented in this article can be extended in several directions:

- *Application of mobile agents*: Recently the literature has shows a growing interest in the application and incorporation of mobile agent paradigm in an information retrieval system (14). When mobile agents are introduced into the system, mobile agents can roam the network and fulfill their tasks without the owner's intervention; consequently, mobile users only need to maintain the communication connection during the agent submission and retraction. In addition, by migrating from the mobile device to the core network, the agents can take full advantage of the high bandwidth of the wired portion of the network and the high computation capability of servers/workstations. Moreover, mobile agents' migration capability allows them to handle tasks locally instead of passing messages between the involved data sources and, hence, reduce the number of messages that are needed in accomplishing a task. To summarize, the use of mobile agents relaxes requirements on mobile users' critical resources such as connectivity, bandwidth, and energy. Therefore, within the scope of the MDAS, it would be of interest to investigate concurrency control algorithms at the presence of the mobile agents in the system.
- *Multimedia databases*: The demand of image data management has raised the research on content-based retrieval models (15, 16). In contrast to the traditional text-based systems, these applications usually consist of a large volume of image data whose semantic contents cannot be represented efficiently using the traditional database models. Most of the present content-based image retrieval approaches employ the feature vectors to facilitate content-based query processing—the features are extracted from image pixels, heuristically or empirically, and combined into vectors according to the application criterion. However, these low-level features cannot represent the semantic contents and therefore do not provide an ideal basis for semantic-based image retrieval. To introduce novel schemes to facilitate semantic-based image content query/transaction management in a distributed heterogeneous database environment, i.e., MDAS, will be of great interest to both academia and industry.
- *Quality-of-Service Cache Coherence*: The p-caching strategy showed promising results in improving the performance of the read-only query and hence overall system's throughput. The effect of the cache on the MDAS environment should be studied. This could be done by changing the cache-hit ratio to a raw probability that a read-only query is valid or invalid. A quality-of-service QOS approach is ideally suited for such a general-purpose cache coherence protocol, providing strong consistency for those data items that require it while permitting weaker consistency for less critical data (12). Therefore, it would be interesting to investigate the effectiveness of such a QOS approach on the performance metrics that determine the effectiveness of the concurrency control policies.

BIBLIOGRAPHY

1. A. R. Hurson and M. W. Bright, Multidatabase systems: An advanced concept in handling distributed data, *Adv. Comput.*, **32**: 149–200, 1991.
2. J. B. Lim and A. R. Hurson, Heterogeneous data access in a mobile environment—issues and solutions, *Adv. Comput.*, **48**: 119–178, 1999.
3. A. R. Hurson and Y. Jiao, Data broadcasting in a mobile environment, in D. Katsaros et al. (eds.), *Wireless Information Highway.*, Hershey, PA: IRM Press, 2004, pp. 96–154.
4. M. W. Bright, A. R. Hurson, and S. H. Pakzad, Automated resolution of semantic heterogeneity in multidatabases, *ACM Trans. Database Syst.*, **19**(2): 212–253, 1994.
5. T. Berners-Lee, Semantic Web Roadmap. Available: http://www.w3.org/DesignIssues/Semantic.html.
6. W3C(a), Extensible Markup Language. Available: http://www.w3.org/XML/.
7. W3C(b), Resource Description Framework. Available: http://www.w3.org/RDF/.
8. W3C(c), OWL Web Ontology Language Overview. Available: http://www.w3.org/TR/owl-features/.
9. J. Lim and A. R. Hurson, Transaction processing in mobile, heterogeneous database systems, *IEEE Trans. Knowledge Data Eng.*, **14**(6): 1330–1346, 2002.
10. Y. Breitbart, H. Garcia–Molina, and A. Silberschatz, Overview of multidatabase transaction management, *VLDB J.*, **1**(2): 181–239, 1992.
11. K. Segun, A. R. Hurson, V. Desai, A. Spink, and L. L. Miller, Transaction management in a mobile data access system, *Ann. Rev. Scalable Comput.*, **3**: 85–147, 2001.
12. J. Sustersic and A. R. Hurson, Coherence protocols for bus-based and scalable multiprocessors, Internet, and wireless distributed computing environment: A survey, *Adv. Comput.*, **59**: 211–278, 2003.
13. P. Cao and C. Liu, Maintaining strong cache consistency in the World Wide Web, *IEEE Trans. Comput.*, **47**(4): 445–457, 1998.
14. Y. Jiao and A. R. Hurson, Application of mobile agents in mobile data access systems—a prototype, *J. Database Manage.*, **15**(4): 1–24, 2004.
15. B. Yang and A. R. Hurson, An extendible semantic-based content representation method for distributed image databases, *Proc. International Symposium on Multimedia Software Engineering*, 2004, pp. 222–226.
16. B. Yang, A. R. Hurson, and Y. Jiao, On the Content Predictability of Cooperative Image Caching in Ad Hoc Networks,

Proc. *International Conference on Mobile Data Management*, 2006.

FURTHER READING

A. Elmagarmid, J. Jing, and T. Furukawa, Wireless client/server computing for personal information services and applications, *ACM Sigmod Record*, **24**(4): 16–21, 1995.

M. Franklin, M. Carey, and M. Livny, Transactional client-server cache consistency: Alternatives and performance, *ACM Trans. Database Syst.*, **22**(3): 315–363, 1995.

S. Mehrotra, H. Korth, and A. Silberschatz, Concurrency control in hierarchical multi-database systems, *VLDB J.*, **6**: 152–172, 1997.

E. Pitoura and B. Bhargava, A framework for providing consistent and recoverable agent-based access to heterogeneous mobile databases, *ACM Sigmod Record*, **24**(3): 44–49, 1995.

A. Brayner and F. S. Alencar, A semantic-serializability based fully-distributed concurrency control mechanism for mobile multidatabase systems, proc. *International Workshop on Database and Expert Systems Applications*, 2005.

R. A. Dirckze and L. Gruenwald, A pre-serialization transaction management technique for mobile multi-databases, *Mobile Networks Appl.*, **5**: 311–321, 2000.

J. B. Lim
MJL Technology
Seoul, South Korea
A. R. Hurson
Y. Jiao
The Pennsylvania State
University
State College, Pennsylvania

VERY LARGE DATABASES

A growing number of database applications require online, interactive access to very large volumes of data to perform a variety of data analysis tasks. As an example, large telecommunication and Internet service providers typically collect and store Gigabytes or Terabytes of detailed usage information (Call Detail Records, SNMP/RMON packet flow data, etc.) from the underlying network to satisfy the requirements of various network management tasks, including billing, fraud/anomaly detection, and strategic planning. Such large datasets are typically represented either as massive *alphanumeric data tables* (in the relational data model) or as massive *labeled data graphs* (in richer, semistructured data models, such as extensible markup language (XML)). In order to deal with the huge data volumes, high query complexities, and interactive response time requirements characterizing these modern data analysis applications, the idea of *effective, easy-to-compute approximations over precomputed, compact data synopses* has recently emerged as a viable solution. Due to the exploratory nature of most target applications, there are a number of scenarios in which a (reasonably accurate) fast approximate answer over a small-footprint summary of the database is actually preferable over an exact answer that takes hours or days to compute. For example, during a drill-down query sequence in ad hoc data mining, initial queries in the sequence frequently have the sole purpose of determining the truly interesting queries and regions of the database. Providing fast approximate answers to these initial queries gives users the ability to focus their explorations quickly and effectively, without consuming inordinate amounts of valuable system resources. The key, of course, behind such approximate techniques for dealing with massive datasets lies in the use of appropriate *data reduction techniques* for constructing compact *data synopses* that can accurately approximate the important features of the underlying data distribution. In this article, we provide an overview of date reduction and approximation methods for massive databases and discuss some of the issues that develop from different types of data, large data volumes, and applications-specific requirements.

APPROXIMATION TECHNIQUES FOR MASSIVE RELATIONAL DATABASES

Consider a relational table R with d data attributes X_1, $X_2, \dots X_d$. We can represent the information in R as a d-dimensional array A_R , whose *jth* dimension is indexed by the values of attribute X_j and whose cells contain the count of tuples in R having the corresponding combination of attribute values. A_R is essentially the *joint frequency distribution* of all the data attributes of R. More formally, let $D = \{D_1, D_2, \dots, D_d\}$ denote the set of dimensions of A_R, where dimension D_j corresponds to the *value domain* of attribute X_j . Without loss of generality, we assume that each dimension D_j is indexed by the set of integers $\{0, 1, \dots, |D_j| - 1\}$, where $|D_j|$ denotes the size of dimension D_j. We assume that the attributes $\{X_1, \dots, X_d\}$ are ordinal in nature, that is, their domains are naturally ordered, which captures all numeric attributes (e.g., age, income) and some categorical attributes (e.g., education). Such domains can always be mapped to the set of integers mentioned above while preserving the natural domain order and, hence, the locality of the distribution. It is also possible to map unordered domains to integer values; however, such mappings do not always preserve locality. For example, mapping countries to integers using alphabetic ordering can destroy data locality. There may be alternate mappings that are more locality preserving, (e.g., assigning neighboring integers to neighboring countries). (Effective mapping techniques for unordered attributes are an open research issue that lies beyond the scope of this article.) The d-dimensional joint-frequency array A_R comprises $N = \Pi_{i=1}^{d} |D_i|$ cells with cell $A_R[i_1, i_2, \dots, i_d]$ containing the count of tuples in R having $X_j = i_j$ for each attribute $1 \le j \le d$.

The common goal of all relational data reduction techniques is to produce compact synopsis data structures that can effectively approximate the d-dimensional joint-frequency distribution A_R. In what follows, we give an overview of a few key techniques for relational data reduction, and discuss some of their main strengths and weaknesses as well as recent developments in this area of database research. More exhaustive and detailed surveys can be found elsewhere; see, for example, Refs. 1 and 2.

Sampling-Based Techniques

Sampling methods are based on the notion that a large dataset can be represented by a small *uniform random sample* of data elements, an idea that dates back to the end of the nineteenth century. In recent years, there has been increasing interest in the application of sampling ideas as a tool for data reduction and approximation in relational database management systems (3–8). Sample synopses can be either precomputed and incrementally maintained (e.g., Refs. 4 and 9) or they can be obtained progressively at run-time by accessing the base data using specialized data access methods (e.g., Refs. 10 and 11). Appropriate *estimator functions* can be applied over a random sample of a data collection to provide approximate estimates for quantitative characteristics of the entire collection (12). The adequacy of sampling as a data-reduction mechanism depends crucially on how the sample is to be used. Random samples can typically provide accurate estimates for aggregate quantities (e.g., COUNTs or AVERAGEs) of a (sub)population (perhaps determined by some selection predicate), as witnessed by the long history of successful applications of random sampling in population surveys

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

(12,13). An additional benefit of random samples is that they can provide probabilistic guarantees (i.e., *confidence intervals*) on the quality of the approximation (7,11). On the other hand, as a query approximation tool, random sampling is limited in its query processing scope, especially when it comes to the "workhorse" operator for correlating data collections in relational database systems, the *relational join*. The key problem here is that a join operator applied on two uniform random samples results in a *nonuniform* sample of the join result that typically contains *very few tuples*, even when the join selectivity is fairly high (9). Furthermore, for *nonaggregate* queries, execution over random samples of the data is guaranteed to always produce a small subset of the exact answer, which is often *empty* when joins are involved (9,14). The recently proposed "join synopses" method (9) provides a (limited) sampling-based solution for handling *foreign-key joins that are known beforehand* (based on an underlying "star" or "snowflake" database schema). Techniques for appropriately biasing the base-relation samples for effective approximate join processing have also been studied recently (3).

Histogram-Based Techniques

Histogram synopses or approximating *one-dimensional* data distributions have been extensively studied in the research literature (15–19), and have been adopted by several commercial database systems. Briefly, a histogram on an attribute X is constructed by employing a *partitioning rule* to partition the data distribution of X into a number of mutually disjoint subsets (called *buckets*), and approximating the frequencies and values in each bucket in some common fashion. Several partitioning rules have been proposed for the bucketization of data distribution points—some of the most effective rules seem to be ones that explicitly try to minimize the overall variance of the approximation in the histogram buckets (17–19). The summary information stored in each bucket typically comprises (1) *the number of distinct data values* in the bucket, and (2) *the average frequency* of values in the bucket, which are used to approximate the actual bucket contents based on appropriate *uniformity assumptions* about the spread of different values in the bucket and their corresponding frequencies (19).

One-dimensional histograms can also be used to approximate a (multidimensional) joint-frequency distribution A_R through a *mutual-independence assumption* for the data attributes $\{X_1, \ldots, X_d\}$. Mutual independence essentially implies that the joint-frequency distribution can be obtained as a product of the one-dimensional marginal distributions of the individual attributes X_i. Unfortunately, experience with real-life datasets offers overwhelming evidence that this independence assumption is almost always invalid and can lead to gross approximation errors in practice (20,21). Rather than relying on heuristic independence assumptions, *multidimensional histograms* [originally introduced by Muralikrishna and DeWitt (22)] try to directly approximate the joint distribution of $\{X_1,\ldots, X_d\}$ by strategically partitioning the data space into d-dimensional buckets in a way that captures the variation in data frequencies and values. Similar to the one-dimensional case, uniformity assumptions are made to approximate the distribution of frequencies and values within each bucket (21). Finding optimal histogram bucketizations is a hard optimization problem that is typically *NP*-complete even for two dimensions (23). Various greedy heuristics for multidimensional histogram construction have been proposed (21,22,24) and shown to perform reasonably well for low to medium data dimensionalities (e.g., $d = 2$–5).

Recent work has demonstrated the benefits of histogram synopses (compared with random samples) as a tool for providing fast, approximate answers to both *aggregate* and *nonaggregate* (i.e., "set-valued") user queries over low-dimensional data (14). Other studies have also considered the problem of *incrementally* maintaining a histogram synopsis over updates (25,26) or using query feedback (27,28), and the effectiveness of random sampling for approximate histogram construction (29). Unfortunately, like most techniques that rely on space partitioning (including the wavelet-based techniques of the next section), multidimensional histograms also fall victim to the "*curse of dimensionality*," which renders them ineffective above 5–6 dimensions (24).

Wavelet-Based Techniques

Wavelets are a mathematical tool for the hierarchical decomposition of functions with several successful applications in signal and image processing (30,31). Broadly speaking, the wavelet decomposition of a function consists of a coarse overall approximation along with detail coefficients that influence the function at various scales (31). A number of recent studies have also demonstrated the effectiveness of the wavelet decomposition (and Haar wavelets, in particular) as a data reduction tool for database problems, including selectivity estimation (32) and approximate query processing over massive relational tables (33–35).

Suppose we are given the one-dimensional data frequency vector A containing the $N = 8$ values $A = [2, 2, 0, 2, 3, 5, 4, 4]$. The Haar wavelet decomposition of A can be computed as follows. We first average the values together pairwise to get a new "lower-resolution" representation of the data with the following average values [2,1,4,4]. In other words, the average of the first two values (that is, 2 and 2) is 2, that of the next two values (that is, 0 and 2) is 1, and so on. Obviously, some information has been lost in this averaging process. To be able to restore the original values of the frequency array, we need to store some *detail coefficients* that capture the missing information. In Haar wavelets, these detail coefficients are simply the differences of the (second of the) averaged values from the computed pairwise average. Thus, in our simple example, for the first pair of averaged values, the detail coefficient is 0 because $2 - 2 = 0$; for the second sample, we again need to store -1 because $1 - 2 = -1$. Note that no information has been lost in this process—it is fairly simple to reconstruct the eight values of the original data frequency array from the lower-resolution array containing the four averages and the four detail coefficients. Recursively applying the above pairwise averaging and differencing process on the lower-resolution array containing the averages, we get the following full decomposition:

Resolution	Averages	Detail Coefficients
3	[2, 2, 0, 2, 3, 5, 4, 4]	—
2	[2, 1, 4, 4]	[0, −1, −1, 0]
1	[3/2, 4]	[1/2, 0]
0	[11/4]	[−5/4]

The *Haar wavelet decomposition* of A is the single coefficient representing the overall average of the frequency values followed by the detail coefficients in the order of increasing resolution. Thus, the one-dimensional Haar wavelet transform of A is given by $W_A = [11/4, -5/4, 1/2, 0, 0, -1, -1, 0]$. Each entry in W_A is called a *wavelet coefficient*. The main advantage of using W_A instead of the original frequency vector A is that for vectors containing similar values most of the detail coefficients tend to have very small values. Thus, eliminating such small coefficients from the wavelet transform (i.e., treating them as zeros) introduces only small errors when reconstructing the original data, resulting in a very effective form of lossy data compression (31). Furthermore, the Haar wavelet decomposition can also be extended to *multidimensional* joint-frequency distribution arrays through natural generalizations of the one-dimensional decomposition process described above (33,35). Thus, the key idea is to apply the decomposition process over an input dataset along with a thresholding procedure in order to obtain a compact data synopsis comprising of a selected small set of *Haar wavelet coefficients*. The results of several research studies (32–37) have demonstrated that fast and accurate approximate query processing engines (for both aggregate and nonaggregate queries) can be designed to operate solely over such compact *wavelet synopses*.

Other recent work has proposed probabilistic counting techniques for the efficient online maintenance of wavelet synopses in the presence of updates (38), as well as time- and space-efficient techniques for constructing wavelet synopses for datasets with multiple measures (such as those typically found in OLAP applications) (39). All the above-mentioned studies rely on conventional schemes for eliminating small wavelet coefficients in an effort to minimize the overall sum-squared error (SSE). Garofalakis and Gibbons (34,36) have shown that such conventional wavelet synopses can suffer from several important problems, including the introduction of severe bias in the data reconstruction and wide variance in the quality of the data approximation, as well as the lack of nontrivial guarantees for individual approximate answers. In contrast, their proposed *probabilistic wavelet synopses* rely on a probabilistic thresholding process based on *randomized rounding* that tries to *probabilistically* control the maximum relative error in the synopsis by minimizing appropriate probabilistic metrics.

In more recent work, Garofalakis and Kumar (40) show that the pitfalls of randomization can be avoided by introducing efficient schemes for *deterministic* wavelet thresholding with the objective of optimizing a *general class of error metrics* (e.g., maximum or mean relative error). Their optimal and approximate thresholding algorithms are based on novel Dynamic-Programming (DP) techniques that take advantage of the *coefficient-tree structure* of the Haar decomposition. This turns out to be a fairly powerful idea for wavelet synopsis construction that can handle a broad, natural class of *distributive error metrics* (which includes several useful error measures for approximate query answers, such as maximum or mean weighted relative error and weighted L_p-norm error) (40). The above wavelet thresholding algorithms for non-SSE error metrics consider only the *restricted* version of the problem, where the algorithm is forced to select values for the synopsis from the standard Haar coefficient values. As observed by Guha and Harb (41), such a restriction makes little sense when optimizing for non-SSE error, and can, in fact, lead to suboptimal synopses. Their work considers *unrestricted* Haar wavelets, where the values retained in the synopsis are specifically chosen to optimize a general (weighted) L_p-norm error metric. Their proposed thresholding schemes rely on a DP over the coefficient tree (similar to that in (40) that *also iterates over the range of possible values for each coefficient*. To keep time and space complexities manageable, techniques for bounding these coefficient-value ranges are also discussed (41).

Advanced Techniques

Recent research has proposed several sophisticated methods for effective data summarization in relational database systems. Getoor et al. (42) discuss the application of *Probabilistic Relational Models (PRMs)* (an extension of Bayesian Networks to the relational domain) in computing accurate selectivity estimates for a broad class of relational queries. Deshpande et al. (43) pro-posed *dependency-based histograms*, a novel class of histogram-based synopses that employs the solid foundation of *statistical interaction models* to explicitly identify and exploit the statistical characteristics of the data and, at the same time, address the dimensionality limitations of multidimensional histogram approximations. Spiegel and Polyzotis (44) propose the Tuple-Graph synopses that view the relational database as a semi-structured data graph and employ summarization models inspired by XML techniques in order to approximate the joint distribution of join relationships and values. Finally, Jagadish et al. (45) and Babu et al. (46) develop semantic compression techniques for massive relational tables based on the idea of extracting *data mining models* from an underlying data table, and using these models to effectively compress the table to within user-specified, per-attribute error bounds.

Traditional database systems and approximation techniques are typically based on the ability to make multiple passes over *persistent datasets* that are stored reliably in stable storage. For several emerging application domains, however, data arrives at high rates and needs to be processed on a continuous (24 × 7) basis, without the benefit of several passes over a static, persistent data image. Such *continuous data streams* occur naturally, for example, in the network installations of large telecom and Internet service providers where detailed usage information (call detail records (CDRs), SNMP/RMON packet-flow data, etc.) from different parts of the underlying network needs to be continuously collected and analyzed for interesting trends. As a result, we are witnessing a recent surge of interest in data stream computation, which has led to several (theoretical and practical) studies proposing novel one-pass algorithms for effectively summarizing massive relational data streams in a limited amount of memory (46–55).

APPROXIMATION TECHNIQUES FOR MASSIVE XML DATABASES

XML (56) has rapidly evolved from a markup language for web documents to an emerging standard for data exchange and integration over the Internet. The simple, self-describing nature of the XML standard promises to enable a broad suite of next-generation Internet applications, ranging from intelligent web searching and querying to electronic commerce. In many respects, XML represents an instance of *semistructured data* (57): The underlying data model comprises a labeled graph of *element* nodes, where each element can be either an atomic data item (i.e., raw character data) or a composite data collection consisting of references (represented as graph edges) to other elements in the graph. More formally, an XML database can be modeled as a directed graph $G(V_G, E_G)$, where each node $u \in V_G$ corresponds to a document element, or an element attribute, with label `label`(u). If u is a leaf node, then it can be associated with a value `value`(u). An edge (u, v) denotes either the nesting of v under u in the XML document, or a reference from u to v, through `ID/IDREF` attributes or XLink constructs (58–60).

XML query languages use two basic mechanisms for navigating the XML data graph and retrieving qualifying nodes, namely, *path expressions* and *twig queries*. A path expression specifies a sequence of navigation steps, where each step can be predicated on the existence of sibling paths or on the value content of elements, and the elements at each step can be linked through different structural relationships (e.g., parent-child, ancestor-child, or relationships that involve the order of elements in the document). As an example, the path expression `//author [/book//year = 2003] //paper` will select all paper elements with an author ancestor, which is the root of at least one path that starts with book and ends in year, and the value of the ending element is 2003. The example expression is written in the XPath (61) language, which lies at the core of XQuery (62) and XSLT (63), the dominant proposals from W3C for querying and transforming XML data.

A twig query uses multiple path expressions in order to express a complex navigation of the document graph and retrieve combinations of elements that are linked through specific structural relationships. As an example, consider the following twig query, which is expressed in the XQuery (62) language: `for $a in //author, $p in $a//paper/title, $b in $a//book/title`. The evaluation of the path expressions proceeds in a nested-loops fashion, by using the results of "parent" paths in order to evaluate "nested" paths. Thus, the first expression retrieves all authors, and, for each one, the nested paths retrieve the titles of their papers and books. The final result contains all possible combinations of an author node, with a paper title node and a book title node that it reaches. Twig queries represent the equivalent of the SQL `FROM` clause in the XML world, as they model the generation of element tuples, which will eventually be processed to compute the final result of the XML query.

The goal of existing XML data reduction techniques is to summarize, in limited space, the key statistical properties of an XML database in order to provide *selectivity estimates* for the result size of path expressions or twig queries. Selectivity estimation is a key step in the optimization of declarative queries over XML repositories and is thus key for the effective implementation of high-level query languages (64–66). Given the form of path expressions and twig queries, an effective XML summary needs to capture accurately both the path structure of the data graph and the value distributions that are embedded therein. In that respect, summarizing XML data is a more complex problem than relational summarization, which focuses mainly on value distributions. As with any approximation method, the proposed XML techniques store compressed distribution information on specific characteristics of the data, and use statistical assumptions in order to compensate for the loss of detail due to compression. Depending on the specifics of the summarization model, the proposed techniques can be broadly classified in three categories: (1) techniques that use a graph synopsis, (2) techniques that use a relational summarization method, such as histograms or sampling, and (3) techniques that use a Markovian model of path distribution. It should be noted that, conceptually, the proposed summarization techniques can also be used to provide approximate answers for XML queries; this direction, however, has not been explored yet in the current literature and it is likely to become an active area of research in the near future.

Graph-Synopsis-Based Techniques

At an abstract level, a graph synopsis summarizes the basic path structure of the document graph. More formally, given a data graph $G = (V_G, E_G)$, a graph synopsis $S(G) = (V_S, E_S)$ is a directed node-labeled graph, where (1) each node $v \in V_S$ corresponds to a subset of element (or attribute) nodes in V_G (termed the *extent* of) that have the *same label*, and (2) an edge in $(u, v) \in E_G$ is represented in E_S as an edge between the nodes whose extents contain the two endpoints u and v. For each node u, the graph synopsis records the common tag of its elements and a count field for the size of its extent.

In order to capture different properties of the underlying path structure and value content, a graph synopsis is augmented with appropriate, localized distribution information. As an example, the structural XSKETCH-summary mechanism (67), which can estimate the selectivity of simple path expressions with branching predicates, augments the general graph-synopsis model with localized per-edge stability information, indicating whether the synopsis edge is *backward-stable or forward-stable*. In short, an edge (u, v) in the synopsis is said to be *forward-stable* if all the elements of u have at least one child in v; similarly, (u, v) is backward-stable if all the elements in v have at least one parent in u [note that backward/forward (B/F) stability is essentially a localized form of *graph bisimilarity* (68)]. Overall, edge stabilities capture key properties of the connectivity between different synopsis nodes and can summarize the underlying *path structure* of the input XML data.

In a follow-up study (69), the structural XSketch model is augmented with localized per-node value distribution summaries. More specifically, for each node u that represents elements with values, the synopsis records a summary $H(u)$, which captures the corresponding value distribution and thus enables selectivity estimates for value-based predicates. Correlations among different value distributions can be captured by a multidimensional summary $H(u)$, which approximates the *joint* distribution of values under u and under different parts of the document. It should be noted that, for the single-dimensional case, $H(u)$ can be implemented with any relational summarization technique; the multidimensional case, however, imposes certain restrictions due to the semantics of path expressions, and thus needs specialized techniques that can estimate the number of *distinct* values in a distribution [examples of such techniques are range-histograms (69), and distinct sampling (70)].

The TwigXSketch (71) model is a generalization of the XSketch synopses that deals with selectivity estimation for twig queries. Briefly, the key idea in TwigXSketches is to capture, in addition to localized stability information, the distribution of document edges for the elements in each node's extent. In particular, each synopsis node records an *edge histogram*, which summarizes the distribution of child counts across different stable ancestor or descendant edges. As a simple example, consider a synopsis node u and two emanating synopsis edges (u, v) and (u, w); a two-dimensional edge histogram $H_u(c_1, c_2)$ would capture the fraction of data elements in $\mathtt{extent}(u)$ that have exactly c_1 children in $\mathtt{extent}(v)$ and c_2 children in $\mathtt{extent}(w)$. Overall, TwigXSketches store more fine-grained information on the path structure of the data, and can thus capture, in more detail, the *joint* distribution of path counts between the elements of the XML dataset.

Recent studies (73–75) have proposed a variant of graph-synopses that employ a clustering-based model in order to capture the path and value distribution of the underlying XML data. Under this model, each synopsis node is viewed as a "cluster" and the enclosed elements are assumed to be represented by a corresponding "centroid," which is derived in turn by the aggregate characteristics of the enclosed XML elements. The TreeSketch (72) model, for instance, defines the centroid of a node u_i as a vector of average child counts $(c_1, c_2, \cdots, c_n)$, where c_j is the average child count from elements in A_i to every other node u_j. Thus, the assumption is that each element in u_i has exactly c_j children to node u_j. Furthermore, the clustering error, that is, the difference between the actual child counts in u_i and the centroid, provides a measure of the error of approximation. The TreeSketch study has shown that a partitioning of elements with low clustering error provides an accurate approximation of the path distribution, and essentially enables low-error selectivity estimates for structural twig queries. A follow-up study has introduced the XClusters (73) model that extends the basic TreeSketch synopses with information on element content. The main idea is to augment the centroid of each cluster with a value-summary that approximates the distribution of values in the enclosed elements. The study considers three types of content: numerical values queried with range predicates, string values queried with substring predicates, and text values queried with term-containment predicates. Thus, the key novelty of XClusters is that they provide a unified platform for summarizing the structural and heterogeneous value content of an XML data set. Finally, Zhang et al. have proposed the XSeed (74) framework for summarizing the recursive structure of an XML data set. An XSeed summary resembles a TreeSketch synopsis where all elements of the same tag are mapped to a single cluster. The difference is that each synopsis edge may be annotated with multiple counts, one per recursive level in the underlying data. To illustrate this, consider an element path $/e_1/e'_1/e_2/e'_2$ where e_1 and e_2 correspond to cluster u and e'_1 and e'_2 to cluster u'. The sub-path e_1/e'_1 will map to an edge between u and u' and will contribute to the first-level child count. The sub-path e_2/e'_2 will map to the same edge, but will contribute to the second-level child count from u to u'. Hence, XSeed stores more fine-grained information compared to a TreeSketch synopsis that uses a single count for all possible levels. This level-based information is used by the estimation algorithm in order to approximate more accurately the selectivity of recursive queries (i.e., with the "//" axis) on recursive data.

Histogram- and Sampling-Based Techniques

Several XML-related studies attempt to leverage the available relational summarization techniques by casting the XML summarization problem into a relational context. More specifically, the proposed techniques represent the path structure and value content of the XML data in terms of flat value distributions, which are then summarized using an appropriate relational technique.

The StatiX (75) framework uses histogram-based techniques and targets selectivity estimation for twig queries over tree-structured data (note, however, that StatiX needs the schema of the XML data in order to determine the set of histograms, which makes the technique nonapplicable to the general case of schema-less documents). StatiX partitions document elements according to their schema type and represents each group as a set of (*pid*, *count*) pairs, where *pid* is the id of some element p and *count* is the number of elements in the specific partition that have p as parent. Obviously, this scheme encodes the joint distribution of children counts for the elements of each partition. This information is then compressed using standard relational histograms, by treating *pid* as the value and *count* as the frequency information.

A similar approach is followed in *position histograms* (76), which target selectivity estimation for two-step path expressions of the form A//B. In this technique, each element is represented as a point (s, e) in 2-dimensional space, where s and e are the start and end values of the element in the depth-first traversal of the document tree; thus, (s_a, e_a) is an ancestor of (s_b, e_b), if $s_a < s_b < e_b < e_a$. The proposed summarization model contains, for each tag in the document, one spatial histogram that summarizes the distribution of the corresponding element points. A spatial join between histograms is then sufficient to approximate the ancestor-descendant relationship between elements of different tags.

A recent study (77) has introduced two summarization models, namely the Position Model and the Interval Model, which are conceptually similar to position histograms but use a different encoding of structural relationships. Again, the focus is on selectivity estimation for two-step paths of the form A//B. The Position Model encodes each element in A as a point (s_a, e_a), and each element in B as a point s_b; the selectivity is then computed as the number of s_b points contained under an (s_a, e_a) interval. In the Interval Model, a covering matrix C records the number of points in A whose interval includes a specific start position, whereas a position matrix P includes the start positions of elements in B; the estimate is then computed by joining the two tables and summing up the number of matched intervals. Clearly, both models reduce the XML estimation problem to operations on flat value distributions, which can be approximated using relational summarization techniques.

Markov-Model-Based Techniques

At an abstract level, the path distribution of an XML dataset can be modeled with the probability of observing a specific tag as the next step of an existing path. Recent studies have investigated data reduction techniques that summarize the path structure by approximating, in limited space, the resulting path probability distribution. The principal idea of the proposed techniques is to compress the probability distribution through a Markovian assumption: If p is a path that appears in the document and l is a tag, then the probability that p/l is also a path depends only on a suffix $\bar{p}$ of p (i.e., the next step is not affected by distant ancestor tags). Formally, this assumption can be expressed as $P[p/l] = P[p] \cdot P[l|\bar{p}]$, where $P[q]$ is the probability of observing path q in the data. Of course, the validity of this independence assumption affects heavily the accuracy of the summarization methods.

Recent studies (78,79) have investigated the application of a k-order Markovian assumption, which limits the statistically correlated suffix of p to a maximum predefined length of k. Thus, only paths of length up to k need to be stored in order to perform selectivity estimation. The proposed techniques further compress this information with a *Markov histogram*, which records the most frequently occurring such paths. Less frequent paths are grouped together, either according to their prefix (if the aggregate frequency is high enough), or in a generic ’*’ bucket. In order to estimate the occurrence probability of a longer path, the estimation framework identifies subpaths that are present in the Markov histogram and combines the recorded probabilities using the Markovian independence assumption.

Correlated Suffix Trees (CSTs) (80) employ a similar Markovian assumption in order to estimate the selectivity of twig queries. The path distribution of the document is stored in a tree structure, which records the most frequently occurring suffixes of root-to-leaf paths; thus, the tree encodes frequent paths of variable lengths, instead of using a predefined fixed length as the Markov Histogram approach. In addition to frequent path suffixes, the summary records a *hash signature* for each outgoing path of a tree node, which encodes the set of elements in the node’s extent that have at least one matching outgoing document path. Intuitively, an “intersection” of hash signatures, where each signature corresponds to a different label path, approximates the number of elements that have descendants along all represented paths. Combined with path frequency information, this information yields an approximation of the joint path-count distribution for different subsets of document elements.

BIBLIOGRAPHY

1. D. Barbarà, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik, The New Jersey data reduction report, *IEEE Data Eng. Bull.*, **20**(4): 3–45, 1997, (Special Issue on Data Reduction Techniques).
2. M. Garofalakis and P. B. Gibbons, Approximate query processing: Taming the Terabytes, Tutorial in *27th Intl. Conf. on Very Large Data Bases, Roma, Italy*, September 2001.
3. S. Chaudhuri, R. Motwani, and V. Narasayya, On random sampling over joins, in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, May 1999, pp. 263–274.
4. P. B. Gibbons and Y. Matias, New sampling-based summary statistics for improving approximate query answers, in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, WA, June 1998, pp. 331–342.
5. P. J. Haas and A. N. Swami, Sequential sampling procedures for query size estimation, in *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 1992, pp. 341–350.
6. R. J. Lipton, J. F. Naughton, and D. A. Schneider, Practical selectivity estimation through adaptive sampling, in *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, Atlantic City, NJ, May 1990, pp. 1–12.
7. R. J. Lipton, J. F. Naughton, D. A. Schneider, and S. Seshadri, Efficient sampling strategies for relational database operations, *Theoret. Comp. Sci.*, **116**: 195–226, 1993.
8. F. Olken and D. Rotem, Simple random sampling from relational databases, in *Proceedings of the Twelfth International Conference on Very Large Data Bases*, Kyoto, Japan, August 1986, pp. 160–169.
9. S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, Join synopses for approximate query answering, in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, May 1999, pp. 275–286.
10. P. J. Haas and J. M. Hellerstein, Ripple joins for online aggregation, in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, May 1999, pp. 287–298.

11. J. M. Hellerstein, P. J. Haas, and H. J. Wang, Online aggregation, in *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, AZ, May 1997.

12. W. G. Cochran, *Sampling Techniques*, 3rd ed. New York: John Wiley & Sons, 1977.

13. C.-E. Särndal, B. Swensson, and J. Wretman, *Model Assisted Survey Sampling*, New York: Springer-Verlag (Springer Series in Statistics), 1992.

14. Y. E. Ioannidis and V. Poosala, Histogram-based approximation of set-valued query answers, in *Proceedings of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, September 1999.

15. Y. E. Ioannidis, Universality of serial histograms, in *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, Dublin, Ireland, August 1993, pp. 256–267.

16. Y. E. Ioannidis and S. Christodoulakis, Optimal histograms for limiting worst-case error propagation in the size of join results, *ACM Trans. Database Sys.*, **18**(4): 709–748, 1993.

17. Y. E. Ioannidis and V. Poosala, Balancing histogram optimality and practicality for query result size estimation, in *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, May 1995, pp. 233–244.

18. H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel, Optimal histograms with quality guarantees, in *Proceedings of the 24th International Conference on Very Large Data Bases*, New York City, NY, August 1998.

19. V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita, Improved histograms for selectivity estimation of range predicates, in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, June 1996, pp. 294–305.

20. C. Faloutsos and I. Kamel, Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension, in *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Minneapolis, MN, May 1994, pp. 4–13.

21. V. Poosala and Y. E. Ioannidis, Selectivity estimation without the attribute value independence assumption, in *Proceedings of the 23rd International Conference on Very Large Data Bases*, Athens, Greece, August 1997, pp. 486–495.

22. M. Muralikrishna and D. J. DeWitt, Equi-depth histograms for estimating selectivity factors for multi-dimensional queries, in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, Chicago, IL, June 1988, pp. 28–36.

23. S. Muthukrishnan, V. Poosala, and T. Suel, On rectangular partitionings in two dimensions: Algorithms, complexity, and applications, in *Proceedings of the Seventh International Conference on Database Theory (ICDT'99)*, Jerusalem, Israel, January 1999.

24. D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi, Approximating multi-dimensional aggregate range queries over real attributes, in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, TX, May 2000.

25. D. Donjerkovic, Y. Ioannidis, and R. Ramakrishnan, Dynamic histograms: Capturing evolving data sets, in *Proceedings of the Sixteenth International Conference on Data Engineering*, San Diego, CA, March 2000.

26. P. B. Gibbons, Y. Matias, and V. Poosala, Fast incremental maintenance of approximate histograms, in *Proceedings of the 23rd International Conference on Very Large Data Bases*, Athens, Greece, August 1997, pp. 466–475.

27. A. Aboulnaga and S. Chaudhuri, Self-tuning histograms: Building histograms without looking at data, in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, May 1999, pp. 181–192.

28. N. Bruno, S. Chaudhuri, and L. Gravano, STHoles: A Multidimensional workload-aware histogram, in *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, May 2001.

29. S. Chaudhuri, R. Motwani, and V. Narasayya, Random sampling for histogram construction: How much is enough?, in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, Seattle*, WA, June 1998.

30. B. Jawerth and W. Sweldens, An overview of wavelet based multiresolution analyses, *SIAM Rev.*, **36**(3): 377–412, 1994.

31. E. J. Stollnitz, T. D. DeRose, and D. H. Salesin, *Wavelets for Computer Graphics—Theory and Applications*, San Francisco, CA: Morgan Kaufmann Publishers, 1996.

32. Y. Matias, J. S. Vitter, and M. Wang, Wavelet-based histograms for selectivity estimation, in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, WA, June 1998, pp. 448–459.

33. K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim., Approximate query processing using wavelets, in *Proceedings of the 26th International Conference on Very Large Data Bases*, Cairo, Egypt, September 2000, pp. 111–122.

34. M. Garofalakis and P. B. Gibbons, Wavelet synopses with error guarantees, in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, WI, June 2002, pp. 476–487.

35. J. S. Vitter and M. Wang, Approximate computation of multidimensional aggregates of sparse data using wavelets, in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, May 1999.

36. M. Garofalakis and P. B. Gibbons, Probabilistic wavelet synopses, *ACM Trans. Database Syst.*, **29** (1): 2004. (SIGMOD/PODS Special Issue).

37. R. R. Schmidt and C. Shahabi, ProPolyne: A fast wavelet-based algorithm for progressive evaluation of polynomial range-sum queries, in *Proceedings of the 8th International Conference on Extending Database Technology (EDBT'2002)*, Prague, Czech Republic, March 2002.

38. Y. Matias, J. S. Vitter, and M. Wang, Dynamic maintenance of wavelet-based histograms, in *Proceedings of the 26th International Conference on Very Large Data Bases*, Cairo, Egypt, September 2000.

39. A. Deligiannakis and N. Roussopoulos, Extended wavelets for multiple measures, in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 2003.

40. M. Garofalakis and A. Kumar, Wavelet synopses for general error metrics, *ACM Trans. Database Syst.*, **30**(4), 2005.

41. S. Guha and B. Harb, Wavelet synopsis for data streams: Minimizing non-euclidean error, in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, IL, August 2005.

42. L. Getoor, B. Taskar, and D. Koller, Selectivity estimation using probabilistic models, in *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, May 2001.

43. A. Deshpande, M. Garofalakis, and R. Rastogi, Independence is good: Dependency-based histogram synopses for high-dimensional data, in *Proceedings of the 2001 ACM SIGMOD*

International Conference on Management of Data, Santa Barbara, CA, May 2001.

44. J. Spiegel and N. Polyzotis, Graph-based synopses for relational selectivity estimation, in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, Chicago, IL, 2006, pp. 205–216.
45. H. V. Jagadish, J. Madar, and R. Ng, Semantic compression and pattern extraction with fascicles, in *Proceedings of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, September 1999, pp. 186–197.
46. S. Babu, M. Garofalakis, and R. Rastogi, SPARTAN: A model-based semantic compression system for massive data tables, in *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, May 2001.
47. N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy, tracking join and self-join sizes in limited storage, in *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Philadeplphia, PA, May 1999.
48. N. Alon, Y. Matias, and M. Szegedy, The space complexity of approximating the frequency moments, in *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, Philadelphia, PA, May 1996, pp. 20–29.
49. A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi, Processing complex aggregate queries over data streams, in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, WI, June 2002, pp. 61–72.
50. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, An approximate L^1-difference algorithm for massive data streams, in *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, New York, NY, October 1999.
51. S. Ganguly, M. Garofalakis, and R. Rastogi, Processing set expressions over continuous update streams, in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 2003.
52. M. Greenwald and S. Khanna, Space-efficient online computation of quantile summaries, in *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, May 2001.
53. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, Surfing wavelets on streams: One-pass summaries for approximate aggregate queries, in *Proceedings of the 27th International Conference on Very Large Data Bases*, Roma, Italy, September 2001.
54. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, How to summarize the universe: Dynamic maintenance of quantiles, in *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002, pp. 454–465.
55. P. Indyk, Stable distributions, pseudorandom generators, embeddings and data stream computation, in *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, Redondo Beach, CA, November 2000, pp. 189–197.
56. N. Thaper, S. Guha, P. Indyk, and N. Koudas, Dynamic multidimensional histograms, in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, WI, June 2002, pp. 428–439.
57. T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, Extensible Markup Language (XML) 1.0, 2nd ed. W3C Recommendation. Available: http://www.w3.org/TR/REC-xml/).
58. S. Abiteboul, Querying semi-structured data, in *Proceedings of the Sixth International Conference on Database Theory (ICDT'97)*, Delphi, Greece, January 1997.
59. R. Goldman and J. Widom, DataGuides: Enabling query formulation and optimization in semistructured databases, in *Proceedings of the 23rd International Conference on Very Large Data Bases*, Athens, Greece, August 1997, pp. 436–445.
60. R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes, Exploiting local similarity for efficient indexing of paths in graph structured data, in *Proceedings of the Eighteenth International Conference on Data Engineering*, San Jose, CA, February 2002.
61. T. Milo and D. Suciu, Index structures for path expressions, in *Proceedings of the Seventh International Conference on Database Theory (ICDT'99)*, Jerusalem, Israel, January 1999.
62. J. Clark, and S. DeRose, XML Path Language (XPath), Version 1.0, W3C Recommendation. Available: http://www.w3.org/TR/xpath/.
63. D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu, XQuery 1.0: An XML query language, W3C Working Draft 07. Available. http://www.w3.org/TR/xquery/).
64. J. Clark, XSL Transformations (XSLT), Version 1.0, W3C Recommendation. Available: http://www.w3.org/TR/xslt/).
65. Z. Chen, H. V. Jagadish, L. V. S. Lakshmanan, and S. Paparizos, From tree patterns to generalized tree patterns: On efficient evaluation of XQuery, in *Proceedings of the 29th International Conference on Very Large* Data Bases, Berlin, Germany, September 2003.
66. A. Halverson, J. Burger, L. Galanis, A. Kini, R. Krishnamurthy, A. N. Rao, F. Tian, S. Viglas, Y. Wang, J. F. Naughton, and D. J. DeWitt, Mixed mode XML query processing, in *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, September 2003.
67. J. McHugh and J. Widom, Query optimization for XML, in *Proceedings of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, September 1999.
68. N. Polyzotis and M. Garofalakis, Statistical synopses for graph-structured XML databases, in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, WI, June 2002.
69. R. Milner, *Communication and Concurrency*, Englewood Cliffs, NJ: Prentice Hall (Intl. Series in Computer Science), 1989.
70. N. Polyzotis and M. Garofalakis, Structure and value synopses for XML data graphs, in *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002.
71. P. B. Gibbons, Distinct sampling for highly-accurate answers to distinct values queries and event reports, in *Proceedings of the 27th International Conference on Very Large Data Bases*, Roma, Italy, September 2001.
72. N. Polyzotis, M. Garofalakis, and Y. Ioannidis, Selectivity estimation for XML twigs, in *Proceedings of the Twentieth International Conference on Data Engineering*, Boston, MA, March 2004.
73. N. Polyzotis, M. Garofalakis, and Y. Ioannidis, Approximate XML query answers, in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004, pp. 263–274.
74. N. Polyzotis and M. Garofalakis, XCluster synopses for structured XML content, in *Proceedings of the 22nd International Conference on Data Engineering*, 2006.
75. N. Zhang, M.T. Ozsu, A. Aboulnaga, and I.F. Ilyas, XSEED: Accurate and fast cardinality estimation for XPath queries, in *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

76. J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Siméon, StatiX: Making XML count, in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, WI, June 2002.

77. Y. Wu, J. M. Patel, and H. V. Jagadish, Estimating answer sizes for XML queries, in *Proceedings of the 8th International Conference on Extending Database Technology (EDBT'2002)*, Prague, Czech Republic, March 2002.

78. W. Wang, H. Jiang, H. Lu, and J. X. Yu, Containment join size estimation: Models and methods, in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 2003.

79. A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton, Estimating the selectivity of XML path expressions for internet scale applications, in *Proceedings of the 27th International Conference on Very Large Data Bases*, Roma, Italy, September 2001.

80. L. Lim, M. Wang, S. Padamanabhan, J. S. Vitter, and R. Parr, XPath-Learner: An on-line self-tuning Markov histogram for XML path selectivity estimation, in *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002.

81. Z. Chen, H. V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. Ng, and D. Srivastava, Counting twig matches in a tree, in *Proceedings of the Seventeenth International Conference on Data Engineering*, Heidelberg, Germany, April 2001.

Minos Garofalakis
Yahoo! Research
Berkeley, California, and
University of California, Berkeley
Santa Clara, California

Neoklis Polyzotis
University of California,
Santa Cruz
Santa Cruz, California

VISUAL DATABASE

INTRODUCTION

We humans are most adept and comfortable with visual information, using our vision for routine activities as well as for sophisticated reasoning and planning. The increased use of images, video, and graphics in computing environments has catalyzed the development of data management tools for accessing, retrieving, and presenting visual information or alphanumeric information in visual ways. Some of these tools are developed to create, store, and retrieve visual data in the form of images or video to realize advanced multimedia applications. Other tools use extensively visual representation in user interfaces and provide essential visual mechanisms to access alphanumeric or visual data in databases. The development of these tools involves concepts, algorithms, and techniques from diversified fields such as database management systems, image processing, information retrieval, and data visualization. Consequently, the field of visual database represents several diversified areas. These areas include at least the following:

1. Database systems to manage visual information in the form of images, video, or geometric entities; that is, the data subject is visual.
2. Visual interfaces to query databases; that is, the query interface is visual.
3. Visual interfaces to interpret information retrieved from databases; that is, the data presentation is visual.

The first area is related closely to the study of image and multimedia databases. The second area has an impact beyond databases and has been studied in programming languages and programming environments. The third area is the focus of data visualization study. These areas are related closely with each other: The management of visual data needs visual queries and visual presentation of query results; on the other hand, the last two areas are applicable not only to databases managing visual data but also to alphanumeric databases. This article focuses on the first area and the visual query interfaces of databases in the first area. In other words, this article introduces principles and techniques developed for the management of visual data and techniques on visual interfaces to formulate queries against databases that manage the visual data.

In a database system that manages visual data, many issues exist to consider beyond those in alphanumeric databases. Using image database as an example, these issues typically include:

- Image features—what image features are most useful in each particular application?
- Content-based retrieval—how are images searched effectively and efficiently based on image contents?
- Storage and indexing—what are the proper storage techniques for large amounts of image data, and what are the proper indexing techniques to support content-based retrieval of image data?
- User interface—how should the user browse and search for images? What is a suitable user interface? How is an image database made accessible through the World Wide Web?

Other issues include those unique to visual databases in query processing, transaction management, and database recovery. In recent years, researchers have paid attention to bridging the semantic gap between low-level image features and high-level semantic concepts. This article gives an overview of principles and techniques on these topics. Given the broad and fast-expanding nature of this field, we are bound to have omitted significant contents and references. In addition, we devote a disproportionate amount of attention to some issues at the expense of others. It is better to introduce basic concepts and techniques well rather than to give a quick run-through of broad topics. To give real-world examples, nevertheless, this article presents an overview of three visual database systems. For more comprehensive coverage and technical details, interested readers are invited to consult the survey papers listed in the last section.

HISTORY AND THE STATE OF THE ART

Database management systems, especially relational database management systems, have been extensively studied in the past 40 years. Rigorous theory has been discovered, and efficient techniques have been developed for the management of alphanumeric data. When new types of data such as images and video were first brought into a database environment, it was natural that the data needed to be transformed so that they could be represented in existing database management systems. In a relational database, therefore, visual data are usually represented by their alphanumeric attributes, which are stored as tuples over relations. Kunii et al. (1) first attempted extending a relational database scheme to accommodate visual information by describing the color and textural properties of images. Physical visual data were stored in a separate storage and were linked through references to their tuples in the relational database. The alphanumeric attributes of a visual object are called *features* of the object. The features, which act together as a proxy of the object, are used in answering a user's queries. In the 1970s and the early 1980s, several image database systems (2–5) were developed to manage images and graphics using relational databases. Researchers believed initially that this setting is all what they would need for the management of visual data. They also believed

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

that most relational database techniques, including indexing, query optimization, concurrency control, and recovery, would also work in parallel in the intended environments of visual data management.

It was only after some experience working with visual data that researchers realized the mismatch between the nature of visual data and the way both the user and the system were forced to query and operate on them. One major technical challenge to visual database management is data retrieval. The creation of mere visual data repositories is of little value unless methods are available for flexible retrieval of visual data based on their contents. *Content-based retrieval* means that the search will analyze the actual contents of the image. The term "content" in this context might refer to image features, such as colors, shapes, textures, or any other information that can be derived from the image itself. Without the ability to examine image content, searches must rely on textual information, such as captions or keywords, which in many applications have to be entered manually for every image in the database.

The way people access visual information is fundamentally different to the way they access alphanumeric information. Relational queries and operations are not enough for querying visual data, for which browsing, descriptive query, and query by examples are important query paradigms. Therefore, unique issues in visual database query include how to support the content-based retrieval, how to support image manipulations such as browsing and zooming, and how to support image processing operations for feature extraction. Visual database systems differentiate from each other in their ways to work around these issues. As an example, the QBIC (6) system has a visual user interface in which queries are posed by drawing, selection, and other graphical means. It supports image queries using image features such as color, texture, shape, and motion of images and video. Researchers also attempted to manage geometric and spatial data in relational databases. For example, PSQL (7) extends the relational model in a disciplined way over three spatial data types: points, line segments, and regions. Geometric and spatial data pose unique problems to visual data management. These problems include storage and indexing of spatial data and database query supported by spatial operations.

For relational databases, a classic example of a visual query interface is QBE (8). It is a very user-friendly tabular query language based on domain relational calculus. To formulate a query, a user enters example elements and fills out tables supplied by the system. QBE has influenced the graphical query facilities in modern database products, particularly personal database software. As an extension of QBE to the image database, QPE (9) introduced built-in support of image operations. PICQUERY (10) proposed a comprehensive list of image operations that should be supported by an image database system and classified these operations into image manipulation operations, pattern recognition operations, spatial operations, function operations, user-defined operations, and input/output operations.

After the development of object-oriented databases and logical databases, attempts were made to deal with semantic modeling of images with complex structures (11) and logic for visual databases (12). PROBE (13) is a spatial object-oriented database system whose data model supports two basic types of data, objects and functions, where functions uniformly represent properties, relationships, and operations associated with objects. Chang et al.(14) presented an image database system based on spatial reasoning by two-dimensional (2-D) strings and image database operations. Brink et al. (15) developed a multimedia system that provides a shell/compiler to build metadata atop the physical medium for most media data. With this kind of independence, visual database systems can be built for a variety of applications.

Once image features were extracted, the question remained as to how they could be matched against each other for retrieval. The success of content-based image retrieval depends on the definition of similarity measures to measure quantitatively the similarities between images. A summary of similarity measures was given in Ref. (16), where similarity measures were grouped as feature-based matching, object silhouette-based matching, structural feature matching, salient feature matching, matching at the semantic level, and learning-based approaches for similarity matching.

In addition to designing powerful low-level feature extraction algorithms, researchers have also focused on reducing the semantic gap (16,17) between the image features and the richness of human semantics. One common approach is to use object ontology, which is a simple vocabulary providing qualitative definitions of high-level concepts. Images are classified into different categories by mapping descriptions of image features to semantic keywords. Such an association of low-level features with high-level semantic concepts can take advantages of classification and clustering algorithms developed in machine learning and data mining. In addition to object ontology, query results can be refined by learning the user's intention using *relevance feedback*, which is an interactive process of automatically adjusting the execution of a query using information fed back from the user about the relevance of the results of the query.

In recent years, image retrieval on the World Wide Web has become an active area of research. Systems like Webseer (18) and WebSEEK (19) can locate images on the Web using image contents in addition to texts. MetaSEEK (20) can distribute a user query to several independent image search engines on the Web and then combine the results to give the benefits of all of them. An image search on the Web has enormous commercial potential. Many popular commercial search engines have image and video search functionalities. These search engines include Google Image Search (http://images.google.com), Lycos Search (http://search.lycos.com), AltaVista Image Search (http://www.altavista.com/image), MSN Search (http://www.msn.com), and Yahoo! Image Search (http://images.search.yahoo.com). These search engines use text to search for images or video, with little or no consideration of image contents. Recently, Web-based communities become popular where developers and end users use the Web as a platform for collaboration and resource sharing. Popular examples include Flickr (http://www.flickr.com), which is a photo-sharing website, and YouTube (http://www.youtube.com), which is a video-sharing website. Both websites support image or video search as part of the services they offer.

Given distinct methods and solutions to a problem as open-ended as visual data retrieval, a natural question that develops is how to make a fair comparison among them. Evaluation of visual data retrieval has been an ongoing challenging problem. Perhaps the most complete video evaluation project has been the TRECVID evaluation (21) (http://www-nlpir.nist.gov/projects/trecvid). The project keeps close connections between private industry and academic research where a realistic task-specific test set is gathered, discussed, and agreed to. Several research teams then attempt to provide the best video retrieval system for the test set. TRECVID participants meet regularly for continual evolution toward improving the test sets.

IMAGE FEATURES

Features of an image are a set of terms or measurements that can be used to represent the image. An image can be assigned features either manually or automatically.

A simple and reliable way to catch the content of an image is to annotate manually the image with keywords. In fact, it may be the only way to describe images in many applications. A good example is the San Francisco de Young Museum (http://www.thinker.org), which uses manual annotation to identify images. Each image in the museum's extensive collection has a description, and some descriptions are remarkably detailed. Free text retrieval techniques are used to search for images based on the text descriptions.

Many kinds of images cannot be annotated manually. A typical example is a database of human facial images where it is sometimes impossible to assign a unique keyword to each facial image. Manual annotation is impractical for very large databases or for images that are generated automatically or streamed, for example, from surveillance cameras. In such applications, it is attractive to extract automatically features of visual data for content-based retrieval. However, automatic extraction of image contents is difficult. Except for a few special applications, such as optical character recognition, the current state of the art in image understanding cannot label an image with descriptive terms that convincingly characterize the content of the image. Furthermore, objects in an image can be occluded by other objects or by shadows, be articulated, or have a shape that is difficult to describe. The fundamental problem is that so far it is not clear how human beings recognize images and what features are used by human beings to distinguish one image from another. It is not yet possible for cognitive science to provide practical hints to computer recognition of images.

An image in a database is represented typically by multiple measures. According to the scope on the image these measures are collected, image features can be classified as follows:

- Global features, which include average values, standard derivations, and histograms of measurements calculated for the entire image.
- Regional features, which are measured on fixed regions or regions having homogeneous properties.
- Object features, which are calculated for each object in the image. Objects can be segmented either manually, semiautomatically or automatically. Segmentation techniques include thresholding, flood-fill using seed points, and manual segmentation using shape models.

Image features are calculated typically off-line, and thus, efficient computation is not a critical criterion as opposed to efficient feature matching in answering user queries. Commonly used primitive features include color histogram, texture, shape, and contour measures. These image features are well explained in image processing textbooks such as Refs. (22 and 23).

A histogram of a grayscale image is a plot of the number of pixels for each particular intensity value in the image. A color image can be characterized by three histograms, one for each of the three color components, in terms of either RGB (red, green, and blue) or HSV (hue, saturation, and value). Location information is lost totally in a histogram. A histogram of an given image can be compared with the stored set of histograms in the database to find an image that is similar in color distribution to the given image.

Texture measures include almost all local features such as energy, entropy, contrast, coarseness, correlation, anisotropy, directionality, stripes, and granularity. Texture measures have been used to browse large-scale aerial photographs (24), where each photograph is segmented to obtain a texture image thesaurus. The texture image thesaurus is clustered. The user can indicate a small region of interest on a photograph, and the system will retrieve photographs that have similar textured regions.

Edges and their directions in images are also valuable features. Edge information can be extracted in several ways, for example, by using Gabor filters, Sobel operators, or various edge detectors.

An effective method for representing an image is image transformation, for example, wavelet transformation. The low-frequency coefficients of the transform of an image often represent objects in the image and can be used as features of the image.

The above primitive features can be used to derive new features, for example, curvatures, shapes of regions or objects, locations, and spatial relations. In fact, these "intermediate" features are primitive building blocks of object ontology, which provides qualitative descriptions of image contents and associates images with semantic concepts. The user can continue to mark interesting areas, not necessarily objects, from which features are calculated. An example is a physician marking interesting areas in the positions of lung in chest x-ray images and calculating texture features only from the marked areas.

In addition to the above features that are extracted from individual images, features can also be extracted from a set of images by analysis of correlations among the images. Statistical and machine learning techniques are often applied to a set of images to find uncorrelated components that best describe the set of images. For example, the eigenface technique (25) applies principal component analysis for image feature extraction. Given a set of images and by arranging pixel values of each image into a vector, one can form a covariance matrix of the set of images. Eigenvectors that correspond to the largest eigenvalues of the covariance matrix contain the principal components of the set of images. Each image is then approximated with a linear combination of these eigenvectors and can be represented by the weights of these eigenvectors in the combination. Recently, researchers have developed nonlinear methods (26–28) in manifold learning for feature extraction and dimensionality reduction. By assuming images as points distributed on a manifold in high-dimensional space, these methods can often unfold the manifold to a very low-dimensional space and use the coordinates in the low-dimensional space as image features.

Techniques have also been proposed for feature extraction and matching of three-dimensional (3-D) shapes (29). A histogram can be used as a feature for 3-D shapes in the same way as it is used for 2-D images. Curvature and curvature histogram have also been used as features of 3-D shapes. Another promising way to represent 3-D shapes is to use skeletal structures. One widely used skeletal structure is the medial axis model (30). Recently, an interesting approach called topological matching (31) has been proposed to represent 3-D shapes using multiresolution Reeb graphs. The graphs represent skeletal and topological structures of 3-D shapes, are invariant to geometrical transformations, and are particularly useful for interactive search of 3-D objects.

A video is often characterized by selecting key frames that form a “story board” of the video (32). Once key frames are obtained, one can use image retrieval techniques to search a video database for frames that match a key frame. The story-board frames can be obtained by carrying out a frame-to-frame analysis that looks for significant changes and by taking advantage of domain knowledge of how a video is constructed.

Using an improper set of features may get irrelevant query results. For example, an image histogram loses location information of image pixels. If the query image contains a small region of particular importance, the region will get lost in an overall histogram. A solution to the problem is to partition the image and to calculate the features for each partition of the image. The user can indicate a region of interest in the query image, and only features of that portion will be used in the similarity measure.

Different image features are often used together for image retrieval. In a human facial image retrieval system (33), for example, a human facial image is characterized by six facial aspects: chin, hair, eyes, eyebrows, nose, and mouth. Each facial aspect contains numerical attributes as well as possibly descriptive values (such as large and small) for each attribute. In practice, these feature measures are often used together with other statistical measures, such as textures or principal components of colors.

Matching long complex feature vectors is often expensive computationally. Because some features are usually more useful than the others, a subset of the most useful feature measures can often be selected manually in practice to accelerate the matching process. Techniques such as principal component analysis and vector quantization can be applied to a set of image feature vectors. Principal component analysis is applied to find a subset of significant feature measures. Vector quantization is used to quantize the feature space. For instance, a set of images can be clustered, and each cluster is represented by a prototype vector. The search starts by comparing the query feature vector with the prototype vectors to find the closest cluster and then to find the closest feature vectors within that cluster. In this way, a visual data search can be directed in a hierarchical manner.

VISUAL DATA RETRIEVAL

One of the biggest challenges for a visual database comes from the user requirement for visual retrieval and techniques to support visual retrieval. Visual retrieval has to support more retrieval methods in addition to those conventional retrieval methods in alphanumeric databases. Correspondingly, these extra retrieval methods call for new data indexing mechanisms. Figure 1 summarizes typical retrieval methods for image databases and their relationships with different kinds of data indexes. Retrieval by attributes is a conventional data access method. The user selects features that might characterize the type of expected images; for example, use color and texture measures to find dresses or wallpapers. The user can also select the weight of each feature for feature matching. Image retrieval can then be made by the comparison of weighted differences of the features of the query image against the features of the images in the database.

Other retrieval methods, namely, free text retrieval, fuzzy retrieval, visual browsing, and similarity retrieval, are content-based retrieval and are special to visual databases. Free text retrieval and fuzzy retrieval are descriptive. Visual browsing and similarity retrieval are visual and require visual interfaces. Unlike query processing with well-defined formalities in alphanumeric databases, similarity retrieval intends to combine image features and texts to retrieve images similar to an example image. A user can specify the example image in several ways. Usually, the user shows a real image and tells the system what is important in the image by selecting features of or regions/objects in the image (the regions/objects can either be automatically or manually segmented). Alternatively, the user can sketch an example image. The sketch can be a contour sketch showing the shapes of the objects to retrieve or in the form of colored boxes indicating colors, sizes, and positions of the objects to retrieve. Using a well-defined set of features and a good similarity measure, the system retrieves images similar to the one supplied by the user.

One problem in content-based retrieval is how to define the similarity measure. In general, we cannot expect to find a generic similarity measure that suites all user needs.

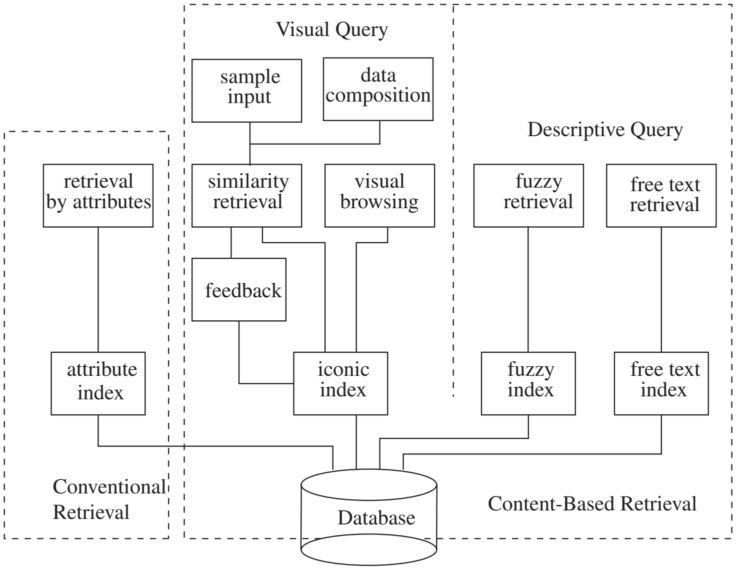


Figure 1. Visual retrieval methods.

Manually selecting relevant features and weights in a similarity measure can be difficult, especially when the features are complicated. A user should therefore be given a chance to interact with the system and to have the similarity learned by the system. This interaction can be more or less natural for the user. *Relevance feedback*(34) is a technique that removes the burden of specifying manually the weights of features. It allows the user to grade the retrieved images by their relevance, for example, highly relevant, relevant, nonrelevant, or highly nonrelevant. The system then uses the grades to learn relevant features and similarity metrics. The grading can be made after each retrieval, thereby progressively improving the quality of the retrieved results.

Other problems in content-based retrieval include how to organize and index the extracted features, how to incorporate these features and indexes into existing databases, and how to formulate visual queries through user interfaces.

Descriptive Retrieval

Free text retrieval uses text descriptions of visual data to retrieve the visual data. Many techniques, typically those (35) developed for and used in Web search engines, can be used for free text retrieval of visual data. Many of these approaches work in similar ways. First, every word in the text description is checked against a list of stop words. The list consists of the commonly used words, such as “the”, and “a,” which bear little semantic significance within the context. Words in the stop word list are ignored. Words such as “on,” “in,” and “near” are essential to represent the location of special features, and therefore, they cannot be included in the stop word list. The remaining words are then stemmed to remove the word variants. Stemmed words are indexed using conventional database indexing techniques. In free text retrieval, users are asked to submit a query in free text format, which means that the query can be a sentence, a short paragraph, or a list of keywords and/or phrases. After initial search, records that are among the best matches are presented to the user in an sorted order according to their relevances to the search words.

Human descriptions of image contents are neither exact nor objective. Image retrieval based on fuzzy queries (36) is a natural way to get information in many applications. An image is represented by a set of segmented regions, each of which is characterized by a fuzzy feature (fuzzy set) reflecting color, texture, and shape properties. As a result, an image is associated with a family of fuzzy sets corresponding to regions. The fuzzy sets together define a *fuzzy space*. When a fuzzy query is defined incompletely, it represents a hypercube in the fuzzy space. To evaluate a fuzzy query is to map image feature measures from the feature space to the fuzzy space. After mapping, images in the database are represented as points in the fuzzy space. The similarity between a fuzzy query and an image can then be computed as the distance between the point that represents the image and the hypercube that represents the fuzzy query. Images that are close to the query definition are retrieved. The fuzzy query processing produces an ordered set of images that best fit a query. Fuzzy space is not Cartesian; ordinary correlation and Euclidean distance measurements may not be used as similarity measures.

Visual Retrieval

Two types of visual retrievals exist: similarity retrieval and visual browsing. Similarity retrieval allows users to search for images most similar to a given sample image. Visual browsing allows users to browse through the entire collection of images in several different ways based on similarities between images.

Similarity retrieval uses similarity measures to search for similar images. To accelerate the search process, similarity retrieval is often implemented by traversing a multidimensional tree index. It behaves in a similar way as pattern classification via a decision tree. At each level of the tree, a decision is made by using a similarity measure. At the leaf node level, all leaf nodes similar to the sample image will be selected. By modifying weights of different features, one gets different similarity measures and, consequently, different query results.

Visual browsing allows a user to thumb through images, which again can be achieved via a traversal of the index tree, where each internal node piggybacks an iconic image representing the node. The system presents the user with the root of the index tree by displaying the icons of its descendants. At each node of the tree, the user chooses browsing directions: up, down, left, and right. Going up is implemented via a pointer to its parent node, whereas moving down is accomplished by selecting the icon representing a specific descendant node. The selected descendant node is then considered as the current node, and the icons of its children are displayed. This process is vertical browsing. By selecting icons of sibling nodes, a user can perform horizontal browsing in a similar way.

A graphic user interface is required to retrieve images or video in an interactive “thumb through” mode. The system displays to the user retrieved icons of images that are relevant to the query image, and the user can iterate by clicking an image that best satisfies the query. Such an intuitive user interface combines browsing, search, navigation, and relevance feedback. It may address demands from a wide variety of users, including nontechnical users.

VISUAL DATA INDEXING

Indexing is a process of associating a key with corresponding data records. In addition to traditional tree-based and hash-based indexing techniques used in alphanumeric databases, visual databases need special indexing techniques to manage, present, and access visual data. Over the years, many indexing techniques have been proposed for various types of applications. Using image database as an example, we can classify these indexing techniques into two categories: interimage indexing and intraimage indexing. Interimage indexing assumes each image as an element and indexes a set of such images by using their features. A representative interimage indexing technique is feature-based iconic indexing (33). It indexes a set of visual data to support both feature-based similarity query and visual browsing in addition to exact-match query and range query on the set of data. On the other hand, intraimage indexing indexes elements (regions, points, and lines) within an image. Representative techniques are quadtree and related hierarchical data structures.

Feature-Based Iconic Indexing

Because features can be assumed as vectors in high-dimensional space, any multidimensional indexing mechanisms (37) that support a nearest-neighbor search can be used to index visual data. Although this generic mechanism is universal, various specialized, more efficient indexing methodologies may develop for particular types of visual database applications.

An image indexing mechanism should support both a feature-based similarity search and visual browsing. As an example, feature-based iconic indexing (33) is an indexing mechanism that is specialized for visual retrieval. In alphanumeric databases, tree indexes are organized such that (*1*) a tree is built on a set of key attributes; (*2*) key attributes are of primitive data types, such as string, integer, or real; and (*3*) the grouping criterion usually specifies a range of values on a particular attribute. To make index trees applicable to visual features, feature-based iconic indexing makes the following generalizations:

1. The first generalization is to allow attributes to be structured data. For example, an attribute can be a feature vector that is a multidimensional array.
2. As a result of the first generalization, the grouping criteria are defined on similarity measures. The most commonly used similarity measure is distance in the feature space.
3. The third generalization is that different levels of the index tree may have different key attributes. This generalization facilitates visual browsing and navigation.

The iconic index tree can be built either top-down or bottom-up. A top-down algorithm typically consists of two steps. The first step selects a feature aspect and clusters the images into *m* classes. Many approaches are available to do this step, for example, the *k*-means clustering algorithm (38) or the self-organization neural network (39). The second step repeats the first step until each node has at most *m* descendants. Each node of the index tree has the following fields: (*1*) node type: root, internal, or leaf node; (*2*) feature vector; (*3*) iconic image; (*4*) pointer to its parent node; (*5*) number of children and pointers to its children; and (*6*) number of sibling nodes and pointers to the sibling nodes. The pointers to sibling nodes are referred to as horizontal links and provide a means for horizontal browsing. An example node structure of a feature-based iconic index tree is shown in Fig. 2. Horizontal links are created only for the feature aspects that have already been used as the key feature aspect at levels above the current level. For example, if F3 is the key feature aspect at the current level, F1 and F2 are key feature aspects at above levels; then the nodes under the same parent node at this level represent images having similar F1, similar F2, and different F3s. For horizontal browsing of images of similar F1, similar F2, and different F3s, a horizontal link is created to link these nodes at this level. Other horizontal links such as a link for similar F1, similar F3, and different F2 can be created as well.

Quadtree and Other Hierarchical Data Structures

Quadtree and related hierarchical data structures provide mechanisms for intraimage indexing. As an example of the type of problems to which these data structures are applicable, consider a geographical information system that consists of several of maps, each of which contains images, regions, line segments, and points. A typical query is to

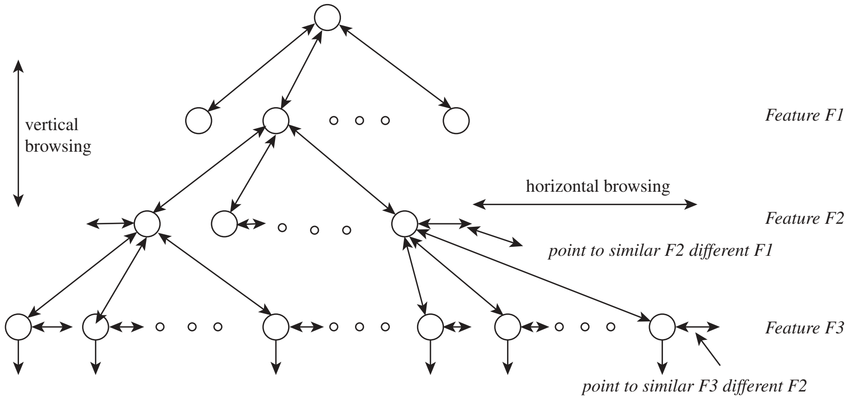


Figure 2. Feature-based iconic index tree facilitates similarity search and visual browsing. Reprinted with permission from Ref. 33.

determine all hotels within 20 miles from the user's current location. Another typical query is to determine the regions that are within 1000 feet above sea level in a geographic area. Such analyses could be costly, depending on the way the data are represented. Hierarchical data structures are based on the principle of recursive decomposition of space. They are useful because of their ability to organize recursively subparts of an image in a hierarchical way.

Quadtree has been studied extensively to represent regional data. It is based on successive subdivision of a binary image into four equal-sized quadrants. If the binary image does not consist entirely of 1s or 0s, it is then divided into quadrants, subquadrants, and so on, until blocks (single pixels in extreme cases) that consist entirely of 1s or 0s are obtained. Each nonleaf node of a quadtree has four children. The root node corresponds to the entire image. Each child of a node represents a quadrant of the region represented by the node. Leaf nodes of the tree correspond to those blocks for which no additional subdivision is necessary.

Hierarchical data structures have been studied extensively, and many variants have been proposed (37). They are differentiated mainly on the following aspects: (*1*) the type of data (points, line segments, or regions) that they are used to represent, (*2*) the principle guiding the decomposition process, and (*3*) tree balance (fixed or variable depth). Hierarchical data structures have been used to represent point data, regions, curves, surfaces, and volumes. It is worthwhile to mention that, instead of equal subdivision in quadtrees, the decomposition process may divide a binary image according to the distribution of elements in the image. Furthermore, the idea of decomposition is not restricted to 2-D images and can be extended to data distributions in high-dimensional space. For example, kd-tree and its derivatives are studied extensively to index point data in high-dimensional space. These techniques have other applications beyond visual databases. These applications include alphanumeric databases, data warehousing, and data visualization.

OTHER DATABASE ISSUES

A typical database management system consists of modules for query processing, transaction management, crash recovery, and buffer management. For a complete discussion of technical details, readers may refer to a database textbook such as Ref. (40). These modules in a visual database management system are different from the corresponding ones in traditional alphanumeric databases.

Query Processing

As we have introduced, query processing in visual databases must support a similarity match in addition to an exact match. Similarity measures between visual objects are usually real valued, say, ranging from 0 (completely different) to 1 (exactly the same). Strictly speaking, the result of a visual query can be all images in the entire database, each ranked from 0 to 1 based on its similarity to the query image. The user will specify a threshold so that an image is not retrieved if its rank is below the threshold value. Images with ranks above the threshold value are then ordered according to their ranks. A similarity search is often computing intensive. For query optimization, therefore, visual database systems supporting a similarity search and user-defined access methods need to know the costs associated with these methods. The costs of user-defined functions in terms of low-level access methods, such as those related to similarity search, must also be made known to the database system (41).

Transaction Management

In visual databases, traditional lock-based and timestamp-based concurrency control techniques can be used; the results would still be correct. However, the concurrency of the overall system would suffer because transactions in visual databases tend to be long, computing-intensive, interactive, cooperative, and to refer to many other database objects. Suppose an updating transaction inserts interactively subtitles to a video, for example; traditional lock-based concurrency control locks the entire video, which decreases the throughput of other transactions referring to other frames in the same video. Visual data are large in size, making it impractical to create multiple copies of the data, as is necessary in the multiversion approach to concurrency control. Optimistic methods for concurrency control are not suitable either, as frequent abortion and restart of a visual presentation would be unacceptable to the viewer.

To increase system concurrency in such an environment, transaction models defined for object-oriented environments, long cooperative activities, real-time database applications, and workflow management are usually considered. Techniques developed for nested transactions are particularly useful, in which a transaction consists of subtransactions, each of which can commit or abort independently. Another observation is that even though a nonserializable schedule may leave the database in an inconsistent state, the inconsistent state may not be fatal. If a few contiguous frames of a video presentation have been changed by another transaction, then, such subtle changes usually would not cause any problem to the viewers.

Storage Management

One challenge of storage management of visual data is to serve multiple requests for multiple data to guarantee that these requests do not starve while minimizing the delay and the buffer space used. Techniques such as data striping, data compression, and storage hierarchies have been employed to reduce this bottleneck. Data striping, as is used in redundant array of inexpensive disks (RAIDs), allocates space for a visual object across several parallel devices to maximize data throughput. Also studied are storage hierarchies in which tertiary storage can be used for less frequently used or high-resolution visual objects and faster devices for more frequently used or low-resolution visual objects. As hard disks become larger and cheaper, however, tertiary storage is less necessary than it was before.

Recovery

Many advanced transaction models have generalized recovery methods. In a long cooperative design environment, undoing complete transactions is wasteful. A potentially large amount of work, some of it correct, might not have to be undone. It makes more sense to remove the effects of individual operations rather than undo a whole transaction. To do so, however, the log must contain not only the history of a transaction but also the dependencies among individual operations.

Advanced transaction models for long-running activities include compensating transactions and contingency transactions. A compensating transaction undoes the effect of an already committed transaction. In video delivery, for example, a unique concern is how to compensate a transaction with acceptable quality of service. Unlike compensating transaction, a contingency transaction provides alternative routes to a transaction that could not be committed. For example, a contingency transaction for showing an image in GIF format might show the image in JPEG format.

EXAMPLE VISUAL DATABASE SYSTEMS

Because of space limitations, this section introduces three example visual database systems: QBIC (6) by IBM; MARS (34) developed at the University of Illinois at Urbana-Champaign; and VisualSEEK (19,42) developed at Columbia University. The last section gives references to other visual database systems

QBIC

QBIC was probably the first commercial content-based image retrieval system. It is available either in stand-alone form or as part of other IBM DB2 database products. The features it uses include color, texture, shape, and keywords.

QBIC quantizes each color in the RGB color space into k predefined levels, giving the color space a total of k^3 cells. Color cells are aggregated into super cells using a clustering algorithm. The color histogram of an image represents the normalized count of pixels falling into each super cell. To answer a similarity query, the histogram of the query image is matched with the histograms of images in the database. The difference z_i is computed for each color super cell.The similarity measure is given as $\sum_{i,j} a_{ij} z_i z_j$, where a_{ij} measures the similarity of the ith and the jth colors.

Texture measures used by QBIC include coarseness, contrast, and directionality. Coarseness measures texture scale (average size of regions that have the same intensity). Contrast measures vividness of the texture (depending on the variance of the gray-level histogram). Directionality gives the main direction of the image texture (depending on the number and shape of peaks of the distribution of gradient directions).

QBIC also uses shape information, such as area, major axis orientation (eigenvector of the principal component), and eccentricity (ratio of the largest eigenvalue against the smallest eigenvalue); various invariants; and tangent angles as image features. It also supports search of video data by using motion and shot detection.

QBIC supports queries based on example images, user-constructed sketches and drawings, and color and texture patterns. The user can select colors and color distributions, select textures, and adjust relative weights among the features. Segmentation into objects can be done either fully automatically (for a restricted class of images) or semiautomatically. In the semiautomatic approach, segmentation is made in a flood-fill manner where the user starts by marking a pixel inside an object and the system grows the area to include adjacent pixels with sufficiently similar colors. Segmentation can also be made by snakes (active contours) where the user draws an approximate contour of the object that is aligned automatically with nearby image edges.

MARS

MARS uses relevance feedback to learn similarity measures. The system has access to a variety of features and similarity measures and learns the best ones for a particular query by letting the user grade retrieved images as highly relevant, relevant, no-opinion, nonrelevant, or highly nonrelevant. MARS uses image features such as color, texture, shape and wavelet coefficients. MARS calculates color histogram and color moments represented in the HSV color space. It uses co-occurrence matrices in different directions to extract textures such as coarseness, contrast, directionality, and inverse difference moments. It also uses Fourier transform and wavelet coefficients to characterize images. The information in each type of features (e.g., color) is represented by a set of subfeatures (e.g., color histogram or color moments). The subfeatures and similarity measures are normalized. Users from various disciplines have been invited to compare the performance between the relevance feedback approach in MARS and the computer centric approach where the user specifies the relative feature weights. Almost all users have rated the relevance feedback approach much higher than the computer-centric approach in terms of capturing their perception subjectivity and information needed.

VisualSEEK/WebSEEK

VisualSEEK is an image database system that integrates feature-based image indexing with spatial query methods. Because global features such as color histograms lack spatial information, VisualSEEK uses salient image regions and their colors, sizes, spatial locations, and relationships, to describe images. The integration of global features and spatial locations relies on the representation of color regions by color sets. Because color sets can be indexed for retrieval of similar color sets, unconstrained images are decomposed into nearly symbolic images that lend to efficient spatial query.

VisualSEEK quantizes the HSV color space into 166 regions. Quantized images are then filtered morphologically and analyzed to reveal spatially localized color regions. A region is represented by the dominant colors in the region. The color distribution in a region is represented by a color set, which is a 166-dimensional binary vector approximating the color histogram of the region. The similarity between two color sets, c_1 and c_2, is given by $(c_1 - c_2)^T A(c_1 - c_2)$, where elements of the matrix A denote the similarity between colors. In addition, VisualSEEK uses features such as centroid and minimum bounding box of each region. Measurements such as distances between regions and relative spatial locations are also used in the query.

VisualSEEK allows the user to sketch spatial arrangements of color regions, position them on the query grid, and assign them properties of color, size, and absolute location. The system then finds the images that contain the most similar arrangements of regions. The system automatically extracts and indexes salient color regions from the images. By using efficient indexing techniques on color, region sizes, and both absolute and relative spatial locations, a wide variety of color/spatial visual queries can be computed.

WebSEEK searches the World Wide Web for images and video by looking for file extensions such as .jpg, .mov, .gif, and .mpeg. The system extracts keywords from the URL and hyperlinks. It creates a histogram on the keywords to find the most frequent ones, which are put into classes and subclasses. An image can be put into more than one classes. An image taxonomy is constructed in a semiautomatic way using text features (such as associated html tags, captions, and articles) and visual features (such as color, texture, and spatial layout) in the same way as they are used in VisualSEEK. The user can search for images by walking through the taxonomy. The system displays a selection of images, which the user can search by using similarity of color histograms. The histogram can be modified either manually or through relevance feedback.

BIBLIOGRAPHIC NOTES

Comprehensive surveys exist on the topics of content-based image retrieval, which include surveys by Aigrain et al. (43), Rui et al. (44), Yoshitaka and Ichikawa (45), Smeulders et al. (16), and recently, Datta et al. (46,47). Multimedia information retrieval as a broader research area covering video, audio, image, and text analysis has been surveyed extensively by Sebe et al. (48), Snoek and Worring (32), and Lew et al. (49). Surveys also exist on closely related topics such as indexing methods for visual data by Marsicoi et al. (50), face recognition by Zhao et al. (51), applications of content-based retrieval to medicine by Müller et al. (52), and applications to cultural and historical imaging by Chen et al. (53).

Image feature extraction is of ultimate importance to content-based retrieval. Comparisons of different image measures for content-based image retrieval are given in Refs. (16,17 and 54).

Example visual database systems that support content-based retrieval include IBM QBIC (6), Virage (55), and NEC AMORE (56) in the commercial domain and MIT Photobook (57), UIUC MARS (34,58), Columbia VisualSEEK (42), and UCSB NeTra (59) in the academic domain. Practical issues such as system implementation and architecture, their limitations and how to overcome them, intuitive result visualization, and system evaluation were discussed by Smeulders et al. in Ref. (16). Many of these systems such as AMORE or their extensions such as UIUC WebMARS (60) and Columbia WebSEEK (19) support image search on the World Wide Web. Webseer (18) and ImageScape (61) are systems designed for Web-based image retrieval. Kherfi et al. (62) provides a survey of these Web-based image retrieval systems.

Interested readers may refer to Refs. (16,17, 49 and 63) for surveys and technical details of bridging the semantic gap. Important early work that introduced relevance feedback into image retrieval included Ref. (34), which was implemented in the MARS system (58). A review of techniques for relevance feedback is given by Zhou and Huang (64).

Readers who are interested in the quadtree and related hierarchical data structures may consult the survey (65) by Samet. An encyclopedic description of high-dimensional metric data structures was published recently as a book (37). Bohm et al. (66) give a review of high-dimensional indexing techniques of multimedia data.

Relational database management systems have been studied extensively for decades and have resulted in numerous efficient optimization and implementation techniques. For concepts and techniques in relational database management, readers may consult popular database textbooks, such as Ref. (40). Common machine learning and data mining algorithms for data clustering and classification can be found in data mining textbooks, such as Ref. (61). Image feature extraction depends on image processing algorithms and techniques, which are well explained in image processing textbooks, such as Refs. (22 and 23).

BIBLIOGRAPHY

1. T. Kunii, S. Weyl, and J. M. Tenenbaum, A relational database scheme for describing complex picture with color and texture, *Proc. 2nd Internat. Joint Conf. Pattern Recognition*, Lyngby-Coperhagen, Denmark, 1974, pp. 73–89.
2. S. K. Chang and K. S. Fu, (eds.), *Pictorial Information Systems, Lecture Notes in Computer Science 80*, Berlin: Springer-Verlag, 1980.
3. S. K. Chang and T. L. Kunii, Pictorial database systems, *IEEE Computer*, **14** (11): 13–21, 1981.
4. S. K. Chang, (ed.), Special issue on pictorial database systems, *IEEE Computer*, **14** (11): 1981.
5. H. Tamura and N. Yokoya, Image database systems: a survey, *Pattern Recognition*, **17** (1): 29–43, 1984.
6. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, Query by image and video content: the QBIC system, *IEEE Computer*, **28** (9): 23–32, 1995.
7. N. Roussopoulos, C. Faloutsos, and T. Sellis, An efficient pictorial database system for PSQL, *IEEE Trans. Softw. Eng.*, **14** (5): 639–650, 1988.
8. M. M. Zloof, Query-by-example: a database language, *IBM Syst. J.*, **16** (4): 324–343, 1977.
9. N. S. Chang and K. S. Fu, Query-by-pictorial example, *IEEE Trans. Softw. Eng.*, **6** (6): 519–524, 1980.
10. T. Joseph and A. F. Cardenas, PICQUERY: a high level query language for pictorial database management, *IEEE Trans. Softw. Eng.*, **14** (5): 630–638, 1988.
11. L. Yang and J. K. Wu, Towards a semantic image database system, *Data and Knowledge Enginee*, **22** (2): 207–227, 1997.
12. K. Yamaguchi and T. L. Kunii, PICCOLO logic for a picture database computer and its implementation, *IEEE Trans. Comput.*, **C-31** (10): 983–996, 1982.
13. J. A. Orenstein and F. A. Manola, PROBE spatial data modeling and query processing in an image database application, *IEEE Trans. Softw. Eng.*, **14** (5): 611–629, 1988.
14. S. K. Chang, C. W. Yan, D. C. Dimitroff, and T. Arndt, An intelligent image database system, *IEEE Trans. Softw. Eng.*, **14** (5): 681–688, 1988.
15. A. Brink, S. Marcus, and V. Subrahmanian, Heterogeneous multimedia reasoning, *IEEE Computer*, **28** (9): 33–39, 1995.
16. A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, Content-based image retrieval at the end of the early years, *IEEE Trans. Patt. Anal. Machine Intell.*, **22** (12): 1349–1380, 2000.
17. Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma, A survey of content-based image retrieval with high-level semantics, *Pattern Recognition*, **40** (1): 262–282, 2007.
18. C. Frankel, M. J. Swain, and V. Athitsos, Webseer: an image search engine for the world wide web, Technical Report TR-96-14, Chicago, Ill.: University of Chicago, 1996.
19. J. R. Smith and S.-F. Chang, Visually searching the web for content, *IEEE Multimedia*, **4** (3): 12–20, 1997.
20. A. B. Benitez, M. Beigi, and S.-F. Chang, Using relevance feedback in content-based image metasearch, *IEEE Internet Computing*, **2** (4): 59–69, 1998.
21. A. F. Smeaton, P. Over, and W. Kraaij, Evaluation campaigns and TRECVid, *MIR'06: Proc. 8th ACM Internat. Workshop on Multimedia Information Retrieval*, Santa Barbara, California, 2006, pp. 321–330.
22. J. C. Russ, The Image Processing Handbook, 5th Ed. Boca Raton, FL: CRC Press, 2006.
23. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd Ed. Englewood Cliffs, NJ: Prentice Hall, 2007.
24. W.-Y Ma and B. S. Manjunath, A texture thesaurus for browsing large aerial photographs, *J. Am. Soc. Informa. Sci*. **49** (7): 633–648, 1998.
25. M. Turk and A. Pentland, Eigen faces for recognition, *J. Cognitive Neuro-science*, **3** (1): 71–86, 1991.
26. J. B. Tenenbaum, V. deSilva, and J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, *Science*, **290**: 2319–2323, 2000.
27. S. T. Roweis and L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, *Science*, **290**: 2323–2326, 2000.
28. L. Yang, Alignment of overlapping locally scaled patches for multidimensional scaling and dimensionality reduction, *IEEE Trans. Pattern Analysis Machine Intell.*, In press.
29. A. D. Bimbo and P. Pala, Content-based retrieval of 3D models, *ACM Trans. Multimedia Computing, Communicat. Applicat.*, **2** (1): 20–43, 2006.
30. H. Du and H. Qin, Medial axis extraction and shape manipulation of solid objects using parabolic PDEs, *Proc. 9th ACM Symp. Solid Modeling and Applicat.*, Genoa, Italy, 2004, pp. 25–35.
31. M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii, Topology matching for fully automatic similarity estimation of 3D shapes, *Proc. ACM SIGGRAPH 2001*, Los Angeles, CA, 2001, pp. 203–212.
32. C. G. Snoek and M. Worring, Multimodal video indexing: a review of the state-of-the-art, *Multimedia Tools and Applicat.*, **25** (1): 5–35, 2005.
33. J.-K. Wu, Content-based indexing of multimedia databases, *IEEE Trans. Knowledge Data Engineering*, **9** (6): 978–989, 1997.
34. Y. Rui, T. S. Huang, M. Ortega, and S. Mehrotra, Relevance feedback: A power tool in interactive content-based image retrieval, *IEEE Trans. Circuits Systems Video Technol.*, **8** (5): 644–655, 1998.
35. M. Kobayashi and K. Takeda, Information retrieval on the web, *ACM Comput. Surv.*, **32** (2): 144–173, 2000.
36. Y. Chen and J. Z. Wang, A region-based fuzzy feature matching approach to content-based image retrieval, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **24** (9): 1252–1267, 2002.
37. H. Samet, *Foundations of Multidimensional and Metric Data Structures*, San Francisco, CA: Morgan Kaufmann, 2006.

38. J. B. MacQueen, Some methods for classification and analysis of multivariate observations, *Proc. 5th Berkeley Symp. Mathematical Statistics and Probability*, Berkeley, CA, 1967, pp. 281–297.
39. T. Kohonen, *Self-Organizating Maps*, 3rd Ed. Berlin: Springer, 2000.
40. R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd Ed. New York: McGraw-Hill, 2002.
41. S. Chaudhuri and L. Gravano, Optimizing queries over multimedia repositories, *Proc. 1996 ACM SIGMOD Internat. Conf. Management of Data (SIGMOD'96)*, Montreal, Quebec, Canada, 1996, pp. 91–102.
42. J. R. Smith and S.-F. Chang, VisualSEEk: A fully automated content-based image query system, *Proc. 4th ACM Internat. Conf. Multimedia*, Boston, MA, 1996, pp. 87–98.
43. P. Aigrain, H. Zhang, and D. Petkovic, Content-based representation and retrieval of visual media: A state-of-the-art review, *Multimedia Tools and Applicat.*, **3** (3): 179–202, 1996.
44. Y. Rui, T. S. Huang, and S.-F. Chang, Image retrieval: current techniques, promising directions and open issues, *J. Visual Communicat. Image Represent.*, **10** (1): 39–62, 1999.
45. A. Yoshitaka and T. Ichikawa, A survey on content-based retrieval for multimedia databases, *IEEE Trans. Knowledge and Data Engineering*, **11** (1): 81–93, 1999.
46. R. Datta, D. Joshi, J. Li, and J. Z. Wang, Image retrieval: Ideas, influences, and trends of the new age, *ACM Comput. Surv.* In press.
47. R. Datta, J. Li, and J. Z. Wang, Content-based image retrieval: approaches and trends of the new age, *Proc. 7th ACM SIGMM Internat. Workshop on Multimedia Information Retrieval (MIR 2005)*, 2005, pp. 253–262.
48. N. Sebe, M. S. Lew, X. S. Zhou, T. S. Huang, and E. M. Bakker, The state of the art in image and video retrieval, *Proc. 2nd Internat. Conf. Image and Video Retrieval*, Lecture Notes in Computer Science 2728, Urbana-Champaign, IL, 2003. pp. 1–8.
49. M. S. Lew, N. Sebe, C. Djeraba, and R. Jain, Content-based multimedia information retrieval: State of the art and challenges, *ACM Trans. Multimedia Computing, Communicat. Applicat.*, **2** (1): 1–19, 2006.
50. M. D. Marsicoi, L. Cinque, and S. Levialdi, Indexing pictorial documents by their content: A survey of current techniques, *Image and Vision Computing*, **15** (2): 119–141, 1997.
51. W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, Face recognition: a literature survey, *ACM Comput. Surv.*, **35** (4): 399–458, 2003.
52. H. Müller, N. Michoux, D. Bandon, and A. Geissbuhler, A review of content-based image retrieval systems in medical applications - clinical benefits and future directions, *Internat. J. Medical Informatics*, **73** (1): 1–23, 2004.
53. C.-C. Chen, H. D. Wactlar, J. Z. Wang, and K. Kiernan, Digital imagery for significant cultural and historical materials - an emerging research field bridging people, culture, and technologies, *Internat. J. Digital Libraries*, **5** (4): 275–286, 2005.
54. B. M. Mehtre, M. S. Kankanhalli, and W. F. Lee, Shape measures for content based image retrieval: a comparison, *Information Processing and Management*, **33** (3): 319–337, 1997.
55. A. Gupta and R. Jain, Visual information retrieval, *Commun. ACM*, **40** (5): 70–79, 1997.
56. S. Mukherjea, K. Hirata, and Y. Hara, Amore: a world wide web image retrieval engine, *World Wide Web*, **2** (3): 115–132, 1999.
57. A. Pentland, R. W. Picard, and S. Sclaroff, Photobook: Tools for content-based manipulation of image databases. Technical Report TR-255, MIT Media Lab, 1996.
58. M. Ortega, Y. Rui, K. Chakrabarti, K. Porkaew, S. Mehrotra, and T. S. Huang, Supporting ranked boolean similarity queries in MARS, *IEEE Trans. Knowledge Data Engineering*, **10** (6): 905–925, 1998.
59. W.-Y. Ma and B. S. Manjunath, Netra: a toolbox for navigating large image databases, *Multimedia Systems*, **7** (3): 184–198, 1999.
60. M. Ortega-Binderberger, S. Mehrotra, K. Chakrabarti, and K. Porkaew, WebMARS: a multimedia search engine. Technical Report TR-DB-00-01, Information and Computer Science, University of California at Irvine, 2000.
61. M. S. Lew, Next-generation web searches for visual content, *IEEE Computer*, **33** (11): 46–53, 2000.
62. M. L. Kherfi, D. Ziou, and A. Bernardi, Image retrieval from the world wide web: Issues, techniques, and systems, *ACM Comput. Surv.*, **36** (1): 35–67, 2004.
63. N. Vasconcelos, From pixels to semantic spaces: advances in content-based image retrieval, *IEEE Computer*, **40** (7): 20–26, 2007.
64. X. S. Zhou and T. S. Huang, Relevance feedback in image retrieval: a comprehensive review, *Multimedia Systems*, **8** (6): 536–544, 2003.
65. H. Samet, The quadtree and related hierarchical data structures, *ACM Comput. Surv.*, **16** (2): 187–260, 1984.
66. C. B—hm, S. Berchtold, and D. A. Keim, Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases, *ACM Comput. Surv.*, **33** (3): 322–373, 2001.
67. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd Ed. San Francisco, CA: Morgan Kaufmann, 2005.

Li Yang
Western Michigan University
Kalamazoo, Michigan
Tosiyasu L. Kunii
Kanazawa Institute of
Technology
Tokyo, Japan

A

ALGEBRAIC GEOMETRY

INTRODUCTION

Algebraic geometry is the mathematical study of geometric objects by means of algebra. Its origins go back to the coordinate geometry introduced by Descartes. A classic example is the circle of radius 1 in the plane, which is the geometric object defined by the algebraic equation $x^2 + y^2 = 1$. This generalizes to the idea of a systems of polynomial equations in many variables. The solution sets of systems of equations are called *varieties* and are the geometric objects to be studied, whereas the equations and their consequences are the algebraic objects of interest.

In the twentieth century, algebraic geometry became much more abstract, with the emergence of *commutative algebra* (rings, ideals, and modules) and *homological algebra* (functors, sheaves, and cohomology) as the foundational language of the subject. This abstract trend culminated in Grothendieck's *scheme theory*, which includes not only varieties but also large parts of algebraic number theory. The result is a subject of great power and scope—Wiles' proof of Fermat's Last Theorem makes essential use of schemes and their properties. At the same time, this abstraction made it difficult for beginners to learn algebraic geometry. Classic introductions include Refs. 1 and 2, both of which require a considerable mathematical background.

As the abstract theory of algebraic geometry was being developed in the middle of the twentieth century, a parallel development was taking place concerning the algorithmic aspects of the subject. Buchberger's theory of *Gröbner bases* showed how to manipulate systems of equations systematically, so (for example) one can determine algorithmically whether two systems of equations have the same solutions over the complex numbers. Applications of Gröbner bases are described in Buchberger's classic paper [3] and now include areas such as computer graphics, computer vision, geometric modeling, geometric theorem proving, optimization, control theory, communications, statistics, biology, robotics, coding theory, and cryptography.

Gröbner basis algorithms, combined with the emergence of powerful computers and the development of *computer algebra* (see SYMBOLIC COMPUTATION), have led to different approaches to algebraic geometry. There are now several accessible introductions to the subject, including Refs. 4–6.

In practice, most algebraic geometry is done over a field, and the most commonly used fields are as follows:

- The rational numbers $\mathbf{Q}$ used in symbolic computation.
- The real numbers $\mathbf{R}$ used in geometric applications.
- The complex numbers $\mathbf{C}$ used in many theoretical situations.
- The finite field $\mathbf{F}_q$ with $q = p^m$ elements (p prime) used in cryptography and coding theory.

In what follows, k will denote a field, which for concreteness can be taken to be one of the above. We now explore the two main flavors of algebraic geometry: *affine* and *projective*.

AFFINE ALGEBRAIC GEOMETRY

Given a field k, we have *n-dimensional affine space* k^n, which consists of all n-tuples of elements of k. In some books, k^n is denoted $\mathrm{A}^n(k)$. The corresponding algebraic object is the *polynomial ring* $k[\mathrm{x}_1, \ldots, x_n]$ consisting of all polynomials in variables $x_1, \ldots, x_n$ with coefficients in k. By *polynomial*, we mean a finite sum of terms, each of which is an element of k multiplied by a *monomial*

$$x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$$

where $a_1, \ldots, a_n$ are non-negative integers. Polynomials can be added and multiplied, and these operations are commutative, associative, and distibutive. This is why $k[x_1, \ldots, x_n]$ is called a *commutative ring*.

Given polynomials $f_1, \ldots, f_s$ in $k[x_1, \ldots, x_n]$, the *affine variety* $\mathbf{V}(f_1, \ldots, f_s)$ consists of all points $(u_1, \ldots, u_n)$ in k^n that satisfy the system of equations

$$f_1(u_1, \ldots, u_n) = \cdots = f_s(u_1, \ldots, u_n) = 0.$$

Some books (such as Ref. 1) call $\mathbf{V}(f_1, \ldots, f_s)$ an *affine algebraic set*.

The algebraic object corresponding to an affine variety is called an *ideal*. These arise naturally from a system of equations $f_1 = \cdots = f_s = 0$ as follows. Multiply the first equation by a polynomial h_1, the second by h_2, and so on. This gives the equation

$$h = h_1 f_1 + \cdots + h_s f_s = 0,$$

which is called a *polynomial consequence* of the original system. Note that $h(u_1, \ldots, u_n) = 0$ for every $(u_1, \ldots, u_n)$ in $\mathbf{V}(f_1, \ldots, f_s)$. The *ideal* $\langle f_1, \ldots, f_s \rangle$ consists of all polynomial consequences of the system $f_1 = \cdots = f_s = 0$. Thus, elements of $\langle f_1, \ldots, f_s \rangle$ are linear combinations of $f_1, \ldots, f_s$, where the coefficients are allowed to be arbitrary polynomials.

A general definition of ideal applies to any commutative ring. The *Hilbert Basis Theorem* states that all ideals in a polynomial ring are of the form $\langle f_1, \ldots, f_s \rangle$. We say that $f_1, \ldots, f_s$ is a *basis* of $\langle f_1, \ldots, f_s \rangle$ and that $\langle f_1, \ldots, f_s \rangle$ is *generated* by $f_1, \ldots, f_s$. This notion of "basis" differs from how the term is used in linear algebra because linear independence fails. For example, x, y is a basis of the ideal $\langle x, y \rangle$ in $k[x, y]$, even though $y \cdot x + (-x) \cdot y = 0$.

A key result is that if $\mathbf{V}(f_1, \ldots, f_s) = \mathbf{V}\{g_1, \ldots, g_t)$ whenever $\langle f_1, \ldots, f_s \rangle = \langle g_1, \ldots, g_t \rangle$. This is useful in practice because switching to a different basis may make it easier to understand the solutions of the equations. From the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

theoretical point of view, this shows that an affine variety depends on the ideal I generated by the defining equations, so that the affine variety can be denoted $\mathbf{V}(I)$. Thus, every ideal gives an affine variety.

We can also reverse this process. Given an affine variety V, let $\mathbf{I}(V)$ consist of all polynomials that vanish on all points of V. This satisfies the abstract definition of ideal. Thus, every affine variety gives an ideal, and one can show that we always have

$$V = \mathbf{V}(\mathbf{I}(V)).$$

However, the reverse equality may fail. In other words, there are ideals I such that

$$I \neq \mathbf{I}(\mathbf{V}(I)).$$

An easy example is provided by $I = \langle x^2 \rangle$ in $k\,|\,x\,|$, because $\mathbf{I}(\mathbf{V}(I)) = \langle x \rangle \neq I$. Hence, the correspondence between ideals and affine varieties is not a perfect match. Over the complex numbers, we will see below that there is nevertheless a nice relation between I and $\mathbf{I}(\mathbf{V}(I))$.

One can prove that the *union* and *intersection* of affine varieties are again affine varieties. In fact, given ideals I and J, one has

$$\begin{aligned} V(I) \cup V(J) &= V(I \cap J) \\ V(I) \cap V(J) &= V(I + J), \end{aligned}$$

where

$$\begin{aligned} I \cap J &= \{g \mid g \text{ is in both } I \text{ and } J\} \\ I + J &= \{g + h \mid g \text{ is in } I \text{ and } h \text{ is in } J\} \end{aligned}$$

are the *intersection* and *sum* of I and J (note that $I \cap J$ and $I + J$ are analgous to the intersection and sum of subspaces of a vector space). In this way, algebraic operations on ideals correspond to geometric operations on varieties. This is part of the *ideal-variety correspondence* explained in Chapter 4 of Ref. 4.

Sometimes an affine variety can be written as a union of strictly smaller affine varieties. For example,

$$\mathbf{V}((x - y)(x^2 + y^2 - 1)) = \mathbf{V}(x - y) \cup \mathbf{V}(x^2 + y^2 - 1)$$

expresses the affine variety $\mathbf{V}((x - y)(x^2 + y^2 - 1))$ as the union of the line $y = x$ and the unit circle (Fig. 1).

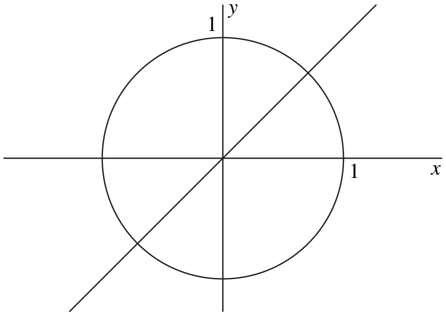


Figure 1. Union of a line and a circle.

In general, an affine variety is *irreducible* if it has no such decomposition. In books such as Ref. 1, varieties are always assumed to be irreducible.

One can show that every affine variety V can be written as

$$V = V_1 \cup \cdots \cup V_m$$

where each V_i is irreducible and no V_i is contained in V_j for $j \neq i$. We say that the V_i are the *irreducible components* of V. Thus, irreducible varieties are the "building blocks" out of which all varieties are built. Algebraically, the above decomposition means that the ideal of V can be written as

$$\mathbf{I}(V) = P_1 \cap \cdots \cap P_m$$

where each P_i is *prime* (meaning that if a product ab lies in P_i, then so does a or b) and no P_i contains P_j for $j \neq i$. This again illustrates the close connection between the algebra and geometry. (For arbitrary ideals, things are a bit more complicated: The above intersection of prime ideals has to be replaced with what is called a *primary decomposition*—see Chapter 4 of Ref. 4).

Every variety has a *dimension*. Over the real numbers $\mathbf{R}$, this corresponds to our geometric intuition. But over the complex numbers $\mathbf{C}$, one needs to be careful. The affine space $\mathbf{C}^2$ has dimension 2, even though it looks four-dimensional from the real point of view. The dimension of a variety is the maximum of the dimensions of its irreducible components, and irreducible affine varieties of dimensions 1, 2, and 3 are called curves, surfaces, and 3-folds, respectively. An affine variety in k^n is called a *hypersurface* if every irreducible component has dimension $n - 1$.

PROJECTIVE ALGEBRAIC GEOMETRY

One problem with affine varieties is that intersections sometimes occur "at infinity." An example is given by the intersection of a hyperbola with one of its asymptotes in Fig. 2. (Note that a line has a single point at infinity.)

Points at infinity occur naturally in computer graphics, where the horizon in a perspective drawing is the "line at infinity" where parallel lines meet. Adding points at infinity to affine space leads to the concept of projective space.

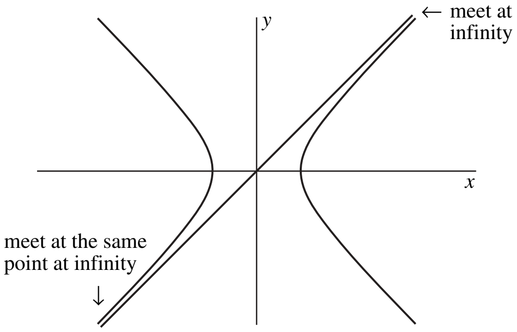


Figure 2. A hyperbola and one of its asymptotes.

The most common way to define *n-dimensional projective space* $\mathbf{P}^n(k)$ is via *homogeneous coordinates*. Every point in $\mathbf{P}^n(k)$ has homogeneous coordinates $[u_0, \ldots, u_n]$, where $(u_0, \ldots, u_n)$ is an element of k^{n+1} different from the zero element $(0, \ldots, 0)$. The square brackets in $[u_0, \ldots, u_n]$ indicate that homogeneous coordinates are *not* unique; rather,

$$[u_0, \ldots, u_n] = [v_0, \ldots, v_n]$$

if and only if there is a nonzero λ in k such that $\lambda u_i = v_i$ for $i = 0, \ldots, n$, i.e., $\lambda(u_0, \ldots, u_n) = (v_0, \ldots, v_n)$. This means that two nonzero points in k^{n+1} give the same point in $\mathbf{P}^n(k)$ if and only if they lie on the same line through the origin.

Consider those points in $P^n(k)$ where $u_0 \neq 0$. As $(1/u_0)$ $(u_0, u_1, \ldots, u_n) = (1, u_1/u_0, \ldots, u_n/u_n)$, one sees easily that

$$\mathbf{P}^n(k) = k^n \cup \mathbf{P}^{n-1}(k).$$

We call $\mathbf{P}^{n-1}$ (k) the *hyperplane at infinity* in this situation. One virtue of homogeneous coordinates is that they have a rich supply of coordinate changes. For example, an invertible 4 × 4 matrix with real entries gives an invertible transformation from $\mathbf{P}^3(\mathbf{R})$ to itself. The reason you see 4 × 4 matrices in computer graphics is that you are really working in three-dimensional projective space $\mathbf{P}^3(\mathbf{R})$, although this is rarely mentioned explicitly. See **THREE-DIMENSIONAL GRAPHICS**.

Now that we have $\mathbf{P}^n(k)$, we can define projective varieties as follows. A polynomial F in $k[x_0, \ldots, x_n]$ is *homogeneous of degree d* if every monomial $x_0^{a_0} \ldots x_n^{a_n}$ appearing in F has degree d, i.e., $a_0 + \cdots + a_n = d$. Such a polynomial has the property that

$$F(\lambda x_0, \ldots, \lambda x_n) = \lambda^d F(x_0, \ldots, x_n).$$

For a point $[u_0, \ldots, u_n]$ of $\mathbf{P}^n(k)$, the quantity $F(u_0, \ldots, u_n)$ is not well defined because of the ambiguity of homogeneous coordinates. But when F is homogeneous, the equation $F(u_0, \ldots, u_n) = 0$ is well defined. Then, given homogeneous polynomials $F_1, \ldots, F_s$, the *projective variety* $\mathbf{V}(F_1, \ldots, F_s)$ consists of all points $[u_0, \ldots, u_n]$ in $\mathbf{P}^n(k)$ that satisfy the system of equations

$$F_1(u_1, \ldots, u_n) = \cdots = F_s(u_1, \ldots, u_n) = 0.$$

Some books (such as Ref. 1) call $\mathbf{V}(F_1, \ldots, F_s)$ a *projective algebraic set*.

The algebraic object corresponding to $\mathbf{P}^n(k)$ is the polynomial ring $k[x_0, \ldots, x_n]$, which we now regard as a *graded ring*. This means that by grouping together terms of the same degree, every polynomial f of degree d can be uniquely written as

$$f = f_0 + f_1 + \cdots + f_d,$$

where f_i is homogeneous of degree i (note that f_i may be zero). We call the f_i the *homogeneous components* of f. An ideal I is *homogeneous* if it is generated by homogeneous polynomials. If I is homogeneous, then a polynomial lies in I if and only if its homogeneous components lie in I.

Most concepts introduced in the affine context carry over to the projective setting. Thus, we can ask whether a projective variety is irreducible and what is its dimension. We also have a projective version of the ideal-variety correspondence, where homogeneous ideals correspond to projective varieties. This is a bit more sophisticated than the affine case, in part because the ideal $\langle x_0, \ldots, x_n \rangle$ defines the empty variety because homogeneous coordinates are not allowed to all be zero.

Given a projective variety V in $\mathbf{P}^n(k)$, we get a homogeneous ideal $I = \mathbf{I}(V)$ in $k[x_0, \ldots, x_n]$. Let I_d consist of all homogeneous polynomials of degree d that lie in I. Then I_d is a finite-dimensional vector space over k, and by a theorem of Hilbert, there is a polynomial $P(x)$, called the *Hilbert polynomial*, such that for all sufficiently large integers d sufficiently large, we have

$$\binom{n+d}{n} - \dim_k I_d = P(d),$$

where the binomial coefficient $\binom{n+d}{n}$ is the dimension of the space of all homogeneous polynomials of degree n. Then one can prove that the dimension m of V equals the degree of $P(x)$. Furthermore, if we write the Hilbert polynomial $P(x)$ in the form

$$P(x) = \frac{D}{m!}x^m + \text{terms of lower degree},$$

then D is a positive integer called the *degree* of V. For example, when V is defined by $F = 0$ over the complex numbers, where F is irreducible and homogeneous of degree d, then V has degree d according to the above definition. This shows just how much information is packed into the ideal I. Later we will discuss the algorithmic methods for computing Hilbert polynomials.

THE COMPLEX NUMBERS

Although many applications of algebraic geometry work over the real numbers $\mathbf{R}$, the theory works best over the complex numbers $\mathbf{C}$. For instance, suppose that $V = \mathbf{V}(f_1, \ldots, f_s)$ is a variety in $\mathbf{R}^n$ of dimension d. Then we expect V to be defined by at least $n - d$ equations because (roughly speaking) each equation should lower the dimension by one. But if we set $f = f_1^2 + \cdots + f_s^2$, then $f = 0$ is equivalent to $f_1 = \cdots = f_s = 0$ because we are working over $\mathbf{R}$. Thus, $V = \mathbf{V}(f_1, \ldots, f_s)$ can be defined by one equation, namely $f = 0$. In general, the relation between ideals and varieties is complicated when working over $\mathbf{R}$.

As an example of why things are nicer over $\mathbf{C}$, consider an ideal I in $\mathbf{C}[x_1, \ldots, x_n]$ and let $V = \mathbf{V}(I)$ be the corresponding affine variety in $\mathbf{C}^n$. The polynomials in I clearly vanish on V, but there may be others. For example, suppose that f is not in I but some power of f, say f^ℓ, is in I. Then f^ℓ and hence f vanish on V. The *Hilbert Nullstellensatz* states that these are the only polynomials that vanish on V, i.e.,

$$\begin{aligned}\mathbf{I}(V) &= \mathbf{I}(\mathbf{V}(I)) \\ &= \{\, f \text{ in } \mathbf{C}[x_1, \ldots, x_n] \,|\, f^\ell \text{ is in } I \text{ for some integer } \ell \geq 0\}.\end{aligned}$$

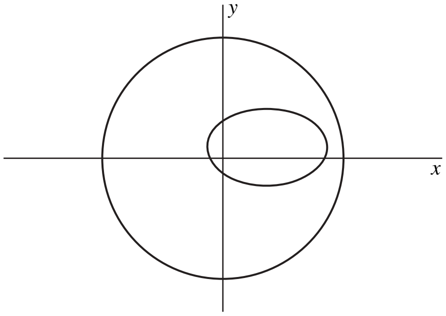


Figure 3. A circle and an ellipse.

The ideal on the right is called the *radical* of I and is denoted $\mathrm{rad}(I)$. Thus, the Nullstellensatz asserts that over $\mathbf{C}$, we have $\mathbf{I}(\mathbf{V}(I)) = \mathrm{rad}(I)$. It is easy to find examples where this fails over $\mathbf{R}$.

Another example of why $\mathbf{C}$ is nice comes from *Bézout's Theorem* in Fig. 3. In its simplest form, this asserts that distinct irreducible plane curves of degrees m and n intersect in mn points, counted with multiplicity. For example, consider the intersection of a circle and an ellipse. These are curves of degree 2, so we should have four points of intersection. But if the ellipse is really small, it can fit entirely inside the circle, which makes it seem that there are no points of intersection, as in Fig. 3. This is because we are working over $\mathbf{R}$; over $\mathbf{C}$, there really are four points of intersection.

Bézout's Theorem also illustrates the necessity of working over the projective plane. Consider, for example, the intersection of a hyperbola and one of its asymptotes in Fig. 4. These are curves of degree 2 and 1, respectively, so there should be 2 points of intersection. Yet there are none in $\mathbf{R}^2$ or $\mathbf{C}^2$. But once we go to $\mathbf{P}^2(\mathbf{R})$ or $\mathbf{P}^2(\mathbf{C})$, we get one point of intersection at infinity, which has multiplicity 2 because the asymptote and the hyperbola are tangent at infinity. We will say more about multiplicity later in the article.

In both the Nullstellensatz and Bézout's theorem, we can replace $\mathbf{C}$ with any *algebraically closed field*, meaning a field where every nonconstant polynomial has a root. A large part of algebraic geometry involves the study of irreducible projective varieties over algebraically closed fields.

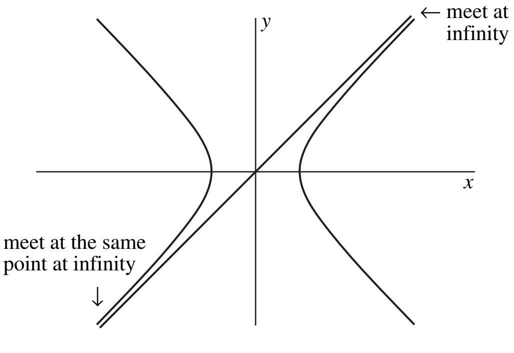


Figure 4. A hyperbola and one of its asymptotes.

FUNCTIONS ON AFFINE AND PROJECTIVE VARIETIES

In mathematics, one often studies objects by considering the functions defined on them. For an affine variety V in k^n, we let $k[V]$ denote the set of functions from V to k given by polynomials in $k[x_1,\dots,x_n]$. One sees easily that $k[V]$ is a ring, called the *coordinate ring* of V.

An important observation is that two distinct polynomials f and g in $k[x_1,\dots,x_n]$ can give the same function on V. This happens precisely when $f-g$ vanishes on V, i.e., when $f-g$ is in the ideal $\mathbf{I}(V)$. We express this by writing

$$f \equiv g \bmod \mathbf{I}(V),$$

similar to the congruence notation introduced by Gauss. It follows that computations in $k[x_1,\dots,x_n]$ modulo $\mathbf{I}(V)$ are equivalent to computations in $k[V]$. In the language of abstract algebra, this is expressed by the ring isomorphism

$$k[x_1,\dots,x_n]/\mathbf{I}(V) \simeq k[V],$$

where $k[x_1,\dots,x_n]/\mathbf{I}(V)$ is the set of equivalence classes of the equivalence relation $f \equiv g$ mod $\mathbf{I}(V)$. More generally, given any ideal I in $k[x_1,\dots,x_n]$, one gets the *quotient ring* $k[x_1,\dots,\mathrm{x}_\mathrm{n}]/I$ coming from the equivalence relation $f \equiv g$ mod I. We will see later that Gröbner bases enable us to compute effectively in quotient rings.

We can use quotients to construct finite fields as follows. For a prime p, we get $\mathbf{F}_p$ by considering the integers modulo p. To get $\mathbf{F}_{p^m}$ when $m > 1$, take an irreducible polynomial f in $\mathbf{F}_p[x]$ of degree m. Then the quotient ring $\mathbf{F}_p[x]/\langle f\rangle$ is a model of F_{p^m}. Thus, for example, computations in $\mathbf{F}_2[x]$ modulo x^2+x+1 represent the finite field $\mathbf{F}_4$. See ALGEBRAIC CODING THEORY for more on finite fields.

The coordinate ring $\mathbf{C}[V]$ of an affine variety V in $\mathbf{C}^n$ has an especially strong connection to V. Given a point $(u_1,\dots,u_n)$ of V, the functions in $\mathbf{C}[V]$ vanishing at $(u_1,\dots,u_n)$ generate a *maximal ideal*, meaning an ideal of $\mathbf{C}[V]$ not equal to the whole ring but otherwise as big as possible with respect to inclusion. Using the Nullstellensatz, one can show that *all* maximal ideals of $\mathbf{C}[V]$ arise this way. In other words, there is a one-to-one correspondence

$$\text{points of } V \longleftrightarrow \text{maximal ideals of } \mathbf{C}[V].$$

Later we will use this correspondence to motivate the definition of *affine scheme*.

Functions on projective varieties have a different flavor, since a polynomial function defined everywhere on a connected projective variety must be constant. Instead, two approaches are used, which we will illustrate in the case of $\mathbf{P}^n(k)$. In the first approach, one considers *rational functions*, which are quotients

$$\frac{F(x_0,\dots,x_n)}{G(x_0,\dots,x_n)}$$

of homogeneous polynomials of the same degree, say d. This function is well defined despite the ambiguity of homogeneous coordinates, because

$$\frac{F(\lambda x_0,\ldots,\lambda x_n)}{G(\lambda x_0,\ldots,\lambda x_n)}=\frac{\lambda^d F(x_0,\ldots,x_n)}{\lambda^d G(x_0,\ldots,x_n)}=\frac{F(x_0,\ldots,x_n)}{G(x_0,\ldots,x_n)}.$$

However, this function is *not* defined when the denominator vanishes. In other words, the above quotient is only defined where $G \neq 0$. The set of all rational functions on $\mathbf{P}^n(k)$ forms a field called the *field of rational functions on* $\mathbf{P}^n(k)$. More generally, any irreducible projective variety V has a field of rational functions, denoted $k(V)$.

The second approach to studying functions on $\mathbf{P}^n(k)$ is to consider the polynomial functions defined on certain large subsets of $\mathbf{P}^n(k)$. Given projective variety V in $\mathbf{P}^n(k)$, its *complement* U consists of all points of $\mathbf{P}^n(k)$ not in V. We call U a *Zariski open subset* of $\mathbf{P}^n(k)$. Then let $\Gamma(U)$ be the ring of all rational functions on $\mathbf{P}^n(k)$ defined at all points of U. For example, the complement U_0 of $\mathbf{V}(x_0)$ consists of points where $x_0 \neq 0$, which is a copy of k^n. So here $\Gamma(U_0)$ is the polynomial ring $k[x_1/x_0,\ldots,x_n/x_0]$. When we consider the rings $\Gamma(U)$ for all Zariki open subsets U, we get a mathematical object called the *structure sheaf of* $\mathbf{P}^n(k)$. More generally, any projective variety V has a structure sheaf, denoted O_V. We will see below that sheaves play an important role in abstract algebraic geometry.

GRÖBNER BASES

Buchberger introduced Gröbner bases in 1965 in order to do algorithmic computations on ideals in polynomial rings. For example, suppose we are given polynomials f, $f_1,\ldots,f_s \in k[x_1,\ldots,x_n]$, where k is a field whose elements can be represented exactly on a computer (e.g., k is a finite field or the field of rational numbers). From the point of view of pure mathematics, either f lies in the ideal $\langle f_1,\ldots,f_s\rangle$ or it does not. But from a practical point of view, one wants an algorithm for deciding which of these two possibilities actually occurs. This is the *ideal membership question*.

In the special case of two univariate polynomials f, g in $k[x]$, f lies in $\langle g\rangle$ if and only if f is a multiple of g, which we can decide by the division algorithm from high-school algebra. Namely, dividing g into f gives $f = qg + r$, where the remainder r has degree strictly smaller than the degree of g. Then f is a multiple of g if and only if the remainder is zero. This solves the ideal membership question in our special case.

To adapt this strategy to $k[x_1,\ldots,x_n]$, we first need to order the monomials. In $k[x]$, this is obvious: The monomials are $1, x, x^2$, etc. But there are many ways to do this when there are two or more variables. A *monomial order* $>$ is an order relation on monomials $U, V, W,\ldots$ in $k[x_1,\ldots,x_n]$ with the following properties:

1. Given monomials U and V, exactly one of $U > V$, $U = V$, or $U < V$ is true.
2. If $U > V$, then $UW > VW$ for all monomials W.
3. If $U \neq 1$, then $U > 1$; i.e., 1 is the least monomial with respect to $>$.

These properties imply that $>$ is a *well ordering*, meaning that any strictly decreasing sequence with respect to $>$ is finite. This is used to prove termination of various algorithms. An example of a monomial order is *lexicographic order*, where

$$x_1^{a_1}x_2^{a_2}\ldots x_n^{a_n} > x_1^{b_1}x_2^{b_2}\ldots x_n^{b_n}$$

provided

$$a_1 > b_1;\ \text{or}\ a_1 = b_1\ \text{and}\ a_2 > b_2;\ \text{or}\ a_1 = b_1,\\ a_2 = b_2\ \text{and}\ a_3 > b_3;\ \text{etc.}$$

Other important monomial orders are *graded lexicographic order* and *graded reverse lexicographic order*. These are described in Chapter 2 of Ref. 4.

Now fix a monomial order $>$. Given a nonzero polynomial f, we let $\mathrm{lt}(f)$ denote the *leading term* of f, namely the nonzero term of f whose monomial is maximal with respect to $>$ (in the literature, $\mathrm{lt}(f)$ is sometimes called the *initial term* of f, denoted $\mathrm{in}(f)$). Given $f_1,\ldots,f_s$, the *division algorithm* produces polynomials $q_1,\ldots,q_s$ and r such that

$$f = q_1 f_1 + \cdots + q_s f_s + r$$

where every nonzero term of r is divisible by none of $\mathrm{lt}(f_1),\ldots,\mathrm{lt}(f_s)$. The remainder r is sometimes called the *normal form* of f with respect to $f_1,\ldots,f_s$. When $s = 1$ and f and f_1 are univariate, this reduces to the high-school division algorithm mentioned earlier.

In general, multivariate division behaves poorly. To correct this, Buchberger introduced a special kind of basis of an ideal. Given an ideal I and a monomial order, its *ideal of leading terms* $\mathrm{lt}(I)$ (or *initial ideal* $\mathrm{in}(I)$is the ideal generated by $\mathrm{lt}(f)$ for all f in I. Then elements $g_1,\ldots,g_t$ of I form a *Gröbner basis* of I provided that $\mathrm{lt}(g_1),\ldots,\mathrm{lt}(g_t)$ form a basis of $\mathrm{lt}(I)$. Buchberger showed that a Gröbner basis is in fact a basis of I and that, given generators $f_1,\ldots,f_s$ of I, there is an algorithm (the *Buchberger algorithm*) for producing the corresponding Gröbner basis. A description of this algorithm can be found in Chapter 2 of Ref. 4.

The complexity of the Bucherger algorithm has been studied extensively. Examples are known where the input polynomials have degree $\leq d$, yet the corresponding Gröbner basis contains polynomials of degree 2^{2^d}. Theoretical results show that this doubly exponential behavior is the worst that can occur (for precise references, see Chapter 2 of Ref. 4). However, there are many geometric situations where the complexity is less. For example, if the equations have only finitely many solutions over $\mathbf{C}$, then the complexity drops to a single exponential. Furthermore, obtaining geometric information about an ideal, such as the dimension of its associated variety, often has single exponential complexity. When using graded reverse lexicographic order, complexity is related to the *regularity* of the ideal. This is discussed in Ref. 7. Below we will say more about the practical aspects of Gröbner basis computations.

Using the properties of Gröbner bases, one gets the following *ideal membership algorithm*: Given $f, f_1, \ldots, f_s$, use the Buchberger algorithm to compute a Gröbner basis $g_1, \ldots, g_t$ of $\langle f_1, \ldots, f_s \rangle$ and use the division algorithm to compute the remainder of f on division by $g_1, \ldots, g_t$. Then f is in the ideal $\langle f_1, \ldots, f_s \rangle$ if and only if the remainder is zero.

Another important use of Gröbner bases occurs in *elimination theory*. For example, in geometric modeling, one encounters surfaces in $\mathbf{R}^3$ parametrized by polynomials, say

$$x = f(s,t), \quad y = g(s,t), \quad z = h(s,t).$$

To obtain the equation of the surface, we need to eliminate s, t from the above equations. We do this by considering the ideal

$$\langle x - f(s,t), y - g(s,t), z - h(s,t) \rangle$$

in the polynomial ring $\mathbf{R}[s, t, x, y, z]$ and computing a Gröbner basis for this ideal using lexicographic order, where the variables to be eliminated are listed first. The *Elimination Theorem* (see Chapter 3 of Ref. 4) implies that the equation of the surface is the only polynomial in the Gröbner basis not involving s, t. In practice, elimination is often done by other methods (such as *resultants*) because of complexity issues. See also the entry on SURFACE MODELING.

Our final application concerns a system of equations $f_1 = \cdots = f_s = 0$ in n variables over $\mathbf{C}$. Let $I = \langle f_1, \ldots, f_s \rangle$, and compute a Gröbner basis of I with respect to any monomial order. The *Finiteness Theorem* asserts that the following are equivalent:

1. The equations have finitely many solutions in $\mathbf{C}^n$.
2. The Gröbner basis contains elements whose leading terms are pure powers of the variables (i.e., x_1 to a power, x_2 to a power, etc.) up to constants.
3. The quotient ring $\mathbf{C}[x_1, \ldots, x_n]/I$ is a finite-dimensional vector space over $\mathbf{C}$.

The equivalence of the first two items gives an algorithm for determining whether there are finitely many solutions over $\mathbf{C}$. From here, one can find the solutions by several methods, including *eigenvalue methods* and *homotopy continuation*. These and other methods are discussed in Ref. 8. The software PHCpack (9) is a freely available implementation of homotopy continuation. Using homotopy techniques, systems with 10^5 solutions have been solved. Without homotopy methods but using a robust implementation of the Buchberger algorithm, systems with 1000 solutions have been solved, and in the context of computational biology, some highly structured systems with over 1000 equations have been solved.

However, although solving systems is an important practical application of Gröbner basis methods, we want to emphasize that many theoretical objects in algebraic geometry, such as Hilbert polynomials, free resolutions (see below), and sheaf cohomology (also discussed below), can also be computed by these methods. As more and more of these theoretical objects are finding applications, the ability to compute them is becoming increasingly important.

Gröbner basis algorithms have been implemented in computer algebra systems such as Maple (10) and Mathematica (11). For example, the solve command in Maple and Solve command in Mathematica make use of Gröbner basis computations. We should also mention CoCoA (12), Macaulay 2 (13), and Singular (14), which are freely available on the Internet. These powerful programs are used by researchers in algebraic geometry and commutative algebra for a wide variety of experimental and theoretical computations. With the help of books such as Ref. 5 for Macaulay 2, Ref. 15 for CoCoA, and Ref. 16 for Singular, these programs can be used by beginners. The program Magma (17) is not free but has a powerful implementation of the Buchberger algorithm.

MODULES

Besides rings, ideals, and quotient rings, another important algebraic structure to consider is the concept of *module over a ring*. Let R denote the polynomial ring $k[x_0, \ldots, x_n]$. Then saying that M is an R-module means that M has addition and scalar multiplication with the usual properties, except that the "scalars" are now elements of R. For example, the *free R-module* R^m consists of m-tuples of elements of R. We can clearly add two such m-tuples and multiply an m-tuple by an element of R.

A more interesting example of an R-module is given by an ideal $I = \langle f_1, \ldots, f_s \rangle$ in R. If we choose the generating set $f_1, \ldots, f_s$ to be as small as possible, we get a minimal basis of I. But when $s \geq 2$, $f_1, \ldots, f_s$ cannot be linearly independent over R, because $f_j \cdot f_i + (-f_i) \cdot f_j = 0$ when $i \neq j$. To see how badly the f_i fail to be independent, consider

$$R^s \to I \to 0,$$

where the first arrow is defined using dot product with $(f_1, \ldots, f_s)$ and the second arrow is a standard way of saying the first arrow is onto, which is true because $I = \langle f_1, \ldots, f_s \rangle$. The kernel or nullspace of the first arrow measures the failure of the f_i to be independent. This kernel is an R-module and is called the *syzygy module* of $f_1, \ldots, f_s$, denoted $\mathrm{Syz}(f_1, \ldots, f_s)$.

The *Hilbert Basis Theorem* applies here so that there are finitely many syzygies $\mathbf{h}_1, \ldots, \mathbf{h}_\ell$ in Syz $(f_1, \ldots, f_s)$ such that every syzygy is a linear combination, with coefficients in R, of $\mathbf{h}_1, \ldots, \mathbf{h}_\ell$. Each h_i is a vector of polynomials; if we assemble these into a matrix, then matrix multiplication gives a map

$$R^\ell \to R^s$$

whose image is $\mathrm{Syz}(f_1, \ldots, f_s)$. This looks like linear algebra, except that we are working over a ring instead of a field. If we think of the variables in $R = k[x_1, \ldots, x_n]$ as parameters, then we are doing *linear algebra with parameters*.

The generating syzgyies $\mathbf{h}_i$ may fail to be independent, so that the above map may have a nonzero kernel. Hence we can iterate this process, although the *Hilbert Syzygy Theorem* implies that kernel is eventually zero. The result is a collection of maps

$$0 \to R^t \to \cdots \to R^\ell \to R^s \to I \to 0,$$

where at each stage, the image of one map equals the kernel of the next. We say that this is a *free resolution* of I. By adapting Gröbner basis methods to modules, one obtains algorithms for computing free resolutions.

Furthermore, when I is a homogeneous ideal, the whole resolution inherits a graded structure that makes it straightforward to compute the Hilbert polynomial of I. Given what we know about Hilbert polynomials, this gives an algorithm for determining the dimension and degree of a projective variety. A discussion of modules and free resolutions can be found in Ref. 18.

Although syzygies may seem abstract, there are situations in geometric modeling where syzygies arise naturally as *moving curves* and *moving surfaces* (see Ref. 19). This and other applications show that algebra needs to be added to the list of topics that fall under the rubric of applied mathematics.

LOCAL PROPERTIES

In projective space $\mathbf{P}^n(k)$, let U_i denote the Zariski open subset where $x_i \neq 0$. Earlier we noted that U_0 looks the affine space k^n; the same is true for the other U_i. This means that $\mathbf{P}^n(k)$ locally looks like affine space. Furthermore, if V is a projective variety in $\mathbf{P}^n(k)$, then one can show that $V_i = V \cap U_i$ is a affine variety for all i. Thus, every projective variety locally looks like an affine variety.

In algebraic geometry, one can get even more local. For example, let $\mathbf{p} = [u_0, \ldots, u_n]$ be a point of $\mathbf{P}^n(k)$. Then let O_{p} consist of all rational functions on $\mathbf{P}^n(k)$ defined at $\mathbf{p}$. Then O_{p} is clearly a ring, and the subset consisting of those functions that vanish at $\mathbf{p}$ is a maximal ideal. More surprising is the fact that this is the unique maximal ideal of O_{p}. We call O_{p} the *local ring* of $\mathbf{P}^n(k)$ at $\mathbf{p}$, and in general, a commutative ring with a unique maximal ideal is called a *local ring*. In a similar way, a point $\mathbf{p}$ of an affine or projective variety V has a local ring $O_{V,\mathbf{p}}$.

Many important properties of a variety at a point are reflected in its local ring. As an example, we give the definition of *multiplicity* that occurs in Bézout's Theorem. Recall the statement: Distinct irreducible curves in $\mathbf{P}^2(\mathbf{C})$ of degrees m and n intersect at mn points, counted with multiplicity. By picking suitable coordinates, we can assume that the points of intersection lie in $\mathbf{C}^2$ and that the curves are defined by equations $f = 0$ and $g = 0$ of degrees m and n, respectively. If $\mathbf{p}$ is a point of intersection, then its *multiplicity* is given by

$$\mathrm{mult}(\mathbf{p}) = \dim_{\mathbf{C}} O_{\mathbf{p}}/\langle f, g\rangle, \quad O_{\mathbf{p}} = \text{local ring of } \mathbf{P}^2(\mathbf{C}) \text{ at } \mathbf{p},$$

and the precise version of Bézout's Theorem states that

$$m\,n = \sum_{f(\mathbf{p})=g(\mathbf{p})=0} \mathrm{mult}(\mathbf{p}).$$

A related notion of multiplicity is the *Hilbert–Samuel multiplicity* of an ideal in O_{p}, which arises in geometric modeling when considering the influence of a basepoint on the degree of the defining equation of a parametrized surface.

SMOOTH AND SINGULAR POINTS

In multivariable calculus, the *gradient* $\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j}$ is perpendicular to the level curve defined by $f(x, y) = 0$. When one analzyes this carefully, one is led to the following concepts for a point on the level curve:

- A *smooth point*, where ∇f is nonzero and can be used to define the tangent line to the level curve.
- A *singular point*, where ∇f is zero and the level curve has no tangent line at the point.

These concepts generalize to arbitrary varieties. For any variety, most points are smooth, whereas others—those in the *singular locus*—are singular. Singularities can be important. For example, when one uses a variety to describe the possible states of a robot arm, the singularities of the variety often correspond to positions where the motion of the arm is less predictable (see Chapter 6 of Ref. 4 and the entry on ROBOTICS).

A variety is *smooth* or *nonsingular* when every point is smooth. When a variety has singular points, one can use *blowing up* to obtain a new variety that is less singular. When working over an algebraically closed field of *characteristic* 0 (meaning fields that contain a copy of $\mathbf{Q}$), Hironaka proved in 1964 that one can always find a sequence of blowing up that results in a smooth variety. This is called *resolution of singularities*. Resolution of singularities over a field of *characteristic p* (fields that contain a copy of $\mathbf{F}_p$) is still an open question. Reference 20 gives a nice introduction to resolution of singularities. More recently, various groups of people have figured out how to do this algorithmically, and work has been done on implementing these algorithms, for example, the software desing described in Ref. 21. We also note that singularities can be detected numerically using condition numbers (see Ref. 22).

SHEAVES AND COHOMOLOGY

For an affine variety, modules over its coordinate ring play an important role. For a projective variety V, the corresponding objects are *sheaves of O_V-modules*, where O_V is the structure sheaf of V. Locally, V looks like an affine variety, and with a suitable hypothesis called *quasi-coherence*, a sheaf of O_V-modules locally looks like a module over the coordinate ring of an affine piece of V.

From sheaves, one is led to the idea of *sheaf cohomology*, which (roughly speaking) measures how the local pieces of the sheaf fit together. Given a sheaf F on V, the sheaf cohomology groups are denoted $H^i(V,F)$. We will see below that the sheaf cohomology groups are used in the classification of projective varieties. For another application of sheaf cohomology, consider a finite collection V of points in $\mathbf{P}^n(k)$. From the sheaf point of view, V is defined by an *ideal sheaf* I_V. In interpolation theory, one wants to model arbitrary functions on V using polynomials of a fixed degree, say m. If m is too small, this may not be possible, but we always succeed if m is large enough. A precise description of which degrees m work is given by sheaf cohomology. The ideal sheaf I_V has a *twist* denoted $I_V(m)$. Then all functions on V come from polynomials of degree m if and only if $H^1(\mathbf{P}^n(k), I_V(m)) = \{0\}$. We also note that vanishing theorems for sheaf cohomology have been used in geometric modeling (see Ref. 23).

References 1, 2, and 24 discuss sheaves and sheaf cohomology. Sheaf cohomology is part of *homological algebra*. An introduction to homological algebra, including sheaves and cohomology, is given in Ref. 5.

SPECIAL VARIETIES

We next discuss some special types of varieties that have been studied extensively.

1. *Elliptic Curves and Abelian Varieties*. Beginning with the middle of the eighteenth century, *elliptic integrals* have attracted a lot of attention. The study of these integrals led to both *elliptic functions* and *elliptic curves*. The latter are often described by an equation of the form

$$y^2 = ax^3 + bx^2 + cx + d,$$

where $ax^3 + bx^2 + cx + d$ is a cubic polynomial with distinct roots. However, to get the best properties, one needs to work in the projective plane, where the above equation is replaced with the homogeneous equation

$$y^2z = ax^3 + bx^2z + cxz^2 + dz^3.$$

The resulting projective curve E has an extra structure: Given two points on E, the line connecting them intersects E at a third point by Bézout's Theorem. This leads to a group structure on E where the point at infinity is the identity element.

Over the field of rational numbers $\mathbf{Q}$, elliptic curves have a remarkably rich theory. The group structure is related to the *Birch–Swinnerton-Dyer Conjecture*, and Wiles's proof of Fermat's Last Theorem was a corollary of his solution of a large part of the *Taniyama–Shimura Conjecture* for elliptic curves over $\mathbf{Q}$. On the other hand, elliptic curves over finite fields are used in cryptography (see Ref. 25). The relation between elliptic integrals and elliptic curves has been generalized to *Hodge theory*, which is described in Ref. 24. Higher dimensional analogs of elliptic curves are called *abelian varieties*.

2. *Grassmannians and Schubert Varieties*. In $\mathbf{P}^n(k)$, we use homogeneous coordinates $[u_0, \ldots, u_n]$, where $[u_0, \ldots, u_n] = [v_0, \ldots, v_n]$ if both lie on the same line through the origin in k^{n+1}. Hence points of $\mathbf{P}^n(k)$ correspond to one-dimensional subspaces of k^{n+1}. More generally, the *Grassmannian* $G(N, m)(k)$ consists of all m-dimensional subspaces of k^n. Thus, $G(n+1, 1)(k) = \mathbf{P}^n(k)$.

Points of $G(N, m)(k)$ have natural coordinates, which we describe for $m = 2$. Given a two-dimensional sub-space W of k^N, consider a $2 \times N$ matrix

$$\begin{pmatrix} u_1 & u_2 & \ldots & u_N \\ v_1 & v_2 & \ldots & v_N \end{pmatrix}$$

whose rows give a basis of W. Let p_{ij}, $i < j$, be the determinant of the 2×2 matrix formed by the ith and jth columns. The $M = \binom{N}{2}$ numbers p_{ij} are the *Plücker coordinates* of W. These give a point in $\mathbf{P}^{M-1}(k)$ that depends only on W and not on the chosen basis. Furthermore, the subspace W can be reconstructed from its Plücker coordinates. The Plücker coordinates satisfy the *Plücker relations*

$$p_{ij}p_{kl} - p_{ik}p_{jl} + p_{il}p_{jk} = 0,$$

and any set of numbers satisfying these relations comes from a subspace W. It follows that the Plücker relations define $G(N, 2)(k)$ as a projective variety in $\mathbf{P}^{M-1}(k)$. In general, $G(N, m)(k)$ is a smooth projective variety of dimension $m(N - m)$.

The Grassmannian $G(N, m)(k)$ contains interesting varieties called *Schubert varieties*. The *Schuberi calculus* describes how these varieties intersect. Using the Schubert calculus, one can answer questions such as how many lines in $\mathbf{P}^3(k)$ intersect four lines in general position? (The answer is two.) An introduction to Grassmannians and Schubert vareties can be found in Ref. 26.

The question about lines in $\mathbf{P}^3(k)$ is part of *enumerative algebraic geometry*, which counts the number of geometrically interesting objects of various types. Bezout's Theorem is another result of enumerative algebraic geometry. Another famous enumerative result states that a smooth cubic surface in $\mathbf{P}^3(\mathbf{C})$ contains exactly 27 lines.

3. *Rational and Unirational Varieties*. An irreducible variety V of dimension n over $\mathbf{C}$ is *rational* if there is a one-to-one rational parametrization $U \to V$, where U is a Zariski open subset of $\mathbf{C}^n$. The simplest example of a rational variety is $\mathbf{P}^n(\mathbf{C})$. Many curves and surfaces that occur in geometric modeling are rational.

More generally, an irreducible variety of dimension n is *unirational* if there is a rational parametrization $U \to V$ whose image fills up most of V, where U is a Zariski open subset of $\mathbf{C}^m$, $m \geq n$. For varieties of dimension 1 and 2, unirational and rational coincide, but in dimensions 3 and greater, they differ. For example, a smooth cubic hypersurface in $\mathbf{P}^4(\mathbf{C})$ is unirational but not rational.

A special type of rational variety is a *toric variety*. In algebraic geometry, a *torus* is $(\mathbf{C}^*)^n$, which is the Zariski open subset of $\mathbf{C}^n$ where all coordinates are nonzero. A toric variety V is an n-dimensional irreducible variety that contains a copy of $(\mathbf{C}^*)^n$ as a Zariski open subset in a suitably nice manner. Both $\mathbf{C}^n$ and $\mathbf{P}^n(\mathbf{C})$ are toric varieties. There are strong relations between toric varieties and polytopes, and toric varieties also have interesting applications in geometric modeling (see Ref. 27), algebraic statistics, and computational biology (see Ref. 28). The latter includes significant applications of Gröbner bases.

4. *Varieties over Finite Fields*. A set of equations defining a projective variety V over $\mathbf{F}_p$ also defines V as a projective variety over $\mathbf{F}_{p^m}$ for every $m \geq 1$. As $\mathbf{P}^n(\mathbf{F}_{p^m})$ is finite, we let N_m denote the number of points of V when regarded as lying in $\mathbf{P}^n(\mathbf{F}_{p^m})$. To study the asymptotic behavior of N_m as m gets large, it is convenient to assemble the N_m into the *zeta function*

$$Z(V,\, t) = \exp\left(\sum_{m=1}^{\infty} N_m t^m/m\right).$$

The behavior of $Z(V,\, t)$ is the subject of some deep theorems in algebraic geometry, including the *Riemann hypothesis* for smooth projective varieties over finite fields, proved by Deligne in 1974.

Suppose for example that V is a smooth curve. The *genus* g of V is defined to be the dimension of the sheaf cohomology group $H^1(V,\, O_V)$. Then the Riemann hypothesis implies that

$$|N_m - p^m - 1| \leq 2g\, p^{m/2}.$$

Zeta functions, the Riemann hypothesis, and other tools of algebraic geometry such as the *Riemann–Roch Theorem* have interesting applications in algebraic coding theory. See Ref. 29 and the entry on ALGEBRAIC CODING THEORY. References 18 and 30 discuss aspects of coding theory that involve Gröbner bases.

CLASSIFICATION QUESTIONS

One of the enduring questions in algebraic geometry concerns the classification of geometric objects of various types. Here is a brief list of some classification questions that have been studied.

1. *Curves*. For simplicity, we work over $\mathbf{C}$. The main invariant of smooth projective curve is its genus g, defined above as the dimension of $H^1(V,\, O_V)$. When the genus is 0, the curve is $\mathbf{P}^1(\mathbf{C})$, and when the genus is 1, the curve is an elliptic curve E. After a coordinate change, the affine equation can be written as

$$y^2 = x^3 + ax + b, \quad 4a^3 + 27b^2 \neq 0.$$

The *j-invariant* $j(E)$ is defined to be

$$j(E) = \frac{2^8 3^3 a^3}{4a^3 + 27b^2}$$

and two elliptic curves over $\mathbf{C}$ are isomorphic as varieties if and only if they have the same j-invariant. It follows that isomorphism classes of elliptic curves correspond to complex numbers; one says that $\mathbf{C}$ is the *moduli space* for elliptic curves. Topologically, all elliptic curves look like a torus (the surface of a donut), but algebraically, they are the same if and only if they have the same j-invariant.

Now consider curves of genus $g \geq 2$ over $\mathbf{C}$. Topologically, these look like a surface with g holes, but algebraically, there is a moduli space of dimension $3g - 3$ that records the algebraic structure. These moduli spaces and their compactifications have been studied extensively. Curves of genus $g \geq 2$ also have strong connections with non-Euclidean geometry.

2. *Surfaces*. Smooth projective surfaces over $\mathbf{C}$ have a richer structrure and hence a more complicated classification. Such a surface S has its *canonical bundle* ω_S, which is a sheaf of O_S-modules that (roughly speaking) locally looks like multiples of $dxdy$ for local coordinates $x,\, y$. Then we get the associated bundle ω_S^m, which locally looks like multiples of $(dxdy)^m$. The dimension of the sheaf cohomology group $H^0(S,\, \omega_S^m)$ grows like a polynomial in m, and the degree of this polynomial is the *Kodaira dimension*κ of S, where the zero polynomial has degree $-\infty$. Using the Kodaira dimension, we get the following *Enriques-Kodaira classification*:

$\kappa = -\infty$: Rational surfaces and ruled surfaces over curves of genus > 0.

$\kappa = 0$: K3 surfaces, abelian surfaces, and Enriques surfaces.

$\kappa = 1$: Surfaces mapping to a curve of genus ≥ 2 whose generic fiber is an elliptic curve.

$\kappa = 2$: Surfaces of general type.

One can also define the Kodaira dimension for curves, where the possible values $\kappa = -\infty$, 0, 1 correspond to the classication by genus $g = 0$, 1 or ≥ 2. One difference in the surface case is that blowing up causes problems. One needs to define the *minimal model* of a surface, which exists in most cases, and then the minimal model gets "classified" by describing its moduli space. These moduli spaces are well understood except for surfaces of general type, where many unsolved problems remain.

To say more about how this classification works, we need some terminology. Two irreducible varieties are *birational* if they have Zariski open subsets that are isomorphic. Thus, a variety over $\mathbf{C}$ is rational if and only if it is birational to $\mathbf{P}^n(\mathbf{C})$, and two smooth projective surfaces are birational if and only if they have the same minimal model. As for moduli, consider the equation

$$a\left(x_0^4 + x_1^4 + x_2^4 + x_3^4\right) + x_0x_1x_2x_3 = 0.$$

This defines a K3 surface in $\mathbf{P}^3(\mathbf{C})$ provided $a \neq 0$. As we vary a, we get different K3 surfaces that can be *deformed* into each other. This (very roughly) is what happens in a moduli space, although a lot of careful work is needed to make this idea precise.

The Enriques–Kodaira classification is described in detail in Ref. 31. This book also discuss the closely related classication of smooth complex surfaces, not necessarily algebraic.

1. *Higher Dimensions*. Recall that a three-fold is a variety of dimension 3. As in the surface case, one uses the Kodaira dimension to break up all three-folds into classes, this time according to $k = -\infty$, 0, 1, 2, 3. One new feature for three-folds is that although minimal models exist, they may have certain mild singularities. Hence, the whole theory is more sophisticated than the surface case. The general strategy of the *minimal model program* is explained in Ref. 32.
2. *Hilbert Schemes*. Another kind of classification question concerns varieties that live in a fixed ambient space. For example, what sorts of surfaces of small degree exist in $\mathbf{P}^4(\mathbf{C})$? There is also the *Hartshorne conjecture*, which asserts that a smooth variety V of dimension n in $\mathbf{P}^N(\mathbf{C})$, where $N < \frac{3}{2}n$, is a *complete intersection*, meaning that V is defined by a system of exactly $N - n$ equations.

   In general, one can classify *all* varieties in $\mathbf{P}^n(\mathbf{C})$ of given degree and dimension. One gets a better classification by looking at all varieties with given Hilbert polynomial. This leads to the concept of a *Hilbert scheme*. There are many unanswered questions about Hilbert schemes.
3. *Vector Bundles*. A *vector bundle of rank r* on a variety V is a sheaf that locally looks like a free module of rank r. For example, the tangent planes to a smooth surface form its *tangent bundle*, which is a vector bundle of rank 2.

   Vector bundles of rank 1 are called *line bundles* or *invertible sheaves*. When V is smooth, line bundles can be described in terms of *divisors*, which are formal sums $a_1D_1 + \cdots + a_mD_m$, where a_i is an integer and D_i is an irreducible hypersurface. Furthermore, line bundles are isomorphic if and only if their corresponding divisors are *rationally equivalent*. The set of isomorphism classes of line bundles on V forms the *Picard group* Pic(V).

   There has also been a lot of work classifying vector bundles on $\mathbf{P}^n(\mathbf{C})$. For $n = 1$, a complete answer is known. For $n > 2$, one classifies vector bundles E according to their rank r and their *Chern classes* $c_i(E)$. One important problem is understanding how to compactify the corresponding moduli spaces. This involves the concepts of *stable* and *semistable* bundles. Vector bundles also have interesting connections with mathematical physics (see Ref. 33).
4. *Algebraic Cycles*. Given an irreducible variety V of dimension n, a variety W contained in V is called a *subvariety*. Divisors on V are integer combinations of irreducible subvarieties of dimension $n - 1$. More generally, an *m-cycle* on V is an integer combination of irreducible subvarieties of dimension m. Cycles are studied using various equivalence relations, including *rational equivalence, algebraic equivalence, numerical equivalence*, and *homological equivalence*. The *Hodge Conjecture* concerns the behavior of cycles under homological equivalence, whereas the *Chow groups* are constructed using rational equivalence.

   Algebraic cycles are linked to other topics in algebraic geometry, including *motives, intersection theory*, and *variations of Hodge structure*. An introduction to some of these ideas can be found in Ref. 34.

REAL ALGEBRAIC GEOMETRY

In algebraic geometry, the theory usually works best over **C** or other algebraically closed fields. Yet many applications of algebraic geometry deal with real solutions of polynomial equations. We will explore several aspects of this question.

When dealing with equations with finitely many solutions, there are powerful methods for estimating the number of solutions, including a multivariable version of Bézout's Theorem and the more general *BKK bound*, both of which deal with complex solutions. But these bounds can differ greatly from the number of real solutions. An example from Ref. 35 is the system

$$\begin{aligned} axyz^m &+ bx &+ cy &+ d &= 0\\ a'xyz^m &+ b'x &+ c'y &+ d' &= 0\\ a''xyz^m &+ b''x &+ c''y &+ d'' &= 0 \end{aligned}$$

where m is a positive integer and $a, b, \ldots, c'', d''$ are random real coefficients. The BKK bound tells us that there are m complex solutions, and yet there are at most two real solutions.

Questions about the number of real solutions go back to *Descartes' Rule of Signs* for the maximum number of positive and negative roots of a real univariate polynomial. There is also *Sturm's Theorem*, which gives the number of real roots in an interval. These results now have multivariable generalizations. Precise statements can be found in Refs. 18 and 30.

Real solutions also play an important role in enumerative algebraic geometry. For example, a smooth cubic surface S defined over **R** has 27 lines when we regard S as lying in $\mathbf{P}^3(\mathbf{C})$. But how many of these lines are real? In other words, how many lines lie on S when it is regarded as lying in $\mathbf{P}^3(\mathbf{R})$? (The answer is 27, 15, 7, or 3, depending on the equation of the surface.) This and other examples from *real enumerative geometry* are discussed in Ref. 35.

Over the real numbers, one can define geometric objects using inequalities as well as equalities. For example, a solid sphere of radius 1 is defined by $x^2 + y^2 + z^2 \leq 1$. In general, a finite collection of polynomial equations and inequalities define what is known as a *semialgebraic variety*. Inequalities arise naturally when one does *quantifier elimination*. For example, given real numbers a and b, the question

$$\text{Does there exist } x \text{ in } \mathbf{R} \text{ with } x^2 + bx + c = 0?$$

is equivalent to the inequality

$$b^2 - 4c \geq 0$$

by the quadratic formula. The theory of real quantifier elimination is due to Tarksi, although the first practical algorithmic version is Collin's *cylindrical algebraic decomposition*. A brief discussion of these issues appears in Ref. 30. Semialgebraic varieties arise naturally in robotics and motion planning, because obstructions like floors and walls are defined by inequalities (see ROBOTICS).

SCHEMES

An affine variety V is the geometric object corresponding to the algebraic object given by its coordinate ring $k[V]$. More generally, given *any* commutative ring R, Grothendieck defined the *affine scheme* Spec (R) to be the geometric object corresponding to R. The points of Spec(R) correspond to prime ideals of R, and Spec(R) also has a structure sheaf $O_{\mathrm{Spec(R)}}$ that generalizes the sheaves O_V mentioned earlier.

As an example, consider the coordinate ring $\mathbf{C}[V]$ of an affine variety V in $\mathbf{C}^n$. We saw earlier that the points of V correspond to maximal ideals of $\mathbf{C}[V]$. As maximal ideals are prime, it follows that Spec$(\mathbf{C}[V])$ contains a copy of V. The remaining points of Spec$(\mathbf{C}[V])$ correspond to the other irreducible varieties lying in V. In fact, knowing Spec$(\mathbf{C}[V])$ is equivalent to knowing V in a sense that can be made precise.

Affine schemes have good properties with regard to maps between rings, and they can be patched together to get more general objects called *schemes*. For example, every projective variety has a natural scheme structure. One way to see the power of schemes is to consider the intersection of the curves in $\mathbf{C}^2$ defined by $f = 0$ and $g = 0$, as in our discussion of Bezout's Theorem. As varieties, this intersection consists of just points, but if we consider the intersection as a scheme, then it has the additional structure consisting of the ring $O_{\mathbf{p}}/\langle f, g\rangle$ at every intersection point $\mathbf{p}$. So the scheme–theoretic intersection knows the multiplicities. See Ref. 36 for an introduction to schemes. Scheme theory is also discussed in Refs. 1 and 2.

BIBLIOGRAPHY

1. R. Hartshorne, *Algebraic Geometry*, New York: Springer, 1977.
2. I. R. Shafarevich, Basic Algebraic Geometry, New York: Springer, 1974.
3. B. Buchberger, Gröbner bases: An algorithmic method in polynomial ideal theory, in N. K. Bose (ed.), *Recent Trends in Multidimensional Systems Theory*, Dordrecht: D. Reidel, 1985.
4. D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties and Algorithms*, 3rd ed., New York: Springer, 2007.
5. H. Schenck, *Computational Algebraic Geometry*, Cambridge: Cambridge University Press, 2003.
6. K. Smith, L. Kahanpää, P. Kekäläinen, and W. Traves, *An Invitation to Algebraic Geometry*, New York: Springer, 2000.
7. D. Bayer and D. Mumford, What can be computed in algebraic geometry? in D. Eisenbud and L. Robbiano (eds.), *Computational Algebraic Geometry and Commutative Algebra*, Cambridge: Cambridge University Press, 1993.
8. A. Dickenstein and I. Emiris, *Solving Polynomial Systems*, New York: Springer, 2005.
9. PHCpack, a general purpose solver for polynomial systems by homotopy continuation. Available: http://www.math.uic.edu/~jan/PHCpack/phcpack.html.
10. Maple. Available: http://www.maplesoft.com.
11. Mathematica. Available: http://www.wolfram.com.
12. CoCoA, Computational Commutative Algebra. Available: http://www.dima.unige.it.
13. Macaulay 2, a software for system for research in algebraic geometry. Available: http://www.math.uiuc.edu/Macaulay2.
14. Singular, a computer algebra system for polynomial computations. Available: http://www.singular.uni-kl.de.
15. M. Kreuzer and L. Robbiano, *Computational Commutative Algebra 1*, New York: Springer, 2000.
16. G.-M. Greuel and G. Pfister, *A Singular Introduction of Commutative Algebra*, New York: Springer, 2002.
17. Magma, The Magma Computational Algebra System. Available: http://magma.maths.usyd.edu.au/magma/.
18. D. Cox, J. Little, and D. O'Shea, *Using Algebraic Geometry*, 2nd ed., New York: Springer, 2005.
19. T. W. Sederberg and F. Chen, Implicitization using moving curves and surfaces, in S. G. Mair and R. Cook (eds.), *Proceedings of the 22nd Annual Conference on Computer graphics and interactive techniques (SIGGRAPH1995)*, New York: ACM Press, 1995, pp. 301–308.
20. H. Hauser, The Hironaka theorem on resolution of singularities (or: a proof we always wanted to understand), *Bull. Amer. Math. Soc.*, **40**: 323–403, 2003.
21. G. Bodnár and J. Schicho, Automated resolution of singularities for hypersurfaces, *J. Symbolic Computation.*, **30**: 401–429, 2000. Available: http://www.rise.uni-linz.ac.at/projects/basic/adjoints/blowup.
22. H. Stetter, *Numerical Polynomial Algebra*, Philadelphia: SIAM, 2004.
23. D. Cox, R. Goldman, and M. Zhang, On the validity of implicitization by moving quadrics for rational surfaces with no base points, *J. Symbolic Comput.*, **29**: 419–440, 2000.
24. P. Griffiths and J. Harris, *Principles of Algebraic Geometry*, New York: Wiley, 1978.
25. N. Koblitz, *A Course in Number Theory and Cryptography*, 2nd ed., New York: Springer, 1994.
26. S. L. Kleiman and D. Laksov, Schubert calculus, *Amer. Math. Monthly*, **79**: 1061–1082, 1972.
27. R. Goldman and R. Krasauskas (eds.), *Topics in Algebraic Geometry and Geometric Modeling*, Providence, RI: AMS, 2003.
28. L. Pachter and B. Sturmfels (eds.), *Algebraic Statistics for Computational Biology*, Cambridge: Cambridge University Press, 2005.

29. C. Moreno, *Algebraic Curves over Finite Fields*, Cambridge: Cambridge University Press, 1991.

30. A. M. Cohen, H. Cuypers, and H. Sterk (eds.), *Some Tapas of Computer Algebra*, New York: Springer, 1999.

31. W. P. Barth, C. A. Peters, and A. A. van de Ven, *Compact Complex Surfaces*, New York: Springer, 1984.

32. C. Cadman, I. Coskun, K. Jarbusch, M. Joyce, S. Kovács, M. Lieblich, F. Sato, M. Szczesny, and J. Zhang, A first glimpse at the minimal model program, in R. Vakil (ed.), *Snowbird Lectures in Algebraic Geometry*, Providence, RI: AMS, 2005.

33. V. S. Vardarajan, Vector bundles and connections in physics and mathematics: Some historical remarks, in V. Lakshmibai, V. Balaji, V. B. Mehta, K. R. Nagarajan, K. Paranjape, P. Sankaran, and R. Sridharan (eds), *A Tribute to C. S. Seshadri*, Basel: Birkhäuser-Verlag, 2003, pp. 502–541.

34. W. Fulton, *Introduction to Intersection Theory in Algebraic Geometry*, Providence, RI: AMS, 1984.

35. F. Sottile, Enumerative real algebraic geometry, in S. Basu and L. Gonzalez-Vega (eds.), *Algorithmic and quantitative real algebraic geometry (Piscataway, NJ, 2001)*, Providence, RI: AMS, 2003, pp. 139–179.

36. D. Eisenbud and J. Harris, *The Geometry of Schemes*, New York: Springer, 2000.

David A. Cox
Amherst College
Amherst, Massachusetts

C

CHOICE UNCERTAINTY PRINCIPLE[1]

The choice uncertainty principle says that it is impossible to make an unambiguous choice between near-simultaneous events under a deadline. This principle affects the design of logic circuits in computer hardware, real-time systems, and decision systems.

One of the first persons to notice that a fundamental principle might be at work in circuits that make decisions was David Wheeler of the University of Cambridge. In the early 1970s, he sought to build a computer whose hardware did not suffer from "hardware freezes" that were common in earlier computers. Wheeler noticed the lockups never occurred when the interrupts were turned off. Interrupt signals were recorded on a flip-flop the CPU consulted between instructions: The CPU decided either to enter the next instruction cycle or to jump to a dedicated subroutine that responded to the interrupt signal. He suspected that the timing of the interrupt signal's arrival to that flip-flop occasionally caused it to misbehave and hang the computer. Imagine that: The simplest, most fundamental memory circuit of a computer could malfunction.

THE HALF-SIGNAL

A digital machine consists of storage elements interconnected by logic circuits. The storage elements, implemented as arrays of flip-flops, hold the machine's state. The machine operates in a cycle: (1) flip-Flops enter a state; the switching time is 10^{-12} to 10^{-15} seconds. (2) The logic circuits take the state as input and produce a new state; the propagation time of all inputs through the circuits is slower, 10^{-9} to 10^{-10} seconds. (3) The new state is read into the flip-flops. A clock sends pulses that tell the flip-flops when to read in the next state.

The clock cycle must be longer than the propagation delay of the logic circuits. If it is any shorter, the inputs to some flip-flops may still be changing at the moment the clock pulse arrives. If an input voltage is changing between the 0 and 1 values at the time the clock pulse samples it, the flip-flop sees a "half signal"—an in-between voltage but not a clear 0 or 1. Its behavior becomes unpredictable. A common malfunction is that the flip-flop ends up in the wrong state: The clock-sampled value of an input intended to switch the flip-flop to 1 might not be strong enough, so the flip-flop remains 0.

THE METASTABLE STATE

Unfortunately, there is a worse malfunction. A half-signal input can cause the flip-flop to enter a "metastable state" for an indeterminate time that may exceed the clock interval by a large amount. The flip-flop eventually settles into a stable state, equally likely to be 0 or 1.

A flip-flop's state is actually a voltage that moves continuously between the 0 and 1 values. The 0 and 1 states are stable because they are attractors: Any small perturbation away from either is pulled back. A flip-flop switches because the input adds enough energy to push the state voltage closer to the other attractor. However, a half-signal input can sometimes leave the state voltage poised precisely at the midpoint between the attractors. The state balances precariously there until some noise pushes it closer to one of the attractors. That midpoint is called the metastable state. The metastable state is like a ball poised perfectly on the peak of a roof: It can sit there for a long time until air molecules or roof vibrations cause it to lose its balance, causing it to roll down one side of the roof.

In 1973, Chaney and Molnor at Washington University in St Louis measured the occurrence rate and holding times of metastable states (1); see Fig. 1. By synchronizing clock frequency with external signal frequency, they attempted to induce a metastable event on every external signal change. They saw frequent metastable events on their oscilloscope, some of which persisted 5, 10, or even 20 clock intervals. Three years later, Kinniment and Woods documented metastable states and mean times until failure for a variety of circuits (2).

In 2002, Sutherland and Ebergen reported that contemporary flip-flops switched in about 100 picoseconds (10^{-10} seconds) and that a metastable state lasting 400 picoseconds or more occurred once every 10 hours of operation (3).

Xilinx.com reports that its modern flip-flops have essentially zero chance of observing a metastable state when clock frequencies are 200 MHz or less (4). At these frequencies, the time between clock pulses (5 nanoseconds) is longer than all metastable events. But in experiments with interrupt signals arriving 50 million times a second, a metastable state occurs about once a minute at a clock frequency of 300 MHz, and about once every two milliseconds at a clock frequency of 400 MHz.

WHEELER'S THRESHOLD FLIP-FLOP

Aware of the Chaney–Molnor experiments, Wheeler realized that an interrupt flip-flop-driven metastable by an ill-timed interrupt signal can still be metastable at the next clock tick. Then the CPU reads neither a definite 0 nor a definite 1 and can malfunction.

Wheeler saw he could prevent the malfunction if he could guarantee that the CPU could only read the interrupt flip-flop while it was stable. He made an analogy with the common situation of two people about to collide on a sidewalk. On sensing their imminent collision, they both stop. They exchange eye signals, gestures, head bobs, sways, dances, and words until finally they reach an agreement that one person goes to the right and the other to the left. This could take a fraction or a second or minutes. They then

[1] This article is adapted from the author's article of the same title, ACM *Communications 50* 50: 9–14, 2007.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

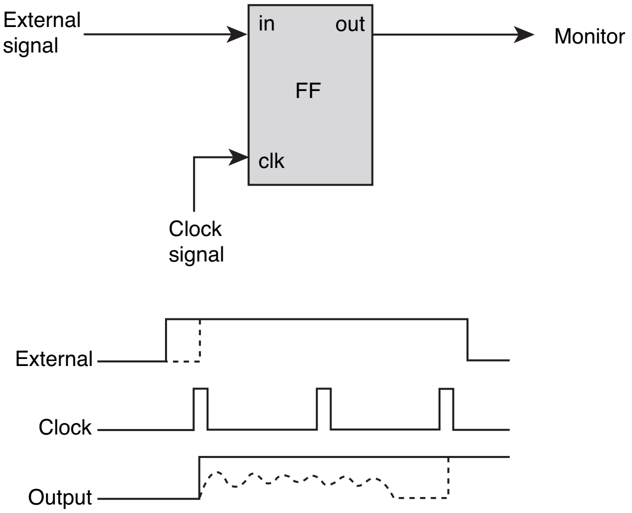


Figure 1. Experimental setup for observing flip-flop (FF) metastability. Each clock pulse triggers the FF state to match the input signal. If the input signal is changing when the clock pulse arrives, FF may enter an indefinite state that lasts more than one clock interval (dotted lines). The test repeats cyclically after the external signal returns to 0. To maximize metastable events, the clock frequency is tuned to a multiple of the external signal frequency. In a digital computer, the indefinite output becomes the input of other logic circuits at the next clock pulse, causing half-signal malfunctions.

resume walking and pass one another without a collision. The key is that the two parties stop for as long as is needed until they decide.

Wheeler designed a flip-flop with an added threshold circuit that output 1 when the flip-flop state was near 0 or 1. He used this for the interrupt flip-flop with the threshold output wired to enable the clock to tick (see Fig. 2). The CPU could not run as long as the interrupt flip-flop was in a metastable state, and thus, it could observe that flip-flop only when it was stable.

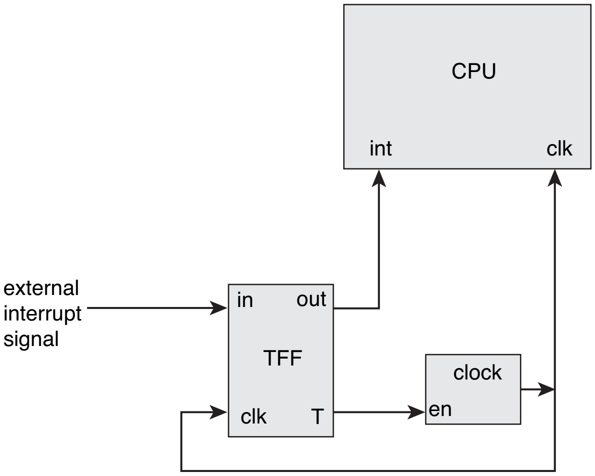


Figure 2. Threshold flip-flop (TFF) output T is 1 when the state is 0 or 1 and is 0 when the state is metastable. Output T enables the clock. TFF can become metastable if the external interrupt signal changes just as the clock pulse arrives. Since the CPU does not run when the clock is off, it always sees a definite 0 or 1 when it samples for interrupts.

With threshold interrupt flip-flops, Wheeler's computer achieved the reliability he wanted.

ARBITER CIRCUIT

Unambiguous choices between near-simultaneous signals must be made in many parts of computing systems, not just at interrupt flip-flops. Examples:

- Two CPUs request access to the same memory bank.
- Two transactions request a lock on the same record of a database.
- Two external events arrive at an object at the same time.
- Two computers try to broadcast on an Ethernet at the same time.
- Two packets arrive together to the network card.
- An autonomous agent receives two request signals at the same time.
- A robot perceives two alternatives at the same time.

In each case, a chooser circuit must select one of the alternatives for immediate action and defer the other for later action. There is no problem if the signals are separated enough that the chooser can tell which one came first. But if the two signals are near simultaneous, the chooser must make an arbitrary selection. This selection problem is also called the arbitration problem, and the circuits that accomplish it are called arbiter or synchronizer circuits (5,6). Ran Ginosar gives a nice account of modern synchronizers (7).

The arbiter incorporates circuits, as in the Wheeler flip-flop, that will prevent it from sending signals while in a metastable state. Therefore, all entities interacting with the arbiter will be blocked while the arbiter is metastable, and there is no need to stop a clock. The main effect of the metastable state is to add an unknown delay to the access time to the shared entity.

THE UNCERTAINTY PRINCIPLE

We can summarize the analysis above as the Choice Uncertainty Principle (8): "No choice between near-simultaneous events can be made unambiguously within a preset deadline." The source of the uncertainty is the metastable state that can be induced in the chooser by conflicting forces generated when two distinct signals change at the same time.

In 1984, Leslie Lamport stated this principle in a slightly different way: "A discrete decision based upon an input having a continuous range of values cannot be made within a bounded length of time" (9). He gave numerous examples of decision problems involving continuous inputs with inherent uncertainty about decision time. The source of the uncertainty, however, is not necessarily the attempt to sample a continuous signal; it is the decision procedure itself. A device that selects among alternatives can become metastable if the signals denoting alternatives arrive at nearly the same time.

It might be asked whether there is a connection between the choice uncertainty principle and the Heisenberg Uncertainty Principle (HUP) of quantum physics. The HUP says that the product of the standard deviations of position and momentum is lower-bounded by a number on the order of 10^{-34} joule-seconds. Therefore, an attempt to reduce the uncertainty of position toward zero may increase the uncertainty of momentum; we cannot know the exact position and speed of a particle at once. This principle manifests at quantum time scales and subatomic particle sizes—look at how small that bound is—but does not say much about the macro effects of millions of electrons flowing in logic circuits.

The HUP is sometimes confused with a simpler phenomenon, which might be called the observer principle. This principle states that if the process of observing a system either injects or withdraws energy from the system, the act of observation may influence the state of the system. There is, therefore, uncertainty about whether what is observed is the same as what is in the system when there is no observer. The observer principle plays an important role in quantum cryptography, where the act of reading the quantum state of a photon destroys the state. The information of the state is transferred to the observer and is no longer in the system.

The choice uncertainty principle is not an instance of the Heisenberg principle because it applies to macrolevel choices as well as to microscopic circuit choices. Neither is it an instance of the observer principle because the metastable state is a reaction of the observer (arbiter) to the system and does not exchange information with the system. (Neither is it related to the Axiom of Choice in mathematics, which concerns selecting one representative from each of an infinite number of sets.)

CHOICE UNCERTAINTY AS A GREAT PRINCIPLE

The choice uncertainty principle is not about how a system reacts to an observer, but how an observer reacts to a system. It also applies to choices at time scales much slower than computer clocks. For example,

- A teenager must choose between two different, equally appealing prom invitations.
- Two people on a sidewalk must choose which way each goes to avoid a collision.
- A driver approaching an intersection must choose to brake or accelerate on seeing the traffic light change to yellow.
- The commander in the field must choose between two adjutants, both demanding quick decisions on complex tactical issues at different locations.
- A county social system must choose between a development plan that limits growth and one that promotes growth.

These examples all involve perceptions; the metastable (indecisive) state occurs in single or interacting brains as they try to choose between equally attractive perceptions. At these levels, a metastable (indecisive) state can persist for seconds, hours, days, months, or even years.

The possibility of indefinite indecision is often attributed to the fourteenth century philosopher Jean Buridan, who described the paradox of the hungry dog that, being placed midway between two equal portions of food, starved (5). [Some authors use the example of an ass (donkey) instead of a dog, but it is the same problem (3,9)]. If he were discussing this today with cognitive scientists, Buridan might say that the brain can be immobilized in a metastable state when presented with equally attractive alternatives.

At these levels, it is not normally possible to turn off clocks until the metastable state is resolved. What happens if the world is impatient and demands a choice from a metastable chooser? A common outcome is that no choice is made and the opportunities represented by the choices are lost. For example, the teenager gets no prom date, the pedestrians collide, the driver runs a red light, the commander loses both battles, or the county has no plan at all. Another outcome is that the deciding parties get flustered, adding to the delay of reaching a conclusion.

CONCLUSION

Modern software contains many external interactions with a network and must frequently choose between near-simultaneous signals. The process of choosing will always involve the possibility of a metastable state and, therefore, a long delay for the decision. Real-time control systems are particularly challenging because they constantly make choices under deadlines.

The metastable state can occur in any choice process where simultaneous alternatives are equally attractive. In that case, the choosing hardware, software, brain, or social process cannot make a definitive choice within any preset interval. If we try to force the choice before the process exits a metastable state, we are likely to get an ambiguous result or no choice at all.

The choice uncertainty principle applies at all levels, from circuits, to software, to brains, and to social systems. Every system of interactions needs to deal with it. It, therefore, qualifies as a Great Principle.

It is a mistake to think that the choice uncertainty principle is limited to hardware. Suppose that your software contains a critical section guarded by semaphores. Your proof that the locks choose only one process at a time to enter the critical section implicitly assumes that only one CPU at a time can gain access to the memory location holding the lock value. If that is not so, then occasionally your critical section will fail no matter how careful your proofs. Every level of abstraction at which we prove freedom from synchronization errors always relies on a lower level at which arbitration is solved. But arbitration can never be solved absolutely.

Therefore, software's assumption that variables denoting alternatives are well defined and unchanging when we look at them is not always valid. The choice uncertainty principle warns us of this possibility and helps to manage it.

REFERENCES

1. T. J. Chaney and C. E. Molnor, Anomalous behavior of synchronizer and arbiter circuits, IEEE, *Trans. Comput.*, **22**: 421–422, 1973.

2. D. J. Kinniment and J. V. Woods, Synchronization and arbitration circuits in digital systems, IEEE *Proc.*, 961–966, 1976.

3. I. Sutherland and J. Ebergen, Computers without clocks, *Scientif. Am.* 62–69, August 2002. Available from Sun Microsystems. http://research.sun.com/async/Publications/KPDisclosed/SciAm/SciAm.pdf

4. P. Alfke, Metastable recovery in Virtex-II Pro FPGAs. Technical Report xapp094 (Feb 2005). Available from the Xilinx.com website.

5. P. Denning, The arbitration problem, *Am. Scient.*, **73**: 516–518, 1985. It is interesting that some authors ascribe the indecision paradox to an "ass," although Buridan's original text refers to a "dog."

6. C. L. Seitz, System timing, in C. Mead and L. Conway (ed.), *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980, pp. 218–262.

7. R. Ginosar, Fourteen ways to fool your synchronizer, *Proc. 9th Int'l Symp. on Asynchronous Circuits and Systems*, IEEE, 2003, 8pp. Available: http://www.ee.technion.ac.il/~ran/papers/Sync_Errors_Feb03.pdf

8. Great Principles Web site: http://cs.gmu.edu/cne/pjd/GP

9. L. Lamport, Buridan's Principle. Technical Report. 1984. Available: http://research.microsoft.com/users/lamport/pubs/buridan.pdf

Peter J. Denning
Naval Postgraduate School
Monterey, California

COMPUTATIONAL COMPLEXITY THEORY

Complexity theory is the part of theoretical computer science that attempts to prove that certain transformations from input to output are impossible to compute using a reasonable amount of resources. Theorem 1 below illustrates the type of "impossibility" proof that can sometimes be obtained (1); it talks about the problem of determining whether a logic formula in a certain formalism (abbreviated WS1S) is true.

Theorem 1. *Any circuit of* AND, OR, *and* NOT *gates that takes as input a WS1S formula of 610 symbols and outputs a bit that says whether the formula is true must have at least* 10^{125}*gates.*

This is a very compelling argument that no such circuit will ever be built; if the gates were each as small as a proton, such a circuit would fill a sphere having a diameter of 40 billion light years! Many people conjecture that somewhat similar intractability statements hold for the problem of factoring 1000-bit integers; many public-key cryptosystems are based on just such assumptions.

It is important to point out that Theorem 1 is specific to a particular circuit technology; to prove that there is no efficient way to compute a function, it is necessary to be specific about *what* is performing the computation. Theorem 1 is a compelling proof of intractability precisely because every deterministic computer that can be purchased today can be simulated efficiently by a circuit constructed with AND, OR, and NOT gates. The inclusion of the word "deterministic" in the preceding paragraph is significant; some computers are constructed with access to devices that are presumed to provide a source of random bits. Probabilistic circuits (which are allowed to have some small chance of producing an incorrect output) might be a more powerful model of computing. Indeed, the intractability result for this class of circuits (1) is slightly weaker:

Theorem 2. *Any probabilistic circuit of* AND, OR, *and* NOT *gates that takes as input a WS1S formula of 614 symbols and outputs a bit that says whether the formula is true (with error probability at most 1/3) must have at least* 10^{125}*gates.*

The underlying question of the appropriate *model of computation* to use is central to the question of how relevant the theorems of computational complexity theory are. Both deterministic and probabilistic circuits are examples of "classical" models of computing. In recent years, a more powerful model of computing that exploits certain aspects of the theory of quantum mechanics has captured the attention of the research communities in computer science and physics. It seems likely that some modification of theorems 1 and 2 holds even for quantum circuits. For the factorization problem, however, the situation is different. Although many people conjecture that classical (deterministic or probabilistic) circuits that compute the factors of 1000-bit numbers must be huge, it is known that small quantum circuits can compute factors (2). It remains unknown whether it will ever be possible to build quantum circuits, or to simulate the computation of such circuits efficiently. Thus, complexity theory based on classical computational models continues to be relevant.

The three most widely studied general-purpose "realistic" models of computation today are deterministic, probabilistic, and quantum computers. There is also interest in *restricted* models of computing, such as *algebraic circuits* or *comparison-based* algorithms. Comparison-based models arise in the study of sorting algorithms. A comparison-based sorting algorithm is one that sorts n items and is not allowed to manipulate the representations of those items, other than being able to test whether one is greater than another. Comparison-based sorting algorithms require time $\Omega(n \log n)$, whereas faster algorithms are sometimes possible if they are allowed to access the bits of the individual items. Algebraic circuits operate under similar restrictions; they cannot access the individual bits of the representations of the numbers that are provided as input, but instead they can only operate on those numbers via operations such as $+$, $\times$, and $\div$. Interestingly, there is also a great deal of interest in "unrealistic" models of computation, such as nondeterministic machines. Before we explain why unrealistic models of computation are of interest, let us see the general structure of an intractability proof.

DIAGONALIZATION AND REDUCIBILITY

Any intractability proof has to confront a basic question: How can one prove that there is *not* a clever algorithm for a certain problem? Here is the basic strategy that is used to prove theorems 1 and 2. There are three steps.

Step 1 involves showing that there is program A that uses roughly 2^n bits of memory on inputs of size n such that, for every input length n, the function that A computes on inputs of length n requires circuits *as large as are required by any function on n bits*. The algorithm A is presented by a "diagonalization" argument (so-called because of similarity to Cantor's "diagonal" argument from set theory). The same argument carries through essentially unchanged for probabilistic and quantum circuits. The problem computed by A is hard to compute, but this by itself is not very interesting, because it is probably not a problem that anyone would ever *want* to compute.

Step 2 involves showing that there is an efficiently computable function f that transforms any input instance x for A into a WS1S formula ϕ(i.e., $f(x) = \phi$) with the property that A outputs "1" on input x if and only if the formula ϕ is true. If there were a small circuit deciding whether a formula is true, then there would be a small circuit for the problem computed by A. As, by step 1, there is

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

no such small circuit for A, it follows that there is no small circuit deciding whether a formula is true.

Step 3 involves a detailed analysis of the first two steps, in order to obtain the concrete numbers that appear in Theorem 1.

Let us focus on step 2. The function f is called a *reduction*. Any function g such that x is in B if and only if $g(x)$ is in C is said to *reduce* B to C. (This sort of reduction makes sense when the computational problems B and C are problems that require a "yes or no" answer; thus, they can be viewed as *sets*, where x is in B if B outputs "yes" on input x. Any computational problem can be viewed as a set this way. For example, computing a function h can be viewed as the set $\{(x, i)$: the ith bit of $f(x)$ is $1\}$.)

Efficient reducibility provides a remarkably effective tool for classifying the computational complexity of a great many problems of practical importance. The amazing thing about the proof of step 2 (and this is typical of many theorems in complexity theory) is that it makes *no use at all* of the algorithm A, other than the fact that A uses at most 2^n memory locations. *Every* problem that uses at most this amount of memory is efficiently reducible to the problem of deciding whether a formula is true. This provides motivation for a closer look at the notion of efficient reducibility.

EFFICIENT COMPUTATION, POLYNOMIAL TIME

Many notions of efficient reducibility are studied in computational complexity theory, but without question the most important one is polynomial-time reducibility. The following considerations explain why this notion of reducibility arises.

Let us consider the informal notion of "easy" functions (functions that are easy to compute). Here are some conditions that one might want to satisfy, if one were trying to make this notion precise:

- If f and g are easy, then the composition $f \circ g$ is also easy.
- If f is computable in time n^2 on inputs of length n, then f is easy.

These conditions might seem harmless, but taken together, they imply that some "easy" functions take time n^{100} to compute. (This is because there is an "easy" function f that takes input of length n and produces output of length n^2. Composing this function with itself takes time n^4, etc.) At one level, it is clearly absurd to call a function "easy" if it requires time n^{100} to compute. However, this is precisely what we do in complexity theory! When our goal is to show that certain problems require *superpolynomial* running times, it is safe to consider a preprocessing step requiring time n^{100} as a "negligible factor".

A polynomial-time reduction f is simply a function that is computed by some program that runs in time bounded by $p(n)$ on inputs of length n, for some polynomial p. (That is, for some constant k, the running time of the program computing f is at most $n^k + k$ on inputs of length n.) Note that we have not been specific about the programming language in which the program is written. Traditionally, this is made precise by saying that f is computed by a *Turing machine* in the given time bound, but exactly the same class of polynomial-time reductions results, if we use any other reasonable programming language, with the usual notion of running time. This is a side-benefit of our overly generous definition of what it means to be "easy" to compute.

If f is a polynomial-time reduction of A to B, we denote this $A \leq_m^p B$. Note that this suggests an ordering, where B is "larger" (i.e., "harder to compute") than A. Any efficient algorithm for B yields an efficient algorithm for A; if A is hard to compute, then B must also be hard to compute. If $A \leq_m^p B$ and $B \leq_m^p A$, this is denoted $A \equiv_m^p B$.

One thing that makes complexity theory useful is that naturally-arising computational problems tend to clump together into a shockingly small number of equivalence classes of the $\equiv_m^p$ relation. Many thousands of problems have been analyzed, and most of these fall into about a dozen equivalence classes, with perhaps another dozen classes picking up some other notably interesting groups of problems.

Many of these equivalence classes correspond to interesting time and space bounds. To explain this connection, first we need to talk about *complexity classes*.

COMPLEXITY CLASSES AND COMPLETE SETS

A *complexity class* is a set of problems that can be computed within certain resource bounds on some model of computation. For instance, P is the class of problems computable by programs that run in time at most $n^k + k$ for some constant k; "P" stands for "polynomial time." Another important complexity class is EXP: the set of problems computed by programs that run for time at most 2^{n^k+k} on inputs of length n. P and EXP are both defined in terms of time complexity. It is also interesting to bound the amount of memory used by programs; the classes PSPACE and EXPSPACE consist of the problems computed by programs whose space requirements are polynomial and exponential in the input size, respectively.

An important relationship exists between EXP and the game of checkers; this example is suitable to introduce the concepts of "hardness" and "completeness."

Checkers is played on an 8-by-8 grid. When the rules are adapted for play on a 10-by-10 grid, the game is known as "draughts" (which can also be played on boards of other sizes). Starting from any game position, there is an *optimal strategy*. The task of finding an optimal strategy is a natural computational problem. $N \times N$-*Checkers* is the function that takes as input a description of a $N \times N$ draughts board with locations of the pieces, and returns as output the move that a given player should make, using the optimal strategy. It is known that there is a program computing the optimal strategy for $N \times N$-*Checkers* that runs in time exponential in N^2; thus, $N \times N$-*Checkers* $\in$ EXP.

More interestingly, it is known that for every problem $A \in \mathrm{EXP}$, $A \leq_m^p N \times N$-*Checkers*. We say that $A \leq_m^p N\times$ N-*Checkers* is *hard* for EXP (3).

More generally, if $\mathcal{C}$ is any class of problems, and B is a problem such that $A \leq_m^p B$ for every $B \in \mathcal{C}$, then we say that B is hard for $\mathcal{C}$. If B is hard for $\mathcal{C}$ and $B \in \mathcal{C}$, then we say that B is *complete* for $\mathcal{C}$. Thus, in particular, $N \times N$-*Checkers* is complete for EXP. This means that the complexity of $N \times N$-*Checkers* is well understood, in the sense that the fastest program for this problem cannot be too much faster than the currently known program. Here is why: We know (via a diagonalization argument) that there is some problem A in EXP that cannot be computed by any program that runs in time asymptotically less than 2^n. As $N \times N$-*Checkers* is complete for EXP, we know there is a reduction from A to $N \times N$-*Checkers* computable in time n^k for some k, and thus $N \times N$-*Checkers* requires running time that is asymptotically at least $2^{n^{1/k}}$.

It is significant to note that this yields only an *asymptotic* lower bound on the time complexity of $N \times N$-*Checkers*. That is, it says that the running time of any program for this problem must be very slow *on large enough inputs*, but (in contrast to Theorem 1) it says *nothing* about whether this problem is difficult for a given fixed input size. For instance, it is still unknown whether there could be a handheld device that computes optimal strategies for 100 × 100-Checkers (although this seems very unlikely). To mimic the proof of Theorem 1, it would be necessary to show that there is a problem in EXP that requires large circuits. Such problems are known to exist in EXPSPACE; whether such problems exist in EXP is one of the major open questions in computational complexity theory.

The complete sets for EXP (such as $N \times N$-*Checkers*) constitute one of the important $\equiv_m^p$-equivalence classes; many other problems are complete for PSPACE and EXPSPACE (and of course every nontrivial problem that can be solved in polynomial time is complete for P under $\leq_m^p$ reductions). However, this accounts for only a few of the several $\equiv_m^p$-equivalence classes that arise when considering important computational problems. To understand these other computational problems, it turns out to be useful to consider *unrealistic* models of computation.

UNREALISTIC MODELS: NONDETERMINISTIC MACHINES AND THE CLASS NP

Nondeterministic machines *appear* to be a completely unrealistic model of computation; if one could *prove* this to be the case, one would have solved one of the most important open questions in theoretical computer science (and even in all of mathematics).

A *nondeterministic Turing machine* can be viewed as a program with a special "guess" subroutine; each time this subroutine is called, it returns a random bit, zero or one. Thus far, it sounds like an ordinary program with a random bit generator, which does not sound so unrealistic. The unrealistic aspect comes with the way that we define how the machine produces its output. We say that a nondeterministic machine *accepts* its input (i.e., it outputs one) if there is some sequence of bits that the "guess" routine could return that causes the machine to output one; otherwise it is said to *reject* its input. If we view the "guess" bits as independent coin tosses, then the machine rejects its input if and only if the probability of outputting one is zero; otherwise it accepts. If a nondeterministic machine runs for t steps, the machine can flip t coins, and thus, a nondeterministic machine can do the computational equivalent of finding a needle in a haystack: If there is even one sequence r of length t (out of 2^t possibilities) such that sequence r leads the machine to output one on input x, then the nondeterministic machine will accept x, and it does it in time t, rather than being charged time 2^t for looking at all possibilities.

A classic example that illustrates the power of nondeterministic machines is the Travelling Salesman Problem. The input consists of a labeled graph, with nodes (cities) and edges (listing the distances between each pair of cities), along with a bound B. The question to be solved is as follows: *Does there exist a cycle visiting all of the cities, having length at most B?* A nondeterministic machine can solve this quickly, by using several calls to the "guess" subroutine to obtain a sequence of bits r that can be interpreted as a list of cities, and then outputting one if r visits all of the cities, and the edges used sum up to at most B.

Nondeterministic machines can also be used to factor numbers; given an n-bit number x, along with two other numbers a and b with $a < b$, a nondeterministic machine can accept whether there is a factor of x that lies between a and b.

Of course, this nondeterministic program is of no use at all in trying to factor numbers or to solve the Traveling Salesman Problem on realistic computers. In fact, it is hard to imagine that there will ever be an efficient way to simulate a nondeterministic machine on computers that one could actually build. This is *precisely* why this model is so useful in complexity theory; the following paragraph explains why.

The class NP is the class of problems that can be solved by *nondeterministic* machines running in time at most $n^k + k$ on inputs of size n, for some constant k; NP stands for Nondeterministic Polynomial time. The Traveling Salesman Problem is one of many hundreds of very important computational problems (arising in many seemingly unrelated fields) that are *complete* for NP. Although it is more than a quarter-century old, the volume by Garey and Johnson (4) remains a useful catalog of NP-complete problems. The NP-complete problems constitute the most important $\equiv_m^p$-equivalence class whose complexity is unresolved. If any one of the NP-complete problems lies in P, then P = NP. As explained above, it seems much more likely that P is *not* equal to NP, which implies that any program solving any NP-complete problem has a worst-case running time greater than $n^{100{,}000}$ on all large inputs of length n.

Of course, even if P is not equal to NP, we would still have the same situation that we face with $N \times N$-*Checkers*, in that we would not be able to conclude that instances of some fixed size (say n = 1,000) are hard to compute. For that, we would seem to need the stronger assumption that there are problems in NP that require very large circuits; in fact, this is widely conjectured to be true.

Although it is conjectured that deterministic machines require exponential time to simulate nondeterministic machines, it is worth noting that the situation is very different when memory bounds are considered instead. A classic theorem of complexity theory states that a nondeterministic machine using space $s(n)$ can be simulated by a deterministic machine in space $s(n)^2$. Thus, if we define NPSPACE and NEXPSPACE by analogy to PSPACE and EXPSPACE using nondeterministic machines, we obtain the equalities PSPACE = NPSPACE and EXPSPACE = NEXPSPACE.

We thus have the following six complexity classes:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP} \subseteq \mathrm{NEXP} \subseteq \mathrm{EXPSPACE}$$

Diagonalization arguments tell us that $\mathrm{P} \neq \mathrm{EXP}$, $\mathrm{NP} \neq \mathrm{NEXP}$, and $\mathrm{PSPACE} \neq \mathrm{EXPSPACE}$. All other relationships are unknown. For instance, it is unknown whether P = PSPACE, and it is also unknown whether PSPACE = NEXP (although at most one of these two equalities can hold). Many in the community conjecture that all of these classes are distinct, and that no significant improvement on any of these inclusions can be proved. (That is, many people conjecture that there are problems in NP that require exponential time on deterministic machines, that there are problems in PSPACE that require exponential time on nondeterministic machines, that there are problems in EXP that require exponential space, etc.) These conjectures have remained unproven since they were first posed in the 1970s.

A THEORY TO EXPLAIN OBSERVED DIFFERENCES IN COMPLEXITY

It is traditional to draw a distinction between mathematics and empirical sciences such as physics. In mathematics, one starts with a set of assumptions and derives (with certainty) the consequences of the assumptions. In contrast, in a discipline such as physics one starts with external reality and formulates theories to try to explain (and make predictions about) that reality.

For some decades now, the field of computational complexity theory has dwelt in the uncomfortable region between mathematics and the empirical sciences. Complexity theory is a mathematical discipline; progress is measured by the strength of the theorems that are proved. However, despite rapid and exciting progress on many fronts, the fundamental question of whether P is equal to NP remains unsolved.

Until that milestone is reached, complexity theory can still offer to the rest of the computing community some of the benefits of an empirical science, in the following sense. All of our observations thus far indicate that certain problems (such as the Traveling Salesman Problem) are intractible. Furthermore, we can observe that with surprisingly few exceptions, natural and interesting computational problems can usually be shown to be complete for one of a handful of well-studied complexity classes. Even though we cannot currently *prove* that some of these complexity classes are distinct, the fact that these complexity classes correspond to natural or unnatural models of computation gives us an intuitively appealing explanation for *why* these classes appear to be distinct. That is, complexity theory gives us a vocabulary and a set of plausible conjectures that helps explain our observations about the differing computational difficulty of various problems.

NP AND PROVABILITY

There are important connections between NP and mathematical logic. One equivalent way of defining NP is to say that a set A is in NP if and only if there are short *proofs* of membership in A. For example, consider the Traveling Salesman Problem. If there is a short cycle that visits all cities, then there is a short *proof* of this fact: Simply present the cycle and compute its length. Contrast this with the task of trying to prove that there is *not* a short cycle that visits all cities. For certain graphs this is possible, but nobody has found a general approach that is significantly better than simply listing all (exponentially many) possible cycles, and showing that all of them are too long. That is, for NP-complete problems A, it seems to be the case that the complement of A (denoted co-A) is not in NP.

The complexity class coNP is defined to be the set of all complements of problems in NP; $\mathrm{coNP} = \{\mathrm{co} - A : A \in \mathrm{NP}\}$. This highlights what appears to be a fundamental difference between deterministic and nondeterministic computation. On a deterministic machine, a set and its complement always have similar complexity. On nondeterministic machines, this does not appear to be true (although if one could prove this, one would have a proof that P is different from NP).

To discuss the connections among NP, coNP, and logic in more detail, we need to give some definitions related to propositional logic. A propositional logic formula consists of variables (which can take on the values True and False), along with the connectives And, Or, and Not. A formula is said to be *satisfiable* if there is some assignment of truth values to the variables that causes it to evaluate to True; it is said to be a *tautology* if *every* assignment of truth values to the variables causes it to evaluate to True. SAT is the set of all satisfiable formulas; TAUT is the set of all tautologies. Note that the formula ϕ is in SAT if and only if "Notϕ" is not in TAUT.

SAT is complete for NP; TAUT is complete for coNP. [This famous theorem is sometimes known as "Cook's Theorem" (5) or the "Cook-Levin Theorem" (6).]

Logicians are interested in the question of how to *prove* that a formula is a tautology. Many proof systems have been developed; they are known by such names as *resolution*, *Frege systems*, and *Gentzen calculus*. For some of these systems, such as resolution, it is known that certain tautologies of n symbols require proofs of length nearly 2^n(7). For Frege systems and proofs in the Gentzen calculus, it is widely suspected that similar bounds hold, although this remains unknown. Most logicians suspect that for *any* reasonable proof system, some short tautologies will require very long proofs. This is equivalent to the conjecture that NP and coNP are different classes; if every tautology had a short proof, then a nondeterministic machine could "guess" the proof and accept if the proof is correct. As TAUT is complete for coNP, this would imply that NP = coNP.

The P versus NP question also has a natural interpretation in terms of logic. Two tasks that occupy mathematicians are as follows:

1. Finding proofs of theorems.
2. Reading proofs that other people have found.

Most mathematicians find the second task to be considerably simpler than the first one. This can be posed as a computational problem. Let us say that a mathematician wants to prove a theorem ϕ and wants the proof to be at most 40 pages long. A nondeterministic machine can take as input ϕ followed by 40 blank pages, and "guess" a proof, accepting if it finds a legal proof. If P = NP, the mathematician can thus determine fairly quickly whether there is a short proof. A slight modification of this idea allows the mathematician to efficiently *construct* the proof (again, assuming that P = NP). That is, the conjecture that P is different than NP is consistent with our intuition that finding proofs is more difficult than verifying that a given proof is correct.

In the 1990s researchers in complexity theory discovered a very surprising (and counterintuitive) fact about logical proofs. Any proof of a logic statement can be *encoded* in such a way that it can be verified by picking a few bits at random and checking that these bits are sufficiently consistent. More precisely, let us say that you want to be 99.9% sure that the proof is correct. Then there is some constant k and a procedure such that, no matter how long the proof is, the procedure flips $O(\log n)$ coins and picks k bits of the encoding of the proof, and then does some computation, with the property that, if the proof is correct, the procedure accepts with probability one, and if the proof is incorrect, then the procedure detects that there is a flaw with probability at least 0.999. This process is known as a *probabilistically checkable proof*. Probabilistically checkable proofs have been very useful in proving that, for many optimization algorithms, it is NP-complete not only to find an optimal solution, but even to get a very rough approximation to the optimal solution.

Some problems in NP are widely believed to be intractable to compute, but are not believed to be NP-complete. Factoring provides a good example. The problem of computing the prime factorization of a number can be formulated in several ways; perhaps the most natural way is as the set FACTOR = {(x, i, b): the ith bit of the encoding of the prime factorization of x is b}. By making use of the fact that primality testing lies in P (8), set FACTOR is easily seen to lie in NP ∩ coNP. Thus, FACTOR cannot be NP-complete unless NP = coNP.

OTHER COMPLEXITY CLASSES: COUNTING, PROBABILISTIC, AND QUANTUM COMPUTATION

Several other computational problems appear to be intermediate in complexity between NP and PSPACE that are related to the problem of *counting* how many accepting paths a nondeterministic machine has. The class #P is the class of functions f for which there is an NP machine M with the property that, for each string x, $f(x)$ is the number of guess sequences r that cause M to accept input x. #P is a class of *functions*, instead of being a class of *sets* like all other complexity classes that we have discussed. #P is equivalent in complexity to the class PP (probabilistic polynomial time) defined as follows. A set A is in PP if there is an NP machine M such that, for each string x, x is in A if and only if more than half of the guess sequences cause M to accept x. If we view the guess sequences as flips of a fair coin, this means that x is in A if and only the probability that M accepts x is greater than one half. It is not hard to see that both NP and coNP are subsets of PP; thus this is not a very "practical" notion of probabilistic computation.

In practice, when people use probabilistic algorithms, they want to receive the correct answer with *high* probability. The complexity class that captures this notion is called BPP (bounded-error probabilistic polynomial time). Some problems in BPP are not known to lie in P; a good example of such a problem takes two algebraic circuits as input and determines whether they compute the same function.

Early in this article, we mentioned quantum computation. The class of problems that can be solved in polynomial time with low error probability using quantum machines is called BQP (bounded-error quantum polynomial time). FACTOR (the problem of finding the prime factorization of a number) lies in BQP (2). The following inclusions are known:

$$\mathrm{P} \subseteq \mathrm{BPP} \subseteq \mathrm{BQP} \subseteq \mathrm{PP} \subseteq \mathrm{PSPACE}$$
$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PP}$$

No relationship is known between NP and BQP or between NP and BPP. Many people conjecture that neither NP nor BQP is contained in the other.

In contrast, many people now conjecture that BPP = P, because it has been proved that if there is any problem computable in time 2^n that requires circuits of nearly exponential size, then there is an efficient deterministic simulation of any BPP algorithm, which implies that P = BPP (9). This theorem is one of the most important in a field that has come to be known as *derandomization*, which studies how to simulate probabilistic algorithms deterministically.

INSIDE P

Polynomial-time reducibility is a very useful tool for clarifying the complexity of seemingly intractible problems, but it is of no use at all in trying to draw distinctions among problems in P. It turns out that some very useful distinctions can be made; to investigate them, we need more refined tools.

Logspace reducibility is one of the most widely used notions of reducibility for investigating the structure of P; a logspace reduction f is a polynomial-time reduction with the additional property that there is a Turing machine computing f that has (1) a read-only input tape, (2) a write-only output tape, and (3) the only other data structure it can use is a read/write worktape, where it uses only $O(\log n)$ locations on this tape on inputs of length n. If A is logspace-reducible to B, then we denote this by $A \leq_m^{log} B$. Imposing this very stringent memory restriction seems to place severe limitations on polynomial-time computation; many people conjecture that many functions computable in polynomial time are not logspace-computable. However, it is also true that the full power of polynomial time is not exploited in most proofs of NP-completeness. For essentially all natural problems that have been shown to be complete for the classes NP, PP, PSPACE, EXP, and so on using polynomial-time reducibility, it is known that they are also complete under logspace reducibility. That is, for large classes, logspace reducibility is essentially as useful as polynomial-time reducibility, but logspace reducibility offers the advantage that it can be used to find distinctions among problems in P.

Logspace-bounded Turing machines give rise to some natural complexity classes inside P: If the characteristic function of a set A is a logspace reduction as defined in the preceding paragraph, then A lies in the complexity class L. The analogous class, defined in terms of nondeterministic machines, is known as NL. The class #P also has a logspace analog, known as #L. These classes are of interest primarily because of their complete sets. Some important complete problems for L are the problem of determining whether two trees are isomorphic, testing whether a graph can be embedded in the plane, and the problem of determining whether an undirected graph is connected (10). Determining whether a *directed* graph is connected is a standard complete problem for NL, as is the problem of computing the length of the shortest path between two vertices in a graph. The complexity class #L characterizes the complexity of computing the determinant of an integer matrix as well as several other problems in linear algebra.

There are also many important complete problems for P under logspace reducibility, such as the problem of evaluating a Boolean circuit, linear programming, and certain network flow computations. In fact, there is a catalog of P-complete problems (11) that is nearly as impressive as the list of NP-complete problems (4). Although many P-complete problems have very efficient algorithms in terms of time complexity, there is a sense in which they seem to be resistent to extremely fast parallel algorithms. This is easiest to explain in terms of circuit complexity. The *size* of a Boolean circuit can be measured in terms of either the number of gates or the number of wires that connect the gates. Another important measure is the *depth* of the circuit: the length of the longest path from an input gate to the output gate. The problems in L, NL, and #L all have circuits of polynomial size and very small depth ($O(\log^2 n)$). In contrast, all polynomial-size circuits for P-complete problems seem to require a depth of at least $n^{1/k}$.

Even a very "small" complexity class such as L has an interesting structure inside it that can be investigated using a more restricted notion of reducibility than $\leq_m^{log}$ that is defined in terms of very restricted circuits. Further information about these small complexity classes can be found in the textbook by Vollmer (12).

We have the inclusions $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PP} \subseteq \mathrm{PSPACE}$. Diagonalization shows that $\mathrm{NL} \neq \mathrm{PSPACE}$, but no other separations are known. In particular, it remains unknown whether the "large" complexity class PP actually coincides with the "small" class L.

TIME-SPACE TRADEOFFS

Logspace reducibility (and in general the notion of Turing machines that have very limited memory resources) allows the investigation of another aspect of complexity: the tradeoff between time and space. Take, for example, the problem of determining whether an undirected graph is connected. This problem can be solved using logarithmic space (10), but currently all "space-efficient" algorithms that are known for this problem are so slow that they will never be used in practice, particularly because this problem can be solved in linear time (using linear space) using a standard depth-first-search algorithm. However, there is no strong reason to believe that no fast small-space algorithm for graph connectivity *exists* (although there have been some investigations of this problem, using "restricted" models of computation, of the type that were discussed at the start of this article).

Some interesting time-space tradeoffs have been proved for the NP-complete problem SAT. Recall that it is still unknown whether SAT lies in the complexity class L. Also, although it is conjectured that SAT is not solvable in time n^k for any k, it remains unknown whether SAT is solvable in time $O(n)$. However, it *is* known that if SAT *is* solvable in linear time, then any such algorithm must use much more than logarithmic space. In fact, any algorithm that solves SAT in time $n^{1.7}$ must use memory $n^{1/k}$ for some k(13,14).

CONCLUSION

Computational complexity theory has been very successful in providing a framework that allows us to understand why several computational problems have resisted all efforts to find efficient algorithms. In some instances, it has been possible to prove very strong intractibility theorems, and in many other cases, a widely believed set of conjectures explains why certain problems appear to be hard to compute. The field is evolving rapidly; several developments discussed here are only a few years old. Yet the central questions (such as the infamous P vs. NP question) remain out of reach today.

By necessity, a brief article such as this can touch on only a small segment of a large field such as computational complexity theory. The reader is urged to consult the texts listed below, for a more comprehensive treatment of the area.

FURTHER READING

D.-Z. Du and K.-I. Ko, *Theory of Computational Complexity*. New York: Wiley, 2000.

L. A. Hemaspaandra and M. Ogihara, *The Complexity Theory Companion*. London: Springer-Verlag, 2002.

D. S. Johnson, A catalog of complexity classes, in J. van Leeuwen, (ed.), *Handbook of Theoretical Computer Science, Vol. A:*

Algorithms and Complexity. Cambridge, MA: MIT Press, 1990, pp. 69–161.

D. Kozen, *Theory of Computation*. London: Springer-Verlag, 2006.

C. Papadimitriou, *Computational Complexity*. Reading, MA: Addison-Wesley, 1994.

I. Wegener, *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Berlin: Springer-Verlag, 2005.

BIBLIOGRAPHY

1. L. Stockmeyer and A. R. Meyer, Cosmological lower bound on the circuit complexity of a small problem in logic, *J. ACM*, **49**: 753–784, 2002.
2. P. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Comput*. **26**: 1484–1509, 1997.
3. J. M. Robson, N by N Checkers is EXPTIME complete, *SIAM J. Comput.*13: 252–267, 1984.
4. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
5. S. Cook, The complexity of theorem proving procedures, *Proc. 3rd Annual ACM Symposium on Theory of Computing (STOC)*, 1971, pp. 151–158.
6. L. Levin, Universal search problems, *Problemy Peredachi Informatsii*, **9**: 265–266, 1973 (in Russian). English translation: B. A. Trakhtenbrot, A survey of Russian approaches to perebor (brute-force search) algorithms, *Ann. History Comput.*, **6**: 384–400, 1984.
7. A. Haken, The intractability of resolution, *Theor. Comput. Sci.*, **39**: 297–308, 1985.
8. M. Agrawal, N. Kayal, and N. Saxena, PRIMES is in P, *Ann. Math.*, **160**: 781–793, 2004.
9. R. Impagliazzo and A. Wigderson, P=BPP unless E has subexponential circuits, *Proc. 29th ACM Symposium on Theory of Computing (STOC)*, 1997, pp. 220–229.
10. O. Reingold, Undirected ST-connectivity in log-space, *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, 2005, pp. 376–385.
11. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo, *Limits to Parallel Computation: P-Completeness Theory*. New York: Oxford University Press, 1995.
12. H. Vollmer, *Introduction to Circuit Complexity*. Berlin: Springer-Verlag, 1999.
13. L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas, Time-space lower bounds for satisfiability, *J. ACM*, **52**: 835–865, 2005.
14. R. Williams, Better time-space lower bounds for SAT and related problems, *Proc. 20th Annual IEEE Conference on Computational Complexity (CCC)*, 2005, pp. 40–49.

ERIC ALLENDER
Rutgers University
Piscataway, New Jersey

C

COMPUTATIONAL NUMBER THEORY

In the contemporary study of mathematics, number theory stands out as a peculiar branch, for many reasons. Most development of mathematical thought is concerned with the identification of certain structures and relations in these structures. For example, the study of algebra is concerned with different types of operators on objects, such as the addition and multiplication of numbers, the permutation of objects, or the transformation of geometric objects—and the study of algebra is concerned with the classification of the many such types of operators.

Similarly, the study of analysis is concerned with the properties of operators that satisfy conditions of continuity.

Number theory, however, is the study of the properties of those few systems that arise naturally, beginning with the natural numbers (which we shall usually denote $\mathbb{N}$), progressing to the integers ($\mathbb{Z}$), the rationals ($\mathbb{Q}$), the reals ($\mathbb{R}$), and the complex numbers ($\mathbb{C}$). Rather than identifying very general principles, number theory is concerned with very specific questions about these few systems.

For that reason, for many centuries, mathematicians thought of number theory as the purest form of inquiry. After all, it was not inspired by physics or astronomy or chemistry or other "applied" aspects of the physical universe. Consequently, mathematicians could indulge in number theoretic pursuits while being concerned only with the mathematics itself.

But number theory is a field with great paradoxes. This purest of mathematical disciplines, as we will see below, has served as the source for arguably the most important set of applications of mathematics in many years!

Another curious aspect of number theory is that it is possible for a very beginning student of the subject to pose questions that can baffle the greatest minds. One example is the following: It is an interesting observation that there are many triples of natural numbers (in fact, an infinite number) that satisfy the equation $x^2 + y^2 = z^2$. For example, $3^2 + 4^2 = 5^2$, $5^2 + 12^2 = 13^2$, $7^2 + 24^2 = 25^2$, and so on.

However, one might easily be led to the question, can we find (nonzero) natural numbers x, y, and z such that $x^3 + y^3 = z^3$? Or, indeed, such that $x^n + y^n = z^n$, for any natural number $n > 2$ and nonzero integers x, y and z?

The answer to this simple question was announced by the famous mathematician, Pierre Auguste de Fermat, in his last written work in 1637. Unfortunately, Fermat did not provide a proof but only wrote the announcement in the margins of his writing. Subsequently, this simply stated problem became known as "Fermat's Last Theorem," and the answer eluded the mathematics community for 356 years, until 1993, when it was finally solved by Andrew Wiles (1).

The full proof runs to over 1000 pages of text (no, it will not be reproduced here) and involves mathematical techniques drawn from a wide variety of disciplines within mathematics. It is thought to be highly unlikely that Fermat, despite his brilliance, could have understood the true complexity of his "Last Theorem." (Lest the reader leave for want of the answer, what Wiles proved is that there are no possible nonzero x, y, z, and $n > 2$ that satisfy the equation.)

Other questions that arise immediately in number theory are even more problematic than Fermat's Last Theorem. For example, a major concern in number theory is the study of prime numbers—those natural numbers that are evenly divisible only by themselves and 1. For example, 2, 3, 5, 7, 11, and 13 are prime numbers, whereas 9, 15, and any even number except 2 are not (1, by convention, is not considered a prime).

One can easily observe that small even numbers can be described as the sum of two primes: $2 + 2 = 4$, $3 + 3 = 6$, $3 + 5 = 8$, $3 + 7 = 5 + 5 = 10$, $5 + 7 = 12$; $7 + 7 = 14$, and so on. One could ask, can all even numbers be expressed as the sum of two primes? Unfortunately, no one knows the answer to this question. It is known as the *Goldbach Conjecture,* and fame and fortune (well, fame, anyway) await the person who successfully answers the question (2).

In this article, we will attempt to describe some principal areas of interest in number theory and then to indicate what current research has shown to be an extraordinary application of this purest form of mathematics to several very current and very important applications.

Although this will in no way encompass all of the areas of development in number theory, we will introduce:

Divisibility: At the heart of number theory is the study of the multiplicative structure of the integers under multiplication. What numbers divide (i.e., are factors of) other numbers? What are all the factors of a given number? Which numbers are prime?

Multiplicative functions: In analyzing the structure of numbers and their factors, one is led to the consideration of functions that are *multiplicative*: In other words, a function is multiplicative if $f(a \times b) = f(a) \times f(b)$ for all a and b.

Congruence: Two integers a and b are said to be congruent modulo n (where n is also an integer), and written $a \equiv b \pmod{n}$, if their difference is a multiple of n; alternatively, that a and b yield the same remainder when divided (integer division) by n. The study of the integers under congruence yields many interesting properties and is fundamental to number theory. The modular systems so developed are called modulo n arithmetic and are denoted either $\mathbb{Z}/n\mathbb{Z}$or$\mathbb{Z}_n$.

Residues: In $\mathbb{Z}_n$ systems, solutions of equations (technically, *congruences*) of the form $x^2 \equiv a \pmod{n}$ are often studied. In this instance, if there is a solution for x, a is called a quadratic residue of n. Otherwise, it is called a quadratic nonresidue of n.

Prime numbers: The prime numbers, with their special property that they have no positive divisors other than themselves and 1, have been of continuing interest to number theorists. In this section, we will see, among other things, an estimate of how many prime numbers there are less than some fixed number n.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Diophantine equations: The term *Diophantine equation* is used to apply to a family of algebraic equations in a number system such as $\mathbb{Z}$ or $\mathbb{Q}$. A good deal of research in this subject has been directed at polynomial equations with integer or rational coefficients, the most famous of which being the class of equations $x^n + y^n = z^n$, the subject of Fermat's Last Theorem.

Elliptic curves: A final area of discussion in number theory will be the theory of elliptic curves. Although generally beyond the scope of this article, this theory has been so important in contemporary number theory that some discussion of the topic is in order. An elliptic curve represents the set of points in some appropriate number system that are the solutions to an equation of the form $y^2 = \mathrm{A}x^3 + \mathrm{B}x^2 + \mathrm{C}x + \mathrm{D}$ when A, B, C, D $\in \mathbb{Z}$.

Applications: The final section of this article will address several important applications in business, economics, engineering, and computing of number theory. It is remarkable that this, the purest form of mathematics, has found such important applications, often of theory that is hundreds of years old, to very current problems in the aforementioned fields!

It should perhaps be noted here that many of the results indicated below are given without proof. Indeed, because of space limitations, proofs will only be given when they are especially instructive. Several references will be given later in which proofs of all of the results cited can be found.

DIVISIBILITY

Many questions arising in number theory have as their basis the study of the divisibility of the integers. An integer n is divisible by k if another integer m exists such that $k \times m = n$. We sometimes indicate divisibility of n by k by writing $k|n$ or $k|n$ if n is not divisible by k.

A fundamental result is the division algorithm. Given m, $n \in \mathrm{Z}$, with $n > 0$, unique integers c and d exist such that $m = c \times n + d$ and $0 \leq d < n$.

Equally as fundamental is the Euclidean algorithm.

Theorem 1. Let $m, n \in \mathbb{Z}$, both $m, n \neq 0$. A unique integer c exists satisfying $c > 0, c|m, c|n$; and if $d|m$ and $d|n$, then $d|c$.

Proof. Consider $\{d | d = am + bn, \forall\ a, b \in \mathbb{Z}\}$. Let c^* be the smallest natural number in this set. Then c^* satisfies the given conditions.

Clearly $c^* > 0. c^*|a$, because by the division algorithm, s and t there exist such that $a = cs + t$ with $0 \leq t < u$. Thus, $a = c^*s + t = ams + bns + t$; thus, $a(1{-}ms) + b(-ns) = t$. As $t < c^*$, this implies that $t = 0$, and thus $a = c^*s$ or $c^* \mid a$. Similarly $c^* \mid b$. c^* is unique because, if c' also meets the conditions, then $c^* \mid c$' and c' $\mid c^*$, so c' $= c^*$.

The greatest common divisor of two integers m and n is the largest positive integer [denoted GCD(m,n)] such that GCD(m,n) $\mid m$ and GCD (m,n) $\mid n$.

Theorem 2 (Greatest Common Divisor). The equation $am + bn = r$ has integer solutions $a, b \Leftrightarrow \mathrm{GCD}(m, n)|r$.

Proof. Let $r \neq 0$. It is not possible that GCD(m,n) $\mid r$ because GCD(m,n) $\mid m$ and GCD(m,n) $\mid n$; thus, GCD(m,n) $\mid (am + bn) = r$. By Theorem 1, this means that there exist a', b' such that $a'm + b'n = \mathrm{GCD}(m,n)$.

Thus, $a'' = ra'/\mathrm{GCD}(m,n), b'' = rb'/\mathrm{GCD}(m,n)$ represent an integer solution to $a''m + b''n = r$.

A related concept to the GCD is the least common multiple (LCM). It can be defined as the smallest positive integer that a and b both divide. It is also worth noting that $\mathrm{GCD}(m,n) \times \mathrm{LCM}(m,n) = m \times n$.

Primes

A positive integer $p > 1$ is called prime if it is divisible only by itself and 1. An integer greater than 1 and not prime is called composite. Two numbers m and n with the property that GCD(m,n) = 1 are said to be relatively prime (or sometimes coprime).

Here are two subtle results.

Theorem 3. Every integer greater than 1 is a prime or a product of primes.

Proof. Suppose otherwise. Let n be the least integer that is neither; thus, n is composite. Thus $n = ab$ and a,$b < n$. Thus, a and b are either primes or products of primes, and thus so is their product, n.

Theorem 4. There are infinitely many primes.

Proof (Euclid). If not, let $p_1, \ldots, p_n$ be a list of all the primes, ordered from smallest to largest. Then consider $q = (p_1 p_2 \ldots p_n) + 1$. By the previous statement,

$$q = p'_1 p'_2 \ldots p'_k \tag{1}$$

p'_1 must be one of the $p_1, \ldots, p_n$, say p_j (as these were all the primes), but $p_j \,|\, q$, because it has a remainder of 1. This contradiction proves the theorem.

Theorem 5 (Unique Factorization). Every positive integer has a unique factorization into a product of primes.

Proof. Let

$$\begin{aligned} a &= p_1^{\alpha_1} \ldots p_k^{\alpha_k} = P \\ &= q_1^{\beta_1} \ldots q_m^{\beta_m} = Q \end{aligned} \tag{2}$$

For each p_i, $p_i \mid Q \Rightarrow p_i \mid q_s^{\beta_s}$ for some s, $1 \leq s \leq m$.

Since p_i and q_s are both primes, $p_i = q_s$. Thus, $k = m$ and $Q = p_1^{\beta_1} \ldots p_k^{\beta_k}$. We only need to show that the α_i and β_i are equal. Divide both decompositions P and Q by $p_i^{\alpha_i}$. If $\alpha_i \neq \beta_i$, on the one hand, one decomposition $a/p_i^{\alpha_i}$ will contain p_i, and the other will not. This contradiction proves the theorem.

What has occupied many number theorists was a quest for a formula that would generate all of the infinitely many prime numbers.

For example, Marin Mersenne (1644) examined numbers of the form $M_p = 2^p - 1$, where p is a prime (3). He discovered that some of these numbers were, in fact, primes. Generally, the numbes he studied are known as Mersenne numbers, and those that are prime are called Mersenne primes. For example, $M_2 = 2^2 - 1 = 3$ is prime; as are $M_3 = 2^3 - 1 = 7$; $M_5 = 2^5 - 1 = 31$; $M_7 = 2^7 - 1 = 127$. Alas, $M_{11} = 2^{11} - 1 = 2047 = 23 \times 89$ is not. Any natural number ≥ 2 ending in an even digit 0, 2, 4, 6, 8 is divisible by 2. Any number ≥ 5 ending in 5 is divisible by 5. There are also convenient tests for divisibility by 3 and 9—if the sum of the digits of a number is divisible by 3 or 9, then so is the number; and by 11—if the sum of the digits in the even decimal places of a number, minus the sum of the digits in the odd decimal places, is divisible by 11, then so is the number.

Several other Mersenne numbers have also been determined to be prime. At the current writing, the list includes 42 numbers:

M_n, where $n = 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503, 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593, 13466917, 20996011, 24036583, 25964951$.

With current technology, particularly mathematical software packages such as *Mathematica* or *Maple*, many of these computations can be done rather easily. For example, a one-line program in Mathematica, executing for 6.27 hours on a Pentium PC, was capable of verifying the primality of all M_n up to M_{1000}, and thus determining the first 14 Mersenne primes (up to M_{607}). It is not known whether an infinite number of the Mersenne numbers are prime.

It is known that $m^c - 1$ is composite if $m > 2$ or if c is composite; for if $c = de$, we have:

$$(m^c - 1) = (m^e - 1)(m^{e(d-1)} + m^{e(d-1)} + \ldots + m^{e(d-1)} + 1) \tag{3}$$

We can also show that $m^c + 1$ is composite if m is odd or if c has an odd factor. Certainly if m is odd, m^c is odd, and $m^c + 1$ is even and thus composite. If $c = d(2k + 1)$, then

$$m^c + 1 = (m^d + 1)(m^{2de} - m^{d(2e-1)} + m^{d(2e-2)} - \ldots + 1) \tag{4}$$

and $m^d + 1 > 1$.

Another set of numbers with interesting primality properties are the Fermat numbers, $F_n = 2^{2^n} + 1$. Fermat's conjecture was that they were primes. He was able to verify this for $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, and $F_4 = 2^{16} + 1 = 65537$. But then, Euler showed that

Theorem 6. $641 \mid F_5$.

Proof. $641 = 2^4 + 5^4 = 5 \times 2^7 + 1$; thus $2^4 = 641 - 5^4$. Since $2^{32} = 2^4 \times 2^{28} = 641 \times 2^{28} - (5 \times 2^7)^4 = 641 \times 2^{28} - (641 - 1)^4 = 641\text{k} - 1$.

To this date, no other Fermat numbers F_n with $n > 4$ have been shown to be prime. It has been determined that the other Fermat numbers through F_{20} are composite.

MULTIPLICATIVE FUNCTIONS

Functions that preserve the multiplicative structure of the number systems studied in number theory are of particular interest, not only intrinsically, but also for their use in determining other relationships among numbers.

A function f defined on the integers that takes values in a set closed under multiplication is called a *number theoretic function*. If the function preserves multiplication for numbers that are relatively prime ($f(m) \times f(n) = f(m \times n)$), it is called a *multiplicative function*; it is called *completely multiplicative* if the restriction $\text{GCD}(m, n) = 1$ can be lifted.

Consider any number theoretic function f, and define a new function F by the sum of the values of f taken over the divisors of n

$$F(n) = \sum_{d|n} f(d) = \sum_{d|n} f(n/d), \text{ the latter being the former sum in reverse order} \tag{5}$$

Theorem 7. If f is a multiplicative function, then so is F.

Proof. Let $\text{GCD}(m, n) = 1$; then $d \mid mn$ can be uniquely expressed as gh, where $g \mid m, h \mid n$, with $\text{GCD}(g,h) = 1$.

$$\begin{aligned} F(mn) &= \sum_{d|mn} f(d) = \sum_{g|m}\sum_{h|n} f(gh) = \sum_{g|m}\sum_{h|n} f(g)\,f(h) \\ &= \sum_{g|m} f(g)\,f(h_1) + \ldots + \sum_{g|m} f(g)\,f(h_k) \\ &= F(m)\left(\sum_{h|n} f(h)\right) = F(m)F(n) \end{aligned} \tag{6}$$

Two multiplicative functions of note are the divisor function $\tau(n)$ defined as the number of positive divisors of n and the function $\sigma(n)$, defined as the sum of the positive divisors of n.

Theorem 8. τ and σ are multiplicative.

Proof. Both τ and σ are the "uppercase" functions for the obviously multiplicative functions $1(n) = 1$ for all n, and $i(n) = n$ for all n. In other words,

$$\tau(n) = \sum_{d|n} 1(d) \quad \text{and} \quad \sigma(n) = \sum_{d|n} i(d) \tag{7}$$

In order to compute τ and σ for a prime p, note that

$$\tau(p^n) = 1 + n (\text{because the divisors of } p^n \text{ are } 1, p, p^2, \ldots, p^n) \tag{8}$$

$$\sigma(p^n) = 1 + p + p^2 + \ldots + p^n = (p^{n+1} - 1)/(p - 1) \tag{9}$$

Thus, for any $n = p_1a^{n_1}\, p_2a^{n_2} \ldots p_ka^{n_k}$,

$$\tau(n) = \prod_{i=1}^{k}(1 + n_i) \quad \text{and} \quad \sigma(n) = \prod_{i=1}^{k}(p_i^{n_i+1} - 1)/(p_i - 1) \tag{10}$$

In very ancient times, numbers were sometimes considered to have mystical properties. Indeed, the Greeks identified numbers that they called perfect: numbers that were exactly the sums of all their proper divisors, in other words, that $\sigma(n) = n$. A few examples are $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$, and $496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248$. It is not known whether there are any odd perfect numbers, or if an infinite number of perfect numbers exists.

Theorem 9. n is even and perfect $\Leftrightarrow n = 2^{p-1}(2^p - 1)$ where both p and $2^p - 1$ are primes.

In other words, there is one perfect number for each Mersenne prime.

Another multiplicative function of considerable importance is the Möbius function μ.

$\mu(1) = 1$; $\mu(n) = 0$ if n has a square factor; $\mu(p_1, p_2 \ldots p_k) = (-1)^k$ if $p_1, \ldots, p_k$ are distinct primes.

Theorem 10. μ is multiplicative, and the “uppercase” function is 0 unless $n = 1$, when it takes the value 1.

Theorem 11. (Möbius Inversion Formula). If f is a number theoretic function and $F(n) = \sum_{d|n} f(d)$, then

$$f(n) = \sum_{d/n} F(d)\mu(n/d) = \sum_{d/n} F(n/d)\mu(n) \qquad (11)$$

Proof.

$$\sum_{d/n} \mu(d)F(n/d)$$

$$= \sum_{d_1d_2=n} \mu(d_1)F(n/d_2) \text{ (taking pairs } d_1d_2 = n) \qquad (12)$$

$$= \sum_{d_1d_2=n} [\mu(d_1) \sum_{d|d_2} f(d)] \text{ (definition of } F) \qquad (13)$$

$$= \sum_{d_r d|n} \mu(d_1)\, f(d) \text{ (multiplying the terms in brackets)} \qquad (14)$$

$$= \sum_{d|n} f(d) \sum_{d_1|(n/d)} \mu(d_1) \text{ (collecting multiples of } f(d)) \qquad (15)$$

$$= f(n)$$

Theorem 12. If F is a multiplicative function, then so is f.

A final multiplicative function of note is the Euler function $\phi(n)$. It is defined for n to be the number of numbers less than n and relatively prime to n. For primes p, $\phi(p) = p-1$.

Theorem 13. ϕ is multiplicative.

CONGRUENCE

The study of congruence leads to the definition of new algebraic systems derived from the integers. These systems, called *residue systems*, are interesting in and of themselves, but they also have properties that allow for important applications.

Consider integers a, b, n with $n > 0$. Note that a and b could be positive or negative. We will say that *a is congruent to b, modulo n* [written $a \equiv b \pmod{n}$] $\Leftrightarrow n|(a - b)$. Alternatively, a and b yield the same remainder when divided by n. In such a congruence, n is called the modulus and b is called a residue of a.

An algebraic system can be defined by considering classes of all numbers satisfying a congruence with fixed modulus. It is observed that congruence is an equivalence relation and that the definition of addition and multiplication of integers can be extended to the equivalence classes. Thus, for example, in the system with modulus 5 (also called mod 5 arithmetic), the equivalence classes are $\{\ldots,-5,0,5,\ldots\}$, $\{\ldots,-4,1,6,\ldots\}$, $\{\ldots,-3,2,7,\ldots\}$, $\{\ldots,-2,3,8,\ldots\}$, and $\{\ldots,-1,4,9,\ldots\}$. It is customary to denote the class by the (unique) representative of the class between 0 and $n-1$. Thus the five classes in mod 5 arithmetic are denoted 0, 1, 2, 3, 4. Formally, the mod n system can be defined as the algebraic quotient of the integers $\mathbb{Z}$ and the subring defined by the multiples of n ($n\mathbb{Z}$). Thus, the mod n system is often written $\mathbb{Z}/n\mathbb{Z}$. An alternative, and more compact notation, is $\mathbb{Z}_n$.

Addition and multiplication are defined naturally in $\mathbb{Z}_n$. Under addition, every $\mathbb{Z}_n$ forms an Abelian group [that is, the addition operation is closed, associative, and commutative; 0 is an identity; and each element has an additive inverse—for any a, $b = n-a$ always yields $a + b \equiv 0 \pmod{n}$].

In the multiplicative structure, however, only the closure, associativity, commutativity, and identity (1) are assured. It is not necessarily the case that each element will have an inverse. In other words, the congruence $ax \equiv 1 \pmod{n}$ will not always have a solution.

Technically, an algebraic system with the properties described above is called a *commutative ring with identity*. If it is also known that, if each (nonzero) element of $\mathbb{Z}_n$ has an inverse, the system would be called a *field*.

Theorem 14. Let a, b, n be integers with $n > 0$. Then $ax \equiv 1 \pmod{n}$ has a solution $\Leftrightarrow$ GCD$(a, n) = 1$. If x_0 is a solution, then there are exactly GCD(a,n) solutions given by $\{x_0, x_0 + n/\text{GCD}(a,n), x_0 + 2n/\text{GCD}(a,n), \ldots, x_0 + (\text{GCD}(a,n) -1)n/\text{GCD}(a,n)\}$.

Proof. This theorem is a restatement of Theorem 2.

For an element a in $\mathbb{Z}_n$ to have an inverse, alternatively to be a unit, by the above theorem, it is necessary and sufficient for a and n to be relatively prime; i.e., GCD$(a,n) = 1$. Thus, by the earlier definition of the Euler function, the number of units in $\mathbb{Z}_n$ is $\phi(n)$.

The set of units in $\mathbb{Z}_n$ is denoted $(\mathbb{Z}_n)^*$, and it is easily verified that this set forms an Abelian group under multiplication. As an example, consider $\mathbb{Z}_{12}$ or $\mathbb{Z}/12\mathbb{Z}$. Note that $(Z_{12})^* = \{1,5,7,11\}$, and that each element is its own inverse: $1 \times 1 \equiv 5 \times 5 \equiv 7 \times 7 \equiv 11 \times 11 \equiv 1 \pmod{12}$. Further more, closure is observed because $5 \times 7 \equiv 11$, $5 \times 11 \equiv 7$, and $7 \times 11 \equiv 5 \pmod{12}$.

Theorem 15. If p is a prime number, then $\mathbb{Z}_p$ is a field with p elements. If n is composite, $\mathbb{Z}_n$ is not a field.

Proof. If p is a prime number, every element $a \in (\mathbb{Z}_p)^*$ is relatively prime to p; that is, GCD(a,p) = 1. Thus $ax \equiv 1$ (mod p) always has a solution. As every element in $(\mathbb{Z}_p)^*$ has an inverse, $(\mathbb{Z}_p)^*$ is a field. If n is composite, there are integers $1 < k,\ l < n$ such that $kl = n$. Thus, $kl \equiv 0$ (mod n), and so it is impossible that k could have an inverse. Otherwise, $l \equiv (k^{-1}k) \times l \equiv k^{-1}(k \times l) \equiv k^{-1} \times 0 \equiv 0 \pmod{n}$, which contradicts the assumption that $l < n$.

One of the most important results of elementary number theory is the so-called *Chinese Remainder Theorem*(4). It is given this name because a version was originally derived by the Chinese mathematician Sun Tse almost 2000 years ago. The Chinese Remainder Theorem establishes a method of solving simultaneously a system of linear congruences in several modular systems.

Theorem 16 (Chinese Remainder). Given a system $x \equiv a_i$ (mod n_i), $i = 1,2,\ldots m$. Suppose that, for all $i = j$, GCD(n_i,n_j) = 1. Then there is a unique common solution modulo $n = n_1n_2\ldots n_m$.

Proof (By construction). Let $n_i' = n/n_i$, $i = 1,\ldots,m$. Note that GCD(n_i,n_i') = 1. Thus, an integer exists n_i'' such that $n_i'n_i'' \equiv 1$(mod n_i). Then

$$x \equiv a_1n_1'n_1'' + a_2n_2'n_2'' + \cdots + a_mn_m'n_m''(\bmod n) \quad (16)$$

is the solution. As $n_i \mid n_j'$ if $i \neq j$, $x \equiv a_in_i'n_i'' \equiv a_i$ (mod n_i). The solution is also unique. If both x and y are common solutions, then $x - y \equiv 0$ (mod n).

An interesting consequence of this theorem is that there is a $1 - 1$ correspondence, preserved by addition and multiplication, of integers modulo n, and m-tuples of integers modulo n_i(5). Consider $\{n_1,n_2,n_3,n_4\} = \{7,11,13,17\}$, and $n = 17017$. Then

$$95 \rightarrow (\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3, \mathrm{a}_4) = (95 \bmod 7, 95 \bmod 11, 95 \bmod 13, 95 \bmod 17) = (4,7,4,10) \text{ also } 162 \rightarrow (1,8,6,9)$$

Performing addition and multiplication tuple-wise:

$$(4,7,4,10) + (1,8,6,9) = (5 \bmod 7,\ 15 \bmod 11,\ 10 \bmod 13, 19 \bmod 17) = (5,4,10,2); (4,7,4,10) \times (1,8,6,9) = (4 \bmod 7, 56 \bmod 11, 24 \bmod 13, 90 \bmod 17) = (4,1,11,5)$$

Now verify that $95 + 162 = 257$ and $95 \times 162 = 15{,}390$ are represented by (5,4,10,2) and (4,1,11,5) by reducing each number mod $n_1, \ldots, n_4$.

Another series of important results involving the products of elements in modular systems are the theorems of Euler, Fermat, and Wilson. Fermat's Theorem, although extremely important, is very easily proved; thus, it is sometimes called the "Little Fermat Theorem" in contrast to the famous Fermat's Last Theorem described earlier.

Theorem 17 (Euler). GCD If(a,n) = 1, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Theorem 18 (Fermat). If p is prime, then $a^p \equiv a \pmod{p}$.

Proof (of Euler's Theorem). Suppose $A = \{a_1,\ldots,a_{\phi(n)}\}$ is a list of the set of units in $\mathbb{Z}_n$. By definition, each of the a_i has an inverse a_i^{-1}. Now consider the product $b = a_1a_2\ldots a_{\phi(n)}$. It also has an inverse, in particular $b^{-1} = a_1^{-1}a_2^{-1}\cdots a_{\phi_{(n)}}^{-1}$. Choose any of the units; suppose it is a. Now consider the set $A' = \{aa_1, aa_2, aa_{\phi_{(n)}}\}$. We need to show that as a set, $A' = A$. It is sufficient to show that the aa_i are all distinct. As there are $\phi(n)$ of them, and they are all units, they represent all of the elements of A.

Suppose that $aa_i \equiv aa_j$ for some $i \neq j$. Then, as a is a unit, we can multiply by a^{-1}, yielding $(a^{-1}a)a_i \equiv (a^{-1}a)a_j$ or $a_i \equiv a_j$ (mod n), which is a contradiction. Thus, the aa_i are all distinct and $A = A'$.

Now compute

$$\prod_{i=1}^{\phi(n)}(aa_i) \equiv b(\bmod n) \text{ because } A = A' \quad (17)$$

$$a^{\phi(n)}b \equiv b(\bmod n)$$

$$a^{\phi_{(n)}}bb^{-1} \equiv bb^{-1}(\bmod n) \text{ multiplying by } b^{-1}$$
$$a^{\phi_{(n)}} \equiv 1(\bmod n) \quad (18)$$

Although the Chinese Remainder Theorem gives a solution for linear congruences, one would also like to consider nonlinear or higher degree polynomial congruences. In the case of polynomials in one variable, in the most general case,

$$f(x) \equiv 0(\bmod n) \quad (19)$$

If $n = p_1^{a1} \cdots p_k^{ak}$ is the factorization of n, then the Chinese Remainder Theorem assures solutions of (19) ⇔ each of

$$f(x) \equiv 0(\bmod p_i^{a_i})\ i = 1, 2, \ldots, k \quad (20)$$

has a solution.

As $\phi(p) = p - 1$ for p a prime, Fermat's Theorem is a direct consequence of Euler's Theorem.

Other important results in the theory of polynomial congruences are as follows:

Theorem 19 (Lagrange). If $f(x)$ is a nonzero polynomial of degree n, whose coefficients are elements of $\mathbb{Z}_p$ for a prime p, then $f(x)$ cannot have more than n roots.

Theorem 20 (Chevalley). If $f(x_1,\ldots,x_n)$ is a polynomial with degree less than n, and if the congruence

$$f(x_1, x_2, \ldots, x_n) \equiv 0(\bmod p)$$

has either zero or at least two solutions.

The Lagrange Theorem can be used to demonstrate the result of Wilson noted above.

Theorem 21 (Wilson). If p is a prime, then $(p-1)! \equiv -1$ (mod p).

Proof. If $p = 2$, the result is obvious. For p an odd prime, let

$$f(x) = x^{b-1} - (x-1)(x-2)\ldots(x-p+1) - 1.$$

Consider any number $1 \le k \le p - 1$. Substituting k for x causes the term $(x-1)(x-2)\ldots(x-p+1)$ to vanish; also, by Fermat's theorem, $k^{p-1} \equiv 1 (\mathrm{mod}\ p)$. Thus, $f(k) \equiv 0 (\mathrm{mod}\ p)$. But k has degree less than $p - 1$; and so by Lagrange's theorem, $f(x)$ must be identically zero, which means all of the coefficients must be divisible by p. The constant coefficient is

$$-1 - (p-1)!(\mathrm{mod}\ p)$$

and thus

$$(p-1)! \equiv -1 (\mathrm{mod}\ p)$$

QUADRATIC RESIDUES

Having considered general polynomial congruences, now we restrict consideration to quadratics. The study of quadratic residues leads to some useful techniques as well as having important and perhaps surprising results.

The most general quadratic congruence (in one variable) is of the form $ax^2 + bx + c \equiv 0$ (mod m). Such a congruence can always be reduced to a simpler form. For example, as indicated in the previous section, by the Chinese remainder theorem, we can assume the modulus is a prime power. As in the case $p = 2$ we can easily enumerate the solutions, we will henceforth consider only odd primes. Finally, we can use the technique of "completing the square" from elementary algebra to transform the general quadratic into one of the form $x^2 \equiv a$ (mod p).

If $p + a$, then if $x^2 \equiv a$ (mod p) is soluble, a is called a quadratic residue mod p; if not, a is called a quadratic nonresidue mod p.

Theorem 22. Exactly one half of the integers a, $1 \le a \le p - 1$, are quadratic residues mod p.

Proof. Consider the set of mod p values $QR = \{1^2, 2^2, \ldots, ((p-1)/2)^2\}$. Each of these values is a quadratic residue. If the values in QR are all distinct, there are at least $(p-1)/2$ quadratic residues. Suppose two of these values, say t and u, were to have $t^2 \equiv u^2$ (mod p). Then since $t^2 - u^2 \equiv 0$ (mod p) $\Rightarrow t + u \equiv 0$ (mod p) or $t - u \equiv 0$ (mod p). Since t and u are distinct, the second case is not possible; and since t and u must be both $< (p-1)/2$, so $t + u < p$ and neither is the first case. Thus there are at least $(p-1)/2$ quadratic residues.

If x_0 solves $x^2 \equiv a$ (mod p), then so does $p - x_0$, since $p^2 - 2px_0 + x_0^2 \equiv x_0^2 \equiv a(\mathrm{mod}\ p)$, and $p - x_0 T x_0$ (mod p). Thus we have found $(p-1)/2$ additional elements of Z_p, which square to elements of QR and therefore $p - 1$ elements overall. Thus there can be no more quadratic residues outside of QR, so the result is proved.

An important number theoretic function to evaluate quadratic residues is the Legendre symbol. For p an odd prime, the Legendre symbol for a (written alternatively (a / p) or $(\binom{a}{p})$) is

$$\binom{a}{p} = \begin{cases} +1 & \text{if } a \text{ is a quadratic residue mod } p \\ 0 & \text{if } p|a \\ -1 & \text{if } a \text{ is a quadratic nonresidue mod } p \end{cases}$$

One method of evaluating the Legendre symbol uses *Euler's criterion*. If p is an odd prime and GCD$(a,p) = 1$, then

$$\binom{a}{p} = 1 \Leftrightarrow a^{(p-1)/2} \equiv 1(\mathrm{mod}\ p)$$

Equivalently, $\binom{a}{p} \equiv a^{(p-1)/2}(\mathrm{mod}\ p)$.

Here are some other characteristics of the Legendre symbol:

Theorem 23. (i) $\binom{ab}{p} = \binom{a}{p}\binom{b}{p}$; (ii) if $a \equiv b$ (mod p), then $\binom{a}{p} = \binom{b}{p}$; (iii) $\binom{a^2}{p} = 1, \binom{1}{p} = 1$; (iv) $\binom{1}{p}(-1/p) = (-1)^{(p-1)/2}$.

Suppose we want to solve $x^2 \equiv 518$ (mod 17). Then, compute $\binom{518}{17} = \binom{8}{17} = \binom{2}{17}$. But $\binom{2}{17} = 1$ since $6^2 = 36 \equiv 2(\mathrm{mod}\ 17)$. Thus, $x^2 \equiv 518$ (mod 17) is soluble.

Computation of the Legendre symbol is aided by the following results. First, define an absolute least residue modulo p as the representation of the equivalence class of a mod p, which has the smallest absolute value.

Theorem 24 (Gauss' Lemma). Let GCD$(a,p) = 1$. If d is the number of elements of $\{a, 2a, \ldots, (p-1)a\}$ whose absolute least residues modulo p are negative, then

$$\binom{a}{p} = (-1)^d$$

Theorem 25. 2 is a quadratic residue (respectively, quadratic nonresidue) of primes of the form $8k \pm 1$ (respectively, $8k \pm 3$). That is,

$$\binom{2}{p} = (-1)^{(p^2-1)/8}$$

Theorem 26. (i) If $k > 1$, $p = 4k + 3$, and p is prime, then $2p + 1$ is also prime $\Leftrightarrow 2^p \equiv 1$ (mod $2p + 1$).

(ii) If $2p + 1$ is prime, then $2p + 1 \mid M_p$, the pth Mersenne number, and M_p is composite.

A concluding result for the computation of the Legendre symbol is one that, by itself, is one of the most famous—and surprising—results in all of mathematics. It is called *Gauss' Law of Quadratic Reciprocity*. What makes it so astounding is that it manages to relate prime numbers and their residues that seemingly bear no relationship to one another (6).

Suppose that we have two odd primes, p and q. Then, the Law of Quadratic Reciprocity relates the computation of their Legendre symbols; that is, it determines the quadratic residue status of each prime with respect to the other.

The proof, although derived from elementary principles, is long and would not be possible to reproduce here. Several sources for the proof are listed below.

Theorem 27 (Gauss' Law of Quadratic Reciprocity). $(\genfrac{}{}{0pt}{}{p}{q})(\genfrac{}{}{0pt}{}{q}{p}) = (-1)^{(p-1)(q-1)/4}$.

A consequence of the Law of Quadratic Reciprocity is as follows.

Theorem 28. Let p and q be distinct odd primes, and $a \geq 1$. If $p \equiv \pm\, q \pmod{4a}$, then $(\genfrac{}{}{0pt}{}{a}{p}) = (\genfrac{}{}{0pt}{}{a}{q})$.

An extension of the Legendre symbol is the Jacobi symbol. The Legendre symbol is defined only for primes p. By a natural extension, the Jacobi symbol, also denoted $({}^{a}{}_{n})$, is defined for any $n > 0$, assuming that the prime factorization of n is $p_1 \ldots p_k$, by

$$(\genfrac{}{}{0pt}{}{a}{n})\,[\text{jacobi symbol}] = (\genfrac{}{}{0pt}{}{a}{p_1})(\genfrac{}{}{0pt}{}{a}{p_2})\cdots(\genfrac{}{}{0pt}{}{a}{p_k})[\text{Legendre symbols}]$$

PRIME NUMBERS

The primes themselves have been a subject of much inquiry in number theory. We have observed earlier that there are an infinite number of primes, and that they are most useful in finite fields and modular arithmetic systems.

One subject of interest has been the development of a function to approximate the frequency of occurrence of primes. This function is usually called $\pi(n)$—it denotes the number of primes less than or equal to n.

In addition to establishing various estimates for $\pi(n)$, concluding with the so-called Prime Number Theorem, we will also state a number of famous unproven conjectures involving prime numbers.

An early estimate for $\pi(n)$, by Chebyshev, follows:

Theorem 29 (Chebyshev). If $n > 1$, then $n/(8 \log n) < \pi(n) < 6n/\log n$.

The Chebyshev result tells us that, up to a constant factor, the number of primes is of the order of $n/\log n$. In addition to the frequency of occurrence of primes, the greatest gap between successive primes is also of interest (7).

Theorem 30 (Bertrand's Partition). If $n \geq 2$, there is a prime p between n and $2n$.

Two other estimates for series of primes are as follows.

Theorem 31. $\sum_{p \leq n} (\log p)/p = \log n + O(1)$.

Theorem 32. $\sum_{p \leq n} 1/p = \log\log x + a + O(1/\log x)$

Another very remarkable result involving the generation of primes is from Dirichlet.

Theorem 33 (Dirichlet). Let a and b be fixed positive integers such that GCD$(a,b) = 1$. Then there are an infinite number of primes in the sequence $\{a + bn \mid n = 1, 2, \ldots\}$.

Finally, we have the best known approximation to the number of primes (8).

Theorem 34 (Prime Number Theorem). $\pi(n) \sim n/\log n$. The conjectures involving prime numbers are legion, and even some of the simplest ones have proven elusive for mathematicians. A few examples are the Goldbach conjecture, the twin primes conjecture, the interval problem, the Dirichlet series problem, and the Riemann hypothesis.

Twin Primes. Two primes p and q are called twins if $q = p + 2$. Examples are $(p,q) = (5,7)$; $(11,13)$; $(17,19)$; $(521,523)$. If $\pi_2(n)$ counts the number of twin primes less than n, the twin prime conjecture is that $\pi_2(n) \to \infty$ as $n \to \infty$. It is known, however, that there are infinitely many pairs of numbers (p,q), where p is prime, $q = p + 2$, and q has at most two factors.

Goldbach Conjecture. As stated, this conjecture is that every even number >2, is the sum of two primes. What is known is that every large even number can be expressed as $p + q$, where p is prime and q has at most two factors. Also, it is known that every large odd integer is the sum of three primes.

Interval Problems. It was demonstrated earlier that there is always a prime number between n and $2n$. It is not known, however, whether the same is true for other intervals, for example, such as n^2 and $(n + 1)^2$.

Dirichlet Series. In Theorem 33, a series containing an infinite number of primes was demonstrated. It was not known whether there are other series that have a greater frequency of prime occurrences, at least until recent research by Friedlander and Iwaniec (9), who showed that series of the form $\{a^2 + b^4\}$ not only have an infinite number of primes, but also that they occur more rapidly than in the Dirichlet series.

Riemann Hypothesis. Although the connection to prime numbers is not immediately apparent, the Riemann hypothesis has been an extremely important pillar in the theory of primes. It states that, for the complex function

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s} \quad s = \sigma + \mathrm{it} \in \mathbb{C}$$

there are zeros at $s = -2, -4, -6, \ldots$, and no more zeros outside of the "critical strip" $0 \leq \sigma \leq 1$. The Riemann hypothesis states that all zeros of ζ in the critical strip lie on the line $s = ½ + \mathrm{it}$.

Examples of important number theoretic problems whose answer depends on the Riemann hypothesis are (*1*) the existence of an algorithm to find a nonresidue mod p in polynomial time and (*2*) if n is composite, there is at least one b for which neither $b^t \equiv 1 (\mathrm{mod}\, n)\, nor\, b^{2^r t} \equiv -1\, (\mathrm{mod}\, n)$. This latter example is important in algorithms needed to find large primes (10).

DIOPHANTINE EQUATIONS

The term *Diophantine equation* is used to apply to a family of algebraic equations in a number system such as $\mathbb{Z}$ or $\mathbb{Q}$. To date, we have certainly observed many examples of Diophantine equations. A good deal of research in this subject has been directed at polynomial equations with integer or rational coefficients, the most famous of which being the class of equations $x^n + y^n = z^n$, the subject of Fermat's Last Theorem.

One result in this study, from Legendre, is as follows.

Theorem 35. Let $a, b, c, \in Z$ such that (i) $a > 0$, b, $c < 0$; (ii) a, b, and c are square-free; and (iii) GCD(a,b) = GCD(b,c) = GCD(a,c) = 1. Then

$$ax^2 + by^2 + cz^2 = 0$$

has a nontrivial integer solution $\Leftrightarrow -ab \in QR(c), -bc \in QR(a)\, and\, -ca \in QR(b)$.

Example. Consider the equation $3x^2 - 5y^2 - 7z^2 = 0$. With $a = 3$, $b = -5$, and $c = -7$, apply Theorem 35. Note that $ab \equiv -1 \pmod 7$, $ac \equiv -1 \pmod 5$, and $bc \equiv -1 \pmod 3$. Thus, all three products are quadratic residues, and the equation has an integer solution. Indeed, the reader may verify that $x = 3$, $y = 2$, and $z = 1$ is one such solution.

Another result, which consequently proves Fermat's Last Theorem in the case $n = 4$, follows. (Incidentally, it has also been long known that Fermat's Last Theorem holds for $n = 3$.)

Theorem 36. $x^4 + y^4 = z^2$ has no nontrivial solutions in the integers.

A final class of Diophantine equations is known generically as Mordell's equation: $y^2 = x^3 + k$. In general, solutions to Mordell's equation in the integers are not known. Two particular solutions are as follows.

Theorem 37. $y^2 = x^3 + m^2 - jn^2$ has no solution in the integers if

(i) $j = 4$, $m \equiv 3 \pmod 4$ and $p \mathbb{T}\, 3 \pmod 4$ when $p \mid n$.
(ii) $j = 1$, $m \equiv 2 \pmod 4$, n is odd, and $p \mathbb{T}\, 3 \pmod 4$ when $p \mid n$.

Theorem 38. $y^2 = x^3 + 2a^3 - 3b^2$ has no solution in the integers if $ab \neq 0$, $a \mathbb{T}\, 1 \pmod 3$, $3 \mid b$, a is odd if b is even, and $p = t^2 + 27u^2$ is soluble in integers t and u if $p \mid a$ and $p \equiv 1$ (mod 3).

ELLIPTIC CURVES

Many recent developments in number theory have come as the byproduct of the extensive research performed in a branch of mathematics known as elliptic curve theory (11).

An elliptic curve represents the set of points in some appropriate number system that are the solutions to an equation of the form $y^2 = Ax^3 + Bx^2 + Cx + D$ when $A, B, C, D \in \mathbb{Z}$.

A major result in this theory is the theorem of Mordell and Weil. If K is any algebraic field, and $C(K)$ the points with rational coordinates on an elliptic curve, then this object $C(K)$ forms a finitely generated Abelian group.

APPLICATIONS

In the modern history of cryptology, including the related area of authentication or digital signatures, dating roughly from the beginning of the computer era, there have been several approaches that have had enormous importance for business, government, engineering, and computing. In order of their creation, they are (*1*) the Data Encryption Standard or DES (1976); (*2*) the Rivest–Shamir–Adelman Public Key Cryptosystem, or RSA (1978); (*3*) the Elliptic Curve Public Key Cryptosystem or ECC (1993); and (*4*) Rijndael or the Advanced Encryption Standard, or AES (2001). RSA, DSS (defined below), and ECC rely heavily on techniques described in this article.

Data Encryption Standard (DES)

DES was developed and published as a U.S. national standard for encryption in 1976 (12). It was designed to transform 64-bit messages to 64-bit ciphers using 56-bit keys. Its structure is important to understand as the model for many other systems such as AES. In particular, the essence of DES is a family of nonlinear transformations known as the S-boxes. The S-box transformation design criteria were never published, however, and so there was often reluctance in the acceptance of the DES. By the mid-1990s, however, effective techniques to cryptanalyze the DES had been developed, and so research began to find better approaches.

Historically, cryptology required that both the sending and the receiving parties possessed exactly the same information about the cryptosystem. Consequently, that information that they both must possess must be communicated in some way. Encryption methods with this requirement, such as DES and AES, are also referred to as "private key" or "symmetric key" cryptosystems.

The Key Management Problem

Envision the development of a computer network consisting of 1000 subscribers where each pair of users requires a separate key for private communication. (It might be instructive to think of the complete graph on n vertices, representing the users; with the $n(n\ -1)/2$ edges corresponding to the need for key exchanges. Thus, in the 1000-user network, approximately 500,000 keys must be exchanged in some way, other than by the network!)

In considering this problem, Diffie and Hellman asked the following question: Is it possible to consider that a key might be broken into two parts, $k = (k_p, k_s)$, such that only k_p is necessary for encryption, while the entire key $k = (k_p, k_s)$ would be necessary for decryption (13)?

If it were possible to devise such a cryptosystem, then the following benefits would accrue. First of all, as the information necessary for encryption does not, a priori, provide an attacker with enough information to decrypt, then there is no longer any reason to keep it secret. Consequently k_p can be made public to all users of the network. A cryptosystem devised in this way is called a public key cryptosystem (PKC).

Furthermore, the key distribution problem becomes much more manageable. Consider the hypothetical network of 1000 users, as before. The public keys can be listed in a centralized directory available to everyone, because the rest of the key is not used for encryption; the secret key does not have to be distributed but remains with the creator of the key; and finally, both parts, public and secret, must be used for decryption.

Therefore, if we could devise a PKC, it would certainly have most desirable features.

In 1978, Rivest, Shamir, and Adelman described a public-key cryptosystem based on principles of number theory, with the security being dependent on the inherent difficulty of factoring large integers.

Factoring

Factoring large integers is in general a very difficult problem (14). The best-known asymptotic running-time solution today is $O(e^{((64n/9))^{1/3}(\log n)^{2/3}})$ for an n-bit number. More concretely, in early 2005, a certain 200-digit number was factored into two 100-digit primes using the equivalent of 75 years of computing time on a 2.2-GHz AMD Opteron processor.

Rivest–Shamir–Adelman Public Key Cryptosystem (RSA)

The basic idea of Rivest, Shamir, and Adelman was to take two large prime numbers, p and q (for example, p and q each being $\sim 10^{200}$, and to multiply them together to obtain $n = pq$. n is published. Furthermore, two other numbers, d and e, are generated, where d is chosen randomly, but relatively prime to the Euler function $\phi(n)$ in the interval [max(p,q) + 1, n −1]. As we have observed, $\phi(n) = (p - 1)\ (q - 1)$ (15).

Key Generation

1. Choose two 200-digit prime numbers randomly from the set of all 200-digit prime numbers. Call these p and q.
2. Compute the product $n = pq$. n will have approximately 400 digits.
3. Choose d randomly in the interval [max(p,q) + 1, n −1], such that GCD(d, $\phi(n)$) = 1.
4. Compute $e \equiv d^{-1}$ (modulo $\phi(n)$).
5. Publish n and e. Keep p, q, and d secret.

Encryption

1. Divide the message into blocks such that the bit-string of the message can be viewed as a 400-digit number. Call each block, m.
2. Compute and send $c \equiv m^e$ (modulo n).

Decryption

1. Compute $c^d \equiv (m^e)^d \equiv m^{ed} \equiv m^{k\phi(n)+1} \equiv m^{k\phi(n)} \times m \equiv m$ (modulo n)

Note that the result $m^{k\phi(n)} \equiv 1$ used in the preceding line is the Little Fermat Theorem.

Although the proof of the correctness and the security of RSA are established, there are several questions about the computational efficiency of RSA that should be raised.

Is it Possible to Find Prime Numbers of 200 Decimal Digits in a Reasonable Period of Time? The Prime Number Theorem 34 assures us that, after a few hundred random selections, we will probably find a prime of 200 digits.

We can never actually be certain that we have a prime without knowing the answer to the Riemann hypothesis; instead we create a test (the Solovay–Strassen test), which if the prime candidate passes, we assume the probability that p is not a prime is very low (16). We choose a number (say 100) numbers a_i at random, which must be relatively prime to p. For each ai, if the Legendre symbol $(\frac{a}{p}) = a_i^{(p-1)/2}$(mod p), then the chance that p is not a prime and passes the test is 1/2 in each case; if p passes all tests, the chances are $1/2^{100}$ that it is not prime.

Is it Possible to Find an e Which is Relatively Prime to $\phi(n)$? Computing the GCD(e,$\phi(n)$) is relatively fast, as is computing the inverse of e mod n. Here is an example of a (3 × 2) array computation that determines both GCD and inverse [in the example, GCD(1024,243) and 243^{-1} (mod 1024)]. First, create a (3 × 2) array where the first row are the given numbers; the second and third rows are the (2 × 2) identity matrix. In each successive column, subtract the largest multiple m of column k less than the first row entry of column (k − 1) to form the new column (for example, 1024 −4 × 243 = 52). When 0 is reached in the row A[1,*], say A[1,n], both the inverse (if it exists) and the GCD are found. The GCD is the element A[1,n −1]m and if this value is 1, the inverse exists and it is the value A[3,n −1] [243 × 59 ≡ 1 (mod 1024)].

Column	1	2	3	4	5	6	7
m			4	4	1	2	17
A[1,*]	1024	243	52	35	17	1	0
A[2,*]	1	0	1	4	5	−14	
A[3,*]	0	1	−4	17	−21	59	

Is it Possible to Perform the Computation $e \times d$ Where e and d are Themselves 200-digit Numbers? Computing m^e (mod n) consists of repeated multiplications and integer divisions. In a software package such as Mathematica, such a computation with 200-digit integers can be performed in 0.031 seconds of time on a Pentium machine.

One shortcut in computing a large exponent is to make use of the "fast exponentiation" algorithm: Express the exponent as a binary, $e = b_n b_{n-1} \ldots b_0$. Then compute m^e as follows:

```
$ans = m$
for $i = n - 1$ to 0 do
        $ans = ans \times ans$
        if $b_i = 1$ then $ans = ans \times x$
        end;
```

The result is ans. Note that the total number of multiplications is proportional to the log of e.

Example: Compute x^{123}.

$123 = (1111011)_{\text{binary}}$. So $n = 6$. Each arrow below represents one step in the loop. All but the fourth pass through the loop require squaring and multiplying by x, since the bits processed are 111011.

$$\begin{aligned} ans &= x \to x^2 \times x = x^3 \to x^6 \times x = x^7 \to x^{14} \times x \\ &= x^{15} \to x^{30} \to x^{60} \times x = x^{61} \to x^{122} \times x = x^{123} \end{aligned}$$

In a practical version of the RSA algorithm, it is recommended by Rivest, Shamir, and Adelman that the primes p and q be chosen to be approximately of the same size, and each containing about 100 digits.

The other calculations necessary in the development of an RSA cryptosystem have been shown to be relatively rapid. Except for finding the primes, the key generation consists of two multiplications, two additions, one selection of a random number, and the computation of one inverse modulo another number.

The encryption and decryption each require at most 2 $\log_2 n$ multiplications (in other words, one application of the Fast Exponentiation algorithm) for each message block.

DIGITAL SIGNATURES

It seems likely that, in the future, an application similar to public key cryptology will be even more widely used. With vastly expanded electronic communications, the requirements for providing a secure way of authenticating an electronic message—a digital signature—will be required far more often than the requirement for transmitting information in a secure fashion.

As with public key cryptology, the principles of number theory have been essential in establishing methods of authentication.

The authentication problem is the following: Given a message m, is it possible for a user u to create a "signature", s_u; dependent on some information possessed only by u, so that the recipient of the message (m,s_u) could use some public information for u (a public key), to be able to determine whether the message was authentic.

Rivest, Shamir, and Adelman showed that their public key encryption method could also be used for authentication.

For example, for A to send a message to B that B can authenticate, assume an RSA public key cryptosystem has been established with encryption and decryption keys e_A, d_A, e_B, d_B, and a message m to authenticate. A computes and sends $c \equiv m^{e_B d_A} (\bmod\, n)$. B both decrypts and authenticates by computing $(c^{e_A d_B}) \equiv (m^{e_B d_A})^{eAd_B} \equiv m^{e_B d_A e_A d_B} \equiv m^{e_B d_B} \equiv m (\bmod\, n)$.

However, several other authors, particularly El Gamal (17) and Ong et al. (18), developed more efficient solutions to the signature problem. More recently, in 1994, the National Institute of Standards and Technology, an agency of the United States government, established such a method as a national standard, now called the DSS or Digital Signature Standard (19).

The DSS specifies an algorithm to be used in cases where a digital authentication of information is required. We assume that a message m is received by a user. The objective is to verify that the message has not been altered, and that we can be assured of the originator's identity. The DSS creates for each user a public and a private key. The private key is used to generate signatures, and the public key is used in verifying signatures.

The DSS has two parts. The first part begins with a mechanism for hashing or collapsing any message down to one of a fixed length, say 160 bits. This hashing process is determined by a secure hash algorithm (SHA), which is very similar to a (repeated) DES encryption. The hashed message is combined with public and private keys derived from two public prime numbers. A computation is made by the signatory using both public and private keys, and the result of this computation, also 160 bits, is appended to the original message. The receiver of the message also performs a separate computation using only the public key knowledge, and if the receiver's computation correctly computes the appended signature, the message is accepted. Again, as with the RSA, the critical number theoretic result for the correctness of the method is the Little Fermat Theorem.

Elliptic Curve Public Key Cryptosystem (ECC)

One form of the general equation of an elliptic curve, the family of equations $y^2 = x^3 + ax + b$ $(a, b \in Z)$ has proven to be of considerable interest in their application to cryptology (20). Given two solutions $p = (x_1, y_1)$ and $Q = (x_2, y_2)$, a third point R in the Euclidean plane can be found by the following construction: Find a third point of intersection between the line PQ and the elliptic curve defined by all solutions to $y^2 = x^3 + ax + b$; then the point R is the reflection of this point of intersection in the x-axis. This point R may also be called $P + Q$, because the points so found follow the rules for an Abelian group by the Theorem of Mordell and Weil, with the identity $0 = P + (-P)$ being the point at infinity.

As an analogy to RSA, a problem comparable with the difficulty of factoring is the difficulty of finding k when $kP = P + P + \ldots + P$ (k times) is given. Specifically, for a cryptographic elliptic curve application, one selects an elliptic curve over a Galois field $GF(p)$ or $GF(2^m)$, rather than the integers. Then, as a public key, one chooses a secret k and a public p and computes $kP = Q$, with Q also becoming public.

One application, for example, is a variation of the widely used Diffie–Hellman secret sharing scheme (13). This scheme enables two users to agree on a common key without revealing any information about the key. If the two parties are A and B, each chooses at random a secret integer x_A and x_B. Then, A computes $y_A = x_A P$ in a public elliptic curve with

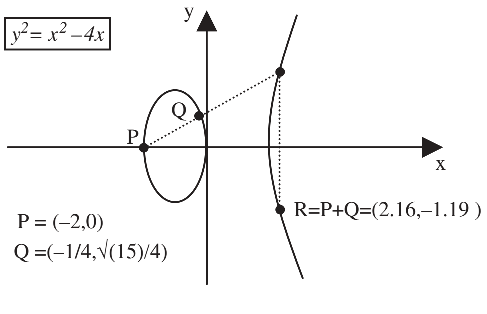


Figure 1. An example of addition of points on the elliptic curve $y^2 = x^3 - 4x$.

base point P. Similarly, B computes $y_B = x_B P$. A sends y_A to B and vice versa. Then, A computes $x_A y_B P$, and the resulting point $Q = (x_A y_B)P = (x_B y_A)P$ is the shared secret.

For computational reasons, it is advised to choose elliptic curves over specific fields. For example, a standard called P192 uses $GF(p)$, where $p = 2^{192} - 2^{64} + 1$ for efficient computation. Thus, x_A, x_B, y_A, y_B must all be less than p.

Advanced Encryption Standard or Rijndael (AES)

After it was generally accepted that a new U.S. standard was needed for symmetric key encryption, an international solicitation was made, with an extensive review process, for a new national standard. In late 2001, a system known as Rijndael, created by Vincent Rijmen and Joan Daemen, was selected as the U.S. Advanced Encryption Standard, or AES (21).

Although this method has some similarities to the DES, it has been widely accepted, particularly since the nonlinear S-boxes have, in contrast to the DES, a very understandable, although complex structure. The S-box in AES is a transformation derived from an operator in one of the so-called Galois fields $GF(2^8)$. It is a well-known result of algebra that all finite fields have several elements that is a prime power, that all finite fields with the same number of elements are isomorphic, and that a finite field with p^n elements can be represented as the quotient field of the ring of polynomials whose coefficients are mod p numbers, modulo an irreducible polynomial whose highest power is n.

BIBLIOGRAPHY

1. A. Wiles, Modular elliptic-curves and Fermat's Last Theorem. *Ann. Math.* **141**, 443–551, 1995.
2. Y. Wang. (ed.), *Goldbach conjecture*, Series in Pure Mathematics volume 4, Singapore: World Scientific, 1984, pp. xi + 311.
3. H. de Coste, *La vie du R P Marin Mersenne, theologien, philosophe et mathématicien, de l'Ordre der Peres Minim* (Paris, 1649).
4. O. Ore, *Number Theory and Its History*, New York: Dover Publications, 1976.
5. M. Abdelguerfi, A. Dunham, W. Patterson, MRA: A Computational Technique for Security in High-Performance Systems, *Proc. of IFIP / Sec '93, International Federation of Information Processing Societies, World Symposium on Computer Security*, Toronto, Canada, 1993, pp. 381–397.
6. E. W. Weisstein. Quadratic Reciprocity Theorem. From *Math-World*–A Wolfram Web Resource. http://mathworld.wolfram.-com/QuadraticReciprocityTheorem.html.
7. Wikipedia, Proof of Bertrand's postulate, http://www.absoluteastronomy.com/encyclopedia/p/pr/proof_of_bertrands_postulate.htm.
8. A. Selberg An elementary proof of the prime number theorem, *Ann. Math.*, **50**: 305–313, 1949. MR 10,595b [[HW79, sections 22.15–16] gives a slightly simpler, but less elementary version of Selberg's proof.]
9. J. Friedlander and H. Iwaniec, Using a parity-sensitive sieve to count prime values of a polynomial, *Proc. Nat. Acad. Sci.*, **94**: 1054–1058, 1997.
10. S. Smale, Mathematical problems for the next century. *Math. Intelligencer* **20**(2): 7–15, 1998.
11. J. H. Silverman, *The Arithmetic of Elliptic Curves*, New York: Springer, 1994.
12. National Bureau of Standards, *Data Encryption Standard,* Federal Information Processing Standards Publication 46, January 1977.
13. W. Diffie and M. E. Hellman, *New Directions in Cryptography*, IEEE Trans. on Information Theory, IT-22, 1976, pp. 644–654.
14. C. Pomerance, Analysis and comparison of some integer factoring algorithms, in H. W. Lenstra, Jr., and R. Tijdeman, (eds.), *Computational Number Theory: Part 1*, Math. Centre, Tract 154, Math. Centre, The Netherlands: Amsterdam, 1982, pp. 89–139.
15. R. L. Rivest, A. Shamir, and L. Adelman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Comm. of the ACM*, 1978, pp. 120–126.
16. R. Solovay and V. Strassen, A fast monte-carlo tests for primality, *SIAM Journal on Computing*, **6**: 84–85, 1977.
17. T. El Gamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *Proc. of Crypto 84*, New York: Springer, 1985, pp. 10–18.
18. H. Ong, C. P. Schnorr, and A. Shamir, An efficient signature scheme based on polynomial equations, *Proc. of Crypto 84,* New York: Springer, 1985, pp. 37–46.
19. National Institute of Standards and Technology, *Digital Signature Standard*, Federal Information Processing Standards Publication 186, May 1994.
20. N. Koblitz, Elliptic curve cryptosystems, *Mathematics of Computation* **48**: 203–209, 1987.
21. J. Daemen and V. Rijmen, *The Design of Rijndael: AES—The Advanced Encryption Standard*, Berun, Germany: Springer-Verlag, 2002.

FURTHER READING

D. M. Burton, *Elementary Number Theory*, Boston: Allyn and Bacon, 1980.

H. Cohen, *A Second Course in Number Theory*, New York: Wiley, 1962.

L. E. Dickson, *A History of the Theory of Numbers*, 3 vols., Washington, D.C.: Carnegie Institution, 1919–23.

G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, 3rd ed., Oxford: Clarendon Press, 1954.

K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, 2nd ed., New York: Springer-Verlag, 1990.

I. Niven and H. S. Zuckerman, *An Introduction to the Theory of Numbers*, 4th ed., New York: Wiley, 1980.

W. Patterson, *Mathematical Cryptology*, Totowa, NJ: Rowman and Littlefield, 1987.

H. E. Rose, *A Course in Number Theory*, Oxford: Oxford University Press, 1988.

H. N. Shapiro, *Introduction to the Theory of Numbers*, New York: Wiley-Interscience, 1983.

H. Stark, *An Introduction to Number Theory*, Chicago, IL: Markham, 1970.

WAYNE PATTERSON
Howard University
Washington, D.C.

C

CONVEX OPTIMIZATION

MOTIVATION AND A BRIEF HISTORY

An optimization problem can be stated in the so-called standard form as follows:

$$\begin{array}{l} \text{minimize } f(\mathbf{x}) : \mathbf{R}^n \rightarrow \mathbf{R} \\ \text{subject to } \boldsymbol{g}(\mathbf{x}) \leq 0,\ \boldsymbol{g} : \mathbf{R}^m \rightarrow \mathbf{R}^n \end{array} \tag{1}$$

representing the minimization of a function f of n variables under constraints specified by inequalities determined by functions $g = [g_1, g_2, \ldots, g_m]^{\mathrm{T}}$. The functions f and g_i are, in general, nonlinear functions. Note that "$\geq$" inequalities can be handled under this paradigm by multiplying each side by -1 and equalities by representing them as a pair of inequalities. The maximization of an objective function function $f(\mathbf{x})$ can be achieved by minimizing $-f(\mathbf{x})$. The set $\mathcal{F} = \{\mathbf{x} | \boldsymbol{g}(\mathbf{x}) \leq \mathbf{0}\}$ that satisfies the constraints on the nonlinear optimization problem is known as the feasible set or the feasible region. If $\mathcal{F}$ covers all of (a part of) $\mathbf{R}^n$, then the optimization is said to be *unconstrained (constrained)*. The above standard form may not directly be applicable to design problems where multiple conflicting objectives must be optimized. In such a case, multicriterion optimization techniques and Pareto optimality can be used to identify noninferior solutions (1). In practice, however, techniques to map the problem to the form in Equation (1) are often used.

When the objective function is a convex function and the constraint set is a convex set (both terms will be formally defined later), the optimization problem is known as a convex programming problem. This problem has the remarkable property of unimodality; i.e., any local minimum of the problem is also a global minimum. Therefore, it does not require special methods to extract the solution out of local minima in a quest to find the global minimum. Although the convex programming problem and its unimodality property have been known for a long time, it is only recently that efficient algorithms for solving of these problems have been proposed. The genesis of these algorithms can be traced back to the work of Karmarkar (2) that proposed a polynomial-time interior-point technique for linear programming, a special case of convex programming. Unlike the simplex method for linear programming, this technique was found to be naturally extensible from the problem of linear programming to general convex programming formulations. It was shown that this method belongs to the class of interior penalty methods proposed by Fiacco and McCormick (3) using barrier functions. The work of Renegar (4) showed that a special version of the method of centers for linear programming is polynomial. Gonzaga (5) showed similar results with the barrier function associated with a linear programming problem, with a proof of polynomial-time complexity. Nesterov and Nemirovsky (6) introduced the concept of self-concordance, studying barrier methods in their context. Further improvements in the computational complexity using approximate solutions and rank-one updates were shown in the work of Vaidya (7). The work of Ye (8) used a potential function to obtain the same complexity as Renegar's work without following the central path too closely.

DEFINITIONS OF CONVEXITY

Convex Sets

Definition. A set $C \in \mathbf{R}^n$ is said to be a **convex set** if, for every $\mathbf{x}_1,\ \mathbf{x}_2 \in C$ and every real number α, $0 \leq \alpha \leq 1$, the point $\alpha\ \mathbf{x}_1 + (1 - \alpha)\mathbf{x}_2 \in C$.

This definition can be interpreted geometrically as stating that a set is convex if, given two points in the set, every point on the line segment joining the two points is also a member of the set. Examples of convex and nonconvex sets are shown in Fig. 1.

Elementary Convex Sets

Ellipsoids. An ellipsoid $\mathrm{E}(\mathbf{x}, \mathcal{B}, \mathrm{r}) \in \mathbf{R}^n$ centered at point $\mathbf{x} \in \mathbf{R}^n$ is given by the equation

$$\{\mathbf{y} | (\mathbf{y} - \mathbf{x})^{\mathrm{T}} \mathcal{B} (\mathbf{y} - \mathbf{x}) \leq \mathrm{r}^2\}$$

If $\mathcal{B}$ is a scalar multiple of the unit matrix, then the ellipsoid is called a *hypersphere*. The axes of the ellipsoid are given by the eigenvectors, and their lengths are related to the corresponding eigenvalues of $\mathcal{B}$.

Hyperplanes. A hyperplane in n dimensions is given by the region

$$\mathbf{c}^{\mathrm{T}}\mathbf{x} = b,\ \mathbf{c} \in \mathbf{R}^n,\ b \in \mathbf{R}.$$

Half-Spaces. A half-space in n dimensions is defined by the region that satisfies the inequality

$$\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq b,\ \mathbf{c} \in \mathbf{R}^n,\ b \in \mathbf{R}.$$

Polyhedra. A (convex) polyhedron is defined as an intersection of half-spaces and is given by the equation

$$\mathrm{P} = \{\mathbf{x} | A\mathbf{x} \leq \mathbf{b}\},\ A \in \mathbf{R}^{m \times n},\ \mathbf{b} \in \mathbf{R}^m,$$

corresponding to a set of m inequalities $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq \mathrm{b}_i$, $\mathbf{a}_i \in \mathbf{R}^n$. If the polyhedron has a finite volume, it is referred to as a *polytope*. An example of a polytope is shown in Fig. 2.

Some Elementary Properties of Convex Sets

Property. The intersection of convex sets is a convex set. The union of convex sets is not necessarily a convex set.

Property. (Separating hyperplane theorem) Given two nonintersecting convex sets A and B, a separating hyperplane $\mathbf{c}^{\mathrm{T}}\mathbf{x} = \mathbf{b}$ exists such that A lies entirely within the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

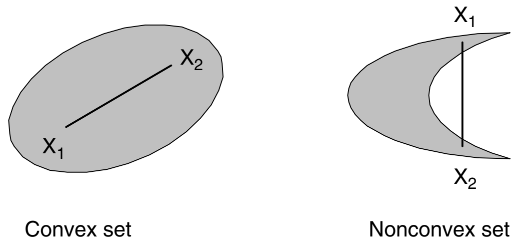


Figure 1. Convex and nonconvex sets.

half-space $\mathbf{c}^T\mathbf{x} \geq \mathbf{b}$ and B lies entirely within the half-space $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}$. This is pictorially illustrated in Fig. 3.

Property. (Supporting hyperplane theorem) Given a convex set C and any point $\mathbf{x}$ on its boundary, a supporting hyerplane $\mathbf{c}^T\mathbf{x} = \mathbf{b}$ exists such that the convex set C lies entirely within the half-space $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}$. This is illustrated in Fig. 4.

Definition. The **convex hull** of m points, $\mathbf{x}_1, \mathbf{x}_2 \ldots,$ $\mathbf{x}_\mathrm{m} \in \mathbf{R}^n$, denoted $co(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\mathrm{m})$, is defined as the set of points $\mathbf{y} \in \mathbf{R}^n$ such that

$$\mathbf{y} = \sum_{i=1\,\mathrm{to}\,m} \alpha_i \mathbf{x}_i \qquad \alpha_i \geq 0 \,\forall\, i \qquad \sum_{i=1\,\mathrm{to}\,m} \alpha_i = 1.$$

The convex hull is thus the smallest convex set that contains the m points. An example of the convex hull of five points in the plane is shown by the shaded region in Fig. 5. If the set of points $\mathbf{x}_i$ is of finite cardinality (i.e., m is finite), then the convex hull is a polytope. Hence, a polytope is also often described as the convex hull of its vertices.

Convex Functions

Definition. A function f defined on a convex set Ω is said to be a **convex function** if, for every $\mathbf{x}_1, \mathbf{x}_2 \in \Omega$, and every α, $0 \leq \alpha \leq 1$,

$$\mathrm{f}(\alpha\, \mathbf{x}_1 + (1-\alpha)\mathbf{x}_2) \leq \alpha\, \mathrm{f}(\mathbf{x}_1) + (1-\alpha)\, \mathrm{f}(\mathbf{x}_2)$$

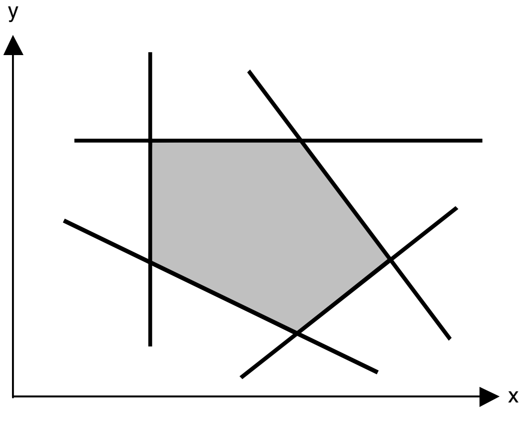


Figure 2. An example convex polytope in two dimensions as an intersection of five half-spaces.

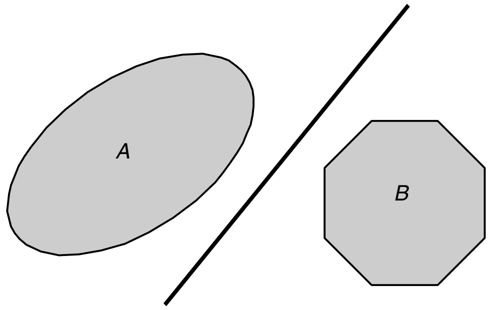


Figure 3. A separating hyperplane (line) in two dimensions between convex sets A and B.

f is said to be **strictly convex** if the above inequality is strict for $0 < \alpha < 1$.

Geometrically, a function is convex if the line joining two points on its graph is always above the graph. Examples of convex and nonconvex functions on $\mathbf{R}^n$ are shown in Fig. 6.

Some Elementary Properties of Convex Functions

Property. A function $f(\mathbf{x})$ is convex over the set S if and only if

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + [\nabla f(\mathbf{x}_0)]^T(\mathbf{x} - \mathbf{x}_0) \,\forall\, \mathbf{x},\, \mathbf{x}_0 \in S,$$

where ∇f corresponds to the gradient of f with respect to the vector $\mathbf{x}$. Strict convexity corresponds to the case in which the inequality is strict.

Property. A function $f(\mathbf{x})$ is convex over the convex set S if and only if

$$\nabla^2 f(\mathbf{x}_0) \geq 0 \,\forall\, \mathbf{x}_0 \in S$$

i.e., its Hessian matrix is positive semidefinite over S. For strict convexity, $\nabla^2 f(\mathbf{x}_0)$ must be positive definite.

Property. If $f(\mathbf{x})$ and $g(\mathbf{x})$ are two convex functions on the convex set S, then the functions $f+g$ and $\max(f, g)$ are convex over S.

Definition. The **level set** of a function $f(\mathbf{x})$ is the set defined by $f(\mathbf{x}) \leq c$ where c is a constant.

An example of the level sets of $f(\mathrm{x}, \mathrm{y}) = \mathrm{x}^2 + \mathrm{y}^2$ is shown in Fig. 7. Observe that the level set for $f(x,y) \leq c_1$ is contained in the level set of $f(x,y) \leq c_2$ for $c_1 < c_2$.

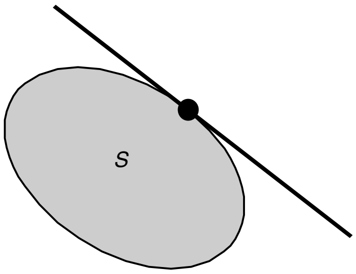


Figure 4. A supporting hyperplane (line) in two dimensions at the bounary point of a convex set S.

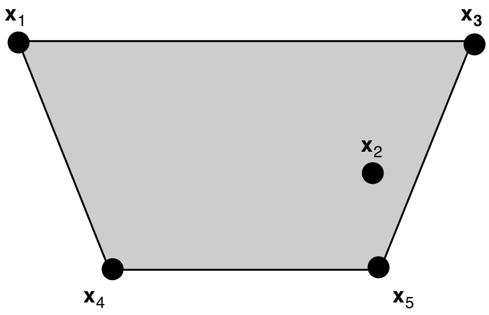


Figure 5. An example showing the convex hull of five points.

Property. If f is a convex function in the space $\mathcal{S}$, then the level set of f is a convex set in S. This is a very useful property, and many convex optimization algorithms depend on the fact that the constraints are defined by an intersection of level sets of convex functions.

Definition. A function g defined on a convex set Ω is said to be a **concave function** if the function $f = -g$ is convex. The function g is **strictly concave** if $-g$ is strictly convex.

For a fuller treatment on convexity properties, the reader is referred to Ref. (9).

CONVEX OPTIMIZATION

Convex Programming

Definition. A **convex programming problem** is an optimization problem that can be stated as follows:

$$\begin{aligned}&\text{minimize } f(\mathbf{x})\\&\text{such that } \mathbf{x} \in s\end{aligned} \qquad (2)$$

where f is a convex function and S is a convex set. This problem has the property that any local minimum of f over S is a global minimum.

Comment. The problem of maximizing a convex function over a convex set does not have the above property. However, it can be shown (10) that the maximum value for such a problem lies on the boundary of the convex set.

For a convex programming problem of the type in Equation (2), we may state without loss of generality that the objective function is linear. To see this, note that Equation (2) may equivalently be written as $\{\min t : f(\mathbf{x}) \le t, \mathbf{g}(\mathbf{x}) \le 0\}$.

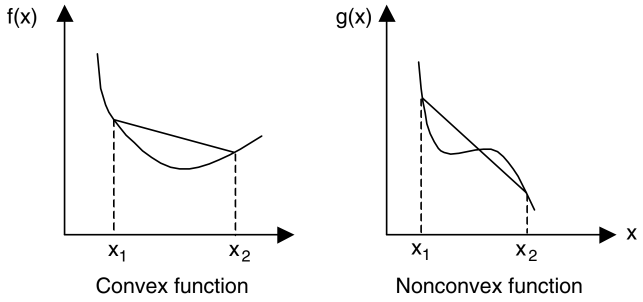


Figure 6. Convex and nonconvex functions.

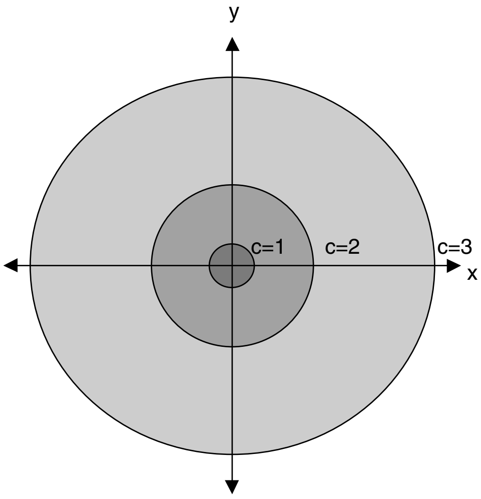


Figure 7. Level sets of $f(x,y) = x^2 + y^2$.

Linear programming is a special case of nonlinear optimization and, more specifically, a special type of convex programming problem where the objective and constraints are all linear functions. The problem is stated as

$$\begin{aligned}&\text{mnimize } \mathbf{c}^{\mathrm{T}}\mathbf{x}\\&\text{subject to } A\,\mathbf{x} \le \mathbf{b},\ \mathbf{x} \ge 0\\&\text{where } A \in \mathbf{R}^{m\times n},\ \mathbf{b} \in \mathbf{R}^{m},\ \mathbf{c} \in \mathbf{R}^{n},\ \mathbf{x} \in \mathbf{R}^{n}.\end{aligned} \qquad (3)$$

The feasible region for a linear program corresponds to a polyhedron in $\mathbf{R}^n$. It can be shown that an optimal solution to a linear program must necessarily occur at one of the vertices of the constraining polyhedron. The most commonly used technique for solution of linear programs, the simplex method (11), is based on this principle and operates by a systematic search of the vertices of the polyhedron. The computational complexity of this method can show an exponential behavior for pathological cases, but for most practical problems, it has been observed to grow linearly with the number of variables and sublinearly with the number of constraints. Algorithms with polynomial time worst-case complexity do exist; these include Karmarkar's method (2), and the Shor–Khachiyan ellipsoidal method (12). The computational times of the latter, however, are often seen to be impractical from a practical standpoint.

In the remainder of this section, we will describe various methods used for convex programming and for mapping problems to convex programs.

Path-Following Methods

This class of methods proceeds iteratively by solving a sequence of unconstrained problems that lead to the solution of the original problem. In each iteration, a technique based on barrier methods is used to find the optimal solution. If we denote the optimal solution in the kth iteration as $\mathbf{x}_k^*$, then the path $\mathbf{x}_1^*$, $\mathbf{x}_2^*$, $\ldots$ in $\mathbf{R}^n$ leads to the optimal solution, and hence, this class of techniques are known as path-following methods.

Barrier Methods. The barrier technique of Fiacco and McCormick (3) is a general technique to solve any constrained nonlinear optimization problem by solving a sequence of unconstrained nonlinear optimization problems. This method may be used for the specific case of minimizing a convex function over a convex set S described by an intersection of convex inequalities

$$S = \{\mathbf{x} | g_i(\mathbf{x}) \leq 0, i = 1, 2, \ldots, m\}$$

where each $g_i(\mathbf{x})$ is a convex function. The computation required of the method is dependent on the choice of the barrier function. In this connection, the logarithmic barrier function (abbreviated as the log barrier function) for this set of inequalities is defined as

$$\Phi(\mathbf{x}) = \begin{cases} -\sum_{i=1}^{n} \log[-g_i(\mathbf{x})] & \text{for } \mathbf{x} \in S \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, the idea of the barrier is that any iterative gradient-based method that tries to minimize the barrier function will be forced to remain in the feasible region, due to the singularity at the boundary of the feasible region. It can be shown that $\Phi(\mathbf{x})$ is a convex function over S and its value approaches ∞. as $\mathbf{x}$ approaches the boundary of S. Intuitively, it can be seen that $\Phi(\mathbf{x})$ becomes smallest when $\mathbf{x}$ is, in some sense, farthest away from all boundaries of S. The value of $\mathbf{x}$ at which the function $\Phi(\mathbf{x})$ is minimized is called the *analytic center* of S.

Example. For a linear programming problem of the type described in Equation (3), with constraint inequalities described by $a_i^T \mathbf{x} \leq b_i$, the barrier function in the feasible region is given by

$$\Phi(\mathbf{x}) = -\sum_{i=1 \text{ to } n} \log(b_i - a_i^T \mathbf{x})$$

The value of $(b_i - a_i^T \mathbf{x})$ represents the slack in the *ith* inequality, i.e., the distance between the point $\mathbf{x}$ and the corresponding constraint hyperplane. The log barrier function, therefore, is a measure of the product of the distances from a point $\mathbf{x}$ to each hyperplane, as shown in Fig. 8(a). The value of $\Phi(\mathbf{x})$ is minimized when $\prod_{i=1 \text{ to } n}(b_i - a_i^T \mathbf{x})$ is maximized. Coarsely speaking, this occurs when the distance to each constraint hyperplane is sufficiently large.

As a cautionary note, we add that although the analytic center is a good estimate of the center in the case where all constraints present an equal contribution to the boundary, the presence of redundant constraints can shift the analytic center. The effect on the analytic center of repeating the constraint p five times is shown in Fig. 8(b).

We will now consider the convex programming problem specified as

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) \\ &\text{such that} g_i(\mathbf{x}) \leq 0, i = 1, 2, \ldots, m \end{aligned}$$

where each $g_i(\mathbf{x})$ is a convex function. The traditional barrier method (3) used to solve this problem would formulate the corresponding unconstrained optimization problem

$$\text{minimize} B(\alpha) = \alpha \ f(\mathbf{x}) + \Phi(\mathbf{x})$$

and solve this problem for a sequence of increasing (constant) values of the parameter α. When α is zero, the problem reduces to finding the center of the convex set constraining the problem. As α increases, the twin objectives of minimizing $f(\mathbf{x})$ and remaining in the interior of the convex set are balanced. As α approaches ∞, the problem amounts to the minimization of the original objective function, f.

In solving this sequence of problems, the outer iteration consists of increasing the values of the parameter α. The inner iteration is used to minimize the value of $B(\alpha)$ at that value of α, using the result of the previous outer iteration as an initial guess. The inner iteration can be solved using Newton's method (10). For positive values of α, it is easy to see that $B(\alpha)$ is a convex function. The notion of a central path for a linearly constrained optimization problem is shown in Fig. 9.

Method of Centers. Given a scalar value $t > f(\mathbf{x}^*)$, the method of centers finds the analytic center of the set of inequalities $f(\mathbf{x}) \leq t$ and $g_i(\mathbf{x}) \leq 0$ by minimizing the

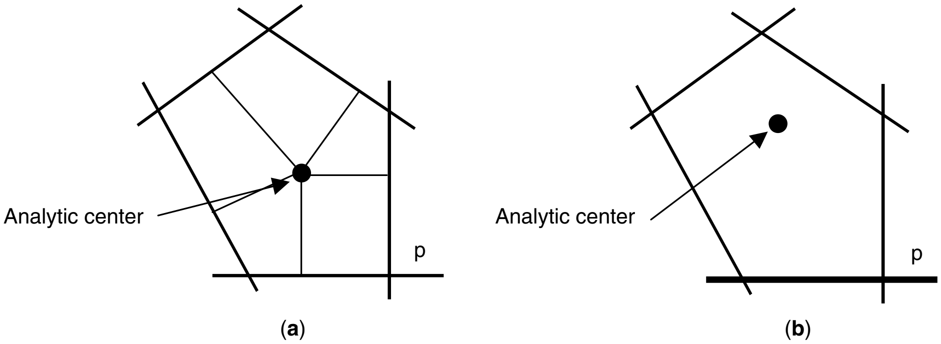


Figure 8. (a) Physical meaning of the barrier function for the feasible region of a linear program. (b) The effect of redundant constraints on the analytic center.

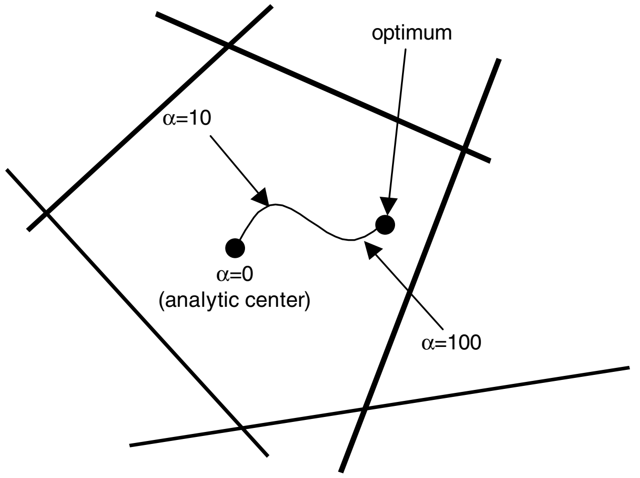


Figure 9. Central path for a linearly constrained convex optimization problem.

function

$$-\log(t - f(\mathbf{x})) + \Phi(\mathbf{x})$$

where $\Phi(\mathbf{x})$ is the log barrier function defined earlier. The optimal value $\mathbf{x}^*$ associated with solving the optimization problem associated with finding the analytic center for this barrier function is found and the value of t is updated to be a convex combination of t and $f(\mathbf{x}^*)$ as

$$t = \theta t + (1-\theta) f(\mathbf{x}^*),\ \theta > 0$$

The Self-Concordance Property and Step Length. The value of θ above is an adjustable parameter that would affect the number of Newton iterations required to find the optimum value of the analytic center. Depending on the value of θ chosen, the technique is classified as a short-step, medium-step, or long-step (possibly with even $\theta > 1$) method. For a short-step method, one Newton iteration is enough, whereas for longer steps, further Newton iterations may be necessary.

Nesterov and Nemirovsky (17) introduced the idea of the self-concordance condition, defined as follows.

Definition. A convex function $\varphi : S \to \mathbf{R}$ is **self-concordant** with parameter $a \geq 0$ (a-selfconcordant) on S; if φ is three times continuously differentiable on S and for all $\mathbf{x} \in S$ and $\mathbf{h} \in \mathbf{R}^m$, the following inequality holds:

$$|D^3\varphi(\mathbf{x})[\mathbf{h},\mathbf{h},\mathbf{h}]| \leq 2\,a^{-1/2}(D^2\varphi(\mathbf{x})[\mathbf{h},\mathbf{h}])^{3/2}$$

where $D^k\,\varphi(\mathbf{x})[\mathbf{h}_1, \mathbf{h}_2, \ldots \mathbf{h}_k]$ denotes the value of the kth differential of φ taken at $\mathbf{x}$ along the collection of directions $[\mathbf{h}_1, \mathbf{h}_2, \ldots \mathbf{h}_k]$.

By formulating logarithmic barrier functions that are self-concordant, proofs of the polynomial-time complexity of various interior point methods have been shown. An analysis of the computational complexity in terms of the number of outer and inner (Newton) iterations is presented in Refs. (6) and (13).

Other Interior Point Methods

Affine Scaling Methods. For a linear programming problem, the nonnegativity constraints $\mathbf{x} \geq 0$ are replaced by constraints of the type $\|X^{-1}(\mathbf{x}-\mathbf{x}_c)\| < \beta \leq 1$, representing an ellipsoid centered at $\mathbf{x_c}$. The linear program is then relaxed to the following form whose feasible region is contained in that of the original linear program:

$$\min\{\mathbf{c}^{\mathrm{T}}\,\mathbf{x} : A\mathbf{x} = \mathbf{b}, \|X^{-1}(\mathbf{x}-\mathbf{x}_c)\| < \beta \leq 1\}$$

Note that the linear inequalities in Equation (3) have been converted to equalities by the addition of slack variables. This form has the following closed-form solution:

$$\mathbf{x}(\beta) = \mathbf{x} - \beta\frac{XP_{AX}X\mathbf{c}}{\|P_{AX}X\mathbf{c}\|}$$

where $P_{AX} = (1 - XA^T(AX^2A^T)^{-1}AX)$. The updated value of $\mathbf{x}$ is used in the next iteration, and so on. The search direction $XP_{AX}X_{\mathbf{c}}$ is called the primal affine scaling direction and corresponds to the scaled projected gradient with respect to the objective function, with scaling matrix X. Depending on the value of β, the method may be a short-step or a long-step (with $\beta > 1$) method, and convergence proofs under various conditions are derived. For details of the references, the reader is referred to Ref. (13).

For a general convex programming problem of the type (note that the linear objective function form is used here),

$$\{\min f(\mathbf{y}) = \mathbf{b}^{\mathrm{T}}\mathbf{y} : g_i(\mathbf{y}) \leq 0\}$$

The constraint set here is similarly replaced by the ellipsoidal constraint $((\mathbf{y}-\mathbf{y}_c)^T H(\mathbf{y}-\mathbf{y}_c) \leq \beta^2$, where $\mathbf{y}_c$ is the center of the current ellipsoid, $\mathbf{y}$ is the variable over which the minimization is being performed, and H is the Hessian of the log-barrier function $-\sum_{i=1\,\mathrm{to}\,n}\log(-g_i(\mathbf{y}))$. The problem now reduces to

$$\{\min \mathbf{b}^{\mathrm{T}}\mathbf{y} : (\mathbf{y}-\mathbf{y}_c)^{\mathrm{T}}H(\mathbf{y}-\mathbf{y}_c) \leq \beta^2\}$$

which has a closed-form solution of the form

$$\mathbf{y}(\beta) = \mathbf{y} - \beta\frac{H^{-1}\mathbf{b}}{\sqrt{\mathbf{b}^{\mathrm{T}}H^{-1}\mathrm{b}}}$$

This is used as the center of the ellipsoid in the next iteration. The procedure continues iteratively until convergence.

Potential-Based Methods. These methods formulate a potential function that provides a coarse estimate of how far the solution at the current iterate is from the optimal value. At each step of a potential reduction method, a direction and step size are prescribed; however, the potential may be minimized further by the use of a line search (large steps). This is in contrast with a path-following method that must maintain proximity to a prescribed central path. An example of a potential-based technique is one that uses a weighted sum of the gap between the value of the primal optimization problem and its dual, and of the log-barrier function value as the potential. For a more detailed description of this and other potential-based methods, the reader is referred to Refs. (6) and (13).

Localization Approaches

Polytope Reduction. This method begins with a polytope $\mathcal{P} \in \mathbf{R}^n$ that contains the optimal solution $\mathbf{x}_{\text{opt}}$. The polytope P may, for example, be initially selected to be an n-dimensional box described by the set

$$\{\mathbf{x}|X_{\min} \leq \mathbf{x}(i) \leq X_{\max}\},$$

where $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of each variable, respectively. In each iteration, the volume of the polytope is shrunk while keeping $\mathbf{x}_{\text{opt}}$ within the polytope, until the polytope becomes sufficiently small. The algorithm proceeds iteratively as follows.

Step 1. A *center* $\mathbf{x}_c$ deep in the interior of the polytope $\mathcal{P}$ is found.

Step 2. The feasibility of the center $\mathbf{x}_c$ is determined by verifying whether all of the constraints of the optimization problem are satisfied at $\mathbf{x}_c$. If the point $\mathbf{x}_c$ is infeasible, it is possible to find a separating hyperplane passing through $\mathbf{x}_c$ that divides $\mathcal{P}$ into two parts, such that the feasible region lies entirely in the part satisfying the constraint

$$\mathbf{c}^T\mathbf{x} \geq \beta$$

where $\mathbf{c} = -[\nabla g_p(\mathbf{x})]^T$ is the negative of the gradient of a violated constraint, g_p, and $\beta = \mathbf{c}^{\mathrm{T}}\mathbf{x}_c$. The separating hyperplane above corresponds to the tangent plane to the violated constraint. If the point $\mathbf{x}_c$ lies within the feasible region, then a hyperplane $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \beta$ exists that divides the polytope into two parts such that $\mathbf{x}_{\text{opt}}$ is contained in one of them, with $\mathbf{c} = -[\nabla f(\mathbf{x})]^T$ being the negative of the gradient of the objective function and β being defined as $\beta = \mathbf{c}^T\mathbf{x}_c$ once again. This hyperplane is the supporting hyperplane for the set $f(\mathbf{x}) \leq f(\mathbf{x}_c)$ and thus eliminates from the polytope a set of points whose value is larger than the value at the current center. In either case, a new constraint $\mathbf{c}^T\mathbf{x} \geq \beta$ is added to the current polytope to give a new polytope that has roughly half the original volume.

Step 3. Go to Step 1 and repeat the process until the polytope is sufficiently small.

Note that the center in *Step 1* is ideally the center of gravity of the current polytope because a hyperplane passing through the center of gravity is guaranteed to reduce the volume of the polytope by a factor of $(1 - 1/e)$ in each iteration. However, because finding the center of gravity is prohibitively expensive in terms of computation, the analytic center is an acceptable approximation.

Example. The algorithm is illustrated by using it to solve the following problem in two dimensions:

$$\begin{aligned}&\text{minimize } f(x_1, x_2)\\ &\text{such that}(x_1, x_2) \in S\end{aligned}$$

where S is a convex set and f is a convex function. The shaded region in Fig. 10(a) is the set S, and the dotted lines show the level curves of f. The point $\mathbf{x}_{\text{opt}}$ is the solution to this problem. The expected solution region is first bounded by a rectangle with center $\mathbf{x}_c$, as shown in Fig. 10(a). The feasibility of $\mathbf{x}_c$ is then determined; in this case, it can be seen that $\mathbf{x}_c$ is infeasible. Hence, the gradient of the constraint function is used to construct a hyperplane through $\mathbf{x}_c$ such that the polytope is divided into two parts of roughly equal volume, one of which contains the solution $\mathbf{x}_c$. This is

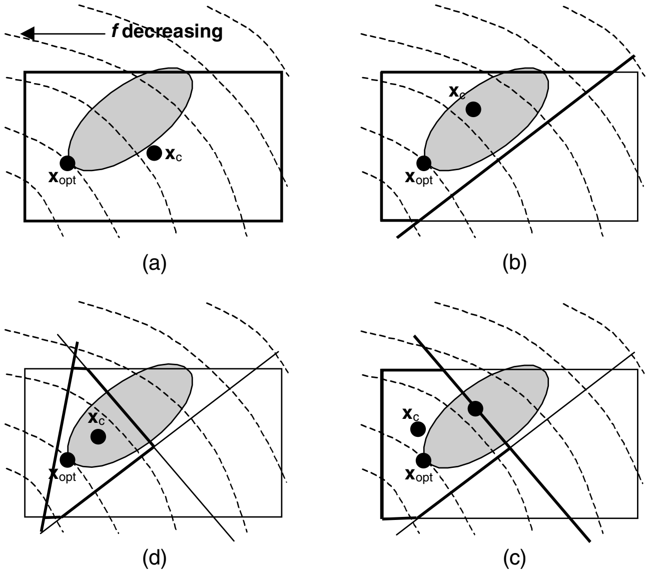


Figure 10. Cutting plane approach.

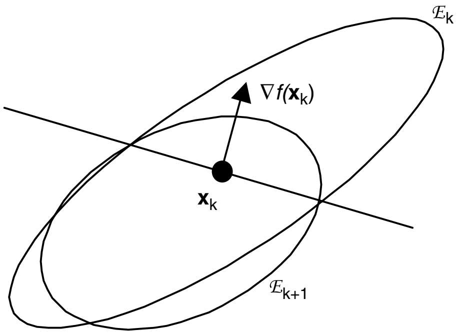


Figure 11. The ellipsoidal method.

illustrated in Fig. 10(b), where the region enclosed in darkened lines corresponds to the updated polytope. The process is repeated on the new smaller polytope. Its center lies inside the feasible region, and hence, the gradient of the objective function is used to generate a hyperplane that further shrinks the size of the polytope, as shown in Fig. 10(c). The result of another iteration is illustrated in Fig. 10(d). The process continues until the polytope has been shrunk sufficiently.

Ellipsoidal Method. The ellipsoidal method begins with a sufficiently large ellipsoid that contains the solution to the convex optimization problem. In each iteration, the size of the ellipsoid is shrunk, while maintaining the invariant that the solution lies within the ellipsoid, until the ellipsoid becomes sufficiently small. The process is illustrated in Fig. 11.

The kth iteration consists of the following steps, starting from the fact that the center $\mathbf{x}_k$ of the ellipsoid $\mathcal{E}_k$ is known.

Step 1. In case the center is not in the feasible region, the gradient of the violated constraint is evaluated; if it is feasible, the gradient of the objective function is found. In either case, we will refer to the computed gradient as $\nabla h(\mathbf{x}_k)$.

Step 2. A new ellipsoid containing the half-ellipsoid given by

$$\mathcal{E}_k \cap \{\mathbf{x} | \nabla h(\mathbf{x}_k)^T \mathbf{x} \leq \nabla h(\mathbf{x}_k)^T \mathbf{x}_k\}$$

is computed. This new ellipsoid $\mathcal{E}_{k+1}$ and its center $\mathbf{x}_{k+1}$ are given by the following relations:

$$\mathbf{X}_{k+1} = \mathbf{X}_k - \frac{1}{n+1} A_k \tilde{\mathbf{g}}_k$$
$$A_{k+1} = \frac{n^2}{n^2-1}(A_k - \frac{2}{n+1} A_k \tilde{\mathbf{g}}_k \tilde{\mathbf{g}}_k{}^T A_k)$$

where

$$\tilde{\mathbf{g}}_k = \frac{\nabla h(\mathbf{x}_k)}{\sqrt{\nabla h(\mathbf{x}_k)^T A_k \nabla h(\mathbf{x}_k)}}$$

Step 3. Repeat the iterations in Steps 1 and 2 until the ellipsoid is sufficiently small.

RELATED TECHNIQUES

Quasiconvex Optimization

Definition. A function $f : S \rightarrow \mathbf{R}$, where S is a convex set, is **quasiconvex** if every level set $L_a = \{\mathbf{x} | f(\mathbf{x}) \leq a\}$ is a convex set. A function g is **quasiconcave** if $-g$ is quasiconvex over S. A function is **quasilinear** if it is both quasiconvex and quasiconcave.

Examples. Clearly, any convex (concave) function is also quasiconvex (quasiconcave).

Any monotone function $f : \mathbf{R} \rightarrow \mathbf{R}$ is quasilinear.

The linear fractional function $f(\mathbf{x}) = (\mathbf{a}^T\mathbf{x} + b)/(\mathbf{c}^T\mathbf{x} + d)$, where $\mathbf{a}, \mathbf{c}, \mathbf{x} \in \mathbf{R}^n$, is quasilinear over the half-space $\{\mathbf{x} | \mathbf{c}^T\mathbf{x} + d > 0\}$.

Other Characterizations of Quasiconvexity. A function f defined on a convex set Ω is quasiconvex if, for every $\mathbf{x}_1$, $\mathbf{x}_2 \in \Omega$,

1. For every α, $0 \leq \alpha \leq 1$, $f(\alpha\,\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2) \leq \max[f(\mathbf{x}_1), f(\mathbf{x}_2)]$.
2. If f is differentiable, $f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \Rightarrow (\mathbf{x}_2 - \mathbf{x}_1)^T \nabla f(\mathbf{x}_1) \leq 0$.

Property. If f, g are quasiconvex over Ω, then the functions αf for $\alpha > 0$, and max(f,g) are also quasiconvex over Ω. The composed function $g(f(\mathbf{x}))$ is quasiconvex provided g is monotone increasing. In general, the function $(f+g)$ is *not* quasiconvex over Ω.

As in the case of convex optimization, the gradient of a quasiconvex objective can be used to eliminate a half-space from consideration. The work in Ref. (14) presents an adaptation of the ellipsoidal method to solve quasiconvex optimization problems.

Semidefinite Programming

The problem of semidefinite programming (15) is stated as follows:

$$\begin{array}{l} \text{minimize}\, \mathbf{c}^T\mathbf{x} \\ \text{subject to}\, F(\mathbf{x}) \geq 0 \\ \text{where}\, F(\mathbf{x}) \in \mathbf{R}^{m \times m},\ \mathbf{c},\ \mathbf{x} \in \mathbf{R}^n \end{array} \tag{4}$$

Here, $F(\mathbf{x}) = F_0 + F_1x_1 + \ldots + F_nX_n$ is an affine matrix function of $\mathbf{x}$ and the constraint $F(\mathbf{x}) \geq 0$ represents the fact that this matrix function must be positive semidefinite, i.e., $\mathbf{z}^T F(\mathbf{x})\mathbf{z} \geq 0$ for all $\mathbf{z} \in \mathbf{R}^n$. The constraint is referred to as a linear matrix inequality. The objective function is linear, and hence convex, and the feasible region is convex because if $F(\mathbf{x}) \geq 0$ and $F(\mathbf{y}) \geq 0$, then for all $0 \leq \lambda \leq 1$, it can be readily seen that $\lambda F(\mathbf{x}) + (1-\lambda)F(\mathbf{y}) \leq 0$.

A linear program is a simple case of a semidefinite program. To see this, we can rewrite the constraint set; $A\mathbf{x} \geq \mathbf{b}$ (note that the "$\geq$" here is a component-wise inequality and is not related to positive semidefiniteness), as $F(\mathbf{x}) = \mathbf{diag}(A\,\mathbf{x} - \mathbf{b})$; i.e., $F_0 = \mathbf{diag}(\mathbf{b})$, $F_{\mathrm{j}} = \mathbf{diag}(\mathbf{a}_{\mathrm{j}})$, $\mathrm{j} = 1, \ldots, n$, where $A = [\mathbf{a}_1\, \mathbf{a}_2\, \ldots \mathbf{a}_{\mathrm{n}}] \in \mathbf{R}^{m \times n}$.

Semidefinite programs may also be used to represent nonlinear optimization problems. As an example, consider the problem

$$\begin{array}{l} \text{minimize} \dfrac{(\mathbf{c}^T\mathbf{x})^2}{\mathbf{d}^T\mathbf{x}} \\ \text{subject to} A\mathbf{x} + \mathbf{b} \geq 0 \end{array}$$

where we assume that $\mathbf{d}^T\mathbf{x} > 0$ in the feasible region. Note that the constraints here represent, as in the case of a linear program, component-wise inequalities. The problem is first rewritten as minimizing an auxiliary variable t subject to the original set of constraints and a new constraint

$$\frac{(\mathbf{c}^T\mathbf{x})^2}{\mathbf{d}^T\mathbf{x}} \leq t$$

The problem may be cast in the form of a semidefinite program as

$$\begin{array}{ll} & \text{minimize } t \\ \text{subject to} & \begin{bmatrix} \mathbf{diag}(A\mathbf{x} + \mathbf{b}) & 0 & 0 \\ 0 & t & \mathbf{c}^T\mathbf{x} \\ 0 & \mathbf{c}^T\mathbf{x} & \mathbf{d}^T\mathbf{x} \end{bmatrix} \geq 0 \end{array}$$

The first constraint row appears in a manner similar to the linear programming case. The second and third rows use Schur complements to represent the nonlinear convex constraint above as the 2 × 2 matrix inequality

$$\begin{bmatrix} t & \mathbf{c}^T\mathbf{x} \\ \mathbf{c}^T\mathbf{x} & \mathbf{d}^T\mathbf{x} \end{bmatrix} \geq 0$$

The two "tricks" shown here, namely, the reformulation of linear inequations and the use of Schur complements, are often used to formulate optimization problems as semidefinite programs.

Geometric Programming

Definition. A **posynomial** is a function h of a positive variable $\mathbf{x} \in \mathbf{R}^m$ that has the form

$$h(\mathbf{x}) = \sum_{\mathrm{j}} \gamma_{\mathrm{j}} \prod_{i=1 \text{ to } n} x_i^{\alpha(i,j)}$$

where the exponents $\alpha(i, j) \in \mathbf{R}$ and the coefficients $\gamma_j > 0$, $\gamma_j \in \mathbf{R}$.

For example, the function $f(x,y,z) = 7.4\,x + 2.6\,y^{3.18}z^{-4.2} + \sqrt{3}\,y^{-2}y^{-1.4}z^{\sqrt{5}}$ is a posynomial in the variables x, y, and z. Roughly speaking, a posynomial is a function that is similar to a polynomial, except that the coefficients γ_j must be positive, and an exponent $\alpha(i, j)$ could be any real number and not necessarily a positive integer, unlike a polynomial.

A posynomial has the useful property that it can be mapped onto a convex function through an elementary variable transformation, $x(i) = e^{z(i)}$(16). Such functional forms are useful because in the case of an optimization problem where the objective function and the constraints are posynomial, the problem can easily be mapped onto a convex programming problem.

For some geometric programming problems, simple techniques based on the use of the arithmetic-geometric inequality may be used to obtain simple closed-form solutions to the optimization problems (17). The arithmetic–geometric inequality states that if $u_1, u_2, \ldots, u_n > 0$, then the arithmetic mean is no smaller than the geometric mean; i.e.,

$$(u_1 + u_2 + \ldots + u_n)/n \geq ((u_1)(u_2) \ldots (u_n))^{1/n},$$

with equality occurring if and only if $u_1 = u_2 = \ldots = u_n$.

A simple illustration of this technique is in minimizing the outer surface area of an open box of fixed volume (say, four units) and sides of length x_1, x_2, x_3. The problem can be stated as

$$\begin{array}{ll} \text{minimize} & x_1x_2 + 2\,x_1x_3 + 2\,x_1x_3 \\ \text{subject to} & x_1x_2x_3 = 4 \end{array}$$

By setting $u_1\ x_1, x_2,\ u_2 = 2x_1\,x_3,\ u_3 = 2x_1x_3$, and applying the condition listed above, the minimum value of the objective function is $3((u_1)(u_2)(u_3))^{1/3} = 3(4\,x_1^2x_2^2x_3^2) = 12$. It is easily verified that this corresponds to the values $x_1 = 2$, $x_2 = 2$, and $x_3 = 1$. We add a cautionary note that some, but not all, posynomial programming problems may be solved using this simple solution technique. For further details, the reader is referred to Ref. (17).

An extension of posynomials is the idea of *generalized posynomials* (18), which are defined recursively as follows:

A generalized posynomial of order 0, $G_0(\mathbf{x})$, is simply a posynomial, defined earlier; i.e.,

$$G_0(\mathbf{x}) = \sum_j \gamma_{\mathrm{j}} \prod_{i=1 \text{ to } n} x_i^{\alpha(i,j)}$$

A generalized posynomial of order $k > 1$ is defined as

$$G_k(\mathbf{x}) = \sum_j \lambda_j \prod_{i=1 \text{ to } n} [G_{(k-1),i}(\mathbf{x})]^{\alpha(i,j)}$$

where $G_{(k-1),i}(\mathbf{x})$ is a generalized posynomial of order less than or equal to $k - 1$, each exponent $\alpha(i, j) \in \mathbf{R}$, and the coefficients $\gamma_j > 0$, $\gamma_j \in \mathbf{R}$.

Like posynomials, generalized posynomials can be transformed to convex functions under the transform $x(i) = e^{z(i)}$, but with the additional restriction that $\alpha(i, j) \geq 0$ for each i, j. For a generalized posynomial of order 1, observe that the term in the innermost bracket, $G_{0,i}(\mathbf{x})$, represents a posynomial function. Therefore, a generalized posynomial of order 1 is similar to a posynomial, except that the place of the $x(i)$ variables is taken by posynomial. In general, a generalized posynomial of order k is similar to a posynomial, but $x(i)$ in the posynomial equation is replaced by a generalized posynomial of a lower order.

Optimization Involving Logarithmic Concave Functions

A function *f* is a logarithmic concave (log-concave) function if log(*f*) is a concave function. Logconvex functions are similarly defined. The maximization of a log-concave function over a convex set is therefore a unimodal problem; i.e., any local minimum is a global minimum. Log-concave functional forms are seen among some common probability distributions on $\mathbf{R}^n$, for example:

(a) The Gaussian or normal distribution

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \det\Sigma}} e^{-0.5(\mathbf{x}-\mathbf{x}_c)^T\Sigma^{-1}(\mathbf{x}-\mathbf{x}_c)}$$

where $\Sigma \geq 0$.

(b) The exponential distribution

$$f(\mathbf{x}) = \left(\prod_{i=1\,\mathrm{to}\,n} \lambda(i)\right) e^{-(\lambda(1)x(1)+\lambda(2)x(2)+\ldots+\lambda(n)x(n))}$$

The following properties are true of log-concave functions:

1. If *f* and *g* are log-concave, then their product *f*.*g* is log-concave.
2. If $f(\mathbf{x})$ is log-concave, then the integral $\int_S f(\mathbf{x})d\mathbf{x}$ is log-concave provided S is a convex set.
3. If $f(\mathbf{x})$ and $g(\mathbf{x})$ are log-concave, then the convolution $\int_S f(\mathbf{x}-\mathbf{y})g(\mathbf{y})d\mathbf{y}$ is log-concave if S is a convex set (this follows from Steps 1 amd 2).

Convex Optimization Problems in Computer Science and Engineering

There has been an enormous amount of recent interest in applying convex optimization to engineering problems, particularly as the optimizers have grown more efficient. The reader is referred to Boyd and Vandenberghe's lecture notes (19) for a treatment of this subject. In this section, we present a sampling of computer science and computer engineering problems that can be posed as convex programs to illustrate the power of the technique in practice.

Design Centering

While manufacturing any system, it is inevitable that process variations will cause design parameters, such as component values, to waver from their nominal values. As a result, after manufacture, the system may no longer meet some behavioral specifications, such as requirements on the delay, gain, and bandwidth, which it has been designed to satisfy. The procedure of design centering attempts to select the nominal values of design parameters to ensure that the behavior of the system remains within specifications, with the greatest probability and thus maximize the manufacturing yield.

The random variations in the values of the design parameters are modeled by a probability density function, $\psi(\mathbf{x}, \mathbf{x}_c) : \mathbf{R}^n \to [0, 1]$, with a mean corresponding to the nominal value of the design parameters. The yield of the manufacturing process Y as a function of the mean $\mathbf{x}_c$ is given by

$$Y(\mathbf{x}_c) = \int_{\mathcal{F}} \psi(\mathbf{x}, \mathbf{x}_c)d\mathbf{x}$$

where $\mathcal{F}$ corresponds to the feasible region where all design constraints are satisfied.

A common assumption made by geometrical design centering algorithms for integrated circuit applications is that $\mathcal{F}$ is a convex bounded body. Techniques for approximating this body by a polytope P have been presented in Ref. (20). When the probability density functions that represent variations in the design parameters are Gaussian in nature, the design centering problem can be posed as a convex programming problem. The design centering problem is formulated as (21)

$$\text{maximize}\, Y(\mathbf{x}_c) = \int_P \psi(\mathbf{x}, \mathbf{x}_c)d\mathbf{x}$$

where $\mathcal{P}$ is the polytope approximation to the feasible region $\mathcal{R}$. As the integral of a logconcave function over a convex region is also a log-concave function (22), the yield function $Y(\mathbf{x})$ is log-concave, and the above problem reduces to a problem of maximizing a log-concave function over a convex set. Hence, this can be transformed into a convex programming problem.

VLSI Transistor and Wire Sizing

Convex Optimization Formulation. Circuit delays in integrated circuits often have to be reduced to obtain faster response times. Given the circuit topology, the delay of a combinational circuit can be controlled by varying the sizes of transistors, giving rise to an optimization problem of finding the appropriate area-delay tradeoff. The formal statement of the problem is as follows:

$$\begin{array}{l} \text{minimize}\, \mathit{Area} \\ \text{subject to}\, \mathit{Delay} \leq T_{\mathrm{spec}} \end{array} \tag{5}$$

The circuit area is estimated as the sum of transistor sizes; i.e.,

$$\mathit{Area} = \sum_{i=1\,\mathrm{to}\,n} x_i$$

where x_i is the size of the ith transistor and n is the number of transistors in the circuit. This is easily seen to be a posynomial function of the x_i's. The circuit delay is estimated using the Elmore delay estimate (23), which calculates the delay as maximum of path delays. Each path delay is a sum of resistance-capacitance products. Each resistance term is of the form a/x_i, and each capacitance term is of the type $\Sigma b_i x_i$, with the constants a and b_i being positive. As a result, the delays are posynomial functions of the x_i's, and the feasible region for the optimization problem is an intersection of constraints of the form

$$(\textit{posynomial function in}\, x_i\text{'s}) \leq t_{\text{spec}}$$

As the objective and constraints are both posynomial functions in the x_i's, the problem is equivalent to a convex programming problem. Various solutions to the problem have been proposed, for instance, in Refs. (24) and (25). Alternative techniques that use curve-fitted delay models have been used to set up a generalized posynomial formulation for the problem in Ref. (21).

Semidefinite Programming Formulation. In Equation (5), the circuit delay may alternatively be determined from the dominant eigenvalue of a matrix $G^{-1}C$, where G and C are, respectively, matrices representing the conductances (corresponding to the resistances) and the capacitances referred to above. The entries in both G and C are affine functions of the x_i's. The dominant time constant can be calculated as the negative inverse of the largest zero of the polynomial $\det(s\ C + G)$. It is also possible to calculate it using the following linear matrix inequality:

$$T^{\text{dom}} = \min\{T | TG - C \geq 0\}$$

Note that the "≥ 0" here refers to the fact that the matrix must be positive definite. To ensure that $T^{\text{dom}} \leq T_{\max}$ for a specified value of $T_{\max}$, the linear matrix inequality $T_{\max} = G(\mathbf{x}) - C(\mathbf{x}) \geq 0$ must be satisfied. This sets up the problem in the form of a semidefinite program as follows (26):

$$\begin{aligned}
&\text{minimize} \sum_{i=1}^{n} l_i x_i \\
&\text{subject to}\, T_{\max} G(\mathbf{x}) - C(\mathbf{x}) \geq 0 \\
&\mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}
\end{aligned}$$

Largest Inscribed Ellipsoid in a Polytope

Consider a polytope in $\mathbf{R}^n$ given by $\mathcal{P} = \{\mathbf{x} | \mathbf{a}_i^T \mathbf{x} \leq b_i,\, i = 1, 2, \ldots, L\}$ into which the largest ellipsoid $\mathcal{E}$, described as follows, is to be inscribed:

$$\mathcal{E} = \{B\ \mathbf{y} + \mathbf{d} | \|\mathbf{y}\| \leq 1\}, B = B^T > 0$$

The center of this ellipsoid is $\mathbf{d}$, and its volume is proportional to $\det(B)$. The objective here is to find the entries in the matrix B and the vector $\mathbf{d}$. To ensure that the ellipsoid is contained within the polytope, it must be ensured that for all $\mathbf{y}$ such that $\|\mathbf{y}\| \leq 1$,

$$\mathbf{a}_i^T (B\mathbf{y} + \mathbf{d}) \leq b_i$$

Therefore, it must be true that $sup_{\|\mathbf{y}\| \leq 1} (\mathbf{a}_i^T B\ \mathbf{y} + \mathbf{a}_i^T \mathbf{d}) \leq \mathrm{b}_i$, or in other words, $\|B\, \mathbf{a}_i\| \leq b_i - \mathbf{a}_i^T \mathbf{d}$. The optimization problem may now be set up as

$$\begin{aligned}
&\text{maximize log det}\, B \\
&\text{subject to}\, B = B^T > 0 \\
&\|B\, \mathbf{a}_i\| \leq b_i - \mathbf{a}_i\, \mathbf{d},\ \mathrm{i} = 1, 2, \ldots, L
\end{aligned}$$

This is a convex optimization problem (6) in the variables B and $\mathbf{d}$, with a total dimension of $n(n+1)/2$ variables corresponding to the entries in B and n variables corresponding to those in $\mathbf{d}$.

CONCLUSION

This overview has presented an outline of convex programming. The use of specialized techniques that exploit the convexity properties of the problem have led to rapid recent advances in efficient solution techniques for convex programs, which have been outlined here. The applications of convex optimization to real problems to engineering design have been illustrated.

BIBLIOGRAPHY

1. W. Stadler, *Multicriteria Optimization in Engineering and in the Sciences*, New York: Plenum Press, 1988.
2. N. Karmarkar, A new polynomial-time algorithm for linear programming, *Combinatorica*, **4**: 373–395, 1984.
3. A. V. Fiacco and G. P. McCormick, *Nonlinear Programming*, New York: Wiley, 1968.
4. J. Renegar, A Polynomial Time Algorithm, based on Newton's method, for linear programming, *Math. Programming,* **40**: 59–93, 1988.
5. C. C. Gonzaga, An algorithm for solving linear programming problems in *O(nL)* operations, in N. Meggido, (ed.), *Progress in Mathematical Programming: Interior Point and Related Methods,*, New York: Springer-Verlag, 1988, PP. 1–28,.
6. Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM Studies in Applied Mathematics series, Philadelphia, PA: Society for Industrial and Applied Mathematics, 1994.
7. P. M. Vaidya, An algorithm for linear programming which requires $O(((m+n)n^2 + (m+n)^{1.5}n)L)$ arithmetic operations, *Math. Programming*, **47**: 175–201, 1990.
8. Y. Ye, An $O(n^3L)$ potential reduction algorithm for linear programming, *Math. Programming*, **50**: 239–258, 1991.
9. R. T. Rockafellar, *Convex Analysis*, Princeton, NJ:Princeton University Press, 1970.
10. D. G. Luenberger, *Linear and Nonlinear Programming*, Reading, MA: Addison-Wesley, 1984.
11. P. E. Gill, W. Murray and M. H. Wright, *Numerical Linear Algebra and Optimization*, Reading, MA: Addison-Wesley, 1991.
12. A. J. Schrijver, *Theory of Linear and Integer Programming*, New York: Wiley, 1986.
13. D. den Hertog, *Interior Point Approach to Linear, Quadratic and Convex Programming,* Boston, MA: Kluwer Academic Publishers, 1994.
14. J. A. dos Santos Gromicho, *Quasiconvex Optimization and Location Theory*, Amsterdam, The Netherlands:Thesis Publishers, 1995.
15. L. Vandenberghe and S. Boyd, Semidefinite programming, *SIAM Rev.*, **38** (1): 49–95, 1996.
16. J. G. Ecker, Geometric programming: methods, computations and applications, *SIAM Rev.*, **22** (3): 338–362, 1980.
17. R. J. Duffin and E. L. Peterson, *Geometric Programming: Theory and Application*, New York: Wiley, 1967.

18. K. Kasamsetty, M. Ketkar, and S. S. Sapatnekar, A new class of convex functions for delay modeling and their application to the transistor sizing problem, *IEEE Trans. on Comput.-Aided Design*, **19** (7): 779–778, 2000.

19. S. Boyd and L. Vandenberghe, *Introduction to Convex Optimization with Engineering Applications*, Lecture notes, Electrical Engineering Department, Stanford University, CA, 1995. Available http://www-isl.stanford.edu/~boyd.

20. S. W. Director and G. D. Hachtel, "The simplicial approximation approach to design centering," *IEEE Trans. Circuits Syst.*, **CAS-24** (7): 363–372, 1977.

21. S. S. Sapatnekar, P. M. Vaidya and S. M. Kang, Convexity-based algorithms for design centering, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, **13** (12): 1536–1549, 1994.

22. A. Prekopa, Logarithmic concave measures and other topics, in M. Dempster, (ed.), *Stochastic Programming*, London, England: Academic Press, 1980, PP. 63–82.

23. S. S. Sapatnekar and S. M. Kang, *Design Automation for Timing-driven Layout Synthesis*, Boston, MA:Kluwer Academic Publishers, 1993.

24. J. Fishburn and A. E. Dunlop, TILOS: a posynomial programming approach to transistor sizing, *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1985, PP. 326–328.

25. S. S. Sapatnekar, V. B. Rao, P. M. Vaidya and S. M. Kang, An exact solution to the transistor sizing problem using convex optimization, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, **12** (11): 1621–1632, 1993.

26. L. Vandenberghe, S. Boyd and A. El Gamal, Optimal wire and transistor sizing for circuits with non-tree topology, *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1997, PP. 252–259.

Sachin S. Sapatnekar
University of Minnesota
Minneapolis, Minnesota

D

DYNAMIC PROGRAMMING

The objective of the DP algorithm is to solve the sequential decision problem optimally. The problem typically uses an automaton model. Therefore, we start by constructing an automaton model of a sequential decision process. Then the principle of optimality is explained as a basic and driving concept to establish a DP algorithm for the sequential decision process. To explain how the principle works in establishing the algorithm, we focus on the relationship between the representation of the automaton and its optimal solution algorithm by DP. At this stage, the DP algorithm still seems abstract. We describe a more concrete DP algorithm to optimally solve a so-called nonlinear time alignment problem, which is widely used in many application domains. The relationship between an automaton model and the DP algorithm of nonlinear time alignment is also described.

Many types of DP algorithms have been developed due to the various applications that are possible. We use a DP algorithm as an example to show how a basic DP algorithm can be extended to create another algorithm applicable to another field.

This article comprises sections that describe these subtopics: (*1*) automaton model of sequential decision process as a simple framework for dealing with DP, (*2*) principle of optimality as key concept of DP, (*3*) baseline algorithm as a simplest realization of DP, (*4*) time-alignment algorithm as one of the most popular applications using DP, (*5*) numerical computation of time-alignment algorithm for understanding each step of computation, (*6*) continuous dynamic programming as a useful extension of time-alignment algorithm, and (*7*) other applications including Markov decision process.

AUTOMATON MODEL OF SEQUENTIAL DECISION PROCESS

A sequential decision process can be modeled by an automaton composed of four components: state S_k decision $q_k \in Q$, state transition $S_k = T_k(S_{k-1}, q_k)$, and gain $g_k(S_{k-1}, q_k)$, where Q is a set of decisions and $1 \leq k \leq K$ Then the total gain function of the so-called additive gain model is defined as follows and is used to evaluate the decision sequence:

$$G = \sum_{k=1}^{K} g_k(S_{k-1}, q_k)$$

The additive gain model is widely used in real application domains. The relationship of these four components and the total gain G is shown in Fig. 1.

Dynamic programming can be understood simply through the procedure for developing an algorithm that gives the maximum value of G. A dynamic programming algorithm is a realization of the so-called principle of optimality proposed by R. Bellman (1, 2); therefore, that principle is described first.

PRINCIPLE OF OPTIMALITY

Understanding the principle of optimality is heuristically easy. Assume that the path formed by A and B as shown in Fig. 2, is *the full optimal path* uniquely determined from initial state S_0 to terminal state S_k and including state S_k. If two other optimal paths exist, namely A' (not A) from S_0 to S_k and B' (not B) from S_k to S_K, then the path formed by the two paths A' and B' becomes another full optimal path from S_0 to S_K. This mechanism contradicts the assumption that the path formed by A and B is the full optimal path uniquely determined from S_0 to S_K. The principle of optimality guarantees that an arbitrary state S_k on the full optimal path from S_0 to S_K can uniquely divide a full optimal path into two optimal partial paths.

Formal Definition of the Principle of Optimality

Let q_1 be an initial decision. Then the optimal path to reach state S_{k-1} gives a maximum gain $G^*_{k-1}(S_0, q_1)$ for $2 \leq k \leq K$ The principle of optimality defines the following equation for $2 \leq k \leq K$:

$$G^*_k(S_0) = \max_{q_1 \in Q} G^*_{k-1}(S_0, q_1)$$

The equation for $k = 1$ is defined using a gain function $G_1(S_0, q_1)$ by

$$G^*_1(S_0) = \max_{q_1 \in Q} G_1(S_0, q_1)$$

The maximum gain, $G^*_K(S_0)$, has only two parameters, K and S_0 [see (3)].

The Curse of Dimensionality

When obtaining an optimal solution of optimization problem, DP can reduce the computational burden compared with that of naÿve algorithms. But the computational burden of DP increases rapidly with numbers of states and decisions because the problem domain is still combinatorial. This is called “the curse of dimensionality.” Some methods (4,5) have been proposed to avoid the obstacle in practical applications.

BASE-LINE DP ALGORITHM OF ITERATIVE COMPUTATION

When applying the principle of optimality to an automaton of a sequential decision process, as shown in Fig. 1, we can obtain the following iterative computation.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.
Copyright © 2008 John Wiley & Sons, Inc.

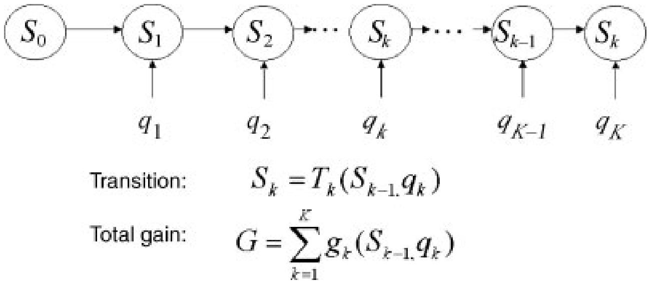

S_0
S_1
S_2
S_k
S_{k-1}
S_k
q_1
q_2
q_k
q_{K-1}
q_K
Transition: $S_k = T_k(S_{k-1.}q_k)$
Total gain: $G = \sum_{k=1}^{K} g_k(S_{k-1.}q_k)$

Figure 1. Automaton model of sequential decision process.

First, we describe total gain G of an additive gain model as follows:

$$G = g_1(S_0, q_1) + g_2(S_1, q_2) + g_3(S_2, q_3) + \cdots \\ + g_K(S_{K-1}, q_K)$$

The states $S_k, k = 0, 1, 2, \ldots, K-1$, are not the states on the full optimal path shown in Fig. 2. A state S_k in this section is regarded as a *working* state.

Second, let G_1 be g_1+g_2 assuming that state S_0 in G is given. Then G_1 has two variables, q_1 and q_2, recalling that $S_1 = T_1(S_0, q_1)$. If G_1 is maximized with respect to q_1 assuming that q_2 is a fixed value, then $G_1^*(q_2) = \max_{q_1} G_1(q_1, q_2)$ and $q_1^* = q_1^*(q_2) = \arg\{\max_{q_1} G_1(q_1, q_2)\}$ are obtained, where the superscript asterisks denote optimality. Let G_2 be $G_1^*(q_2) + g_3(S_2, q_3)$; then G_2 has two variables, q_2 and q_3, recalling that $S_2 = T_2(S_1, q_2)$. If we maximize G_2 with respect to q_2 assuming that q_3 is a fixed value, then $G_2^*(q_3) = \max_{q_2} G_2(q_2, q_3)$ and $q_2^* = q_2^*(q_3) = \arg\{\max_{q_2} G_2(q_2, q_3)\}$ are obtained. Then the following equation holds for k, $2 \leq k \leq K-1$:

$$G_k^*(q_{k+1}) = \max_{q_k}\{G_{k-1}^*(q_k) + g_{k+1}(S_{k.}q_{k+1})\}$$

where $q_k^* = q_k^*(q_{k+1}) = \arg\{\max_{q_k} G_k(q_k, q_{k+1})\}$

Third, let us consider the case where $k = K-1$. The function $G_{K-1}(= G_{K-2}^*(q_{K-1}) + g_K(S_{K-1}, q_K))$ takes into account all gain functions, and G_{K-1} has two variables q_{K-1}, q_K. If we maximize G_{K-1} with respect to q_{K-1} assuming that q_K is a fixed value, then $G_{K-1}^*(q_K) = \max_{q_{K-1}} G_{K-1}(q_{K-1}, q_K)$ and $q_{K-1}^* = q_{K-1}^*(q_K) = \arg\{\max_{q_{K-1}} G_{K-1}(q_{K-1}, q_K)\}$ are obtained. Here, q_K is still a variable.

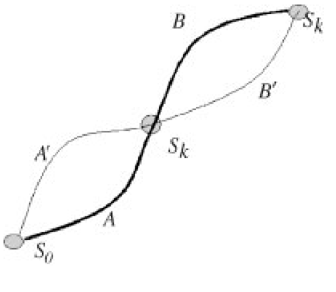


Figure 2. Schematic diagram of the principle of optimality.

Finally, optimal value q_K^* is determined by maximizing $G_{K-1}^*(q_K)$ with respect to q_K under the constraints of both terminal state S_K and transition $S_K = T_K(S_{K-1}, q_K)$ if such constraints are given. Here, q_K^* is not a variable but a *value*. After optimal value q_K^* is determined, q_{K-1}^* is determined by $q_{K-1}^* = q_{K-1}^*(q_K^*)$. Here, q_{K-1}^* is also no longer a function of q_K, being different from that of forward computation. This backward computation continues to determine values $q_{K-2}^*, q_{K-3}^*, q_{K-4}^* \ldots, q_1^*$. At the same time, the optimal state sequence $S_0, S_1^*, S_2^*, S_3^*, \ldots, S_{K-1}^*, S_K^*$, is determined by using $S_k^* = T_k(S_{K-1}^*, q_k^*), k = 1, 2, \ldots, K$ where $S_0^* = S_0$. The maximum value of G,G,* is also determined by using q_k^* and S_k^*, $k = 1, 2, \ldots, K$.

The above iterative procedure considers all possible combinations of $q_1, q_2, q_3, \ldots, q_{K-1}, q_K$ to obtain the maximum value of G applying the principle of optimality. It is also confirmed when we consider that the sequence $S_0, S_1, S_2, S_3, \ldots, S_{K-1}, S_K$ in Fig. 2 is taken by the sequence $S_0, S_1^*, S_2^*, S_3^*, \ldots, S_{K-1}^*, S_K^*$ obtained in this section. In other words, each kth step of the iterative computation discovers state S_k on the full optimal path in Fig. 2 satisfying the principal of optimality.

The key factor for understanding the procedural aspects of iterative DP computation consists of three parts:

1) consideration of G_k with two variables, q_{k-1} and q_k,
2) treatment of q_k as a fixed value when determining the optimum q_{k-1} in $G_k(q_{k-1}, q_k)$ in the forward computation, and
3) backward determination of values q_k^* from $k = K-1$ to $k = 1$ using the formula $q_k^* = q_k^*(q_{k+1}^*) = \arg\{\max_{q_k} G_k(q_k, q_{k+1}^*)\}$,where $q_k^*(q_k+1)$ and $G_k(q_k, q_{k+1})$ should be stored (memoization) at each kth step of the forward computation.

Constraints

We must consider two kinds of constraints, local and global, when establishing an iterative computation of DP. The constraints model the requirements provided by each application.

1) A local constraint is represented by $T_k(S_{k-1}, q_k)$. It determines the relationship between S_{k-1} and q_k to reach S_k.
2) A global constraint is represented by spatial allocation or algebraic structure of states $\{S_0, S_1, S_2, S_3, \ldots, S_{K-1}, S_K\}$. It is independent of local constraints. Assignment of conditions for S_0 and S_K is a global constraint.

Because the explanations in this section may not be sufficient to gain a thorough understanding of DP, we present more concrete examples.

TIME ALIGNMENT

In the fields of speech recognition (6), biological information processing (7), and other applications including artificial intelligence and information retrieval, so-called nonlinear

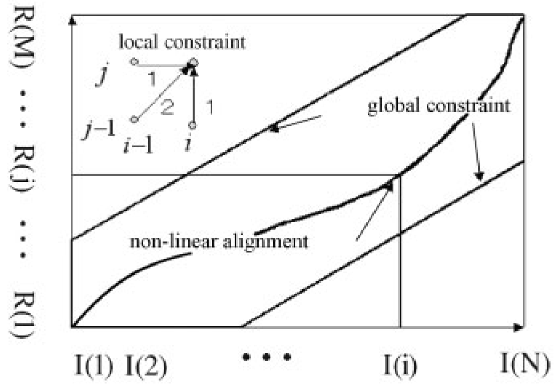


Figure 3. Nonlinear time alignment using dynamic programming.

alignment algorithms based on DP are widely used. Each of them is a different realization of dynamic programming for a different sequential decision problem.

We consider a simple algorithm of a time alignment problem to explain how DP is implemented in such application domains. The main objective of this time alignment problem is to obtain an optimal correspondence between two time sequences that minimizes the total sum of the mutual distance by choosing optimal time pairs from two time sequences under local and global constraints, as shown in Fig. 3. In the figure, $\{I(i) : 1 \leq i \leq N\}$ is an input time sequence of a feature vector, whereas $\{R(j) : 1 \leq j \leq M\}$ is another sequence of a feature vector called a reference time sequence. The bold line in Fig. 3 shows a sequence of optimal pairs composed of $(i_k^*, j_k^*), (k = 1, 2, \ldots, K)$, where the optimality means minimization of the total sum of distance function, considering possible pairs (i, j) under the following conditions:

1) Boundary conditions $(i_1^*, j_1^*) = (1, 1)$ and $(i_K^*, j_K^*) = (N, M)$ are satisfied,
2) Local paths are determined using a local constraint, and
3) The optimal path must be included in the area determined by global constraint.

The nonlinear time-alignment problem belongs to the class of sequential decision problems, and its solution creates an automaton as the other representation mentioned above. The automaton and its DP algorithm are two representations of a sequential decision process, and they are coupled tightly with each other.

Through the correspondence between them we can understand how DP is actually implemented. Therefore, we make an automaton of a nonlinear time-alignment problem by taking into account the correspondences between components of the two representations.

First, state S_k of the automaton of Fig. 1 corresponds to a location (i,j) on a two-dimensional array of the alignment problem, where i and j are discrete points on time axes of input and reference time sequences, respectively. A decision q_k of the automaton of Fig. 1 corresponds to selection of a path among a set of local paths to reach location (i, j). In Fig. 3, the set of local paths consists of three paths, namely, paths from location $(i-1, j), (i-1, j-1)$, or $(i, j-1)$ to (i, j). The selection of a path is the result of evaluating each path based on a criterion described later.

Second, a transition $S_k = T_k(S_{k-1}, q_k)$ of Fig. 1 corresponds to a transition from one of three locations $(i-1, j), (i-1, j-1), (i, j-1)$, to location (i,j) by selecting a path, and S_{k-1} corresponds to all three locations.

Finally, we consider a distance function instead of a gain function $g_k(S_{k-1}, q_k)$. Here, we recall the automaton model of the sequential decision process and its iterative computation of dynamic programming mentioned above. Let us assign a variable G_k^* of the iterative computation to a state S_k of the automaton model. Then a similar assignment for the alignment problem leads to the correspondence of a variable $D(i,j)$ to a location (i, j) by introducing $D(i, j)$ as a minimum accumulated distance. To introduce $D(i,j)$, we use the normalized sum of weighted distance, defined by

$$J = \frac{1}{W}\sum_{k=1}^{K} w_k d_k(i_k, j_k)$$

The boundary conditions, $(i_1, j_1) = (1, 1), (i_K, j_K) = (N, M)$, and the total sum of weights $W = \sum w_k$ are used. If we use weight w_k attached to each local path shown in Fig. 3, then value W takes a constant value $N+M$ for an arbitrary path from $(i_1, j_1) = (1, 1)$ to $(i_K, j_K) = (N, M)$ because of the property of rectilinear city block distance. Therefore, weight W_k takes a value of 1 for transition from $(i-1, j)$ or $(i, j-1)$ to (i, j) and 2 for transition from $(i-1, j-1)$ to (i, j), as shown in Fig. 3, because weights attached to local paths are parts of rectilinear city block distance. The term $w_k d_k(i_k, j_k)$ corresponds to a gain function $g_k(S_{k-1}, q_k)$. In a real application, $d(i, j) = d(I(i), R(j))$ is usually defined by a distance measure between feature vectors $I(i)$ and $R(j)$. Then we can obtain the following iterative form of dynamic programming regarding $D(i, j)$ along with the transition of the state by choosing one of the following paths:

$$D(i, j) = \min\begin{cases} D(i-1, j) + d(i, j) \\ D(i-1, j-1) + 2d(i, j) \\ D(i, j-1) + d(i, j) \end{cases}$$

The boundary condition of the above equation is $D(0, j) = D(i, 0) = \infty; i = 1, 2, \ldots, N; j = 1, 2, \ldots, M$. The role of decision q_k in the automaton model is regarded as selection of the minimum of three values in the alignment model. In other words, decision q_k does not appear explicitly in the alignment model. The selection is determined by finding the minimum among three candidates, each of which is composed of $D(i, j)$ and $d(i, j)$.

Local Constraint and Semimonotonic Property

Generally, the form of iterative computation depends mainly on the type of local constraint. The local constraint shown in Fig. 3 has the property of semimonotonic increase of the location parameter (i, j) in local transition: $i_k \leq i_{k+1}$ and $j_k \leq j_{k+1}$ hold for any transition from (i_k, j_k) to (i_{k+1}, j_{k+1}). The semimonotonic property of a time parameter is a natural characteristic of a time sequence because of causality. Embedding the property into the local constraint guarantees retention of the property of a semimonotonic increase in time parameters in optimally corresponding sequences. In most real applications, we must consider a global constraint for the states to properly execute the iterative computation for *D(i, j)* and eliminate the effects of irregular situations. In Fig. 3, the global constraint determines the area bordered by six solid lines in which actual working states are allowed, and such constraint is realized by the assignment of an infinite value for all $D(i,j)$ beyond the area during the computation.

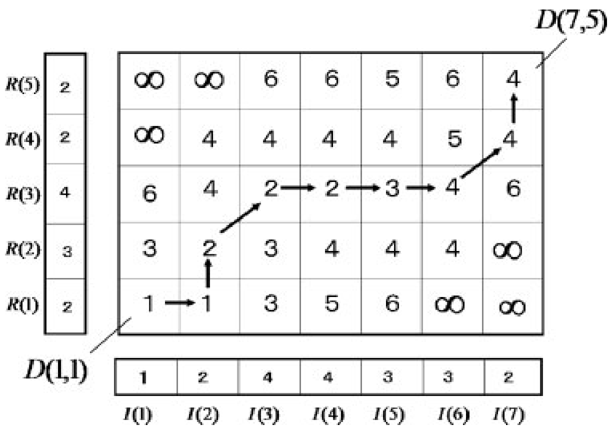
Recognition

The value $\frac{1}{M+N}D(N,M)$ becomes the output, which indicates the minimum value of J obtained by choosing the optimal path from $(i_1,j_1)=(1,1)$ to $(i_K,j_K)=(N,M)$. When m reference sequences are used to represent m classes and one input sequence, then m output values are obtained. These m values can be compared and the minimum found to determine the nearest class because their values are normalized for different lengths of reference sequence.

NUMERICAL EXAMPLE

Figure 4 shows a simple numerical example of nonlinear time alignment. The input and reference time sequences are $I(i)=1,2,4,4,3,3,2$ and $R(j)=2,3,4,2,2$, respectively. The distance measure is defined by $d(i,j)=|I(i)-R(j)|$, and the start and end states are given by $(i_1,\ j_1)=(1,1)$ and $(i_k,j_k)=(7,5)$. A global constraint for the states is modeled by the assignment of infinite value ∞ around two corners of the array. Figure 4 shows a 5×7 array of values assigned to $D(i,j)$ as the result of iterative computation, where parameter i indicates the horizontal axis, and parameter j indicates the vertical axis and counts from the bottom of the array.

The numerical computation takes the following procedure. We use a variable of 6×8 array including marginal components, namely, $D(i,j): i=0,1,2,3,4,5,6,7;$ $j=0,1,2,3,4,5$. The initial condition of marginal components of $D(i,j)$ is determined by $D(i,0)=\infty, i=0,1,2,3,4,5,6,7$ and $D(0,j)=\infty, j=0,1,2,3,4,5$.



R(5)	2	∞	∞	6	6	5	6	4
R(4)	2	∞	4	4	4	4	5	4
R(3)	4	6	4	2	2	3	4	6
R(2)	3	3	2	3	4	4	4	∞
R(1)	2	1	1	3	5	6	∞	∞
		1	2	4	4	3	3	2
		I(1)	I(2)	I(3)	I(4)	I(5)	I(6)	I(7)

Figure 4. Numerical example of time alignment.

To avoid extreme nonlinear alignment we establish a global constraint such as $D(1,4)=D(1,5)=D(2,5)=D(6,1)=D(7,1)=D(7,2)=\infty$. We initially determine $D(1,1)=|I(1)-R(1)|=|1-2|=1$. Then an iterative computation starts from $i=1$ to $i=7$ for $j=,1,2,3,4,5$.

In the case of $i=1$ and $j=2,3,4,5$, the following equation

$$D(1,j)=\min\begin{cases}D(0,j)+d(1,j)\\ D(0,j-1)+2d(1,j)\\ D(1,j-1)+d(1,j)\end{cases}$$

determines the values of the first column $(i=1)$, namely, $D(1{,}2)=1{+}2=3$, $D(1{,}3)=3{+}3=6, D(1{,}4)=D(1{,}5)=\infty$ using $d(1,2)=2, d(1,3)=3, d(1,4)=1, d(1,5)=1$. The result $D(1,4)=D(1,5)=\infty$, is because of the global constraint.

In the case of $i=2$ and $j=1,2,3,4,5$, the equation

$$D(2,j)=\min\begin{cases}D(1,j)+d(2,j)\\ D(1,j-1)+2d(2,j)\\ D(2,j-1)+d(2,j)\end{cases}$$

determines the values of the second column $(i=2)$, namely, $D(2{,}1)=1+0, D(2{,}2)=1{+}1=2, D(2{,}3)=2{+}2=4, D(2{,}4)=4+0=4, D(2{,}5)=\infty$ using $d(2,1)=0, d(2,2)=1, d\times(2,3)=2, d\ (2,4)=d(2,5)=0$. The result $D(2,5)=\infty$ is because of the global constraint.

We continue this computation until $i=7$. Then for the case $j=1,2,3,4,5$, the equation

$$D(7,j)=\min\begin{cases}D(6,j)+d(7,j)\\ D(6,j-1)+2d(7,j)\\ D(7,j-1)+d(7,j)\end{cases}$$

determines the values of the seventh column $(i=7)$, namely, $D(7,1)=\infty, D(7,2)=\infty, D(7,3)=6, D(7,4)=4$, $D(7,5)=4$ using $d(7,1)=0, d(7,2)=1, d(7,3)=2, d(7{,}4)=d(7,5)=0$. The result $D(7,1)=D(7,2)=\infty$ is because of the global constraint.

The final results of the time alignment are the minimum of $J=D(7,5)/(7+5)=4/12=1/3$, and the alignment trajectory indicates a chain of arrows in the figure between two sequences obtained by tracing back from location $(7,5)$ to $(1,1)$. Moreover, the value of K becomes 9. The tracing back from the terminal location $(7,5)$ selects the local optimal path for $D(7,5)$ in the equation

$$D(7,5)=\min\begin{cases}D(6,5)+d(7,5)\\ D(6,4)+2d(7,5)\\ D(7,4)+d(7,5),\end{cases}$$

where $d(7,5) = 0$ and $D(6,5) + d(7,5) = 6, D(6,4) + 2d(7,5) = 5, D(7,4) + d(7,5) = 4$. It is clear that the backtracing path from (7,5) reaches location (7,4). Then we continue tracing back from location (7,4) using the equation

$$D(7,4) = \min \begin{cases} D(6,4) + d(7,4) \\ D(6,3) + 2d(7,4) \\ D(7,3) + d(7,4) \end{cases}$$

where $d(7,4) = 0$ and $D(6,4) + d(7,4) = 5, D(6,3) + 2d(7,4) = 4, D(7,3) + d(7,4) = 6$. It is clear that the backtracing path recovers location (6,3). We continue the same neighborhood-bound computation until reaching location (1,1). Then the complete path of tracing back is obtained and also $K = 9$ is determined. The arrow sequence of Fig. 4 indicates alignment path from (1,1) to (7,5) using the reverse of complete backtrace.

CONTINUOUS DYNAMIC PROGRAMMING

The time-alignment algorithm would seem to be very useful in many application domains. However, the algorithm described in the previous section still has restricted properties when considered for application to other real-world time-alignment problems. Therefore, many extended algorithms have been developed. One of them is called continuous dynamic programming (CDP) (8). CDP extends the previously described alignment problem, with an input sequence of a fixed length, to accept an endless stream of an input time sequence while the reference sequence is still a fixed interval of a time sequence. In other words, CDP performs segmentation-free matching between an endless input sequence and a reference sequence with fixed length without a priori segmentation of the input stream. The automaton model of CDP is simply described by taking a set of states, $\{(i,j) : -\infty < i < +\infty, 1 \leq j \leq T\}$, while the alignment problem outlined in the previous section takes a rectangular array $\{(i,j) : 1 \leq i \leq N, 1 \leq j \leq M\}$ as the set of states. Then the iterative algorithm of CDP is described using the parameters t,τ instead of i, j so that input and reference time sequences are respectively denoted $f(t), -\infty < t < +\infty$, and $Z(\tau), 1 \leq \tau \leq T$. We use the type of local constraint with weights shown in Fig. 5 and the local distance function $d(t,\tau) = d(f(t), Z(\tau))$, but no global constraint is imposed on the states. However, at each time t the local constraint creates a moving triangular area, as shown in Fig. 5. This area is made from possible combinations of local constraints so that the optimal path is included. The moving triangular area behaves similarly to a global constraint by restricting the admissible area of the (t,τ)-plane at time t. Moreover, the local constraint in Fig. 5 guarantees the property of monotonic increase in the alignment trajectory constructed by possible combinations of local paths to reach (t,T).

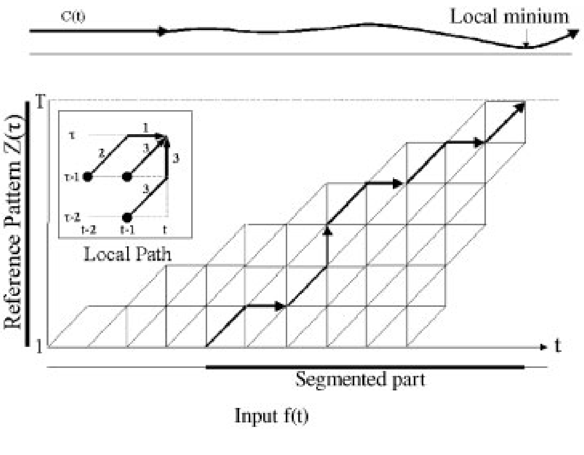


Figure 5. Continuous dynamic programming.

An accumulated minimum distance $P(t,\tau)$ is introduced and assigned to location (t,τ). Then the iterative computation of CDP is expressed by

$$P(t,1) = 3d(t,1),$$

$$P(t,2) = \min \begin{cases} P(t-2,1) + 2d(t-1,2) + d(t,2) \\ P(t-1,1) + 3d(t,2) \\ P(t,1) + 3d(t,2) \end{cases}$$

$$P(t,\tau) = \min \begin{cases} P(t-2,\tau-1) + 2d(t-1,\tau) + d(t,\tau) \\ P(t-1,\tau-1) + 3d(t,\tau) \\ P(t-1,\tau-2) + 3d(t,\tau-1) + 3d(t,\tau) \end{cases}$$

$$(3 \leq \tau \leq T)$$

where the initial condition is given by

$$P(-1,\tau) = P(0,\tau) = \infty \quad (1 \leq \tau \leq T)$$

The normalized minimum accumulated distance at time t is determined by

$$C(t) = \frac{P(t,T)}{3T}$$

where the total sum of weights is a constant value $3T$ for any optimal path because the weights in the local constraint take city block distances, as shown in Fig. 5. A simple comparison with the relationship between J and $D(i,j)$ in the previous section reveals that the output of CDP at time t can be expressed by

$$C(t) = \frac{1}{3T} \min_{(t_1,\tau_1) \leq (t_2,\tau_2) \leq \cdots \leq (t_K.\tau_K)} \left\{ \sum_{k=1}^{K} w_k d(t_k,\tau_k) \right\}$$

where causal monotonicity $(t_k,\tau_k) \leq (t_{k+1},\tau_{k+1})$ means that both $t_k \leq t_{k+1}$ and $\tau_k \leq \tau_{k+1}$ hold.

Segmentation-Free Recognition

The variable $C(t)$ is regarded as the output stream of CDP that is used to *frame-wisely* recognize a category represented by a reference pattern $\{Z(\tau) : \tau = 1, 2, \ldots, T\}$ and also detect an interval in an endless input stream at time t^* when $C(t)$ takes a local minimum at t^* under threshold value h. Here, the detection is dependent only on the local minimum of $C(t)$. At the same time, by tracing back the trajectory starting at location (t^*, T), we can find the location $(t_s^*, 1)$ on the two-dimensional array $\{(t, \tau) : -\infty < t < +\infty, 1 \leq \tau \leq T\}$. Then it is possible to segment the interval $[t_s^*, t^*]$ in the input stream that is optimally aligned to the reference time interval. To carry out the backtracing processing, it is necessary to recall the optimal local paths to reach location (t^*, T) from location $(t_s^*, 1)$, where $(t_s^*, 1)$ is determined as the end state of the backtracing. The detection of the interval $[t_s^*, t^*]$ is a segmentation that needs no assumption beforehand for segmentation.Therefore, this processing performs segmentation-free or spotting recognition of the reference time sequence that represents a category, such as a word. In other words, CDP simultaneously performs both segmentation and recognition by detecting a local minimum of the output stream of CDP.

We must now describe an important factor regarding CDP in which the segmentation-free recognition is based not on a global but a local minimum of the output stream of CDP. The detection of the local minimum is easily accomplished. From the viewpoint of real-time processing, this scope is suitable for its realization, because it means simply monitoring the stream of $C(t)$. The detection works well to perform segmentation-free recognition or retrieval of intervals frame-wisely from a database of sequences that are similar to a query sequence of a fixed length (8).

OTHER APPLICATIONS

Modifications of Local Distance and Constraints

The time-alignment algorithm described can be modified to solve a problem of string matching by using another type of local distance measure that deals with insertion, erasure, and confusion (replacement) between symbols in two strings.

Global and local constraints of time-alignment problems mentioned above are relatively simple, but generally, local and global constraints in many applications take different types. In creating DP algorithms based on more complex or different constraints, constraints are usually transformed into those of the baseline algorithm. One method to transform a given problem into the problem solved by the baseline algorithm is to transform a global constraint into a local constraint. For instance, consider the case that a global constraint of the automaton model of Fig. 1 imposes the relationship $q_1 + q_2 + \cdots + q_k = C$, ($C$: constant value), assuming that each decision q_k takes a value, and a local constraint is taken such as $q_k \geq 0$ for $k = 1, 2, \cdots, K$. We introduce a variable P_k with the relationship $q_k = p_k - p_{k-1}$. Then the global constraint changes to a part of the local constraint, namely $p_K = C$, and the rest of the local constraint is composed of $p_1 \geq 0$, $p_k \geq p_{k-1}$ for $k = 2, 3 \cdots, K$ and $S_k = T_k(S_{k-1}, p_k)$ while gain g_k becomes a function of $(p_k \geq p_{k-1})$and S_{k-1}. In other words,we have the expression $g_1(p_1), g_2(p_1, p_2), g_3(p_2, p_3), \ldots, g_K(p_{K-1}, p_k)$, so the DP algorithm to solve the transformed problem is formally equivalent to the baseline algorithm of DP. The key factor for understanding the procedural aspects mentioned in baseline DP algorithm is also valid.

Markov Decision Process

DP is applicable to a stochastic decision model. A simple algorithm for Markov decision process has been described (9). Let us define N states, $S_1, S_2, S_3, \ldots, S_i, \ldots, S_N$, and M decisions, $d_1, d_2, d_3, \ldots, d_m, \ldots, d_M$. The set of decisions is denoted by D and any decision is applicable to each S_i. A decision $d_m \in D$ for state S_j determines both transition probability $r_{ij}(m)$ from state S_i to S_j and gain $g_{ij}(m)$ of state S_j. Then each kth step of the optimal decision process with total gain $G_i^*(k)$ starting state from S_i satisfies the recursive equation of DP:

$$G_i^*(k+1) = \max_{d_m \in D} \left\{ \sum_{j=1}^{N} r_{ij}(m)(g_{ij}(m) + G_j^*(k)) \right\}$$

Building Block

Alternatively, DP can be embedded into another algorithm. For instance, a Hidden Markov Model uses a DP kernel as its important search engine called the Viterbi algorithm. Moreover we find the usage of DP in the following problems and algorithms: the Cocke-Younger-Kasami algorithm, the Earley algorithm, the knapsack problem, computer chess, games, traveling salesman problem, optimizing the order of chain matrix multiplication, stereo image matching, and image registration. Therefore, we often use DP as a building block to construct larger algorithms for specific purposes.

BIBLIOGRAPHY

1. R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.
2. R. E. Bellman, *Applied Dynamic Programming*. Princeton, NJ: Princeton University Press, 1962.
3. Mathematical Society of Japan, KiyosiIt Itô (ed.), *Encyclopedic Dictionary of Mathematics*, 2nd ed., Cambridge, MA: MIT Press, 1993.
4. A. Yamamoto, R. G. Haight, and J. D. Brodie, A comparison of the pattern search algorithm and modified path algorithm for optimizing an individual tree model, *Forest Sci.*, **36** (2): 394–412, 1990.
5. B. V. Roy, Neuro-dynamic programming: overview and recent trends, in E. Feinberg and A. Schwartz, (Eds.) *Handbook of Markov Decision Process: Method and Application*, Amsterdam, The Netherlands: Kluwer, 2001.
6. H. Sakoe and S. Chiba, Dynamic programming algorithm optimization for spoken word recognition, *IEEE Trans. Acoustic Speech Signal Proc.*, **26** (2), 43–49, 1978.

7. R. P. William, Protein sequence comparison and protein evolution, Tutorial- *ISBM2000*, UC San Diego, CA, 2000.

8. R. Oka, Spotting method for classification of real world data, *Comput. J.*, **42** (**8**): 559–565, 1998.

9. R. A. Howard, *Dynamic Programming and Markov Process*, New York: Technology Press/John Wiley, 1960.

FURTHER READING

M. Sniedovich, *Dynamic Programming*. Boca Raton, FL: CRC Press, 1992.

S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 2nd ed. New York: Academic Press, 2003. Dynamic programming. Available: http://en.wikipedia.org/w/index.php?title=Dynamic_programming & oldid=157608766.

Ryuichi Oka
Aizu-Wakamatsu
University of Aizu
Aizu-Wakamatsu,
Japan

F

FORMAL LOGIC

INTRODUCTION

Logic studies the validity of arguments. A typical case in point is that of syllogisms: logical arguments in which, starting from two premises, a conclusion is reached. For example, given that

> There are horses in Spain.
> All horses are mammals.

it can be inferred that

> There are mammals in Spain.

Of course, if instead of the second premise we had the weaker one

> Some horses are mammals.

where the universal (*all*) has been replaced with an existential (*some/exists*), then the argument would not be valid. In Ancient Greece, Aristotle exhaustively considered all possible combinations of universals and existentials in syllogisms, allowing also for the possibility of negations, and collected those corresponding to valid inferences in a classification theorem. For many centuries, that classification (slightly enhanced by the scholastics during the Middle Ages) was all there was to know about logic.

In its origin, the term "formal" logic used to be a reference to the form of the arguments. The validity of an argument depends exclusively on the *form* of the premises and on the conclusion, not on whether these are true or false. In the previous example, if we were to replace "horses" with "unicorns," the argument would still be valid, regardless of the fact that unicorns do no exist.

Nowadays, however, "formal" refers to the use of the formal and rigorous methods of mathematics in the study of logic that began to be put into practice in the second half of the nineteenth century with George Boole and, especially, Gottlob Frege (see Ref. 1). This trend started with a shift to a symbolic notation and artificial languages, and gradually it evolved until, in 1933 with Tarski (2), it culminated with the withdrawal from an absolute notion of Truth and instead focused on the particular truths of concrete structures or models.

Propositional logic

Arguably, the simplest logic is *propositional logic*, and we will use it to introduce the underlying elements of every logic. Given a set $A = \{p, q, r, \ldots\}$ of *atomic propositions,* the language of propositional logic is constructed according to the following rules:

$$\varphi ::= p \in A \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \to \varphi \mid \varphi \leftrightarrow \varphi$$

$\neg$ means 'No' $\to$ means 'Implies/Then'
$\vee$ means 'Or' $\leftrightarrow$ means 'If and only if'
$\wedge$ means 'And'

For example, let us assume that p represents "The chef is competent," q represents "The ingredients are expired," and r represents "The cake is delicious." Then, the premise "If the chef is competent and the ingredients are not expired, then the cake will be delicious" could be represented in the language of propositional logic as follows:

$$(p \wedge \neg q) \to r$$

Furthermore, if we assume that the chef is actually competent, that is, if we assume p as the second premise, we can conclude that "If the cake is not delicious, the ingredients are expired", or, formally:

$$\neg r \to q$$

But how and why can we conclude that this last sentence follows from the previous two premises? Or, more generally, how can we determine whether a formula φ is a valid consequence of a set of formulas $\{\varphi_1, \ldots, \varphi_n\}$? Modern logic offers two possible ways, which used to be fused in the time of syllogisms: the model-theoretic approach and the proof-theoretic one.

In model theory, it is necessary to assign a meaning to the formulas, to define a *semantics* for the language. The central notion is that of *truth* and of deciding the circumstances under which a formula is true. The more complex the logic, the more difficult this assignment is and hence the more complex the semantics. In propositional logic, we have to start by assigning arbitrary values to the atomic propositions: A *valuation* V is defined as a function that maps atomic propositions to either 0 (meaning, intuitively, false) or 1 (true). The meaning $\mathcal{I}^V(\varphi)$ of an arbitrary formula φ is defined recursively:

$$\begin{array}{lcl}
\mathcal{I}^V(p) & = & V(p) \\
\mathcal{I}^V(\neg\varphi) & = & \begin{cases} 1 & \text{if } \mathcal{I}^V(\varphi) = 0 \\ 0 & \text{if } \mathcal{I}^V(\varphi) = 1 \end{cases} \\
\mathcal{I}^V(\varphi \vee \psi) & = & \begin{cases} 1 & \text{if } \mathcal{I}^V(\varphi) = 1 \quad \text{or} \quad \mathcal{I}^V(\psi) = 1 \\ 0 & \text{otherwise} \end{cases} \\
\mathcal{I}^V(\varphi \wedge \psi) & = & \begin{cases} 1 & \text{if } \mathcal{I}^V(\varphi) = 1 \quad \text{and} \quad \mathcal{I}^V(\psi) = 1 \\ 0 & \text{otherwise} \end{cases} \\
\mathcal{I}^V(\varphi \to \psi) & = & \begin{cases} 1 & \text{if } \mathcal{I}^V(\varphi) = 0 \quad \text{or} \quad \mathcal{I}^V(\psi) = 1 \\ 0 & \text{otherwise} \end{cases} \\
\mathcal{I}^V(\varphi \leftrightarrow \psi) & = & \begin{cases} 1 & \text{if } \mathcal{I}^V(\varphi) = \mathcal{I}^V(\psi) \\ 0 & \text{otherwise} \end{cases}
\end{array}$$

For example, if $V(p) = V(q) = 0$ and $V(r) = 1$, then $\mathcal{I}^V(\neg p) = 1$, $\mathcal{I}^V(\neg p \wedge q) = 0$, and $\mathcal{I}^V(r \to (\neg p \wedge q)) = 0$.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

If $\mathcal{I}^V(\varphi) = 1$, then it is said that V is a *model* of φ, or that V *satisfies* φ; it is a "world" in which φ is true. A formula is said to be *valid* if it is true under all circumstances, that is, if every valuation is a model of φ:

$$\varphi \text{ is valid if } \mathcal{I}^V(\varphi) = 1 \text{ for all valuations } V$$

For instance, it is easy to check that $p \rightarrow (q \rightarrow p)$ is a valid formula. Similarly, if V is a model of all the formulas in a set Γ, then V is said to be a model of Γ. A formula φ is a *semantic consequence* of a set Γ of formulas, written $\Gamma \models \varphi$, if every model of Γ is also a model of φ or, alternatively, if φ is true whenever all formulas in Γ are true:

$$\Gamma \models \varphi \text{ if } \mathcal{I}^V(\varphi) = 1 \text{ whenever } \mathcal{I}^V(\psi) = 1 \text{ for all } \psi \in \Gamma.$$

In the proof-theoretic approach, the central concept is that of *proof*: to show that a statement follows from some others, one has to make use of a *deduction system*. Deduction systems are syntactic in nature, caring only about the form of sentences and not about what they represent or their possible meaning. One such system, the *axiomatic method*, distinguishes a subset of formulas, called *axioms*, formed by all sentences that match any of the following patterns:

$$\begin{array}{l} \varphi \rightarrow (\psi \rightarrow \varphi) \\ (\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi)) \\ (\neg \varphi \rightarrow \neg \psi) \rightarrow (\psi \rightarrow \varphi) \end{array}$$

Given a set of formulas Γ, a formula φ is said to be a *logical consequence* of Γ, written $\Gamma \vdash \varphi$, if there is a sequence of formulas $\varphi_1, \varphi_2, \ldots, \varphi_n$ such that:

1. $\varphi_n = \varphi$.
2. For all $i \leq n$, either φ_i is an axiom, or φ_i belongs to Γ, or $j, k < i$ exist such that $\varphi_k = \varphi_j \rightarrow \varphi_i$.

This corresponds to an iterative process in which we can add to the set of provable sentences, at every stage, either an axiom (whose validity is clear according to the defined semantics), an element of Γ (a hypothesis we are assuming as given), or a formula φ_i whenever we have previously proved φ_j and that φ_j implies φ_i.

The axiomatic method is not the only deduction system. In *natural deduction*, rules are associated with the connectives: introduction rules, to prove formulas containing the connective, and elimination rules, to obtain consequences from a formula with a given connective. For example, the following are prototypical:

$$\frac{\varphi \;\; \psi}{\varphi \wedge \psi} \quad \frac{\varphi \wedge \psi}{\varphi} \quad \frac{\varphi \wedge \psi}{\psi} \quad \frac{}{\varphi} \;\; \text{if } \varphi \in \Gamma$$

The first rule is the introduction rule for the logical "and," and it captures the idea that if φ and ψ can both be proved, so can their conjunction; the next two are elimination rules, stating that from a conjunction, both its conjuncts can be derived; the last allows the derivation of a hypothesis. Similar rules are associated with the remaining connectives. Compared with the axiomatic method, natural deduction has a richer set of rules: As a result, it is easier to prove things *in* the logic using natural deduction, but the simplicity of the axiomatic method comes in handy if one is interested in proving something *about* the logic. Although the presentations of deduction systems vary, they all allow the derivation of the same set of sentences (assuming they are properly designed).

Let us return to the example of the chef and the cake. In symbols, the argument can now be expressed as follows:

$$\{(p \wedge \neg q) \rightarrow r, p\} \models \neg r \rightarrow q$$

Although it is a bit tedious, one can consider all eight possible assignments of values to p, q, and r and check that it is actually a semantic consequence.

But then the following question can be raised. How come the argument cannot be expressed instead as follows:

$$\{(p \wedge \neg q) \rightarrow r, p\} \vdash \neg r \rightarrow q$$

Indeed, we have defined two different notions of consequence, semantic and logical, and although both are reasonable, they do not seem to have much in common: Which one should be chosen? This is an important *metalogical* question. Fortunately, in the case of propositional logic, it does not matter for it can be proved that

$$\Gamma \vdash \varphi \quad \text{if and only if} \quad \Gamma \models \varphi$$

The implication from left to right, that asserts that any proof-theoretic logical consequence is also a semantic consequence, is known as the *soundness of propositional logic*. The implication from right to left, which claims that any semantic consequence has a syntactic derivation, is the *completeness of propositional logic*.

Assume that we have a finite set Γ of assumptions. We then have two methods at our disposal to decide whether a given formula follows from Γ. Either we build a syntactic proof, or we show that all models of the assumptions also satisfy the formula. In particular, since we are working with a finite set of formulas, there is only a finite number of atomic propositions involved. We can consider all possible valuations and study whether there is one that satifies Γ but not φ. Hence, for propositional logic, the validity problem is *decidable*, in the precise sense that there is an effective procedure or algorithm that solves it (see the article on Computability).

This ends the presentation of propositional logic. Summing up, the most important elements introduced, common to all logics are as follows: a syntax that defines the language; a semantics to assign meaning to the formulas; logical and semantic consequences and relationships between them; and the validity problem.

PREDICATE LOGIC

The simplicity of propositional logic comes at a price. Its expressive power is limited; in particular, it cannot deal with syllogisms like that at the beginning of this article:

> There are horses in Spain.
> All horses are mammals.

which implies that

> There are mammals in Spain.

In propositional logic we would have to formalize the first premise by means of an atomic proposition p, the second with q, and the conclusion r would not be a valid consequence.

To remedy this issue, predicate logic (also known as first-order logic) introduces *predicates* and *quantifiers*. Assume that we use $H(x)$, $S(x)$, and $M(x)$ to express that x is a horse, x dwells in Spain, and x is a mammal, respectively. Then, the syllogism can be presented as a valid argument in predicate logic as follows:

$$\begin{array}{c} \exists x(H(x)\wedge S(x)) \\ \forall x(H(x)\rightarrow M(x)) \\ \hline \exists x(M(x)\wedge S(x)) \end{array}$$

where the quantifier $\forall$ means "for all" and $\exists$ means "there exists."

But predicate logic goes beyond syllogisms. It not only allows multiple premises, but also predicates with an arbitrary number of arguments. For example, a statement like "the ancestor of an ancestor is also an ancestor" has no room in the syllogistic theory, whereas it can be dealt with in predicate logic by using a binary predicate $A(x, y)$ with the meaning x is an ancestor of y:

$$A(x,y)\wedge A(y,z)\rightarrow A(x,z)$$

In predicate logic, one can distinguish two levels: terms and formulas. Terms denote individuals, whereas formulas are statements about those individuals. Terms are constructed from a set of variables $(x_0, x_1, x_2, \ldots)$ and a set of constants and function symbols with arbitrary arities:

$$t ::= x \mid c \mid f(t_1,\ldots,t_n) \qquad f \text{ function symbol of arity } n$$

Thus, if *mother* is a unary function symbol, the fact that one's mother is an ancestor can be expressed with $A(\mathit{mother}(x), x)$.

Formulas, in turn, are constructed from terms and a set of predicate symbols:

$$\begin{aligned} \varphi ::= {} & t_1 = t_2 \\ & \mid R(t_1,\ldots,t_n) \mid \neg\varphi \mid \varphi\vee\varphi \mid \varphi\wedge\varphi \mid \varphi\rightarrow\varphi \mid \varphi\leftrightarrow\varphi \mid \forall x\varphi \mid \exists x\varphi \end{aligned}$$

where R is a predicate symbol of arity n. The resulting set of formulas depends on the concrete sets of function and predicate symbols, F and P. We will write $L(F, P)$ to denote the set of formulas, or language, built using the sets F and P, or just L if no ambiguity develops. For example, if *peter* is a constant that represents a concrete boy and S is a binary predicate that stands for "sibling," the sentence

$$\forall x(S(x,\mathit{peter})\rightarrow \forall z(A(z,\mathit{peter})\leftrightarrow A(z,x)))$$

expresses that *peter* has the same ancestors as his siblings.

For another, less-contrived example, consider the function symbols 0 (constant), *suc* (unary), and $+$, $*$(binary), and the predicate $<$ (binary), which give rise to the language of arithmetic for obvious reasons. Commutativity of addition is then expressed as follows:

$$\forall x\,\forall y + (x,y) = +(y,x)$$

Although awkward, this is the correct notation; however, we will usually stick to the standard infix notation and write $+(x, y)$ as $x + y$.

Note that quantifiers are variable binding operators in the same sense as the summation symbol $\sum$ in an expression like $\sum_{x=1}^{9} x$, where the variable x cannot "vary" and take any arbitrary value. (This will become more clear once the semantics of quantifiers is presented in the next section.) In $\forall x\varphi$ and $\exists x\varphi$, the formula φ is said to be the scope of the quantifier. Then, an occurrence of a variable in an arbitrary formula is said to be *free* if it falls under the scope of no quantifier; otherwise, that occurrence is called *bound*.

Semantics

First of all, the universe of discourse, the elements to be referred to by terms, has to be fixed; then, function and predicate symbols from the sets F and P have to be interpreted over it. More precisely, a structure $\mathcal{A}$ is a tuple

$$\langle A, c^A, \ldots, f^A, \ldots, R^A \ldots\rangle$$

such that

- A is a nonempty set;
- for every constant $c\in F, c^A \in A$;
- for every n-ary function symbol $f\in F$, f^A is a function from A^n to A;
- for every n-ary predicate symbol $R\in P, R^A$ is a subset of A^n.

An obvious structure $\mathcal{N}$ for the language of arithmetic consists of the set $\mathcal{N}$ of natural numbers as universe, with 0^N the number zero, suc^N the successor operation, $+^N$ and $*^N$ addition and multiplication of natural numbers, and $<^N$ the "less-than" relation. Note, however, that the structure can be arbitrary. Another valid structure is $\mathcal{M}$, where:

- the universe M is the set of digits $0, 1, 2, \ldots, 9$;
- 0^M is the digit 9;
- suc^M returns 0 when applied to 9 and the following digit in the usual order otherwise;
- $+^M$, when applied to two digits, returns the smallest one;
- $*^M$ returns the greatest digit;
- $<^M$ is the set of all pairs (u, v) with u greater than v.

More generally, the set A can consist of letters, derivable functions, matrices, or whatever elements one chooses.

Before meaning can be ascribed to terms, yet another component is needed: an *assignment* mapping variables to elements of A. Then, an *interpretation* $\mathcal{I} = (\mathcal{A}, V)$ is a pair formed by a structure and an assignment V. The meaning of a term in a given interpretation, that is, the individual it refers to in the universe, can now be defined recursively:

$$\mathcal{I}(x) = V(x)$$
$$\mathcal{I}(c) = c^A$$
$$\mathcal{I}(f(t_1,\ldots,t_n)) = f^A(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n))$$

Let us consider the previously defined structure $\mathcal{N}$, and let V be an assignment such that $V(x) = 3$ and $V(y) = 5$. In the interpretation $\mathcal{I} = (\mathcal{N}, V), \mathcal{I}(x * suc(suc(0))) = 6$ and $\mathcal{I}(x + suc(y)) = 9$. On the other hand, in the interpretation $\mathcal{J} = (\mathcal{M}, W)$, where $W(x) = 3$ and $W(y) = 5$, those same two terms get very different meanings: $\mathcal{J}(x*suc(suc(0))) = 3$ and $\mathcal{J}(x + suc(y)) = 3$.

A last piece of machinery is needed. Given an assignment V, a variable x, and an element a of the universe, we write $V[a/x]$ for the assignment that maps x to a and coincides with V in the remaining variables. The truth value of a formula, 0 (false) or 1 (true), with respect to an interpretation $\mathcal{I}$ can finally be defined:

- $\mathcal{I}(t_1 = t_2) = 1$ if $\mathcal{I}(t_1) = \mathcal{I}(t_2)$, and 0 otherwise;
- $\mathcal{I}(R(t_1,\ldots,t_n)) = 1$ if $(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n)) \in R^A$;
- $\mathcal{I}(\neg\varphi), \mathcal{I}(\varphi \wedge \psi), \mathcal{I}(\varphi \vee \psi), \mathcal{I}(\varphi \to \psi), \mathcal{I}(\varphi \leftrightarrow \psi)$ are defined analogously to the propositional case ;
- $\mathcal{I}(\forall x\varphi) = 1$ if $\mathcal{J}(\varphi) = 1$ for all $a \in A$, where $\mathcal{J} = (\mathcal{A}, V[a/x])$;
- $\mathcal{I}(\exists x\varphi) = 1$ if $a \in A$ exists with $\mathcal{J}(\varphi) = 1$, where $\mathcal{J} = (\mathcal{A}, V[a/x])$.

As in propositional logic, if $\mathcal{I}(\varphi)$ is 1, we say that $\mathcal{I}$ is a model of φ or that $\mathcal{I}$ satisfies φ, and we denote it by $\mathcal{I} \models \varphi$. We write $\Gamma \models \varphi$ if every model of all formulas in Γ is also a model of φ.

Note that assignments are only needed to ascribe a meaning to free occurrences of variables. If there are none, the interpretation of a formula is the same regardless of the assignment. Now it can be checked that the formulas

$$\forall x(0 < suc(x)) \quad \text{and} \quad \forall x \exists y(x < y)$$

are both true in the interpretation $\mathcal{N}$ but false in $\mathcal{M}$.

Proof Theory

Deduction systems for predicate logic extend those of propositional logic to take care of predicates and quantifiers. In the axiomatic method, the main difference originates from the extension of the set of axioms.

A valuation in predicate logic is a function from the set of formulas to the set $\{0, 1\}$ that respects the meaning of propositional connectives; that is, $f(\neg\varphi) = 1 - f(\varphi)$, $f(\varphi \wedge \psi) = 1$ if and only if $f(\varphi) = f(\psi) = 1, \ldots$. The set of axioms is then the set of formulas with one of the following forms:

1. Formulas that are mapped to 1 by all valuations.
2. $\forall x(\varphi \to \psi) \to (\forall x\varphi \to \forall x\psi)$.
3. $\varphi \to \forall x\varphi$, with no free occurrences of x in φ.
4. $\exists x(x = t)$ where t is a term that does not contain x.
5. $t_1 = t_2 \to (\varphi \to \psi)$ where φ contains no quantifiers and ψ is obtained from φ by replacing an occurrence of t_1 in φ with t_2.

A derivation of φ from Γ is a sequence $\varphi_1,\ldots,\varphi_n$ such that:

1. $\varphi_n = \varphi$.
2. For all $i \leq n$, either:
   a. φ_i is an axiom;
   b. φ_i belongs to Γ;
   c. j, $k < i$ exist such that $\varphi_k = \varphi_j \to \varphi_i$;
   d. $j < i$ exists and a variable x such that φ_i is $\forall x\varphi_j$.

The formula φ is then a logical consequence of Γ, and it is denoted with $\Gamma \vdash \varphi$.

Likewise, a system of natural deduction for predicate logic is obtained, essentially, by extending the propositional one with the rules

$$\frac{\varphi}{\forall x\varphi} \quad \frac{\forall x\varphi}{\varphi[t/x]}$$

subject to a couple of technical conditions that are omitted and where $\varphi[t/x]$ means that every free occurrence of x in φ is replaced by t.

Completeness and Decidability

Deduction systems are designed so that they are sound, that is, so that a logical consequence is also a semantic consequence, and it is not hard to show that this is the case in predicate logic. The converse implication, completeness, is much harder to prove, but it was also shown to hold for predicate logic by Gödel (3) in 1929. Therefore, for a set of formulas Γ and a formula φ,

$$\Gamma \vdash \varphi \quad \text{if and only if} \quad \Gamma \models \varphi$$

Faced with the problem of deciding whether a given formula is a consequence of a set of premises, we again have the same two alternatives as in propositional logic: either build a derivation or consider the models involved. Now, however, there is a crucial difference. The set of models is not finite, and thus, in general, it will not be possible from their study to conclude whether a formula is a semantic consequence of the premises. Hence, if we intend to obtain a mechanical procedure, an algorithm, to decide the validity of a formula, we are only left with the proof-theoretic approach.

Assume that we want to determine whether a formula φ is valid, that is, whether $\vdash \varphi$ (it can be derived from no hypothesis). By using the axiomatic method, we can enumerate all valid formulas. First, all derivations φ_1 that use axioms of length up to, say 10, are listed; since there can only be a finite number of these, the process ends. Next, derivations φ_1 and φ_1, φ_2 with axioms of length up to 11 are considered; again, there is only a finite number of such derivations. In the next step, derivations of up to three steps with axioms of length less than or equal to 12 are listed; and so on. The process is tedious, but it is mechanizable and considers all posible derivations. If φ is valid, then it has a corresponding derivation and this procedure will eventually produce it. But what if φ is not valid? Then the procedure will not terminate and will offer no clue as to the validity of φ. Indeed, that it is no accident but an unavoidable shortcoming of any procedure to decide the validity of a formula is the content of the *undecidability theorem* proved independently by Church and Turing (4, 5) in 1936.

Note that this does not mean that it is not possible to decide whether a given formula is valid, but that it is not possible to develop a general procedure that always works. For example, if all predicates considered are monadic (take one argument), the resulting language is decidable, and in this case, a computer could be programmed to solve the validity problem.

A GLIMPSE OF OTHER LOGICS

Second-Order and Higher Order Logic

In predicate logic, variables range over the elements of the universe in the structure; this is why it is also called first-order logic. But in mathematics it is often necessary to refer, not to single individuals, but to collections of these. As a result, it is sometimes convenient to consider an extension of predicate logic with second-order variables that range over subsets or, more generally, over n-ary relations of the universe.

The syntax and semantics of second-order logic are defined similarly to those of predicate logic. Now, if X is an n-ary variable and $t_1, \ldots, t_n$ are terms, $X(t_1, \ldots, t_n)$ is also a formula. Second-order logic is more expressive than predicate logic. For example, the structure of the natural numbers cannot be characterized by means of predicate formulas because the induction principle can only be approximated by means of all formulas of the form

$$\varphi(0) \wedge \forall x(\varphi(x) \rightarrow \varphi(suc(x))) \rightarrow \forall x \varphi$$

In second-order logic, however, the induction principle is formally captured by the single formula

$$\forall X(X(0) \wedge \forall x(X(x) \rightarrow X(suc(x))) \rightarrow \forall x X(x))$$

where X is a unary variable, and the structure of natural numbers is characterizable.

Second-order logic allows the expression of mathematical facts in a more natural way; however, this additional expressive power makes the logic much more complex, with many useful properties of predicate logic no longer holding. In particular, there is no deduction system both sound and complete; of course, this is no obstacle for setting up correct and useful (although incomplete) systems. Also, the validity problem is even more undecidable (in a precise technical sense, see the Computability article) than in the first-order case.

Although second-order logic allows for quantifying over predicates, higher order logic (historically, proposed a couple of decades earlier than predicate logic) goes a step beyond and considers, and allows quantification over, predicates that take other predicates as arguments, and predicates that take predicates that take predicates, ... The resulting logic is, again, very complex but extremely expressive, and it has proved to be very useful in computer science.

Intuitionistic Logic

In traditional (also called classical) mathematics, nonconstructive arguments are valid proof methods. It is possible to show that there has to exist an element satisfying a certain property without actually producing a witness to it, and the statement that either a proposition or its negation holds is admitted as true even if none of them has been proved. Some mathematicians object against such principles when working with infinite sets and advocate the practice of constructive procedures. In particular:

- a statement of the form $\exists x \varphi$ is not proved until a concrete term t has been constructed such that $\varphi[t/x]$ is proved;
- a proof of a disjunction $\varphi \vee \psi$ is a pair $\langle a, b \rangle$ such that if $a = 0$ then b proves φ, and if $a \neq 0$ then b proves ψ.

Intuitionistic logic captures the valid principles of inference for constructive mathematics. In this new setting, familiar statements cease to hold. For example, $\varphi \vee \neg \varphi$ is not universally true, and, although the implication $(\neg\varphi \vee \neg\psi) \rightarrow \neg(\varphi \wedge \psi)$ can still be proved, it is not possible to strenghten it to a biconditional. Perhaps surprisingly, a deductive system for intuitionistic logic can be obtained from a classical one just by preventing the *law of double negation*

$$\neg \neg \varphi \rightarrow \varphi$$

or any other equivalent to it, from being used as an axiom.

In this sense, intuitionistic logic is a subset of classical logic. On the other hand, classical logic can also be embedded in intuitionistic logic. For every formula φ, a formula ψ can be constructed such that φ is derivable in the classical calculus if and only if ψ is derivable in the intuitionistic one.

Since the deduction system has been restricted, it is still sound with respect to the semantics for predicate logic, but it is no longer complete. Several semantics for intuitionistic logic have been defined, the simplest of which is probably the one introduced by Kripke. For propositional intuitionistic logic, a *Kripke structure* is a partially ordered set $\langle K, \leq \rangle$ together with an assignment V of atomic propositions to the elements of K such that, if $k \leq k', V(k) \subseteq V(k')$. One can think of the elements in K as stages in time and then $V(k)$ would be the "basic facts" known at instant k. Satisfaction of a formula by a model, called *forcing* and representing the knowledge at a certain stage, is defined recursively:

- k forces p if $p \in V(k)$.
- k forces $\neg \varphi$ if for no $k' \geq k$ does k' force φ.
- k forces $\varphi \wedge \psi$ if k forces φ and k forces ψ.

- k forces $\varphi \vee \psi$ if k forces φ or k forces ψ.
- k forces $\varphi \rightarrow \psi$ if, for every $k' \geq k$, if k' forces φ, then k' forces ψ.

The intuition for all clauses except the second and the fifth is clear. One knows $\varphi \rightarrow \psi$ at instant k, even if none of φ or ψ is yet known, if one knows that for all future instants one can establish ψ if φ can be established. As for the negation, $\neg\varphi$ is known when no evidence for φ can be found at a later stage.

A Kripke structure forces a formula if all its elements do so, and the intuitionistic calculus is sound and complete with respect to forcing. Kripke structures and forcing can be extended to predicate logic so that the corresponding deduction system also becomes sound and complete. Like their classical counterparts, intuitionistic propositional logic is decidable, whereas intuitionistic predicate logic is not.

Predicate Logic in Perspective

In mathematics, predicate logic has been a great success, to the point of being often deemed as *the* logic. It is in principle sufficient for mathematics, has a sound and complete deduction system, and satisfies important semantic results. Indeed Lindstrom's theorems show that there can be no logical system with more expressive power than predicate logic and with the same good semantic properties.

By contrast, computer science (and other fields) has seen a cornucopia of logics suited for different purposes. To cite a few:

- *Modal logic*. It is a logic to reason about concepts such as possibility or necessity.
- *Temporal logic*. It is a brand of modal logic with operators to talk about the passage of time.
- *Fuzzy logic*. To deal with approximate, or vague concepts such as the distinction between "warm" and "hot."
- *Probabilistic logic*. In which the truth values of formulas are probabilities.
- *Nonmonotonic logic*. In this logic, an established piece of knowledge may have to be retracted if additional facts are later known.

Moreover, these logics come in different flavors, usually admitting propositional, first-order, higher order, and intuitionistic presentations, as well as combinations of these and many ad hoc variants.

LOGIC AND COMPUTER SCIENCE

Although profound and important, the practical significance for mathematics of the results obtained during the blossoming of logic in the twentieth century has been limited. By contrast, logic has risen to prominence in computer science where it is expected to play a role analogous to that of calculus in physics. Specification and programming rank among its most important areas of application, which also include fields as diverse as descriptive complexity, compiler techniques, databases, or type theory (6).

Specification and Verification

Computer programs are extremely complex entities, and reasoning about them, except for the smallest instances, is no easy feat. A given computer program poses two related problems: deciding what it is supposed to do and then checking whether it does it. Logic can help here, first, with the formal specification of the expected behavior of a program and, second, in the process of verifying that the program indeed abides by its specification.

In its first use, logic can be seen as a tool to resolve ambiguities. Assume that a programmer is asked to write a program that, given two integers a and b, returns the quotient and remainder of dividing a by b. The behavior of the program when a and b are positive should be obvious, but if one of them is negative, say $a = 5$ and $b = -2$, and the mathematical training of the programmer is a bit shaky, he might be inclined to admit -3 and -1 as valid quotient and remainder. Furthermore, what should the behavior be when the divisor is 0? A bullet-proof, unambiguos specification of the behavior of the program would look like:

$$\begin{array}{l} \varphi \equiv a, b \text{ integers} \\ \textbf{fun } \mathit{division}(a,b) \textbf{ returns } \langle q, r \rangle \\ \psi \equiv (q, r \text{ integers}, a = b * q + r, 0 \leq r < |b|) \vee (b = 0 \wedge q = r = -1) \end{array}$$

In this specification, the *precondition* φ requires the arguments to be integers, whereas the *postcondition* ψ imposes that q and r are the appropriate quotient and remainder if $b \neq 0$, and sets them both to -1 if b is 0 to mark the error condition. Alternatively, one could assume/require that b is never instantiated with 0 as value:

$$\begin{array}{l} \varphi \equiv a, b \text{ integers}, \ b \neq 0 \\ \textbf{fun } \mathit{division}(a,b) \textbf{ returns } \langle q, r \rangle \\ \psi \equiv q, r \text{ integers}, a = b * q + r, 0 \leq r < |b| \end{array}$$

In this case, the postcondition ψ leaves unspecified the behavior of the program if b is 0, so that it should be used only at one's own risk.

In general, the precondition imposes some requirements on the input, whereas the postcondition states all properties that can be assumed about the output. Anything not reflected in them falls outside the programmer's responsibility. To express the precondition and postcondition, any logic can be used.

A specification imposes a contract on the programmer who, given that programming is an error-prone task, would like to have a means to verify that his final code actually satisfies those requirements (see the Formal Program Verification article). The prototypical example of the use of logic in this regard is *Hoare logic*, designed for imperative programming languages. It defines a derivation system consisting of a set of rules of the form

$$\frac{\text{Condition}}{\varphi \; \mathit{instruction} \; \psi}$$

for every instruction in the programming language, establishing the conditions under which a precondition φ and postcondition ψ hold. For example, to the assignment instruction it corresponds the rule

$$\frac{}{\varphi[e/x]\, x := e\, \varphi}$$

which states that some property holds for variable x after the instruction is executed only if it already held when the expression e was substituted for x. Similarly, for sequential composition of instructions, we have the rule

$$\frac{\varphi\, instruction_1\, \psi \quad \psi\, instruction_2\, \chi}{\varphi\, instruction_1\, ;\, instruction_2\, \chi}$$

which states the conditions required for χ to hold after executing $instruction_1$ followed by $instruction_2$. Ideally, to show that a program P satisfies a specification $\varphi P \psi$, one would start with ψ and the final line of P and proceed backward by applying the corresponding rule in the calculus. The fact that programs are hundreds of thousands of lines long and that the application of some of the rules require human intervention makes this direct approach unfeasible in practice.

A different flavor in which logic supports the verification process comes in the form of *proof assistants* or *theorem provers* (see the Automated Theorem Proving article). These are usually embedded within verification systems that allow for formally specifying functions and predicates, to state mathematical theorems and to develop formal proofs. Many of these systems are based on higher order logic, but some use predicate logic. These environments do not directly work with the program code but instead focus on the algorithms that implement, by translating them into an appropriate logical language and then formally proving the properties they are required to satisfy. These systems are very powerful, and some impressive results have been achieved in the verification of certain hardware architectures and protocols, but they present as a major drawback their dependency on user interaction.

In contrast to theorem provers, model checkers are fully automated and their underlying logic is some variation of temporal logic. *Model checking* was proposed in the early 1980s to verify complex hardware controllers and has also come to be used in the verification of software since then. A model checker explores all posible states in a system to check for a counterexample of the desired property and herein lies its limitation: Whereas hardware controllers have a finite, if huge, number of states, typical programs have an infinite number. Even for hardware systems, the number of states can grow exponentially giving rise to what is known as the *explosion problem*. Hence, the devise of abstraction techniques that significantly reduce the size of a system to make it amenable to model checking, without altering its "main" properties, is an active area of research.

Programming

Logic also serves as the foundation of programming languages and has given rise to the *declarative paradigm*.

Logic Programming. Imagine a problem for which all assumptions and requirements have been expressed as predicate logic formulas and gathered in a set Γ. For this problem, we are interested in determining whether a solution exists, an element that under the given requirements satisfies a certain condition. Formally, we are interested in the entailment

$$\Gamma \vdash \exists x\, \varphi(x)$$

Furthermore, we are probably interested not only in finding out if such an element exists but also in a concrete instance, that is, a term t such that $\Gamma \vdash \varphi[t/x]$.

In general, from $\Gamma \vdash \exists x\, \varphi$, it does not follow that $\varphi[t/x]$ for some term t; think of $\Gamma = \{\exists x\, R(x)\}$ for a counterexample. *Logic programming* is the area of research that studies the conditions that guarantee the existence of t and the ways of obtaining it (see the Answer Set Programming article). In logic programming, the formulas in Γ are restricted to *universal Horn formulas* of the forms

$$\forall x_1 \ldots \forall x_n \varphi \quad \text{and} \quad \forall x_1 \ldots \forall x_n (\varphi_1 \wedge \ldots \wedge \varphi_m \to \varphi)$$

whereas the goal is an existential formula

$$\exists x_1 \ldots \exists x_n (\varphi_1 \wedge \ldots \wedge \varphi_m)$$

and all φ_i and φ have the form $t = t'$ or $R(t_1, \ldots, t_n)$. Under these conditions, if the goal can be proved, then concrete terms $t_1, \ldots, t_n$ exist for which φ holds. In principle, these terms can be found by systematically applying the rules of any of the deduction systems presented earlier. However, given the restricted form of Horn formulas, a rule of inference more suitable for logic programming called *resolution* has been developed that computes all terms that make $\varphi_1 \wedge \ldots \wedge \varphi_m$ true. Recall that the validity problem in predicate logic is undecidable; this is reflected in logic programming in the fact that every implementation of resolution may loop forever if there are no solutions to the problem.

As an example of a logic program, let us consider the problem of finding paths in a directed graph. Assuming we use constants $a, b, \ldots, f$ to represent the nodes and corresponding binary predicates *arc* and *path*, the conditions of the problem can be represented as follows (omitting quantifiers, as customary in this context):

$$\begin{array}{l} arc(a,b) \\ arc(a,c) \\ arc(b,d) \\ arc(e,f) \\ arc(f,e) \\ arc(x,y) \to path(x,y) \\ arc(x,z) \wedge path(z,y) \to path(x,y) \end{array}$$

A path from x to y is specified either as a direct arc between the nodes or as an arc from x to an intermediate node z followed by a path from z to y. Now, to obtain all nodes reachable from a, the goal $path(a,x)$ would be used and the resolution procedure would return b, c, and d as possible values for x.

Imposing additional restrictions on the form of the formulas has led to the definition of query languages for deductive databases.

Functional Programming. *Functional programming* languages are based on the *lambda calculus*. Although there are many variants and they all favor an operational interpretation, lambda calculi can be endowed with both semantics and sound and complete deduction systems (but note that it took almost 40 years to define a semantics for the original, untyped lambda calculus), thus partaking in the main logical features. Unlike predicate logic, functions are first-class citizens in the lambda calculus and can therefore be passed as arguments to other functions and returned as results.

Functional languages come in many different flavors (see the Functional Programming article), but the central notion in all of them is that of definition $t \equiv s$, which is interpreted operationally as a rule that allows the transformation of the term in the left-hand side t into the one in the right-hand side s in a process called *rewriting*. A program consists simply of a set of definitions. For example, the program

$$\begin{aligned}
&\mathit{square} : \mathit{Integer} \to \mathit{Integer}\\
&\mathit{square}\ x \equiv x * x\\
&\mathit{apply} : (\mathit{Integer} \to \mathit{Integer}) \to \mathit{Integer}\\
&\mathit{apply}\ f\ x \equiv f\ x
\end{aligned}$$

defines a function *square* that returns the square of its argument, and a function *apply* that takes a function *f* and an integer as arguments and applies *f* to that integer. The term *apply square* 2 would then be rewritten first to *square* 2 and then to $2 * 2$ (which the compiler would evaluate to 4). For some terms, however, the process of reduction may never stop:

$$\begin{aligned}
&\mathit{infinity} : \mathit{Integer}\\
&\mathit{infinity} \equiv \mathit{infinity} + 1
\end{aligned}$$

Terms such as this do not denote well-defined values in the normal mathematical sense. Another potential difficulty lies in the possible existence of different rewriting sequences for the same term. Given definitions $t \equiv t'$ and $t \equiv t''$, in principle t could be reduced to either t' or t'', and there are no guarantees that these two terms can be further reduced to a common one. In functional programming, if two different rewriting sequences terminate, then they *converge*; that is, they lead to the same result.

Other Paradigms and Tidbits. Logic and functional programming are the two main representatives of declarative programming. Their main advantage is their logical foundation, which makes it possible to mathematically reason about the programs themselves and enormously facilitates the verification process. This has led many researchers to seek ways of enhancing their expressive power while remaining in the corresponding framework and thus retaining their good properties, and has produced languages in which convergence may not happen and nondeterminism is allowed or where (some) *a priori* infinite computations can be dealt with. Also, both approaches have been integrated in what is known as *functional logic programming;* here the computational process is guided not by resolution nor rewriting, but by a technique called *narrowing*.

Some languages have withdrawn from predicate logic and the lambda calculus and are based on *equational logic*, which is quite a restrictive subset of them both and precludes the use of functions. The loss in expressivity is made up with the gain in efficiency and simplicity of the mathematical reasoning about the programs. More recently, a whole family of languages has been designed based on yet another logic, *rewriting logic*, which extends equational logic with rules to allow a natural treatment of concurrency and nondeterminism. Unlike equations, a rule $t \to s$ does not express the identity between the meanings of the two terms but rather that the state represented by t evolves to that represented by s.

CODA

The previous sections may convey the impression that a logic is a precisely defined entity, when there is actually no consensus about what constitutes a logic. It is true that most logics have both a model and a proof-theory, but that is not always the case and, even when it is, one of the approaches may be clearly emphasized over the other. For Quine (7) on more philosophical grounds, even a full-fledged logic like second-order logic should be regarded as a mathematical theory since its logical truths by themselves capture substantial mathematical statements. Together with the explosion in the number of proposed logics, this diversity has spurred work in research with a unifying aim that has resulted in the development of:

- logical frameworks that are logics in which other logics can be represented and used, and
- extremely abstract formalisms, such as *institutions*, that intend to capture the defining characteristics of any logic.

A detailed discussion on these topics, as well as on many others not mentioned in this article, can be found in the comprehensive surveys (8–10).

Much of the impetus for the development of mathematical logic came from the desire of providing a solid foundation for mathematics. Nowadays it is computer science that has taken the leading role, and it will be the applications and needs in this area that are bound to guide the future of formal logic.

BIBLIOGRAPHY

1. M. Davis, *Engines of Logic: Mathematicians and the Origin of the Computer*, 2nd edition. New York: W. W. Norton & Company, 2001.

2. A. Tarski, The concept of truth in the languages of the deductive sciences (in Polish), *Prace Towarzystwa Naukowego*

Warszawskiego, Wydzial III Nauk Matematyczno-Fizycznych, **34**, 1933. Expanded English translation in Ref. (11).

3. K. Gödel, The completeness of the axioms of the functional calculus of logic (in German). *Monatshefte für Mathematik und Physik*, **37**: 349–360, 1930. Reprinted in Ref. (12).
4. A. Church, A note on the Entscheidungsproblem, *J. Symbolic Logic*, **1**: 40–41, 1936.
5. A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Pro. Lond. Math. Soc.*, **42**: 230–265, 1936.
6. J. Y. Halpern, R. Harper, N. Immerman, P. G. Kolaitis, M. Y. Vardi, and V. Vianu, On the unusual effectiveness of logic in computer science. *Bull. Symbolic Logic*, **7**(2): 213–236, 2001.
7. W. V. Quine, *Philosophy of Logic*, 2nd edition. Cambridge, MA: Harvard University Press, 1986.
8. S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, eds. *Handbook of Logic in Computer Science*. Cambridge, UK: Oxford Science Publications, 1993.
9. D. M. Gabbay, C. J. Hogger, and J. A. Robinson, eds. *Handbook of Logic in Artificial Intelligence and Logic Programming*. Cambridge, UK: Oxford Science Publications, 1993.
10. D. M. Gabbay and F. Guenthner, eds. *Handbook of Philosophical Logic*. 2nd edition. New York: Springer, 2001.
11. A. Tarski, *Logic, Semantics, Metamathematics, Papers from 1923 to 1938*. Indianapolis, IN: Hackett Publishing Company, 1983.
12. K. Gödel, *Collected Works I: Publications 1929–1936*. London, UK: Oxford University Press, 1986.

FURTHER READING

D. van Dalen, *Logic and Structure*. Fourth Edition. New York: Springer, 2004.

H.-D. Ebbinghaus, J. Flum, and W. Thomas, *Mathematical Logic*, Second Edition. New York: Springer, 1996.

J. A. Goguen and G. Malcolm, *Algebraic Semantics of Imperative Programs*. Cambridge, MA: The MIT Press, 1996.

Miguel Palomino
Universidad Complutense
Madrid, Spain

F

FRACTALS

Even though no consensus exists on a mathematical definition of what constitutes a fractal, it is usually clear what one means by a fractal. A fractal object has strong scaling behavior. That is, some relation exists between the "behavior" at one scale and at (some) finer scales.

Benoit Mandelbrot suggested defining a fractal as an object that has a fractional dimension (see the section on Fractal Dimension). This definition captures the idea that a fractal has complex behavior at all size scales.

Figure 1 illustrates some geometric fractals (that is, the "behavior" is geometrical). The first two images are exactly self-similar fractals. Both of them consist of unions of shrunken copies of themselves. In the first fractal (the one that looks like a complicated plus sign), five copies of the large figure combine together, and in the second fractal (the *twindragon tile*), it only requires two copies. The third fractal in the figure is not exactly self-similar. It is made up of two distorted copies of the whole. The fourth fractal is an image of a Romanesco Broccoli, which is an amazingly beautiful vegetable! The fractal structure of this plant is evident, but it is clearly not an exact fractal.

In Fig. 2, the well-known Mandelbrot Set is explored by successively magnifying a small portion from one frame into the next frame. The first three images in the series indicate the region that has been magnified for the next image. These images illustrate the fact that the Mandelbrot Set has infinitely fine details that look similar (but not exactly the same) at all scales.

One amazing feature of many fractal objects is that the process by which they are defined is not overly complicated. The fact that these seemingly complex systems can arise from simple rules is one reason that fractal and "chaotic" models have been used in many areas of science. These models provide the possibility to describe complex interactions among many simple agents. Fractal models cannot efficiently describe all complex systems, but they are incredibly useful in describing some of these systems.

THE MANDELBROT SET

The *Mandelbrot Set* (illustrated in the first frame of Fig. 2), named after Benoit Mandelbrot (who coined the term *fractal* and introduced fractals as a subject), is one of the most famous of all fractal objects. The Mandelbrot Set is definitely not an exactly self-similar fractal. Zooming in on the set continues to reveal details at every scale, but this detail is never exactly the same as at previous scales. It is conformally distorted, so it has many features that are the same, but not exactly the same.

Although the definition of the Mandelbrot Set might seem complicated, it is (in some sense) much less complicated than the Mandelbrot Set itself. Consider the polynomial with complex coefficients $Q_c(x) = x^2 + c$, where $c = a + bi$ is some (fixed) complex number. We are interested in what happens when we iterate this polynomial starting at $x = 0$. It turns out that one of two things happens, depending on c. Either $Q_c^n(0) \to \infty$ or it stays bounded for all time. That is, either the iterates get larger and larger, eventually approaching infinitely large, or they stay bounded and never get larger than a certain number. The Mandelbrot Set is defined as

$$M = \{c : Q_c^n(0) \nrightarrow \infty\}$$

that is, it is those c in the complex plane for which the iteration does not tend to infinity. Most images of the Mandelbrot Set are brilliantly colored. This coloration is really just an artifact, but it indicates *how fast* those points that are not in the Mandelbrot Set tend to infinity. The Mandelbrot Set itself is usually colored black. In our gray-level images of the Mandelbrot Set (Fig. 2), the Mandelbrot Set is in black.

FRACTAL DIMENSION

Several parameters can be associated with a fractal object. Of these parameters, fractal dimension (of which there are several variants) is one of the most widely used. Roughly speaking, this "dimension" measures the scaling behavior of the object by comparing it to a power law.

An example will make it more clear. Clearly the line segment $L = [0,1]$ has dimension equal to one. One way to think about this is that, if we scale L by a factor of s, the "size" of L changes by a factor of s^1. That is, if we reduce it by a factor of 1/2, the new copy of L has 1/2 the length of the original L.

Similarly the square $S = [0,1] \times [0,1]$ has dimension equal to two because, if we scale it by a factor of s, the "size" (in this case area) scales by a factor of s^2. How do we know that the "size" scales by a factor of s^2? If $s = 1/3$, say, then we see that we can tile S by exactly $9 = 3^2$ reduced copies of S, which means that each copy has "size" $(1/3)^2$ times the size of the original.

For a fractal example, we first discuss a simple method for constructing geometric fractals using an "initiator" and a "generator." The idea is that we start with the "initiator" and replace each part of it with the "generator" at each iteration. We see this in Figs. 3 and 4. Figure 3 illustrates the initiator and generator for the von Koch curve, and Fig. 4 illustrates the iterative stages in the construction, where at each stage we replace each line segment with the generator. The limiting object is called the *von Koch curve.*

In this case, the von Koch curve K is covered by four smaller copies of itself, each copy having been reduced in size by a factor of three. Thus, for $s = 1/3$, we need four reduced copies to tile K. This gives

$$\begin{aligned} \text{size of original copy } &= 4 \times \text{ size of smaller copy} \\ &= 4 \times (1/3)^D \text{ size of orginal copy} \end{aligned}$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

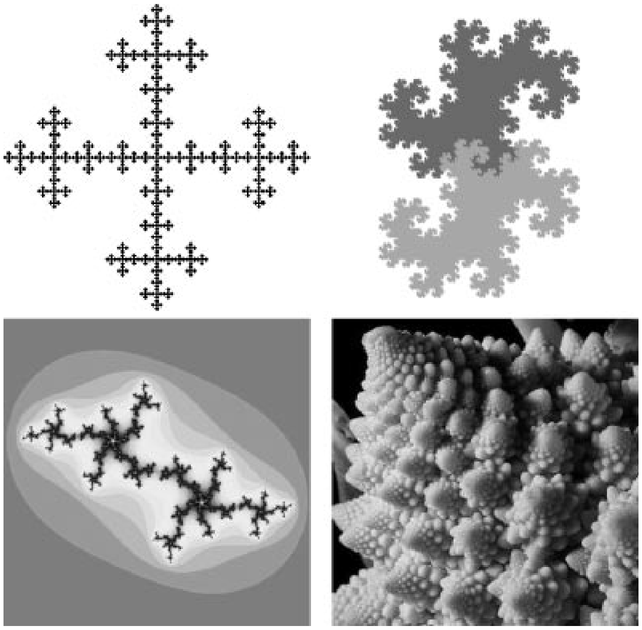


Figure 1. Some fractal shapes.

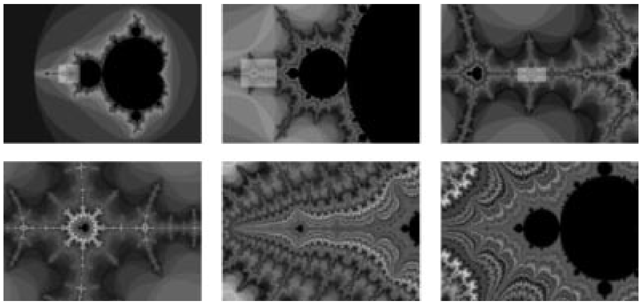


Figure 2. Zooming into the Mandelbrot Set.

so $4(1/3)^D = 1$ or $D = \log(3)/\log(2)$. That is, for the von Koch curve, if we shrink it by a factor of 1/3, then the "size" gets reduced by a factor of $(1/3)^{\log(4)/\log(3)} = 1/4$. In this sense, the von Koch curve has dimension $\log(4)/\log(3) \approx 1.2619$, so it has a dimension that is fractional and is a "fractal."

This way of computing a fractal dimension is very intuitive, but it only works for sets that are exactly self-similar (and thus is called the *similarity dimension*). We need a different method/definition for the fractal dimension of objects that are not exactly self-similar. The diffusion-limited aggregation (DLA) fractal shown later in Fig. 16 is a fractal, but it is not strictly self-similar.

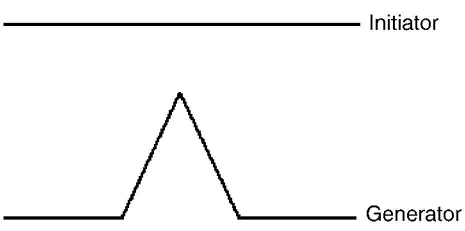


Figure 3. The *initiator* and *generator* for the von Koch curve.

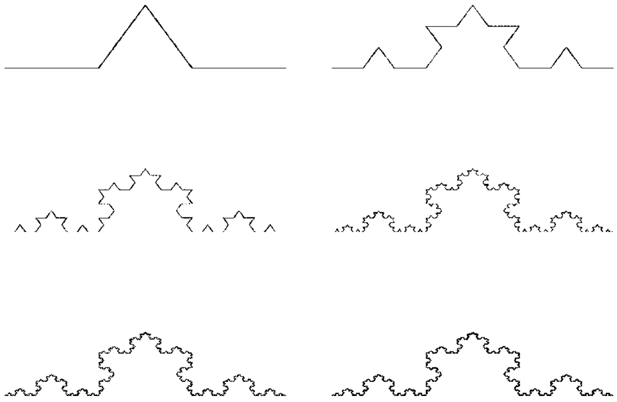


Figure 4. Construction of the von Koch curve.

One very common way to estimate the dimension of such an object is the *box counting method*. To do this, cover the image with a square grid (of side length ε) and count how many of these boxes are occupied by a point in the image. Let $N(\varepsilon)$ be this number. Now repeat this for a sequence of finer and finer grids (letting ε tend to 0). We want to fit the relation $N(\varepsilon) = a\varepsilon^{-D}$, so we take logarithms of both sides to get $\log(N) = \log(a) - D\log(\varepsilon)$. To estimate D, we plot $\log(N(\varepsilon))$ versus $\log(\varepsilon)$ and find the slope of the least-squares line.

For the object in Fig. 16, we have the data in Table 1, which gives a fractal dimension of 1.60.

Table 1.

ε	$N(\varepsilon)$
1/2	4
1/4	16
1/8	52
2^{-4}	174
2^{-5}	580
2^{-6}	1893
2^{-7}	6037
2^{-8}	17556
2^{-9}	44399
2^{-10}	95432

It is interesting to apply this method to exactly self-similar objects, such as the von Koch curve. For this curve, it is convenient to use boxes of size $\varepsilon = 3^{-n}$ and to align them so that they form a nice covering of the von Koch curve (that is, we do not necessarily have to have the boxes form a grid). Assume that the initial lengths of the line segments in the generator for the von Koch curve are all equal to 1/3. In this case, it is easy to see that we require 4^n boxes of side length $1/3^n$ to cover the curve, so we solve

$$4^n = a(1/3^n)^{-D} \Rightarrow a = 1,\ D = \log(4)/\log(3)$$

as before.

IFS FRACTALS

The Iterated Function Systems (IFS, for short) are a mathematical way of formalizing the notion of self-similarity. An IFS is simply a collection of functions from some "space" to itself, w_i: $X \to X$. Usually it is required that these functions are contractive, in that the distance between the images of any two points is strictly less than the distance between the original points. Mathematically, w is a contraction if there is some number s with $0 \leq s < 1$ and $d(w(x), w(y)) \leq sd(x,y)$ for any x and y, where $d(x,y)$ somehow measures the distance between the points x and y. In this case, it is well known (by the *Banach Contraction Theorem*) that there is a unique *fixed point* $\bar{x}$ with $w(\bar{x}) = \bar{x}$.

Given an IFS $\{w_i\}$, the *attractor* of the IFS is the unique set A (A should be nonempty and compact, to be technically precise), which satisfies the *self-tiling* (or fixed point) condition

$$A = w_1(A) \cup w_2(A) \cup \cdots \cup w_n(A)$$

so that A is made up of "smaller" copies of itself (smaller by the individual w_i, that is). Under some general conditions this attractor always exists and is uniquely specified by the functions w_i in the IFS. Furthermore, one can recover the attractor A by starting with any set B and iterating the IFS. The iteration proceeds by first generating the n distorted and smaller copies of B (distorted and shrunk by the individual w_i's) and combining them to get a new set B_1

$$B_1 = w_1(B) \cup w_2(B) \cup \cdots \cup w_n(B)$$

Repeating this, we get a sequence of sets $B_2, B_3, \ldots, B_n, \ldots,$ which will converge to the attractor A.

To illustrate, take the three contractions w_1, w_2, w_3 given by

$$w_1(x,y) = (x/2, y/2), \quad w_2(x,y) = (x/2 + 1/2, y/2),$$
$$w_3(x,y) = (x/2, y/2 + 1/2)$$

Each of these three maps shrinks all distances by a factor of two (because we are reducing in both the horizontal and the vertical direction by a factor of 2). These maps are examples of *similarities*, because they preserve all angles and lines but only reduce lengths; geometric relationships within an object are (mostly) preserved.

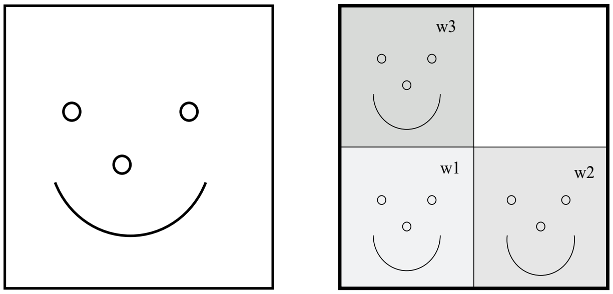


Figure 5. Action of the three maps.

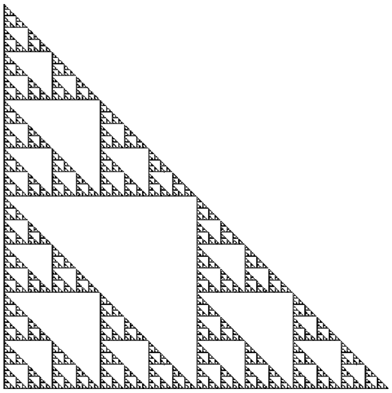


Figure 6. The Sierpinski Gasket.

We think of these maps as acting on the unit square, that is, all points (x, y) with $0 \leq x \leq 1$ and $0 \leq y \leq 1$. In this case, the action of each map is rather simple and is illustrated in Fig. 5. For instance, map w_2 takes whatever is in the square, shrinks it by a factor of two in each direction, and then places it in the lower right-hand corner of the square.

The attractor of this 3-map IFS is the Sierpinski Gasket, which is illustrated in Fig. 6. The self-tiling property is clear, because the Sierpinski Gasket is made up of three smaller copies of itself. That is, $S = w_1(S) \cup w_2(S) \cup w_3(S)$.

Now, one amazing thing about IFS is that iterating the IFS on any initial set will yield the same attractor in the limit. We illustrate this in Fig. 7 with two different starting images. The attractor of the IFS is completely encoded by the IFS maps themselves; no additional information is required.

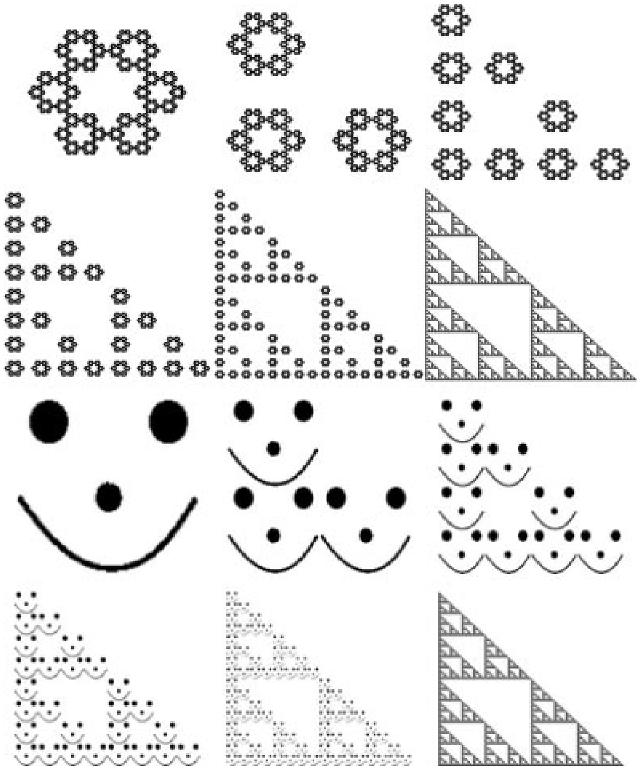


Figure 7. Two different deterministic iterations.

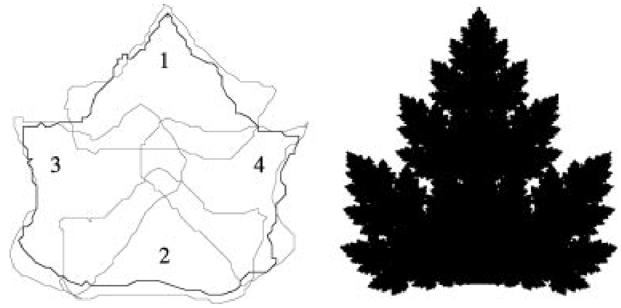


Figure 8. Collage for maple leaf and attractor of the IFS.

The Collage Theorem

Given an IFS it is easy to compute the attractor of the IFS by iteration. The inverse problem is more interesting, however. Given a geometric object, how does one come up with an IFS whose attractor is close to the given object?

The Collage Theorem provides one answer to this question. This theorem is a simple inequality but has strong practical applications.

Collage Theorem. Suppose that T is a contraction with contraction factor $s < 1$ and A is the attractor of T (so that $T(A) = A$). Then for any B we have

$$d(A,B) \leq \frac{d(T(B),B)}{1-s}$$

What does this mean? In practical terms, it says that, if we want to find an IFS whose attractor is close to a given set B, then what we do is look for an IFS that does not "move" the set B very much. Consider the maple leaf in Fig. 8. We see that we can "collage" the leaf with four transformed copies of itself. Thus, the IFS that is represented by these four transformations should have an attractor that is close to the maple leaf. The second image in Fig. 8 is the attractor of the given IFS, and it is clearly very similar to a maple leaf.

As another example, it is easy to find the collage to generate the attractor in Fig. 9. It requires 11 maps in the IFS.

However, the real power of the Collage Theorem comes in when one wants to find an IFS for more complicated self-affine fractals or for fractal objects that are not self-similar. One such application comes when using IFS for image representation (see the section on Fractals and Image Compression).

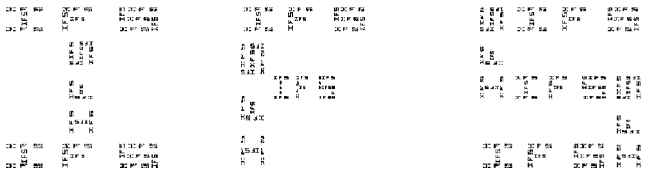


Figure 9. An easy collage to find.

Fractal Functions

Geometric fractals are interesting and useful in many applications as models of physical objects, but many times one needs a functional model. It is easy to extend the IFS framework to construct fractal functions.

There are several different IFS frameworks for constructing fractal functions, but all of them have a common core, so we concentrate on this core. We illustrate the ideas by constructing fractal functions on the unit interval; that is, we construct functions $f : [0, 1] \to \mathbb{R}$. Take the three mappings $w_1(x) = x/4$, $w_2(x) = x/2 + 1/4$, and $w_3(x) = x/4 + 3/4$ and notice that $[0, 1] = w_1[0, 1] \cup w_2[0, 1] \cup w_3[0, 1]$, so that the images of [0, 1] under the w_i's tile [0, 1].

Choose three numbers $\alpha_1, \alpha_2 \alpha_3$ and three other numbers $\beta_1, \beta_2, \beta_3$ and define the operator T by

$$T(f)(x) = \alpha_i f(w_i^{-1}(x)) + \beta_i \quad \text{if } x \in w_i([0, 1])$$

where $f : [0, 1] \to \mathbb{R}$ is a function. Then clearly $T(f)$: $[0, 1] \to \mathbb{R}$ is also a function, so T takes functions to functions.

There are various conditions under which T is a contraction. For instance, if $|\alpha_i| < 1$ for each i, then T is contractive in the supremum norm given by

$$\| f\|_{\text{sup}} = \sup_{x \epsilon [0, 1]} | f(x)|$$

so $T^n(f)$ converges uniformly to a unique fixed point $\bar{f}$ for any starting function f.

Figure 10 illustrates the limiting fractal functions in the case where $\alpha_1 = \alpha_2 = \alpha_3 = 0.3$ and $\beta_1 = -\beta_2 = \beta_3 = -1$.

It is also possible to formulate contractivity conditions in other norms, such as the L^p norms. These tend to have weaker conditions, so apply in more situations. However, the type of convergence is clearly different (the functions need not converge pointwise anywhere, for instance, and may be unbounded).

The same type of construction can be applied for functions of two variables. In this case, we can think of such functions as grayscale images on a screen. For example,

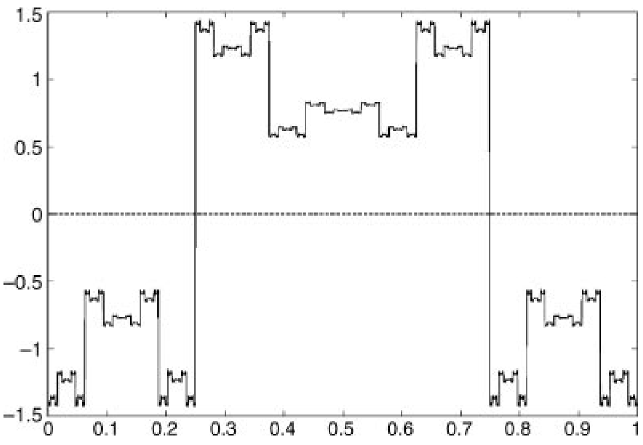


Figure 10. The attractor of an IFS on functions.

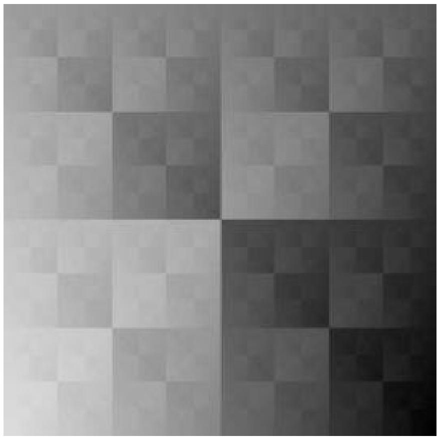


Figure 11. The attractor of an IFS on functions $f : \mathbb{R}^2 \to \mathbb{R}$.

using the four "geometric" mappings

$$w_1(x,y) = \left(\frac{x}{2},\frac{y}{2}\right), \quad w_2(x,y) = \left(\frac{x+1}{2},\frac{y}{2}\right),$$

$$w_3(x,y) = \left(\frac{x}{2},\frac{y+1}{2}\right), \quad w_4(x,y) = \left(\frac{x+1}{2},\frac{y+1}{2}\right)$$

and the α and β values given by

α	0.5	0.5	0.4	0.5
β	80	40	132	0

we get the attractor function in Fig. 11. The value 255 represents white and 0 represents black.

FRACTALS AND IMAGE COMPRESSION

The idea of using IFS for compressing images occurred to Barnsley and his co-workers at the Georgia Institute of Technology in the mid-1980s. It seems that the idea arose from the amazing ability of IFS to encode rather complicated images with just a few parameters. As an example, the fern leaf in Fig. 12 is defined by four affine maps, so it is encoded by 24 parameters.

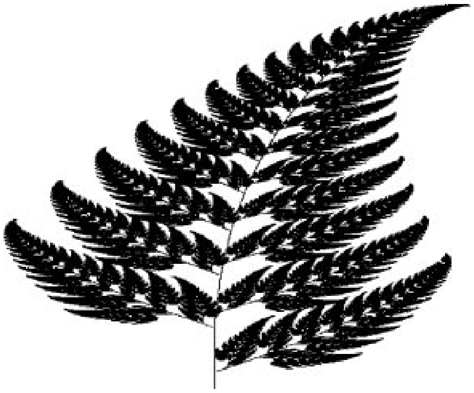


Figure 12. The fern leaf.

Of course, this fern leaf is an exact fractal and most images are not. The idea is to try to find approximate self-similarities, because it is unlikely there will be many exact self-similarities in a generic image. Furthermore, it is also highly unlikely that there will be small parts of the image that look like reduced copies of the entire image. Thus, the idea is to look for "small" parts of the image that are similar to "large" parts of the same image. There are many different ways to do this, so we describe the most basic here.

Given an image, we form two partitions of the image. First, we partition the image into "large" blocks, we will call these the *parent blocks*. Then we partition the image into "small" blocks; here we take them to be one half the size of the large blocks. Call these blocks *child blocks*. Figure 13 illustrates this situation. The blocks in this figure are made large enough to be clearly observed and are much too large to be useful in an actual fractal compression algorithm.

Given these two block partitions of the image, the *fractal block encoding* algorithm works by scanning through all the small blocks and, for each such small block, searching among the large blocks for the best match. The likelihood of finding a good match for all of the small blocks is not very high. To compensate, we are allowed to modify the large block. Inspired by the idea of an IFS on functions, the algorithm is:

```
1. for SB in small blocks do
2. for LB in large blocks do
```

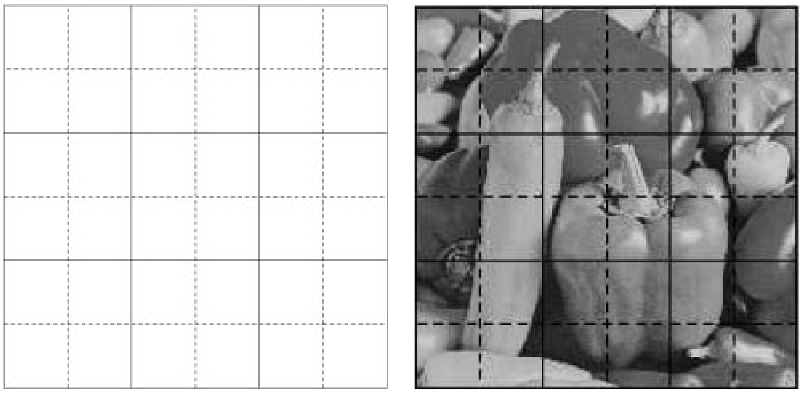


Figure 13. The basic block decomposition.

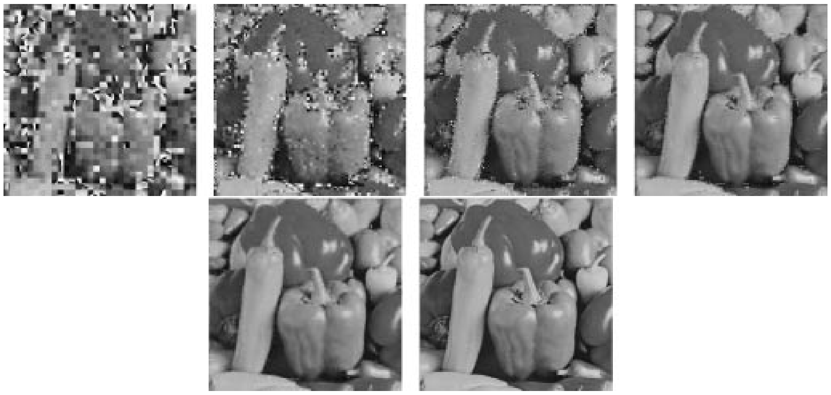


Figure 14. Reconstruction: 1, 2, 3, 4, and 10 iterations and the original image.

```
3. Downsample LB to the same size as SB
4. Use leastsquares to find the best parameters $\alpha$ and $\beta$
   for this combination of LB and SB. That is, to make
   SB $\approx \alpha$LB $+ \beta$.
5. Compute error for these parameters. If error is smaller
   than for any other LB, remember this pair along
   with the $\alpha$ and $\beta$.
6. end for
7. end for
```

At the end of this procedure, for each small block, we have found an optimally matching large block along with the α and β parameters for the match. This list of triples (large block, α, β) forms the encoding of the image. It is remarkable that this simple algorithm works! Figure 14 illustrates the first, second, third, fourth, and tenth iterations of the reconstruction. One can see that the first stage of the reconstruction is basically a downsampling of the original image to the "small" block partition. The scheme essentially uses the β parameters to store this coarse version of the image and then uses the α parameters along with the choice of which parent block matched a given child block to extrapolate the fine detail in the image from this coarse version.

STATISTICALLY SELF-SIMILAR FRACTALS

Many times a fractal object is not self-similar in the IFS sense, but it is self-similar in a statistical sense. That is, either it is created by some random self-scaling process (so the steps from one scale to the next are random) or it exhibits similar statistics from one scale to another. In fact, most naturally occurring fractals are of this statistical type.

The object in Fig. 15 is an example of this type of fractal. It was created by taking random steps from one scale to the next. In this case, it is a subset of an exactly self-similar fractal.

These types of fractals are well modeled by random IFS models, where there is some randomness in the choice of the maps at each stage.

MORE GENERAL TYPES OF FRACTALS

IFS-type models are useful as approximations for self-similar or almost self-similar objects. However, often these models are too hard to fit to a given situation. For these cases we have a choice—we can either build some other type of model governing the growth of the object OR we can give up on finding a model of the interscale behavior and just measure aspects of this behavior.

An example of the first instance is a simple model for DLA, which is illustrated in Fig. 16. In this growth model, we start with a seed and successively allow particles to drift

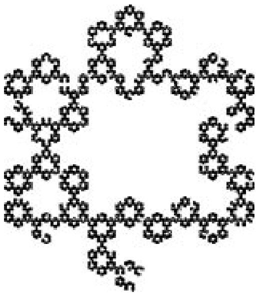


Figure 15. A statistically self-similar fractal.

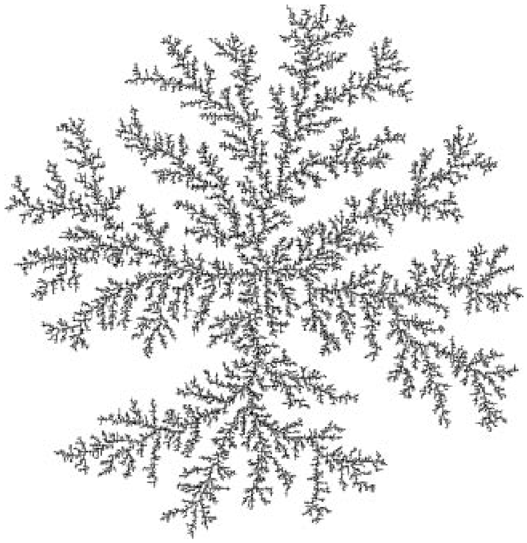


Figure 16. A DLA fractal.

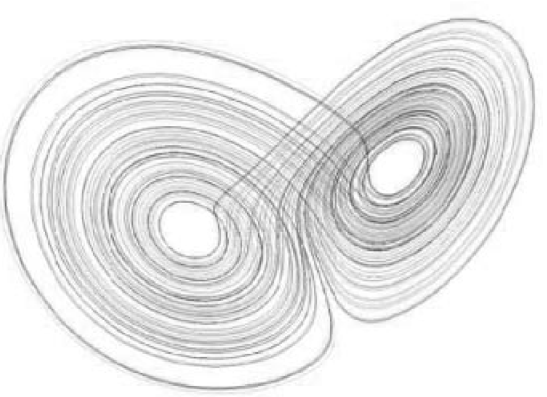


Figure 17. The Lorentz attractor.

around until they "stick" to the developing object. Clearly the resulting figure is fractal, but our model has no explicit interscale dependence. However, these models allow one to do simulation experiments to fit data observed in the laboratory. They also allow one to measure more global aspects of the model (like the fractal dimension).

Fractal objects also frequently arise as so-called *strange attractors* in chaotic dynamical systems. One particularly famous example is the butterfly shaped attractor in the Lorentz system of differential equations, seen in Fig. 17. These differential equations are a toy model of a weather system that exhibits "chaotic" behavior. The attractor of this system is an incredibly intricate filigree structure of curves. The fractal nature of this attractor is evident by the fact that, as you zoom in on the attractor, more and more detail appears in an approximately self-similar fashion.

FRACTAL RANDOM PROCESSES

Since the introduction of fractional Brownian motion (fBm) in 1968 by Benoit Mandelbrot and John van Ness, self-similar stochastic processes have been used to model a variety of physical phenomena (including computer network traffic and turbulence). These processes have power spectral densities that decay like $1/f^{\alpha}$.

A fBm is a Gaussian process $x(t)$ with zero mean and covariance

$$E(x(t)x(s)] = (\sigma^2/2)\,[|t|^{2H} + |s|^{2H} - |t-s|^{2H}]$$

and is completely characterized by the *Hurst exponent H* and the variance $E[x(1)^2] = \sigma^2$. fBm is statistically sell-similar in the sense that, for any scaling $a > 0$, we have

$$x(at) = a^H x(t)$$

where by equality we mean equality in distribution. (As an aside and an indication of one meaning of H, the sample paths ol fBm with parameter H are almost surely Hölder continuous with parameter H, so the larger the value of H, the smoother the sample paths of the fBm). Because of this scaling behavior, fBm exhibits very strong long-time dependence. It is also clearly not a stationary (time-invariant) process, which causes many problems with traditional methods of signal synthesis, signal estimation, and parameter estimation. However, wavelet-based methods work rather well, as the scaling and finite time behavior of wavelets matches the scaling and nonstationarity of the fBm. With an appropriately chosen (i.e., sufficiently smooth) wavelet, the wavelet coefficients of an fBm become a stationary sequence without the long range dependence properties. This aids in the estimation of the Hurst parameter H.

Several generalizations of fBm have been defined, including a multifractal Brownian motion (mBm). This particular generalization allows the Hurst parameter to be a changing function of time $H(t)$ in a continuous way. Since the sample paths of fBm have Hölder continuity H (almost surely), this is particularly interesting for modeling situations where one expects the smoothness of the sample paths to vary over time.

FRACTALS AND WAVELETS

We briefly mention the connection between fractals and wavelets. Wavelet analysis has become a very useful part of any data analyst's toolkit. In many ways, wavelet analysis is a supplement (and, sometimes, replacement) for Fourier analysis; the wavelet functions replace the usual sine and cosine basis functions.

The connection between wavelets and fractals comes because wavelet functions are nearly self-similar functions. The so-called "scaling function" is a fractal function, and the "mother wavelet" is simply a linear combination of copies of this scaling function. This scaling behavior of wavelets makes it particularly nice for examining fractal data, especially if the scaling in the data matches the scaling in the wavelet functions. The coefficients that come from a wavelet analysis are naturally organized in a hierarchy of information from different scales, and hence, doing a wavelet analysis can help one to find scaling relations in data, if such relations exist.

Of course, wavelet analysis is much more than just an analysis to find scaling relations. There are many different wavelet bases. This freedom in choice of basis gives greater flexibility than Fourier analysis.

FURTHER READING

M. G. Barnsley, *Fractals Everywhere*, New York: Academic Press, 1988.

M. G. Barnsley and L. Hurd, *Fractal Image Compression*, Wellesley, Mass: A.K. Peters, 1993.

M. Dekking, J. Lévy Véhel, E. Lutton, and C. Tricot (eds). *Fractals: Theory and Applications in Engineering*, London: Springer, 1999.

K. J. Falconer, *The Geometry of Fractal Sets*, Cambridge, UK: Cambridge University Press, 1986.

K. J. Falconer, *The Fractal Geometry: Mathematical Foundations and Applications*, Toronto, Canada: Wiley, 1990.

J. Feder, *Fractals*, New York: Plenum Press, 1988.

Y. Fisher, *Fractal Image Compression, Theory and Applications*, New York: Springer, 1995.

J. E. Hutchinson, Fractals and self-similarity, *Indiana Univ. Math. J.* **30**: 713–747.

J. Lévy Véhel, E. Lutton, and C. Tricot (eds). *Fractals in Engineering*, London: Springer, 1997.

J. Lévy Véhel and E. Lutton (eds). *Fractals in Engineering: New Trends in Theory and Applications*, London: Springer, 2005.

S. Mallat, *A Wavelet Tour of Signal Processing*, San Diego, CA: Academic Press, 1999.

B. Mandelbrot, *The Fractal Geometry of Nature*, San Francisco, CA: W. H. Freeman, 1983.

B. Mandelbrot and J. van Ness, Fractional Brownian motions, fractional noises and applications, *SIAM Review* **10**: 422–437, 1968.

H. Peitgen and D. Saupe (eds). *The Science of Fractal Images*, New York: Springer, 1998.

H. Peitgen, H. Jürgens, and D. Saupe, *Chaos and Fractals: New Frontiers, of Science*, New York: Springer, 2004.

D. Ruelle, *Chaotic Evolution and Strange Attractors: The Statistical Analysis of Time Series for Deterministic Nonlinear Systems*, Cambridge, UK: Cambridge University Press, 1988.

F. Mendivil
Acadia University
Wolfville, Nova Scotia, Canada

GEOMETRIC PROGRAMMING

INTRODUCTION

Nonlinear programming deals with optimization problems in which both the objective and constraint functions are nonlinear. Of the many special cases of a nonlinear program, geometric programs are a class that involves a highly nonlinear function called a posynomial (as detailed later). Functions of this type were first formalized by Duffin et al. (1) in the seminal work entitled *Geometric Programming*. In that book, many intriguing properties of posynomial functions are given along with many applications to engineering design. Subsequent work developed in two significantly different directions: Signomial programming and Generalized Geometric Programming, both of which are reviewed in this article (2).

Signomial programming, which was first developed by Passy and Wilde (3), sought to relax the restriction of the functions treated by geometric programming to the form of polynomials. With the relaxation to polynomials, one loses convexity and, hence, one can only hope for a local minimum, not a global minimum. Generalized geometric programming proceeds to generalize the type of function considered from posynomial to convex. This generalization maintains the convexity of the problem, and hence, one can find the global solution to the mathematical program.

The relationship among the various developments may outwardly seem tenuous, because geometric programming should not be thought of as a class of mathematical programs but instead as a method of analysis of mathematical programs. In this respect, geometric programming has much in common with dynamic programming. Dynamic programming makes use of recursive functions. Geometric programming makes use of linearity, separability, convexity, and duality.

Linearity

Most mathematical programming is conducted over linear vector spaces. The usual choice is Euclidean n-dimensional space (E_n). Here, a vector is represented by an n-tuple of real components. Taking any two vectors x_1 and x_2 that belong to the space, and real scalars α and β, a linear vector space has the property that the linear combination $\alpha x_1 + \beta x_2$ belongs to the space. In the context of functions, a function $\ell(x)$ is said to be linear if for any scalars α and β, $\ell(\alpha x + \beta y) = \alpha\ell(x) + \beta\ell(y)$. Linear functions have many important properties. They are both convex and concave. Furthermore, a first-order Taylor series expansion approximates the linear function exactly. The value of linearity to mathematical programming is best exemplified by the fruitful area of linear programming.

Separability

A function $f(x_1, x_2, \ldots, x_n)$ is separable if it can be written as $f(x_1, x_2, \ldots, x_n) = f_1(x_1) + f_2(x_2) + \cdots + f_n(x_n)$. This property is important as it allows one to treat an n-dimensional function as n functions of one-dimension, which is much simpler from a computational point of view.

Convexity

A set A is convex if for any $x_1, x_2 \in A$ and $0 \leq \lambda \leq 1$, $\lambda x_1 + (1-\lambda)x_2 \in A$. To extend the concept to a function $f(x)$ defined for $x \in C$, a convex set, we require the following definition. The epigraph of a function f defined on C is the set A defined by: $A = \{(\alpha, x) | \alpha \geq f(x), x \in C\}$. $f(x)$ is a convex function if A is a convex set. Convex functions have several powerful properties. The most important for mathematical programs with a convex objective function and a convex constraint set is that any local minimum is also a global minimum. Additional properties of convexity will be given in the section on Generalized Geometric Programming.

Duality

For a linear space $X = E_n$, linear functions may be written in the form $\langle a, x \rangle = \sum_{i=1}^{n} a_i x_i$, where a is a vector parameter. The set of all linear functions (parameterized by the vector a) is a space and is called the dual space (X^*). The duality develops from the fact that $\langle a, x \rangle$ is symmetric in a and x. For finite dimensional spaces, the set of linear functions on X^* turns out to be the original space X. Hence there is a symmetric relationship between X and X^*.

Of importance to mathematical programming is the pairing of sub-spaces in primal and dual spaces with respect to the inner product. Let χ be a subspace of X. Then $\chi^{\perp} = \{y | \langle y, x \rangle = 0, \forall x \in \chi\}$ is called the orthogonal complementary subspace of χ. Any point in $\chi^{\perp}$ is orthogonal to any point in χ by construction. If $y_1, y_2 \in \chi^{\perp}$ and α, β are scalars, then it is easy to observe that $\alpha y_1 + \beta y_2 \in \chi^{\perp}$.

The importance of the concepts of linearity, separability, convexity, and duality in the development of geometric programming will remain obscure in the sections on unconstrained polysynomial programming constrained polysynomial programming and signomial programming. However, their paramount role will become clear in the section on Generalized geometric programming. The last section uses this generalized theory to rederive the theory of posynomial programs, and the central role played by these concepts will be clarified even more. Throughout this article, theory will be reinforced by examples drawn mainly from the area of inventory theory. The examples presented are relatively simple to highlight the use of geometric programming theory. The article concludes with examples from many other areas.

UNCONSTRAINED POSYNOMIAL PROGRAMMING

Here we consider the following nonlinear problem:

$$\text{Minimize } g(t) = \sum_{i=1}^{n} u_i \qquad (1)$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

where $u_1, i = 1, \ldots, n$, are posynomials defined by

$$u_i = c_i \prod_{j=1}^{m} t_j^{a_{ij}} \tag{2}$$

$c_i > 0, \forall i, t_j > 0, \forall j$, and a_{ij} are arbitrary real constants; $t = (t_1, \ldots, t_m)^{\mathrm{T}}$ and T denotes a transpose. In the usual situation, $g(t)$ represents a composite cost made up of component costs u_i, $i = 1, \ldots, n$. We term this the primal problem.

In principle, the primal problem could be solved by the methods of differential calculus. This problem will develop a set of nonlinear equations which, in general, are difficult to solve. Hence, in practice, one would invariably resort to a numerical solution by an iterative descent-type method. Here, we propose to construct a geometric programming dual program to the primal. In certain cases, this proposition admits a trivial solution to the problem. Furthermore, this dual problem has an interesting and informative practical interpretation. A central idea here is the arithmetic–geometric mean inequality, henceforth termed the *geometric inequality*. In fact, the name "geometric programming" comes from the importance of this inequality to the original theory. It states that

$$\sum_{i=1}^{n} \delta_i v_i \geq \prod_{i=1}^{n} (v_i)^{\delta_i} \tag{3}$$

where $\sum_{i=1}^{n} \delta_i = 1, \quad \delta_i \geq 0, \quad \forall i, \quad v_i > 0, \quad \forall i$. Equality is attained when $v_1 = v_2 = \cdots = v_n$. It is convenient to set $v_i = u_i / \delta_i, \ \forall i$ in Equation (3) to obtain

$$\sum_{i=1}^{n} u_i \geq \prod_{i=1}^{n} (u_i/\delta_i)^{\delta_i} \tag{4}$$

In this form, we can use the geometric inequality to obtain a lower bound on our primal optimization problem. Hence, substituting Equation (2) into Equation (1) and using Equation (4), we have that

$$g(t) \geq \prod_{i=1}^{n} (c_i/\delta_i)^{\delta_i} \prod_{j=1}^{m} t_j^{\sum_{i=1}^{n} a_{ij}\delta_i} \tag{5}$$

If we choose the weights, $\delta_i, \ \forall i$, such that

$$\sum_{i=1}^{n} a_{ij}\delta_i = 0, \quad j = 1, \ldots, m \tag{6}$$

then the variables t_j, $j = 1, \ldots, m$ on the right hand side of inequality (5) may be eliminated. In this case, we have that

$$g(t) \geq \prod_{i=1}^{n} (c_i/\delta_i)^{\delta_i} = v(\delta) \tag{7}$$

where $v(\delta)$ is the dual function which gives a lower bound on the minimum of $g(t)$. Equation (7) implies that

$$\min g(t) \geq \max v(\delta) \tag{8}$$

under conditions $t > 0, \ \delta \geq 0, \ \sum_{i=1}^{n} \delta_i = 1$ and Equation (6).

It may be shown that there exists an optimal t^* [in the sense of minimizing $g(t)$] and an optimal δ^* [in the sense of maximizing $v(\delta)$] to satisfy inequality (8) at equality. In this case the relation between t^* and δ^* is given by

$$\delta_i^* = u_i(t^*)/g(t^*) \tag{9}$$

Hence the following dual program to the original primal program may be constructed:

$$\text{Maximize } v(\delta) = \prod_{i=1}^{n} (c_i/\delta_i)^{\delta_i} \tag{10}$$

subject to the normalization condition

$$\sum_{i=1}^{n} \delta_i = 1, \quad \delta_i \geq 0 \tag{11}$$

and the orthogonality condition

$$\sum_{i=1}^{n} a_{ij}\delta_i = 0, \quad j = 1, \ldots, m \tag{12}$$

At optimality $g(t^*) = v(\delta^*)$.

From Equation (9), we can interpret the dual variables at optimality, $\delta_i^*, \forall i$, as the relative contribution of each component cost u_i to the composite cost $g(t)$. Furthermore, we obtain the following relationship between the primal and dual variables

$$\text{In}(\delta_i^* v(\delta^*)/c_i) = \sum_{j=1}^{m} a_{ij} \log t_j^* \tag{13}$$

This equation is a system of linear equations in $\log t_j$, $j = 1, \ldots, m$, which are readily solvable once the dual program has been solved.

We note that the dual program maximizes a nonlinear function subject to linear equality constraints. A dual variable δ_i is associated with each term $i = 1, \ldots, n$ in the primal formulation. In the case that the number of terms in the primal n is equal to the number of variables in the primal m plus one (i.e., $n = m + 1$) the linear constraints admit a unique solution δ^*, and the optimization problem in the dual is trivial. The quantity $n - (m + 1)$ is termed the degree of difficulty of a geometric program and is, in some sense, a measure of the computational complexity of the dual program.

Example. We consider the well known "Economic Lot Size" problem from inventory theory. Items are withdrawn continuously from inventory at a known constant rate a. Items are ordered in equal numbers Q at a time and production is instantaneous. The problem is to determine how often to make a production run and how much to order, Q, each time to minimize the cost C per unit time, where $C = aK/Q + hQ/2 + ac$, and K is the fixed set-up cost, h is the inventory holding cost per item per unit time, and c is the variable production cost per item. Neglecting the constant part, ac, we require to

$$\text{Minimize}\, C_1 = aK/Q + hQ/2$$

This equation is an unconstrained posynomial program. The corresponding dual program, from Equations (10), (11), and (12), is to maximize

$$\begin{aligned}
&(aK/\delta_1)^{\delta_1}(h/2\delta_2)^{\delta_2}\\
&\text{subject to}\, \delta_1 + \delta_2 = 1 \quad \text{(normality)}\\
&-\delta_1 + \delta_2 = 0 \quad \text{(orthogonality)}\\
&\delta_1 \geq 0, \quad \delta_2 \geq 0
\end{aligned}$$

Here the constraint equations can be uniquely solved to yield $\delta_1^* = \delta_2^* = 1/2$, and no maximization problem exists. We have a program with zero degree of difficulty. The interpretation of this result is that at optimality, the set-up cost per unit time and the holding cost per unit time contribute equal amounts to the optimal cost, that is, the optimal distribution of cost is an invariant with respect to the cost coefficients. Hence, from Equations (10) and (13), the optimal cost is $\sqrt{2aKh}$, and the optimal order quantity is $\sqrt{2aK/h}$.

Suppose now that the production process is a lengthy one so that the assumption of instantaneous availability is no longer acceptable. Furthermore, the size of the lot determines the length of the production process, and this in turn determines the in-process inventory holding cost. In this case, a modified economic lot size model could be to minimize $C_1 = aK/Q + hQ/2 + IQT/2$, where I is the in-process inventory holding cost per unit time, and T is the length of the production process, given by $T(Q) = nQ + m$, where n and m are empirically determined constants. In this case, we have the unconstrained posynomial program:

$$\text{Minimize}\, C_1 = aK/Q + Q(h/2 + Im/2) + InQ^2/2$$

The corresponding dual program is given by

$$\text{Maximize}\, (aK/\delta_1)^{\delta_1}((h + Im)/2\delta_2)^{\delta_2}(In/2\delta_3)^{\delta_3}$$

subject to

$$\begin{aligned}
&\delta_1 + \delta_2 + \delta_3 = 1\\
&-\delta_1 + \delta_2 + 2\delta_3 = 0\\
&\delta_1 > 0, \quad \delta_2 > 0, \quad \delta_3 > 0
\end{aligned}$$

Here, we have two equations and three unknowns; in effect, a degree of difficulty of one. One could proceed to solve this problem as an unconstrained maximization problem in one variable or as a constrained maximization. However, insight may be gained without going into this detail. Manipulation of the constraint equations gives the following inequalities: $0.5 \leq \delta_1 \leq 1.0$ and $0.33 \leq \delta_1 \leq 0.66$, which imply that at optimality, the set-up cost makes a contribution between 50% and 66% to the optimal composite cost. Similar insights may be obtained from the other dual variables.

CONSTRAINED POSYNOMIAL PROGRAMMING

Here we consider the minimization of a posynomial form subject to constraints that are also posynomials, which is the primal formulation.

$$\text{Minimize}\, g_0(\mathrm{t}) \tag{14}$$

$$\text{subject to}\, g_k(\mathrm{t}) \leq 1, \quad k = 1, \ldots, \mathrm{p} \tag{15}$$

$$t_j > 0, \quad j = 1, \ldots, m \tag{16}$$

where

$$g_k(t) = \sum_{i \in [k]} c_i \prod_{j=1}^{m} t_j^{a_{ij}}, \quad k = 0, \ldots, p \tag{17}$$

$$\begin{aligned}
&[k] = \{m_k, m_k + 1, \ldots, n_k\}, \quad k = 0, \ldots, p\\
&\text{and}\\
&m_0 = 1, m_1 = n_0 + 1, \ldots, m_p = n_{p-1} + 1, n_p = n.
\end{aligned} \tag{18}$$

$\{[k]\}$, $k = 0, 1, \ldots, p$ is a sequential partition of the integers 1 to n. As before a_{ij} are arbitrary real exponents and the coefficients c_i are positive.

To handle constraints, we need a generalization of the geometric inequality in which the δ_i's are not normalized. We let

$$\lambda_k = \sum_{i \in [k]} \delta_i \tag{19}$$

for a particular constraint k. Hence the geometric inequality Equation (4) may be written in a convenient form as

$$\left(\sum_{i \in [k]} u_i\right)^{\lambda_k} \geq \prod_{i \in [k]} (u_k/\delta_i)^{\delta_i} \lambda_k^{\lambda_k} \tag{20}$$

Combining Equations (15), (17), and (20) we have that

$$1 \geq g_k(t)^{\lambda_k} \geq \prod_{i \in \{k\}} \left(c_i \prod_{j\, 1}^{m} t_j^{a_{ij}}/\delta_i\right)^{\delta_i} \lambda_k^{\lambda_k} \tag{21}$$

for any constraint $k = 1, \ldots, p$. For the objective function, we normalize the $\delta_i, i \in [0]$. Hence

$$g_0(t) \geq \prod_{i \in [0]} \left(c_i \prod_{j=1}^{n} t_j^{a_{ij}}/\delta_i\right)^{\delta_i} \tag{22}$$

and

$$\sum_{i \in [0]} \delta_i = 1 \tag{23}$$

Combining results from Equations (21) and (22), we obtain the inequality

$$g_0(t) \geq \prod_{i=1}^{n} (c_i/\delta_i)^{\delta_i} \prod_{k=1}^{p} \lambda_i^{\lambda_k} \tag{24}$$

Hence, using the same lines of reasoning as for the unconstrained problem, the dual problem is given by:

$$\text{Maxmize}\, v(\delta) = \prod_{i=1}^{n} (c_i/\delta_i)^{\delta_i} \prod_{k=1}^{p} \lambda_k^{\lambda_k} \tag{25}$$

subject to the linear constraints

$$\sum_{i \in [0]} \delta_i = 1 \quad \text{(normalization)} \tag{26}$$

$$\sum_{i=1}^{n} a_{ij}\delta_i = 0, \quad j = 1, \ldots, m \quad \text{(orthogonality)} \tag{27}$$

$$\delta_i \geq 0, \quad i = 1, \ldots, n \tag{28}$$

We note that the primal problem is a highly nonlinear mathematical program, whereas the dual is the maximization of a concave function subject to linear constraints. Once again, if $n = m + 1$, the program has zero degree of difficulty, and the dual program is trivial. It may be shown in Equation (3) that at optimality the primal variables t^* and the dual variables δ^* are related by

$$c_i \prod_{j=1}^{m} t_j^{*a_{ij}} = \begin{cases} \delta_i^* v(\delta^*), & i \in [0] \\ \delta_i^*/\lambda_k(\delta^*), & i \in [k], \lambda_k(\delta^*) > 0,\, k = 1, \ldots, p \end{cases} \tag{29}$$

Hence $\delta_i^*, i \in [0]$ gives the relative contribution of each component cost to the composite cost. Also at optimality, the primal and dual objective functions are equal in value [i.e., $g_0(t^*) = v(\delta^*)$].

Example. We consider an economic lot size model with multiproducts and a resource constraint. In this case, we have the following constrained posynomial program:

$$\text{Minimize} \sum_{i=1}^{N} (a_i K_i/Q_i + h_i Q_i/2)$$

subject to a resource constraint

$$\sum_{i=1}^{N} b_i Q_i \leq W$$

where b_i and W are given constants. From Equations (25–28), we find the corresponding dual program:

$$\begin{aligned}
&\text{Maximize} \prod_{i=1}^{N} (a_i K_i/\delta_i)^{\delta_i} \prod_{i=N+1}^{2N} (h_i/2\delta_i)^{\delta_i} \prod_{i=2N+1}^{3N} (b_i/W\delta_i)^{\delta_i} \lambda^{\lambda} \\
&\text{subject to} \sum_{i=1}^{2N} \delta_i = 1 \\
&\lambda = \sum_{i=2N+1}^{3N} \delta_i \\
&-\delta_i + \delta_{N+i} + \delta_{2N+i} = 0, \quad i = 1, \ldots, N \\
&\delta_i \geq 0, \quad i = 1, \ldots, 3N
\end{aligned}$$

SIGNOMIAL PROGRAMMING

Although posynomial programming has found application in a variety of areas, many problems of interest have fallen outside the posynomial form. Often, it was the positivity condition on the coefficients that were violated. However, the benefits of posynomial programming were too tempting to be rejected. Passy and Wilde (2) introduced a generalization of posynomial programming, in which the coefficients of the terms were not required to be positive. This class of programs is called signomial. In signomial programming, a global minimum cannot be guaranteed. However, the properties of the dual subspace are maintained.

Duffin et al. (3) have developed methods for the analysis of signomials and an algorithm for solving them (at least for a local solution, if not for a global solution). To use the analysis, signomial programs must first be converted into posynomial programs. Any signomial $f(t)$ can be written as a difference of two posynomials, say $r(t)$ and $s(t)$ hence

$$\begin{aligned}
f(t) &\leq 1 \leftrightarrow r(t) - s(t) \leq 1 \leftrightarrow r(t) \leq s(t) + 1 \\
f(t) &\geq 1 \leftrightarrow r(t) - s(t) \geq 1 \leftrightarrow r(t) \geq s(t) + 1 \\
f(t) &\leq 0 \leftrightarrow r(t) - s(t) \leq 0 \leftrightarrow r(t) \leq s(t)
\end{aligned}$$

Because $r(t)$, $s(t)$, and $s(t) + 1$ are all non-negative for all values of t, in each case a new variable τ can be introduced so that

$$\begin{aligned}
&r(t) \leq \tau \leq s(t) + 1 \leftrightarrow r(t)/\tau \leq 1 \text{ and } 1 \leq s(t)/\tau + 1/\tau \\
&r(t) \geq \tau \geq s(t) + 1 \leftrightarrow r(t)/\tau \leq 1 \text{ and } 1 \leq s(t)/\tau \\
&r(t) \leq \tau \leq s(t) \leftrightarrow r(t)/\tau \leq 1 \text{ and } 1 \leq s(t)/\tau
\end{aligned}$$

These transformations on a signomial produce a posynomial program with reversed constraints:

$$\text{Minimize}\, g_0(t) \tag{30}$$

$$\text{subject to}\, g_k(t) \leq 1 \quad k = 1, \ldots, p \tag{31}$$

$$g_k(t) \geq 1 \quad k = p + 1, \ldots, r \tag{32}$$

$$t > 0 \tag{33}$$

where the functions $g_k(t)$ are posynomials defined by Equation (17). The difference in this formulation is the reversed constraints [Equation (32)]. Although the constraints [Equation (31)] define a convex region, the reversed constraints make the region nonconvex. The dual to the primal program defined by Equations (30)–(33) is given by:

$$\text{Maximize } v(\delta) = \prod_{i\in[0]}(c_i/\delta_i)^{\delta_i}\prod_{k=1}^{p}\prod_{i\in[k]}(c_i\lambda_k/\delta_i)^{\delta_k}\prod_{k=p+1}^{r}\prod_{i\in[k]}(c_i\lambda_k/\delta_i)^{-\delta_i} \tag{34}$$

subject to same constraints as for posynomials, namely

$$\sum_{i\in[0]}\delta_i = 1 \tag{35}$$

$$\lambda_k = \sum_{i\in[k]}\delta_i,\ k = 1,\ldots,r \tag{36}$$

$$\sum_{k=1}^{p}\sum_{i\in[k]}a_{ij}\delta_i - \sum_{k=p+1}^{r}\sum_{i\epsilon[k]}a_{ij}\delta_i = 0,\quad j = 1,\ldots,m \tag{37}$$

$$\delta_i \geq 0,\quad i = 1,\ldots,n \tag{38}$$

We note that $\log v(\delta)$ is concave in the variables associated with the regular constraints $k = 1,\ldots,p$ and the objective function. However, it is convex in the variables associated with the reversed constraints $k = p+1,\ldots,r$. In the posynomial case, the dual program required finding the maximum of a concave function subject to linear constraints. Here we are looking for the maximum of several saddle points of a concave-convex function subject to linear constraints. If the program has degree of difficulty of zero, then it is possible that a solution can be found readily but, in the general case, the program is very difficult to solve. To cope with this problem, Duffin et al. (3) apply the arithmetic–harmonic mean inequality to convert the reversed constraints into regular constraints. This approximation has the advantage that the linear constraints in the dual are the same for all such approximations. As such, the sequence of all regular posynomial programs generated is monotone and decreases in the value of the objective function. Thus one can only improve any feasible solution.

The arithmetic–harmonic inequality, given parameters $\alpha_i,\quad i\in[k]$, positive and $\sum_{i\in[k]}\alpha_i = 1$, is

$$\left(\sum_{i\in[k]}u_i\right)^{-1} \leq \prod_{i\in[k]}(\alpha_i/u_i)^{\alpha_i} \leq \sum_{i\in[k]}\alpha_i^2/u_i \tag{39}$$

with equality holding when

$$\alpha_i = u_i/\left(\sum_{i\in[k]}u_i\right) \tag{40}$$

The means in Equation (30) are the harmonic, geometric and arithmetic respectively. Consider any reversed constraint

$$g_k(t) \geq 1 \quad or \quad (g_k(t))^{-1} \leq 1$$

We set $g_k(t,\alpha) = \sum_{i\in[k]}\alpha_i^2/u_i$, which is a posynomial for fixed α. We note that $g_k(t,\alpha) \leq 1$ implies that $g_k(t) \geq 1$. Using Equation (40) for any t such that $g_k(t) \geq 1$, one can generate an α such that $g_k(t,\alpha) \leq 1$. Hence, by use of the arithmetic–harmonic inequality, we may reduce a posynomial program with reversed constraints to a regular posynomial program parameterized with respect to α, viz.

$$\text{Minimize}\quad g_0(t) \tag{41}$$

$$\text{subject to}\quad g_k(t) \leq 1,\quad k = 1,\ldots,p \tag{42}$$

$$g_k(t,\alpha) \leq 1,\quad k = p+1,\ldots,r \tag{43}$$

$$t > 0 \tag{44}$$

To obtain a monotone-decreasing sequence of programs, we solve the program above for a particular α and then reset the αs using Equation (40).

Example. We return to the multiproduct economic lot size problem given in the section on Constrained polysynomial programming. Here, we add consideration of the distribution effort. Sales are now a function of the allocation of effort to distribution, and we seek to maximize profit. The problem becomes:

$$\begin{array}{ll} \text{Maximize} & \sum_{i=1}^{N}((p_i - c_i)a_iD_i^{d_i} - D_i - K_ia_iD_i^{d_i}/Q_i - h_iQ_i/3) \\ \text{subject to} & \sum_{i=1}^{N}b_iQ_i \leq W \\ Q_i \geq 0, & D_i \geq 0,\quad \forall i \end{array}$$

Here $p_i - c_i$ is the sales price minus the cost for product i, $a_iD_i^{d_i}$ are the sales generated by a distribution effort of D_i and a_i and d_i are parameters. All other symbols have been defined previously. This model is a signomial program and is equivalent to the following program:

$$\begin{array}{ll} \text{Minimize} & 1/P \\ \text{subject to} & \sum_{i=1}^{N}b_iQ_i/W \leq 1 \end{array}$$

$$\begin{array}{l} \sum_{i=1}^{N}(p_i - c_i)a_iD_i^{d_i} - D_i - K_ia_iD_i^{d_i}/Q_i - h_iQ_i/2 \geq P \\ Q_i \geq 0,\quad D_i \geq 0,\quad \forall i \end{array} \tag{45}$$

P is a new variable that indicates profit. Taking the negative terms in Equation (45) to the right hand side and inserting a new variable R between the sides of the inequality as in the theory, we obtain the program:

$$\begin{aligned} &\text{Minimize} \quad 1/P \\ &\text{subject to} \quad \sum_{i=1}^{N} b_i Q_i/W \le 1 \end{aligned}$$

$$\begin{aligned} &P/R + \sum_i (D_i/R + K_i a_i D_i^{d_i}/(Q_i R) + h_i Q_i/(2R)) \le 1 \\ &\sum_i (p_i - c_i) a_i D_i^{d_i}/R \ge 1 \qquad (46) \\ &Q_i \ge 0, \quad D_i \ge 0 \quad \forall i \end{aligned}$$

Our original signomial program is now in the form of a posynomial program with one reversed constraint [Equation (46)]. This equation may be harmonized to become

$$\sum_i \alpha_i^2 R/((p_i - c_i) a_i D_i^{d_i}) \le 1 \qquad (47)$$

where the $\alpha_i > 0$ (and $\sum_i \alpha_i = 1$) are parameters to be varied. To find a local solution, we take a feasible set of distribution efforts D_i to evaluate α_i which is given by [see equation (40)].

$$\alpha_i = (p_i - c_i) a_i D_i^{d_i} \Bigg/ \left(\sum_i (p_i - c_i) a_i D_i^{d_i} \right), \quad \forall i \qquad (48)$$

We now solve the posynomial program with Equation (46) replaced by Equation (47). With the resulting value of D_i obtained, we reset α_i from Equation (48) and resolve the program with this new value of α_i. We repeat this procedure until no significant change is observe in the solution.

GENERALIZED GEOMETRIC PROGRAMMING

Generalized geometric programming deals with the analysis of convex mathematical programs. Commonly these appear in the form as follows:

$$\text{Minimize}\, g_0(z) \qquad (49)$$

$$\text{subject to}\, g_i(z) \le 0, \quad i \in I \qquad (50)$$

$$z \in C \qquad (51)$$

where C is a convex set, and $g_i, i \in \{0\} \cup I$ are convex functions. In the generalized geometric programming approach, we first *separate* the arguments of the constraints and objective functions. We introduce the following notation:

$$\begin{aligned} z &= x^0 = x^i, \quad \forall i \in I \\ x &= x^0 \times \underset{i \in I}{x^i} \in X. \end{aligned}$$

$$\begin{aligned} &\chi = \{x | x^0 = x^i, \, \forall i \in I\} \quad \text{a subspace of} \quad X \\ &C_i = C, \quad \forall i \in \{0\} \cup I \end{aligned}$$

The program defined by Equations (49)–(51) may now be rewritten as:

$$\text{Minimize} \quad g_0(x^0) \qquad (52)$$

$$\text{subject to} \quad g_i(x^i) \le 0 \quad i \in I \qquad (53)$$

$$x^i \in C_i, \quad i \in \{0\} \cup I \qquad (54)$$

$$x \in \chi \qquad (55)$$

This formulation *separates* the constraints and the objective function excepting for the subspace condition which ties the problem together. The subspace captures the *linearity* of the problem.

At this stage we require further ideas relating to *convexity*. Recall from the section on convexity the definition of an epigraph to a pair $[g, C]$ of a convex function g defined on a convex set C. Each nonvertical hyperplane that supports the epigraph of a convex function g at a boundary point $[x', g(x')]$ produces a subgradient of g at x' , i.e., a vector $y \in R^*$ with

$$g(x') + \langle y, x - x' \rangle \le g(x), \quad \forall x \in C \qquad (56)$$

The subgradient set $[\partial g(x')]$ that consists of all such vectors y is generally a closed convex subset of E_n, which contains only a single vector iff g is differentiable at x'. In this case the single vector is the usual gradient vector $\nabla g(x')$.

A convex function $[g, C]$ is said to be closed if its epigraph is a closed set. We shall assume that all functions are closed.

The conjugate transform $[h, D]$ of an arbitrary convex function $[g, C]$ is defined by:

$$h(y) = \sup_{x \in C} (\langle y, x \rangle - g(x)) \qquad (57)$$

and

$$D = \{y | \sup_{x \in C} (\langle y, x \rangle - g(x)) < +\infty\} \qquad (58)$$

We note that $h(y) = \langle y, x' \rangle - g(x')$ for each $y \in \partial g(x')$.

By construction $[h, D]$ is a closed convex function defined on a convex set. Another consequence of the conjugate transform is the conjugate inequality which states that

$$g(x) + h(y) \ge \langle x, y \rangle \qquad (59)$$

for $x \in C$ and $y \in D$, with equality iff $y \in \partial g(x)$ or equivalently $x \in \partial h(y)$. To handle constraints, we require the conjugate transform of a particular function $[0, g(x) \le 0, x \in C]$. This transform is related to the transform of $[g, C]$ and is called the positive homogeneous extension of $[h^+, D^+]$. This extension is denoted by $[h', D']$ where

$$h^+(y,\lambda) = \begin{cases} \lambda h(y/\lambda), & \lambda > 0 \\ \sup\limits_{x\in C} \langle y,x\rangle, & \lambda = 0 \end{cases} \tag{60}$$

and

$$D^+ = \{(y,\lambda) | y/\lambda \in D, \lambda > 0\} \cup \{(y,0) | \sup_{x\in C} \langle y,x\rangle < \infty\} \tag{61}$$

The conjugate inequality in this case is given by

$$0 + h^+(y,\lambda) \geq \langle x,y\rangle \tag{62}$$

for $(y,\lambda)\in D^+$ and $x \in C$ with equality iff $y \in \lambda\partial g(x)$ or $x \in \partial h^+(y,\lambda)$.

We now apply these ideas of conjugate transform theory to the program defined by Equations (52)–(55). This program is the primal mathematical program. For the objective function, we have from Equation (59) that

$$g_0(x^0) + h_0(y^0) \geq \langle x^0, y^0\rangle \tag{63}$$

and for each constraint, from Equation (62), we have that

$$0 + h_i^+(y^i,\lambda_i) \geq \langle x^i, y^i\rangle, \quad i \in I \tag{64}$$

Adding the inequalities in Equations (63) and (64), we have that

$$g_0(x^0) + h_0(y^0) + \sum_{i\in I} h_i^+(y^i,\lambda_i) \geq \langle x,y\rangle \tag{65}$$

where $x = x^0 \times_{i\in I} x^i$ and $y = y^0 \times_{i\in I} y^i$. By restricting $x \in \chi$ and $y \in \chi^\perp$, Equation (65) becomes

$$g_0(x^0) + h_0(y^0) + \sum_{i\in I} h_i^+(y^i,\lambda_i) \geq 0 \tag{66}$$

with equality when

$$y^0 \in \partial g_0(x^0) \quad \text{or} \quad x^0 \in \partial h_0(y^0) \tag{67}$$

and

$$y^i \in \lambda_i \partial g_i(x^i) \quad \text{or} \quad x^i \in \partial h_i^+(y^i,\lambda_i), \quad i \in I \tag{68}$$

Hence the geometric programming dual to the primal program [equations (52)–(55)] is given by:

$$\text{Minimize} \quad h_0(y^0) + \sum_{i\in I} h_i^+(y^i,\lambda_i) \tag{69}$$

$$\underset{y\in\chi^\perp}{\text{subject to}} \quad y^0 \in D_0, \quad (y^i,\lambda_i) \in D_i^+, \quad i \in I \tag{70}$$

The dual program can have certain advantages over the original primal program:

- The nonlinear constraints are incorporated into the objective function.
- If χ is of dimension n and χ is of dimension m, then $\chi^\perp$, the orthogonal complement of χ, is of dimension $n - m$. With the possibility of $n - m$ being much smaller than m and the elimination of explicit nonlinear constraints, the dual may turn out to be a much simpler problem to solve.
- The fact that the primal and dual objectives sum to zero at optimality, that is $g_0(x^0) + h_0(y^0) + \sum_{i\in I} h_i^+(y^i,\lambda_i) = 0$ provides a good stopping criterion for an algorithm. The conditions for optimality are

$$\begin{array}{lll} x^0 \in C_0 & & y^0 \in D_0 \\ g_i(x_i) \leq 0,\ x^i \in C_i & & (y_i,\lambda_i) \in D_i^+,\ i \in I \\ x \in \chi & & y \in \chi^\perp \\ x^0 \in \partial h_0(y^0) & \text{or} & y^0 \in \partial g_0(x^0) \\ x^i \in \partial h_i(y^i,\lambda_i) & \text{or} & y^i \in \lambda_i \partial g_i(x^i),\ i \in I \end{array} \tag{71}$$

From these conditions, one may calculate the optimal solution for one problem from the optimal solution of the other.

It is important to note the roles played by the four concepts of linearity, separability, convexity, and duality in generalized geometric programming theory. Linearity may occur naturally in the problem (e.g., as linear constraints), or it may be induced by the need to separate the variables as in the program at the beginning of this section. Any linearity is usually captured conveniently in the subspace condition. Separability is necessary to facilitate a simple computation of conjugate transforms. Convexity (and closure) guarantees that there is no duality gap, that is the primal and dual objectives sum to zero at optimality. Duality is the goal of generalized geometric programming theory.

Example. A well-known problem in inventory control is to select a set $\{x_i; t = 1, \ldots, T\}$ of production levels to minimize, over a planning horizon of length T, the sum of production and holding costs while meeting demand. Formally the problem may be posed as follows:

$$\text{Minimize} \sum_{t=1}^{T} c(x_t) + h_t y_t \tag{72}$$

subject to the inventory balance dynamics

$$\begin{aligned} & y_1 = x_1 - d_1 \\ & y_t - y_{t-1} = x_t - d_t,\ t = 2, \ldots, T-1 \\ & -y_{T-1} = x_T - d_T \end{aligned} \tag{73}$$

and the non-negativity constraints

$$\begin{aligned} & y_t \geq 0, \quad t = 1, \ldots, T \\ & x_t \geq 0, \quad t = 1, \ldots, T \end{aligned} \tag{74}$$

Here y_t denotes the inventory level in period t, d_t is the demand in period t, $c(x_t)$ is the production cost (assumed convex and strictly monotonically increasing) and h_t is the holding cost per unit in period t.

To invoke the theory of generalized geometric programming, we need to put the constraints from Equation (73) into a subspace. Hence we introduce a new variable $\alpha_t, t = 1, \ldots, T$, and restrict it to a one point domain $\{d_t\}$, $t = 1, \ldots, T$. This variable is then associated with an additive component of the objective function, which is identically zero. Hence, we obtain a subspace condition

$$\begin{aligned} & y_1 - x_1 + \alpha_1 = 0 \\ & y_t - y_{t-1} - x_t + \alpha_t = 0, \quad t = 2, \ldots, T-1 \\ & -y_{T-1} - x_T + \alpha_T = 0 \end{aligned} \tag{75}$$

It is convenient to treat the non-negativity constraints $x_t \geq 0, y_t \geq 0, t = 1, \ldots, T$, in an implicit way (i.e., $C_0 = \{x_t | x_t \geq 0, y_t \geq 0, d_t = \alpha_t, t = 1, \ldots, T\}$). Our problem is now in a form that is directly suitable for application of the theory. We note that in this problem no explicit constraints are evident. The dual objective is given by

$$\sup_{\substack{x_t, y_t, \alpha_t \\ x_t \geq 0 \\ y_t \geq 0}} \sum_t (x_t u_t + y_t v_t + \alpha_t \beta_t - c(x_t) - h_t y_t)$$

$$\begin{aligned} &= \sup_{x_t \geq 0} \sum_t (x_t u_t - c(x_t)) + \sup_{\alpha_t} \sum_t \alpha_t \beta_t + \sup_{y_t \geq 0} \sum_t (y_t v_t - h_t y_t) \\ &= \sum_t c^*(u_t) + d_t \beta_t \end{aligned}$$

where

$$c^*(u_t) = \sup_{\substack{x_t \geq 0 \\ u_t \in \partial c(x_t)}} x_t u_t - c(x_t)$$

and

$$v_t \leq h_t$$

Usually $c(x_t)$ is a quadratic function and $c^*(u_t)$ is readily calculated. Furthermore we require the orthogonal complement to the subspace defined by Equation (61), i.e.

$$\{(u_t, v_t, \beta_t) | \sum_t (u_t x_t + v_t y_t + \beta_t \alpha_t) = 0, \quad \forall x_t, y_t, \alpha_t\}$$

satisfying Equation (75). A straightforward calculation shows that

$$\begin{aligned} v_t &= p_t - p_{t+1}, & t &= 1, \ldots, T-1 \\ u_t &= -p_t, & t &= 1, \ldots, T \\ \beta_t &= p_t, & t &= 1, \ldots, T \end{aligned}$$

Hence the dual problem is

$$\text{Minimize} \quad \sum_{t=1}^{T} c^*(-p_t) + d_t p_t$$

$$\text{subject to} \quad p_t - p_{t+1} - h_t \leq 0, \quad t = 1, \ldots, T-1$$

For constant h_t, we have a minimization over a monotonically increasing set of decision variables.

THE ANALYSIS OF POSYNOMIAL PROGRAMMING BY GENERALIZED GEOMETRIC PROGRAMMING

Recall that in the preceding section we used the convexity of the functions of a convex mathematical program by applying separability and then using linearity and duality to derive a potentially simpler problem. To bring this structure out in posynomial programming, we consider the following convex function:

$$\sum_{i \in [k]} \delta_i \log \delta_i / c_i \tag{76}$$

defined on

$$\delta_i \geq 0, \quad i \in [k] \quad \text{and} \quad \sum_{i \in [k]} \delta_i = 1 \tag{77}$$

Here the parameters $c_i > 0$, $i \in [k]$. The conjugate transform of the function defined by Equation (76) taken over the set defined by Equation (77) may be shown to be

$$\log \sum_{i \in [k]} c_i \exp(z_i) \tag{78}$$

Hence the conjugate inequality from Equation (59) is:

$$\sum_{i \in [k]} \delta_i \log \delta_i / c_i + \log \left(\sum_{i \in [k]} c_i \exp(z_i) \right) \geq \sum_{i \in [k]} \delta_i z_i \tag{79}$$

with equality when

$$\delta_i = c_i \exp(z_i) / \left(\sum_{i \in [k]} c_i \exp(z_i) \right) \tag{80}$$

To handle constraints of the form

$$\log \sum_{i \in [k]} c_i \exp(z_i) \leq 0 \tag{81}$$

we require the positive homogeneous extension of function [Equation (76)] defined over the set [Equation (77)], which is given by

$$\sum_{i \in [k]} \delta_i \log(\delta_i/(\lambda_k c_i)) \tag{82}$$

defined for

$$\lambda_i = \sum_{i \in [k]} \delta_i,\ \delta_i \geq 0,\ i \in [k] \tag{83}$$

To relate functions of the form in Equation (64) to posynomial geometric programs discussed in Section 3 we set

$$x_j = \log t_j,\ \forall j \tag{84}$$

and

$$z_i = \sum_{j=1}^{m} a_{ij} x_j, \quad \forall i \tag{85}$$

in Equations (14) and (15). Equation (85) is the subspace condition in the generalized theory. Making the above substitution and taking logs, our posynomial program is as follows:

$$\begin{aligned} &\text{Minimize} \quad \log \sum_{i \in [0]} c_i \exp(z_i) \\ &\text{subject to} \quad \log \sum_{i \in [k]} c_i \exp(z_i) \leq 0, \quad i = 1, \ldots, p \\ &z \in X = \{z | z_i = \sum_{j=1}^{m} a_{ij} x_j, j = 1, \ldots, n\} \end{aligned}$$

In the above form, the theory of generalised geometric programming may be invoked since we have already calculated the relevant conjugate transforms [see Equations (76), (77), (82) and (83)]. Hence, using Equations (69), (70), and (71), the dual to a posynomial program is given by

$$\begin{aligned} &\text{Minimize} \quad \sum_{i=1}^{n} \delta_i \log \delta_i / c_i - \sum_{k=1}^{p} \lambda_k \log \lambda_k \\ &\text{subject to} \quad \sum_{i \in [0]} \delta_i = 1 \\ &\delta_i \geq 0,\ \forall i \\ &\lambda_k = \sum_{i \in [k]} \delta_i,\ k = 1, \ldots, p \\ &\delta \in X^{\perp} = \{\delta | \sum_{i=1}^{n} a_{ij} \delta_i = 0, j = 1, \ldots, m\} \end{aligned}$$

The above objective function is equivalent to maximizing

$$\prod_{i=1}^{n} (c_i/\delta_i)^{\delta_i} \prod_{k=1}^{p} \lambda_k^{\lambda_k}$$

which is the original form of the dual objective function for posynomial programming [see Equations (25)–(28)]. Furthermore, it is straightforward to obtain the optimality conditions between the primal and dual variables, Equation (29) from the general optimality conditions in the section on Generalized geometric programming.

APPLICATIONS

In the previous sections, we have shown the development of the theory of geometric programming. Included in the references are various books and papers that describe applications of geometric programming. Here, we provide a couple of relevant applications and then a survey of geometric programming applications.

Transformer Design

Transformer design is a major application of geometric programming. Geometric programming produces a minimum cost design that can be quickly redesigned when costs or specifications of materials change. Consider the design of a simple interlocking coil and core transformer. We wish to convert a primary electric power with voltage V_p and current J_p into a secondary electric power of voltage V_s and current J_s. To design a transformer with such specifications, we use fundamental relationships of electricity and magnetism. Faraday's Law of Induction is

$$\oint E \cdot ds = 10^{-8} \dot{B} A_{Fc}$$

The electromotive force equals the ratio of voltage over the number of turns of the copper wire for the primary and secondary circuits. We have

$$\oint E \cdot ds = V_p / N_p$$

and

$$\oint E \cdot ds = V_s / N_s$$

Thus $V_p/N_p = V_s/N_s$.

Ampere's Law for an electric current with the integral taken around both the primary and secondary coils is

$$\oint H \cdot ds = 0.4\pi J_{\text{total}}$$

Because J_{total} is approximately zero, we have

$$J_p N_p = J_s N_s$$

Combining this result with that of voltage and turns we have primary power approximately equal to secondary power.

$$J_pV_p = J_sV_s$$

With this knowledge of electromagnetics, we are ready to construct the geometric program for the transformer design. We must have a large enough cross-sectional area of the iron core to be able to generate a sufficient electromotive force, which means that

$$10^{-8}\dot{B}A_{\text{Fe}} \geq V_p/N_p$$

With the assumption that there is a equal split between the primary and secondary coils the requirement for copper must satisfy the following inequality:

$$iA_{\text{Cu}}/2 \geq N_pJ_p$$

Replace $\dot{B}$ by $\omega B/2^{t/2}$ with B being the magnetic flux amplitude and ω the circular frequency. Combining the above two inequalities, we can eliminate N_p.

$$2^{-3/2} \cdot 10^{-8}\omega BiA_{\text{Fe}}A_{\text{Cu}} \geq J_pV_p$$

We must also limit magnetic loss and ohmic loss to have a functioning transformer.

$$D\omega B^{3.5}V_{\text{Fe}} \leq M_l$$

The magnetic loss constraint is approximate for the level of 20,000 gauss.

$$\rho i^2V_{\text{Cu}} \leq O_l$$

The cross-sectional areas and volumes can be related to the transformer dimensions by using the mean iron length L_{Fe} and mean copper length L_{Cu}, as well as the dimensions of the iron core window and the copper coil window.

$$\begin{aligned}
&A_{\text{Fe}} = a'\dot{b}' \\
&A_{\text{Cu}} = ab \\
&2a + 2b + 2a' + \Delta \leq L_{\text{Fe}} \\
&2a' + 2b' + \pi a/2 + \Delta' \leq L_{\text{Cu}} \\
&V_{\text{Fe}} = L_{\text{Fe}}A_{\text{Fe}} \\
&V_{\text{Cu}} = L_{\text{Cu}}A_{\text{Cu}}
\end{aligned}$$

Putting this all together into a geometric program we will minimize the cost of materials subject to design constraints and eliminate variables to simplify the formulation.

$$\text{Min}\, C_{\text{Cu}}abL_{\text{Cu}} + C_{\text{Fe}}a'b'L_{\text{Fe}}$$

subject to

$$\begin{aligned}
&D\omega a'b'B^{3.5}L_{\text{Fe}}/M_l \leq 1 \\
&\rho i^2abL_{\text{Cu}}/O_t \leq 1 \\
&J_pV_p2^{3/2} \cdot 10^8\omega^{-1}B^{-1}i^{-1}(a'b')^{-1}a^{-1}b^{-1} \leq 1 \\
&2a/L_{\text{Fe}} + 2b/L_{\text{Fe}} + 2a'/L_{\text{Fe}} + \Delta/L_{\text{Fe}} \leq 1 \\
&2a'/L_{\text{Cu}} + 2b'/L_{\text{Cu}} + \pi a/2L_{\text{Cu}} + \Delta'/L_{\text{Cu}} \leq 1
\end{aligned}$$

Parameter	Definition	Units
Vp	Primary voltage	Volts
Jp	Primary current	Ampres
Vs	Secondary voltage	Volts
Js	Secondary current	Ampres
J_{total}	Primary and secondary current	Ampres
H	Magnetomotive force	Oersteds
M_l	Magnetic loss	Gauss
D	Magnetic loss factor	
O_l	Ohmic loss	Ohms
ω	Circular frequency	Integer
ρ	Resistivity	Ohm cm
C_{Cu}	Copper cost per cm^3	$
C_{Fe}	Iron cost per cm^3	$
Δ	Clearance for Iron	Cm
Δ'	Clearance for Copper	Cm

Tables 1 and 2 show of parameters and variables. Eight design variables remain in the formulation, and there are 13 terms. The degree of difficulty for this problem is 4 (this is the dimension of the dual geometric program). It can be solved using a standard nonlinear programming package or one specifically designed for geometric programming. Because it is a geometric program, we are assured of finding the optimum solution. Additional analysis of this and other transformer designs can be found in Duffin et al.(3).

Information Theory

C. E. Lemke recognized that the dual geometric program was of the form of chemical equilibrium or free energy models in chemistry. Shannon used a similar formulation to model problems in information theory. Here, we will look at the problem of finding the most efficient source for

Variables	Design Variable	Units
E	Electric field	Volts per centimeter
B	Magnetic flux density	Gauss
A_{Fe}	Cross-sectional area of the iron core	Cm^2
Np	Number of primary turns	Integer
Ns	Number of secondary turns	Integer
i	Current current density	Amperes per cm^2
A_{Cu}	Cross-sectional area of copper coils	Cm^2
a	Width of iron core window	Cm
b	Height of iron core window	Cm
L_{Fe}	Mean iron length	Cm
a'	Width of copper coil window	Cm
b'	Height of copper coil window	Cm
L_{Cu}	Mean copper length	Cm
V_{Fe}	Volume of iron	Cm^3
V_{Cu}	Volume of Copper	Cm^3

communications as a dual of a geometric program in infinite dimensions.

The problem is first formulated as an entropy maximation problem with m moment constraints.

$$\operatorname{Max} H(p) = -\int_{-\infty}^{+\infty} p(x) \ln p(x) dx$$

subject to

$$\int_{-\infty}^{+\infty} p(x) dx = 1$$
$$p(x) \geq 0$$
$$\int_{-\infty}^{+\infty} f_i(x) p(x) dx = \alpha_i,\ i = 1, \ldots, m$$

Here, we are optimizing over the probability distribution p.

The dual of the above problem is the following geometric program:

$$\operatorname{Min} G(q) = \ln \int_{-\infty}^{+\infty} \exp[q(x)] dx$$

subject to

$$q(x) = -\sum_i z_i(f_i(x) - \alpha_i)$$

With the z_i are m scalar variables. This mathematical program can be reduced to an unconstrained finite dimensional optimization problem:

$$\operatorname{Min} G(z) = \ln \int_{-\infty}^{+\infty} \exp[-\sum_i z_i(f_i(x) - \alpha_i)] dx$$

Because it is a geometric program, we know that the global optimum can be found. At optimality the we can recognize that the optimal primal distribution will take the form:

$$p(x) = \frac{\exp[-\sum_i z_i(f_i(x) - \alpha_i)]}{\int_{-\infty}^{+\infty} \exp[-\sum_i z_i(f_i(x') - \alpha_i)] dx'}$$

Thus, we know that the solution to this communications problem comes from the exponential family.

Survey of Geometric Programming Applications

Engineering Design. The reason for geometric programming was Zener's interest in engineering design problems. Electrical engineering design appears in both books (1,4) in the form of transformer design. In fact, the method can be used for other devices that involve the use of induction coils. More recently, it has been used in circuit design and manufacture (5,6). The second example was mechanical engineering. Duffin et al. (1) used geometric programming to design an ocean thermal energy conversion system. Power turbines and cooling towers can be designed using geometric programming (7). Control system design is a geometric program (8). Steel plate girders can be designed by geometric programming as well (9).

In Zener's book (4), the third application of geometric programming was civil engineering (bridges and buildings). Also designed are floor and roof systems, frames structures, hydraulic networks and pipelines, and wastewater treatment (10).

Lemke identified the dual of a geometric program as having the same form as chemical equilibrium. The consequence of this comment is included in the textbook (3) as an appendix. Wilde and his students (11) introduced geometric programming to chemical engineering. Chemical engineering applications appear in the Beightler and Phillips textbook (12). Geometric programming has been used in ethylene plant optimization. Chemical reactors, complex chemical equilibria, and mineral processing can also benefit from geometric programming.

Passy (13) was the first person to apply geometric programming to industrial engineering problems [see also Jung et al. (14)]. Machining economics (tool life) has been modelled using geometric programming (15). It is used in bottleneck scheduling (16). Replacement (reliability) problems can be solved by geometric programming (17). Optimal tolerancing (18) can be solved as a geometric program. Generalized geometric programming can solve location problems (19).

Geometric programming has been applied to nuclear engineering, naval architecture, aeronautical engineering, and agricultural engineering. A collection of papers on optimal design (using geometric programming) was published in 1973 (20).

Business. In 1974, Balachandran and Gensch (21) first analyzed the marketing mix problem using geometric programming. Capital budgeting benefits from geometric programming analysis (22,23). Accounting estimates and valuations are found by entropy (24,25). Cost benefit analysis can be analyzed using generalized geometric programming (26). A variety of applications from the management sciences, using geometric programming, are presented in a book (27).

Transportation and Regional Planning. Jefferson (1972) was the first to apply geometric programming to transportation planning and trip distribution (28). The application of geometric programming to entropy models for trip distribution and regional planning was particularly worthwhile. See for example, Refs. 29 and 30.

Information and Communications. In 1977, Scott and Jefferson (31) were the first to apply geometric programming to information theory. Modern communications problems can be modeled as geometric programs (32,33).

Stochastic Models and Statistics. In 1966, Avriel (34) was the first to work on stochastic geometric programming.

Stochastic geometric programming has been applied to industrial engineering and finance (22, 35). Probability can be analyzed by geometric programming (36). Geometric programming aids in estimation and sampling (25).

Curve Fitting. Peterson and Ecker (37) first applied geometric programming concepts to l_p-approximation (37). Geometric programming provides an effective framework for curve fitting with spline functions (38,39).

ALGORITHMS

Initially, the geometric programs were small, and the dual program was only a single point or up to a few dimensions. Problems could be solved by hand or with a simple program. As geometric programming grew in popularity, people wanted to solve larger problems and problems modeled as signomials. By the mid-1970s, the computer code of choice written by Dembo et al. (40) used cutting planes to approximate the posynomials with a single term (called a monomial) and linear programming. It could solve moderate-sized problems efficiently. For signomials, it could improve on a starting solution but there was no guarantee on the solution being globally optimal.

In the 1990s, interior point methods were applied with success to geometric programming. Nesterov and Nemirovsky (41) developed an interior point method for convex programs that could solve geometric programs. Separately Zhu et al. (42) provided a good stable interior point algorithm built on the work of Fang et al. (43). Next Kortanek et al. (44) developed an infeasible interior point algorithm that searches for the Karush-Kuhn-tucker point for the geometric program and that makes use of the sparse structure of the logarithmic dual objective function (this is the form described in the section entitled "The analysis of posynomial programming by generalized geometric programming"). Anderson and Ye (45) have also developed a solution method for large-scale geometric programs with a single-phase interior point method.

The research into algorithms for geometric programs has taught us that geometric programs are nicely structured problems, and if one makes the transformations used in the section entitled "The analysis of posynomial programming by generalised geometric programming", they can be solved easily. If one does not have access to special geometric programming packages, then a general nonlinear programming package can solve moderate-sized problems. These packages are available on most computers by simply using the nonlinear feature of your spreadsheet program on your. AMPL personal computer provides an excelent modeling language and a GRG nonlinear programming package to solve medium-sized problems. MOSEK can be interfaced with AMPL to allow one to work on large geometric programs.

CONCLUSIONS

Geometric programming has evolved from the study of posynomial programs to signomial programs and, through generalized geometric programming, to convex programs. The areas of application are ever growing (over 500 publications use geometric programming across its 40-year history). The reason for its popularity is that it is fairly easy to formulate many problems as geometric programs because it advantageously captures the inherent structure found in real problems. Many programs so formulated can be converted into convex programs. They can now be solved easily using special purpose algorithms or by generally available nonlinear programming packages, even if they are large. Geometric programming provides a dual that can help in understanding the problem and in the sensitivity analysis of the solution. Thus geometric programming remains an important method in optimization. See the Further Reading list at the end of the chapter for more information.

BIBLIOGRAPHY

1. R. J. Duffin, E. L. Peterson, and C. Zener, *Geometric Programming*, New York: Wiley, 1967.
2. E. L. Peterson, Geometric programming, *SIAM Review*, **18**: 1–52, 1976.
3. U. Passy and D. J. Wilde, Generalized polynomial optimization, *SIAM Appl. Math*, **15**: 1344–1356, 1987.
4. C. Zener, *Engineering Design by Geometric Programming*, New York: Wiley, 1971.
5. M. Chiang, C. W. Tan, D. P. Palomar, D.O'Neill, and D. Julian, power control by geometric programming, *IEEE Trans. Wireless Communicat.*, **6**: 2640–2651, 2007.
6. S. P.Boyd , S. J. Kim, D. D. Patil, and M. A. Horowitz, Digital circuit optimization via geometric programming, *Operat. Res.*, **53**: 899–932, 2005.
7. J. G. Ecker and R. D. Wiebking, Optimal design of a dry type natural-draft cooling tower by geometric programming, *Jour. Opt. Thy. Appns*, **26**: 305–323, 1978.
8. E. V. Bohn, Optimum design of control-system compensators by geometric programming, *J. Dynamic Syst. Meas. Control-Trans. ASME*, **102**: 106–113, 1980.
9. S. Abuyounes and H. Adeli, Optimization of steel plate girders via general geometric programming, *J. Struct. Mech.*, **14**: 501–524, 1986.
10. J. G. Ecker, A geometric programming model for optimal allocation of stream dissolved oxygen, *Manage. Sci.*, **21**: 658–668, 1975.
11. D. J. Wilde, A review of optimization theory, *Indust. Engineer. Chem.*, **57**: 1965.
12. C. S. Beightler and D. T. Phillips, *Applied Geometric Programming*, New York: Wiley, 1976.
13. U. Passy, Nonlinear assignment problems treated by geometric programming, *Operat Res.*, **19**: 1675–1690, 1971.
14. H. Jung, C. M. Klein, and M. Cerry, Optimal inventory policies for profit maximizing EOQ models, under various cost functions, *Euro. J. Operat. Res.*, **174**: 689–705, 2006.
15. C. H. Scott , T. R. Jefferson, and A. Lee, Stochastic tool management via composite geometric programming, *Optimization*, **36**: 59–74, 1996.
16. G. J. Plenert, Generalized-model for solving constrained bottlenecks, *J. Manufact. Sys.*, **12**: 506–513, 1993.
17. T. C. E. Cheng, Optimal replacement of aging equipment using geometric programming, *Internat. J. of Product. Res.*, **30**: 2151–2158, 1992.

18. T. R. Jefferson and C. H. Scott, Quality tolerancing and conjugate duality, *Ann. Operat. Res.*, **105**: 183–198, 2001.

19. E. L. Peterson and R. E. Wendell, Optimal location by geometric programming, Research. Report, IE, Chicago, IL: Northwestern, 1974.

20. M. Avriel, M. S. Rijckaert, and D. J. Wilde, eds., *Optimization and Design*, Englewood Cliffs, NJ: Prentice-Hall, 1973.

21. V. Balachandran and D. H. Gensch, Solving the marketing mix problem using geometric programming, *Managen. Sci.* **21**: 160–171, 1974.

22. T. R. Jefferson, C. H. Scott, and J. Cozzolino, The analysis of risky portfolios by geometric programming, *Zeitschrift fur Operations Research*, Series A **23**: 207–218, 1979.

23. C. H. Scott , T. R. Jefferson, and P. Kerdvonbundit, Geometric programming duality applied to the resource allocation problem, *INFOR*, **1**: 133–137, 1979.

24. T. R. Jefferson, W. Rodgers, and C. H. Scott, Estimating financial statement Information: an entropy approach, *J. Accoun. Fin.*, **13**: 1–19, 1999.

25. J. R. Rajasekera and M. Yamada, Estimating the firm value distribution function by entropy optimization and geometric programing, *Ann. Operat. Res.*, **105**: 61–75, 2001.

26. C. H. Scott and T. R. Jefferson, A convex dual for quadratic-concave fractional programs, *J. Optimiz. Theory Applicat.*, **91**: 115–122, 1996.

27. P. Hayes, *Mathematical Methods in the Social and Managerial Sciences*, New York: Wiley, 1975.

28. T. R. Jefferson, Geometric programming with an application to transportation planning, Ph.D dissertation, Evanstion, IL: Northwestern University, 1972.

29. J. J. Dinkel, G. A. Kochenberger, and S. N. Wong, Entropy maximization and geometric programming, *Environment and Planning A*, **9**: 419–427, 1977.

30. T. R. Jefferson and C. H. Scott, Geometric programming applied to transportation planning, *Opsearch*, **IS**: 22–34, 1978.

31. C. H. Scott and T. R. Jefferson, A generalisation of geometric programming with an application to information theory, *Informat. Sci.*, **12**: 263–269, 1977.

32. M. Chiang and S. Boyd, Geometric programming duals of channel Capacity and rate distortion, *IEEE Trans. Informat. Theory*, **50**: 245–258, 2004.

33. M. Chiang, C. W. Tan, D. Palomar, D.O'Neill, and D. Julian, Geometric programming for wireless network power control, *Resource Allocation in Next Generation Wireless Networks*, 2005.

34. M. Avriel, Topics in optimization; block search; applied and stochastic geometric programming, Ph. D. Thesis, New York: Stanford University, 1966.

35. C. H. Scott and T. R. Jefferson, Dynamic financial planning: certainty equivalents, stochastic constraints and functional conjugate duality, *IEEE Trans.*, **37**: 931–938, 2005.

36. M. Mazumdar and T. R. Jefferson, Maximum-likelihood estimates for multinomial probabilities via geometric-programming, *Biometrika*, **70**: 257–261, 1983.

37. E. L. Peterson and J. G. Ecker, Geometric programming: duality in quadratic programming and l_p-approximation II: canonical programs, *SIAM J. Appl. Math.* **17**: 317–340, 1969.

38. S. C. Fang, Y. B. Zhao, and J. E. Lavery, Geometric Dual Formulation of the first derivative based C1-smooth univariate cubic L1 spline functions, J. Global Optimiz, In Press.

39. S. C. Fang and W. Zhang, Y. Wang , and J. E. Lavery, Cubic L1 splines on triangulated irregular networks, *Pacific J. Optimizat.*, **2**: 289–317, 2006.

40. M. Avriel, R. Dembo, and U. Passy, Solution of generalised geometric programs, *Int. J. Numerical Methods in Eng.*, **9**: 149–169, 1975.

41. Y. Nesterov and A. Nemirovski, Multi-parameter surfaces of analytic centers and long-step surface-following interior point methods, *Mathemat. Operat. Res. Inst.*, **23**: 38, 1998.

42. K. Zhu, K. O. Kortanek, and S. Huang, Controlled dual perturbations *for* central path trajectories in geometric-programming, *Europ. J. Operat. Res.*, **73**: 524–531, 1994.

43. S. C. Fang, E. L. Peterson, J. R. Rajasekera , Minimum cross-entropy analysis with entropy-type constraints, *J. Computat. Applied Mathemt.*, **39**: 165–178, 1992.

44. K. O. Kortanek, X. Xu, and Y. Ye, An infeasible interior-point algorithm for solving primal and dual geometric programs, *Mathemat. Prog.*, **76**: 155–181, 1997.

45. E. D. Andersen and Y. Y. Ye, A computational study of the homogeneous algorithm for large scale convex optimization, *Computat. Optimiz. Applicat.*, **10**: 243–269, 1998.

FURTHER READING

C. S. Beightler and D. T. Phillips, *Applied Geometric Programming*, New York: John Wiley and Sons, Inc., 1976.

M. Chiang, *Geometric Programming for Communications System*, Hanover, MA: Now Publishers Inc., 2005.

R. J. Duffin, Peterson, and C. Zener, *Geometric Programming*, New York: John Wiley and Sons, Inc., 1967.

S. C. Fang, J. R. Rajasekera, and H.-S. J. Tsao, *Entropy Optimization and Mathematical Programming*, Dordrecht, The Netherlands: Kluwer International, 1997.

S. C. Fang and C. H. Scott, *Annals of Operations Research Special Issue on Geometric Programing, Entropy Optimization and Generalizations*, Vol. 105, 2001.

C. Zener, *Engineering Design by Geometric Programming*, New York: John Wiley and Sons, Inc., 1971.

T. R. Jefferson
C. H. Scott
University of California, Irvine
Irvine, California

G

GRAPH THEORY AND ALGORITHMS

GRAPH THEORY FUNDAMENTALS

A *graph* $G(V,E)$ consists of a set V of *nodes* (or *vertices*) anda set E of pairs of nodes from V, referred to as *edges*. An edge may have associated with it a direction, in which case, the graph is called *directed* (as opposed to *undirected*), or a *weight*, in which case, the graph is called *weighted*. Given an undirected graph $G(V,E)$, two nodes $u,v \in V$ for which an edge $e = (u, v)$ exists in E are said to be *adjacent* and edge e is said to be *incident* on them. The *degree* of a node is the number of edges adjacent to it. A (*simple*) *path* is a sequence of distinct nodes $(a_0, a_1, \ldots, a_k)$ of V such that every two nodes in the sequence are adjacent. A (*simple*) *cycle* is a sequence of nodes $(a_0, a_1, \ldots, a_k, a_0)$ such that $(a_0, a_1, \ldots, a_k)$ is a path and a_k, a_0 are adjacent. A graph is *connected* if there is a path between every pair of nodes. A graph $G'(V',E')$ is a *subgraph* of graph $G(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A *spanning tree* of a connected graph $G(V, E)$ is a subgraph of G that comprises all nodes of G and has no cycles. A graph $G'(V', E')$ is a *subgraph* of graph $G(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. An *independent set* of a graph $G(V, E)$ is a subset $V' \subseteq V$ of nodes such that no two nodes in V' are adjacent in G. A *clique* of a graph $G(V, E)$ is a subset $V' \subseteq V$ of nodes such that any two nodes in V' are adjacent in G. Given a subset $V' \subseteq V$, the *induced subgraph of* $G(V, E)$ by V' is a subgraph $G'(V', E')$, where E' comprises all edges (u, v) in E with $u, v \in V'$.

In a directed graph, each edge (sometimes referred to as *arc*) is an ordered pair of nodes and the graph is denoted by $G(V, A)$. For an edge $(u, v) \in A, v$ is called the *head* and u the *tail* of the edge. The number of edges for which u is a tail is called the *out-degree* of u and the number of edges for which u is a head is called the *in-degree* of u. A (*simple*) *directed path* is a sequence of distinct nodes $(a_0, a_1, \ldots, a_k)$ of V such that (a_i, a_{i+1}), $\forall i,\ 0 \leq i \leq k-1$, is an edge of the graph. A (*simple*) *directed cycle* is a sequence of nodes $(a_0, a_1, \ldots, a_k, a_0)$ such that $(a_0, a_1, \ldots, a_k)$ is a directed path and $(a_k,\ a_0) \in A$. A directed graph is *strongly connected* if there is a directed path between every pair of nodes. A directed graph is *weakly connected* if there is an undirected path between every pair of nodes.

Graphs (1–5) are a very important modeling tool that can be used to model a great variety of problems in areas such as operations research (6–8), architectural level synthesis (9), computer-aided Design (CAD) for very large-scale integration (VLSI) (9,10) and computer and communications networks (11).

In operations research, a graph can be used to model the assignment of workers to tasks, the distribution of goods from warehouses to customers, etc. In architectural level synthesis, graphs are used for the design of the data path and the control unit. The architecture is specified by a graph, a set of resources (such as adders and multipliers), and a set of constraints. Each node corresponds to an operation that must be executed by an appropriate resource, and a directed edge from u to v corresponds to a constraint and indicates that task u must be executed before task v. Automated data path synthesis is a scheduling problem where each task must be executed for some time at an appropriate resource. The goal is to minimize the total time to execute all tasks in the graph subject to the edge constraints. In computer and communications networks, a graph can be used to represent any given interconnection, with nodes representing host computers or routers and edges representing communication links.

In CAD for VLSI, a graph can be used to represent a digital circuit at any abstract level of representation. CAD for VLSI consists of logic synthesis and optimization, verification, analysis and simulation, automated layout, and testing of the manufactured circuits (12). Graphs are used in multilevel logic synthesis and optimization for combinational circuits, and each node represents a function that is used for the implementation of more than one output (9). They model synchronous systems by means of finite state machines and are used extensively in sequential logic optimization (9). Binary decision diagrams are graphical representations of Boolean functions and are used extensively in verification. Algorithms for simulation and timing analysis consider graphs where the nodes typically correspond to gates or flip-flops. Test pattern generation and fault grading algorithms also consider graphs where faults may exist on either the nodes or the edges. In the automated layout process (10), the nodes of a graph may represent modules of logic (partitioning and floor-planning stages), gates or transistors (technology mapping, placement, and compaction stages), or simply pins (routing, pin assignment, and via minimization stages).

There are many special cases of graphs. Some of the most common ones are listed below. A *tree* is a connected graph that contains no cycles. A *rooted tree* is a tree with a distinguished node called a *root*. All nodes of a tree that are distinguished from the root and are adjacent to only one node are called *leaves*. Trees are used extensively in the automated layout process of intergated circuits. The clock distribution network is a rooted tree whose root corresponds to the clock generator circuit and whose leaves are the flip-flops. The power and ground supply networks are rooted trees whose root is either the power or the ground supply mechanism and whose leaves are the transistors in the layout. Many problem formulations in operations research, architectural level synthesis, CAD for VLSI, and networking allow for fast solutions when restricted to trees. For example, the simplified version of the data path scheduling problem where all the tasks are executed on the same type of resource is solvable in polynomial time if the task graph is a rooted tree. This restricted scheduling problem is also common in multiprocessor designs (9).

A *bipartite graph* is a graph with the property that its node set V can be partitioned into two disjoint subsets V_1 and V_2, $V_1 \cup V_2 = v$, such that every edge in E comprises

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

one node from V_1 and one node from V_2. Bipartite graphs appear often in operations research (6–8) and in VLSI CAD applications (9,10).

A *directed acyclic graph* is a directed graph that contains no directed cycles. Directed acyclic graphs can be used to represent combinational circuits in VLSI CAD. Timing analysis algorithms in CAD for VLSI or software timing analysis algorithms of synthesizable hardware description language code operate on directed acyclic graphs.

A *transitive graph* is a directed graph with the property that for any nodes $u, v, w \in V$ for which there exist edges $(u, v), (v, w) \in A$, edge (u, w) also belongs to A. Transitive graphs find applications in resource binding of tasks in architectural level synthesis (9) and in fault collapsing for digital systems testing (12).

A *planar graph* is a graph with the property that its edges can be drawn on the plane so as not to cross each other. Many problems in the automated layout for VLSI are modeled using planar graphs. Many over-the-cell routing algorithms require the identification of circuit lines to be routed on a single layer. Single-layer routing is also desirable for detailed routing of the interconnects within each routing layer (10).

A *cycle graph* is a graph that is obtained by a cycle with chords as follows: For every chord (a,b) of the cycle, there is a node $v_{(a,b)}$ in the cycle graph. There is an edge $(v_{(a,b)}, v_{(c,d)})$ in the cycle graph if and only if the respective chords (a, b) and (c, d) intersect. Cycle graphs find application in VLSI CAD, as a channel with two terminal nets, or a switchbox with two terminal nets can be represented as a cycle graph. Then the problem of finding the maximum number of nets in the channel (or switchbox) that can be routed on the plane amounts to finding a maximum independent set in the respective cycle graph (10).

A *permutation graph* is a special case of a cycle graph. It is based on the notion of a *permutation diagram*. A permutation diagram is simply a sequence of N integers in the range from 1 to N (but not necessarily ordered). Given an ordering, there is a node for every integer in the diagram and there is an edge (u,v) if and only if the integers u,v are not in the correct order in the permutation diagram. A permutation diagram can be used to represent a special case of a *permutable channel* in a VLSI layout, where all nets have two terminals that belong to opposite channel sides. The problem of finding the maximum number of nets in the permutable channel that can be routed on the plane amounts to finding a maximum independent set in the respective permutation graph. This, in turn, amounts to finding the maximum increasing (or decreasing) subsequence of integers in the permutation diagram (10).

Graphs have also been linked with random processes, yielding what is known as *random graphs*. Random graphs consist of a given set of nodes while their edges are added according to some random process. For example, $G(n,p)$ denotes a random graph with n nodes but whose edges are chosen independently with probability p. Typical questions that arise in the study of random graphs concern their probabilistic behavior. For example, given n and p we may want to find the probability that $G(n,p)$ is connected (which has direct application to the reliability of a network whose links fail probabilistically). Or we may want to determine whether there is a threshold on some parameter (such as the average node degree) of the random graph after which the probability of some attribute (such as connectivity, colorability, etc) changes significantly.

ALGORITHMS AND TIME COMPLEXITY

An *algorithm* is an unambiguous description of a finite set of operations for solving a computational problem in a finite amount of time. The set of allowable operations corresponds to the operations supported by a specific computing machine (computer) or a model of that machine.

A *computational problem* comprises a set of parameters that have to satisfy a set of well-defined mathematical constraints. A specific assignment of values to these parameters constitutes an *instance* of the problem. For some computational problems, there is no algorithm as defined above to find a solution. For example, the problem of determining whether an arbitrary computer program terminates in a finite amount of time given a set of input data cannot be solved (it is “undecidable”) (13). For the computational problems for which there does exist an algorithm, the point of concern is how “efficient” is that algorithm. The *efficiency* of an algorithm is primarily defined in terms of how much time the algorithm takes to terminate. (Sometimes, other considerations such as the space requirement in terms of the physical information storage capacity of the computing machine are also taken into account, but in this exposition we concentrate exclusively on time.)

To formally define the efficiency of an algorithm, the following notions are introduced.

The *size* of an instance of a problem is defined as the total number of symbols for the complete specification of the instance under a finite set of symbols and a “succinct” encoding scheme. A “succinct” encoding scheme is considered to be a logarithmic encoding scheme, in contrast to a unary encoding scheme. The time requirement (*time complexity*) of an algorithm is expressed then as a function $f(n)$ of the size n of an instance of the problem and gives the total number of “basic” steps that the algorithm needs to go through to solve that instance. Most of the time, the number of steps is taken with regard to the worst case, although alternative measures like the average number of steps can also be considered. What constitutes a “basic” step is purposely left unspecified, provided that the time the basic step takes to be completed is bounded from above by a *constant*, that is, a value not dependent on the instance. This hides implementation details and machine-dependent timings and provides the required degree of general applicability.

An algorithm with time complexity $f(n)$ is said to be of the *order of* $g(n)$ [denoted as $O(g(n))$], where $g(n)$ is another function, if there is a constant c such that $f(n) \leq c \cdot g(n)$, for all $n \geq 0$. For example, an algorithm for finding the minimum element of a list of size n takes time $O(n)$, an algorithm for finding a given element in a sorted list takes time $O(\log n)$, and algorithms for sorting a list of elements can take time $O(n^2)$, $O(n \log n)$, or $O(n)$ (the latter when additional information about the range of the elements is known). If moreover, there are constants c_L and c_H such that $c_L \cdot g(n) \leq f(n) \leq c_H \cdot g(n)$, for all $n \geq 0$, then $f(n)$ is said to be $\Theta(g(n))$.

The smaller the "order-of" function, the more efficient an algorithm is generally taken to be, but in the analysis of algorithms, the term "efficient" is applied liberally to any algorithm whose "order-of" function is a polynomial $p(n)$. The latter includes time complexities like $O(n \log n)$ or $O(n\sqrt{n})$, which are clearly bounded by a polynomial. Any algorithm with a nonpolynomial time complexity is not considered to be efficient. All nonpolynomial-time algorithms are referred to as *exponential* and include algorithms with such time complexities as $O(2^n)$, $O(n!)$, $O(n^n)$, $O(n^{\log n})$ (the latter is sometimes referred to as *subexponential*). Of course, in practice, for an algorithm of polynomial-time complexity $O(p(n))$ to be actually efficient, the degree of polynomial $p(n)$ as well as the constant of proportionality in the expression $O(p(n))$ should be small. In addition, because of the worst-case nature of the $O()$ formulation, an "exponential" algorithm might exhibit exponential behavior in practice only in rare cases (the latter seems to be the case with the simplex method for linear programming). However, the fact on the one hand that most polynomial-time algorithms for the problems that occur in practice tend indeed to have small polynomial degrees and small constants of proportionality, and on the other that most nonpolynomial algorithms for the problems that occur in practice eventually resort to the trivial approach of exhaustively searching (enumerating) all candidate solutions, justifies the use of the term "efficient" for only the polynomial-time algorithms.

Given a new computational problem to be solved, it is of course desirable to find a polynomial-time algorithm to solve it. The determination of whether such a polynomial-time algorithm actually exists for that problem is a subject of primary importance. To this end, a whole discipline dealing with the classification of the computational problems and their interrelations has been developed.

P, NP, and *NP*-Complete Problems

The classification starts technically with a special class of computational problems known as decision problems. A computational problem is a *decision problem* if its solution can actually take the form of a "yes" or "no" answer. For example, the problem of determining whether a given graph contains a simple cycle that passes through every node is a decision problem (known as the Hamiltonian cycle problem). In contrast, if the graph has weights on the edges and the goal is to find a simple cycle that passes through every node and has a minimum sum of edge weights is not a decision problem but an *optimization* problem (the corresponding problem in the example above is known as the Traveling Salesman problem). An optimization problem (sometimes also referred to as a *combinatorial optimization problem*) seeks to find the best solution, in terms of a well-defined objective function $Q()$, over a set of feasible solutions. Interestingly, every optimization problem has a "decision" version in which the goal of minimizing (or maximizing) the objective function $Q()$ in the optimization problem corresponds to the question of whether a solution exists with $Q() \leq k$ (or $Q() \geq k$) in the decision problem, where k is now an additional input parameter to the decision problem. For example, the decision version of the Traveling Salesman problem is, given a graph and an integer K, to find a simple cycle that passes through every node and whose sum of edge weights is no more than K.

All decision problems that can be solved in polynomial time comprise the so-called class P (for "Polynomial"). Another established class of decision problems is the *NP* class, which consists of all decision problems for which a polynomial-time algorithm can *verify* if a candidate solution (which has polynomial size with respect to the original instance) yields a "yes" or "no" answer. The initials "*NP*" stand for "Non-deterministic Polynomial" in that if a yes-answer exists for an instance of an *NP* problem, that answer can be obtained *nondeterministically* (in effect, "guessed") and then verified in polynomial time. Every problem in class P belongs clearly to *NP*, but the question of whether class *NP* strictly contains P is a famous unresolved problem. It is conjectured that $NP \neq P$, but there has been no actual proof until now. Notice that to simulate the nondeterministic "guess" in the statement above, an obvious deterministic algorithm would have to enumerate all possible cases, which is an exponential-time task. It is in fact the question of whether such an "obvious" algorithm is actually the best one can do that has not been resolved.

Showing that an *NP* decision problem actually belongs to P is equivalent to establishing a polynomial-time algorithm to solve that problem. In the investigation of the relations between problems in P and in *NP*, the notion of *polynomial reducibility* plays a fundamental role. A problem R is said to be *polynomially reducible* to another problem S if the existence of a polynomial-time algorithm for S implies the existence of a polynomial-time algorithm for R. That is, in more practical terms, if the assumed polynomial-time algorithm for problem S is viewed as a subroutine, then an algorithm that solves R by making a polynomially bounded number of calls to that subroutine and taking a polynomial amount of time for some extra work would constitute a polynomial-time algorithm for R.

There is a special class of *NP* problems with the property that if and only if *any one* of those problems could be solved polynomially, then so would *all NP* problems (i.e., *NP* would be equal to P). These *NP* problems are known as NP-*complete*. An *NP*-complete problem is an *NP* problem to which every other *NP* problem reduces polynomially. The first problem to be shown *NP*-complete was the Satisfiability problem (13). This problem concerns the existence of a truth assignment to a given set of Boolean variables so that the conjunction of a given set of disjunctive clauses formed from these variables and their complements becomes true. The proof (given by Stephen Cook in 1971) was done by showing that every *NP* problem reduces polynomially to the Satisfiability problem. After the establishment of the first *NP*-complete case, an extensive and ongoing list of *NP*-complete problems has been established (13).

Some representative NP-complete problems on graphs that occur in various areas in digital design automation, computer networks, and other areas in electrical and computer engineering are listed as follows:

- Longest path: Given a graph $G(V, E)$ and an integer $K \leq |V|$, is there a simple path in G with at least K edges?
- Vertex cover: Given a graph $G(V, E)$ and an integer $K \leq |V|$, is there a subset $V' \subseteq V$ such that $|V'| \leq K$ and for each edge $(u,v) \in E$, at least one of u, v belongs to V'?
- Clique: Given a graph $G(V, E)$ and an integer $K \leq |V|$, is there a clique C among the nodes in V such that $|C| \geq K$?
- Independent set: Given a graph $G(V, E)$ and an integer $K \leq |V|$, is there a subset $V' \subseteq V$ such that $|V'| \geq K$ and no two nodes in V' are joined by an edge in E?
- Feedback node set: Given a directed graph $G(V, A)$ and an integer $K \leq |V|$, is there a subset $V' \subseteq V$ such that $|V'| \leq K$ and every directed cycle in G has at least one node in V'?
- Hamiltonian cycle: Given a graph $G(V, E)$, does G contain a simple cycle that includes all nodes of G?
- Traveling Salesman: Given a weighted graph $G(V, E)$ and a bound K, is there a path that visits each node exactly once and has a total sum of weights no more than K?
- Graph colorability: Given a graph $G(V, E)$ and an integer $K \leq |V|$, is there a "coloring" function $f: V \rightarrow 1, 2, \ldots, K$ such that for every edge $(u,v) \in E$, $f(u) \neq f(v)$?
- Graph bandwidth: Given a graph $G(V,E)$ and an integer $K \leq |V|$, is there a one-to-one function $f: V \rightarrow 1, 2, \ldots, |V|$ such that for every edge $(u,v) \in E$, $|f(u) - f(v)| \leq K$?
- Induced bipartite subgraph: Given a graph $G(V, E)$ and an integer $K \leq |V|$, is there a subset $V' \subseteq V$ such that $|V'| \geq K$ and the subgraph induced by V' is bipartite?
- Planar subgraph: Given a graph $G(V, E)$ and an integer $K \leq |E|$, is there a subset $E' \subseteq E$ such that $|E'| \geq K$ and the subgraph $G'(V,E')$ is planar?
- Steiner tree: Given a weighted graph $G(V, E)$, a subset $V' \subseteq V$, and a positive integer bound B, is there a subgraph of G that is a tree, comprises at least all nodes in V', and has a total sum of weights no more than B?
- Graph partitioning: Given a graph $G(V, E)$ and two positive integers K and J, is there a partition of V into disjoint subsets $V_1, V_2, \ldots, V_m$ such that each subset contains no more than K nodes and the total number of edges that are incident on nodes in two different subsets is no more than J?
- Subgraph isomorphism: Given two graphs $G(V_G, E_G)$ and $H(V_H, E_H)$, is there a subgraph $H'(V_{H'}, E_{H'})$ of H such that G and H' are isomorphic (i.e., there is a one-to-one function $f: V_G \rightarrow V_{H'}$ such that $(u,v) \in E_G$ if and only if $(f(u), f(v)) \in E_{H'}$)?

The interest in showing that a particular problem R is NP-complete lies exactly in the fact that if it finally turns out that NP strictly contains P, then R cannot be solved polynomially (or, from another point of view, if a polynomial-time algorithm happens to be discovered for R, then $NP = P$). The process of showing that a decision problem is NP-complete involves showing that the problem belongs to NP and that some known NP-complete problem reduces polynomially to it. The difficulty of this task lies in the choice of an appropriate NP-complete problem to reduce from as well as in the mechanics of the polynomial reduction. An example of an NP-complete proof is given below.

Theorem 1. The Clique problem is NP-complete

Proof. The problem belongs clearly to NP, because once a clique C of size $|C| \geq K$ has been guessed, the verification can be done in polynomial (actually $O(|C|^2)$) time. The reduction is made from a known NP-complete problem, the 3-Satisfiability problem, or 3SAT (13). The latter problem is a special case of the Satisfiability problem in that each disjunctive clause comprises exactly three literals.

Let φ be a 3SAT instance with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. For each clause C_j, $1 \leq j \leq m$ seven new nodes are considered, each node corresponding to one of the seven minterms that make C_j true. The total size of the set V of nodes thus introduced is $7m$. For every pair of nodes u, v in V, an edge is introduced if and only if u and v correspond to different clauses and the minterms corresponding to u and v are compatible (i.e., there is no variable in the intersection of u and v that appears as both a negated and an unnegated literal in u and v).

The graph $G(V, E)$ thus constructed has $O(m)$ nodes and $O(m^2)$ edges. Let the lower bound K in the Clique problem be $K = m$. Suppose first that φ is satisfiable; that is, there is a true–false assignment to the variables $x_1, \ldots, x_n$ that make each clause true. This characteristic means that, in each clause C_j, exactly one minterm is true and all such minterms are compatible. The set of the nodes corresponding to these minterms has size m and constitutes a clique, because all compatible nodes have edges among themselves.

Conversely, suppose that there is a clique C in $G(V, E)$ of size $|C| \geq m$. As there are no edges among any of the seven nodes corresponding to clause C_j, $1 \leq j \leq m$, the size of the clique must be exactly $|C| = m$; that is, the clique contains exactly one node from each clause. Moreover, these nodes are compatible as indicated by the edges among them. Therefore, the union of the minterms corresponding to these nodes yields a satisfying assignment for the 3SAT instance. □

It is interesting to note that seemingly related problems and/or special cases of many NP-complete problems exhibit different behavior. For example, the Shortest Path problem, related to the Longest Path problem, is polynomially solvable. The Longest Path problem restricted to directed acyclic graphs is polynomially solvable. The Graph Colorability problem for planar graphs and for $K = 4$ is also polynomially solvable.

On the other hand, the Graph Isomorphism problem, related to the Subgraph Isomorphism but now asking if the two given graphs G and H are isomorphic, is thought to be neither an *NP*-complete problem nor a polynomially solvable problem, although this has not been proved. There is a complexity class called Isomorphic-complete, which is thought to be entirely disjoint from both *NP*-complete and *P*. However, polynomial-time algorithms exist for Graph Isomorphism when restricted to graphs with node degree bounded by a constant (14). In practice, Graph Isomorphism is easy except for pathologically difficult graphs.

NP-Hard Problems

A generalization of the *NP*-complete class is the *NP*-hard class. The *NP*-hard class is extended to comprise optimization problems and decision problems that do not seem to belong to *NP*. All that is required for a problem to be *NP*-hard is that some *NP*-complete problems reduce polynomially to it. For example, the optimization version of the Traveling Salesman problem is an *NP*-hard problem, because if it were polynomially solvable, the decision version of the problem (which is *NP*-complete) would be trivially solved in polynomial time.

If the decision version of an optimization problem is *NP*-complete, then the optimization problem is *NP*-hard, because the "yes" or "no" answer sought in the decision version can readily be given in polynomial time once the optimum solution in the optimization version has been obtained. But it is also the case that for most *NP*-hard optimization problems, a reverse relation holds; that is, these *NP*-hard optimization problems can reduce polynomially to their *NP*-complete decision versions. The strategy is to use a binary search procedure that establishes the optimal value after a logarithmically bounded number of calls to the decision version subroutine. Such *NP*-hard problems are sometimes referred to as NP-*equivalent*. The latter fact is another motivation for the study of *NP*-complete problems: A polynomial-time algorithm for any *NP*-complete (decision) problem would actually provide a polynomial-time algorithm for all such *NP*-equivalent optimization problems.

Algorithms for *NP*-Hard Problems

Once a new problem for which an algorithm is sought is proven to be *NP*-complete or *NP*-hard, the search for a polynomial-time algorithm is abandoned (unless one seeks to prove that $NP = P$), and the following basic four approaches remain to be followed:

(1) Try to improve as much as possible over the straightforward exhaustive (exponential) search by using techniques like branch-and-bound, dynamic programming, cutting plane methods, and Lagrangian techniques.

(2) For optimization problems, try to obtain a polynomial-time algorithm that finds a solution that is provably close to the optimal. Such an algorithm is known as an *approximation algorithm* and is generally the next best thing one can hope for to solve the problem.

(3) For problems that involve numerical bounds, try to obtain an algorithm that is polynomial in terms of the instance size *and* the size of the maximum number occurring in the encoding of the instance. Such an algorithm is known as *pseudopolynomial-time algorithm* and becomes practical if the numbers involved in a particular instance are not too large. An *NP*-complete problem for which a pseudopolynomial-time algorithm exists is referred to as *weakly* NP-*complete* (as opposed to *strongly* NP-*complete*).

(4) Use a polynomial-time algorithm to find a "good" solution based on "rules of thumb" and insight. Such an algorithm is known as a "heuristic." No proof is provided about how good the solution is, but well-justified arguments and empirical studies justify the use of these algorithms in practice.

In some cases, before any of these approaches is examined, one should check whether the problem of concern is actually a polynomially or pseudopolynomially solvable special case of the general *NP*-complete or *NP*-hard problem. Examples include polynomial algorithms for finding a longest path in a directed acyclic graph, and finding a maximum independent set in a transitive graph. Also, some graph problems that are parameterized with an integer k exhibit the *fixed-parameter tractability* property; that is, for each fixed value of k, they are solvable in time bounded by a polynomial whose degree is independent of k. For example, the problem of whether a graph G has a cycle of length at least k, although *NP*-complete in general, is solvable in time $O(n)$ for fixed k, where n is the number of nodes in G. [For other parameterized problems, however, such as the Dominating Set, where the question is whether a graph G has a subset of k nodes such that every node not in the subset has a neighbor in the subset, the only solution known has time complexity $O(n^{k+1})$.]

In the next section, we give some more information about approximation and pseudopolynomial-time algorithms.

POLYNOMIAL-TIME ALGORITHMS

Graph Representations and Traversals

There are two basic schemes for representing graphs in a computer program. Without loss of generality, we assume that the graph is directed [represented by $G(V,A)$]. Undirected graphs can always be considered as bidirected. In the first scheme, known as the *Adjacency Matrix* representation, a $|V| \times |V|$ matrix M is used, where every row and column of the matrix corresponds to a node of the graph, and entry $M(a, b)$ is 1 if and only if $(a, b) \in A$. This simple representation requires $O(|V|^2)$ time and space.

In the second scheme, known as the *Adjacency List* representation, an array $L[1..|V|]$ of linked lists is used. The linked list starting at entry $L[i]$ contains the set of all nodes that are the heads of all edges with tail node i. The time and space complexity of this scheme is $(|V| + |E|)$.

Both schemes are widely used as part of polynomial-time algorithms for working with graphs (15). The Adjacency List representation is more economical to construct, but locating an edge using the Adjacency Matrix representation is very fast [takes $O(1)$ time compared with the $O(|V|)$ time required using the Adjacency List representation]. The choice between the two depends on the way the algorithm needs to access the information on the graph.

A basic operation on graphs is the *graph traversal*, where the goal is to systematically visit all nodes of the graph. There are three popular graph traversal methods: *Depth-first search (DFS), breadth-first search (BFS)*, and *topological search*. The last applies only to directed acyclic graphs. Assume that all nodes are marked initially as unvisited, and that the graph is represented using an adjacency list L.

Depth-first search traverses a graph following the deepest (forward) direction possible. The algorithm starts by selecting the lowest numbered node v and marking it as visited. DFS selects an edge (v, u), where u is still unvisited, marks u as visited, and starts a new search from node u. After completing the search along all paths starting at u, DFS returns to v. The process is continued until all nodes reachable from v have been marked as visited. If there are still unvisited nodes, the next unvisited node w is selected and the same process is repeated until all nodes of the graph are visited.

The following is a recursive implementation of subroutine of *DFS*(v) that determines all nodes reachable from a selected node v. $L[v]$ represents the list of all nodes that are the heads of edges with tail v, and array $M[u]$ contains the visited or unvisited status of every node u.

```
Procedure DFS($v$)
$M[v]$ := visited;
FOR each node $u \in L[v]$DO
    IF $M[u]$ = unvisited THEN Call DFS($u$);
END DFS
```

The time complexity of *DFS*(v) is $O(|V_v| + |E_v|)$, where $|V_v|, |E_v|$ are the numbers of nodes and edges that have been visited by *DFS*(v). The total time for traversing the graph using DFS is $O(|E| + |V|) = O(|E|)$.

Breadth-first search visits all nodes at distance k from the lowest numbered node v before visiting any nodes at distance k+1. Breadth-first search constructs a breadth-first search tree, initially containing only the lowest numbered node. Whenever an unvisited node w is visited in the course of scanning the adjacency list of an already visited node u, node w and edge (u, w) are added to the tree. The traversal terminates when all nodes have been visited. The approach can be implemented using queues so that it terminates in $O(|E|)$ time.

The final graph traversal method is the topological search, which applies only to directed acyclic graphs. In directed acyclic graphs, nodes have no incoming edges and nodes have no outgoing edges. Topological search visits a node only if it has no incoming edges or all its incoming edges have been explored. The approach can be also be implemented to run in $O(|E|)$ time.

Design Techniques for Polynomial-Time Algorithms

We describe below three main frameworks that can be used to obtain polynomial-time algorithms for combinatorial optimization (or decision) problems: (1) greedy algorithms; (2) divide-and-conquer algorithms; and (3) dynamic programming algorithms. Additional frameworks such as linear programming are available, but these are discussed in other parts of this book.

Greedy Algorithms. These algorithms use a greedy (straightforward) approach. As an example of a greedy algorithm, we consider the problem of finding the chromatic number (minimum number of independent sets) in an interval graph. An interval is a line aligned to the y-axis, and each interval i is represented by its left and right endpoints, denoted by l_i and r_i, respectively. This representation of the intervals is also referred to as the interval diagram.

In an interval graph, each node corresponds to an interval and two nodes are connected by an edge if the two intervals overlap. This problem finds immediate application in VLSI routing (10). The minimum chromatic number corresponds to the minimum number of tracks so that all intervals allocated to a track do not overlap with each other.

The greedy algorithm is better described on the interval diagram instead of the interval graph; i.e., it operates on the intervals rather than or the respective nodes on the interval graph. It is also known as the left-edge algorithm because it first sorts the intervals according to their left-edge values l_i and then allocates them to tracks in ascending order of l_i values.

To allocate a track to the currently examined interval i, the algorithm serially examines whether any of the existing tracks can accommodate it. (The existing tracks have been generated to accommodate all intervals j such that $l_j \leq l_i$.) Net i is greedily assigned to the first existing track that can accommodate it. A new track is generated, and interval i is assigned to that track only if i has a conflict with each existing track.

It can be shown that the algorithm results to minimum number of tracks; i.e., it achieves the chromatic number of the interval graph. This happens because if more than one track can accommodate interval i, then any assignment will lead to an optimal solution because all nets k with $l_k \geq l_i$ can be assigned to the unselected tracks.

Divide-and-Conquer. This methodology is based on a systematic partition of the input instance in a top–down manner into smaller instances, until small enough instances are obtained for which the solution of the problem degenerates to trivial computations. The overall optimal solution, i.e., the optimal solution on the original input instance, is then calculated by appropriately working on the already calculated results on the subinstances. The recursive nature of the methodology necessitates the solution of one or more recurrence relations to determine the execution time.

As an example, consider how divide-and-conquer can be applied to transform a *weighted binary tree* into a *heap*. A weighted binary tree is a rooted directed tree in which every nonleaf node has out-degree 2, and there is a value (weight) associated with each node. (Each nonleaf node is also referred to as *parent*, and the two nodes it points to are referred to as its *children*.) A heap is a weighted binary tree in which the weight of every node is no smaller than the weight of either of its children.

The idea is to recursively separate the binary tree into subtrees starting from the root and considering the subtrees rooted at its children, until the leaf nodes are encountered. The leaf nodes constitute trivially a heap. Then, inductively, given two subtrees of the original tree that are rooted at the children of the same parent and have been made to have the heap property, the subtree rooted at the parent can be made to have the heap property too by simply finding which child has the largest weight and exchanging that weight with the weight of the parent in case the latter is smaller than the former.

Dynamic Programming. In dynamic programming, the optimal solution is calculated by starting from the simplest subinstances and combining the solutions of the smaller subinstances to solve larger subinstances, in a bottom-up manner. To guarantee a polynomial-time algorithm, the total number of subinstances that have to be solved must be polynomially bounded. Once a subinstance has been solved, any larger subinstance that needs that subinstance's solution, does not recompute it, but looks it up in table where it has been stored. Dynamic programming is applicable only to problems that obey the *principle of optimality*. This principle holds whenever in an optimal sequence of choices, each subsequence is also optimal. The difficulty in this approach is to come up with a decomposition of the problem into a sequence of subproblems for which the principle of optimality holds and can be applied in polynomial time.

We illustrate the approach by finding the maximum independent set in a cycle graph in $O(n^2)$ time, where n is the number of chordal endpoints in the cycle (10). Note that the maximum independent set is *NP*-hard on general graphs.

Let $G(V, E)$ be a cycle graph, and $v_{ab} \in V$ correspond to a chord in the cycle. We assume that no two chords share an endpoint, and that the endpoints are labeled from 0 to $2n-1$, where n is the number of chords, clockwise around the cycle. Let G_{ij} be the subgraph induced by the set of nodes $v_{ab} \in V$ such that $i \leq a, b \leq j$.

Let $M(i, j)$ denote a maximum independent set of $G_{i,\,j}$. $M(i,j)$ is computed for every pair of chords, but $M(i, a)$ must be computed before $M(i, b)$ if $a < b$. Observe that, if $i \geq j$, $M(i, j) = 0$ because $G_{i,\,j}$ has no chords. In general, to compute $M(i, j)$, the endpoint k of the chord with endpoint j must be found. If k is not in the range $[i, j-1]$, then $M(i, j) = M(i, j-1)$ because graph $G_{i,\,j}$ is identical to graph $G_{i,j-1}$. Otherwise we consider two cases: (*1*) If $v_{kj} \in M(i, j)$, then $M(i,j)$ does not have any node v_{ab}, where $a \in [i, k-1]$ and $b \in [k+1, j]$. In this case, $M(i, j) = M(i, k-1) \cup M(K = 1,\quad j-1) \cup v_{kj}$. (*2*) If $v_{kj} \notin M(i, j)$, then $M(i, j) = M(i, j-1)$. Either of these two cases may apply, but the largest of the two maximum independent sets will be allocated to $M(i,j)$. The flowchart of the algorithm is given as follows:

```
Procedure MIS($V$)
FOR $j = 0$ TO $2n - 1$ DO
   Let $(j, k)$ be the chord whose one endpoint is $j$;
   FOR $i = 0$ TO $j - 1$ DO
      IF $i \leq k \leq j-1$ AND $|M(i,k-1)| + 1 + |M(k+1,j-1)| >$
         $|M(i,j-1)|$ THEN $M(i,j) = M(i, k-1) \cup v_{kj} \cup M(k+$
         $1,j-1)$;
      ELSE $M(i,j) = M(i,j-1)$;
END MIS
```

Basic Graph Problems

In this section, we discuss more analytically four problems that are widely used in VLSI CAD, computer networking, and other areas, in the sense that many problems are reduced to solving these basic graph problems. They are the *shortest path problem*, the *flow problem*, the *graph coloring problem*, and the *graph matching problem*.

Shortest Paths. The instance consists of a graph $G(V, E)$ with lengths $l(u, v)$ on its edges (u, v), a given source $s \in V$, and a target $t \in V$. We assume without loss of generality that the graph is directed. The goal is to find a shortest length path from s to t. The weights can be positive or negative numbers, but there is no cycle for which the sum of the weights on its edges is negative. (If negative length cycles are allowed, the problem is *NP*-hard.) Variations of the problem include the all-pair of nodes shortest paths and the m shortest path calculation in a graph.

We present here a dynamic programming algorithm for the shortest path problem that is known as the Bellman–Ford algorithm. The algorithm has $O(n^3)$ time complexity, but faster algorithms exist when all the weights are positive [e.g., the Dijkstra algorithm with complexity $O(n \cdot \min\{\log|E|, |V|\})$] or when the graph is acyclic (based on topological search and with linear time complexity). All existing algorithms for the shortest path problem are based on dynamic programming. The Bellman–Ford algorithm works as follows.

Let $l(i, k)$ be the length of edge (i, j) if directed edge (i, j) exists and ∞ otherwise. Let $s(j)$ denote the length of the shortest path from the source s to node j. Assume that the source has label 1 and that the target has label $n = |V|$. We have that $s(1) = 0$. We also know that in a shortest path to any node j there must exist a node k, $k \neq j$, such that $s(j) = s(k) + l(k, j)$. Therefore,

$$s(j) = \min_{k \neq j}\{s(k) + l(k, j)\}, j \geq 2$$

Bellman–Ford's algorithm, which eventually computes all $s(j)$, $1 \leq j \leq n$, calculates optimally the quantity

$s(j)^{m+1}$ defined as the length of the shortest path to node j subject to the condition that the path does not contain more than $m + 1$ edges, $0 \leq m \leq |V| - 2$. To be able to calculate quantity $s(j)^{m+1}$ for some value $m + 1$, the $s(j)^m$ values for all nodes j have to be calculated.

Given the initialization $s(1)^1 = 0, s(j)^1 = l(1, j),\ j \neq 1$, the computation of $s(j)^{m+1}$ for any values of j and m can be recursively computed using the formula

$$s(j)^{m+1} = \min\{s(j)^m, \min\{s(k)^m + l(k, j)\}$$

The computation terminates when $m = |V| - 1$, because no shortest path has more than $|V| - 1$ edges.

Flows. All flow problem formulations consider a directed or undirected graph $G = (V, E)$, a designated source s, a designated target t, and a nonnegative integer capacity $c(i, j)$ on every edge (i, j). Such a graph is sometimes referred to as a *network*. We assume that the graph is directed. A *flow* from s to t is an assignment F of numbers $f(i, j)$ on the edges, called the amount of flow through edge (i, j), subject to the following conditions:

$$0 \leq f(i, j) \leq c(i, j) \qquad (1)$$

Every node i, except s and t, must satisfy the *conservation of flow*. That is,

$$\sum_j f(j, i) - \sum_j f(i, j) = 0 \qquad (2)$$

Nodes s and t satisfy, respectively,

$$\begin{aligned} \sum_j f(j, i) - \sum_j f(i, j) &= -v, \text{ if } i = s; \\ \sum_j f(j, i) - \sum_j f(i, j) &= v, \text{ if } i = t \end{aligned} \qquad (3)$$

where $v = \sum_j f(s, j) = \sum_j f(j, t)$ is called the value of the flow.

A flow F that satisfies Equations (1–3) is called *feasible*. In the Max Flow problem, the goal is to find a feasible flow F for which v is maximized. Such a flow is called a *maximum flow*. There is a problem variation, called the Minimum Flow problem, where Equation (1) is substituted by $f(i, j) \geq c(i, j)$ and the goal is to find a flow F for which v is minimized. The minimum flow problem can be solved by modifying algorithms that compute the maximum flow in a graph.

Finally, another flow problem formulation is the Minimum Cost Flow problem. Here each edge has, in addition to its capacity $c(i, j)$, a *cost* $p(i, j)$. If $f(i, j)$ is the flow through the edge, then the cost of the flow through the edge is $p(i, j) \cdot f(i, j)$ and the overall cost C for a flow F of a value v is $\sum_{i,j} p(i, j) \cdot f(i, j)$. The problem is to find a *minimum cost flow* F for a given value v.

Many problems in VLSI CAD, computer networking, scheduling, and so on can be modeled or reduced to one of these three flow problem variations. All three problems can be solved in polynomial time using as subroutines shortest path calculations. We describe below for illustration purposes an $O(|V|^3)$ algorithm for the maximum flow problem. However, faster algorithms exist in the literature. We first give some definitions and theorems.

Let P be an *undirected* path from s to t; i.e., the direction of the edges is ignored. An edge $(i, j) \in P$ is said to be a *forward* edge if it is directed from s to t and *backward* otherwise. P is said to be an *augmenting path* with respect to a given flow F if $f(i, j) < c(i, j)$ for each forward edge and $f(i, j) > 0$ for each backward edge in P.

Observe that if the flow in each forward edge of the augmenting path is increased by one unit and the flow in each backward edge is decreased by one unit, the flow is feasible and its value has been increased by one unit. We will show that a flow has maximum value if and only if there is no augmenting path in the graph. Then the maximum flow algorithm is simply a series of calls to a subroutine that finds an augmenting path and increments the value of the flow as described earlier.

Let $S \subset V$ be a subset of the nodes. The pair (S, T) is called a *cutset* if $T = V - S$. If $s \in S$ and $t \in T$, the (S, T) is called an (s, t) *cutset*. The *capacity of the cutset* (S, T) is defined as $c(S, T) = \sum_{i \in S} \sum_{j \in T} c(i, j)$, i.e., the sum of the capacities of all edges from S to T. We note that many problems in networking, operations research, scheduling, and VLSI CAD (physical design, synthesis, and testing) are formulated as minimum capacity (s, t) cutset problems. We show below that the minimum capacity (s, t) problem can be solved with a maximum flow algorithm.

Lemma 1. *The value of any (s, t) flow cannot exceed the capacity of any (s, t) cutset.*

Proof. Let F be an (s, t) flow with value v. Let (S, T) be an (s, t) cutset. From Equation 3 the value of the flow v is also $v = \sum_{i \in S}(\sum_j f(i, j) - \sum_j f(j, i)) = \sum_{i \in S} \sum_{j \in S}(f(i, j) - f(j, i)) + \sum_{i \in S} \sum_{j \in T}(f(i, j) - f(j, i)) = \sum_{i \in S} \sum_{j \in T}(f(i, j) - f(j, i))$, because $\sum_{i \in S} \sum_{j \in S}(f(i, j) - f(j, i))$ is 0. But $f(i, j) \leq c(i, j)$ and $f(j, i) \geq 0$. Therefore, $v \leq \sum_{i \in S} \sum_{j \in S} c(i, j) = c(S, T)$.□

Theorem 2. *A flow F has maximum value v if and only if there is no augmenting path from s to t.*

Proof. If there is an augmenting path, then we can modify the flow to get a larger value flow. This contradicts the assumption that the original flow has a maximum value.

Suppose on the other hand that F is a flow such that there is no augmenting path from s to t. We want to show that F has the maximum flow value. Let S be the set of all nodes j (including s) for which there is an augmenting path from s to j. By the assumption that there is an augmenting path from s to t, we must have that $t \notin S$. Let $T = V - S$ (recall that $t \in T$). From the definition of S and T, it follows that $f(i, j) = c(i, j)$ and $f(j, i) = 0,\ \forall i \in S, j \in T$.

Now $v = \sum_{i \in S}(\sum_j f(i, j) - \sum_j f(j, i)) = \sum_{i \in S} \sum_{j \in S} (f(i, j) - f(j, i)) + \sum_{i \in S} \sum_{j \in T}(f(i, j) - f(j, i)) = \sum_{i \in S} \sum_{j \in T} (f(i, j) - f(j, i)) = \sum_{i \in S} \sum_{j \in S} c(i, j)$, because $c(i, j) = f(i, j)$ and $f(j, i) = 0,\ \forall i, j$. By Lemma 1, the flow has the maximum value.□

Next, we state two theorems whose proof is straightforward.

Theorem 3. *If all capacities are integers, then a maximum flow F exists, where all f(i,j) are integers.*

Theorem 4. *The maximum value of an (s, t) flow is equal to the minimum capacity of an (s, t) cutset.*

Finding an augmenting path in a graph can be done by a systematic graph traversal in linear time. Thus, a straightforward implementation of the maximum flow algorithm repeatedly finds an augmenting path and increments the amount of the (s, t) flow. This is a pseudopolynomial-time algorithm (see the next section), whose worst-case time complexity is $O(v \cdot |E|)$. In many cases, such an algorithm may turn out to be very efficient. For example, when all capacities are uniform, then the overall complexity becomes $O(|E|^2)$.

In general, the approach needs to be modified using the Edmonds–Karp modification (6), so that each flow augmentation is made along an augmenting path with a minimum number of edges. With this modification, it has been proven that a maximum flow is obtained after no more than $|E| \cdot |V|/2$ augmentations, and the approach becomes fully polynomial. Faster algorithms for maximum flow computation rely on capacity scaling techniques and are described in Refs. 8 and 15, among others.

Graph Coloring. Given a graph $G(V, E)$, a *proper k-coloring* of G is a function f from V to a set of integers from 1 to k (referred to as *colors*) such that $f(u) \neq f(v)$, if $(u, v) \in E$. The minimum k for which a proper k-coloring exists for graph G is known as the *chromatic number* of G. Finding the chromatic number of a general graph and producing the corresponding coloring is an *NP*-hard problem. The decision problem of graph k-colorability is *NP*-complete in the general case for fixed $k \geq 3$. For $k = 2$, it is polynomially solvable (bipartite matching). For planar graphs and for $k = 4$, it is also polynomially solvable.

The graph coloring problem finds numerous applications in computer networks and communications, architectural level synthesis, and other areas. As an example, the Federal Communications Commission (FCC) monitors radio stations (modeled as nodes of a graph) to make sure that their signals do not interfere with each other. They prevent interference by assigning appropriate frequencies (each frequency is a color) to each station. It is desirable to use the smallest possible number of frequencies. As another example, several resource binding algorithms for data path synthesis at the architectural level are based on graph coloring formulations (9).

There is also the version of *edge coloring:* A *proper k-edge-coloring* of G is a function f from E to a set of integers from 1 to k such that $f(e_1) \neq f(e_2)$, if edges e_1 and e_2 share a node. In this case, each color class corresponds to a *matching* in G, that is, a set of pairwise disjoint edges of G. The minimum k for which a proper k-edge-coloring exists for graph G is known as the *edge-chromatic number* of G. It is known (Vizing's theorem) that for any graph with maximum node degree d, there is a $(d + 1)$-edge coloring. Finding such a coloring can be done in $O(n^2)$ time. Notice that the edge-chromatic number of a graph with maximum node degree d is at least d. It is *NP*-complete to determine whether the edge-chromatic number is d, but given Vizing's algorithm, the problem is considered in practice solved.

Graph Matching. A *matching* in a graph is a set $M \subset E$ such that no two edges in M are incident to the same node.

The Maximum Cardinality Matching problem is the most common version of the matching problem. Here the goal is to obtain a matching so that the size (cardinality) of M is maximized.

In the Maximum Weighted Matching version, each edge $(i, j) \in V$ has a nonnegative integer weight, and the goal is to find a matching M so that $\sum_{e \in M} w(e)$ is maximized.

In the Min-Max Matching problem, the goal is to find a maximum cardinality matching M where the minimum weight on an edge in M is maximized. The Max-Min Matching problem is defined in an analogous manner.

All above matching variations are solvable in polynomial time and find important applications. For example, a variation of the min-cut graph partitioning problem, which is central in physical design automation for VLSI, asks for partitioning the nodes of a graph into sets of size at most two so that the sum of the weights on all edges with endpoints in different sets is minimized. It is easy to see that this partitioning problem reduces to the maximum weighted matching problem.

Matching problems often occur on bipartite $G(V_1 \cup V_2, E)$ graphs. The maximum cardinality matching problem amounts to the *maximum assignment* of elements in V_1 ("workers") on to the elements of V_2 ("tasks") so that no "worker" in V_1 is assigned more than one "task." This finds various applications in operations research.

The maximum cardinality matching problem on a bipartite graph $G(V_1 \cup V_2, E)$ can be solved by a maximum flow formulation. Simply, each node $v \in V_1$ is connected to a new node s by an edge (s, v) and each node $u \in V_2$ to a new node t by an edge (u, t). In the resulting graph, every edge is assigned unit capacity. The maximum flow value v corresponds to the cardinality of the maximum matching in the original bipartite graph G.

Although the matching problem variations on bipartite graphs are amenable to easily described polynomial-time algorithms, such as the one given above, the existing polynomial-time algorithms for matchings on general graphs are more complex (6).

Approximation and Pseudopolynomial Algorithms

Approximation and pseudopolynomial-time algorithms concern mainly the solution of problems that are proven to be *NP*-hard, although they can sometimes be used on problems that are solvable in polynomial time, but for which the corresponding polynomial-time algorithm involves large constants.

An α-approximation algorithm A for an optimization problem R is a polynomial-time algorithm such that, for any instance I of R, $\frac{|S_A(I) - S_{\mathrm{OPT}}(I)|}{S_{\mathrm{OPT}}(I)} \leq \alpha + c$, where $S_{\mathrm{OPT}}(I)$ is the cost of the optimal solution for instance I, $S_A(I)$ is the cost of the solution found by algorithm A, and c is a constant.

As an example, consider a special but practical version of the Traveling Salesman problem that obeys the triangular inequality for all city distances. Given a weighted graph $G(V, E)$ of the cities, the algorithm first finds a minimum spanning tree T of G (that is, a spanning tree that has minimum sum of edge weights). Then it finds a minimum weight matching M among all nodes that have odd degree in T. Lastly, it forms the subgraph $G'(V, E')$, where E' is the set of all edges in T and M and finds a path that starts from and terminates to the same node and passes through every edge exactly once (such a path is known as "Eulerian tour"). Every step in this algorithm takes polynomial time. It has been shown that $\frac{|S_A(I)-S_{\mathrm{OPT}}(I)|}{S_{\mathrm{OPT}}(I)} \leq \frac{1}{2}$.

Unfortunately, obtaining a polynomial-time approximation algorithm for an *NP*-hard optimization problem can be very difficult. In fact, it has been shown that this may be impossible for some cases. For example, it has been shown that unless $NP = P$, there is no α-approximation algorithm for the general Traveling Salesman problem for any $\alpha > 0$.

A pseudopolynomial-time algorithm for a problem R is an algorithm with time complexity $O(p(n, m))$, where $p()$ is a polynomial of two variables, n is the size of the instance, and m is the magnitude of the largest number occuring in the instance. Only problems involving numbers that are not bounded by a polynomial on the size of the instance are applicable for solution by a pseudopolynomial-time algorithm. In principle, a pseudopolynomial-time algorithm is exponential given that the magnitude of a number is exponential to the size of its logarithmic encoding in the problem instance, but in practice, such an algorithm may be useful in cases where the numbers involved are not large.

NP-complete problems for which a pseudopolynomial-time algorithm exists are referred to as *weakly* NP-*complete*, whereas *NP*-complete problems for which no pseudopolynomial-time algorithm exists (unless $NP = P$) are referred to as *strongly* NP-*complete*. As an example, the Network Inhibition problem, where the goal is to find the most cost-effective way to reduce the ability of a network to transport a commodity, has been shown to be strongly NP-complete for general graphs and weakly *NP*-complete for series-parallel graphs (16).

Probabilistic and Randomized Algorithms

Probabilistic algorithms are a class of algorithms that do not depend exclusively on their input to carry out the computation. Instead, at one or more points in the course of the algorithm where a choice has to be made, they use a pseudorandom number generator to select "randomly" one out of a finite set of alternatives for arriving at a solution. Probabilistic algorithms are fully programmable (i.e., they are not nondeterministic), but in contrast with the deterministic algorithms, they may give different results for the same input instance if the initial state of the pseudorandom number generator differs each time.

Probabilistic algorithms try to reduce the computation time by allowing a small probability of error in the computed answer ("Monte Carlo" type) or by making sure that the running time to compute the correct answer is small for the large majority of input instances ("Las Vegas" type). That is, algorithms of the Monte Carlo type always run fast, but the answer they compute has a small probability of being erroneous, whereas algorithms of the Las Vegas type always compute the correct answer but they occasionally may take a long time to terminate. If there is an imposed time limit, Las Vegas-type algorithms can alternatively be viewed as always producing either a correct answer or no answer at all within that time limit.

An example of a probabilistic algorithm on graphs is the computation of a minimum spanning in a weighted undirected graph by the algorithm of (17). This algorithm computes a minimum spanning tree in $O(m)$ time, where m is the number of edges, with probability $1 - e^{-\Omega(m)}$.

Randomization is also very useful in the design of heuristic algorithms. A typical example of a randomized heuristic algorithm is the simulated annealing method, which is very effective for many graph theoretic formulations in VLSI design automation (10). In the following we outline the use of simulated annealing in the graph balanced bipartitioning problem, where the goal is to partition the nodes of a graph in two equal-size sets so that the number of edges that connect nodes in two different sets is minimized. This problem formulation is central in the process of obtaining a compact layout for the integrated circuit whose components (gates or modules) are represented by the graph nodes and its interconnects by the edges.

The optimization in balanced bipartitioning with a very large number of nodes is analogous to the physical process of annealing where a material is melted and subsequently cooled down, under a specific schedule, so that it will crystallize. The cooling must be slow so that thermal equilibrium is reached at each temperature so that the atoms are aranged in a pattern that resembles the global energy minimum of the perfect crystal.

While simulating the process, the energy within the material corresponds to a partitioning score. The process starts with a random initial partitioning. An alternative partitioning is obtained by exchanging nodes that are in opposite sets. If the change in the score δ is negative, then the exchange is accepted because this represents reduction in the energy. Otherwise, the exchange is accepted with probability $e^{-\delta/t}$; i.e., the probability of acceptance decreases with the increase in temperature t. This method allows the simulated annealing algorithm to climb out of local optimums in the search for a global optimum.

The quality of the solution depends on the initial value of the temperature value and the cooling schedule. Such parameters are determined experimentally. The quality of the obtained solutions is very good, altough the method (being a heuristic) cannot guarantee optimality or even a provably good solution.

FURTHER READING

E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. New York: Computer Science Press, 1984.

BIBLIOGRAPHY

1. B. Bollobas, *Modern Graph Theory*. New York: Springer Verlag, 1988.
2. S. Even, *Graph Algorithms*. New York: Computer Science Press, 1979.
3. J. L. Gross and J. Yellen, *Graph Theory and its Applications*. CRC Press, 1998.
4. R. Sedgewick, *Algorithms in Java: Graph Algorithms*. Reading, MA: Addison-Wesley, 2003.
5. K. Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*. New York: Wiley, 1992.
6. E. L. Lawler, *Combinatorial Optimization—Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.
7. C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice Hall, 1982.
8. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*. Englewood Cliffs, NJ: Prentice Hall, 1993.
9. G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.
10. N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*. New York: Kluwer Academic, 1993.
11. D. Bertsekas and R. Gallagher, *Data Networks*. Upper Saddle River, NJ: Prentice Hall, 1992.
12. M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. New York: Computer Science Press, 1990.
13. M. R. Garey and D. S. Johnson, *Computers and Intractability—A Guide to the Theory of* NP-*Completeness*. New York: W. H. Freeman, 1979.
14. S. Skiena, Graph isomorphism, in *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Reading, MA: Addison-Wesley, 1990, pp. 181–187.
15. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.
16. C. A. Phillips, The network inhibition problem, *Proc. 25th Annual ACM Symposium on the Theory of Computing*, 1993, pp. 776–785.
17. D. R. Karger, P. N. Klein, and R. E. Tarjan, A randomized linear time algorithm to find minimum spanning trees, *J. ACM*, **42**(2): 321–328, 1995.

DIMITRI KAGARIS
SPYROS TRAGOUDAS
Southern Illinois University
Carbondale, Illinois

I

INFORMATION ALGEBRA

INTRODUCTION

Many apparently very different problems from computer science can be solved by fast generic algorithms because they develop from instances of a very general abstract structure called *information algebra*. To illustrate in a simple way these generic problems and the nature of the fast algorithms derived from them, we consider the following example:

Let $f(X_1, X_2)$, $g(X_2, X_3)$, and $h(X_2, X_4, X_5)$ be given real-valued functions, where $X_1, \ldots, X_5$ are variables that take values from a finite set X with n elements. Such discrete functions can be represented in tabular form. Suppose we are given the task to compute the values of $\alpha(X_1, X_2)$ and $\beta(X_2, X_3)$, which are defined as follows:

$$\begin{aligned}\alpha(X_1,X_2) &= \sum_{X_3,X_4,X_5} f(X_1,X_2)g(X_2,X_3)h(X_2,X_4,X_5)\\ \beta(X_2,X_3) &= \sum_{X_1,X_4,X_5} f(X_1,X_2)g(X_2,X_3)h(X_2,X_4,X_5)\end{aligned} \tag{1}$$

Here, the summation over a list of variables adds up to the summation over all possible assignments to those variables, and the task consists in the computation of the values of α and β for all possible assignments to X_1, X_2 and X_2, X_3, respectively. If we compute the sums directly as they are written here, then both sums require the multiplication of $O(n^5)$ values, which results in a table with the same number of elements, and then $O(n^5)$ additions. This method can be done much more efficiently. So, the first sum can be computed as

$$\alpha(X_1,X_2) = f(X_1,X_2) \left(\left(\sum_{X_3} g(X_2, X_3)\right)\left(\sum_{X_4,X_5} h(X_2, X_4, X_5)\right)\right) \tag{2}$$

Here, we need $O(n^2)$ multiplications and the tables have never more than n^2 elements and only $O(n^3)$ additions. This result is different with respect to the first, direct computation. Of course, we need to verify that Equation (2) yields in fact the same result as Equation (1). This method involves essentially the distributive law of arithmetic. Similarly, the second sum can be computed as

$$\beta(X_2, X_3) = g(X_2, X_3) \sum_{X_1} \left(f(X_1, X_2)\left(\sum_{X_4,X_5} h(X_2, X_4, X_5)\right)\right) \tag{3}$$

But it needs again only $O(n^2)$ multiplications; the resulting tables have never more that n^2 elements, and only $O(n^3)$ additions are needed. Once more, this result represents an important gain. The gain obtained by clever use of the distributive law becomes even more important if we are charged with computations that contain hundreds or thousands of variables.

Distributive laws hold in more general mathematical structures than simple arithmetic, and consequently, fast algorithms based on this law can be developed in more general structures. In the next section, we propose an abstract algebraic framework that proves to be sufficient to enable the generic fast algorithms we have in mind. These algorithms are described in a generic way in the section on Local Computation. In the section on Semiring Valuations we present an important class of models of this axiomatic frame, which contains also the example discussed above. These abstract generic structures cover a great many problems of computer science, which include basic problems from relational databases, constraint systems, propositional and predicate logic, as well as many other logic systems. More problems are located in the field of numerical analysis, which includes systems of linear equations and linear inequalities, decoding algorithms as for instance Viterbi's algorithm, fast Fourier transforms, and belief propagation algorithms from artificial intelligence. Not to be forgotten are methods based on probability networks, possibility theory and fuzzy sets, belief functions from Dempster-Shafer theory of evidence, and many more.

AXIOMATICS

We abstract the illustrating example in the previous section with the interpretation as a framework for the processing of information. In all the examples of computational problems referred to at the end of the previous section, in some way or another information is processed. Now, information pertains to questions, for instance to the possible values of certain variables $X_1, X, \ldots X_m$ out of a set r of variables. Let Φ be a set of elements whose generic elements are denoted by small Greek letters like $\phi, \psi, \ldots$. Each element of Φ pertains to some subset $s \subseteq r$ of variables. This subset is determined by the *labeling* function $d : \Phi \to D$, where D is the powerset of r. The elements of Φ can be thought of as pieces of information ϕ that pertains to sets of variables $d(\phi)$. We call $d(\phi)$ also the domain of ϕ. For instance, the functions f, g, and h in the example of the previous section could be recognized as such pieces of informations that refer to the sets of variables $\{X_1, X_2\}$, $\{X_2, X_3\}$, and $\{X_2, X_4, X_5\}$, respectively.

We consider now a two-sorted algebra (Φ, D) with three operations that are defined as follows:

1. *Labeling*: $\Phi \to D$, $\phi \to d(\phi)$,
2. *Combination*: $\Phi \times \Phi \to \Phi$, $(\phi, \psi) \to \phi \otimes \psi$,
3. *Projection*: $\Phi \times D \to \Phi$, $(\phi, s) \to \phi^{\downarrow s}$, defined $s \subseteq d(\phi)$.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Combination can be thought of as the operation of aggregation of information. In the example of the first section for instance, the combination is defined as multiplication:

$$(f\otimes g)(X_1, X_2, X_3) = f(X_1,X_2)g(X_2, X_3).$$

Projection represents extraction of information regarding a specified set of variables. In the introductory example, projection is defined as summation:

$$f^{\downarrow\{X_1\}}(X_1) = \sum_{X_2} f(X_1,X_2)$$

Now these operations must satisfy certain axioms:

1. *Semigroup*: The operation of combination is commutative and associative, that is,

$$\phi\otimes\psi = \psi\otimes\phi,\ (\phi\otimes\psi)\otimes\eta = \phi\otimes(\psi\otimes\eta).$$

2. *Labeling*: $d(\phi\otimes\psi) = d(\phi)\cup d(\psi)$.
3. *Projection*: $d(\phi^{\downarrow s}) = s$.
4. *Transitivity of projection*: If $t\subseteq s\subseteq d(\phi)$, then

$$(\phi^{\downarrow s})^{\downarrow t} = \phi^{\downarrow t}.$$

5. *Distributivity of combination over projection*: If $d(\phi) = s$, $d(\psi) = t$ and $u\in D$ such that $s\subseteq u\subseteq s\cup t$, then

$$(\phi\otimes\psi)^{\downarrow u} = \phi\otimes\psi^{\downarrow u\cap t}.$$

The main axioms are (4) and especially (5). Axiom (4) states that projection can be done in steps. Axiom (5) is the generalized distributive law that allows fast algorithms (see the section Local Computation). It suggests that projection can, in certain important cases, be done before combination. In terms of the example of the first section, it reads as follows:

$$\sum_{X_3} f(X_1,X_2)g(X_2,X_3) = f(X_1,X_2)\sum_{X_3} g(X_2,X_3)$$

According to axiom (1), Φ is a commutative semigroup under combination. Axiom (2) says that domains of aggregated pieces of information are joined. Finally, axiom (3) assures that the domain of a projected information is the domain on which it is projected. This algebraic structure captures already essential elements of what is expected of a concept of information regarding aggregation (combination) and focusing (projection). Because one important element is still missing, which we would expect for a concept of information (see the section on idempotency), we call a two-sorted algebra (Φ, D) satisfying the axioms (1)–(5) a *valuation algebra* and not yet a proper information algebra. However, valuation algebras represent the framework that defines the generic fast algorithms we have in mind and that will be discussed in the section entitled "Local computation".

Some valuation algebras have additional properties. Some models have *neutral elements* e_s on any domain s, such that $e_s\otimes\phi = \phi$ for all elements with $d(\phi) = s$. These elements represent *vacuous information* with respect to the variables in s. Sometimes, *null elements* z_s are on every domain $s\in D$, such that $z_s\otimes\phi = z_s$ for all elements with $d(\phi) = s$. These elements represent falsity or contradiction; elements such that $\phi\otimes\psi = z_s$ must be considered to represent contradictory information.

In the next section, an important class of valuation algebras will be presented. This class contains also many important examples. More examples are given in the last section.

The general inference problem associated with a valuation algebra (Φ, D) can be stated as follows: We are given a finite set of pieces of information $\phi_1,\dots\phi_m\in\Phi$ with domains $s_i = d(\phi_i)$ for $i = 1,\cdots,m$ and one or several domains $x_1,\dots,x_k\in D$. Then, the following projections have to be computed:

$$(\phi_1\otimes\cdots\otimes\phi_m)^{\downarrow x_j} \text{ for } j = 1,\dots,k \tag{4}$$

This formula is called the *projection problem*. The domains s_i are called data domains and the x_j goal domains. The tasks of computing $\alpha(X_1,X_2)$ and $\beta(X_2,X_3)$ in the first section are examples of projection problems as it will be explained in the next section. Just as in these examples, the naive computation based on the expression given in Equation (4) is not efficient and in many cases is intractable. But again, the distributivity axiom together with the transitivity axiom allow for efficient algorithms to solve the projection problem. This method will be explained in the section entitled "Local compution."

We remark that we can adjoin an element e with domain *d(e)* $=\emptyset$ to a valuation algebra (Φ, D) and define $\phi\otimes e = e\otimes\phi = \phi$ for all $\phi\in\Phi$, as well as $e\otimes e = e$ and $e^{\downarrow\emptyset} = e$. Clearly, the augmented algebra $(\Phi\cup\{e\}, D)$ is still a valuation algebra. The element e is called the *identity element*.

SEMIRING VALUATIONS

Often, configurations of possible values of a set s of variables are evaluated by values from a *commutative semiring*. A commutative semiring is a set A with two binary operations $+$ and $\times$, where both operations are commutative and associative and where $\times$ distributes over $+$ so that for *a,b,c* $\in A$,

$$a\times(b+c) = a\times b + a\times c$$

Often, we simply speak of a semiring, but we always mean a commutative one. With respect to the introduction, in which more efficient computations are found by use of the distributive law, we may say that a semiring constitutes the simplest mathematical structure where the distributive law still holds. Below, several important examples of semirings are given, and we also point out that a *ring* is a semiring with additive inverses, and a *field* is a ring with multiplicative inverses.

To formulate the idea of a semiring valuation more precisely, assume as before that the variables $X_1, \ldots, X_m$ take values in a finite set X. A mapping $\mathrm{x} : s \to X$ is called a tuple on the variable set s. The set of all s-tuples will be denoted by X^s. If s is the empty set, then we define $\diamond$ to be the only s-tuple. If x is a tuple on s and t a subset of s, then $\mathrm{x}^{\downarrow t}$ denotes the tuple on t obtained from the s-tuple x by restriction to the indices in t. If we want to emphasize the decomposition of a s-tuple x into disjoint components that belong to the subsets t and $s \setminus t$ of s, we write $\mathrm{x} = (\mathrm{x}^{\downarrow t}, \mathrm{x}^{\downarrow s\setminus t})$. A *semiring valuation* on s is then a mapping $\phi : X^s \to A$, where A is a semiring. Let Φ_s denote the set of all semiring valuations on s and

$$\Phi_A = \bigcup_{s \subseteq r} \Phi_s$$

We define now a two-sorted algebra (Φ_A, D), where D is as before the powerset of r. In fact, we define the three operations required for a valuation algebra as follows:

1. *Labeling*: $d(\phi) = s$, if $\phi \in \Phi_s$.
2. *Combination*: The semiring valuation $\phi \otimes \psi$, where ϕ and ψ are semiring valuations on s and t, respectively, is defined for all $s \cup t$-tuple x by
$$(\phi \otimes \psi)(\mathrm{x}) = \phi(\mathrm{x}^{\downarrow s}) \times \psi(\mathrm{x}^{\downarrow t})$$
3. *Projection*: The semiring valuation $\phi^{\downarrow t}$, were ϕ is a semiring valuation on s and $t \subseteq s$, is defined for any t-tuple x by
$$\phi^{\downarrow t}(\mathrm{x}) = \sum_{y \in X^{s\setminus t}} \phi(\mathrm{x}, \mathrm{y})$$

The semiring properties are then sufficient to guarantee that the axioms of a valuation algebra are satisfied. Thus, for any semiring A, the algebra (Φ_A, D) is a valuation algebra. If ϕ and ψ are semiring valuations on s and t respectively, then the combination law of the semiring valuation algebra takes the following form; if x is an s-tuple and $s \subseteq u \subseteq s \cup t$,

$$\sum_{\mathrm{z} \in X^{t\setminus u}} \phi(\mathrm{x}) \times \psi(\mathrm{x}^{\downarrow t \cap u}, \mathrm{z}) = \phi(\mathrm{x}) \times \sum_{\mathrm{z} \in X^{t\setminus u}} \psi(\mathrm{x}^{\downarrow t \cap u}, \mathrm{z})$$

Table 1. Semiring example catalog

A	+	×	
$\mathbb{R}_{\geq 0}$	+	·	Arithmetic semiring
$\mathbb{R} \cup \{\pm\infty\}$	max	min	Bottleneck semiring
$\mathbb{B} = \{0, 1\}$	$\vee$	$\wedge$	Boolean semiring
$\mathbb{N} \cup \{0, \infty\}$	min	+	Tropical semiring
$\mathbb{R} \cup \{-\infty\}$	max	+	Tropical semiring
[0, 1]	max	t-norm	T-norm semiring

This method follows directly from the distributive law in semirings. If ϕ is a semiring valuation and $t \subseteq s \subseteq d(\phi) = u$, then the transitivity of projection in the semiring valuation algebra becomes

$$\sum_{\mathrm{z} \in X^{s\setminus t}} \left(\sum_{\mathrm{y} \in X^{u\setminus s}} \phi(\mathrm{x}, \mathrm{y}, \mathrm{z}) \right) = \sum_{\mathrm{y} \in X^{u\setminus s}, \, \mathrm{z} \in X^{s\setminus t}} \phi(\mathrm{x}, \mathrm{y}, \mathrm{z})$$

This result is a consequence of the associativity and commutativity of addition.

In Table 1, a couple of semiring examples are given. The example of the first section is based on the arithmetic semiring. This example yields the valuation algebra underlying probability networks, in particular Bayesian networks, and shows that the problem of computing $\alpha(X_1, X_2)$ and $\beta(X_2, X_3)$ in the first section are indeed instances of the general projection problem in valuation algebras. The Boolean semiring induces the valuation algebra for classic constraint systems and the tropical semirings model optimization problems. Moreover, taking maximization for addition and an arbitrary *t-norm* for multiplication leads always to a semiring. A t-norm (triangular norm) is a binary operation on the unit interval that is associative, commutative, and nondecreasing in both arguments. The corresponding t-norm semirings induce valuation algebras used in maximum likelihood estimation in general, and in particular in decoding theory. But they also cover possibility theory and fuzzy set theory.

LOCAL COMPUTATION

In this section, an efficient algorithm for the projection problem [Equation (4)] in a valuation algebra is described. It uses the concept of a join (or junction) tree. A labeled tree $T = (V, E, \lambda)$ with nodes V, edges $E \subseteq V \times V$, and with labels $\lambda : V \to D$ is called a *join tree*, if

$$\lambda(u) \cap \lambda(v) \subseteq \lambda(w)$$

for every node w on the path between nodes u and v in the tree T. A join tree is *covering* a projection problem [Equation (4)], if for every data domain s_i, there is a node $v \in V$ such that $s_i \subseteq \lambda(v)$, and similarly for every goal domain x_j, there is also a node $u \in V$ such that $x_j \subseteq \lambda(u)$. In this case, we may assign any data domain s_i to a covering node, that is, we can define a mapping $a : \{1, \ldots, m\} \to V$ such that $s_i \subseteq \lambda(a(i))$. If several nodes cover s_i, then we select arbitrarily one of them. Similarly, we can define a mapping $b : \{1, \ldots, n\} \to V$, such that $x_j \subseteq \lambda(b(j))$. Then define for any node $v \in V$

$$\psi(v) = \bigotimes_{i:a(i)=v} \phi_i$$

If no index i exists such that $a(i) = v$, then $\psi(v) = e$, the adjoined identity element (introduced in the second section). Then it holds that

$$\phi_1 \otimes \cdots \otimes \phi_m = \underset{v \in V}{\otimes} \psi(v)$$

Now, an efficient algorithm for solving the projection problem for a selected and fixed goal domain x_j can be formulated: Fix the node $b(j)$ and orient all edges of the covering join tree toward it. Then, every node v, except $b(j)$, has a unique neighbor (child) $ch(v)$ toward $b(j)$. Imagine a storage location associated with each node v and describe its content by $\sigma(v)$. Initially set $\sigma(v) := \psi(v)$ for all $v \in V$.

The algorithm can now be described by the following two rules:

1. Each node v sends a message $\mu(v) := \sigma(v)^{\downarrow d(\psi(v)) \cap \lambda(ch(v))}$ to its neighbor $ch(v)$, when it has received all messages from its parents. Leaf nodes can send messages right away.
2. The receiving node $ch(v)$ updates its storage by $\sigma(ch(v)) := \sigma(ch(v)) \otimes \mu(v)$.

When the root node $b(j)$ has received all messages, the algorithm stops. The solution of the projection problem for the goal domain x_j is then obtained as $\sigma(b(j))^{\downarrow x_j}$. The efficiency of this algorithm stems from the fact that never a combination or projection operation takes place on a domain larger than the domains $\lambda(v)$ of the covering join tree. These domains are usually much smaller than the combined domain $s_1 \cup \ldots \cup s_m$ of the combination $\phi_1 \otimes \cdots \otimes \phi_m$, which occurrs in the naive solution of the projection problem.

As an example, reconsider Equations (2) and (3) for an efficient computation of $\alpha(X_1, X_2)$ and $\beta(X_2, X_3)$, respectively. It turns out that these computations correspond to possible runs of the algorithm above. For an illustration we refer to Fig. 1, which the messages sent among the nodes of the covering join trees are depicted as arrows.

It may be noted that this algorithm allows for parallel and distributed computing. The projection problem for other goal domains x_i different from x_j are obtained in the same way by selecting $b(i)$ as root node. This method involves reorienting certain edges with respect to the first computation. But edges can remain unchanged, as shown in Fig. 1. This result means that the messages on these edges can be reused. In summary, by caching the computations, the solution of several projection problems can be optimized further considerably. Several architectures are known for this (1, 2). Some of them exploit properties of the underlying valuation algebra, such as the possibility of division.

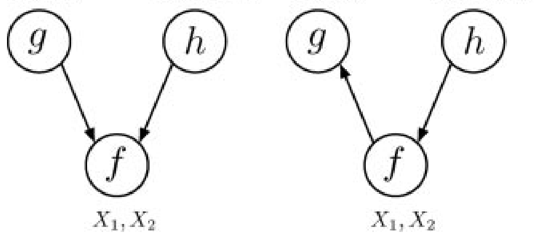


Figure 1. Graphical representation of the efficient computation of $\alpha(X_1, X_2)$ (on the left) and $\beta(X_2, X_3)$ (on the right) as given in Equations (2) and (3), respectively.

IDEMPOTENCY

An information in the usual sense of the word does not change if it is repeated. That is, an idempotency law such as $\phi \otimes \phi = \phi$ holds. More precisely, a piece of information combined with a part of itself should give nothing new. This information can be formulated in an additional axiom:

6. *Idempotency of combination*: If $d(\phi) = s$ and $t \subseteq s$, then

$$\phi \otimes \phi^{\downarrow t} = \phi$$

A valuation algebra with this additional axiom is called an *information algebra*. It has many interesting properties. First, especially efficient architectures of local computation exist for information algebras (1, 2). Second, in information algebras, an *information order* can be introduced by defining $\phi \leq \psi$, reading ϕ is less informative than ψ, if $\phi \otimes \psi = \psi$. This method puts information algebra in relation to domain theory, and the related theory of information systems (3, 4).

Relational algebra from relational database theory is a prototype of an information algebra. Furthermore, many logic systems induce information algebras, which includes classic propositional and predicate logic. Systems of linear equalities or inequalities generate associated information algebras, as well as certain semiring valuation algebras, for instance the constraints systems that arise from to the Boolean semiring, and many more.

BIBLIOGRAPHY

1. J. Kohlas, *Information Algebras: Generic Structures for Inference*. Berlin: Springer-Verlag, 2003.
2. C. Schneuwly, M. Pouly, and J. Kohlas. *Local Computation in Covering Join Trees*. Technical Report 04-16. Department of Informatics, University of Fribourg.
3. B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*. Cambridge UK: Cambridge University Press, 1990.
4. D. S. Scott, Outline of a mathematical theory of computation, *Proc. of the 4th Annual Princeton Conference On Information Science and Systems*, 1970, pp. 169–176.

FURTHER READING

J. Kohlas and P. P. Shenoy, *Computation in Valuation Algebras*. Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 5: Algorithms for Uncertainty and Defeasible Reasoning. The Netherlands: Dordrecht, Kluwer, 2000.

J. Kohlas and N. Wilson, *Semiring Induced Valuation Algebras: Exact and Approximate Local Computation Algorithms*. New York: Elsevier Science, 2006.

C. Schneuwly, Computing in valuation algebras. PhD Thesis, Department of Informatics, Fribourg, Switzerland: University of Fribourg, 2007.

M. Pouly, A generic framework for local computation. PhD Thesis, Department of Informatics, Fribourg, Switzerland: University of Fribourg, 2008.

P. P. Shenoy and G. Shafer, Axioms for probability and belief-function proagation. *Uncertainty in Artificial Intelligence 4. Series: Machine intelligence and pattern recognition*. R. D. Shachter, T. S. Levitt, L. N. Kanal, J. F. Lemmer.

S. M. Aji and R. J. McEliece, The generalized distributive law, *IEEE Trans. Informat. Theory*, **46**(2): 325–343, 2000.

JÜRG KOHLAS
MARC POULY
CESAR SCHNEUWLY
University of Fribourg
Fribourg, Switzerland

L

LINEAR AND NONLINEAR PROGRAMMING

INTRODUCTION

An *optimization problem* is a mathematical problem in which data are used to find the values of n variables so as to minimize or maximize an overall objective while satisfying a finite number of imposed restrictions on the values of those variables. For example:

Linear Program (LP1)		Nonlinear Program (NLP1)	
maximize $x_1 + x_2$		minimize $-3x_1 - 2x_2$	
subject to $x_1 + 3x_2 \leq 6$	(a)	subject to $x_1^2 + x_2^2 \leq 9$	(a)
$2x_1 + x_2 \leq 4$	(b)	$x_1^2 - 6x_1 + x_2^2 \leq 6$	(b)
$x_1 \geq 0$	(c)	x_1 unrestricted	
$x_2 \geq 0$	(d)	$x_2 \geq 0$	(c)

The expression being maximized or minimized is called the *objective function*, whereas the remaining restrictions that the variables must satisfy are called the *constraints*. The problem on the left is a *linear program* (*LP*) because the objective function and all constraints are *linear* and all variables are *continuous*; that is, the variables can assume any values, including fractions, within a given (possibly infinite) interval. In contrast, the problem on the right is a *nonlinear program* (*NLP*) because the variables are continuous but the objective function or at least one constraint is not linear. Applications of such problems in computer science, mathematics, business, economics, statistics, engineering, operations research, and the sciences can be found in Ref. 1.

LINEAR PROGRAMMING

The geometry of LP1 is illustrated in Fig. 1(a). A *feasible solution* consists of values for the variables that simultaneously satisfy all constraints, and the *feasible region* is the set of all feasible solutions. The *status* of every LP is always one of the following three types:

- **Infeasible**, which means that the feasible region is empty; that is, there are no values for the variables that simultaneously satisfy all of the constraints.
- **Optimal**, which means that there is an **optimal solution**; that is, there is a feasible solution that also provides the best possible value of the objective function among all feasible solutions. (The optimal solution for LP1 shown in Fig. 1(a) is $x_1 = 1.2$ and $x_2 = 1.6$.)
- **Unbounded**, which means that there are feasible solutions that can make the objective function as large (if maximizing) or as small (if minimizing) as desired.

Note that the feasible region in Fig. 1(a) has a finite number of *extreme points* (the black dots), which are feasible points where at least n of the constraints hold with equality. Based on this observation, in 1951, George Dantzig (2) developed the *simplex algorithm (SA)* that, in a finite number of arithmetic operations, will determine the status of any LP and, if optimal, will produce an optimal solution. The SA is the most commonly used method for solving an LP and works geometrically as follows:

Step 0 (Initialization). Find an initial extreme point, say a vector $\mathbf{x} = (x_1, \ldots x_n)$, or determine none exists; in which case, stop, the LP is infeasible.

Step 1 (Test for Optimality). Perform a relatively simple computation to determine whether $\mathbf{x}$ is an optimal solution and, if so, stop.

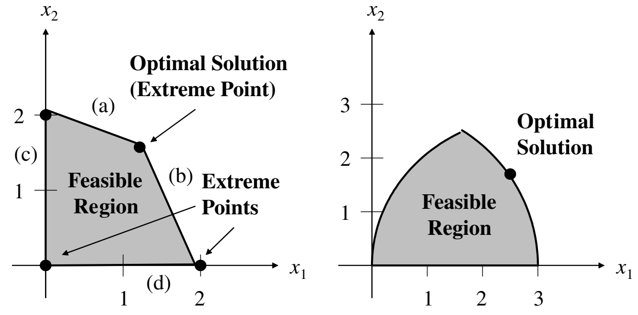
Step 2 (Move to a Better Point). If $\mathbf{x}$ is not optimal, use that fact to

- **(a) (Direction of Movement).** Find a direction in the form of a vector $\mathbf{d} = (d_1, \ldots d_n)$ that points from $\mathbf{x}$ along an *edge* of the feasible region so that the objective function value improves as you move in this direction.
- **(b) (Amount of Movement).** Determine a real number t that represents the maximum amount you can move from $\mathbf{x}$ in the direction $\mathbf{d}$ and stay feasible. If $t = \infty$, stop, the LP is unbounded.
- **(c) (Move).** Move from $\mathbf{x}$ to the new extreme point $\mathbf{x} + t\mathbf{d}$, and return to Step 1.

The translation of these steps to algebra constitutes the SA. Because the SA works with algebra, all inequality constraints are first converted to equality constraints. To do so, a nonnegative *slack variable* is added to (subtracted from) each $\leq$ ($\geq$) constraint. For given feasible values of the original variables, the value of the slack variables represents the difference between the left and the right sides of the inequality constraint. For example, the constraint $x_1 + 3x_2 \leq 6$ in LP1 is converted to $x_1 + 3x_2 + s_1 = 6$, and if $x_1 = 2$ and $x_2 = 0$, then the value of the slack variable is $s_1 = 4$, which is the difference between the left side of $x_1 + 3x_2$ and the right side of 6 in the original constraint. The algebraic analog of an extreme point is called a *basic feasible solution*.

Step 0 of the SA is referred to as *phase 1*, whereas Steps 1 and 2 are called *phase 2*. Phase 2 is finite because there are a finite number of extreme points and no such point visited by the SA is repeated because the objective function strictly improves at each iteration, provided that $t > 0$ in Step 2(b) (even in the *degenerate* case when $t = 0$, the algorithm can be made finite). Also, a finite procedure for phase 1 is to use the SA to solve a special *phase 1 LP* that involves only phase 2. In summary, the SA is finite.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.



(a) The Geometry of LP1 **(b) The Geometry of NLP1**

Figure 1. The geometry of LP1 and NLP1.

See Ref. 3 for improvements in the computational efficiency and numerical accuracy of the SA. Many commercial software packages use the SA, from Solver, an add-in to a Microsoft (Redmond, WA) Excel spreadsheet, to CPLEX (ILOG, Mountain View, CA) and LINDO (LINDO Systems, Inc., Chicago, IL), stand-alone professional codes for solving large real-world problems with many thousands of variables and constraints. These packages provide the optimal solution and other economic information used in *postoptimality and sensitivity analysis* to answer questions of the form, “What happens to the optimal solution and objective function value if some of the data change?” Such questions are asked because the data are often estimated rather than known precisely. Although you can always change data and resolve the problem, you can sometimes answer these questions without having to use a computer when only one objective function coefficient or one value on the right-hand side (rhs) of a constraint is changed.

Much postoptimality analysis is based on *duality theory*, which derives from the fact that associated with every original *primal LP* having n variables and m constraints is another *dual LP* that has m variables and n constraints. Where the primal problem is to be maximized (minimized), the dual problem is to be minimized (maximized), for example:

Primal LP

$$\begin{array}{ll} \text{min} & c_1x_1 + \cdots + c_nx_n \\ \text{s.t.} & a_{11}x_1 + \cdots + a_{1n}x_n \geq b_1 \\ & \vdots \quad \vdots \quad \vdots \quad \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \geq b_m \\ & x_1 \ , \cdots \ , \quad x_n \geq 0 \end{array}$$

Dual LP

$$\begin{array}{ll} \text{max} & b_1u_1 + \cdots + b_mu_m \\ \text{s.t.} & a_{11}u_1 + \cdots + a_{m1}u_m \leq c_1 \\ & \vdots \quad \vdots \quad \vdots \quad \vdots \\ & a_{1n}u_1 + \cdots + a_{mn}u_m \leq c_n \\ & u_1 \ , \cdots \ , \quad u_n \geq 0 \end{array}$$

Duality theory is used in the following ways:

1. As a test for optimality in Step 1 of the SA. (If $\mathbf{x}$ is feasible for the primal LP and $\mathbf{u}$ is feasible for the dual LP, and the objective function values of the primal and dual at $\mathbf{x}$ and $\mathbf{u}$ are equal, then $\mathbf{x}$ and $\mathbf{u}$ are optimal solutions for their respective problems.)
2. To determine the status of an LP without having to use a computer. (For example, if the primal LP is feasible and the dual LP is infeasible, then the primal LP is unbounded.)
3. To provide economic information about the primal LP. (For example, the optimal value of a dual variable, also called the *shadow price* of the associated primal constraint, represents the change in the optimal objective function value of the primal LP per unit of increase in the right-hand side of that constraint—within a certain range of values around that right-hand side—if all other data remain unchanged.)
4. To develop finite algorithms for solving the dual LP and, in so doing, provide a solution to the primal LP, assuming that both problems have optimal solutions.

Although efficient for solving most real-world problems, many versions of the SA can, in the worst case, visit an exponential (in terms of n) number of extreme points (see Ref. 4 for the first such example). It is an open question as to whether there is a version of the SA that, in the worst case, is polynomial (in the size of the data) although Borgwardt (5) proved that a version of the SA is polynomial on average. In 1979, Khachian (6) proposed the first polynomial algorithm for solving every LP. Karmarkar (7) subsequently developed the first polynomial algorithm that was also efficient in practice. These *interior point methods* (*IPM*) go through the inside of the feasible region [see Fig. 1(a)] rather than along the edges like the SA does. Computational experience with current versions of these IPMs indicate that they can be more efficient than the SA for solving specially-structured large LPs but generally are not more efficient than the SA for other LPs. See Refs. 8 and 9 for a discussion of IPMs and Ref. 10 for linear programming and the SA.

NONLINEAR PROGRAMMING

An NLP differs from an LP in both the problem structure and the ability to obtain solutions. Although an LP always has constraints (otherwise, the LP is unbounded), an NLP can be *unconstrained*—that is, have no constraints—or *constrained*, for example, find $\mathbf{x} = (x_1, \ldots, x_n)$ to:

Unconstrained Problem (NLP2)

$$\text{minimize} \quad f(\mathbf{x})$$

Constrained Problem (NLP3)

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & g_1(\mathbf{x}) & \leq \ 0 \\ & \vdots & \vdots \ \ \vdots \\ & g_m(\mathrm{x}) & \leq \ 0 \end{array}$$

Other differences between an LP and an NLP arise due to nonlinearities in the objective function and constraints of an NLP. Where an LP must be either infeasible, optimal, or unbounded, with a nonlinear objective function, an NLP can additionally approach, but never attain, its smallest possible objective function value. For example, when minimizing $f(x) = e^{-x}$, the value of f approaches 0 as x approaches $+\infty$ but never attains the value 0.

The linear objective function and constraints in an LP allow for a computationally efficient test for optimality to determine whether a given feasible solution $\mathbf{x} = (x_1, \ldots, x_n)$ is optimal. In contrast, given a feasible point $\mathbf{x}$ for an NLP with general nonlinear functions, there is no known efficient test to determine whether $\mathbf{x}$ is a *global minimum of an NLP*, that is, a point such that for every feasible point $\mathbf{y} = (y_1, \ldots, y_n)$, $f(\mathbf{x}) \leq f(\mathbf{y})$. There is also no known efficient test to determine whether $\mathbf{x}$ is a *local minimum*, that is, a feasible point such that there is a real number $\delta > 0$ for which every feasible point $\mathbf{y}$ with $\|\mathbf{x} - \mathbf{y}\| < \delta$ satisfies $f(\mathbf{x}) \leq f(\mathbf{y})$ [where $\|\mathbf{x} - \mathbf{y}\| = \sum_{i=1}^{n} (x_i - y_i)^2$].

In the absence of an efficient test for optimality, many NLP algorithms instead use a *stopping rule* to determine whether the algorithm should terminate at the current feasible point $\mathbf{x}$. As described in the next two sections, these stopping rules are typically necessary conditions for $\mathbf{x}$ to be a local minimum and usually have the following desirable properties:

- It is computationally practical to determine whether the current feasible solution $\mathbf{x}$ satisfies the stopping rule.
- If $\mathbf{x}$ does not satisfy the stopping rule, then it is computationally practical to find a direction in which to move from $\mathbf{x}$, stay feasible at least for a small amount of movement in that direction, and simultaneously improve the objective function value.

Be aware that if $\mathbf{x}$ is a feasible point for an NLP that satisfies the stopping rule, this *does not* mean that $\mathbf{x}$ is a global minimum (or even a local minimum) of the NLP. Additional conditions on the nonlinear functions—such as *convexity*—are required to ensure that $\mathbf{x}$ is a global minimum. Indeed, it is often the case that nonconvex functions make it challenging to solve an NLP because there can be many local minima. In such cases, depending on the initial values chosen for the variables, algorithms often terminate at a local minimum that is not a global minimum. In contrast, due to the (convex) linear functions in an LP, the simplex algorithm always terminates at an optimal solution if one exists, regardless of the initial starting feasible solution.

A final difficulty caused by nonlinear functions is that algorithms for attempting to solve such problems are usually not finite (unlike the simplex algorithm for LP). That is, NLP algorithms usually generate an infinite sequence of feasible values for the variables, say $X = (\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots)$. Under fortunate circumstances, these points will *converge*, that is, get closer to specific values for the variables, say $\mathbf{x}^*$, where $\mathbf{x}^*$ is a local or global minimum for the NLP, or at least satisfies the stopping rule. In practice, these algorithms are made to stop after a finite number of iterations with an approximation to $\mathbf{x}^*$.

In the following discussion, details pertaining to developing such algorithms are presented for the unconstrained NLP2 and the constrained NLP3, respectively. From here on, it is assumed that the objective function f and all constraint functions $g_1, \ldots, g_m$ are differentiable. See Refs. 11 and 12 for a discussion of *nondifferentiable optimization*, also called *nonsmooth optimization*.

Unconstrained Nonlinear Programs

The unconstrained problem NLP2 differs from an LP in that (*1*) the objective function of NLP2 is not linear and (*2*) every point is feasible for NLP2. These differences lead to the following algorithm for attempting to solve NLP2 by modifying the simplex algorithm described previously:

Step 0 (Initialization). Start with any values, say $\mathbf{x} = (x_1, \ldots, x_n)$, for NLP2 because NLP2 has no constraints that must be satisfied.

Step 1 (Stopping Rule). Stop if the current point $\mathbf{x} = (x_1, \ldots, x_n)$ for NLP2 satisfies the following *stopping rule*, which uses the partial derivatives of f and is a necessary condition for $\mathbf{x}$ to be a local minimum:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{x}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{x})\right) = (0, \ldots, 0).$$

Note that finding a point $\mathbf{x}$ where $\nabla f(\mathbf{x}) = (0, \ldots, 0)$ is a problem of solving a system of n nonlinear equations in the n unknowns $x_1, \ldots, x_n$, which can possibly be done with Newton's method or some other algorithm for solving a system of nonlinear equations.

Step 2 (Move to a Better Point). If $\nabla f(\mathbf{x}) \neq (0, \ldots, 0)$, use that fact to

(a) (Direction of Movement). Find a *direction of descent* $\mathbf{d}$ so that all small amounts of movement from $\mathbf{x}$ in the direction $\mathbf{d}$ result in points that have smaller objective function values than $f(\mathbf{x})$. (One such direction is $\mathbf{d} = -\nabla f(\mathbf{x})$, but other directions that are usually computationally more efficient exist.)

(b) (Amount of Movement). Perform a possibly infinite algorithm, called a *line search*, that may or may not be successful, to find a value of a real number $t > 0$ that provides the smallest value of $f(\mathbf{x} + t\mathbf{d})$. If $t = \infty$, stop without finding an optimal solution to NLP2.

(c) (Move). Move from $\mathbf{x}$ to the new point $\mathbf{x} + t\mathbf{d}$, and return to Step 1.

The foregoing algorithm may run forever, generating a sequence of values for the variables, say $X = (\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots)$. Under fortunate circumstances, these points will converge to specific values for the variables, say $\mathbf{x}^*$, where $\mathbf{x}^*$

is a local or global minimum for NLP2 or at least satisfies $\nabla f(\mathbf{x}) = (0, \ldots, 0)$.

Constrained Nonlinear Programs

Turning now to constrained NLPs, as seen by comparing Fig. 1(a) and (b), the feasible region of a constrained NLP can be different from that of an LP and the optimal solution need not be one of a finite number of extreme points. These differences lead to the following algorithm for attempting to solve NLP3 by modifying the simplex algorithm described previously:

Step 0 (Initialization). There is no known finite procedure that will find an initial feasible solution for NLP3 or determine that NLP3 is infeasible. Thus, a possibly infinite algorithm is needed for this step and, under favorable circumstances, produces a feasible solution.

Step 1 (Stopping Rule). Stop if the current feasible solution $\mathbf{x} = (x_1, \ldots, x_n)$ for NLP3 satisfies the following *stopping rule*: Real numbers $u_1, \ldots, u_m$ exist such that the following **Karush–Kuhn–Tucker (KKT) conditions** hold at the point **x** (additional conditions on *f* and g_i, such as convexity, are required to ensure that **x** is a global minimum):

(a) (feasibility) For each $i = 1, \ldots, m, g_i(\mathbf{x}) \leq 0$ and $u_i \geq 0$.

(b) (complementarity) For each $i = 1, \ldots, m$, $u_i g_i(\mathbf{x}) \leq 0$.

(c) (gradient condition) $\nabla f(\mathbf{x}) + \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}) = (0, \ldots, 0)$.

Note that when there are no constraints, the KKT conditions reduce to $\nabla f(\mathbf{x}) = (0, \ldots, 0)$. Also, when *f* and g_i are linear, and hence NLP3 is an LP, the values $\mathbf{u} = (u_i, \ldots, u_m)$ that satisfy the KKT conditions are optimal for the dual LP. That is, conditions (a) and (c) of the KKT conditions ensure that **x** is feasible for the primal LP and **u** is feasible for the dual LP, whereas conditions (b) and (c) ensure that the objective function value of the primal LP at **x** is equal to that of the dual LP at **u**. Hence, as stated in the first use of duality theory in the "Linear Programming" section, **x** is optimal for the primal LP and **u** is optimal for the dual LP.

Step 2 (Move to a Better Point). If **x** is not a KKT point, use that fact to

(a) (Direction of Movement). Find a *feasible direction of improvement***d** so that all small amounts of movement from **x** in the direction **d** result in feasible solutions that have smaller objective function values than *f*(**x**).

(b) (Amount of Movement). Determine the maximum amount t^* you can move from **x** in the direction **d**, and stay feasible. Then perform a possibly infinite algorithm, called a *constrained line search*, in an attempt to find a value of *t* that minimizes $f(\mathbf{x} + t\mathbf{d})$ over the interval $[0, t^*]$. If $t = \infty$, stop without finding an optimal solution to NLP3.

(c) (Move). Move from **x** to the new feasible point $\mathbf{x} + t\mathbf{d}$, and return to Step 1.

The foregoing algorithm may run forever, generating a sequence of values for the variables, say $X = (\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots)$. Under fortunate circumstances, these points will converge to specific values for the variables, say $\mathbf{x}^*$, where $\mathbf{x}^*$ is a local or global minimum for NLP3 or a KKT point.

In addition to the foregoing *methods of feasible directions*, many other approaches have been developed for attempting to solve NLP2 and NLP3, including conjugate gradient methods, penalty and barrier methods, sequential quadratic approximation algorithms, and fixed-point algorithms (see Ref. 13 for details). A discussion of interior point methods for solving NLPs can be found in Ref. 14. Commercial software packages from Solver in Excel and GRG exist for attempting to solve such problems and can currently handle up to about 100 variables and 100 constraints. However, if an NLP has special structure, it may be possible to develop an efficient algorithm that can solve substantially larger problems.

Topics related to LP and NLP include integer programming (in which the value of one or more variable must be integer), network programming, combinatorial optimization, large-scale optimization, fixed-point computation, and solving systems of linear and nonlinear equations.

BIBLIOGRAPHY

1. H. P. Williams, *Model Building in Mathematical Programming*, 4th ed., New York: Wiley, 1999.
2. G. B. Dantzig, Maximization of a linear function of variables subject to linear inequalities, in T. C. Koopmans (ed.), *Activity Analysis of Production and Allocation*. New York: Wiley, 1951.
3. R. E. Bixby, Solving real-world linear programs: A decade and more of progress, *Oper. Res.*, **50**(1): 3–15, 2002.
4. V. Klee and G. J. Minty, How good is the simplex algorithm? O. Shisha (ed.), *Inequalities III*, New York: Academic Press, 1972.
5. K. H. Borgwardt, Some distribution independent results about the assymptotic order of the average number of pivot steps in the simplex algorithm, *Math. Oper. Res.*, **7**: 441–462, 1982.
6. L. G. Khachian, Polynomial algorithms in linear programming (in Russian), *Doklady Akademiia Nauk SSSR*, **244**: 1093–1096, 1979; English translation: *Soviet Mathematics Doklady*, **20**: 191–194.
7. N. Karmarkar, A new polynomial-time algorithm for linear programming, *Combinatorica*, **4**: 373–395, 1984.
8. C. Roos, T. Terlaky, and J. P. Vial, *Theory and Algorithms for Linear Optimization: An Interior Point Approach*, New York: Wiley, 1997.
9. Stephen J. Wright, *Primal-dual Interior-Point Methods*, Society for Industrial and Applied Mathematics, Philadelphia, 1997.
10. C. M. S. Bazaraa, J. J. Jarvis, and H. Sherali, *Linear Programming and Network Flows*, 3rd ed., New York: Wiley, 2004.
11. G. Giorgi, A. Guerraggio, and J. Thierfelder, *Mathematics of Optimization: Smooth and Nonsmooth Case*, 1st ed., Amsterdam: Elsevier, 2004.

12. D. Klatte and B. Kummer, *Nonsmooth Equations in Optimization—Regularity, Calculus, Methods and Applications*, Dordrecht: Kluwer Academic, 2002.

13. C. M. S. Bazaraa, H. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 2nd ed., New York: Wiley, 1993.

14. T. Terlaky (ed.), *Interior Point Methods of Mathematical Programming*, Dordrecht, The Netherlands: Kluwer Academic, 1996.

Daniel Solow
Case Western Reserve
University
Cleveland, Ohio

LOCALITY PRINCIPLE

Locality is a universal behavior of all computational processes: They tend to refer repeatedly to subsets of their resources over extended time intervals. System designers have exploited this behavior to optimize performance in numerous ways, which include caching, clustering of related objects, search engines, organization of database, spam filters, and forensics.

Every executing computation generates references to objects, such as memory pages, disk sectors, database records, and web pages. These references are not uniform: Some objects are referenced more often than others, and references to each object come in bursts. Another way to say this is that execution of a computation consists of a series of phases; phase i has holding time Ti and locality set Li. The locality set is the set of objects referenced in the phase. A particular object is referenced only in the phases in which it is a member of the locality set. Thus, the history of the computation appears as a sequence,

$$(L1, T2), (L2, T2), (L3, T3), \ldots, (Li, Ti), \ldots$$

The locality set of a multithreaded computation at a particular time is the union of the individual thread locality sets at that time.

Knowledge of a computation's locality behavior has several significant benefits:

- The local storage of a processor (cache) needs to contain only the current localities, not the entire object space. The cache provides in significant savings in local storage without loss of performance.
- If the phase boundaries are unknown (the usual case), the best predictor of objects the computation will use in the immediate future is its current locality set.
- Objects that tend to be in the same locality sets can be grouped in storage systems so that they can be loaded together (efficiently) into a processor's cache.

There are two aspects of locality: *(1) temporal locality* means that references to the same objects are grouped in time, and *(2) spatial locality* means that objects close to each other tend to be referenced together. These two aspects give the phase-transition definition above.

Locality is among the oldest systems principles in computer science. It was discovered in 1966 during efforts to make early virtual memory systems work well. Working-set memory management was the first exploitation of this principle; it prevented thrashing while maintaining near optimal system throughput, and eventually it enabled virtual memory systems to be reliable, dependable, and transparent. Today the locality principle is being applied to computations that adapt to the neighborhoods in which users are situated, inferring those neighborhoods by observing user actions, and then optimizing performance for users.

The remainder of this article reviews the history of the locality principle and its new applications in context-aware computing.

MODELS OF LOCALITY

A model of locality is a description of a mechanism to generate the locality behavior of a computation without having to run the computation. The earliest notion of locality was a nonuniform distribution of references over a computation's objects. Thus, the objects could be ordered so that their probabilities of use follow the relation

$$p1 > p2 > p3 > \ldots > pk > \ldots$$

When these probabilities are measured, they often follow a Zipf Law, which means that pk is proportional to $1/k$.

This law is called a *static representation* of locality because a single distribution of probabilities holds for all time; there is no differentiation into phases. Empirically, when a computation is modeled this way, the model overestimates the average locality size by factors of 2 or 3.

In contrast are *dynamic representations* that recognize phases and allow for different probability distributions in each phase. Dynamic models tend to estimate average locality size well.

The phase-transition model is a successful dynamic model. It consists of a *macromodel* that generates phase and transition intervals and their holding times, and a *micromodel* that generates references from a locality set associated with the phase. The holding times in the states are random variables, with the average holding in the phase state being much longer than in the transition state. While in the phase state, the model uses a static representation for a single locality set, such as the distribution above. During the transition phase, the model allows for random references to all objects (1).

The working set model defines a program's working set at time t, $W(t,T)$, as the set of objects referenced in the time window of length T extending backward from the current time t. It is usually possible to choose the window size T so that it is contained within phases most of time, in which case the working set measures the current locality set. Thus, the working set is a good way to track the localities and phases of a program dynamically (2).

HISTORY

Locality was discovered from efforts to make virtual memory systems work well. Virtual memory was first developed in 1959 on the Atlas system at the University of Manchester (3). Its superior programming environment doubled or tripled programmer productivity. Its automatic caching boosted performance (4). But its finicky performance was

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

sensitive to the choice of replacement algorithm and to the ways compilers grouped code on to pages. Worse, when it was coupled with multiprogramming, it was prone to thrashing, the near-complete collapse of system throughput because of heavy paging (5). The locality principle guided the design of robust replacement algorithms, compiler code generators, and thrashing-proof systems. It transformed virtual memory from an unpredictable to a robust technology that regulated itself dynamically and optimized throughput without user intervention.

Atlas included the first demand-paged virtual memory, which automated the process of transferring pages between random access memory (RAM) and disk. The designers grappled with two performance problems, either one of which could scuttle the system: One was addresses to locations, the other pages already loaded into RAM. They quickly found a solution to the translation problem using page tables stored in RAM and address caches in the central processing unit (CPU). The replacement problem was much more difficult.

Because the disk access time was about 10,000 times slower than the CPU instruction cycle, each page fault added a significant delay to a job's completion time. Therefore, minimizing page faults was critical to system performance. The ideal page to replace from main memory is the one that will not be used again for the longest time. But next-use time cannot be known with certainty. The Atlas system used a "learning algorithm" that hypothesized a loop cycle for each page, measured each page's period, and estimated which page was not needed for the longest time.

The learning algorithm was controversial. It performed well on programs with well-defined loops and poorly on many other programs. In numerous experimental studies well into the 1960s, researchers sought to determine what replacement rules might work best over the widest possible range of programs. Belady's study in 1966 (6) was the most comprehensive and scientific. Eventually, it became apparent that the volatility resulted from variations in compiling methods: The way in which a compiler grouped code blocks onto pages strongly affected the program's performance under a given replacement strategy (7).

The major computer makers were drawn to multiprogrammed virtual memory because of its superior programming environment. RCA, General Electric, Burroughs, and Univac all included virtual memory in their operating systems of the mid-1960s. Because a bad replacement algorithm could cost a million dollars of lost machine time over the life of a system, they all had a keen interest in finding good replacement algorithms.

Imagine their chagrin when by 1966 these companies reported a new, unexplained, catastrophic problem they called thrashing. It was a sudden collapse of throughput as the multiprogramming level rose. It had nothing to do with the choice of replacement policy. A thrashing system spent most of its time resolving page faults and little running the CPU. Thrashing was far more damaging than a poor replacement algorithm. It scared the daylights out of the computer makers.

IBM avoided these uncertainties by excluding virtual memory from its OS360 in 1964. Instead, it sponsored at its Watson laboratory one of the most comprehensive experimental systems projects of all time. Led by Bob Nelson, Les Belady, and David Sayre, the project team built the first virtual-machine operating system and used it to study the performance of virtual memory. (The term "virtual memory" seems to have come from this project.) By 1966 they had tested every replacement policy that anyone had ever proposed and a few more they invented. Many of their tests involved the use bits built into page tables. By periodically scanning and resetting the bits, the replacement algorithm distinguishes recently referenced pages from others. Belady concluded that policies that favor recently used pages performed better than other policies; least recently used replacement was consistently the best performer among those tested (6). He said that this resulted from reference clustering locality behavior. His colleagues verified that many programs exhibited locality behavior (6).

At MIT Project MAC in 1966, Peter Denning hypothesized that the controversies over replacement algorithms could be settled by defining an intrinsic memory demand: "This process needs p pages at time t." Intrinsic demand was the first notion of "working set." Individual replacement algorithms could then be rated by their abilities to detect working sets. Inspired by Belady's concept of locality, Denning formally defined a process's working set as the set of pages used during a fixed-length sampling window in the immediate past (2). A working set could be measured by periodically reading and resetting the use bits in a page table.

This method solved thrashing because if every process is guaranteed its working set, then no process can overload the paging disk and system throughput can be maintained (5). Thrashing is impossible for a working-set policy. Experiments with real operating systems confirmed that this policy gives high efficiency and prevents thrashing (9).

The working-set idea worked because the pages observed in the backward window were highly likely to be used again in the immediate future. Was this assumption justified? The idea that reference behavior could be described as a sequence of phases and locality sets seemed natural because programmers planned overlays using diagrams that showed subsets and time phases (Fig. 1). But what was strikingly interesting was that programs showed this behavior even when it was not explicitly preplanned. Each program had its own distinctive usage pattern, like a voiceprint (Fig. 2).

Two effects could make this happen: *(1)* temporal clustering caused by looping and executing within modules with private data and *(2)* spatial clustering caused by related values being grouped into arrays, sequences, modules, and other data structures. Both these reasons followed from the human practice of 'divide and conquer'—breaking a large problem into parts and working separately on each part. The locality bit maps captured someone's problem-solving method in action. The working set measures an approximation of a program's intrinsic locality sequence.

A distance function gives a single measure of temporal and spatial locality. $D(x,t)$ measures the distance from the current execution point of the process to an object x at time t.

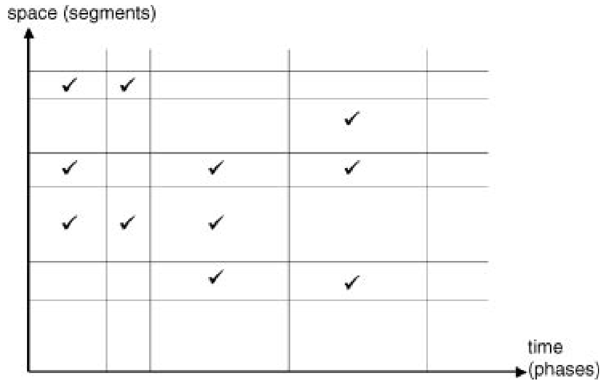


Figure 1. Locality sequence behavior diagrammed by programmer during overlay planning.

Distances can be temporal, such as the time since prior reference or access time within a network; spatial, measuring hops in a network or address separation in a sequence; or cost, which measures any nondecreasing accumulation of cost since prior reference. Object x is in the locality set at time t if the distance is within a threshold: $D(x,t) \leq T$.

By 1980, the locality principle was understood as a package of three ideas (1):

1. Computational processes pass through a sequence of phases.
2. The locality sets of phases can be inferred by applying a distance function to a program's address trace observed during a backward window.
3. Memory management is optimal when it guarantees each program that its locality sets will be present in high-speed memory.

ADOPTION OF LOCALITY PRINCIPLE

The locality principle was adopted almost immediately by operating systems, databases and hardware architects. It was soon adopted into ever-widening circles:

- In virtual memory to organize caches for address translation and to design the replacement algorithms
- In data caches for CPUs, originally as mainframes and now as microchips
- In buffers between main memory and secondary memory devices
- In buffers between computers and networks
- In video boards to accelerate graphics displays
- In modules that implement the information-hiding principle
- In accounting and event logs in that monitor activities within a system
- In alias lists that associate longer names or addresses with short nicknames
- In the "most recently used" object lists of applications
- In Web browsers to hold recent web pages
- In file systems, to organize indexes (e.g., B-trees) for fastest retrieval of file blocks
- In database systems, to manage record-flows between levels of memory
- In search engines to find the most relevant responses to queries
- In classification systems that cluster related data elements into similarity classes
- In spam filters, which infer which categories of e-mail are in the user's locality space and which are not
- In "spread spectrum" video streaming that bypasses network congestion and reduces the apparent distance to the video server
- In "edge servers" to hold recent web pages accessed by anyone in an organization or geographical region
- In the field of computer forensics to infer criminal motives and intent by correlating event records in many caches
- In the field of network science by defining hierarchies of self-similar locality structures within complex power-law networks

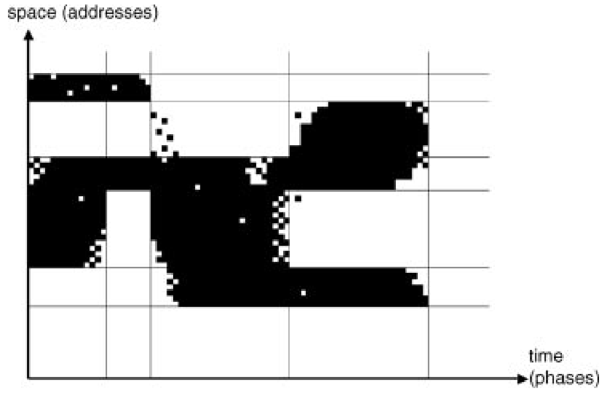


Figure 2. Locality sequence behavior observed by sampling use bits during program execution. Programs exhibit phases and localities naturally, even when overlays are not preplanned.

MODERN MODEL OF LOCALITY: CONTEXT AWARENESS

As the uses of locality expanded into more areas, our understanding of locality has evolved. Locality is the consequence of a more basic principle: Everything we do, we do in a context. Context awareness embraces four key ideas:

- **An observer**
- **Neighborhoods:** One or more sets of objects that are most relevant to the observer at any given time
- **Inference:** A method of identifying the most relevant objects by monitoring the observer's actions and interactions and other information about the observer contained in the environment
- **Optimal actions:** An expectation that the observer will complete work in the shortest time if neighborhood objects are ready accessible in nearby caches

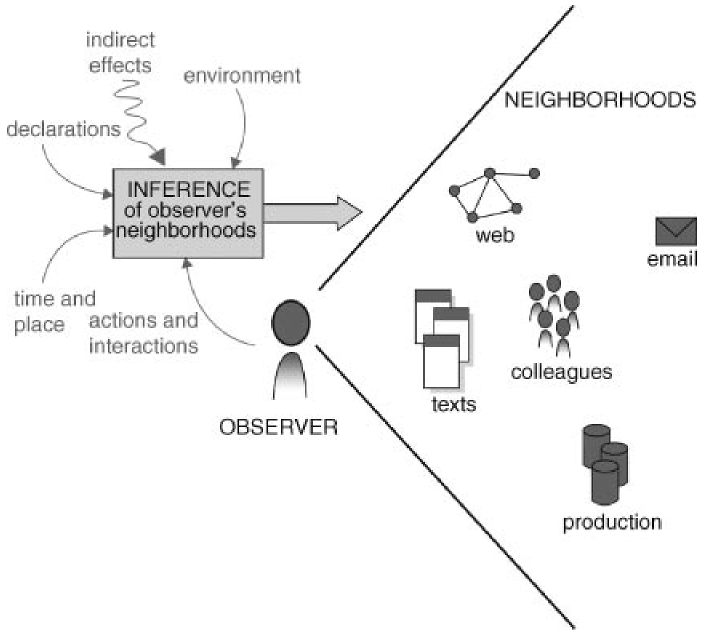


Figure 3. The modern view of locality is a means of inferring the context of an observer using software, so that the software can dynamically adapt its actions to produce optimal behavior for the observer.

These four ideas can be recognized in the original definition. The observer is the execution point of the computational process; the neighborhood is the current locality set; the distance function is the inference mechanism; the optimal action is to guarantee that the current locality is present in a processor's cache. Let us examine the generalizations of these ideas.

The observer is the agent who is trying to accomplish tasks with the help of software, and who places expectations on its function and performance (Fig. 3). In most cases, the observer is the user who interacts with software. In some cases, such as a program that computes a mathematical model, the observer can be built into the software itself.

A neighborhood is a group of objects related to the observer by some metric. Newer examples of neighborhoods include e-mail correspondents, non-spam e-mail, colleagues, teammates, objects used in a project, favorite objects, user's web, items of production, texts, and directories. Some neighborhoods can be known by explicit declarations, for example, a user's file directory, address book, or web pages. But most neighborhoods can only be inferred by monitoring the event sequences of an observer's actions and interactions.

Inference can be any reasonable method that measures the content of neighborhoods. Newer inference methods include Google's counting of incoming hyperlinks to a web page, patterns generated by connectionist networks after being presented with many examples, and Bayesian spam filters.

Optimal actions are performed by the software on behalf of the observer. These actions can come either from inside the software with which the observer is interacting, or from outside that software, in the run-time system.

The matrix below shows four quadrants that correspond to the four combinations of inference data collection and locus of action just mentioned. Examples of software are named in each quadrant and are summarized below. "Inside" and "outside" are relative to the context-aware software.

		Origin of Data for Inference	
		Inside	**Outside**
LOCUS OF ADAPTIVE ACTION	**Inside**	Amazon.com Bayesian spam filter	Semantic web Google
	Outside	Linkers and loaders	Working sets, Ethernet load control

- **Amazon.com, Bayesian spam filters.** Amazon collects data about user purchasing histories and recommends other purchases, by the user or others, that resemble the user's previous purchases. Bayesian spam filters gather data about which e-mails the user considers relevant and then block irrelevant e-mails. (Data collection inside, optimal actions inside.)
- **Semantic web, Google.** Semantic web is a set of declarations of structural relationships that constitute context of objects and their connections. Programs read and act on it. Google gathers data from the Web and uses it to rank pages that seem to be most relevant to a keyword query posed by user. (Data collection outside, optimal actions inside.)
- **Linkers and loaders.** These workhorse systems gather library modules mentioned by a source program and link them together into a self-contained executable module. The libraries are neighborhoods of the source

program. (Data collection inside, optimal action outside.)

- **Working sets, ethernet load controls.** Virtual memory systems measure working sets and guarantee programs enough space to contain them, which thereby prevents thrashing. Ethernet prevents the contention-resolving protocol from getting overloaded by making competing transactions wait longer for retries if load is heavy (10). (Data collection outside, optimal action outside.)

In summary, the modern principle of locality is that observers operate in one or more neighborhoods that can be inferred from dynamic action sequences and static structural declarations. Systems can optimize the observer's productivity by adapting to the observer's neighborhoods, which they can estimate by distance metrics or other inferences.

FUTURE USES OF LOCALITY PRINCIPLE

Locality principles are certain to remain at the forefront of systems design, analysis, and performance, because locality flows from human cognitive and coordinative behavior. The mind focuses on a small part of the sensory field and can work most quickly on the objects of its attention. People organize their social and intellectual systems into neighborhoods of related objects, and they gather the most useful objects of each neighborhood close around them to minimize the time and work of using them. These behaviors are transferred into computational systems they design and into the expectations users have about how their systems should interact with them.

Here are seven modern areas that offer challenging research problems that locality may be instrumental in solving.

Architecture

Computer architects have heavily exploited the locality principle to boost the performance of chips and systems. Putting cache memory near the CPU, either on board the same chip or on a neighboring chip, has enabled modern CPUs to pass the 1-GHz speed mark. Locality within threaded instruction sequences is being exploited by a new generation of multicore processor chips. The "system on a chip" concept places neighboring functions on the same chip to decrease delays of communicating between components significantly. Animated sequences of pictures can be compressed by locality: by detecting the common neighborhood behind a sequence, transmitting it once, and then transmitting the differences for each picture. Architects will continue to examine locality carefully to find new ways to speed up chips, communications, and systems.

Caching

The locality principle is useful wherever there is an advantage in reducing the apparent distance from a process to the objects it can access. Objects in the neighborhood of the process are kept in a local cache with fast access time. The performance acceleration of a cache generally justifies the modest investment in the cache storage. Novel forms of caching have sprung up in the Internet. One prominent example is edge servers that store copies of Web objects near their users. Another example is the clustered databases built by search engines (like Google) to retrieve relevant objects instantly from the same neighborhoods as the asker. Similar capabilities are available in MacOS Windows to speed up finding relevant objects.

Bayesian Inference

A growing number of inference systems exploit Bayes's principle of conditional probability to compute the most likely internal (hidden) states of a system given observable data about the system. Spam filters, for example, use it to infer the e-mail user's mental rules for classifying certain objects as spam. Connectionist networks use it for learning: Their internal states abstract from desired input-output pairs shown to the network; the network gradually acquires a capability for new action. Bayesian inference is an exploitation of locality because it infers a neighborhood given observations of what a user or process is doing.

Forensics

The burgeoning field of computer forensics owes much of its success to the ubiquity of caches. They are literally everywhere in an operating systems and applications. By recovering evidence from these caches, forensics experts can reconstruct (infer) an amazing amount of a criminal's motives and intent (11). Criminals who erase data files are still not safe, because experts use advanced signal-processing methods to recover the faint magnetic traces of the most recent files from the disk (12). Learning to draw valid inferences from data in a computer's caches, and from correlated data in caches in other computers with which the subject has communicated, is a challenging research problem.

Web-Based Business Processes

Web-based business systems allow buyers and sellers to engage in transactions using Web interfaces to sophisticated database systems. Amazon.com illustrates how a system can infer "book interest neighborhoods" of customers and (successfully) recommend additional sales. Many businesses employ customer relationship management systems that infer "customer interest neighborhoods" and allow the company to provide better, more personalized service. Database, network, server, memory, and other caches optimize the performance of these systems (13).

Context-Aware Software

More software designers are coming to believe that most software failures can be traced to the inability of software to be aware of and act on the context in which it operates. More and more modern software uses inferred context to be consistently more reliable, dependable, usable, safe, and secure.

Network Science

Many scientists have begun to apply statistical mechanics to large random networks, typically finding that the distribution of node connections is power law with degree −2 to −3 (14,15). These networks are self-similar, which means that if all neighborhoods (nodes within a maximum distance of each other) are collapsed to single nodes, then the resulting network has the same power distribution as the original (16). The idea that localities are natural in complex systems is not new; in 1976, Madison and Batson (17) reported that program localities have self-similar sublocalities ; in 1977, P. J. Courtois (18) applied it to cluster similar states of complex systems to simplify their performance analyses. The locality principle may offer new understandings of the structure of complex networks.

Researchers looking for challenging problems can find many in these areas and can exploit the principle of locality to solve them.

BIBLIOGRAPHY

1. P. J. Denning, Working sets past and present, *IEEE Trans. Softw. Eng.*, **SE-6**(1): 64–84, 1980.
2. P. J. Denning, The working set model for program behavior, *ACM Commun.*, **11**(5): 323–333, 1968.
3. T. Kilburn, D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner, One-level storage system, *IRE Trans.,* **EC-11**: 223–235, 1962.
4. M. V. Wilkes, Slave memories and dynamic storage allocation, *IEEE Trans. Comp.*, **EC-14**(4): 270–271, 1965.
5. P. J. Denning, Thrashing: Its causes and prevention, *Proc. AFIPS Fall Joint Computer Conference 33*, Thompson, 1968, pp. 915–922.
6. L. A. Belady, A study of replacement algorithms for virtual storage computers, *IBM Systems J.*, **5**(2): 78–101, 1966.
7. D. Ferrari, Improving locality by critical working sets, *ACM Commun.***17**(11): 614–620, 1974.
8. B. Brawn, and F. G. Gustavson, Program behavior in a paging environment, *Proc. AFIPS Fall Joint Computer Conference 33*, Thompson, 1968, pp. 1019–1032.
9. J. Rodriguez-Rosell, and J. P. , Dupuy, The design, implementation, and evaluation of a working set dispatcher, *ACM Commun.,***16**(4): 247–253, 1973.
10. R. M. Metcalfe, and D. Boggs, Ethernet: Distributed packet switching for local networks, *ACM Commun.*, **19**(7): 395–404, 1976.
11. D. Farmer, and W. Venema, *Forensic Discovery*, Reading, MA: Addison Wesley, 2004.
12. B. Carrier, *File System Forensic Analysis*. Reading, MA: Addison Wesley, 2005.
13. D. Menasce, and V. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Englewood Cliffs, NJ: Prentice-Hall, 2000.
14. A. L. Barabasi, *Linked: The New Science of Networks*, Perseus Books, 2002.
15. P. J. Denning, Network laws, *ACM Commun,* **47**(11):15–20, 2004.
16. C. , Song, S. Havlin, and H. Makse, Self-similarity of complex networks, *Nature,* **433**(1): 392–395, 2005.
17. A. W. Madison, and A. Batson, Characteristics of program localities, *ACM Commun.,* **19**(5): 285–294, 1976.
18. P. J. Courtois, *Decomposability*. New York: Academic Press, 1977.

PETER J. DENNING
Naval Postgraduate School
Monterey, California

M

MARKOV CHAIN MONTE CARLO SIMULATIONS

Markov Chain Monte Carlo (MCMC) simulations are widely used in many branches of science. They are nearly as old as computers themselves, since they started in earnest with a 1953 paper by Nicholas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller (1) at the Los Alamos National Laboratory, New Mexico.These authors invented what is nowadays called the Metropolis algorithm. Various applications and the history of the basic ideas are reviewed in the proceedings (2) of the 2003 Los Alamos conference, which celebrated the fiftieth anniversary of the Metropolis algorithm.

OVERVIEW

The Monte Carlo method to compute integrals (3) amounts to approximate an integral $\int_\Omega d\mu(X)\mathcal{O}(X)$, inside a volume Ω, where $d\mu(X)$ is some integration measure and $\int_\Omega d\mu(X) < \infty$, by sampling. Without loss of generality, one can assume that $\int_\Omega d\mu(X) = 1$, and consider accordingly $d\mu(X)$ as a probability measure. Drawing independently N_{sample} values of $X \in \Omega$ according to this probability measure, one has

$$\int_\Omega d\mu(X)\mathcal{O}(X) \approx \frac{1}{N_{\text{samples}}} \sum_{s=1}^{N_{\text{samples}}} \mathcal{O}(X_s) \tag{1}$$

where X_s is the s'th random variable drawn. The right hand side of Equation 1 is an unbiased estimator of the integral, namely it is exact in average, for any N_{sample}. It converges toward the correct value when $N_{\text{samples}} \to \infty$, with a $1/\sqrt{N_{\text{samples}}}$ leading correction. This method is of practical use if one can easily draw, in a computer program, random values $X_s \in \Omega$ according to the measure $d\mu(X)$. This is the case in particular if one integrates inside an N- dimensional hypercube with the flat measure $d\mu^{\text{flat}}(X) \propto \prod_{k=1}^{N} dX^{(k)}$, where the $X^{(k)}$'s are the Cartesian components of X, or when the integral reduces to a finite sum. This is not the case in the situation where this simple measure is multiplied by a nontrivial function $\omega(X)$ of the components $X^{(k)}$. If $\omega(X)$ is a smooth function in Ω with a limited range of variations, one can still draw values of X according to the simple measure $d\mu^{\text{flat}}$ and write (assuming without loss of generality that $\int_\Omega d\mu(X) = 1$ and $\int_\Omega d\mu^{\text{flat}}(X) = 1$)

$$\begin{aligned}\int_\Omega d\mu(X)\mathcal{O}(X) &= \int_\Omega d\mu^{\text{flat}}(X)\omega(X)\mathcal{O}(X) \\ &\approx \frac{1}{N_{\text{samples}}} \sum_{s=1}^{N_{\text{sample}}} \omega(X_s)\mathcal{O}(X_s)\end{aligned} \tag{2}$$

If the function $\omega(X)$ has a very large range of variations with one or several sharp peaks, the sum in Equation 2 is dominated by a few rare configurations, and the Monte Carlo method does not work (most samples are drawn in vain). This is the case of the problem considered in Ref. 1, where $\omega(X)$ is a Boltzmann weight, the exponential of N times a function of order one, with N the number of molecules in a box, which is a notoriously large number.

The Markov chain Monte Carlo method allows overcoming this problem by generating a set of random $X \in \Omega$, distributed according to the full measure $d\mu(X)$, using an auxiliary Markov chain. Note that often, in particular in the physics literature, Monte Carlo is used as a short name for Markov chain Monte Carlo (MCMC). MCMC is sometimes called dynamic Monte Carlo, in order to distinguish it from the usual, "static," Monte Carlo. Many probability distributions, which cannot be sampled directly, allow for MCMC sampling. From now on we write a formula for a discrete space Ω with K_{st} states, although the results can be generalized. A Markov chain (4–7) is a sequence of random variables $X_1, \quad X_2, \quad X_3, \ldots$, that can be viewed as the successive states of a system as a function of a discrete time t, with a transition probability $\mathbf{P}(X_{t+1} = r | X_t = s) = W_{r,s}$ that is a function of r and s only. The next future state X_{t+1} is a function of the current state X_t alone. Andrey Markov (8) was the first to analyze these processes. In order to analyze Markov chains, one introduces an ensemble of chains with an initial probability distribution $\mathbf{P}(X_0 = s)$. By multiplying this vector by the transition matrix repeatedly, one obtains $\mathbf{P}(X_1 = s)$, and then $\mathbf{P}(X_2 = s), \ldots$, successively. The natural question is whether this sequence converges. One says that a probability distribution $w = \{w_s\}_{s \in \Omega}$ is an equilibrium distribution of a Markov chain if it is let invariant by the chain, namely if

$$\sum_{s=1}^{K_{st}} W_{r,s}\omega_s = \omega_r \tag{3}$$

This condition is called balance in the physics literature. A Markov chain is said to be ergodic (irreducible and aperiodic in the mathematical literature) if for all states $r, s \in \Omega$ there is a $N_{r,s}$ such that for all $t > N_{r,s}$ the probability $(W^t)_{s,r}$ to go from r to s in t steps is nonzero. If an equilibrium distribution exists and if the chain is irreducible and aperiodic, one can show (4–7) that, starting from any distribution $\mathbf{P}(X_0 = s)$, the distribution after t steps $\mathbf{P}(X_t = s)$ converges when $t \to \infty$ toward the equilibrium configuration ω_s.

The Metropolis algorithm (see the next section) offers a practical way to generate a Markov chain with a desired equilibrium distribution, on a computer using a pseudorandom numbers generator. Starting from an initial configuration X_0, successive configurations are generated. In most cases, the convergence of $\mathbf{P}(X_t = s)$ toward ω_s is exponential in t and one can safely assume that, after some number of steps t_{eq}, "equilibrium is reached" and that the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

configurations $X_{t_{eq}+1}, X_{t_{eq}+2}, \ldots$ are distributed according to $\omega(X)$.

Whereas the random samples used in a conventional Monte Carlo (MC) integration are obviously statistically independent, those used in MCMC are correlated. The effective number of independent events generated by an equilibrated Markov chain is given by the number of steps done divided by a quantity called "integrated autocorrelation time" τ_{int}. In most cases, τ_{int} does not depend on the quantity $\mathcal{O}$ measured, but there are exceptions like, for example the vicinity of a second-order (continuous) phase transition. The algorithm will fail if τ_{int} is too big. In practice it can be quite difficult to estimate τ_{int} reliably and there can also be a residual effect of the starting position. More sophisticated MCMC-based algorithms such as "coupling from the past" (9,10) produce independent samples rigorously but at the cost of additional computation and an unbounded (although finite on average) running time.

The method was originally developed to investigate a statistical physics problem. Suitable applications arise as well in computational biology, chemistry, physics, economics, and other sciences. MCMC calculations have also revolutionized the field of Bayesian statistics.

Most concepts of modern MCMC simulation were originally developed by physicists and chemists, who still seem to be at the cutting edge of new innovative developments into which mathematician, computer scientists, and statisticians have joined. Their interest developed mainly after a paper by Hastings (11), who generalized the accept/reject step of the Metropolis method. Unfortunately a language barrier developed that inhibits cross-fertilization. For instance the "heat bath" algorithm, which may be applied when the conditional distributions can be sampled exactly, was introduced by physicist Refs. (12) and (13). Then it was rediscovered in the context of Bayesian restoration of images under the name "Gibbs sampler" (14). Another example is the "umbrella sampling" algorithm (15) that was invented in chemical physics and later rediscovered and improved by physicists. This is just two of many examples of how different names for the same method emerged. The reader should be aware that this article was written by physicists, so that their notations and views are dominant in this article. The book by Liu (16) tries to some extent to bridge the language barrier. Other textbooks include those by Robert and Casella (17) for the more mathematically minded reader, Landau and Binder (18) for statistical physicists, Berg (19) from a physicist point of view, but also covering statistics and providing extensive computer code (in Fortran). Kendall, et al. (20) edited a book that combines expositions from physicists, statisticians, and computer scientists.

In the following discussion, we will first explain the basic method and illustrate it for a simple statistical physics system, the Ising ferromagnet in two dimensions, followed by a few remarks on autocorrelations and cluster algorithms. An overview of the MCMC updating scheme is subsequently given. The final section of this article focuses on so-called generalized ensemble algorithms.

MCMC AND STATISTICAL PHYSICS

As mentioned, the MCMC algorithm was invented to investigate problems in statistical physics. The aim of statistical physics is to predict the average macroscopic properties of systems made of many constituents, which can be, for example, molecules in a box (as in the original article of Metropolis et al.), magnetic moments (called spins) at fixed locations in a piece of material, or polymer chains, in a solvent. If one considers the so-called canonical ensemble, where the system has a fixed temperature T, one knows that the probability to observe a given microscopic configuration (or micro state) s (defined in the three examples given above by the positions and velocities of all molecules; the orientations of all the spins; and the exact configuration of all the polymer chains, respectively) is proportional to the Boltzmann weight $\exp(-E_s/k_BT) = \exp(-\beta E_s)$, where E_s is the energy of the configuration s, k_B is the Boltzmann constant, and T is the temperature. (In the following discussion, we will use a unit of temperature such that $k_B = 1$). Let $\mathcal{O}_s$ be the value of $\mathcal{O}$ computed in configuration s. The mean value of any macroscopic observable $\mathcal{O}$ (e.g. the energy) is given by the average of $\mathcal{O}_s$ over all possible configurations s, weighted by the Boltzmann weight of the configuration. This sum (or integral if one has continuous variables) is to be normalized by the sum over all configurations of the Boltzmann weight [the so-called partition function $Z(T)$], namely

$$\hat{O} = \hat{O}(T) = \langle O \rangle = Z^{-1}(T) \sum_{s=1}^{K_{st}} O_s e^{-E_s/T} \tag{4}$$

where $Z(T) = \sum_{s=1}^{K_{st}} \exp(-E_s/T)$. The index $s = 1, \ldots, K_{st}$ labels the configuration (states) of the system.

A particularly simple system is the Ising model for which the energy is given by

$$E = \sum_{<ij>} s_i s_j \tag{5}$$

Here the sum is over the nearest-neighbor sites of a hypercubic D-dimensional lattice and the Ising spins take the values $s_i = \pm 1$, $i = 1, \ldots, N$ for a system of $N = \prod_{i=1}^{D} L_i$ spins. Periodic boundary conditions are used in most simulations. The energy per spin is $e = E/N$. The model describes a ferromagnet for which the magnetic moments are simplified to ± 1 spins at the sites of the lattice. In the $N \to \infty$ limit (and for $D > 1$), this model has two phases separated by a phase transition (a singularity) at the critical temperature $T = T_c$. This is a second-order phase transition, which is continuous in the energy, but not in specific heat. This model can be solved analytically when $D = 2$. (This means that one can obtain exact analytical expressions for the thermodynamic quantities, e.g., the energy per spin, as a function of temperature, in the $N \to \infty$ limit.) This makes the two-dimensional (2-D) Ising model an ideal testbed for MCMC algorithms.

The number of configurations of the Ising model is $K_{st} = 2^N$, because each spin can occur in two states (up or down). Already for a moderately sized lattice, say linear dimension $L_1 = L_2 = 100$ in $D = 2$, K_{st} is a tremendously large number. With the exception of very small systems, it is therefore impossible to do the sums in Equation 4 explicitly. Instead, the large value of K_{st} suggests a statistical evaluation.

IMPORTANCE SAMPLING THROUGH MCMC

As explained in the previous section, one needs a procedure that generates configurations s with the Boltzmann probability

$$P_{B,s} = c_B\, w_{B,s} = c_B e^{-\beta E_s} \tag{6}$$

where the constant c_B is determined by the condition $\sum_s P_{B,s} = 1$. The vector $P_B := \{P_{B,s}\}$ is called the Boltzmann state. When configurations are generated with the probability $P_{B,s}$, the expectation values [Equation (4)] are estimated by the arithmetic averages:

$$\langle O \rangle = \lim_{N_{\text{samples}} \to \infty} \frac{1}{N_{\text{samples}}} \sum_{s=1}^{N_{\text{samples}}} O_s \tag{7}$$

With the MCMC algorithm, this is obtained from a Markov chain with the following properties:

1. *Irreducibility and aperiodicity (ergodicity in the physics literature)*: For any two configurations r and s, an integer $N_{r,s}$ exists such that for all $n > N_{r,s}$, the probability $(W^n)_{r,s}$ is nonzero.
2. *Normalization*: $\sum_r W_{r,s} = 1$.
3. *Stationarity of the Boltzmann distribution (balance in the physics literature)*: The Boltzmann state [Equation (6)] is an equilibrium distribution, namely a right eigenvector of the matrix W with eigenvalue one, i.e., $\sum_s W_{r,s} e^{-\beta E_s} = e^{-\beta E_r}$ holds.

There are many ways to construct a Markov process satisfying 1, 2, and 3. In practice, MCMC algorithms are often based on a stronger condition than 3, namely,

3. *Detailed balance*:

$$W_{r,s} e^{-\beta E_s} = W_{s,r} e^{-\beta E_r} \quad \forall\, r, s. \tag{8}$$

METROPOLIS ALGORITHM AND ILLUSTRATION FOR THE ISING MODEL

Detailed balance obviously does not uniquely fix the transition probabilities $W_{r,s}$. The original Metropolis algorithm (1) has remained a popular choice because of its generality (it can be applied straightforwardly to any statistical physics model) and its computational simplicity. The original formulation generates configurations with the Boltzmann weights. It generalizes immediately to arbitrary weights (see the last section of this article). Given a configuration s, the algorithm proposes a new configuration r with a priori probabilities $f(r, s)$. This new configuration r is accepted with probability

$$a^{r,s} = \min\left[1, \frac{P_{B,r}}{P_{B,s}}\right] = \begin{cases} 1 \text{ for } E_r < E_s \\ e^{-\beta(E_r - E_s)} \text{ for } E_r > E_s \end{cases} \tag{9}$$

If the new configuration is rejected, the old configuration is kept and counted again. For such decisions one uses normally a pseudorandom number generator, which delivers uniformly distributed pseudorandom numbers in the range $[0, 1)$; see Refs. (18) and (19) for examples, and Ref. (21) for the mathematics involved. The Metropolis algorithm gives then rise to the transition probabilities

$$W_{r,s} = f(r,s) a_{r,s} \quad r \neq s \tag{10}$$

and

$$W_{s,s} = f(s,s) + \sum_{r \neq s} f(r,s)(1 - a_{r,s}) \tag{11}$$

Therefore, the ratio $W_{r,s}/W_{s,r}$ satisfies the detailed balance condition [Equation (8)] if

$$f(r,s) = f(s,r) \tag{12}$$

holds. This condition can be generalized when also the acceptance probability is changed (11). In particular such "biased" Metropolis algorithms allow for approaching the heat-bath updating scheme (22).

For the Ising model, the new putative configuration differs from the old one by the flip of a single spin. Such a "single spin update" requires a number of operations of order one and leads to a ratio $P_{B,r}/P_{B,s}$ in Equation (9). This is fundamental in order to have an efficient MCMC algorithm. The spin itself may be chosen at random, although some sequential procedure is normally more efficient (19). The latter procedure violates the detailed balance condition but fulfills the balance condition, and thus, it does lead to the probability of states approaching the Boltzmann distribution [Equation (6)]. The unit of Monte Carlo time is usually defined by N single spin updates, aka a lattice sweep or "an update per spin." Many ways to generate initial configurations exist. Two easy to implement choices are as follows:

1. Use random sampling to generate a configuration of ±1 spins.
2. Generate a completely *ordered* configuration, either all spin $+1$ or -1.

Figure 1 shows two Metropolis energy time series (the successive values of the energy as a function of the discrete Monte Carlo time) of 6000 updates per spin for a 2D Ising model on a 100×100 lattice at $\beta = 0.44$, which is close to the (infinite volume) phase transition temperature of this model ($\beta_c = \ln(1 + \sqrt{2})/2 = 0.44068\ldots$). One can compare with the exact value for e (23) on this system size, which is indicated by the straight line in Fig. 1. The time series

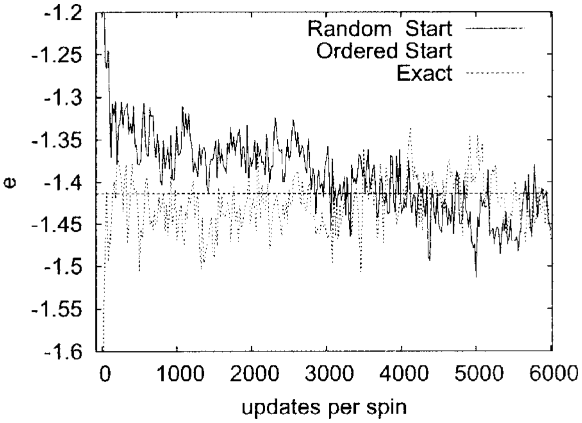


Figure 1. Two-dimensional Ising model: Two initial Metropolis time series for the energy per spin e.

corresponding to the ordered start begins at $e = -2$ and approaches the exact value from below, whereas the other time series begins (up to statistical fluctuation) at $e = 0$ and approaches the exact value from above. It takes a rather long MCMC time of about 3000 to 4000 updates per spin until the two time series start to mix. For estimating equilibrium expectation values, measurements should only be counted from there on. A serious error analysis for the subsequent equilibrium simulation finds an integrated autocorrelation time of $\tau_{\text{int}} \approx 1700$ updates per spin. This long autocorrelation time is related to the proximity of the phase transition (this is the so-called critical slowing down of the dynamics). One can show that, at the transition point, τ_{int} diverges like a power of N.

For this model and a number of other systems with second-order phase transitions, cluster algorithms (24,25) are known, which have much shorter autocorrelation times, in simulations close to the phase transition point. For the simulation of Fig. 1, $\tau_{\text{int}} \approx 5$ updates per spin instead of 1700. Furthermore, for cluster algorithms, τ_{int} grows much slower with the system size N than for the Metropolis algorithm. This happens because these algorithms allow for the instantaneous flip of large clusters of spins (still with order one acceptance probability) in contrast to the local updates of single spins done in the Metropolis and heat-bath-type algorithms. Unfortunately such nonlocal updating algorithms have remained confined to special situations. The interested reader can reproduce these Ising model simulations discussed here using the code that comes with Ref. 19.

UPDATING SCHEMES

We give an overview of MCMC updating schemes that is, because of space limitations, not complete at all.

1. We have already discussed the Metropolis scheme (1) and its generalization by Hastings (7). Variables of the system are locally updated.
2. Within such local schemes, the heat-bath method (12-14) is usually the most efficient. In practice it can be approximated by tabulating Metropolis–Hastings probabilities (22).
3. For simulations at (very) low temperatures, event-driven simulations (26,27), also known as the "N-fold way," are most efficient. They are based on Metropolis or heat-bath schemes.
4. As mentioned before, for a number of models with second order phase transitions the MCMC efficiency is greatly improved by using nonlocal cluster updating (24).
5. Molecular dynamics (MD) moves can be used as proposals in MCMC updating (28,29), a scheme called "hybrid MC", see Ref. (30) for a review of MD simulations.

More examples can be found in the W. Krauth contribution in Ref. (31) and A. D. Sokal in Ref. (32).

GENERALIZED ENSEMBLES FOR MCMC SIMULATIONS

The MCMC method, which we discussed for the Ising model, generates configurations distributed according to the Boltzmann–Gibbs canonical ensemble, with weights $P_{B,s}$. Mean values of physical observables at the temperature chosen are obtained as arithmetic averages of the measurements made [Equation 7]. There are, however, in statistical physics, circumstances where another ensemble (other weights $P_{NB,s}$) can be more convenient. One case is the computation of the partition function $Z(T)$ as a function of temperature. Another is the investigation of configurations of physical interest that are rare in the canonical ensemble. Finally the efficiency of the Markov process, i.e., the computer time needed to obtain a desired accuracy, can depend greatly on the ensemble in which the simulations are performed. This is the case, for example when taking into account the Boltzmann weights, the phase space separates, loosely speaking, into several populated regions separated by mostly vacant regions, creating so-called free energy barriers for the Markov chain.

A first attempt to calculate the partition function by MCMC simulations dates back to a 1959 paper by Salsburg et al. Ref. (33). As noticed by the authors, their method is restricted to very small lattices. The reason is that their approach relies on what is called in the modern language "reweighting." It evaluates results at a given temperature from data taken at another temperature. The reweighting method has a long history. McDonald and Singer (34) were the first to use it to evaluate physical quantities over a range of temperatures from a simulation done at a single temperature. Thereafter dormant, the method was rediscovered in an article (35) focused on calculating complex zeros of the partition function. It remained to Ferrenberg and Swendsen (36), to formulate a crystal clear picture for what the method is particularly good, and for what it is not: The reweighting method allows for focusing on maxima of appropriate observables, but it does not allow for covering a finite temperature range in the infinite volume limit.

To estimate the partition function over a finite energy density range Δe, i.e., $\Delta E \sim N$, one can patch the histograms from simulations at several temperatures. Such multi-histogram methods also have a long tradition. In 1972 Valleau and Card (37) proposed the use of overlapping bridging distributions and called their method "multistage sampling." Free energy and entropy calculations become possible when one can link the temperature region of interest with a range for which exact values of these quantities are known. Modern work (38,39) developed efficient techniques to combine the overlapping distributions into one estimate of the spectral density $\rho(E)$ [with $Z(T) = \int dE\rho(E)\exp(-\beta E)$] and to control the statistical errors of the estimate. However, the patching of histograms from canonical simulations faces several limitations:

1. The number of canonical simulations needed diverges like $\sqrt{N}$ when one wants to cover a finite, noncritical range of the energy density.
2. At a first-order phase transition point, the canonical probability of configurations with an interface decreases exponentially with N.

One can cope with the difficulties of multi-histogram methods by allowing arbitrary sampling distributions instead of just the Boltzmann–Gibbs ensemble. This was first recognized by Torrie and Valleau (15) when they introduced umbrella sampling. However, for the next 13 years, the potentially very broad range of applications of the basic idea remained unrecognized. A major barrier, which prevented researchers from trying such extensions, was certainly the apparent lack of direct and straightforward ways of determining suitable weighting functions $P_{NB,s}$ for problems at hand. In the words of Li and Scheraga (40): *The difficulty of finding such weighting factors has prevented wide applications of the umbrella sampling method to many physical systems*.

This changed with the introduction of the multicanonical ensemble multicanonical ensemble (41), which focuses on well-defined weight functions and offers a variety of methods to find a "working approximation." Here a working approximation is defined as being accurate enough, so that the desired energy range will indeed be covered after the weight factors are fixed. A similar approach can also be constructed for cluster algorithms (42). A typical simulation consists then of three parts:

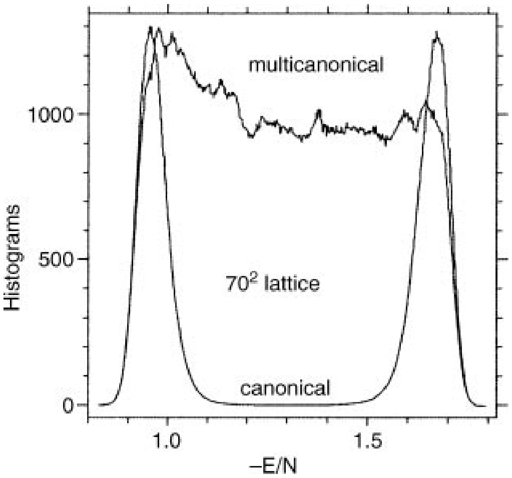


Figure 2. Multicanonical $P_{\text{muca}}(E)$ together with canonical $P(E)$ energy distribution as obtained in Ref. 41 for the $2d$ 10-state Potts model on a 70×70 lattice. In the multicanonical ensemble, the gap between the two peaks present in $P(E)$ is filled up, accelerating considerably the dynamics of the Markov Chain.

1. Construct a working approximation of the weight function P_{muca}. The Wang–Landau recursion (43) is an efficient approach. See Ref. (44) for a comparison with other methods.
2. Perform a conventional MCMC simulation with these weight factors.
3. Reweight the data to the desired ensemble: $\langle \mathcal{O} \rangle = \lim_{N_{\text{samples}} \to \infty} \frac{1}{N_{\text{samples}}} \sum_{s=1}^{N_{\text{samples}}} \mathcal{O}_s P_{B,s}/P_{\text{muca,s}}$ Details can be found in Ref. 19.

Another class of algorithms appeared in a couple of years around 1991 in several papers (45–50), which all aimed at improving MCMC calculations by extending the confines of the canonical ensemble. Practically most important has been the replica exchange method, which is also known under the names parallel tempering and multiple Markov chains. In the context of spin glass simulations, an exchange of partial lattice configurations at different temperatures was proposed by Swendsen and Wang (45). But it was only recognized later (46,50) that the special case for which entire configurations are exchanged is of utmost importance, see Ref. 19 for more details. Closely related is the Jump Walker (J-Walker) approach (51), which feeds replica from a higher temperature into a simulation at a lower temperature instead of exchanging them. But in contrast to the replica exchange method this procedure disturbs equilibrium to some extent. Finally, and perhaps most importantly, from about 1992 on, applications of generalized ensemble methods diversified tremendously as documented in a number of reviews (52–54).

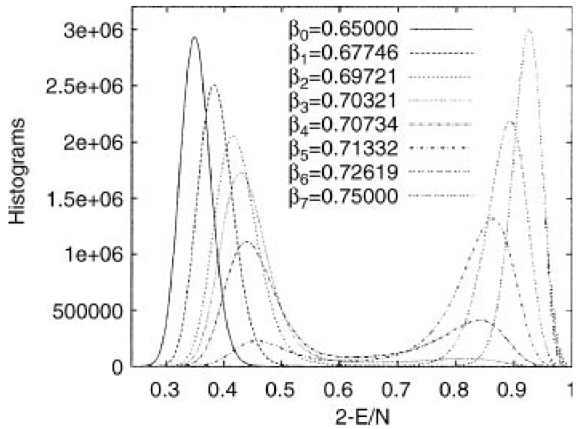


Figure 3. Canonical energy distributions $P(E)$ from a parallel tempering simulation with eight processes for the $2d$ 10-state Potts model on 20×20 lattices (Fig. 6.2 in Ref. 19).

The basic mechanisms for overcoming energy barriers with the multicanonical algorithm are best illustrated for first-order phase transitions (namely a transition with a nonzero latent heat, like the ice–water transition), where one deals with a single barrier. For a finite system the temperature can be fine-tuned to a pseudocritical value, which is defined so that the energy density exhibits two peaks of equal heights. To give an example, Fig. 2 shows for the $2d$ 10-state Potts (55) the canonical energy histogram at a pseudocritical temperature versus the energy histogram of a multicanonical simulation (41). The same barrier can also be overcome by a parallel tempering simulation but in a quite different way. Figure 3 shows the histograms from a parallel tempering simulation with eight processes on 20×20 lattices. The barrier can be "jumped" when there are on both sides temperatures in the ensemble, which are sufficiently close to a pseudocritical temperature for which the two peaks of the histogram are of competitive height. In complex systems with a rugged free energy landscape (spin glasses, biomolecules, . . .), the barriers can no longer be explicitly controlled. Nevertheless it has turned out that switching to the discussed ensembles can greatly enhance the MCMC efficiency (53,54). For a recent discussion of ensemble optimization techniques, see Ref. (56) and references given therein.

BIBLIOGRAPHY

1. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of state calculations by fast computing machines. *J. Chem. Phys.*, **21**: 1087–1092, 1953.
2. J. Gubernatis (Editor). The Monte Carlo method in the physical sciences: Celebrating the 50th anniversary of the Metropolis algorithm, *AIP Conference Proc.* Vol 690, Melville, NY, 2003.
3. N. Metropolis and S. Ulam, The Monte Carlo method. *J. Am. Stat. Assoc.*, **44**: 335–341, 1949.
4. J. G. Kemeny and J. L. Snell, *Finite Markov Chains.*New York: Springer, 1976.
5. M. Iosifescu, *Finite Markov Processes and Their Applications.*Chichester: Wiley, 1980.
6. K. L. Chung, *Markov Chains with Stationary Transition Probabilities*, 2nd ed., New York: Springer, 1967.
7. E. Nummelin, *General Irreducible Markov Chains and Non-Negative Operators.*Cambridge: Cambridge Univ. Press, 1984.
8. A. A. Markov, Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom Universitete, 2-ya seriya, tom 15: 135–156, 1906.
9. J. G. Propp and D. B. Wilson, Exact sampling with coupled Markov chains and applications in statistical mechanics. *Random Structures and Algorithms*, **9**: 223–252, 1996.
10. J. G. Proof and D. B. Wilson, Coupling from the Past User's Guide. DIMACS Series in Discrete Mathematics and Theoretical Computer Science (AMS), **41**: 181–192, 1998.
11. W. K. Hastings, Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**: 97–109, 1970.
12. R. J. Glauber, Time-dependent statistics of the Ising model. *J. Math. Phys.*, **4**: 294–307, 1963.
13. M. Creutz, Monte Carlo study of quantized $SU(2)$ gauge theory. *Phys. Rev. D.*, **21**: 2308–2315, 1980.
14. S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Machine Intelli.* **6**: 721–741, 1984.
15. G. M. Torrie and J. P. Valleau, Nonphysical sampling distributions in Monte Carlo free energy estimation: Umbrella sampling. *J. Comp. Phys.*, **23**: 187–199, 1977.
16. J. S. Liu, Monte Carlo strategies in scientific computing. New York: Springer, 2001.
17. C. P. Robert and G. Casella, Monte Carlo statistical methods (2nd ed.), New York: Springer, 2005.
18. D. P. Landau and K. Binder, A guide to Monte Carlo simulations in statistical physics. Cambridge: Cambridge University Press, 2000.
19. B. A. Berg, Markov chain Monte Carlo simulations and their statistical analysis. Singapore: World Scientific, 2004.
20. W. S. Kendall, F. Liang, and J.-S. Wang (Eds), Markov Chain Monte Carlo: Innovations and applications (Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore). Singapore: World Scientific, 2005.
21. D. Knuth, The art of computer programming, Vol 2: Semi numerical algorithms, Third Edition. Reading, MA: Addison-Wesley, 1997, pp. 1193
22. A. Bazavov and B. A. Berg, Heat bath efficiency with Metropolis-type updating. *Phys. Rev. D.*, **71**: 114506, 2005.
23. A. E. Ferdinand and M. E. Fisher, Bounded and inhomogeneous Ising models. I. Specific-heat anomaly of a finite lattice. *Phys. Rev.*, **185**: 832–846, 1969.
24. R. H. Swendsen and J.-S. Wang, Non-universal critical dynamics in Monte Carlo simulations. *Phys. Rev. Lett.*, **58**: 86–88, 1987.
25. U. Wolff, Collective Monte Carlo updating for spin systems. *Phys. Rev. Lett.*, **62**: 361–363, 1989.
26. A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, A new algorithm for Monte Carlo simulation of Ising spin systems. *J. Comp. Phys.*, **17**: 10–18, 1975.
27. M. A. Novotny, A tutorial on advanced dynamic Monte carlo methods for systems with discrete state spaces. *Ann. Rev. Comp. Phys.*, **9**: 153–210, 2001.
28. S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, Hybrid Monte Carlo. *Phys. Lett. B.*, **195**: 216–222, 1987.
29. B. Mehlig, D. W. Heermann, and B. M. Forrest, Hybrid Monte Carlo Methods for condensed-matter systems. *Phys. Rev. B.*, **45**: 679–685, 1992.
30. D. Frenkel and B. Smit, Understanding molecular simulation. San Diego CA: Academic Press, 1996.
31. J. Kertesz and I. Kondor (Ed.), Advances in Computer Simulation. *Lecture Notes in Physics*, Heidelberg: Springer Verlag, 1998.
32. K. Binder (Ed.), Monte Carlo and molecular dynamics simulations in polymer science, Oxford: Oxford University Press, 1996.
33. Z. W. Salsburg, J. D. Jacobson, W. S. Fickett, and W. W. Wood. Applications of the Monte Carlo method to the lattice-gas model. I. Two-dimensional triangular lattice. *J. Chem. Phys.*, **30**: 65–72, 1959.
34. I. R. McDonald and K. Singer, Calculation of thermodynamic properties of liquid argon from Lennard-Jones parameters by a Monte Carlo Method. *Discussions Faraday Soc.*, **43**: 40–49, 1967.
35. M. Falcioni, E. Marinari, L. Paciello, G. Parisi, and B. Taglienti, Complex zeros in the partition function of four-dimensional $SU(2)$ lattice gauge model. *Phys. Lett. B.*, **108**: 331–332, 1982.

36. A. M. Ferrenberg and R. H. Swendsen, New Monte Carlo technique for studying phase transitions. *Phys. Rev. Lett.,* **61**: 2635–2638, 1988; **63**: 1658, 1989.

37. J. P. Valleau and D. N. Card, Monte Carlo estimation of the free energy by multistage sampling. *J. Chem. Phys.*, **37**: 5457–5462, 1972.

38. A. M. Ferrenberg and R. H. Swendsen, Optimized Monte Carlo data analysis. *Phys. Rev. Lett.*, **63**: 1195–1198, 1989.

39. N. A. Alves, B. A. Berg, and R. Villanova, Ising-Model Monte Carlo simulations: Density of states and mass gap. *Phys. Rev. B.*, **41**: 383–394, 1990.

40. Z. Li and H. A. Scheraga, Structure and free energy of complex thermodynamic systems. *J. Mol. Struct. (Theochem)*, **179**: 333–352, 1988.

41. B. A. Berg and T. Neuhaus, Multicanonical ensemble: A new approach to simulate first-order phase transitions. *Phys. Rev. Lett.*, **68**: 9–12, 1992.

42. W. Janke and S. Kappler, Multibondic cluster algorithm for Monte Carlo simulations of first-order phase transitions. *Phys. Rev. Lett.*, **74**: 212–215, 1985.

43. F. Wang and D. P. Landau, Efficient, multiple-range random walk algorithm to calculate the density of states. *Phys. Rev. Lett.*, **86**: 2050–2053, 2001.

44. Y. Okamoto, Metropolis algorithms in generalized ensemble. In Ref. (2), pp. 248–260. On the web at http://arxiv.org/abs/cond-mat/0308119.

45. R. H. Swendsen and J.-S. Wang, Replica Monte Carlo simulations of spin glasses. *Phys. Rev. Lett.*, **57**: 2607–2609, 1986.

46. C. J. Geyer, Markov Chain Monte Carlo maximum likelihood, in E. M. Keramidas (ed.), *Computing Science and Statistics, Proc. of the 23rd Symposium on the Interface*. Fairfax, VA, Interface Foundation, 1991, pp. 156–163.

47. C. J. Geyer and E. A. Thompson, Annealing Markov chain Monte Carlo with applications to ancestral inference. *J. Am. Stat. Ass.*, **90**: 909–920, 1995.

48. E. Marinari and G. Parisi, Simulated tempering: A new Monte Carlo scheme. *Europhys. Lett.*, **19**: 451–458, 1992.

49. A. P. Lyubartsev, A. A. Martsinovski, S. V. Shevkanov, and P. N. Vorontsov-Velyaminov, New approach to Monte Carlo calculation of the free energy: Method of expanded ensembles. *J. Chem. Phys.*, **96**: 1776–1783, 1992.

50. K. Hukusima and K. Nemoto, Exchange Monte Carlo method and applications to spin glass simulations. *J. Phys. Soc. Japan*, **65**: 1604–1608, 1996.

51. D. D. Frantz, D. L. Freemann, and J. D. Doll, Reducing quasi-ergodic behavior in Monte Carlo simulations by J-walking: Applications to atomic clusters. *J. Chem. Phys.*, **93**: 2769–2784, 1990.

52. W. Janke, Multicanonical Monte Carlo simulations. *Physica A.*, **254**: 164–178, 1998.

53. U. H. Hansmann and Y. Okamoto, The generalized-ensemble approach for protein folding simulations. *Ann. Rev. Comp. Phys.*, **6**: 129–157, 1999.

54. A. Mitsutake, Y. Sugita, and Y. Okamoto, Generalized-ensemble algorithms for molecular simulations of biopolymers. *Biopolymers (Peptide Science)*, **60**: 96–123, 2001.

55. F. Y. Wu, The Potts model. *Rev. Mod. Phys.*, **54**: 235–268, 1982.

56. S. Trebst, D. A. Huse, E. Gull, H. G. Katzgraber, U. H. E. Hansmann, and M. Troyer, Ensemble optimization techniques for the simulation of slowly equilibrating systems, Invited talk at the*19th Annual Workshop on Computer Simulation Studies in Condensed Matter Physics*, in D. P. Landau, S. P. Lewis, and H.-B. Schuettler (eds.), Athens, GA, 20–24 February 2006; Springer Proceedings in Physics, Vol. 115, 2007. Available on the internet at: http://arxiv.org/abs/cond-mat/0606006.

Bernd A. Berg
Florida State University
Tallahassee, Florida
Alain Billoire
Service de Physique Théorique
CEA Saclay
Gif-sur-Yvette, France

MARKOV CHAINS

INTRODUCTION

A Markov chain is, roughly speaking, some collection of random variables with a temporal ordering that have the property that *conditional upon the present, the future does not depend on the past*. This concept, which can be viewed as a form of something known as the *Markov property*, will be made precise below, but the principle point is that such collections lie somewhere between one of *independent* random variables and a completely general collection, which could be extremely complex to deal with.

Andrei Andreivich Markov commenced the analysis of such collections of random variables in 1907, and their analysis remains an active area of research to this day. The study of Markov chains is one of the great achievements of probability theory. In his seminal work (1), Andrei Nikolaevich Kolmogorov remarked, "Historically, the independence of experiments and random variables represents the very mathematical concept that has given probability its peculiar stamp."

However, there are many situations in which it is necessary to consider sequences of random variables that *cannot* be considered to be independent. Kolmogorov went on to observe that "[Markov et al.] frequently fail to assume complete independence, they nevertheless reveal the importance of assuming analogous, weaker conditions, in order to obtain significant results." The aforementioned Markov property, the defining feature of the Markov chain, is such an *analogous, weaker condition* and it has proved both strong enough to allow many, powerful results to be obtained while weak enough to allow it to encompass a great many interesting cases.

Much development in probability theory during the latter part of the last century consisted of the study of sequences of random variables that are not entirely independent. Two weaker, but related conditions proved to be especially useful: the Markov property that defines the Markov chain and the *martingale* property. Loosely speaking, a martingale is a sequence of random variables whose expectation at any point in the future, which is conditional on the past and the present is equal to its current value. A broad and deep literature exists on the subject of martingales, which will not be discussed in this article. A great many people have worked on the theory of Markov chains, as well as their application to problems in a diverse range of areas, over the past century, and it is not possible to enumerate them all here.

There are two principal reasons that Markov chains play such a prominent role in modern probability theory. The first reason is that they provide a powerful yet tractable framework in which to describe, characterize, and analyze a broad class of sequences of random variables that find applications in numerous areas from particle transport through finite state machines and even in the theory of gene expression. The second reason is that a collection of powerful computational algorithms have been developed to provide samples from complicated probability distributions via the simulation of particular Markov chains: These *Markov chain Monte Carlo* methods are now ubiquitous in all fields in which it is necessary to obtain samples from complex probability distributions, and this has driven much of the recent research in the field of Markov chains.

The areas in which Markov chains occur are far too numerous to list here, but here are some typical examples:

- Any collection of independent random variables forms a Markov chain: In this case, given the present, the future is independent of the past *and the present*.
- The celebrated symmetric *random walk* over the integers provides a classic example: The next value taken by the chain is one more or less than the current value with equal probability, regardless of the route by which the current value was reached. Despite its simplicity, this example, and some simple generalizations, can exhibit a great many interesting properties.
- Many popular board games have a Markov chain representation, for example, "Snakes and Ladders," in which there are 100 possible states for each counter (actually, there are somewhat fewer, as it is not possible to end a turn at the top of a snake or the bottom of a ladder), and the next state occupied by any particular counter is one of the six states that can be reached from the current one, each with equal probability. So, the next state is a function of the current state and an external, independent random variable that corresponds to the roll of a die.
- More practically, the current amount of water held in a reservoir can be viewed as a Markov chain: The volume of water stored after a particular time interval will depend only on the volume of water stored now and two random quantities: the amount of water leaving the reservoir and the amount of water entering the reservoir. More sophisticated variants of this model are used in numerous areas, particularly within the field of queueing theory (where water volume is replaced by customers awaiting service).
- The evolution of a finite state machine can be viewed as the evolution of a (usually deterministic) Markov chain.

It is common to think of Markov chains as describing the trajectories of dynamic objects. In some circumstances, a natural dynamic system can be associated with a collection of random variables with the right conditional independence structure—the random walk example discussed previously, for example, can be interpreted as moving from one position to the next, with the nth element of the associated Markov chain corresponding to its position at discrete time index n. As the distribution of each random variable in the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

sequence depends only on the value of the previous element of the sequence, one can endow any such collection (assuming that one can order the elements of the collection, which the definition of a Markov chain employed here ensures is always possible) with a dynamic structure. One simply views the distribution of each element, conditional on the value of the previous one as being the probability of *moving* between those states at that time. This interpretation provides no great insight, but it can allow for simpler interpretations and descriptions of the behavior of collections of random variables of the sort described here. Indeed, it is the image of a chain of states, with each one leading to the next that suggests the term "Markov chain."

STOCHASTIC PROCESSES

To proceed to the formal definition of a Markov chain, it is first necessary to make precise what is meant by a collection of random variables with some temporal ordering. Such a collection of random variables may be best characterized as a stochastic process. An E-valued *process* is a function $X : \mathcal{I} \to E$ that maps values in some index set $\mathcal{I}$ to some other space E. The evolution of the process is described by considering the variation of $X_i := X(i)$ with i. An E-valued *stochastic process* (or *random process*) can be viewed as a process in which, for each $i \in \mathcal{I}$, X_i is a random variable taking values in E.

Although a rich literature on more general situations exists, this article will consider only *discrete time stochastic processes* in which the index set $\mathcal{I}$ is the natural numbers, $\mathbb{N}$ (of course, any index set isomorphic to $\mathbb{N}$ can be used in the same framework by simple relabeling). The notation X_i is used to indicate the value of the process at time i (note that there need be no connection between the index set and *real* time, but this terminology is both convenient and standard). Note that the Markov property may be extended to continuous time processes in which the index set is the positive real numbers, and this leads to a collection of processes known as either *Markov processes* or *continuous time Markov chains*. Such processes are not considered here in more detail, as they are of somewhat lesser importance in computer science and engineering applications. A rich literature on these processes does exist, and many of the results available in the discrete time case have continuous time analog—indeed, some results may be obtained considerably more naturally in the continuous time setting.

At this point, a note on terminology is necessary. Originally, the term "Markov chain" was used to describe any stochastic process with the Markov property and a finite state space. Some references still use this definition today. However, in computer science, engineering, and computational statistics, it has become more usual to use the term to refer to any discrete time stochastic process with the Markov property, regardless of the state space, and this is the definition used here. Continuous time processes with the Markov property will be termed Markov processes, and little reference will be made to them. This usage is motivated by considerations developing from Markov chain Monte Carlo methods and is standard in more recent literature.

Filtrations and Stopping Times

This section consists of some technical details that, although not essential to a basic understanding of the stochastic process or Markov chains in particular, are fundamental and will be encountered in any work dealing with these subjects.

A little more technical structure is generally required to deal with stochastic processes than with simple random variables. Although technical details are avoided as far as possible in this article, the following concept will be needed to understand much of the literature on Markov chains.

To deal with simple random variables, it suffices to consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ in which Ω is the set of events, $\mathcal{F}$ is the σ-algebra corresponding to the collection of measurable outcomes (i.e., the collection of subsets of Ω to which it is possible to assign a probability; typically the collection of all subsets of Ω in the discrete case), and $\mathbb{P}$ is the probability *measure*, which tells us the probability that any element of $\mathcal{F}$ contains the event that occurs as follows: $\mathbb{P} : \mathcal{F} \to [0, 1]$. To deal with stochastic processes, it is convenient to define a *filtered probability space*$(\Omega, \mathcal{F}, \{\mathcal{F}_i\}_{i \in \mathbb{N}}, \mathbb{P})$. The collection of sub-$\sigma$-algebras, $\{\mathcal{F}_i\}_{i \in \mathbb{N}}$, which is termed a *filtration*, has a particular structure:

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \ldots \subset \mathcal{F}_n \subset \mathcal{F}_{n+1} \subset \ldots \subset \mathcal{F}$$

and its most important property is that, for any n, the collection of variables $X_1, X_2, \ldots, X_n$ must be measurable with respect to $\mathcal{F}_n$. Although much more generality is possible, it is usually sufficient to consider the *natural filtration* of a process: that is the one generated by the process itself. Given any collection of random variables of a common probability space, a smallest σ-algebra exists with respect to which those random variables are jointly measurable. The natural filtration is the filtration generated by setting each $\mathcal{F}_n$ equal to the smallest σ-algebra with respect to which $X_1, \ldots, X_n$ are measurable. Only this filtration will be considered in the current article. An intuitive interpretation of this filtration, which provides increasingly fine subdivisions of the probability space, is that $\mathcal{F}_n$ tells us how much information can be provided by knowledge of the values of the first n random variables: It tells us which events can be be distinguished given knowledge of $X_1, \ldots, X_n$.

It is natural when considering a process of this sort to ask questions about *random times*: Is there anything to stop us from defining additional random variables that have an interpretation as the index, which identifies a particular time in the evolution of the process? In general, some care is required if these random times are to be useful: If the temporal structure is real, then it is necessary for us to be able to determine whether the time which has been reached so far is the time of interest, given some realization of the process up to that time. Informally, one might require that $\{\tau = n\}$ can be ascribed a probability of zero or one, given knowledge of the first n states, for any n. In fact, this is a little stronger than the actual requirement, but it provides a simple interpretation that suffices for many purposes. Formally, if $\tau : \Omega \to \mathcal{I}$ is a random time, and the event $\{\omega : \tau(\omega) = n\} \in \mathcal{F}_n$ for all n, then τ is known as a *stopping time*. Note that this condition amounts to requiring that the event $\{\omega : \tau(\omega) = n\}$ is *independent* of all subsequent states of the chain, $X_{n+1}, X_{n+2}, \ldots$ conditional upon $X_1, \ldots, X_n$. The most common example of a stopping time is the *hitting time*, τ_A, of a set A:

$$\tau_A := \inf\{n : X_n \in A\}$$

which corresponds to the first time that the process enters the set A. Note that the apparently similar

$$\tau_A' = \inf\{n : X_{n+1} \in A\}$$

is *not* a stopping time (in any degree of generality) as the state of the chain at time $n+1$ is not necessarily known in terms of the first n states.

Note that this distinction is not an artificial or frivolous one. Consider the chain produced by setting $X_n = X_{n-1} + W_n$ where $\{W_n\}$ are a collection of independent random variables that correspond to the value of a gambler's winnings, in dollars, in the nth independent game that he plays. If $A = [10,000, \infty)$, then τ_A would correspond to the event of having won \$10,000, and, indeed, it would be possible to stop when this occurred. Conversely, if $A = (-\infty, -10,000]$, then τ_A' would correspond to the last time before that at which \$10,000 have been lost. Although many people would like to be able to stop betting immediately *before* losing money, it is not possible to know that one *will* lose the next one of a sequence of independent games.

Given a stopping time, τ, it is possible to define the *stopped process*, $X_1^\tau, X_2^\tau, \ldots$, associated with the process $X_1, X_2, \ldots$, which has the expected definition; writing $m \wedge n$ for the smaller of m and n, define $X_n^\tau = X_{\tau \wedge n}$. That is, the stopped process corresponds to the process itself at all times up to the random stopping time, after which it takes the value it had at that stopping time: It stops. In the case of τ_A, for example, the stopped process mirrors the original process until it enters A, and then it retains the value it had upon entry to A for all subsequent times.

MARKOV CHAINS ON DISCRETE STATE SPACES

Markov chains that take values in a discrete state space, such as the positive integers or the set of colors with elements red, green, and blue, are relatively easy to define and use. Note that this class of Markov chains includes those whose state space is countably infinite: As is often the case with probability, little additional difficulty is introduced by the transition from finite to countable spaces, but considerably more care is needed to deal rigorously with uncountable spaces.

To specify the distribution a Markov chain on a discrete state space, it is intuitively sufficient to provide an initial distribution, the marginal distribution of its first element, and the conditional distributions of each element given the previous one. To formalize this notion, and precisely what the *Markov property* referred to previously means, it is useful to consider the joint probability distribution of the first n elements of the Markov chain. Using the definition of conditional probability, it is possible to write the joint distribution of n random variables, $X_1, \ldots, X_n$, in the following form, using $X_{1:n}$ to denote the vector $(X_1, \ldots, X_n)$:

$$\mathbb{P}(X_{1:n} = x_{1:n}) = \mathbb{P}(X_1 = x_1) \\ \prod_{i=2}^{n} \mathbb{P}(X_i = x_i | X_{1:i-1} = x_{1:i-1})$$

The probability that each of the first n elements takes particular values can be decomposed recursively as the probability that all but one of those elements takes the appropriate value and the conditional probability that the remaining element takes the specified value given that the other elements take the specified values.

This decomposition could be employed to describe the *finite-dimensional distributions* (that is, the distribution of the random variables associated with finite subsets of $\mathcal{I}$) of any stochastic process. In the case of a Markov chain, the distribution of any element is influenced only by the previous state if the entire history is known: This is what is meant by the statement that "conditional upon the present, the future is independent of the past." This property may be written formally as

$$\mathbb{P}(X_n = x_n | X_{1:n-1} = x_{1:n-1}) = \mathbb{P}(X_n = x_n | X_{n-1} = x_{n-1})$$

and so for any discrete state space Markov chain:

$$\mathbb{P}(X_{1:n} = x_{1:n}) = \mathbb{P}(X_1 = x_1) \prod_{i=2}^{n} \mathbb{P}(X_i = x_i | X_{i-1} = x_{i-1})$$

As an aside, it is worthwhile to notice that Markov chains encompass a much broader class of stochastic processes than is immediately apparent. Given any stochastic process in which for all $n > L$ and $x_{1:n-1}$,

$$\mathbb{P}(X_n = x_n | X_{1:n-1} = x_{1:n-1}) = \mathbb{P}(X_n = x_n | X_{n-L:n-1} = x_{n-L:n-1})$$

it suffices to consider a process Y on the larger space E^L defined as

$$Y_n = (X_{n-L+1}, \ldots, X_n)$$

Note that $(X_{1-L}, \ldots, X_0)$ can be considered arbitrary without affecting the argument. Now, it is straightforward to determine that the distribution of Y_{n+1} depends only on Y_n. In this way, any stochastic process with a finite memory may be cast into the form of a Markov chain on an extended space.

The Markov property, as introduced above, is more correctly known as the *weak Markov property*, and in the case of Markov chains in which the transition probability is not explicitly dependent on the time index, it is normally written in terms of expectations of integrable test function $\xi : E^m \to \mathbb{R}$, where m may be any positive integer. The weak Markov property in fact tells us that the expected value of the integral of any integrable test function over the next m states of a Markov chain depends only on the value of the current state, so for any n and any $x_{1:n}$:

$$\mathbb{E}[\xi(X_{n+1},\ldots,X_{n+m})|X_{1:n}] = \mathbb{E}[\xi(X_{n+1},\ldots,X_{n+m+1})|X_n]$$

It is natural to attempt to generalize this by considering random times rather than deterministic ones. The *strong Markov property* requires that, for any stopping time τ, the following holds:

$$\mathbb{E}[\xi(X_{\tau+1},\ldots,X_{\tau+m})|X_{1:\tau}] = \mathbb{E}[\xi(X_{\tau+1},\ldots,X_{\tau+m+1})|X_\tau]$$

In continuous time settings, these two properties allow us to distinguish between weak and strong Markov processes (the latter is a strict subset of the former, because $\tau = n$ is a stopping time). However, in the discrete time setting, the weak and strong Markov properties are equivalent and are possessed by Markov chains as defined above.

It is conventional to view a Markov chain as describing the path of a dynamic object, which *moves* from one state to another as time passes. Many physical systems that can be described by Markov chains have precisely this property—for example, the motion of a particle in an absorbing medium. The position of the particle, together with an indication as to whether it has been absorbed or not, may be described by a Markov chain whose states contain coordinates and an absorbed/not-absorbed flag. It is then natural to think of the initial state as having a particular distribution, say, $\mu(x_1) = \mathbb{P}(X_1 = x_1)$ and, furthermore, for there to be some *transition kernel* that describes the distribution of moves from a state x_{n-1} to a state x_n at time n, say, $K_n(x_{n-1},x_n) = \mathbb{P}(X_n = x_n|X_{n-1} = x_{n-1})$. This allows us to write the distribution of the first n elements of the chain in the compact form:

$$\mathbb{P}(X_{1:n} = x_{1:n}) = \mu(x_1)\prod\nolimits_{i=2}^{n} K_i(x_{i-1},x_i)$$

Nothing is preventing these transition kernels from being explicitly dependent on the time index; for example, in the reservoir example presented above, one might expect both water usage and rainfall to have a substantial seasonal variation, and so the volume of water stored tomorrow would be influenced by the date as well as by that volume stored today. However, it is not surprising that for a great many systems of interest (and most of those used in computer simulation) that the transition kernel has no dependence on the time. Markov chains that have the same transition kernel at all times are termed *time homogeneous* (or sometimes simply homogeneous) and will be the main focus of this article.

In the time homogeneous context, the n-step transition kernels denoted K^n, which have the property that $\mathbb{P}(X_{m+n} = x_{m+n}|X_m = x_m) = K^n(x_m,x_{m+n})$ may be obtained inductively, as

$$K^n(x_m,x_{m+n}) = \sum_{x_{m+1}} K(x_m,x_{m+1})K^{n-1}(x_{m+1},x_{m+n})$$

for any $n > 1$, whereas $K^1(x_m,x_{m+1}) = K(x_m,x_{m+1})$.

A Matrix Representation

The functional notation above is convenient, as it generalizes to Markov chains on state spaces that are not discrete. However, discrete state space Markov chains exist in abundance in engineering and particularly in computer science. It is convenient to represent probability distributions on finite spaces as a row vector of probability values. To define such a vector μ, simply set $\mu_i = \mathbb{P}(X = i)$ (where X is some random variable distributed according to μ). It is also possible to define a Markov kernel on this space by setting the elements of a matrix K equal to the probability of moving from a state i to a state j; i.e.:

$$K_{ij} = \mathbb{P}(X_n = j|X_{n-1} = i)$$

Although this may appear little more than a notational nicety, it has some properties that make manipulations particularly straightforward; for example, if $X_1 \sim \mu$, then:

$$\begin{aligned}\mathbb{P}(X_2 = j) &= \sum_i \mathbb{P}(X_1 = i)\mathbb{P}(X_2 = j|X_1 = i)\\ &= \sum_i \mu_i K_{ij}\\ &= (\mu K)_j\end{aligned}$$

where μK denotes the usual vector matrix product and $(\mu K)_j$ denotes the jth element of the resulting row vector.In fact, it can be shown inductively that $\mathbb{P}(X_n = j) = (\mu K^{n-1})_j$, where K^{n-1} is the usual matrix power of K. Even more generally, the conditional distributions may be written in terms of the *transition matrix*, K:

$$\mathbb{P}(X_{n+m} = j|X_n = i) = (K^m)_{ij}$$

and so a great many calculations can be performed via simple matrix algebra.

A Graphical Representation

It is common to represent a homogeneous, finite-state space Markov chain graphically. A single directed graph with labeled edges suffices to describe completely the transition matrix of such a Markov chain. Together with the distribution of the initial state, this completely characterizes the Markov chain. The vertices of the graph correspond to the states, and those edges that exist illustrate the moves that it is possible to make. It is usual to label the edges with the probability associated with the move that they represent, unless all possible moves are equally probable.

A simple example, which also shows that the matrix representation can be difficult to interpret, consists of the Markov chain obtained on the space $\{0,1,\ldots,9\}$ in which the next state is obtained by taking the number rolled on an unbiased die and adding it, modulo 10, to the current state *unless a 6 is rolled when the state is 9*, in which case, the chain retains its current value. This has a straightforward, but cumbersome matrix representation, in which:

$$K = \frac{1}{6}\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 1 shows a graphical illustration of the same Markov transition kernel—transition probabilities are omitted in this case, as they are all equal. Although it may initially seem no simpler to interpret than the matrix, on closer inspection, it becomes apparent that one can easily determine from which states it is possible to reach any selected state, which states it is possible to reach from it and those states it is possible to move between in a particular number of moves *without performing any calculations*. It is these properties that this representation makes it very easy to interpret even in the case of Markov chains with large state spaces for which the matrix representation rapidly becomes very difficult to manipulate. Note the *loop* in the graph showing the possibility of remaining in state 9—this is equivalent to the presence of a nonzero diagonal element in the transition matrix.

MARKOV CHAINS ON GENERAL STATE SPACES

In general, more subtle measure theoretic constructions are required to define or study Markov chains on *uncountable* state spaces—such as the real numbers or the points in three-dimensional space. To deal with a fully general state space, a degree of measure theoretic probability beyond that which can be introduced in this article is required. Only Markov chains on some subset of d-dimensional, Euclidean space, $\mathbb{R}^d$ with distributions and transition kernels that admit a density (for definiteness, with respect to Lebesgue measure—that which attributes to any interval mass corresponding to its length—over that space) will be considered here. $K_n(x,y)$ (or $K(x,y)$ in the time homogeneous case) denotes a density with the property that

$$\mathbb{P}(X_n \in A | X_{n-1} = x_{n-1}) = \int_A K_n(x_{n-1},y)dy$$

This approach has the great advantage that many concepts may be written for discrete and continuous state space cases in precisely the same manner, with the understanding that the notation refers to probabilities in the discrete case and densities in the continuous setting. To generalize things, it is necessary to consider Lebesgue integrals with respect to the measures of interest, but essentially, one can replace equalities of densities with those of integrals over any measurable set and the definitions and results presented below will continue to hold. For a rigorous and concise introduction to general state space Markov chains, see Ref. 2. For a much more detailed exposition, Ref. 3 is recommended highly.

STATIONARY DISTRIBUTIONS AND ERGODICITY

The ergodic hypothesis of statistical mechanics claims, loosely, that given a thermal system at equilibrium, the long-term average occurrence of any given system configuration corresponds precisely to the average over an infinite

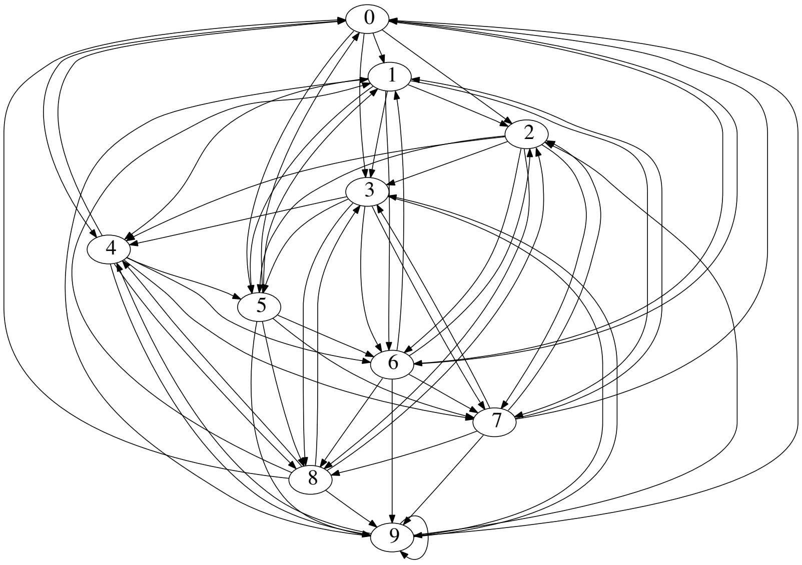


Figure 1. A graphical representation of a Markov chain.

ensemble of identically prepared systems at the same temperature. One area of great interest in the analysis of Markov chains is that of establishing conditions under which a (mathematically refined form of) this assertion can be shown to be true: When are averages obtained by considering those states occupied by a Markov chain over a long period of its evolution close to those that would be obtained by calculating the average under some distribution associated with that chain? Throughout this section, integrals over the state space are used with the understanding that in the discrete case these integrals should be replaced by sums. This process minimizes the amount of duplication required to deal with both discrete and continuous state spaces, which allows the significant differences to be emphasized when they develop.

One of the most important properties of homogeneous Markov chains, particularly within the field of simulation, is that they can admit a *stationary* (or *invariant*) distribution. A transition kernel K is π-stationary if

$$\int \pi(x)K(x,y)dx = \pi(y)$$

That is, given a sample $X = x$ from π, the distribution of a random variable Y, drawn from $K(x,\cdot)$ is the same as that of X, although the two variables are, of course, not independent. In the discrete case, this becomes

$$\sum_i \pi(i)K(i,j) = \pi(j)$$

or, more succinctly, in the matrix representation, $\pi K = \pi$. The last of these reveals a convenient characterization of the stationary distributions, where they exist, of a transition kernel: They are the left eigenvectors (or eigenfunctions in the general state space case) of the transition kernel with an associated eigenvalue of 1. Viewing the transition kernel as an operator on the space of distributions, the same interpretation is valid in the general state space case.

It is often of interest to simulate evolutions of Markov chains with particular stationary distributions. Doing so is the basis of Markov chain Monte Carlo methods and is beyond the scope of this article. However, several theoretical concepts are required to determine when these distributions exist, when they are unique, and when their existence is enough to ensure that a large enough sample path will have similar statistical properties to a collection of independent, identically distributed random variables from the stationary distribution. The remainder of this section is dedicated to the introduction of such concepts and to the presentation of two results that are of great importance in this area.

One property useful in the construction of Markov chains with a particular invariant distributions is that of *reversibility*. A stochastic process is termed reversible if the statistical properties of its time reversal are the same as those of the process itself. To make this concept more formal, it is useful to cast things in terms of certain joint probability distributions. A stationary process is reversible if for any n, m, the following equality holds for all measurable sets $A_n, \ldots, A_{n+m}$:

$$\mathbb{P}(X_n \in A_n, \ldots, X_{n+m} \in A_{n+m}) = \mathbb{P}(X_n \in A_n, \ldots, X_{n-m} \in A_{n+m})$$

It is simple to verify that, in the context of a Markov chain, this is equivalent to the *detailed balance* condition:

$$\mathbb{P}(X_n \in A_n, X_{n+1} \in A_{n+1}) = \mathbb{P}(X_n \in A_{n+1}, X_{n+1} \in A_n)$$

A Markov chain with kernel K is said to satisfy detailed balance for a distribution π if

$$\pi(x)K(x,y) = \pi(y)K(y,x)$$

It is straightforward to verify that, if K is π-reversible, then π is a stationary distribution of K:

$$\begin{aligned} \int \pi(x)K(x,y)dy &= \int \pi(y)K(y,x)dy \\ \pi(x) &= \int \pi(y)K(y,x)dy \end{aligned}$$

This is particularly useful, as the detailed balance condition is straightforward to verify.

Given a Markov chain with a particular stationary distribution, it is important to be able to determine whether, over a long enough period of time, the chain will explore all of the space, which has a positive probability under that distribution. This leads to the concepts of *accessibility, communication structure*, and *irreducibility*.

In the discrete case, a state j is said to be *accessible* from another state i written as $i \rightarrow j$, if for some n, $K^n(i,j) > 0$. That is, a state that is accessible from some starting point is one that can be reached with positive probability in some number of steps. If i is accessible from j and j is accessible from i, then the two states are said to *communicate*, and this is written as $i \leftrightarrow j$. Given the Markov chain on the space $E = \{0,1,2\}$ with transition matrix:

$$K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

it is not difficult to verify that the uniform distribution $\mu = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ is invariant under the action of K. However, if $X_1 = 0$, then $X_n = 0$, for all n: The chain will never reach either of the other states, whereas starting from $X_1 \in \{1,2\}$, the chain will never reach 0. This chain is *reducible*: Some disjoint regions of the state space do not communicate. Furthermore, it has multiple stationary distributions; $(1,0,0)$ and $(0,\frac{1}{2},\frac{1}{2})$ are both invariant under the action of K. In the discrete setting, a chain is *irreducible* if all states communicate: Starting from any point in the state space, any other point may be reached with positive probability in some finite number of steps.

Although these concepts are adequate for dealing with discrete state spaces, a little more subtlety is required in more general settings: As ever, when dealing with probability on continuous spaces, the probability associated with individual states is generally zero and it is necessary to consider integrals over finite regions. The property that is captured by *irreducibility* is that, wherever the chain starts from, a positive probability of it reaches anywhere in the space. To generalize this to continuous state spaces, it suffices to reduce the strength of this statement very slightly: From "most" starting points, the chain has a positive probability of reaching any region of the space that itself has positive probability. To make this precise, a Markov chain of stationary distribution π is said to be π-irreducible if, for all x (except for those lying in a set of exceptional points that has probability 0 under π), and all sets A with the property that $\int_A \pi(x)dx > 0$,

$$\exists n : \int_A K^n(x,y)dy > 0$$

The terms *strongly irreducible* and *strongly* π*-irreducible* are sometimes used when the irreducibility or π-irreducibility condition, respectively, holds for $n = 1$. Notice that any irreducible Markov chain is π-irreducible with respect to any measure π.

These concepts allow us to determine whether a Markov chain has a "joined-up" state space: whether it is possible to move around the entire space (or at least that part of the space that has mass under π). However, it tells us nothing about *when* it is possible to reach these points. Consider the difference between the following two transition matrices on the space $\{0, 1\}$; for example,

$$K = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \text{and } L = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Both matrices admit $\pi = (\frac{1}{2}, \frac{1}{2})$ as a stationary distribution, and both are irreducible. However, consider their respective marginal distributions after several iterations:

$$K^n = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}, \text{whereas } L^n = \begin{cases} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & n \text{ odd} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & n \text{ even} \end{cases}$$

In other words, if $\mu = (\mu_1, \mu_2)$, then the Markov chain associated with K has distribution $\mu K^n = (\frac{1}{2}, \frac{1}{2})$ after n iterations, whereas that associated with L has distribution (μ_2, μ_1) after any odd number of iterations and distribution (μ_1, μ_2) after any even number. L, then, never forgets its initial conditions, and it is *periodic*.

Although this example is contrived, it is clear that such periodic behavior is significant and that a precise characterization is needed. This is straightforward in the case of discrete state space Markov chains. The period of any state, i in the space is defined, using gcd to refer to the greatest common divisor (i.e., the largest common factor), as

$$d = \gcd\{n : K^n(i,i) > 0\}$$

Thus, in the case of L, above, both states have a period $d = 2$. In fact, it can be shown easily, that any pair of states that communicate must have the same period. Thus, irreducible Markov chains have a single period, 2, in the case of L, above and 1, in the case of K. Irreducible Markov chains may be said to have a period themselves, and when this period is 1, they are termed *aperiodic*.

Again, more subtlety is required in the general case. It is clear that something is needed to fill the role that individual states play in the discrete state space case and that individual states are not appropriate in the continuous case. A set of events that is small enough that it is, in some sense, homogeneous and large enough that it has positive probability under the stationary distribution is required. A set C is termed *small* if some integer n, some probability distribution ν, and some $\epsilon > 0$ exist such that the following condition holds:

$$\inf_{x \in C} K^n(x,y) \geq \epsilon \nu(y)$$

This condition tells us that for any point in C, with probability ϵ, the distribution of the next state the chain enters is independent of *where in* C it is. In that sense, C is small, and these sets are precisely what is necessary to extend much of the theory of Markov chains from the discrete state space case to a more general setting. In particular, it is now possible to extend the notion of period from the discrete state space setting to a more general one. Note that in the case of irreducible aperiodic Markov chains on a discrete state space, the entire state space is small.

A Markov chain has a cycle of length d if a small set C exists such that the greatest common divisor of the length of paths from C to a measurable set of positive probability B is d. If the largest cycle possessed by a Markov chain has length 1, then that chain is *aperiodic*. In the case of π-irreducible chains, every state has a common period (except a set of events of probability 0 under π), and the above definition is equivalent to the more intuitive (but more difficult to verify) condition, in which a partition of the state space, E, into d disjoint subsets $E_1, \ldots, E_d$ exists with the property that $\mathbb{P}(X_{n+1} \neq \in E_j | X_n \in E_i) = 0$ if $j = i + 1 \bmod d$.

Thus far, concepts that allow us to characterize those Markov chains that can reach every important part of the space and that exhibit no periodic structure have been introduced. Nothing has been said about *how often* a given region of the space might be visited. This point is particularly important: A qualitative difference exists between chains that have a positive probability of returning to a set infinitely often and those that can only visit it finitely many times. Let η_A denote the number of times that a set A is visited by a Markov chain; that is, $\eta_A = |\{X_n \in A : n \in \mathbb{N}\}|$. A π-irreducible Markov chain is recurrent if $\mathbb{E}[\eta_A] = \infty$ for every A with positive probability under π. Thus, a recurrent Markov chain is one with positive probability of visiting any significant (with respect to π) part of the state space infinitely often: It does not always escape to infinity. A slightly stronger condition is termed *Harris recurrence*; it requires that every significant state *is* visited infinitely often (rather than this event having positive probability); i.e., $\mathbb{P}(\eta_A = \infty) = 1$ for every set A for which $\int_A \pi(x)dx > 0$. A Markov chain that is not recurrent is termed *transient*.

The following example illustrates the problems that can originate if a Markov chain is π-recurrent but *not* Harris recurrent. Consider the Markov chain over the positive integers with the transition kernel defined by

$$K(x,y) = x^{-2}\delta_1(y) + (1 - x^{-2})\delta_{x+1}(y)$$

where for any state, x, δ_x denotes the probability distribution that places all of its mass at x. This kernel is clearly δ_1-recurrent: If the chain is started from 1, it stays there deterministically. However, as the sum

$$\sum_{k=2}^{\infty} \frac{1}{k^2} < \infty$$

the Borel–Cantelli lemma ensures that whenever the chain is started for any x greater than 1, a positive probability exists that the chain will *never* visit state 1—the chain is π-recurrent, but it is not Harris recurrent. Although this example is somewhat contrived, it illustrates an important phenomenon—and one that often cannot be detected easily in more sophisticated situations. It has been suggested that Harris recurrence can be interpreted as a guarantee that no such pathological system trajectories exist: No parts of the space exist from which the chain will escape to infinity rather than returning to the support of the stationary distribution.

It is common to refer to a π-irreducible, aperiodic, recurrent Markov chain as being ergodic and to an ergodic Markov chain that is also Harris recurrent as being *Harris ergodic*. These properties suffice to ensure that the Markov chain will, on average, visit every part of the state space in proportion to its probability under π, that it exhibits no periodic behavior in doing so, and that it might (or will, in the Harris case) visit regions of the state space with positive probability infinitely often. Actually, ergodicity tells us that a Markov chain eventually forgets its initial conditions—after a sufficiently long time has elapsed, the current state provides arbitrarily little information about the initial state. Many stronger forms of ergodicity provide information about the *rate* at which the initial conditions are forgotten; these are covered in great detail by Meyn and Tweedie (3). Intuitively, if a sequence of random variables forgets where it has been, but has some stationary distribution, then one would expect the distribution of sufficiently widely separated samples to approximate that of independent samples from that stationary distribution. This intuition can be made rigorous and is strong enough to tell us a lot about the distribution of large samples obtained by iterative application of the Markov kernel and the sense in which approximations of integrals obtained by using the empirical average obtained by taking samples from the chain might converge to their integral under the stationary measure. This section is concluded with two of the most important results in the theory of Markov chains.

The *ergodic theorem* provides an analog of the law of large numbers for independent random variables: It tells us that under suitable regularity conditions, the averages obtained from the sample path of a Markov chain will converge to the expectation under the stationary distribution of the transition kernel. This mathematically refined, rigorously proved form of the ergodic hypothesis was alluded to at the start of this section. Many variants of this theorem are available; one particularly simple form is the following: If $\{X_n\}$ is a Harris ergodic Markov chain of invariant distribution π, then the following strong law of large numbers holds for any π-integrable function $f : E \to \mathbb{R}$ (convergence is with probability one):

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} f(X_i) \to \int f(x)\pi(x)dx$$

This is a particular case of Ref. 5 (p. 241, Theorem 6.63), and a proof of the general theorem is given there. The same theorem is also presented with proof in Ref. 3 (p. 433, Theorem 17.3.2).

A *central limit theorem* also exists and tells us something about the rate of convergence of averages under the sample path of the Markov chain. Under technical regularity conditions (see Ref. 4 for a summary of various combinations of conditions), it is possible to obtain a central limit theorem for the ergodic averages of a Harris recurrent, π-invariant Markov chain, and a function that has at least two finite moments, $f : E \to \mathbb{R}$ (with $\mathbb{E}[f] < \infty$ and $\mathbb{E}[f^2] < \infty$).[1]

$$\lim_{n\to\infty} \sqrt{n}\left[\frac{1}{n}\sum_{i=1}^{n} f(X_i) - \int f(x)\pi(x)dx\right] \xrightarrow{d} \mathcal{N}(0, \sigma^2(f))$$

$$\sigma^2(f) = \mathbb{E}[(f(X_1) - \bar{f})^2] + 2\sum_{k=2}^{\infty} \mathbb{E}[(f(X_1) - \bar{f})(f(X_k) - \bar{f})]$$

where $\xrightarrow{d}$ denotes convergence in distribution, $\mathcal{N}(0,\sigma^2)$ is the normal distribution of mean 0, and variance $\sigma^2(f)$ and $\bar{f} = \int f(x)\pi(x)dx$.

A great many refinements of these results exist in the literature. In particular, cases in which conditions may be relaxed or stronger results proved have been studied widely. It is of particular interest in many cases to obtain quantitative bounds on the rate of convergence of ergodic averages to the integral under the stationary distribution. In general, it is very difficult to obtain meaningful bounds of this sort for systems of real practical interest, although some progress has been made in recent years.

SELECTED EXTENSIONS AND RELATED AREAS

It is unsurprising that a field as successful as that of Markov chains has several interesting extensions and related areas. This section briefly describes two of these areas.

So-called *adaptive Markov chains* have received a significant amount of attention in the field of Monte Carlo methodology in recent years. In these systems, the transition kernel used at each iteration is adjusted depending on

[1] Depending on the combination of regularity condition assumed, it may be necessary to have a finite moment of order $2 + \delta$.

the entire history of the system or some statistical summary of that history. Although these adaptive systems are attractive from a practical viewpoint, as they allow for automatic tuning of parameters and promise simpler implementation of Monte Carlo methods in the future, a great deal of care must be taken when analyzing them. It is important to notice that because the transition kernel depends on more than the current state at the time of its application, it does not give rise to a Markov chain.

Feynman–Kac formulas were first studied in the context of describing physical particle motion. They describe a sequence of probability distributions obtained from a collection of Markov transition kernels M_n and a collection of potential functions G_n. Given a distribution η_{n-1} at time $n-1$, the system is *mutated* according to the transition kernel to produce an updated distribution:

$$\hat{\eta}_n(x_n) = \int \eta_{n-1}(x_{n-1}) M_n(x_{n-1}, x_n) dx_{n-1}$$

before weighting the probability of each state/region of the space according to the value of the potential function:

$$\eta_n(x_n) = \frac{\hat{\eta}_n(x_n) G_n(x_n)}{\int \hat{\eta}_n(x) G_n(x) dx}$$

Many convenient ways of interpreting such sequences of distributions exist. One way is that if η_{n-1} describes the distribution of a collection of particles at time $n-1$, which have dynamics described by the Markov kernel M_n in an absorbing medium that is described by the potential function G_n (in the sense that the smaller the value of G_n at a point, the greater the probability that a particle at that location is absorbed), then η_n describes the distribution of those particles that have not been absorbed at time n. These systems have found a great deal of application in the fields of Monte Carlo methodology, particularly sequential and population-based methods, and genetic algorithms. The latter method gives rise to another interpretation: The Markov kernel can be observed as describing the mutation undergone by individuals within a population, and the potential function $G_n(x_n)$ the fitness of an individual with genetic makeup x_n, which governs the success of that individual in a selection step.

Alternatively, one can view the evolution of a Feynman–Kac system as a *nonlinear Markov Chain* in which the distribution of X_n depends on both X_{n-1} and its distribution η_{n-1}. That is, if $X_{n-1} \sim \eta_{n-1}$, then the distribution of X_n is given by

$$\eta_n(\cdot) = \int \eta_{n-1}(x_{n-1}) K_{n,\eta_{n-1}}(x_{n-1}, \cdot) dx_{n-1}$$

where the nonlinear Markov kernel K_{n,η_n} is defined as the composition of selection and mutation steps (numerous such kernels may be associated with any particular Feynman–Kac flow).

An excellent monograph on Feynman–Kac formulas and their mean field approximations has been written recently (6).

BIBLIOGRAPHY

1. A. N. Kolmogorov, *Foundations of the Theory of Probability*, Chelsea Publishing Company, 2nd ed., 1956.
2. E. Nummelin, *General Irreducible Markov Chains and Non-Negative Operators*, Number 83 in Cambridge Tracts in Mathematics, 1st ed., Cambridge, UK: Cambridge University Press, 1984.
3. S. P. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability*, Berlin Springer Verlag, 1993.
4. G. L. Jones, On the Markov chain central limit theorem, *Probability Surv.* **1**: 299–320, 2004.
5. C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2nd ed., New York: Springer Verlag, 2004.
6. P. Del Moral, *Feynman-Kac formulae: genealogical and interacting particle systems with applications*. Probability and Its Applications. New York: Springer Verlag, 2004.

Adam M. Johansen
University of Bristol
Bristol, United Kingdom

M

MIXED INTEGER PROGRAMMING

INTRODUCTION

A (linear) mixed integer program (MIP) is an optimization problem over a set of integer variables (unknowns) and a set of real-valued (continuous) variables, the constraints are all linear equations or inequalities, and the objective is a linear function to be minimized (or maximized). This can be written mathematically as

$$\begin{aligned}&\min cx + hy\\ &Ax + Gy \geq b\\ &x \in \mathbb{R}^p_+, y \in \mathbb{Z}^n_+\end{aligned}$$

where $\mathbb{R}^p_+$ denotes the space of p-dimensional non-negative real vectors: $\mathbb{Z}^n_+$ denotes the space of n-dimensional non-negative integer vectors: x denotes the p continuous variables: y, denotes the n integer variables; A and G are $m \times p$ and $m \times n$ matrices, respectively; b is an m-vector (the requirements or right-hand side vector) and c and h are p-and n-dimensional row vectors. Often one specifically distinguishes constraints of the form $l \leq x \leq u$ or $l' \leq y \leq u'$, known as lower and upper bound constraints—thus, the problem above is the special case where all the coordinates of l and l' are zero and all the coordinates of u and u' are $+\infty$.

MIPs in which $p = 0$ are called (linear) integer programs (IP): those in which the bounds on the integer variables are all 0 and 1 are called binary or 0–1 MIPs, and those in which $n = 0$ are called linear programs (LP). A problem in which the objective function and/or constraints involve nonlinear functions of the form

$$\min\{c(x,y) : g_i(x,y) \geq b_i \, i = 1,\ldots,m, x \in \mathbb{R}^p_+, y \in \mathbb{Z}^n_+\}$$

is a mixed integer nonlinear program (MINLP).

MIPs in general, are difficult problems ($\mathcal{NP}$—hard in the complexity sense), as are 0–1 MIPs and IPs. However, LPs are easy (polynomially solvable), and linear programming plays a very significant role in both the theory and practical solution of linear MIPs. Readers are referred to the entry "linear programming" for background and some basic terminology.

We now briefly outline what follows. In "The Formulation of various MIPs" section, we present the formulations of three typical optimization problems as mixed integer programs. In the "Basic Properties of MIPs" section, we present some basic properties of MIPs that are used later. In "The Branch-and-Cut Algorithms for MIPs" section, we explain how the majority of MIPs are solved in practice. All the state-of-the-art solvers use a combination of linear programming combined with intelligent enumeration (known as branch-and-bound), preprocessing using simple tests to get a better initial representation of the problem, reformulation with additional constraints (valid inequalities or cutting planes), extended reformulation with additional variables, and heuristics to try to find good feasible solutions quickly.

In the "References and Additional Topics" section, we give references for the basic material described in this article and for more advanced topics, including decomposition algorithms, MIP test instances, MIP modeling and optimization software, and nonlinear MIPs.

THE FORMULATION OF VARIOUS MIPS

MIPs are solved on a regular basis in many areas of business, management, science and engineering. Modeling problems as MIPs is nontrivial. One needs to define first the unknowns (or variables), then a set of linear constraints so as to characterize exactly the set of feasible solutions, and finally a linear objective function to be minimized or maximized. Here we present three simplified problems that exemplify such applications.

A Capacitated Facility Location Problem

Given m clients with demands a_i for i =1,..., m, n potential depots with annual throughput of capacity b_j for j =1,..., n where the annual cost of opening depot j is f_j, suppose that the potential cost of satisfying one unit of demand of client i from depot j is c_{ij}. The problem is to decide which depots to open so as to minimize the total annual cost of opening the depots and satisfying the annual demands of all the clients.

Obviously, one wishes to know which set of depots to open and which clients to serve from each of the open depots. This situation suggests the introduction of the following variables:

y_j =1 if depot j is opened, and y_j=0 otherwise.
x_{ij} is the amount shipped from depot j to client i.

The problem now can be formulated as the following MIP:

$$\min \sum_{j=1}^{n} f_i y_j + \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

$$\sum_{j=1}^{n} x_{ij} = a_i \quad \text{for} \quad i = 1,\ldots,m \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} \leq b_j y_j \quad \text{for} \quad j = 1,\ldots,n \tag{3}$$

$$x \in \mathbb{R}^{mn}_+, y \in \{0,1\}^n \tag{4}$$

where the objective function in Equation (1) includes terms for the fixed cost of opening the depots and for the variable shipping costs, the constraints in Equation (2) ensure that

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

the demand of each client is satisfied, the constraints in Equation (3) ensure that, if depot j is closed, nothing is shipped from that depot, and otherwise at most b_j is shipped, and finally Equation (4) indicates the variable types and bounds.

A Multi-Item Production Planning Problem

Given m items to be produced over a time horizon of n periods with known demands d_t^i for $1 \le i \le m, 1 \le t \le n$, all the items have to be processed on a single machine. The capacity of the machine in period t is L_t, and the production costs are as follows: A fixed set-up cost of q_t^i exists if item i is produced in period t, as well as a per unit variable cost p_t^i; a storage cost h_t^i exists for each unit of i in stock at the end of period t. The problem is to satisfy all the demands and minimize the total cost.

Introducing the variables

x_t^i is the amount of item i produced in period t,

s_t^i is the amount of item i in stock at the end of period t,

$y_t^i = 1$ if there is production of item i in period t, and $y_t^i = 0$ otherwise,

a possible MIP formulation of the problem is:

$$\begin{aligned}&\min \sum_{i=1}^{m}\sum_{t=1}^{n}\left(p_t^i x_t^i + q_t^i y_t^i + h_t^i s_t^i\right)\\ &s_{t-1}^i + x_t^i = d_t^i + s_t^i \text{ for } i = 1,\ldots,m, t = 1,\ldots,n\end{aligned} \tag{5}$$

$$x_t^i \le L_t y_t^i \quad \text{for} \quad i = 1,\ldots,m, t = 1,\ldots,n \tag{6}$$

$$\begin{aligned}&\sum_{i=1}^{m} x_t^i \le L_t \quad \text{for} \quad t = 1,\ldots,n\\ &s, x \in \mathbb{R}_+^{mn}, y \in \{0,1\}^{mn}\end{aligned} \tag{7}$$

where constraints in Equation (5) impose conservation of product from one period to the next (namely end-stock of item i in $t-1$ + production of i in t = demand for i in t + end-stock of i in t), Equation (6) forces the set-up variable y_t^i to 1 if there is production ($x_t^i > 0$), and Equation (7) ensures that the total amount produced in period t does not exceed the capacity.

Traveling Salesman Problem with Time Windows

Suppose that a truck (or salesperson) must leave the depot, visit a set of n clients, and then return to the depot. The travel times between clients (including the depot, node 0) are given in an $(n+1) \times (n+1)$ matrix c. Each client i has a time window $[a_i, b_i]$ during which the truck must make its delivery. The delivery time is assumed to be negligible. The goal is to complete the tour and return to the depot as soon as possible while satisfying the time window constraints of each client. Two possible sets of variables are

$y_{ij} = 1$ if the truck travels directly from i to j, $i, j \in \{0, 1 \ldots, n\}$.

t_j is the time of delivery to client j, $j \in \{0,\ldots,n\}$.

τ is the time of return to the depot.

The problem now can be formulated as the following MIP:

$$\min \tau - t_0 \tag{8}$$

$$\sum_{j=0}^{n} y_{ij} = 1, \quad i \in \{0,\ldots,n\} \tag{9}$$

$$\sum_{i=0}^{n} y_{ij} = 1, \quad j \in \{0,\ldots,n\} \tag{10}$$

$$\sum_{i \in S, j \notin S} y_{ij} \ge 1, \quad \phi \subset S \subseteq \{1,\ldots,n\} \tag{11}$$

$$\begin{aligned}&t_j \ge t_i + c_{ij} y_{ij} - M(1 - y_{ij}),\\ &\quad i \in \{0,\ldots,n\}, j \in \{1,\ldots,n\}\end{aligned} \tag{12}$$

$$\tau \ge t_i + c_{i0} y_{i0} - M(1 - y_{i0}), \quad i \in \{1,\ldots,n\} \tag{13}$$

$$\begin{aligned}&a_i \le t_i \le b_i, \quad i \in \{1,\ldots,n\}\\ &\tau \in \mathbb{R}, t \in \mathbb{R}^{n+1}, y \in \{0,1\}^{\frac{(n+1)(n+2)}{2}}\end{aligned} \tag{14}$$

where M is a large value exceeding the total travel time, the objective function in Equation (8) measures the difference between the time of leaving and returning to the depot, Equations (9) and (10) ensure that the truck leaves/arrives once at site i and Equation (11) ensures that the tour of the truck is connected. Constraints in Equations (12) and (13) ensure that if the truck goes from i to j, the arrival time in j is at least the arrival time in i plus the travel time from i to j. Finally, Equation (14) ensures that the time window constraints are satisfied.

BASIC PROPERTIES OF MIPS

Given the MIP

$$z = \min\{cx + hy : Ax + Gy \ge b, x \in \mathbb{R}_+^p, y \in \mathbb{Z}_+^n\}$$

several important properties help us to understand such problems. In particular the linear program obtained by dropping the integrality constraints on the y variables:

$$z_{LP} = \min\{cx + hy : Ax + Gy \ge b, x \in \mathbb{R}_+^p, y \in \mathbb{Z}_+^n\}$$

is called the *linear relaxation* of the original MIP.

Observation 1. *Considering the MIP and its linear relaxation:*

(i) $z_{LP} \le z$, *and*

(ii) *if* (x^*, y^*) *is an optimal solution of LP and* y^* *is integral, then* (x^*, y^*) *is an optimal solution of MIP.*

Definition 2. *A set of the form* $\{x \in \mathbb{R}^n : Ax \ge b\}$ *with* A *an* $m \times n$ *matrix is a polyhedron.*

The convex hull *of a set of points* $X \subseteq \mathbb{R}^n$ *is the smallest convex set containing* X, *denoted* conv(X).

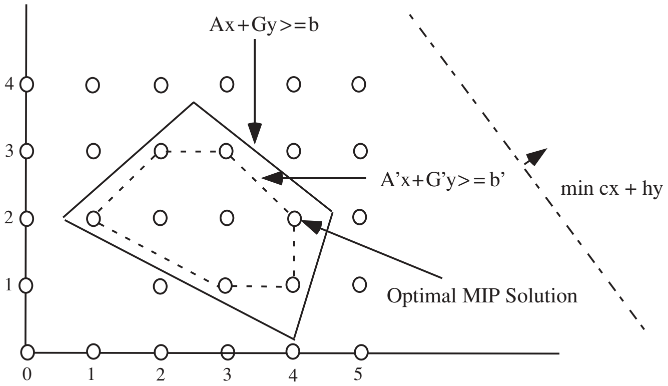


Figure 1. Linear Relaxation and Convex Hull.

Observation 3. *The set* $X_{MIP} = \{(x,y) \in \mathbb{R}^p_+ \times \mathbb{Z}^n_+ : Ax + Gy \geq b\}$*is known as the feasible region of the MIP. When* A, G, b *are rational matrices, then*

(i) $\text{conv}(X_{MIP})$ *is a polyhedron, namely* $\text{conv}(X_{MIP}) = \{(x,y) \in \mathbb{R}^{p+n}_+ : A'x + G'y \geq b'\}$*for some* A', G', b', *and*

(ii) *the linear program* $\min\{cx + hy : (x,y) \in \text{conv}\,(X_{MIP})\}$ *solves MIP. In Fig.1 one sees tha t an optimal vertex of* $\text{conv}(X_{MIP})$ *lies in* X_{MIP}.

The last observation suggests that it it is easy to solve an MIP. Namely, it suffices to find the convex hull of the set of feasible solutions and then solve a linear program. Unfortunately, it is rarely this simple. Finding $\text{conv}(X_{MIP})$ is difficult, and usually an enormous number of inequalities are needed to describe the resulting polyhedron.

Thus, one typically must be less ambitious and examine

(i) whether certain simple classes of MIPs exist for which one can find an exact description of $\text{conv}(X_{\text{MIP}})$, and
(ii) whether one can find a good approximation of $\text{conv}(X_{\text{MIP}})$ by linear inequalities in a reasonable amount of time.

Below, in looking at ways to find such a description of X_{MIP} and in using it in solving an MIP, we often will mix two distinct viewpoints:

(i) Find sets $X^1, \ldots, X^k$ such that

$$X_{\text{MIP}} = \bigcup_{i=1}^{k} X^i$$

where optimizing over X^i is easier than optimizing over X_{MIP} for $i = 1, \ldots, k$, and where possibly good descriptions of the sets $\text{conv}(X^i)$ are known. This decomposition forms the basis of the branch-and-bound approach explained in the next section.

(ii) Find sets $X^1, \ldots, X^k$ such that

$$X_{\text{MIP}} = \bigcap_{i=1}^{k} X^i$$

where a good or exact description of $\text{conv}(X^i)$ is known for $i = 1, \ldots, k$. Then, a potentially effective approximation to $\text{conv}(X_{\text{MIP}})$ is given by the set $\cap_{i=1}^{k}\text{conv}(X^i)$. This decomposition forms the basis of the preprocessing and cut generation steps used in the branch-and-cut approach.

THE BRANCH-AND-CUT ALGORITHM FOR MIPs

Below we will examine the main steps contributing to a branch-and-cut algorithm. The first step is the underlying branch-and-bound algorithm. This algorithm then can be improved by (1) a priori reformulation, (2) preprocessing, (3) heuristics to obtain good feasible solutions quickly, and finally (4) cutting planes or dynamic reformulation in which case we talk of a branch-and-cut algorithm.

Branch-and-Bound

First, we discuss the general ideas, and then we discuss how they typically are implemented. Suppose the MIP to be solved is $z = \min\{cx + hy : (x,y) \in X_{\text{MIP}}\}$ with

$$X_{MIP} = \bigcup_{i=1}^{k} X^i$$

where $X^i = \{(x,y) \in \mathbb{R}^p_+ \times \mathbb{Z}^n_+ : A^i x + G^i y \geq b^i\}$ for $i = 1, \ldots, k$. In addition suppose that we know the value of each linear program

$$z^i_{LP} = \min\{cx + hy : A^i x + G^i y \geq b^i,\ x \in \mathbb{R}^p_+,\ y \in \mathbb{R}^n_+\}$$

and the value $\bar{z}$, of the best known feasible solution $(\bar{x}, \bar{y})$ of MIP found so far, known as the incumbent value.

Observation 4.

(i) $\bar{z} \geq z$ *and* $z \geq \min_i z^i_{LP}$ for $i = 1, \ldots, k$.
(ii) *If* $z^i_{LP} \geq \bar{z}$ *for some i, then no feasible solution with an objective value better than that of the incumbent lies in* X^i. *Thus, the set* X^i *has been enumerated implicitly, and can be ignored (pruned by bound).*
(iii) *If* $z^i_{LP} \geq \bar{z}$ *and the optimal solution of the linear program corresponding to* X^i *has y integer, then using Observation 1, this solution is feasible and optimal in* X^i *and feasible in* X_{MIP}. *Now the incumbent value* $\bar{z}$ *can be improved* $\bar{z} \leftarrow z^i_{LP}$, *and the set* X^i *has been enumerated implicitly and, thus, can be ignored (pruned by optimality).*

Now we outline the steps of the algorithm.

A list L contains a list of unexplored subsets of X_{MIP}, each possibly with some lower bound, as well as an incumbent value $\bar{z}$. Initially, the list just contains X_{MIP} and $\bar{z} = \infty$.

If the list is empty, stop. The best solution found so far is optimal and has value $\bar{z}$.

Otherwise, select and remove a set X^t from the list L.

Solve the corresponding linear program (with optimal solution $(\bar{x}^t, \bar{y}^t)$ and value z^t_{LP}). If it is infeasible, so $X^t = \phi$, or if one of the conditions ii) or iii) of Observation 4 hold, we update the incumbent if appropriate, prune X^t, and return to the list.

If the node is not pruned ($\bar{z} > z^t_{\mathrm{LP}}$ and $\bar{y}^t$ is fractional), we have not succeeded in finding the optimal solution in X^t, so we branch (i.e., break the set X^t into two or more pieces). As the linear programming solution was not integral, some variable y_j takes a fractional value $\bar{y}^t_j$. The simplest and most common branching rule is to replace X^t by two new sets

$$X^t_{\leq} = X^t \cap \{(x,y) : y_j \leq \lfloor \bar{y}^t_j \rfloor\},$$
$$X^t_{\geq} = X^t \cap \{(x,y) : y_j \geq \lceil \bar{y}^t_j \rceil\}$$

whose union is X^t. The two new sets are added to the list L, and the algorithm continues.

Obvious questions that are important in practice are the choice of branching variable and the order of selection/removal of the sets from the list L. "Good" choices of branching variable can reduce significantly the size of the enumeration tree. "Pseudo-costs" or approximate dual variables are used to estimate the costs of different variable choices. "Strong branching" is very effective—this involves selecting a subset of the potential variables and temporarily branching and carrying out a considerable number of dual pivots with each of them to decide which is the most significant variable on which to finally branch. The order in which nodes/subproblems are removed from the list L is a compromise between different goals. At certain moments one may use a depth-first strategy to descend rapidly in the tree so as to find feasible solutions quickly: however, at other moments one may choose the node with the best bound so as not to waste time on nodes that will be cut off anyway once a better feasible solution is found.

The complexity or running time of the branch-and-bound algorithm obviously depends on the number of subsets X^t that have to be examined. In the worst case, one might need to examine 2^n such sets just for a 0–1 MIP. Therefore, it is crucial to improve the formulation so that the value of the linear programming relaxation gives better bounds and more nodes are pruned, and/or to find a good feasible solution as quickly as possible.

Ways to improve the formulation, including preprocessing, cutting planes and a priori reformulation, are discussed in the next three subsections. The first two typically are carried out as part of the algorithm, whereas the MIP formulation given to the algorithm is the responsibility of the user.

Preprocessing

Preprocessing can be carried out on the initial MIP, as well as on each problem X^t taken from the list L. The idea is to improve the formulation of the selected set X^t. This action typically involves *reducing the number of constraints and variables* so that the linear programming relaxation is solved much faster and *tightening the bounds* so as to increase the value of the lower bound z^t_{LP}, thereby increasing the chances of pruning X^t or its descendants.

A variable can be eliminated if its value is fixed. Also, if a variable is unrestricted in value, it can be eliminated by substitution. A constraint can be eliminated if it is shown to be redundant. Also, if a constraint only involves one variable, then it can be replaced by a simple bound constraint. These observations and similar, slightly less trivial observations often allow really dramatic decreases in the size of the formulations.

Now we give four simple examples of the bound tightening and other calculations that are carried out very rapidly in preprocessing:

(i) (Linear Programming) Suppose that one constraint of X^t is $\sum_{j \in N_1} a_j x_j - \sum_{j \in N_2} a_j x_j \geq b, a_j > 0$ for all $j \in N_1 \cup N_2$ and the variables have bounds $l_j \leq x_j \leq u_j$.

If $\sum_{j \in N_1} a_j u_j - \sum_{j \in N_2} a_j l_j < b$, then the MIP is infeasible

If $\sum_{j \in N_1} a_j l_j - \sum_{j \in N_2} a_j u_j \geq b$, then the constraint is redundant and can be dropped.

For a variable $t \in N_1$, we have $a_t x_t \geq b + \sum_{j \in N_2} a_j x_j - \sum_{j \in N_1 \setminus \{t\}} a_j\, x_j \geq b + \sum_{j \in N_2} a_j\, l_j - \sum_{j \in N_1 \setminus \{t\}} a_j u_j$. Thus, we have the possibly improved bound on x_t

$$x_t \geq \max[l_t, \frac{b + \sum_{j \in N_2} a_j l_j - \sum_{j \in N_1 \setminus \{t\}} a_j u_j}{a_t}]$$

One also possibly can improve the upper bounds on x_j for $j \in N_2$ in a similar fashion.

(ii) (Integer Rounding) Suppose that the bounds on an integer variable $l_j \leq y_j \leq u_j$ just have been updated by preprocessing. If $l_j, u_j \notin \mathbb{Z}$, then these bounds can be tightened immediately to

$$\lceil l_j \rceil \leq y_j \leq \lfloor u_j \rfloor$$

(iii) (0-1 Logical Deductions) Suppose that one of the constraints can be put in the form $\sum_{j \in N} a_j y_j \leq b, y_j \in \{0,1\}$ for $j \in N$ with $a_j > 0$ for $j \in N$.

If $b < 0$, then the MIP is infeasible.

If $a_j > b \geq 0$, then one has $y_j = 0$ for all points of X_{MIP}.

If $a_j + a_k > b \geq \max\{a_j, a_k\}$, then one obtains the simple valid inequality $y_j + y_k \leq 1$ for all points of X_{MIP}.

(iv) (Reduced cost fixing) Given an incumbent value $\overline{z}$ from the best feasible solution, and a representation of the objective function in the form $z^t_{LP} + \sum_j \overline{c}_j x_j + \sum_j \tilde{c}^j y_j$ with $\overline{c}_j \geq 0$ and $\tilde{c}_j \geq 0$ obtained by linear programming, any better feasible solution in X^t must satisfy

$$\sum_j \overline{c}_j x_j + \sum_j \tilde{c}_j y_j < \overline{z} - z^t_{LP}$$

Thus, any such solution satisfies the bounds $x_j \leq \frac{\overline{z} - z^t_{LP}}{c_j}$ and $y_j \leq \lfloor \frac{\overline{z} - z^t_{LP}}{c_j} \rfloor$. (Note that reductions such as in item iv) that take into account the objective function actually modify the feasible region X_{MIP}).

Valid Inequalities and Cutting Planes

Definition 5. *An inequality* $\sum_{j=1}^{p} \pi_j x_j + \sum_{j=1}^{n} \mu_j y_j \geq \pi_0$ is a valid inequality (VI) for X$_{MIP}$ if it is satisfied by every point of X_{MIP}.

The inequalities added in preprocessing typically are very simple. Here, we consider all possible valid inequalities, but because infinitely many of them exist, we restrict our attention to the potentially interesting inequalities. In Figure 1, one sees that only a finite number of inequalities (known as facet-defining inequalities) are needed to describe $\text{conv}(X_{MIP})$. Ideally, we would select a facet-defining inequality among those cutting off the present linear programming solution (x^*, y^*) Formally, we need to solve the

Separation Problem: Given X_{MIP} and a point $(x^*, y^*) \in \mathbb{R}^p_+ \times \mathbb{R}^n_+$ either show that $(x^*, y^*) \in \text{conv}(X_{MIP})$, or find a valid inequality $\pi x + \mu y \geq \pi_0$ for X_{MIP} cutting off $(x^*, y^*)(\pi x^* + \mu y^* < \pi_0)$.

Once one has a way of finding a valid inequality cutting off noninteger points, the idea of a cutting plane algorithm is very natural. If the optimal linear programming solution (x^*, y^*) for the initial feasible set X_{MIP} has y^* fractional and a valid inequality cutting off the point is known (for example, given by an algorithm for the Separation Problem), then the inequality is added, the linear program is resolved, and the procedure is repeated until no more cuts are found. Note that this process changes the linear programming representation of the set X_{MIP} and that this new representation must be used from then on.

Below we present several examples of cutting planes.

(i) (Simple Mixed Integer Rounding) Consider the MIP set $X = \{(x, y) \in \mathbb{R}^1_+ \times \mathbb{Z}^1 : y \leq b + x\}$ It can be shown that every point of X satisfies the valid inequality

$$y \leq \lfloor b \rfloor + \frac{x}{1 - f}$$

where $f = b - \lfloor b \rfloor$ is the fractional part of b.

(ii) (Mixed Integer Rounding) Consider an arbitrary row or combination of rows of X_{MIP} of the form:

$$\sum_{j \in P} a_j x_j + \sum_{j \in N} g_j y_j \leq b$$
$$x \in \mathbb{R}^p_+, y \in \mathbb{Z}^n_+$$

Using the inequality from i), it is easy to establish validity of the mixed integer rounding (MIR) inequality:

$$\sum_{j \in P: a_j < 0} \frac{a_j}{1 - f} x_j + \sum_{j \in N} (\lfloor g_j \rfloor + \frac{(f_j - f)^+}{1 - f}) y_j \leq \lfloor b \rfloor$$

where $f = b - \lfloor b \rfloor$ and $f_j = g_j - \lfloor g_j \rfloor$. The Separation Problem for the complete family of MIR inequalities derived from all possible combinations of the initial constraints, namely all single row sets of the form:

$$uAx + uGy \geq ub, x \in \mathbb{R}^p_+, y \in \mathbb{Z}^n_+$$

where $u \geq 0$, is $\mathcal{NP}$-hard.

(iii) (Gomory Mixed Integer Cut) When the linear programming solution $(x*, y*) \notin X_{MIP}$, some row exists from a representation of the optimal solution of the form

$$y_u + \sum_{j \in P_u} a_j x_j + \sum_{j \in N_u} g_j y_j = a_0$$

with $y^*_u = a_0 \notin \mathbb{Z}^1, x^*_j = 0$ for $j \in P_u$,and $y^*_u = 0$ for $j \in N_u$. Applying the MIR inequality and then eliminating the variable y_u by substitution, we obtain the valid inequality

$$\sum_{j \in P^u : a_j > 0} a_j x_j - \sum_{j \in P^u : a_j < 0} \frac{f_0}{1 - f_0} a_j x_j + \sum_{j \in N^u : f_j \leq f_0} f_j y_j$$
$$+ \sum_{j \in N^u : f_j > f_0} \frac{f_0(1 - f_j)}{1 - f_0} y_j \geq f_0$$

called the Gomory mixed integer cut, where $f_j = g_j - \lfloor g_j \rfloor$ *for* $j \in N^u \cup \{0\}$. This inequality cuts off the LP solution (x^*, y^*). Here, the Separation Problem is trivially solved by inspection and finding a cut is guaranteed. However, in practice the cuts may be judged ineffective for a variety of reasons, such as very small violations or very dense constraints that slow down the solution of the LPs.

(iv) (0–1 MIPs and Cover Inequalities) Consider a 0–1 MIP with a constraint of the form

$$\sum_{j \in N} g_j y_j \leq b + x, x \in \mathbb{R}^1_+, y \in \mathbb{Z}^{|N|}_+$$

with $g_j > 0$ for $j \in N$. A set $C \subseteq N$ is a cover if $\sum_{j \in c} g_j = b + \lambda$ with $\lambda > 0$. The MIP cover inequality is

$$\sum_{j \in C} y_j \leq |C| - 1 + \frac{x}{\lambda}$$

Using appropriate multiples of the constraints $y_j \geq 0$ and $y_j \leq 1$, the cover inequality can be obtained as a weakening of an MIR inequality. When $x = 0$, the Separation Problem for such cover inequalities can be shown to be an $\mathcal{NP}$-hard, single row 0–1 integer program.

(v) (Lot Sizing) Consider the single item uncapacitated lot-sizing set

$$\begin{aligned} X^{LS-U} = \quad & \Big\{(x,s,y) \in \mathbb{R}^n_+ \times \mathbb{R}^n_+ \times \{0,1\}^n : \\ & s_{t-1} + x_t = d_t + s_t \; 1 \leq t \leq n \\ & x_t \leq \left(\sum_{u=t}^{n} d_u\right) y_t \; 1 \leq t \leq n \Big\} \end{aligned}$$

Note that the feasible region X^{PP} of the production planning problem formulated in "A Multi-Item Production Planning Problem" section can be written as $X^{\mathrm{PP}} = \cap_{i=1}^{m} X_i^{LS-U} \cap Y$ where $Y \subseteq \{0,1\}^{mn}$ contains the joint machine constraints in Equation (7) and the possibly tighter bound constraints in Equation (6).

Select an interval $[k, k+1, \ldots, l]$ with $1 \leq k \leq l \leq n$ and some subset $T \subseteq \{k, \ldots, l\}$. Note that if $k \leq u \leq l$ and no production exists in any period in $\{k, \ldots, u\} \backslash T$(*i.e.*, $\sum_{j \in \{k,\ldots,u\} \backslash T} y_j = 0$), then the demand d_u in period u must either be part of the stock s_{k-1} or be produced in some period in $T \cap \{k, \ldots, u\}$. This establishes the validity of the inequality

$$s_{k-1} + \sum_{j \in T} x_j \geq \sum_{u=k}^{l} d_u \left(1 - \sum_{j \in \{k,\ldots,u\} \backslash T} y_j\right) \qquad (15)$$

Taking l as above, $L = \{1, \ldots, l\}$, and $S = \{1, \ldots, k-1\} \cup T$,, the above inequality can be rewritten as a so-called (*l,S*)-inequality:

$$\sum_{j \in S} x_j + \sum_{j \in L \backslash S} \left(\sum_{u=j}^{l} d_u\right) y_j \geq \sum_{u=1}^{l} d_u$$

This family of inequalities suffices to describe conv (X^{LS-U})

Now, given a point (x^*, s^*, y^*), the Separation Problem for the (*l,S*) inequalities is solved easily by checking if

$$\sum_{j=1}^{l} \min \left[x_j^*, \left(\sum_{u=j}^{l} d_u\right) y_j^*\right] < \sum_{u=1}^{l} d_u$$

for some l. If it does not, the point lies in conv(X^{LS-U}): otherwise a violated (*l, S*) inequality is found by taking $S = \{j \in \{1, \ldots, l\} : x_j^* < (\sum_{u=j}^{l} d_u) y_j^*\}$.

A Priori Modeling or Reformulation

Below we present four examples of modeling or a priori reformulations in which we add either a small number of constraints or new variables and constraints, called *extended formulations*, with the goal of obtaining tighter linear programming relaxations and, thus, much more effective solution of the corresponding MIPs.

(i) (Capacitated Facility Location—Adding a Redundant Constraint) The constraints in Equations (2) through (4) obviously imply validity of the constraint

$$\sum_{j=1}^{n} b_j y_j \geq \sum_{i=1}^{m} a_i$$

which states that the capacity of the open depots must be at least equal to the sum of all the demands of the clients. As $y \in \{0,1\}^n$, the resulting set is a 0–1 knapsack problem for which cutting planes are derived readily.

(ii) (Lot Sizing—Adding a Few Valid Inequalities) Consider again the single item, uncapacitated lot-sizing set X^{LS-U}. In item v) of the "Valid Inequalities and Cutting Planes" sections, we described the inequalities that give the convex hull. In practice, the most effective inequalities are those that cover a few periods. Thus, a simple a priori strengthening is given by adding the inequalities in Equation (15) with $T = \phi$ and $l \leq k + \kappa$

$$s_{k-1} \geq \sum_{u=k}^{l} d_u \left(1 - \sum_{j=k}^{u} y_j\right)$$

for some small value of κ.

(iii) (Lot Sizing—An Extended Formulation) Consider again the single item, uncapacitated lot-sizing set X^{LS-U}. Define the new variables z_{ut} with $u \leq t$ as the amount of demand for period t produced in period u. Now one obtains the extended formulation

$$\begin{aligned} & \sum_{u=1}^{t} z_{ut} = d_t, 1 \leq t \leq n \\ & z_{ut} \leq d_t y_u, 1 \leq u \leq t \leq n \\ & x_u = \sum_{t=u}^{n} z_{ut}, 1 \leq u \leq n \\ & s_{t-1} + x_t = d_t + s_t, 1 \leq t \leq n \\ & x, s \in \mathbb{R}^n_+, z \in \mathbb{R}^{n(n+1)/2}_+, y \in [0,1]^n \end{aligned}$$

whose (x,s,y) solutions are just the points of conv *(X^{LS-U})*. Thus, the linear program over this set solves the lot-sizing problem, whereas the original description of x^{LS-U} provides a much weaker formulation.

(iv) (Modeling Disjunctive or "Or" Constraints—An Extended Formulation) Numerous problems involve disjunctions,—for instance, given two jobs i, and j to be processed on a machine with processing times p_i, p_j, respectively, suppose that either job i must be completed before job j or vice versa. If t_i, t_j are variables representing the start times, we have the constraint EITHER "job i precedes job j" OR "job j

precedes job i," which can be written more formally as

$$t_i + p_i \leq t_j \, or \, t_j + p_j \leq t_i$$

More generally one often encounters the situation where one must select a point (a solution) from one of k sets or polyhedra (a polyhedron is a set described by a finite number of linear inequalities):

$$x \in \bigcup_{i=1}^{k} P_i \text{ where } P_i = \{x : A^i x \geq b^i\} \subseteq \mathbb{R}^n$$

When each of the sets P_i is nonempty and bounded, the set $\cup_{i=1}^{k} P_i$ can be formulated as the MIP:

$$x = \sum_{i=1}^{k} z^i \tag{16}$$

$$A^i z^i \geq b^i y^i \text{ for } i = 1, \ldots, k \tag{17}$$

$$\sum_{i=1}^{k} y^i = 1 \tag{18}$$

$$x \in \mathbb{R}^n, \quad z \in \mathbb{R}^{nk}, y \in \{0,1\}^k \tag{19}$$

where $y^i = 1$ indicates that the point lies in P_{i}. Given a solution with $y^i = 1$, the constraint in Equation (18) then forces $y^j = 0$ for $j \neq i$ and the constraint in Equation (17) then forces $z^i \in P_i$ and $z^j = 0$ for $j \neq i$. Finally, Equation (16) shows that $x \in P_i$ if and only if $y^i = 1$ as required, and it follows that the MIP models $\cup_{i=1}^{k} P_i$. What is more, it has been shown (6) that the linear programming relaxation of this set describes conv($\cup_{i=1}^{k} P_i$), so this again is an interesting extended formulation.

Extended formulations can be very effective in giving better bounds, and they have the important advantage that they can be added a priori to the MIP problem, which avoids the need to solve a separation problem whenever one wishes to generate cutting planes just involving the original (x, y) variables. The potential disadvantage is that the problem size can increase significantly and, thus, the time to solve the linear programming relaxations also may increase.

Heuristics

In practice, the MIP user often is interested in finding a good feasible solution quickly. In addition, pruning by optimality in branch-and-bound depends crucially on the value of the best known solution value z. We now describe several MIP heuristics that are procedures designed to find feasible, and hopefully, good, solutions quickly.

In general, finding a feasible solution to an MIP is an $\mathcal{NP}$-hard problem, so devising effective heuristics is far from simple. The heuristics we now describe are all based on the solution of one or more MIPs that hopefully are much simpler to solve than the original problem.

We distinguish between *construction heuristics*, in which one attempts to find a (good) feasible solution from scratch, and *improvement heuristics,* which start from a feasible solution and attempt to find a better one. We start with construction heuristics.

Rounding . The first idea that comes to mind is to take the solution of the linear program and to round the values of the integer variables to the nearest integer. Unfortunately, this solution is rarely feasible in $\mathrm{X_{MIP}}$.

A Diving Heuristic. This heuristic solves a series of linear programs. At the tth iteration, one solves

$$\min\{cx + hy : Ax + Gy \geq b, x \in \mathbb{R}_+^p, y \in \mathbb{R}_+^n, y_j = y_j^* \text{ for } j \in N^t\}$$

If this linear program is infeasible, then the heuristic has failed. Otherwise, let $(\overline{x}^t, \overline{y}^t)$ be the linear programming solution.

Now if $\overline{y}^t \in \mathbb{Z}_+^n$, then a diving heuristic solution has been found. Otherwise, if $\overline{y}^t \notin \mathbb{Z}_+^n$, then at least one other variable is fixed at an integer value. Choose $j \in N \backslash N^t$ with $y_j^t \notin \mathbb{Z}^1 \cdot$ Set $N^{t+1} = N^t \cup \{j\}$ and $t \leftarrow t + 1$. For example, one chooses to fix a variable whose LP value is close to an integer, i.e., $j = \operatorname{argmin}_{k: \overline{y}_k^t \notin \mathbb{Z}^1} [\min(\overline{y}_k^t - \lfloor \overline{y}_k^t \rfloor, \lceil \overline{y}_k^t \rceil) - \overline{y}_k^t]$.

A Relax-and-Fix Heuristic. This heuristic works by decomposing the variables into K blocks in natural way, such as by time period, geographical location, or other. Let $N = \{1, \ldots n\} = \cup_{k=1}^{K} I_k$ with intervals $I_k = [s_k, t_k]$ such that $s_1 = 1, t_K = n$, and $s_k = t_{k-1} + 1$ for $k = 2, \ldots, K$.

One solves K MIPs by progressively fixing the integer variables in the sets $I_1, I_2, \ldots, I_K$. Each of these MIPs is much easier because in the k-th problem only the variables in I_k are integer. The k-th MIP is

$$\begin{aligned}
&\min cx + hy \\
&Ax + Gy \geq b \\
&x \in \mathbb{R}_+^p, y_j = \overline{y}_j \text{ for } j \in \cup_{t=1}^{k-1} I_k, \\
&y_j \in \mathbb{Z}_+^1 \text{ for } j \in I_k, y_j \in \mathbb{R}_+^1 \text{ for } j \in \cup_{t=k+1}^{K} I_t
\end{aligned}$$

If $(\tilde{x}, \tilde{y})$ is an optimal solution, then one sets $\overline{y}_j = \tilde{y}_j$ for $j \in I_k$ and $k \leftarrow k + 1$. If the Kth MIP is feasible, then a heuristic solution is found; otherwise, the heuristic fails.

Now we describe three iterative improvement heuristics. For simplicity, we suppose that the integer variables are binary. In each case one solves an easier MIP by restricting the values taken by the integer variables to some neighborhood $N(y^*)$ of the best-known feasible solution (x^*, y^*) and one iterates. Also, let $(x^{\mathrm{LP}}, y^{\mathrm{LP}})$ be the current LP solution. In each case, we solve the MIP

$$\begin{aligned}
&\min cx + hy \\
&Ax + Gy \geq b \\
&x \in \mathbb{R}_+^p, y \in \mathbb{Z}_+^n, y \in N(y^*)
\end{aligned}$$

with a different choice of $N(y^*)$.

The Local Branching Heuristic. This heuristic restricts one to a solution at a (Hamming-) distance at most k from y^*:

$$N(y^*) = \{y \in \{0,1\}^n : |y_j - y_j^*| \leq k\}$$

This neighborhood can be represented by a linear constraint

$$N(y^*) = \{y \in \{0,1\}^n : \sum_{j:y_j^*=0} y_j + \sum_{j:y_j^*=1} (1 - y_j) \leq k\}$$

The Relaxation Induced Neighborhood Search Heuristic (RINS). This heuristic fixes all variables that have the same value in the IP and LP solutions and leaves the others free. Let $A = \{j \in N : y_j^{IP} = y_j^*\}$. Then

$$N(y^*) = \{y : y_j = y_j^* \text{ for } j \in A\}$$

The Exchange Heuristic. This heuristic allows the user to choose the set A of variables that are fixed. As for the Relax-and-Fix heuristic, if a natural ordering of the variables exists then, a possible choice is to fix all the variables except for those in one interval I_k. Now if $A = N \backslash I_k$, the neighborhood can again be taken as

$$N(y^*) = \{y : y_j = y_j^* \text{ for } j \in A\}$$

One possibility then is to iterate over $k = 1,\ldots, K$, and repeat as long as additional improvements are found.

The Branch-and-Cut Algorithm

The branch-and-cut algorithm is the same as the branch-and-bound algorithm except for one major difference. Previously one just selected a subset of solutions X^t from the list L that was described by the initial problem representation $Ax + Gy \geq b, x \in \mathbb{R}_+^p, y \in \mathbb{Z}_+^n$ and the bound constraints on the integer variables added in branching $l^t \leq y \leq u^t$. Now, one retrieves a set X^t from the list, along with a possibly tightened formulation (based on preprocessing and cutting planes)

$$P^t = \{(x,y) \in \mathbb{R}^p + \mathbb{R}_+^n : A^t x + G^t y \geq b^t, l^t \leq y \leq u^t\}$$

where $X^t = P^t \cap (\mathbb{R}^p \times \mathbb{Z}^n)$.

Now the steps, once X^t is taken from the list, L are

(i) Preprocess to tighten the formulation P^t.

(ii) Solve the linear program $z_{LP}^t = \min\{cx + hy : (x,y) \in P^t\}$.

(iii) Prune the node, if possible, as in branch-and-bound.

(iv) Call one or more heuristics. If a better feasible solution is obtained, Then update the incumbent value $\bar{z}$.

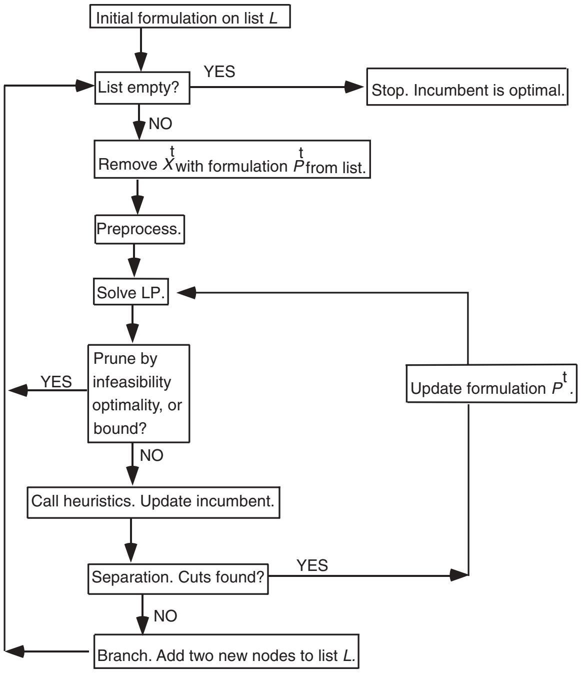

Initial formulation on list L
List empty?
YES
Stop. Incumbent is optimal.
NO
Remove X^t with formulation P^t from list.
Preprocess.
Solve LP.
Prune by infeasibility optimality, or bound?
YES
Update formulation P^t.
NO
Call heuristics. Update incumbent.
Separation. Cuts found?
YES
NO
Branch. Add two new nodes to list L.

Figure 2. Branch-and-cut schema.

(v) Look for violated valid inequalities. If one or more satisfactory cuts are found, then add them as cuts, modify P^t, and repeat ii).

(vi) If no more interesting violated inequalities are found, Then branch as in the branch-and-bound algorithm and add the two new sets $X_{\leq}^t$ and $X_{>}^t$ to the list L, along with their latest formulations $P^{\bar{t}}$.

Then one returns to the branch-and-bound step of selecting a new set from the list and so forth.

In practice, preprocessing and cut generation always are carried out on the original set X_{MIP} and then on selected sets drawn from the list (for example, sets obtained after a certain number of branches or every k-th set drawn from the list). Often, the valid inequalities added for set X^t are valid for the original set X_{MIP}, in which case the inequalities can be added to each set P^t. All the major branch-and-cut systems for MIP use preprocessing, and heuristics, such as diving and RINS, and the valid inequalities generated include MIR inequalities, Gomory mixed integer cuts, 0–1 cover inequalities, and path inequalities, generalizing the (l, S) inequalities. A flowchart of a branch-and-cut algorithm is shown in Fig. 2.

REFERENCES AND ADDITIONAL TOPICS

Formulations of Problems as Mixed Integer Programs

Many examples of MIP models from numerous areas, including air and ground transport, telecommunications, cutting and loading, and finance can be found in Heipcke (2) and Williams (3), as well as in the operations research journals such as *Operations Research, Management*

Science, Mathematical Programming, Informs Journal of Computing, *European Journal of Operational Research*, and more specialized journals such as *Transportation Science*, *Networks*, *Journal of Chemical Engineering*, and so forth.

Basic References

Two basic texts on integer and mixed integer programming are Wolsey (4) and part I of Pochet and Wolsey (5). More advanced texts are Schrijver (6) and Nemhauser and Wolsey (7). Recent surveys on integer and mixed integer programming with an emphasis on cutting planes include Marchand et al. (8), Fugenschuh and Martin (9), Cornuejols (10), and Wolsey (11).

Preprocessing is discussed in Savelsbergh (12) and Andersen and Andersen (13), and branching rules are discussed in Achterberg et al. (14). Much fundamental work on cutting planes is due to Gomory (15,16). The related mixed integer rounding inequality appears in chaper II.1 of Nemhauser and Wolsey (7), and cover inequalities for 0–1 knapsack constraints are discussed in Balas (17), Hammer et al. (18), and Wolsey (19). The local branching heuristic appears in Fischetti and Lodi (29): RINS and diving appears in Danna et al. (21).

Decomposition Algorithms

Significant classes of MIP problems cannot be solved directly by the branch-and-cut approach outlined above. At least three important algorithmic approaches use the problem structure to decompose a problem into a sequence of smaller/easier problems. One such class, known as branch-and-price or column generation, see, for instance, Barnhart et al. (22), extends the well-known Dantzig–Wolfe algorithm for linear programming (23) to IPs and MIPs. Essentially, the problem is reformulated with a huge number of columns/variables, then dual variables or prices from linear programming are used to select/generate interesting columns until optimality is reached, and then the whole is embedded into a branch-and-bound approach. Very many problems in the area of airlines, road and rail transport, and staff scheduling are treated in this way. A related approach, Lagrangian relaxation (24), uses the prices to transfer complicating constraints into the objective function. The resulting, easier problem provides a lower bound on the optimal value, and the prices then are optimized to generate as good a lower bound as possible.

An alternative decomposition strategy, known as Benders' decomposition (25), takes a different approach. If the value of the integer variables is fixed, then the remaining problem is a linear program $\phi(y) = \min\{cx : Ax \geq b - Gy, x \in \mathbb{R}^p_+\}$ and the original problem can be rewritten as $\min\{\phi(y) + hy : y \in \mathbb{Z}^n_+\}$. Although $\phi(y)$ is not known explicitly, whenever a linear program is solved for some y^*, a support of the function $\phi(y)$ is obtained and the algorithm works by simultaneously enumerating over the y variables and continually updating the approximation to $\phi(y)$ until an optimal solution is obtained.

MIP Test Problems and Software

An important source for test instances is the MIPLIB library (26). Several commercial branch-and-cut systems are available, of which three of the most well known are Cplex (27), Xpress-MP (28), and Lindo (29). See OR-MS Today for regular surveys of such systems. Among non commercial systems, several MIP codes exist in the Coin library (30), as well as several other research codes, including SCIP (31) and MINTO (32). In addition, modeling languages such as AMPL (33), LINGO (29) and MOSEL (28) that facilitate the modeling and generation of linear and mixed integer programs.

Nonlinear Mixed Integer Programming

The study of algorithms for nonlinear MIPs is, relatively, in its infancy. Portfolio optimization problems with integer variables are being tackled using convex (second order cone) optimization as relaxations: see Ben Tal and Nemirovsky (34). Two approaches for nonconvex MINLPs are generalized Benders' decomposition, see Geoffrion (35), and outer approximation algorithms (36, 37). References include the book of Floudas (38) and the lecture notes of Weismantel (39). Software includes the Dicopt code (40) and the BARON code of Sahinidis and Tawarmalami (41): see also Ref. 42 for recent computational results. SeDuMi (43) is one of the most widely used codes for convex optimization. The Cplex and Xpress-MP systems cited above allow for nonlinear MIPs with quadratic convex objective functions and linear constraints. Heuristics for nonlinear MIPs are presented in Ref. 44, and a test set of nonlinear MIPs is in preparation (45).

BIBLIOGRAPHY

1. E. Balas, Disjunctive programming: Properties of the convex hull of feasible points, Invited paper with foreword by G. Cornuéjols and W. R. Pulleyblank, *Discrete Applied Mathematics*, **89**: 1–44, 1998.
2. S. Heipcke, *Applications of Optimization with Xpress*. Dash Optimization Ltd, 2002.
3. H. P. Williams, *Model Building in Mathematical Programming*. John Wiley and Sons, 1999.
4. L. A. Wolsey, *Integer Programming*. John Wiley and Sons, 1998.
5. Y. Pochet and L. A. Wolsey, *Production Planning by Mixed Integer Programming*. Springer, 2006.
6. A. Schrijver, *Theory of Linear and Integer Programming.*, John Wiley and Sons, 1986.
7. G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988.
8. H. Marchand, A. Martin, R. Weismantel, and L. A. Wolsey, Cutting planes in integer and mixed integer programming, *Discrete Applied Mathematics*, **123/124**: 397–446, 2002.
9. A. Fugenschuh and A. Martin, Computational integer programming and cutting planes, in K. Aardal, G. L. Nemhauser, and R. Weismantel, (eds.) *Combinatorial Optimization, Vol. 12 of Handbooks in Operations Research and Management Science*, chapter 2, pages 69-121. Elsevier, 2006.

10. G. Cornuéjols. Valid inequalities for mixed integer linear programs, *Mathematical Programming B*, **112**: 3–44, 2007.
11. L. A. Wolsey, Strong formulations for mixed integer programs: Valid inequalities and extended formulations, *Mathematical Programming B*, **97**: 423–447, 2003.
12. M. W. P. Savelsbergh, Preprocessing and probing for mixed integer programming problems, *ORSA J. of Computing*, **6**: 445–454, 1994.
13. E. D. Andersen and K. D. Andersen, Presolving in linear programming, *Mathematical Programming*, **71**: 221–245, 1995.
14. T. Achterberg, T. Koch, and A. Martin, Branching rules revisited, *Operations Research Letters*, **33**: 42–54, 2005.
15. R. E. Gomory, Solving linear programs in integers, in R. E. Belmman and M. Hall, Jr.(eds.), *Combinatorial Analysis*. American Mathematical Society, 211–216, 1960.
16. R. E. Gomory, An algorithm for the mixed integer problem, RAND report RM-2597, 1960.
17. E. Balas, Facets of the knapsack polytope, *Mathematical Programming*, **8**: 146–164, 1975.
18. P. L. Hammer, E. L. Johnson, and U. N. Peled, Facets of regular 0–1 polytopes, *Mathematical Programming*, **8**: 179–206, 1975.
19. L. A. Wolsey, Faces for linear inequalities in 0–1 variables, *Mathematical Programming***8**: 165–178, 1975.
20. M. Fischetti and A. Lodi, Local branching, *Mathematical Programming*, **98**: 23–48, 2003.
21. E. Danna, E. Rothberg, and C. Le Pape, Exploring relaxation induced neighborhoods to improve MIP solutions, *Mathematical Programming*, **102**: 71–90, 2005.
22. C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P Savelsbergh, and P. H. Vance, Branch-and-price: Column generation for huge integer programs, *Operations Research*, **46**: 316–329, 1998.
23. G. B. Dantzig and P. Wolfe, Decomposition principle for linear programs, *Operations Research*, **8**: 101–111, 1960.
24. A. M. Geoffrion, Lagrangean relaxation for integer programming, *Mathematical Programming Study*, **2**: 82–114, 1974.
25. J. F. Benders, Partitioning procedures for solving mixed variables programming problems, *Numerische Mathematik*, **4**: 238–252, 1962.
26. T. Achterberg, T. Koch, and A. Martin, MIPLIB 2003, *Operations Research Letters*, **34**: 1–12, 2006. Available: http://miplib: zib.de.
27. ILOG CPLEX, Using the Cplex callable library. Available: http://www.ilog.com/cplex.
28. Xpress-MP, Xpress-MP optimisation subroutine library. Available: http://www.dashoptimization.com.
29. LINDO, Optimization modeling with Lindo, Available: http://www.lindo.com.
30. COIN-OR, Computational infrastructure for operations research. Available: http://www.coin-or.org/.
31. T. Achterberg, SCIP—a framework to integrate constraint and mixed integer programs, ZIB Report 04-19. Konrad-Zuse Zentrum, Berlin 2004. Available:http://scip.zib.de.
32. MINTO, Mixed integer optimizer, Developed and maintained by M. W. P Savelsbergh, Georgia Institute of Technology. Available: http://www2.isye.gatech.edu/ mwps/software/.
33. R. Fourer, D. M. Gay and B. W. Kernighan, AMPL: A modeling language for mathematical programming Duxbury Press/ Brooks Cole Publishing Co. 2002. Available: http://www.ampl.com/.
34. A. Ben-Tal and A. Nemirovski, Lectures on Modern Convex Optimization: Analysis, Algorithms and Engineering Applications, MPS-SIAM Series on Optimization, Philadelphia, 2001.
35. A. M. Geoffrion, Generalized Benders' decomposition, *Jo. Optimization Theory and Applications*, **10**: 237–260, 1972.
36. R. Fletcher and S. Leyffer, Solving mixed integer nonlinear programs by outer approximation, *Mathematical Programming*, **66**: 327–349, 1994.
37. M. A. Duran and I. E Grossman, An outer approximation algorithm for a class of mixed-integer nonlinear programs, *Mathematical Programming*, **36**: 307–339, 1986.
38. C. A. Floudas, *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, 1995.
39. R. Weismantel, *Mixed Integer Nonlinear Programming*. CORE Lecture Series. CORE, Université catholique de Louvain, Belgium, 2006.
40. Dicopt. Framework for solving MINLP (mixed integer nonlinear programming) models. Available: http://www.gams.-com.
41. BARON,Branch and reduce optimization navigator. Available: http://neos.mcs.anl.gov/neos/solvers/go:BARON/GAMS.html.
42. M. Tawarmalami and N. V. Sahinidis, Global optimization of mixed-integer nonlinear programs: A theoretical and computational study, *Mathematical Programming*, **99**: 563–591, 2004.
43. SeDuMi, Software for optimization over symmetric cones. Available: http://sedumi.mcmaster.ca/.
44. P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuejols, I. E. Grossman, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter, An algorithmic framework for convex mixed integer nonlinear programs, *Technical report* RC23771, IBM T. J. Watson Research Center, *Discrete Optimization. In press.*
45. N. W. Sawaya, C. D. Laird, and P. Bonami, A novel library of nonlinear mixed-integer and generalized disjuctive programming problems. In press, 2006.

Laurence A. Wolsey
Université Catholique de Louvain
Louvain–la–Neuve, Belgium

MULTIGRID METHODS

INTRODUCTION

Numerical modeling in science and engineering has emerged in recent years as a viable alternative to a more conventional experimental approach, which has some shortfalls, such as the cost, the time consumption, the difficulties with accuracy, or the ethical issues. As computer processing power continues to increase, nowadays it is possible to perform modeling and simulation studies for large-scale problems in important areas, such as continuum mechanics, electromagnetism, quantum physics, and so forth. Modern trends also involve modeling of the complex systems with the constitutive parts from different areas, which are often referred to as *multi-physics systems*. The growing appetite for even larger models requires also a development of sophisticated algorithms and numerical techniques for efficient solution of underlying equations.

Computer-aided modeling represents the space and time continuum by a finite set of properly selected discrete coordinate points. These points typically are connected to form a mesh over the domain of interest. A discrete physical or numerical variable is associated with the mesh points. Such a discrete variable is referred to as the *grid variable*. A set of grid variables, together with the algebraic equations that define their implicit dependencies, represent a *grid problem*. A process of approximating a continuous problem by an appropriate grid problem is called the *discretization*. The most common class of continuous problems that require discretization are differential equations (DEs).

DEs are the mathematical expressions that relate unknown functions and their derivatives in continuous space and time. The local connectivity among the mesh points is used to approximate the derivatives of the unknown function. The order of this approximation determines the order of accuracy of the method itself. The size of the resulting grid problem is proportional to the number of mesh points. Some well-known methods for the discretization of DEs are the *finite difference method* (FDM), the *finite element method* (FEM), and the *finite volume method* (1–3). A common feature of the grid problems obtained by discretization of DEs by these methods is the *local dependence of grid variables*. Namely, a single grid variable depends only on a small set of grid variables in its close neighborhood.

The solution of the grid problems created by the discretization of DEs, which usually take the form of linear or non linear systems of algebraic equations, is obtained by applying a certain solution procedure. Iterative methods start from an initial approximation to the solution, which is improved over a number of iterations until the discrete solution is obtained within the prescribed accuracy. The difference between the initial discrete approximation and the discrete solution represents the *iteration error* that is eliminated during an iterative solution procedure. An optimal iterative solver is the one that scales optimally with the problem size. That is, the computing resources employed by the solver and the execution time should be proportional to the problem size. To achieve an optimal iterative solution procedure, we must ensure that it converges to a prescribed accuracy within a constant, presumably small, number of iterations, regardless of the problem size or any other problem-specific parameters.

Simple iterative methods often fail in fulfilling the optimality condition when applied to discretized DEs. To understand the problem, we shall consider the solution error in the Fourier space, represented as a linear combination of the wave-like components having the shape of sine or cosine functions with different wavelengths (or frequencies). Simple iterative methods are very efficient in eliminating the high-frequency (short wavelength) error components because these require only the information from the closest grid neighbors. This behavior is known as the *smoothing property* of the iterative methods (4, p. 412–419). After this initial phase, when rapid convergence is observed within a few iterations, the simple iterative solvers have to work hard to reduce the remaining error that is now dominated by the low-frequency (long wavelength) error components. The reduction of low-frequency errors requires communication among distant grid variables and takes a much larger number of iterations than in the case of the high-frequency error components. This reason is why the simple iterative procedures become nonoptimal.

The splitting into high-frequency and low-frequency error components is introduced, in principle, relative to the characteristic distance between neighboring mesh points or the *mesh size*. Namely, the wavelengths of the high-frequency solution error components are comparable with the mesh size. Obviously, a part of the low-frequency error components can be regarded as the high-frequency components if the problem is discretized using a coarser mesh. This situation naturally leads to the idea of using a coarser grid problem to improve the convergence and reduce the numerical cost of an iterative solution scheme. But we need to keep in mind that only the fine grid problem approximates the continuous DE with the required accuracy. Therefore, both problems should be combined in a proper way to produce an effective solution algorithm. Moreover, some low-frequency error components still can represent a problem for iterative procedures on the coarser mesh. These components can be reduced by introducing a sequence of additional progressively coarser meshes and corresponding grid problems associated with them. This idea leads to multigrid methods (MG) that employ a hierarchy of discrete grid problems to achieve an optimal solution procedure.

In this section, we present only a high-level description of the MG heuristics. For additional technical details, the reader is referred to the next two sections. After a few steps of a simple iterative procedure at the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

finest grid, the high-frequency error components are eliminated from the initial solution error. This procedure is called the *smoothing*. The remaining low-frequency error then is transferred to the coarser grid by a process called the *restriction*. The same procedure (smoothing and restriction) is repeated on the coarser level. The remaining low-frequency error components at the coarser level are transfered further to a coarser grid, and the smoothing procedure is repeated. This pair of operations (the smoothing and the error transfer to a coarser grid) is repeated until a sufficiently coarse grid, with only a few nodes, is reached. The coarsest grid solution (with the low-frequency errors removed by the application of a direct-solution method) then is used to correct the corresponding discrete solutions on the finer grid levels using *prolongation* or *interpolation*. The prolongation steps often are followed by additional postsmoothing steps to eliminate the remaining high-frequency error components that could be introduced.

The first MG scheme was introduced by Fedorenko in the early 1960s (5). It was presented in the context of the Poisson equation on the unit square domain. However, the full potential of the MG approach was not realized until the work of Brandt in the mid 1970s (6). Since then, a tremendous amount of theory and application related to MG methods has been published, including several monographs (7–15). Over time, MG methods have evolved into an independent field of research, interacting with numerous engineering application areas and having major impact in almost all scientific and engineering disciplines.

A typical application for MG is in the numerical solution of discrete elliptic, self-adjoint, partial differential equations (PDEs), where it can be applied in combination with any of the common discretization techniques. In such cases, MG is among the fastest-known solution techniques. MG also is directly applicable to more complicated, nonsymmetric, and non linear DE problems, systems of DEs, evolution problems, and integral equations. In recent years we have seen an increased development of multi level (ML) solvers for the solution of DE problems in various areas, including aerodynamics (16), atmospheric and oceanic modeling (17), structural mechanics (18), quantum mechanics, statistical physics, (19), semiconductor fabrication (20), and electromagnetism (21–23). In all these applications, MG methods can be used as the building blocks of an overall solver, with an aim of reaching the convergence rate that is nearly independent of the number of unknown grid variables or other problem parameters. Such solvers would be capable of reducing the solution error to the order of the computer truncation error with (nearly) optimal computational cost.

In contrast to other numerical methods, MG represents a computational principle rather than a particular computational technique, and, therefore, it is not restricted by the type of problem, domain, and mesh geometry, or discretization procedure. MG methods even may be applied successfully to algebraic problems that do not involve geometric information and do not even originate from the discretization of DEs. To this end, special techniques are developed to create a required hierarchy that uses only algebraic information available for the "grid" variable dependencies. In addition, a broad range of problems in science and engineering require multi scale modeling and simulation techniques (e.g., oil reservoir modeling). The range of scales involved in such problems induce a prohibitively large number of variables in the classic mono scale modeling approach. MG methods also naturally apply to such cases.

BASIC CONCEPTS

The basic MG ideas are developed more in this section within the context of second-order, elliptic, self-adjoint DE problems. We first introduce the model problem in one spatial dimension. The FDM/FEM discretization of this problem produces a linear system that will serve as an example for studying the efficiency of the basic iterative schemes (fixed-point iterations). Lack of efficiency of simple iterations in this context is the main motivation behind the application of these schemes in a recursive fashion. This naturally leads to MG methods. We describe the main algorithmic components of MG, its arithmetic complexity, and its effectiveness.

Continuous and Discrete Single-Grid Problems

A general, time-dependent, DE in d spatial dimensions can be written as

$$A\left(t,u,\frac{\partial u}{\partial t},\frac{\partial^2 u}{\partial t^2},\ldots,\frac{\partial^k u}{\partial x_1^{k_1}\cdots\partial x_d^{k_d}},\cdots\right)=0 \quad \text{on} \quad \Omega\times T \quad (1)$$

where $\bar{x}=(x_1,\ldots,x_d)\in\Omega\subset\mathbb{R}^d$ and $T=[0,\tau]$. To ensure well-posedness of the solution, an appropriate set of boundary conditions (BCs) and initial conditions (ICs) need to be imposed. By covering the domain of interest Ω by a mesh Ω_h and by applying a suitable spatial discretization procedure, such as the FDM or the FEM, the continuous differential problem becomes a system of differential–algebraic equations:

$$A_h\left(t,u_h,\frac{\partial u_h}{\partial t},\frac{\partial^2 u_h}{\partial t^2},\ldots,\frac{\partial^k u_h}{\partial x_1^{k_1}\cdots\partial x_d^{k_d}},\cdots\right)=0 \quad \text{on} \quad \Omega_h\times T \quad (2)$$

where u_h represents a discrete solution variable of dimension n_h associated with the mesh Ω_h. The BCs are included in the formulation in Equation (2). If the discrete problem is stationary, Equation (2) reduces to a non linear algebraic system

$$A_h(u_h)=0 \qquad \text{on} \qquad \Omega_h \quad (3)$$

Finally, in the case of linear stationary problems, the discrete solution u_h is defined as a linear algebraic system

$$A_h u_h = f_h \qquad \text{on} \qquad \Omega_h \quad (4)$$

where A_h is a coefficient matrix and f_h is the right-hand side vector.

A simple practical example that will serve to introduce the basic MG concepts is the one-dimensional model–problem

$$A(u) = -\frac{d^2u}{dx^2} = f \qquad \text{in} \qquad \Omega = (0,1) \tag{5}$$

subject to the homogeneous Dirichlet BCs

$$u(0) = u(1) = 0 \tag{6}$$

Choosing a mesh Ω_h to be a set of $n_h + 1$ uniformly spaced points $x_i = ih,\ i = 0, \ldots, n_h$ where $h = 1/n_h$, and replacing the second derivative at the mesh points by a central finite-difference approximation (3), we obtain a discrete problem

$$\begin{aligned} A_h u_h = \frac{-u_h(x_{i+1}) + 2u_h(x_i) - u_h(x_{i-1})}{h^2} &= f_h(x_i), \\ i = 1, \ldots, n_h - 1 \end{aligned} \tag{7}$$

with $u_h(x_0) = u_h(x_{n_h+1}) = 0$. The coefficient matrix of the linear system in Equation (7) is a symmetric tridiagonal matrix $A_h = \frac{1}{h^2}\ \text{tridiag}[-1 \quad 2 \ -1]$. Although the linear system in Equation (7), because of its simplicity, can be solved efficiently by a direct method, it is used as an example to explain the main algorithmic features of MG.

The extension of the model–problem in Equations (5) and (6) and MG concepts to two or more spatial dimensions is straightforward. For this purpose, we introduce a natural extension of the model–problem in Equations (5) and (6) to two spatial dimensions, known as the Poisson equation (2, Ch. 1):

$$A(u) = -\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}\right) = f \ \text{ in } \ \Omega = (0,1) \times (0,1) \tag{8}$$

with the homogeneous Dirichlet BCs

$$u = 0 \text{ on } \partial\Omega \tag{9}$$

In Equation (8) we adopted, for simplicity, the unit square domain $\Omega = (0,1) \times (0,1) \subset R^2$. Central finite-difference approximation of the second derivatives in Equation (8), defined on a grid of uniformly spaced points $((x_1)_i, (x_2)_j) = (ih, jh), i, j = 1, \ldots, n_h - 1$, results in a linear system in Equation (4), in which the coefficient matrix A_h can be represented in stencil notation as

$$A_h = \frac{1}{h^2}\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} \tag{10}$$

We remark that the discretization in Equation (10) of the two-dimensional model problem on a uniform Cartesian grid is obtained as a tensor product of two one-dimensional discretizations in x_1 and x_2 coordinate directions.

The Smoothing Property of Standard Iterative Methods

When an approximate solution $\tilde{u}_h$ of the system in Equation (4) is computed, it is important to know how close it is to the true discrete solution u_h. Two quantitative measures commonly are used in this context. The *error* is defined as $e_h = u_h - \tilde{u}_h$ and the *residual* as $r_h = f_h - A_h\tilde{u}_h$. Note that if the residual is small, this does not imply automatically that the approximation $\tilde{u}_h$ is close to the solution u_h. The error e_h and the residual r_h are connected by the *residual equation*

$$A_h e_h = r_h \tag{11}$$

The importance of Equation (11) can be seen from the fact that if the approximation $\tilde{u}_h$ to the solution u_h of Equation (4) is computed by some iterative scheme, it can be improved as $u_h = \tilde{u}_h + e_h$.

The simplest iterative schemes that can be deployed for the solution of sparse linear systems belong to the class of *splitting iterations* (see Refs 2,4,7,8, and 10 for more details). In algorithmic terms, each splitting iteration starts from the decomposition of the coefficient matrix A_h as $A_h = M_h - N_h$, with M_h being a regular matrix (det $(M_h) \neq 0$), such that the linear systems of the form $M_h u_h = f_h$ are easy to solve. Then, for a suitably chosen initial approximation of the solution $\tilde{u}^{(0)}$ an iteration is formed:

$$\tilde{u}_h^{(k+1)} = M_h^{-1} N_h \tilde{u}^{(k)} + M_h^{-1} f_h, \quad k = 0, 1, \ldots \tag{12}$$

Note that Equation (12) also can be rewritten to include the residual vector $r_h^{(k)} = f_h - A\tilde{u}_h^{(k)}$:

$$\tilde{u}_h^{(k+1)} = \tilde{u}_h^{(k)} + M_h^{-1} r_h^{(k)}, \qquad k = 0, 1, \ldots \tag{13}$$

Some well-known methods that belong to this category are the *fixed-point iterations*, or the *relaxation* methods, such as the Jacobi method, the Gauss-Seidel method, and its generalizations SOR and SSOR (4). To introduce these methods, we start from the splitting of the coefficient matrix A_h in Equation (4) in the form $A_h = D_h - L_h - U_h$, where $D_h = \text{diag}(A_h)$ and $-L_h$ and $-U_h$ are strictly the lower and the upper triangular part of A_h, respectively. In this way, the system in Equation (4) becomes $(D_h - L_h - U_h)u_h = f_h$ and we can form a variety of iterative methods of the form in Equation (12) by taking suitable choices for the matrices M_h and N_h.

In the Jacobi method, the simplest choice for M_h is taken, that is $M_h = D_h$, $N_h = L_h + U_h$ and the iteration can be written as:

$$\tilde{u}_h^{(k+1)} = D_h^{-1}(L_h + U_h)\tilde{u}_h^{(k)} + D_h^{-1} f_h, \quad k = 0, 1, \ldots \tag{14}$$

A slight modification of the original method in Equation (14) is to take the weighted average between $\tilde{u}_h^{(k+1)}$ and $\tilde{u}_h^{(k)}$ to form a new iteration,

$$\begin{aligned} \tilde{u}_h^{(k+1)} &= [(1-\omega)I + \omega D_h^{-1}(L_h + U_h)]\tilde{u}_h^{(k)} + \omega D_h^{-1} f_h, \\ k &= 0, 1, \ldots \end{aligned} \tag{15}$$

where $\omega \in \mathbb{R}$. Equation (15) represents the *weighted* or *damped* Jacobi method. For $\omega = 1$, Equation (15) reduces to the standard Jacobi method in Equation (14).

The Gauss–Seidel method involves the choice $M_h = D_h - L_h, N_h = U_h$ and can be written as

$$\tilde{u}_h^{(k+1)} = (D_h - L_h)^{-1} U_h \tilde{u}_h^{(k)} + (D_h - L_h)^{-1} f_h, \quad k = 0, 1, \ldots \tag{16}$$

The main advantage of the Gauss–Seidel method is that the components of the new approximation $\tilde{u}_h^{(k+1)}$ can be used as soon as they are computed. Several modifications of the standard Gauss–Seidel method were developed with the aim of improving the convergence characteristics or parallelizability of the original algorithm in Equation (16) (symmetric, red-black, etc. see Refs. 4,7, and 8).

When applied to the solution of linear systems in Equation (7) that arise from the discretization of the model–problem in Equations (5) and (6), the convergence of splitting iterations is initially rapid, only to slow down significantly after a few iterations. More careful examination of the convergence characteristics using Fourier analysis (7–10) reveal different speeds of convergence for different Fourier modes. That is, if different Fourier modes (vectors of the form $(v^l)_i = \sin(il\pi/n_h), i = 1, \ldots, n_h - 1$, where l is the wave number) are used as the exact solutions of the residual Equation (11) with a zero initial guess, the convergence speed of the splitting iterations improves with the increasing wave number l. This means that the convergence is faster when l is larger, that is, when the error in the solution u_h contains the high-frequency (highly oscillatory) components. Thus, when a system in Equation (7) with an arbitrary right-hand side is solved using a simple splitting iteration with an arbitrary initial guess $\tilde{u}_h^{(0)}$, the initial fast convergence is because of the rapid elimination of the high-frequency components in the error e_h. A slow decrease in the error at the later stages of iteration indicates the presence of the low-frequency components. We assume that the Fourier modes in the lower half of the discrete spectrum (with the wave numbers $1 \leq l < n_h/2$) are referred to as the low-frequency (smooth) modes, whereas the modes in the upper half of the discrete spectrum (with the wave numbers $n_h/2 \leq l \leq n_h - 1$) are referred to as the high-frequency (oscillatory) modes.

If we rewrite a general splitting iteration in Equation (12) as $\tilde{u}_h^{(k+1)} = G_h \tilde{u}_h^{(k)} + g_h$, then the matrix G_h is referred to as the iteration matrix. The error $e_h^{(k)} = u_h - \tilde{u}_h^{(k)}$ after k iterations satisfies the relation $e_h^{(k)} = G_h^k e_h^{(0)}$. A sufficient and necessary condition for a splitting iteration to converge to the solution $(\{\tilde{u}_h^{(k)}\} \to u_h)$ is that the spectral radius of the iteration matrix G_h is less than 1 (24). The spectral radius is defined as $\rho(G_h) = \max|\lambda_j(G_h)|$, where $\lambda_j(G_h)$ are the eigenvalues of G_h [recall that for a symmetric and positive definite (SPD) matrix, all the eigenvalues are real and positive (24)]. The speed of convergence of a splitting iteration is determined by the asymptotic convergence rate (4)

$$\tau = -\ln\left(\lim_{k \to \infty} \frac{\|e_h^{(k)}\|}{\|e_h^{(0)}\|}\right)^{1/k} \tag{17}$$

For the case of the linear system in Equation (7) obtained from the discretization of the model–problem in Equations (5) and (6), the eigenvalues of the iteration matrix $G_h^J = I - \frac{\omega h^2}{2} A_h$ for the damped Jacobi method are given by $\lambda_j(G_h^J) = 1 - 2\omega \sin^2(\frac{j\pi}{2n_h}),\ j = 1, \ldots, n_h - 1$. Thus, $|\lambda_j\ (G_h^J)| < 1$ for each j if $0 < \omega < 1$, and the method is convergent. However, different choices of the damping parameter ω have a crucial effect on the amount by which different Fourier components of the error are reduced. One particular choice is to find the value of the parameter ω that maximizes the effectiveness of the damped Jacobi method in reducing the oscillatory components of the error (the components with the wave numbers $n_h/2 \leq l \leq n_h - 1$). The optimal value for the linear system in Equation (7) is $\omega = 2/3(7)$, and for this value we have $|\lambda_j| < 1/3$ for $n_h/2 \leq j \leq n_h - 1$. This means that each iteration of the damped Jacobi method reduces the magnitude of the oscillatory components of the error by at least a factor 3. For the linear system obtained from the FEM discretization of the two-dimensional model–problem in Equation (8), the optimal value of ω is 4/5 for the case of linear approximation and $\omega = 8/9$ for bilinear case, (2, p. 100, 4). For the Gauss–Seidel method, the eigenvalues of the iteration matrix for the model-problem in Equation (7) are given by $\lambda_j(G_h^{GS}) = \cos^2(\frac{j\pi}{n_h}),\ j = 1, \ldots, n_h - 1$. As in the case of the damped Jacobi method, the oscillatory modes in the error are reduced rapidly, whereas the smooth modes persist.

The property of the fixed-point iteration schemes to reduce rapidly the high-frequency components in the error is known as the *smoothing* property, and such schemes commonly are known as the *smoothers*. This property is, at the same time, the main factor that impairs the applicability of these methods as stand-alone solvers for linear systems that arise in FDM/FEM discretizations of PDEs.

Two-Grid Correction Method

Having introduced the smoothing property of standard iterative methods, we investigate possible modifications of such iterative procedures that would enable the efficient reduction of all frequency components of the solution error e_h. Again, we study the model–problem in Equations (5) and (6) discretized on the uniform grid $\Omega_h = \{x_i = ih\},\ h = 1/n_h, i = 1, \ldots, n_h - 1$ yielding a linear system in Equation (7). The mesh Ω_h may be regarded as a "fine" mesh obtained by the uniform refinement of a coarse mesh $\Omega_H = \{x_i = iH\},\ H = 2h, i = 1, \ldots, n_H - 1$. The coarse mesh contains only the points of the fine mesh with the even numbers. After applying several steps of a fixed-point method to the h-system in Equation (7), only the smooth components of the error remain. The questions that naturally arise in this setting concern the properties of the smooth error components from the grid Ω_h, when represented on the coarse grid Ω_H. These components seem to be more oscillatory on Ω_H than on Ω_h (see Ref. 7). Notice that on the coarse grid Ω_H, we have only half as many Fourier modes compared with the fine grid $\Omega_h (n_H = \frac{1}{2} n_h)$. The fact that the smooth error components from the fine grid seem less smooth on the coarse grid offers a potential remedy for the situation when a fixed-point iteration loses its effectiveness—we need to move to a coarse grid Ω_H, where a fixed-point iteration will be more effective in reducing the remaining error components. This idea forms the basis of the two-grid method, which is summarized in the Algorithm 1.

Algorithm 1. Two-grid correction scheme.

```
1:  $\tilde{u}_h \leftarrow G_h^{v_1}(\tilde{u}_h^{(0)}, f_h)$
2:  $r_h = f_h - A_h\tilde{u}_h$
3:  $r_H = R_h^H r_h$
4:  Solve approximately $A_H e_H = r_H$
5:  $e_h = P_H^h e_H$
6:  $\tilde{u}_h = \tilde{u}_h + e_h$
7:  $\tilde{u}_h \leftarrow G_h^{v_2}(\tilde{u}_h, f_h)$
```

In Algorithm 1, $G_h^v(\tilde{u}_h, f_h)$ denotes the application of v iterations of a fixed-point iteration method to a linear system in Equation (7) with the initial guess $\tilde{u}_h$. At Step 4 of Algorithm 1, the coarse-grid problem $A_H e_H = r_H$ needs to be solved. The coarse-grid discrete operator A_H can be obtained either by the direct discretization of the continuous problem on a coarse mesh Ω_H or from the fine-grid discrete operator A_h by applying the Galerkin projection $A_h = R_h^H A_h P_H^h$. After solving the coarse grid problem (Step 4), we need to add the correction e_H, defined on the coarse grid Ω_H, to the current approximation of the solution $\tilde{u}_h$, which is defined on the fine grid Ω_h. It is obvious that these two vectors do not match dimensionally. Thus, before the correction, we need to transform the vector e_H to the vector e_h. The numerical procedure that implements the transfer of information from a coarse to a fine grid is referred to as *interpolation* or *prolongation*. The interpolation can be presented in operator form as $v_h = P_H^h v_H$, where $v_H \in \mathbb{R}^{n_H}, v_h \in \mathbb{R}^{n_h}$, and $P_H^h \in \mathbb{R}^{n_h \times n_H}$. Here, n_h denotes the size of a discrete problem on the fine grid and n_H denotes the size of a discrete problem on the coarse grid. Many different strategies exist for doing interpolation when MG is considered in the FDM setting (see Refs. 4,7,8, and 10). The most commonly used is linear or bilinear interpolation. In a FEM setting, the prolongation operator P_H^h is connected naturally with the FE basis functions associated with the coarse grid (see Refs 1,7, and 8). In the case of the one-dimensional problem in Equations (5) and (6), the elements of the interpolation matrix are given by $(P_H^h)_{g(j),g(l)} = \phi_l^H(x_j)$, where $g(l)$ is the global number of the node l on the coarse grid Ω_H and *g(j)* is the global number of the node *j* on the fine grid Ω_h. $\phi_l^H(x_j)$ is the value of the FE basis function associated with the node l from the coarse grid Ω_H at the point j of the fine grid with the coordinate x_j.

Before solving the coarse grid problem $A_H e_H = r_H$, we need to transfer the information about the fine grid residual r_h to the coarse grid Ω_H, thus getting r_H. This operation is the reverse of prolongation and is referred to as *restriction*. The restriction operator can be represented as $v_H = R_h^H v_h$, where $R_h^H \in R^{n_H \times n_h}$. The simplest restriction operator is *injection*, defined in one dimension as $(v_H)_j = (v_h)_{2j}$, $j = 1, \ldots, n_H - 1$, and the coarse grid vector takes the immediate values from the fine grid vector. Some more sophisticated restriction techniques include *half injection* and *full weighting* (see Refs 4,7, and 8). The important property of the full weighting restriction operator in the FDM setting is that it is the transpose of the linear interpolation operator up to the constant that depends on the spatial dimension d, $P_H^h = 2^d(R_h^H)^T$. In the FEM setting, the restriction and the interpolation operators simply are selected as the transpose of each other, $P_H^h = (R_h^H)^T$. The spatial dimension-dependent factor 2^d that appears in the relation between the restriction R_h^H and the interpolation P_H^h is the consequence of the residuals being taken pointwise in the FDM case and being element-weighted in the FEM case.

If the coarse grid problem is solved with sufficient accuracy, the two-grid correction scheme should work efficiently, providing that the interpolation of the error from coarse to fine grid is sufficiently accurate. This happens in cases when the error e_H is smooth. As the fixed-point iteration scheme applied to the coarse-grid problem smooths the error, it forms a complementary pair with the interpolation, and the pair of these numerical procedures work very efficiently.

V-Cycle Multigrid Scheme and Full Multigrid

If the coarse-grid problem $A_H e_H = r_H$ in Algorithm 1 is solved approximately, presumably by using the fixed-point iteration, the question is how to eliminate successfully the outstanding low-frequency modes on Ω_H? The answer lies in the recursive application of the two-grid scheme. Such a scheme would require a sequence of nested grids $\Omega_0 \subset \Omega_1 \subset \cdots \subset \Omega_L$, where Ω_0 is a sufficiently coarse grid (typically consisting of only a few nodes) to allow efficient exact solution of the residual equation, presumably by a direct solver. This scheme defines the V-cycle of MG, which is summarized in Algorithm 2:

Algorithm 2. V-Cycle multigrid (recursive definition): $u_L = \mathrm{MG}(A_L, f_L, \tilde{u}_L^{(0)}, v_1, v_2, L)$.

```
1:  function MG $(A_l, f_l, \tilde{u}_l^{(0)}, v_1, v_2, l)$
2:      $\tilde{u}_l \leftarrow G_h^{v_1}(\tilde{u}_l^{(0)}, f_l)$
3:      $r_l = f_l - A_l\tilde{u}_l$;
4:      $r_{l-1} = R_l^{l-1} r_l$
5:      if $l - 1 = 0$ then
6:          Solve exactly $A_{l-1}e_{l-1} = r_{l-1}$
7:      else
8:          $e_{l-1} = MG(A_{l-1}, r_{l-1}, 0, v_1, v_2, l-1)$
9:          $e_l = P_{l-1}^l e_{l-1}$
10:         $\tilde{u}_l = \tilde{u}_l + e_l$
11:         $\tilde{u}_l \leftarrow G_h^{v_2}(\tilde{u}_l, f_l)$
12:     end if
13:     return $\tilde{u}_l$
14: end function
```

A number of modifications of the basic V-cycle exist. The simplest modification is to vary the number of recursive calls γ of the MG function in Step 8 of the Algorithm 2. For $\gamma = 1$, we have the so-called V-cycle, whereas $\gamma = 2$ produces the so-called W-cycle (7).

Until now, we were assuming that the relaxation on the fine grid is done with an arbitrary initial guess $\tilde{u}_L^{(0)}$, most commonly taken to be the zero vector. A natural question in this context would be if it is possible to obtain an improved initial guess for the relaxation method. Such an approximation can be naturally obtained by applying a recursive procedure referred to as *nested iteration* (7,8,10). Assume that the model–problem in Equation (7) is discretized using a sequence of nested grids $\Omega_0 \subset \Omega_1 \subset \cdots \subset \Omega_L$. Then we can solve the problem on the coarsest level Ω_0 exactly, interpolate the solution to the next finer level, and use this value as the initial guess for the relaxation (i.e., $\tilde{u}_1^{(0)} = P_0^1 \tilde{u}_0$). This procedure can be continued until we reach the finest level L. Under certain assumptions, the error in the initial guess $\tilde{u}_L^{(0)} = P_{L-1}^L \tilde{u}_{L-1}$ on the finest level is of the order of the discretization error and only a small number of MG V-cycles is needed to achieve the desired level of accuracy. The combination of the nested iteration and the MG V-cycle leads to the *full multigrid* (FMG) algorithm, summarized in Algorithm 3:

Algorithm 3. Full multigrid: $\tilde{u}_L = \text{FMG}(A_L, f_L, v_1, v_2, L)$.

```
function $\tilde{u}_L = \text{FMG}(A_L, f_L, v_1, v_2, l)$
    Solve $A_0 \tilde{u}_0 = f_0$ with sufficient accuracy
    for $l$=1, $L$ do
        $\tilde{u}_l^{(0)} = \hat{P}_{l-1}^l \tilde{u}_{l-1}$
        $\tilde{u}_l = \text{MG}(A_l, f_l, \tilde{u}_l^{(0)}, v_1, v_2, l)$   % Algorithm 2
    end for
    return $\tilde{u}_L$
end function
```

The interpolation operator $\hat{P}_{l-1}^l$ in the FMG scheme can be different, in general, from the interpolation operator P_{l-1}^l used in the MG V-cycle. A FMG cycle can be viewed as a sequence of V-cycles on progressively finer grids, where each V-cycle on the grid l is preceeded by a V-cycle on the grid $l - 1$.

Computational Complexity of Multigrid Algorithms

We conclude this section with an analysis of algorithmic complexity for MG, both in terms of the execution time and the storage requirements. For simplicity, we assume that the model problem in d spatial dimensions is discretized on a sequence of uniformly refined grids (in one and two spatial dimensions, the model–problem is given by Equations (5) and (6) and Equations (8) and (9), respectively). Denote the storage required to fit the discrete problem on the finest grid by M_L. The total memory requirement for the MG scheme then is $M_{MG} = \mu M_L$, where μ depends on d but not on L. This cost does not take into account the storage needed for the restriction/prolongation operators, although in some implementations of MG, these operators do not need to be stored explicitly. For the case of uniformly refined grids, the upper bound for the constant μ is 2 in one dimension, and $\frac{4}{3}$ in two dimensions (by virtue of $\mu = \sum_{j=0}^{L} \frac{1}{2^{jd}} < \sum_{j=0}^{\infty} \frac{1}{2^{jd}}$). For non uniformly refined grids, μ is larger but M_{MG} still is a linear function of M_L, with the constant of proportionality independent of L.

Computational complexity of MG usually is expressed in terms of work units (WU). One WU is the arithmetic cost of one step of a splitting iteration applied to a discrete problem on the finest grid. The computational cost of a V(1, 1) cycle of MG (V(1,1) is a V-cycle with one pre smoothing and one post smoothing iteration at each level and is v times larger than WU, with $v = 4$ for $d = 1$, $v = \frac{8}{3}$ for $d = 2$ and $v = \frac{16}{7}$ for $d = 3$ (7). These estimates do not take into account the application of the restriction/interpolation. The arithmetic cost of the FMG algorithm is higher than that of a V-cycle of MG. For the FMG with one relaxation step at each level, we have $v = 8$ for $d = 1$, $v = \frac{7}{2}$ for $d = 2$, and $v = \frac{5}{2}$ for $d = 3$ (7). Now we may ask how many V-cycles of MG or FMG are needed to achieve an iteration error commensurate with the discretization error of the FDM/FEM. To answer this question, one needs to know more about the convergence characteristics of MG. For this, deeper mathematical analysis of spectral properties of the two-grid correction scheme is essential. This analysis falls beyond the scope of our presentation. For further details see Refs 7–10. If a model–problem in d spatial dimensions is discretized by a uniform square grid, the discrete problem has $O(m^d)$ unknowns, where m is the number of grid lines in each spatial dimension. If the $V(v_1, v_2)$ cycle of MG is applied to the solution of the discrete problem with fixed parameters v_1 and v_2, the convergence factor τ in Equation (17) is bounded independently of the discretization parameter h (for a rigorous mathematical proof, see Refs 8–10). For the linear systems obtained from the discretization of the second-order, elliptic, self-adjoint PDEs, the convergence factor τ of a V-cycle typically is of order 0.1. To reduce the error $e_h = u_h - \tilde{u}_h$ to the level of the disretization error, we need to apply $O(\log m)$ V-cycles. This means that the total cost of the V-cycle scheme is $O(m^d \log m)$. In the case of the FMG scheme, the problems discretized on grids $\Omega_l, l = 0, 1, \ldots, L-1$ already are solved to the level of discretization error before proceeding with the solution of the problem on the finest grid Ω_L. In this way, we need to perform only $O(1)$ V-cycles to solve the problem on the finest grid. This process makes the computational cost of the FMG scheme $O(m^d)$, and it is an optimal solution (method).

ADVANCED TOPICS

The basic MG ideas are introduced in the previous section for the case of a linear, scalar and stationary DE with a simple grid hierarchy. This section discusses some important issues when MG is applied to more realistic problems (non linear and/or time-dependent DEs and systems of DEs), with complex grid structures in several spatial dimensions, and when the implementation is done on modern computer architectures.

Non linear Problems

The coarse grid correction step of the MG algorithm is not directly applicable to discrete non linear problems as the superposition principle does not apply in this case. Two basic approaches exist for extending the application of MG methods to non linear problems. In the indirect approach, the MG algorithm is employed to solve a sequence of linear problems that result from certain iterative global linearization procedures, such as *Newton's method*. Alternately, with the slight modification of the grid transfer and coarse grid operators, the MG algorithm can be transformed into the Brandt *Full Approximation Scheme (FAS)* algorithm (6,7), which can be applied directly to the non linear discrete equations.

In the FAS algorithm, the non linear discrete problem defined on a fine grid Ω_h as $A_h(u_h) = f_h$ is replaced by

$$A_h(\tilde{u}_h + e_h) - A_h(\tilde{u}_h) = f_h - A_h(\tilde{u}_h) \qquad (18)$$

The Equation (18) reduces in the linear case to the residual Equation (11). Define a coarse-grid approximation of Equation (18)

$$A_H\left(\hat{R}_h^H \tilde{u}_h + e_H\right) - A_H\left(\hat{R}_h^H \tilde{u}_h\right) = \hat{R}_h^H\left(f^h - A_h(\tilde{u}_h)\right) \quad (19)$$

where $\hat{R}_h^H$ is some restriction operator. Note that if the discrete non linear problem in Equation (3) comes from the FEM approximation, the restriction operators $\hat{R}_h^H$ in Equation (19) should differ by the factor of 2^d, where d is the spatial dimension (see the discussion in the section "Two-Grid Correction Method"). Introducing the approximate coarse grid discrete solution $\tilde{u}_H = \hat{R}_h^H \tilde{u}_h + e_H$ and the *fine-to-coarse correction* $\tau_H^h = A_H(\hat{R}_h^H \tilde{u}_h) - \hat{R}_h^H A_h(\tilde{u}_h)$, the coarse grid problem in the FAS MG algorithm becomes

$$A_H(\tilde{u}_H) = R_h^H f_h + \tau_H^h \qquad (20)$$

Then, the coarse grid correction

$$\tilde{u}_h \leftarrow \tilde{u}_h + \hat{P}_H^h e_H = \tilde{u}_h + \hat{P}_H^h\left(\tilde{u}_H - \hat{R}_h^H \tilde{u}_h\right) \qquad (21)$$

can be applied directly to non linear problems. In Equation (21) $\hat{R}_h^H \tilde{u}_h$ represents the solution of Equation (20) on coarse grid Ω_H. This means that once the solution of the fine grid problem is obtained, the coarse grid correction does not introduce any changes through the interpolation. The fine-to-coarse correction τ_H^h is a measure of how close the approximation properties of the coarse grid equations are to that of the fine grid equations.

When the FAS MG approach is used, the global Newton linearization is not needed, thus avoiding the storage of large Jacobian matrices. However, the linear smoothing algorithms need to be replaced by non linear variants of the relaxation schemes. Within a non linear relaxation method, we need to solve a non linear equation for each component of the solution. To facilitate this, local variants of the Newton method commonly are used. The FAS MG algorithm structure and its implementation are almost the same as the linear case and require only small modifications. For non linear problems, a proper selection of the initial approximation is necessary to guarantee convergence. To this end, it is recommended to use the FMG method described in the Basic Concepts section above.

Systems of Partial Differential Equations

Many complex problems in physics and engineering cannot be described by simple, scalar DEs. These problems, such as DEs with unknown vector fields or multi-physics problems, usually are described by systems of DEs. The solution of systems of DEs is usually a challenging task. Although no fundamental obstacles exist to applying standard MG algorithms to systems of DEs, one needs to construct the grid transfer operators and to create the coarse grid discrete operators. Moreover, the smoothing schemes need to be selected carefully.

Problems in structural and fluid mechanics frequently are discretized using staggered grids (10). In such cases, different physical quantities are associated with different nodal positions within the grid. The main reason for such distribution of variables is the numerical stability of such schemes. However, this approach involves some restrictions and difficulties in the application of standard MG methods. For instance, using the simple injection as a restriction operator may not be possible. An alternative construction of the restriction operator is based on averaging the fine grid values. Moreover, the non matching positions of fine and coarse grid points considerably complicates the interpolation near the domain boundaries. The alternative approach to the staggered grid discretization is to use non structured grids with the same nodal positions for the different types of unknowns. Numerical stabilization techniques are necessary to apply in this context (2,8).

The effectiveness of standard MG methods, when applied to the solution of systems of DEs, is determined to a large extent by the effectiveness of a relaxation procedure. For scalar elliptic problems, it is possible to create nearly optimal relaxation schemes based on standard fixed-point iterations. The straightforward extension of this scalar approach to systems of DEs is to group the discrete equations either with respect to the grid points or with respect to the discrete representations of each DE in the system (the latter corresponds to the blocking of a linear system coefficient matrix). For effective smoothing, the order in which the particular grid points/equations are accessed is very important and should be adjusted for the problem at hand. Two main approaches exist for the decoupling of a system: *global* decoupling, where for each DE all the grid points are accessed simultaneously, and *local* decoupling, where all DEs are accessed for each grid point.

The local splitting naturally leads to *collective* or *block* relaxation schemes, where all the discrete unknowns at a given grid node, or group of nodes, are relaxed simultaneously. The collective relaxation methods are very attractive within the FAS MG framework for non linear problems. Another very general class of smoothing procedures that are applicable in such situations are *distributive relaxations* (20). The idea is to triangulate locally the discrete operator for all discrete equations and variables within the framework of the smoother. The resulting smoothers also are called the *transforming smoothers* (25).

In the section on grid-coarsening techniques, an advanced technique based on the algebraic multigrid (AMG) for the solution of linear systems obtained by the discretization of systems of PDEs will be presented.

Time-Dependent Problems

A common approach in solving time-dependent problems is to separate the discretization of spatial unknowns in the problem from the discretization of time derivatives. Discretization of all spatial variables by some standard numerical procedure results in such cases in a differential-algebraic system (DAE). A number of numerical procedures, such as the method of lines, have been developed for the solution of such problems. Numerical procedures aimed at the solution of algebraic-differential equations usually are some modifications and generalizations of the formal methods developed for the solution of systems of ODEs [e.g., the Runge–Kutta method, the linear multistep methods, and the Newmark method (26,27)]. In a system of DAEs, each degree of freedom within the spatial discretization produces a single ODE in time. All methods for the integration of DAE systems can be classified into two groups: explicit and implicit. Explicit methods are computationally cheap (requiring only the sparse matrix–vector multiplication at each time step) and easy to implement. Their main drawback lies in stability, when the time step size changes (i.e., they are not stable for all step sizes). The region of stability of these methods is linked closely to the mesh size used in spatial discretization (the so-called CFL criterion). However, if a particular application requires the use of small time steps, then the solution algorithms based on explicit time-stepping schemes can be effective. The implicit time-stepping schemes have no restrictions with respect to the time step size, and they are unconditionally stable for all step sizes. The price to pay for this stability is that at each time step one needs to solve a system of linear or non linear equations. If sufficiently small time steps are used with the implicit schemes, standard iterative solvers based on fixed-point iterations or Krylov subspace solvers (4) can be more effective than the use of MG solvers. This particularly applies if the time extrapolation method or an explicit predictor method (within a predictor–corrector scheme) is used to provide a good initial solution for the discrete problem at each time step.

However, if sufficiently large time steps are used in the time-stepping algorithm, the discrete solution will contain a significant proportion of low-frequency errors introduced by the presence of diffusion in the system (20,27). The application of MG in such situations represents a feasible alternative. In this context, one can use the so-called "smart" time-stepping algorithms (for their application in fluid mechanics see Ref. 27). In these algorithms (based on the predictor–corrector schemes), the time step size is adjusted adaptively to the physics of the problem. If a system that needs to be solved within the corrector is non linear, the explicit predictor method is used to provide a good initial guess for the solution, thus reducing the overall computational cost. MG can be used as a building block for an effective solver within the corrector (see the section on preconditioning for the example of such solver in fluid mechanics).

Multigrid with Locally Refined Meshes

Many practically important applications require the resolution of small-scale physical phenomena, which are localized in areas much smaller than the simulation domain. Examples include shocks, singularities, boundary layers, or non smooth boundaries. Using uniformly refined grids with the mesh size adjusted to the small-scale phenomena is costly. This problem is addressed using adaptive mesh refinement, which represents a process of dynamic introduction of local fine grid resolution in response to unresolved error in the solution. Adaptive mesh refinement techniques were introduced first by Brandt (6). The criteria for adaptive grid refinement are provided by a posteriori error estimation (2).

The connectivity among mesh points is generally specified by the small subgroup of nodes that form the mesh cells. Typically, each cell is a simplex (for example, lines in one dimension; triangles or quadrilaterals in two dimensions; and tetrahedra, brick elements, or hexahedra in three dimensions). The refinement of a single grid cell is achieved by placing one or more new points on the surface of, or inside, each grid cell and connecting newly created and existing mesh points to create a new set of finer cells. A union of all refined cells at the given discretization level forms a new, locally refined, grid patch. Because adaptive local refinement and MG both deal with grids of varying mesh size, the two methods naturally fit together. However, it is necessary to perform some adaptations on both sides to make the most effective use of both procedures. To apply MG methods, the local grid refinement should be performed with the possibility of accessing locally refined grids at different levels.

The adaptive mesh refinement (AMR) procedure starts from a basic coarse grid covering the whole computational domain. As the solution phase proceeds, the regions requiring a finer grid resolution are identified by an error estimator, which produces an estimate of the discretization error, one specific example being $e_h = \hat{R}^h u - u_h$(28). Locally refined grid patches are created in these regions. This adaptive solution procedure is repeated recursively until either a maximal number of refinement levels is reached or the estimated error is below the user-specified tolerance. Such a procedure is compatible with the FMG algorithm. Notice that a locally refined coarse grid should contain both the information on the correction of the discrete solution in a part covered by it and the discrete solution, itself, in the remainder of the grid. The simplest and most natural way to achieve this goal is to employ a slightly modified FAS MG algorithm. The main difference between the FAS MG algorithm and the AMR MG algorithm is the additional interpolation that is needed at the interior boundary of each locally refined grid patch. For unstructured grids, it is possible to refine the grid cells close to the interior boundary in such a way to avoid interpolation at the interor boundaries altogether. For structured meshes, the internal boundaries of locally refined patches contain the so-called *hanging nodes* that require the interpolation from the coarse grid to preserve the FE solution continuity across the element boundaries. This interpolation may need to be defined in a recursive fashion if more than one level of refinement is introduced on a grid patch (this procedure resembles the long-range interpolation from the algebraic multigrid). The interpolation operator at the internal boundaries of the locally refined grids could be the same one used in the standard FMG algorithm. Figure 1 shows an example of

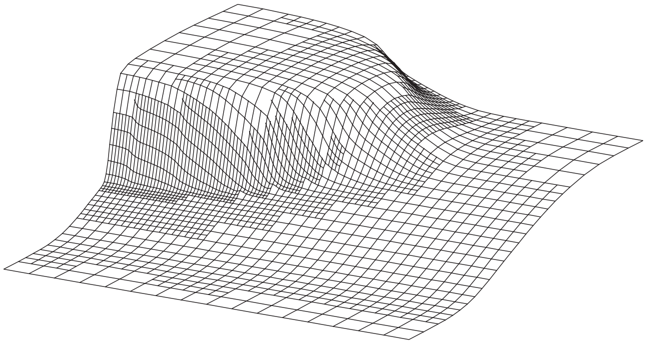


Figure 1. Composite multi level adaptive mesh for the MG solution of a dopant diffusion equation in a semiconductor fabrication.

the adaptive grid structure comprising several locally refined grid patches dynamically created in the simulation of dopant redistribution during semiconductor fabrication (20). Other possibilities exist in the context of the FAS MG scheme, for example, one could allow the existence of hanging nodes and project the non conforming solution at each stage of the FAS to the space where the solution values at the hanging nodes are the interpolants of their coarse grid parent values (29).

Various error estimators have been proposed to support the process of the local grid refinement. The multi level locally refined grid structure and the MG algorithm provide additional reliable and numerically effective evaluation of discretization errors. Namely, the grid correction operator τ_H^h [introduced in Equation (20)] represents the local discretization error at the coarse grid level Ω_H (up to a factor that depends on the ratio H/h). It is the information inherent to the FAS MG scheme and can be used directly as a part of the local grid refinement process. The other possibility for error estimation is to compare discrete solutions obtained on different grid levels of FMG algorithms to extrapolate the global discretization error. In this case, the actual discrete solutions are used, rather than those obtained after fine-to-coarse correction.

Another class of adaptive multi level algorithms are the fast adaptive composite-grid (FAC) methods developed in the 1980s (30). The main strength of the FAC is the use of the existing single-grid solvers defined on uniform meshes to solve different refinement levels. Another important advantage of the FAC is that the discrete systems on locally refined grids are given in the conservative form. The FAC allows concurrent processing of grids at given refinement levels, and its convergence rate is bounded independently of the number of refinement levels. One potential pitfall of both AMR MG and the FAC is in the multiplicative way various refinement levels are treated, thus implying sequential processing of these levels.

Grid-Coarsening Techniques and Algebraic Multigrid (AMG)

A multi level grid hierarchy can be created in a straightforward way by successively adding finer discretization levels to an initial coarse grid. To this end, nested global and local mesh refinement as well as non-nested global mesh generation steps can be employed. However, many practical problems are defined in domains with complex geometries. In such cases, an unstructured mesh or a set of composite block-structured meshes are required to resolve all the geometric features of the domain. The resulting grid could contain a large number of nodes to be used as the coarsest grid level in the MG algorithms. To take full advantage of MG methods in such circumstances, a variety of techniques have been developed to provide multi level grid hierarchy and generate intergrid transfer operators by coarsening a given fine grid.

The first task in the grid-coarsening procedure is to choose the coarse-level variables. In practice, the quality of the selected coarse-level variables is based on heuristic principles (see Refs. 4,7, and 8). The aim is to achieve both the quality of interpolation and a significant reduction in the dimension of a discrete problem on the coarse grids. These two requirements are contradictory, and in practice some tradeoffs are needed to meet them as closely as possible. The coarse-level variables commonly are identified as a subset of the fine grid variables based on the mesh connectivity and algebraic dependencies. The mesh connectivity can be employed in a graph-based approach, by selecting coarse-level variables at the fine mesh points to form the *maximal independent set* (MIS) of the fine mesh points. For the construction of effective coarse grids and intergrid transfer operators, it often is necessary to include the algebraic information in the coarse nodes selection procedure. One basic algebraic principle is to select coarse-level variables with a *strong connection* to the neighboring fine-level variables (strength of dependence principle) (31). This approach leads to a class of MG methods referred to as *algebraic multigrid (AMG)* methods. Whereas geometric MG methods operate on a sequence of nested grids, AMG operates on a hierarchy of progressively smaller linear systems, which are constructed in an automatic coarsening process, based on the algebraic information contained in the coefficient matrix.

Another class of methods identifies a coarse grid variable as a combination of several fine-grid variables. It typically is used in combination with a finite-volume discretization method (10), where the grid variables are associated with the corresponding mesh control volumes. The *volume agglomeration method* simply aggregates the fine control volumes into larger agglomerates to form the coarse grid space. The agglomeration can be performed by a *greedy algorithm*. Alternatively, the fine grid variables can be clustered directly (aggregated) into coarse-level variables based on the algebraic principle of strongly or weakly coupled neighborhoods. The *aggregation method* is introduced in Ref. 32, and some similar methods can be found in Refs. 33 and 34. The agglomeration method works also in the finite element setting (35).

The most common approach in creating the intergrid transfer operators within the grid-coarsening procedure is to formulate first the prolongation operator P_H^h and to use $R_h^H = (P_H^h)^T$ as a restriction operator. One way of constructing the prolongation operator from a subset of fine grid variables is to apply the *Delauney triangulation procedure* to the selected coarse mesh points. A certain finite element space associated to this triangulation can be used to create the interpolation operator P_H^h. The prolongation operators for the agglomeration and aggregation methods are defined in the way that each fine grid variable is represented by a single (agglomerated or aggregated) variable of the coarse grid.

The interpolation operator also can be derived using purely algebraic information. We start from a linear discrete problem in Equation (4) and introduce a concept of *algebraically smooth error* e_h. Algebraically smooth error is the error that cannot be reduced effectively by a fixed-point iteration. Note that the graph of an algebraically smooth error may not necessarily be a smooth function (7). The smooth errors are characterized by small residuals, for example, componentwise $|r_i| \ll a_{ii}|e_i|$. This condition can be interpreted broadly as

$$Ae \approx 0 \tag{22}$$

For the cases of model–problems in Equations (5) and (6) and Equations (8) and (9), where the coefficient matrices are characterized by zero row sums, Equation (22) means that we have component-wise

$$a_{ii}e_i \approx -\sum_{j \neq i} a_{ij}e_j \tag{23}$$

The relation in Equation (23) means that the smooth error can be approximated efficiently at a particular point if one knows its values at the neighboring points. This fact is the starting point for the development of the interpolation operator P_H^h for AMG. Another important feature in constructing the AMG interpolation is the fact that the smooth error varies slowly along the strong connections (associated with large negative off-diagonal elements in the coefficient matrix). Each equation in the Galerkin system describes the dependence between the neighboring unknowns. The i-th equation in the system determines which unknowns u_j affect the unknown u_i the most. If the value u_i needs to be interpolated accurately, the best choice would be to adopt the interpolation points u_j with large coefficients a_{ij} in the Galerkin matrix. Such points u_j are, in turn, good candidates for coarse grid points. The quantitative expression that u_i depends strongly on u_j is given by

$$-a_{ij} \geq \theta \max_{1 \leq k \leq n} \{-a_{ik}\}, \qquad 0 < \theta \leq 1 \tag{24}$$

Assume that we have partitioned the grid points into the subset of coarse grid points C and fine grid points F. For each fine grid point i, its neighboring points, defined by the nonzero off-diagonal entries in the i-th equation, can be subdivided into three different categories, with respect to the strength of dependence: coarse grid points with strong influence on $i(C_i)$, fine grid points with strong influence on $i(F_i^s)$, and the points (both coarse and fine) that weakly influence $i(D_i)$. Then, the relation in Equation (23) for algebraically smooth error becomes (for more details, see Ref. 31)

$$a_{ii}e_i \approx -\sum_{j \in C_i} a_{ij}e_j - \sum_{j \in F_i^s} a_{ij}e_j - \sum_{j \in D_i} a_{ij}e_j \tag{25}$$

The sum over weakly connected points D_i can be lumped with the diagonal terms, as a_{ij} is relatively small compared with a_{ii}. Also, the nodes $j \in F_i^s$ should be distributed to the nodes in the set C_i. In deriving this relation, we must ensure that the resulting interpolation formula works correctly for the constant functions. The action of the interpolation operator P_H^h obtained in this way can be represented as

$$(P_H^h e)_i = \begin{cases} e_i & i \in C \\ \sum\limits_{j \in C_i} \omega_{ij}e_j & i \in F \end{cases} \tag{26}$$

where

$$\omega_{ij} = -\frac{a_{ij} + \sum\limits_{k \in F_i^s} \dfrac{a_{ik}a_{kj}}{\sum\limits_{l \in C_i} a_{kl}}}{a_{ii} + \sum\limits_{m \in D_i} a_{im}} \tag{27}$$

Finally, the coarse grid discrete equations should be determined. The volume agglomeration methods allow us, in some cases, to combine the fine grid discretization equations in the formulation of the coarse grid ones. However, in most cases, the explicit discretization procedure is not possible and coarse grid equations should be constructed algebraically. An efficient and popular method to obtain a coarse-level operator is through the Galerkin form

$$A_H = R_h^H A_h P_H^h \tag{28}$$

For other, more sophisticated methods based on constrained optimization schemes see Ref. 36.

The AMG approach that was introduced previously is suitable for the approximate solution of linear algebraic systems that arise in discretizations of scalar elliptic PDEs. The most robust performance of AMG is observed for linear systems with coefficient matrices being SPD and M-matrices (the example being the discrete Laplace operator) (24). A reasonably good performance also can be expected for the discrete systems obtained from the discretizations of the perturbations of elliptic operators (such as the convection–diffusion–reaction equation). However, when the AMG solver, based on the standard coarsening approach, is applied to the linear systems obtained from the discretization of non scalar PDEs or systems of PDEs (where the coefficient matrices are substantially different from the SPD M-matrices), its performance usually deteriorates significantly (if converging at all). The reason for this is that the classical AMG uses the *variable-based approach*. This approach treats all the unknowns in the system equally. Thus, such approach cannot work effectively for systems of PDEs, unless a very weak coupling exists between the different unknowns. If the different types of unknowns in a PDE system are coupled tightly, some modifications and extensions of the classical approach are needed to improve the AMG convergence characteristics and robustness (see Ref. 37).

An initial idea of generalizing the AMG concept to systems of PDEs resulted in the *unknown-based approach*. In this approach, different types of unknowns are treated separately. Then, the coarsening of the set of variables that corresponds to each of the unknowns in a PDE system is based on the connectivity structure of the submatrix that corresponds to these variables (the diagonal block of the overall coefficient matrix, assuming that the variables corresponding to each of the unknowns are enumerated consecutively), and interpolation is based on the matrix entries in each submatrix separately. In contrast, the coarse-level Galerkin matrices are assembled using the whole fine level matrices. The unknown-based approach is the simplest way of generalizing the AMG for systems of PDEs. The additional information that is needed for the unknown-based approach (compared with the classical approach) is the correspondence between the variables and the unknowns. The best performance of AMG with the unknown-based approach is expected if the diagonal submatrices that corrspond to all the unknowns are close to M-matrices. This approach is proved to be effective for the systems involving non scalar PDEs [such as linear elasticity (38)]. However, this approach loses its effectiveness if different unknowns in the system are coupled tightly.

The latest development in generalizations of AMG for the PDE systems is referred to as the *point-based approach*. In this approach, the coarsening is based on a set of points, and all the unknowns share the same "grid" hierarchy. In a PDE setting, points can be regarded as the physical grid points in space, but they also can be considered in an abstract setting. The coarsening process is based on an auxiliary matrix, referred to as the *primary matrix*. This matrix should be defined to be the representative of the connectivity patterns for all the unknowns in the system, as the same coarse levels are associated to all the unknowns. The primary matrix is defined frequently based on the distances between the grid points. Then, the associated coarsening procedure is similar to that of geometric MG. The interpolation procedure can be defined as block interpolation (by approximating the block equations), or one can use the variable-based approaches, which are the same (s-interpolation) or different (u-interpolation) for each unknown. The interpolation weights are computed based on the coefficients in the Galerkin matrix, the coefficients of the primary matrix, or the distances and positions of the points. The interpolation schemes deployed in classical AMG [such as direct, standard, or multi pass interpolation (8)] generalize to the point-based AMG concept.

Although great progress has been made in developing AMG for systems of PDEs, no unique technique works well for all systems of PDEs. Concrete instances of this concept are found to work well for certain important applications, such as CFD (segregated approaches in solving the Navier–Stokes equations), multiphase flow in porous media (39), structural mechanics, semiconductor process and device simulation (40,41), and so further. Finally, it should be pointed out that the code SAMG (42), based on the point approach, was developed at Fraunhofer Institute in Germany.

Multigrid as a Preconditioner

In the Basic Concepts section, the fixed-point iteration methods were introduced. Their ineffectiveness when dealing with the low-frequency error components motivated the design of MG techniques. Fixed-point iterations are just one member of the broad class of methods for the solution of sparse linear systems, which are based on projection. A projection process enables the extraction of an approximate solution of a linear system from a given vector subspace. The simplest case of the projection methods are one-dimensional projection schemes, where the search space is spanned by a single vector. The representatives of one-dimensional projection schemes are the steepest descent algorithm and the minimum residual iteration (4).

One way of improving the convergence characteristics of the projection methods is to increase the dimension of the vector subspace. Such approach is adopted in the Krylov subspace methods (2,4). Krylov methods are based on the projection process onto the Krylov subspace defined as

$$K_m(A,y) = \mathrm{span}\,(y, Ay, \ldots, A^{m-1}y) \tag{29}$$

In Equation (29), A is the coefficient matrix and the vector y usually is taken to be the initial residual $r^{[0]} = f - Ax^{[0]}$. The main property of the Krylov subspaces is that their dimension increases by 1 at each iteration. From the definition in Equation (29) of the Krylov subspace, it follows that the solution of a linear system $Ax = f$ is approximated by $x = A^{-1} f \approx q_{m-1}(A) r^{[0]}$, where $q_{m-1}(A)$ is the matrix polynomial of degree $m - 1$. All Krylov subspace methods work on the same principle of polynomial approximation. Different types of Krylov methods are obtained from different types of orthogonality constraints imposed to extract the approximate solution from the Krylov subspace (4). The well-known representative of the Krylov methods aimed for the linear systems with SPD coefficient matrices is the conjugate gradient (CG) algorithm.

The convergence characteristics of Krylov iterative solvers depend crucially on the spectral properties of the coefficient matrix. In the case of the CG algorithm, the error norm reduction at the k-th iteration can be represented as (2,4)

$$\frac{\|e^{[k]}\|_A}{\|e^{[0]}\|_A} \le \min_{q_{k-1} \in P^0_{k-1}} \max_j |q_{k-1}(\lambda_j)| \tag{30}$$

where q_{k-1} is the polynomial of degree $k - 1$ such that $q_{k-1}(0) = 1$ and $\| \cdot \|_A^2 = (A\cdot, \cdot)$. The estimate in Equation (30) implies that the contraction rate of the CG algorithm at the iteration k is determined by the maximum value of the characteristic polynomial q_{k-1} evaluated at the eigenvalues λ_j of the coefficient matrix. This implies that the rapid convergence of the CG algorithm requires the construction of the characteristic polynomial q of relatively low degree, which has small value at all eigenvalues of A. This construction is possible if the eigenvalues of A are clustered tightly together. Unfortunately, the coefficient matrices that arise in various discretizations of PDEs have the eigenvalues spread over the large areas of the real axis (or the complex plane). Moreover, the spectra of such coefficient matrices change when the discretization parameter or other problem parameters vary. The prerequisite for rapid and robust convergence of Krylov solvers is the redistribution of the coefficient matrix eigenvalues. Ideally, the eigenvalues of the transformed coefficient matrix should be clustered tightly, and the bounds of the clusters should be independent on any problem or discretization parameters. As a consequence, the Krylov solver should converge to a prescribed tolerance within a small, fixed number of iterations. The numerical technique that transforms the original linear system $Ax = f$ to an equivalent system, with the same solution but a new coefficient matrix that has more favorable spectral properties (such as a tightly clustered spectrum), is referred to as *preconditioning* (2,4). Designing sophisticated preconditioners is problem-dependent and represents a tradeoff between accuracy and efficiency.

Preconditioning assumes the transformation of the original linear system $Ax = f$ to an equivalent linear system $M^{-1}Ax = M^{-1}f$. In practice, the preconditioning matrix (or the preconditioner) M is selected to be spectrally close to the matrix A. The application of a preconditioner requires at each iteration of a Krylov solver that one solves (exactly or approximately) a linear system $Mz = r$, where r is the current residual. This operation potentially can represent a considerable computational overhead. This overhead is the main reason why the design of a good preconditioner assumes the construction of such matrix M that can be assembled and computed the action of its inverse to a vector at optimal cost. Moreover, for preconditioning to be effective, the overall reduction in number of iterations of the preconditioned Krylov solver and, more importantly, the reduction in the execution time should be considerable, when compared with a non preconditioned Krylov solver (43). To design a (nearly) optimal Krylov solver for a particular application, the preconditioner needs to take into account the specific structure and spectral properties of the coefficient matrix.

In previous sections, we established MG as the optimal iterative solver for the linear systems that arise in discretizations of second-order, elliptic, PDEs. As an alternative to using MG as a solver, one may consider using it as a preconditioner for Krylov subspace solvers. When MG is used as a preconditioner, one applies a small number (typically 1 to 2) of V-cycles, with a small number of pre- and post smoothing iterations, at each Krylov iteration to approximate the action of M^{-1} to a residual vector. This approach works well for SPD linear systems arising from the discretization of second-order, elliptic PDEs. A number of similar MG-based approaches work well in the same context.

One group of MG-based preconditioners creates explicitly the transformed system matrix $M_L^{-1}AM_R^{-1}$, where M_L and M_R are the left and the right preconditioning matrices, respectively. A typical representative is the hierarchical basis MG (HBMG) method (44). The HBMG method is introduced in the framework of the FE approximations based on nodal basis. In the HBMG method, the right preconditioning matrix M_R is constructed in such a way that the solution x is contained in the space of hierarchical basis functions. Such space is obtained by replacing, in a recursive fashion, the basis functions associated with the fine grid nodes that also exist at the coarser grid level, by the corresponding coarse grid nodal basis functions. The process is presented in Fig. 2, where the fine grid basis functions that are replaced by the coarse grid basis functions are plotted with the dashed lines. The HBMG may be interpreted as the standard MG method, with the smoother at each level applied to the unknowns existing only at this level (and not present at coarser levels). The common choice $M_L = M_R^t$ extends the important properties of the original coefficient matrix, such as symmetry and positive definitness, to the preconditioned matrix. The HBMG method is suitable for implementation on adaptive, locally refined grids. Good results are obtained for the HBMG in two dimensions (with uniformly/locally refined meshes), but the efficiency deteriorates in 3-D.

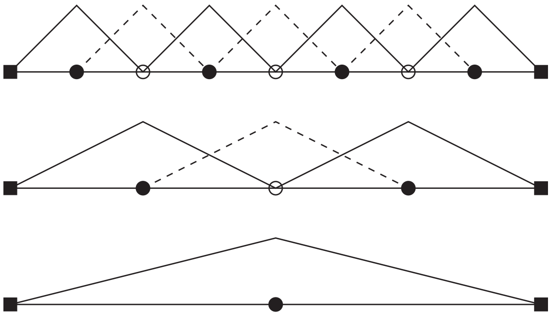


Figure 2. The principles of creating hierarchical basis in HBMG methods.

Another important MG-like preconditioning method is the so-called BPX preconditioner (45). It belongs to a class of parallel multi level preconditioners targeted for the linear systems arising in discretizations of second-order, elliptic PDEs. The main feature of this preconditioner is that it is expressed as a sum of independent operators defined on a sequence of nested subspaces of the finest approximation space. The nested subspaces are associated with the sequence of uniformly or adaptively refined triangulations and are spanned by the standard FEM basis sets associated with them. Given a sequence of nested subspaces $S_1^h \subset S_2^h \subset \cdots S_L^h$ with $S_l^h = \mathrm{span}\{\phi_k^l\}_{k=1}^{N_l}$, the action of the BPX preconditioner to a vector can be represented by

$$P^{-1}v = \sum_{l=1}^{L}\sum_{k=1}^{N_l}(v, \phi_k^l)\phi_k^l \tag{31}$$

The condition number of the preconditioned discrete operator $P^{-1}A$ is at most $O(L^2)$, and this result is independent on the spatial dimension and holds for both quasi-uniform and adaptively refined grids. The preconditioner in Equation (31) also can be represented in the operator form as

$$P^{-1} = \sum_{l=1}^{L} R_l Q_l \tag{32}$$

where R_l is an SPD operator $S_l^h \mapsto S_l^h$ and $Q_l : S_L^h \mapsto S_l^h$ is the projection operator defined by $(Q_l u, v) = (u, v)$ for $u \in S_L^h$ and $v \in S_l^h$ [$(\cdot,\cdot)$ denotes the l_2-inner product]. The cost of applying the preconditioner P will depend on the choice of R_l. The preconditioner in Equation (32) is linked closely to the MG V-cycle. The operator R_l has the role of a smoother; however, in BPX, preconditioner smoothing at every level is applied to a fine grid residual (which can be done concurrently). In MG, smoothing of the residual at a given level cannot proceed until the residual on that level is formed using the information from the previous grids. This process also involves an extra computational overhead in the MG algorithm. Parallelization issues associated with the BPX preconditioner will be discussed in the final section.

Many important problems in fundamental areas, such as structural mechanics, fluid mechanics, and electromagnetics, do not belong to the class of scalar, elliptic PDEs. Some problems are given in so-called mixed forms (2) or as the systems of PDEs. The latter is the regular case when multiphysics and multiscale processes and phenomena are modeled. When the systems of PDEs or PDEs with unknown vector fields are solved, the resulting discrete operators (coefficient matrices) have spectral properties that are not similar to the spectral properties of discrete elliptic, self-adjoint problems, for which MG is particularly suited. Thus, direct application of standard MG as a preconditioner in such complex cases is not effective. In such cases, it is possible to construct the block preconditioners—an approach that proved efficient in many important cases (2). A coefficient matrix is subdivided into blocks that correspond to different PDEs/unknowns in the PDE system (similarly to the unknown-based approach in the AMG for systems of PDEs). If some main diagonal blocks represent the discrete second-order, elliptic PDEs (or their perturbations), they are suitable candidates to be approximated by the standard MG/AMG with simple relaxation methods. The action of the inverse of the whole preconditioner is achieved by the block back-substitution. Successful block preconditioners have been developed for important problems in structural mechanics (46), fluid mechanics (2,47), and electromagnetics (48). In the following list, a brief review of some of these techniques is given.

1. Mixed formulation of second-order elliptic problems. The boundary value problem is given by

$$\begin{aligned} A^{-1}\vec{u} - \nabla p &= 0 \\ \nabla \cdot \vec{u} &= -f \end{aligned} \tag{33}$$

in $\Omega \subset \mathbb{R}^d$, subject to suitable BCs. In Equation (33) $A : \mathbb{R} \mapsto \mathbb{R}^{d\times d}$ is a symmetric and uniformly positive definite matrix function, p is the pressure, and $\vec{u} = A^{-1}p$ is the velocity. Problems of this type arise in modeling of fluid flow through porous media. The primal variable formulation of Equation (33) is $-\nabla \cdot A\nabla p = f$, but if $\vec{u}$ is the quantity of interest, a numerical solution of the mixed form in Equation (33) is preferable. Discretization of Equation (33) using Raviart–Thomas FEM leads to a linear system with an indefinite coeffiient matrix

$$\begin{bmatrix} M_A & B^t \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} g \\ f \end{bmatrix} \tag{34}$$

with $(M_A)_{ij} = (A\vec{\varphi}_i, \vec{\varphi}_j), i, j = 1, \ldots, m$. For details, see Ref. 47. The optimal preconditioner for the linear system in Equation (34) is given by

$$M = \begin{bmatrix} M_A & 0 \\ 0 & BM_A^{-1}B^t \end{bmatrix} \tag{35}$$

The exact implementation of this preconditioner requires the factorization of the dense Schur complement matrix $S = BM_A^{-1}B^t$). The matrix S can be approximated without significant loss of efficiency by the matrix $S_D = B\mathrm{diag}\,(M_A^{-1})B^t$, which is a sparse matrix. The action of S_D^{-1} to a vector can be approximated further by a small number of AMG V-cycles (47).

2. Lame's equations of linear elasticity. This problem is a fundamental problem in structural mechanics. It also occurs in the design of integrated electronic components (microfabrication). The equations by Lamé represent a displacement formulation of linear elasticity:

$$\rho\frac{\partial^2\vec{u}}{\partial t^2} - \mu\Delta\vec{u} + (\lambda + \mu)\nabla(\nabla \cdot \vec{u}) = \vec{f} \tag{36}$$

subject to a suitable combination of Dirichlet and Neumann BCs. Equation (36) is a special case of non linear equations of visco-elasticity. In Equation (36), ρ denotes density of a continuous material body, $\vec{u}$ is the displacement of the continuum from the equilibrium position under the combination of external forces $\vec{f}$, and λ and μ are the Lamé constants (see Ref. 49 for more details). After the discretization of Equation (36) by the FEM, a linear system $A_e u = f$ is obtained, with $A_e \in \mathbb{R}^{nd\times nd}, u$, and $f \in \mathbb{R}^{nd}$, where n is the number of unconstrained nodes and d is the spatial dimension. If the degrees of freedom are enumerated in such a way that the displacements along one Cartesian direction (unknowns that correspond to the same displacement component) are grouped together, the PDE system in Equation (36) consists of d scalar equations and can be preconditioned effectively by a block-diagonal preconditioner. For this problem, effective solvers/preconditioners based on the variable and the point approach in AMG also are developed. However, the block-diagonal preconditioner, developed in (46) uses classical (unknown-based) AMG. From Equation (36) it can be seen that each scalar PDE represents a pertur- bation of the standard Laplacian operator. Thus, the effectiveness of MG applied in this context depends on the relative size of the perturbation term. For compressible materials (λ away from ∞), the block-diagonal preconditioner (46) is very effective. For nearly incompressible cases ($\lambda \to \infty$), an alternative formulation of the linear elasticity problem in Equation (36) is needed (see the Stokes problem).

3. The biharmonic equation. The biharmonic problem $\Delta^2 u = f$ arises in structural and fluid mechanics (2,49). FEM discretization of the original formulation of the biharmonic problem requires the use of Hermitian elements that result in a system with SPD coefficient matrix, which is not an M-matrix. Consequently, direct application of the standard GMG/AMG preconditioners will not be effective.

An alternative approach is to reformulate the biharmonic problem as the system of two Poisson equations:

$$\begin{aligned} -\Delta u &= v \\ -\Delta v &= f \end{aligned} \tag{37}$$

subject to suitable BCs (2). Discretization of Equation (37) can be done using the standard Lagrangian FEM, producing an indefinite linear system

$$\begin{bmatrix} M_B & A^t \\ A & 0 \end{bmatrix} \begin{bmatrix} v \\ u \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} \tag{38}$$

with $(M_B)_{ij} = (\phi_i, \phi_j), i, j = 1, \ldots, m,$ and $(A)_{ij} = (\nabla\phi_j, \nabla\phi_i), i = 1, \ldots, n, j = 1, \ldots, m$. In Ref. 50, the application of a modified MG method to the Schur complement system obtained from Equation (38) is discussed. Another nearly optimal preconditioner for the system in Equation (38) can be obtained by decoupling the free and constrained nodes in the first equation of Equation (37). This decoupling introduces a 3×3 block splitting of the coefficient matrix, and the preconditioner is given by (see Ref. 51)

$$M = \begin{bmatrix} 0 & 0 & A_I^t \\ 0 & M_B & A_B^t \\ A_I & A_B & 0 \end{bmatrix} \tag{39}$$

In Equation (39), the matrix block $A_I \in \mathbb{R}^{n \times n}$ corresponds to a discrete Dirichlet Laplacian operator and it can be approximated readily by a small fixed number of GMG/AMG V-cycles.

The Stokes Problem. The Stokes equations represent a system of PDEs that model visous flow:

$$\begin{aligned} -\Delta \vec{u} + \nabla p &= \vec{0} \\ \nabla \cdot \vec{u} &= 0 \end{aligned} \tag{40}$$

subject to a suitable set of BCs (see Ref. 2). The variable $\vec{u}$ is a vector function representing the velocity, wheras the scalar function p is the pressure. A host of suitable pairs of FE spaces can be used to discretize Equation (40). Some combinations are stable, and some require the stabilization (2,27). If a stable discretization is applied (see Ref. 2 for more discussion), the following linear system is obtained

$$\begin{bmatrix} A & B^t \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} \tag{41}$$

In Equation (41), $A \in \mathbb{R}^{m \times m}$ represents the vector Laplacian matrix (a block-diagonal matrix consisting of d scalar discrete Laplacians where d is the spatial dimension), and $B \in \mathbb{R}^{n \times m}$ is the discrete divergence matrix. The unknown vectors u and p are the discrete velocity and pressure solutions, respectively.

Optimal preconditioners for the discrete Stokes problem in Equation (41) are of the form:

$$M = \begin{bmatrix} A & 0 \\ 0 & M_p \end{bmatrix} \tag{42}$$

where M_p is the pressure mass matrix, which is spectrally equivalent to the Schur complement matrix $BA^{-1}B^t$. The vector Laplacian matrix A can be approximated componentwise by a small number of V-cycles of GMG/AMG (2).

The Navier–Stokes Problem. The Navier–Stokes equations have similar block structure as the Stokes equations, with the vector Laplacian operator replaced by the vector convection-diffusion operator. The steady-state Navier–Stokes system is given by

$$\begin{aligned} -v\nabla^2 \vec{u} + \vec{u} \cdot \nabla \vec{u} + \nabla p &= \vec{f} \\ \nabla \cdot \vec{u} &= 0 \end{aligned} \tag{43}$$

In Equation (43), $\vec{u}$ and p are the velocity and pressure and $v > 0$ is kinematic viscosity. The Navier–Stokes equations represent a model of fluid flow of an incompressible Newtonian fluid. The system in Equation (43) is non linear. The use of mixed FEM for the discretization of Equation (43) leads to a non linear system of algebraic equations. Such systems need to be solved iteratively, either by the Picard method or the Newton method. If a stable discretization method is used in conjuction with the Picard linearization, we obtain the *discrete Oseen problem*(2):

$$\begin{bmatrix} F & B^t \\ B & 0 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} \tag{44}$$

In Equation (44), $F = vA + C$ is the discrete vetor convection-diffusion operator and B is, as before, the divergence matrix. The ideal preconditioning of the system in Equation (44) an be achieved by the blocks-triangular matrix (2)

$$M = \begin{bmatrix} M_F & B_t \\ 0 & -M_S \end{bmatrix} \tag{45}$$

In Equation (45), M_F is the spectrally equivalent approximation of the matrix F. This approximation can be achieved by a small number of V-cycles of GMG/AMG with appropriate smoothing [line, ILU(0)] applied to each component of F (2). The optimal choice for the block M_S would be $BF^{-1}B^t$ (the Schur complement). As the Schur complement is a dense matrix, some effective approximations are needed. One such choice leads to the pressure convection-diffusion preconditioner (2) with

$$BF^{-1}B^t \approx BM_u^{-1}B^tF_p^{-1}M_u \tag{46}$$

In Equation (46), M_u is the velocity mass matrix and F_p is the discretization of the convection-diffusion equation on the pressure space. Another effective choice for the Schur complement matrix approximation is to take (2)

$$BF^{-1}B^t \approx (BM_u^{-1}B^t)(BM_u^{-1}FM_u^{-1}B^t)^{-1}(BM_u^{-1}B^t) \quad (47)$$

Parallel Multigrid Methods

The MG methods commonly are applied to large-scale computational problems that involve millions of unknowns, which frequently are non linear and/or time-dependent, for which the use of modern parallel computers is essential. To benefit from MG in such cases, various techniques to parallelize MG algorithms have been developed.

The simplest and most commonly used approach of implementing MG algorithms in parallel is to employ the *grid partitioning technique* (52,53). This technique involves no real change to the basic MG algorithm. Namely, the finest grid level, level Ω_h, can be partitioned in a non overlapping set of subgrids

$$\Omega_h = \bigcup_{i=1}^{P} \Omega_h^{(i)} \quad (48)$$

and each of the resulting subgrids $\Omega_h^{(i)}$ can be assigned to one of P parallel processors. The partition on the finest grid induces, naturally, a similar partitioning of the coarser levels. The partitioning in Equation (48) should be performed in such a way that the resulting subgrids have approximately the same number of nodes. This requirement is necessary for good load balancing of the computations. In addition, the grid partitioning should introduce a small number of grid lines between the neighboring subdomains. It also is beneficial that the grid lines stretching between the neighboring subdomains define the weak connections between the unknowns. The former requirement is related to the minimization of the communication overhead between the neighboring processors, whereas the latter affects the efficiency and the convergence characteristics of the parallel MG algorithm. For structured grids defined on regular geometries, the partitioning in Equation (48) can be done fairly simply. However, this problem becomes progressively more complicated for non trivial geometries (possibly with the internal boundaries) on which a problem is discretized by unstructured or adaptively refined grids. In such cases, general-purpose algorithms for graph partitioning, such as METIS (54), provide the required balanced grid subdivision.

Two strategies of combining MG and domain decomposition for parallel computers exist. The simpler choice is to apply a serial MG algorithm on each of the subdomain problems, communicating the solution among the processors only at the finest level. This method is particularly effective if the subdomains are connected along physically narrow areas that actually are not visible on coarse levels. A more complex choice is to extend the communication to the coarse grid levels.

In some parallel MG algorithms based on the grid decomposition, different grid levels still are processed sequentially. When the number of grid variables at the coarse grid levels falls below a certain threshold, some processors could become idle. This effect can be particularly pronounced in applications with the local grid refinement. In some extreme cases, no single level, may exist with enough grid points to keep all the processors busy. Moreover, the number of grid levels obtained by the coarsening could be limited by the number of partitions in the domain decomposition. Therefore, the grid decomposition based parallelization of MG methods make sense only if the total number of unknowns in the multi level grid structure is significantly larger than the number of available processors.

The problem of sequential processing of different grid levels in standard MG algorithm can potentially represent a serious bottleneck for large-scale applications. The solution to this problem is to use a class of multi level (ML) methods known as the additive ML methods. In these methods, the solution (or the preconditioning) procedure is defined as a sum of independent operators. It should be emphasised that this type of parallelism is completely independent of the parallelism induced by the grid partitioning. As a result, the two techniques can be combined together, giving a powerful parallel solver/preconditioner for elliptic problems. An example of a parallel ML preconditioner, in which different grid levels can be processed simultaneously, is the BPX preconditioner (45) introduced in the previous section. The smoothing operator R_k in Equation (32) applies to the fine grid residual for all levels k. This application allows the smoothing operation to be performed concurrently for all levels. Another example of additive ML method can be derived from the FAC method introduced in the section on MG with locally refined meshes. The original FAC algorithm is inherently sequential in the sense that, although different grid levels can be processed asynchronously, the various refinement levels are treated in a multiplicative way (i.e., the action of the FAC solver/preconditioner is obtained as a product of the actions from different refinement levels). To overcome this obstacle, an asynchronous version of the FAC algorithm, referred to as AFAC, was developed (55). The AFAC has the convergence rate independent of the number of refinement levels and allows the use of uniform grid solvers on locally refined grid patches. However, these uniform grid solvers are, themselves, MG-based, which can lead potentially to a substantial computational overhead. A new version, AFACx (56), is designed to use the simple fixed-point iterations at different refinement levels without significant deterioration of the convergence characteristics when compared with the previous versions.

In this article, AMG was introduced as a prototype of a black-box solver and a preconditioner for elliptic PDEs and some other class of problems where the coefficient matrix has the properties that resemble the M-matrix. Because of its robustness and ease of use, AMG has become an obvious candidate as a solver for a variety of large-scale scientific applications. In such cases, the performance characteristics of the sequential AMG may not be sufficient. This reason is why a considerable research effort was devoted to the parallelization of AMG.

The application of AMG to the solution of a linear system consists of two phases: the coarsening and the solve phase (which implements the MG V-cycle). The parallelization of AMG is done using the domain decomposition approach based on the grid decomposition in Equation (48). In the solve phase, parallelization is restricted to fixed-point iterations. In this context, instead of the standard versions, some modifications are needed. The most common approach is the CF Gauss–Seidel method, which is performed independently on each subdomain using the frozen values in the boundary areas. The values in these areas are refreshed by the data communication between the neighboring subdomains.

In most of the application areas, the coarsening phase is the most time-consuming part of the algorithm. Thus, the efficient parallelization of AMG coarsening will have a major impact on AMG parallel performance. The problem is that the previously introduced classical coarsening algorithm, which also is referred to as the Ruge–Stüben (RS) coarsening, is inherently sequential. To perform the subdivision of a set of variables at each "grid" level into coarse (C) and fine (F) variables, based on the strength of dependence principle, one needs to start from an arbitrary variable and visit all the remaining variables in succession. To parallelize this process, a combination of the domain decomposition methodology and the RS coarsening was proposed. To facilitate this approach, a partitioning of the graph associated with the coefficient matrix is performed. In this context, it is desirable that the partitioning cuts only (or mostly) the weak connections in the graph. Moreover, it is necessary to mark the unknowns that have some of their connections cut, as these unknowns belong to the boundary layer between the neighboring subdomains and will be involved subsequently in the interprocessor communication. Most of the parallel coarsening schemes are based on the application of the standard RS coarsening scheme applied concurrently on each of the subdomains, employing some kind of special treatment of the points in the boundary layers. Two main reasons exist for this special treatment; the standard coarsening scheme usually creates an unnecessarily high concentration of the C points close to the processor boundaries, and possible coarsening inconsistencies may appear. The latter are manifested in strong F–F couplings across the subdomain boundaries, where both F points do not have a common C point. Several different modifications have been proposed to alleviate such problems. In the sequel, a brief review of some well-known modifications of the classical coarsening scheme is given.

Minimum Subdomain Blocking (MSB). This approach was the first used to parallelize the AMG coarsening phase (57). In this approach, the coarsening process in each subdomain is decoupled into the coarsening of the variables in the boundary layer (done by the classical RS coarsening scheme, taking into account only the connections within the layer) and the coarsening of the remainder of the subdomain (again, done by the classical RS algorithm). Such heuristics ensures that each of the F-points in the boundary layer has at least one connection to a C-point within the boundary layer. The main drawback of this approach is that the strong couplings across the subdomain boundaries are not taken into account. When MSB is used, the assembly of the interpolation operators is local for each subdomain (requiring no communication). The assembly of the coarse grid discrete operators does require some communication; however, it is restricted to the unknowns in the boundary layers of the neighboring subdomains.

Third-Pass Coarsening (RS3). This coarsening is an alternative approach to correct the problem of F–F dependencies across the subdomain boundaries that do not have a common C-point (58). In this approach, a two-pass standard RS coarsening is performed on each subdomain concurrently, before the third pass is performed on the points within the boundary layers. This method requires the communication between the neighboring subdomains. In the RS3 coarsening, additional coarse grid points can be created in each of the subdomains on demand from its neighboring subdomains. This fact may lead potentially to the load imbalance between the subdomains. One drawback of the RS3 coarsening is the concentration of the coarse points near the subdomain boundaries. Another problem is that the introduction of a large number of subdomains will make the coarsest grid problem unacceptably large (as each of the subdomains cannot be coarsened beyond a single grid point).

The CLJP Coarsening. This procedure is based on parallel graph-partitioning algorithms and is introduced in Ref. 59. In this approach, a directed weighted graph is defined with the vertices corresponding to the problem unknowns and the edges corresponding to the strong couplings. A weight is associated with each vertex, being equal to the number of strong couplings of the neighboring vertices to this vertex plus a random number. Random numbers are used to break ties between the unknowns with the same number of strong influences (and thus to enable parallelization of the coarsening procedure). The coarsening process proceeds iteratively, where at each iteration an independent set is chosen from the vertices of the directed graph. A point i is selected to be in an independent set if its weight is larger than the weights of all neighboring vertices. Then, the points in the independent set are declared as C-points. The main advantage of the CLJP coarsening is that it is entirely parallel and it always selects the same coarse grid points, regardless of the number of subdomains (this is not the case with the RS and the RS3 coarsenings). A drawback is that the CLJP coarsening process selects the coarse grids with more points than necessary. This action, in turn, increases the memory requirements and the complexity of the solution phase. A recent modification of the CLJP scheme that addresses these issues is proposed in [Ref. 60]. The interpolation operators and the coarse grid discrete operators are assembled in a usual way.

The Falgout Coarsening. This hybrid scheme involves both the classical RS and the CLJP coarsening, designed with the aim of reducing the drawbacks that each of these two schemes introduce. The Falgout coarsening (58) uses the RS coarsening in the interior of the subdomains and the CLJP coarsening near the subdomain boundaries.

Parallel AMG as a Preconditioner for 3-D Applications. When traditional coarsening schemes are applied to large-scale problems obtained from the discretizations of 3-D PDEs, the computational complexity and the memory requirements increase considerably, diminishing the optimal scalability of AMG. If AMG is used as a preconditioner in this context, the two coarsening schemes, based on the maximal independent set algorithm, referred to as the *parallel modified independent set* (PMIS) and the *hybrid modified independent set* (HIMS) (introduced in Ref. 61), can reduce the complexity issues significantly.

FINAL REMARKS

MG methods have been a subject of considerable research interest over the past three decades. Research groups at many institutions are involved in ongoing projects related to both theoretical and practical aspects of MG. It is virtually impossible to cover all the aspects of this versatile area of research and to cite all the relevant references within the limited space of this review. A comprehensive reference list with over 3500 units on MG can be found at [Ref. 62], together with some publicly available MG software (63). The ongoing interest of the scientific community in MG methods is reflected in two long-running regular conference series on MG methods (64,65), which attract an ever-increasing number of participants. Some additional relevant monographs, that were not previously mentioned in this presentation, include Refs. 11–15 and Ref. 66.

BIBLIOGRAPHY

1. K. Eriksson, D. Estep, P. Hansbo, and C. Johnson, *Computational Differential Equations*. Cambridge: Cambridge University Press, 1996.
2. H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers*. Oxford: Oxford University Press, 2005.
3. A. R. Mitchell and D. F. Griffiths, *The Finite Difference Method in Partial Differential Equations*. Chichester: Wiley, 1980.
4. Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA: SIAM, 2003.
5. R. P. Fedorenko, A relaxation method for solving elliptic difference equations, *USSR Computational Math. and Math. Physics*, **1**: 1092–1096, 1962.
6. A. Brandt, Multi-level adaptive solutions to boundary-value problems, *Math. Comput.*, **31**: 333–390, 1977.
7. W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, 2nd ed. Philadelphia, PA: SIAM, 2000.
8. U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid*. London: Academic Press, 2001.
9. W. Hackbusch, *Multi-grid methods and applications*. Berlin: Springer, 2003.
10. P. Wesseling, *An Introduction to Multigrid Methods*. Philadelphia, PA: R.T. Edwards, 2004.
11. J. H. Bramble, *Multigrid Methods*. Harlow: Longman Scientific and Technical, 1993.
12. U. Rüde, *Mathematical and Computational Techniques for Multilevel Adaptive Methods,* Vol. 13, Frontiers in Applied Mathematics. Philadelphia, PA: SIAM, 1993.
13. S. F. McCormick, *Multilevel Adaptive Methods for Partial Differential Equations, Vol. 6, Frontiers in Applied Mathematics*. Philadelphia, PA: SIAM, 1989.
14. V.V. Shaidurov, *Multigrid Methods for Finite Elements*. Dordrecht: Kluwer, 1995.
15. M. Griebel and C. Zenger: *Numerical simulation in science and engineering, Notes on numerical fluid mechanics, Vol. 48*. Braunschweig: Vieweg Verlag, 1994.
16. B. Koren, *Multigrid and Defect Correction for the Steady Navier–Stokes Equations Applications to Aerodynamics*. Amsterdam: Centrum voor Wiskunde en Informatica, 1991.
17. C. Douglas and G. Haase, Algebraic multigrid and Schur complement strategies within a multilayer spectral element ocean model, *Math. Models Meth. Appl. Sci.*, **13**(3): 309–322, 2003.
18. M. Brezina, C. Tong, and R. Becker, Parallel algebraic multigrid for structural mechanics, *SIAM J. Sci. Comput.*, **27**(5): 1534–1554, 2006.
19. A. Brandt, J. Bernholc, and K. Binder (eds.), *Multiscale Computational Methods in Chemistry and Physics*. Amsterdam: IOS Press, 2001.
20. W. Joppich and S. Mijalković, *Multigrid Methods for Process Simulation*. Wien: Springer-Verlag, 1993.
21. G. Haase, M. Kuhn, and U. Langer, Parallel multigrid 3D Maxwell solvers, *Parallel Comput.*, **27**(6): 761–775, 2001.
22. J. Hu, R. Tuminaro, P. Bochev, C. Garassi, and A. Robinson, Toward an h-independent algebraic multigrid for Maxwell's equations, *SIAM J. Sci. Comput.*, **27**(5): 1669–1688, 2006.
23. J. Jones and B. Lee, A multigrid method for variable coefficient Maxwell's equatons, *SIAM J. Sci. Comput*, **27**(5): 1689–1708, 2006.
24. G. H. Golub and C. F. Vanloan, *Matrix computations*. Baltimore, MD: J. Hopkins University Press, 1996.
25. G. Wittum, Multigrid methods for Stokes and Navier–Stokes equations—transforming smoothers: Algorithms and numerical results, *Numer. Math.*, **54**: 543–563, 1989.
26. U. M. Ascher, R. M. M. Mattheij, and R. D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Philadelphia, PA: SIAM, 1995.
27. P. M. Gresho and R. L. Sani, *Incompressible Flow and the Finite Element Method*. Chichester: Wiley, 1998.
28. R. E. Bank, *PLTMG: A software package for solving elliptic partial differential equations, Users' Guide 7.0, Vol. 15, Frontiers in applied mathematics*, Philadelphia, PA: SIAM, 1994.
29. A. C. Jones, P. K. Jimack, An adaptive multigrid tool for elliptic and parabolic systems, *Int. J. Numer. Meth. Fluids*, **47**: 1123–1128, 2005.
30. S. McCormick, Fast Adaptive Composite Grid (FAC) methods: Theory for the variational case, in: K. Bohmer and H. Setter (eds.), *Defect Correction Methods: Theory and Applications, Computation Supplementation, Vol. 5*. Berlin: Springer Verlag, 1984, pp. 131–144.
31. J. W. Ruge and K. Stüben, Algebraic multigrid, in: S. F. McCormick (ed.), *Multigrid methods, Vol. 3, Frontiers in applied mathematics*, Philadelphia, PA: SIAM, 1987, pp. 73–130.
32. P. Vanek, J. Mandel, and M. Brezina, Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, *Computing*, **56**: 179–196, 1996.

33. M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge, Algebraic multigrid based on element interpolation (AMGe), *SIAM J. Sci. Comput.*, **22**(5): 1570–1592, 2000.
34. T. Chartier, R. D. Falgout, V. E. Henson, J. Jones, T. Manteuffel, S. McCormick, J. Ruge, and P. Vassilevski, Spectral AMGe (ρAMGe), *SIAM J. Sci. Comput.*, **25**(1): 1–26, 2003.
35. J. E. Jones, P. S. Vassilevski, AMGe based on element agglomeration, *SIAM J. Sci. Comput.*, **23**(1): 109–133, 2001.
36. A. Brandt, General highly accurate algebraic coarsening, *Electronic Trans. Numerical Analysis*, **10**: 1–20, 2000.
37. K. Stüben, A review of algebraic multigrid, *J. Comput. Appl. Math.*, **128**: 281–309, 2001.
38. T. Füllenbach, K. Stüben, and S. Mijalković, Application of algebraic multigrid solver to process simulation problems, *Proc. Int. Conf. of Simulat. of Semiconductor Processes and Devices, 2000*, pp. 225-228.
39. K. Stüben, P. Delaney, and S. Chmakov, Algebraic multigrid (AMG) for ground water flow and oil reservoir simulation, *Proc. MODFLOW 2003*.
40. T. Füllenbach and K. Stüben, Algebraic multigrid for selected PDE systems, *Proc. 4th Eur. Conf. on Elliptic and Parabolic Problems, London, 2002*, pp. 399-410.
41. T. Clees and K. Stüben, Algebraic multigrid for industrial semiconductor device simulation, *Proc. 1st Int. Conf. on Challenges in Sci. Comput., 2003*.
42. K. Stüben and T. Clees, *SAMG user's manual*, Fraunhofer Institute SCAI. Available: http://www.scai.fhg.de/samg.
43. J. J. Dongarra, I. S. Duff, D. Sorensen, and H. vander Vorst, *Numerical Linear Algebra for High-Performance Computers*. Philadelphia, PA: SIAM, 1998.
44. R. E. Bank, T. Dupont, and H. Yserentant, The hierarchical basis multigrid method, *Numer. Math.*, **52**: 427–458, 1988.
45. J. H. Bramble, J. E. Pasciak, and J. Xu, Parallel multilevel preconditioners, *Math. Comput.*, **55**: 1–22, 1990.
46. M. D. Mihajlović and S. Ž. Mijalković, A component decomposition preconditioning for 3D stress analysis problems, *Numer. Linear Algebra Appl.*, **9**(6-7): 567–583, 2002.
47. C. E. Powell and D. J. Silvester, Optimal preconditioning for Raviart–Thomas mixed formulation of second-order elliptic problems, *SIAM J. Matrix Anal. Appl.*, **25**: 718–738, 2004.
48. I. Perugia, V. Simoncini, and M. Arioli, Linear algebra methods in a mixed approximation of magnetostatic problems, *SIAM J. Sci. Comput.*, **21**(3): 1085–1101, 1999.
49. G. Wempner, *Mechanics of Solids*. New York: McGraw-Hill, 1973.
50. M. R. Hanisch, Multigrid preconditioning for the biharmonic Dirichlet problem, *SIAM J. Numer. Anal.*, **30**: 184–214, 1993.
51. D. J. Silvester and M. D. Mihajlović, A black-box multigrid preconditioner for the biharmonic equation, *BIT*, **44**(1): 151–163, 2004.
52. B. Smith, P. Bjøstrad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge: Cambridge University Press, 2004.
53. A. Brandt, Multigrid solvers on parallel computers, in M. H. Schultz (ed.), *Elliptic Problem Solvers*. New York: Academic Press, 1981, pp. 39–83.
54. METIS, A family of multilevel partitioning algorithms. Available: http://glaros.dtc.umn.edu/gkhome/views/metis.
55. L. Hart and S. McCormick, Asynchronous multilevel adaptive methods for solving partial differential equations: Basic ideas, *Parallel Comput.*, **12**: 131–144, 1989.
56. B. Lee, S. McCormick, B. Philip, and D. Quinlan, Asynchronous fast adaptive composite-grid methods for elliptic problems: theoretical foundations, *SIAM J. Numer. Anal.*, **42**: 130–152, 2004.
57. A. Krechel and K. Stüben, Parallel algebraic multigrid based on subdomain blocking, *Parallel Comput.*, **27**: 1009–1031, 2001.
58. V. E. Henson and U. Meier Yang, BoomerAMG: A parallel algebraic multigrid solver and preconditioner, *Appl. Numer. Math.*, **41**: 155–177, 2002.
59. A. J. Cleary, R. D. Falgout, V. E. Henson, and J. E. Jones, Coarse-grid selection for parallel algebraic multigrid, *Lecture Notes in Computer Science*. New York: Springer, 1998, pp. 104–115.
60. D. M. Alber, Modifying CLJP to select grid hierarchies with lower operator complexities and better performance, *Numer. Linear Algebra Appl.*, **13**: 87–104, 2006.
61. H. De Sterck, U. Meier Yang, and J. J. Heys, Reducing complexity in parallel algebraic multi grid preconditioners, *SIAM J. Matrix Anal. Appl.*, **27**: 1019–1039, 2006.
62. C. C. Douglas, and M. B. Douglas, MGNet bibliography. Available: http://www.mgnet.org/bib/mgnet.bib. New Haven, CT: Yale University, Department of Computer Science, 1991-2002.
63. MGNet: A repository for multigrid and other methods. Available: http://www.mgnet.org/.
64. Copper mountain conference on multigrid methods. Available: http://amath.colorado.edu/faculty/copper/.
65. European conference on multigrid, multilevel and multiscale methods. Available: http://pcse.tudelft.nl/emg2005/
66. R. Wienands and W. Joppich, *Practical Fourier Analysis for Multigrid Methods*. Boca Raton: Chapman & Hall/CRC, 2005.

Slobodan Ž. Mijalković
Silvaco Technology Centre
Cambridge, United Kingdom
Milan D. Mihajlović
University of Manchester
Manchester, United Kingdom

P

POSETS AND LATTICES

PARTIALLY ORDERED SETS

In this section, we introduce the reader to the basic notion of partially ordered sets according to the following definition.

Definition 1. *A* partially ordered set *(poset, for short) is a structure $\langle \mathcal{P}, \leq \rangle$ where $\mathcal{P}$ is a nonempty set and $\leq$ is a* partial order relation *on $\mathcal{P}$,* i.e., *a binary relation such that for arbitrary a, b, c $\in \mathcal{P}$ the following conditions hold:*

(or1) $a \leq a$ (reflexive)
(or2) $a \leq b$ and $b \leq a$ imply $a = b$ (antisymmetric)
(or3) $a \leq b$ and $b \leq c$ imply $a \leq c$ (transitive)

If $a \leq b$, then we say that a is less than or equal to b, or that b is greater than or equal to a, also written in this case as $b \geq a$. Some other terminology frequently adopted in this case is that a precedes *or is* smaller *than b and that b* follows *or* dominates *or is* larger *than a.*

Example 1. Let us denote by $\mathcal{P}(X)$ the power set of a set (also universe) X, i.e., the collection of all subsets of X. The usual set theoretic inclusion on pairs A, B of subsets of X defined as "$A \subseteq B$ iff every element of A is also an element of B" (formally, "$a \in A$ implies a $\in B$") is a partial order relation on $\mathcal{P}(X)$. Thus $\langle \mathcal{P}(X), \subseteq \rangle$ is a poset.

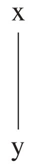
Example 2. Let 2^X denote the collection of all Boolean-valued functionals defined on the set X, i.e., the collection of all mappings $\chi : X \mapsto \{0, 1\}$. For any pair of such mappings $\chi_1, \chi_2 \in 2^X$, the binary relation defined as follows:

$$\chi_1 \leq \chi_2 \quad \text{iff} \quad \forall x \in X, \chi_1(x) \leq \chi_2(x) \tag{1}$$

is a partial order relation for 2^X, also called the pointwise order relation. Thus, $\langle 2^X, \leq \rangle$ is a poset.

Let us introduce some notions that can be derived in any poset structure. First of all, in a poset $\langle \mathcal{P}, \leq \rangle$ we can distinguish a *strict* partial ordering, written $<$, which means "x precedes y and x does not coincide with y"; formally, $x < y$ iff $x \leq y$ and $x \neq y$. For example, if $\leq$ is the set inclusion $\subseteq$, then $A < B$ means that A is a proper subset of B (there exists an element b in B that does not belong to A), i.e., $A \subset B$. Moreover, we say that *z covers y* iff $y < z$ and $y \leq x \leq z$ implies either $x = y$ or $x = z$ ($x < y$ and there is no element between x and y).

It is possible to give a graphical representation of a poset with a finite number of elements representing the fact that *x covers y* by drawing y below x and connecting them by a line segment according to the following figure:



The global figure so obtained is called the *Hasse diagram* of the finite poset.

Example 3. The Hasse diagram of a poset with five elements is presented in Fig. 1.

By the property of transitivity of a partial ordering, we read right from the diagram that $a \leq d$ because there exists the path $a \leq c$ and $c \leq d$ that moves steadily upward from a to d. Note that, in general, transitive relation can be run together without causing confusion: $x \leq y \leq z$ means $x \leq y$ and $y \leq z$, from which $x \leq z$.

Two elements a, b from a poset are called *comparable* iff one of the two cases, either $a \leq b$ or $b \leq a$, holds; two elements c, d that are not comparable are said to be *incomparable*. For example, in Fig. 1, the elements a and c are comparable, whereas a and b are not.

A poset $\mathcal{P}$ is a *totally* (or *linearly*) *ordered poset* iff $\forall\, x, y \in \mathcal{P}$ either $x \leq y$ or $y \leq x$, that is, iff any two elements a, b from the poset $\mathcal{P}$ are comparable. A totally ordered poset is also called *chain*. Examples of totally ordered posets are the set of all natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5 \ldots\}$ with the standard ordering, the set of all real numbers $\mathbb{R}$ and the unit interval $[0, 1] := \{x \in \mathbb{R} : 0 \leq x \leq 1\}$ with the usual order on real numbers, this latter being a bounded chain.

It is possible to define special elements in a poset $\mathcal{P}$ and so to introduce operations in such a structure. First, an element $0 \in \mathcal{P}$ is said to be the *least* (1) (or *minimum, first*, or *zero*) element in $\mathcal{P}$ iff $0 \leq a$ for every $a \in \mathcal{P}$; if the least element 0 exists, then it is unique. Similarly, if a poset contains an element $1 \in \mathcal{P}$ such that $a \leq 1$ for every element a in the poset, then this element is uniquely determined and will be called the *greatest* (1) (or *maximum, last*, or *unit*) element. The poset S5 of Fig. 1 has neither least nor greatest element.

Definition 2. *A partially ordered set with a minimum and a maximum element (also called* bounded poset*) is a*

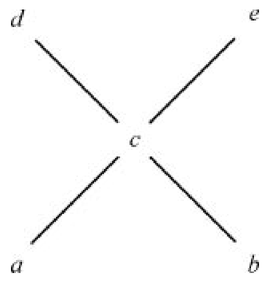


Figure 1. The poset S5.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

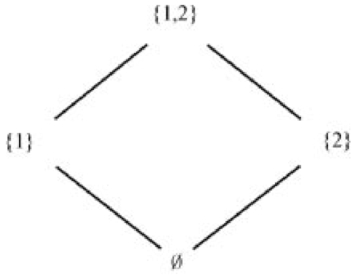


Figure 2. The atomic lattice 2^2 of all subsets of the universe {1,2}.

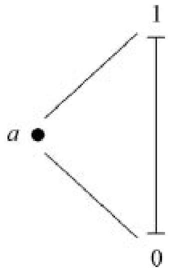


Figure 3. The nonatomic real unit interval plus an atom a.

structure $\langle \mathcal{P}, \leq, 0, 1\rangle$, *where* $\leq$ *is a partial order relation on* $\mathcal{P}$ *such that* 0 *and* 1 *are the* least (minimum) *and the* greatest (maximum) *element with respect to* $\leq$, *respectively. In other words, for all* $b \in \mathcal{P} : 0 \leq b \leq 1$.

In a poset with the least element $\langle \mathcal{P}, \leq, 0\rangle$, every element that covers the 0 element is called *atom*. We say that $\mathcal{P}$ is *atomic* iff for every $x \neq 0$ at least one atom a in $\mathcal{P}$ exists such that $a \leq x$ holds.

Example 4. The power set $\mathcal{P}(X)$ of a universe X is atomic; every singlet set $\{a\}$ for $a \in X$ is an atom of the poset. For instance, in the lattice 2^2 of all subsets of the universe that consist of two elements $X = \{1,2\}$, the singletons {1} and {2} are atoms (see Fig. 2).

Example 5. An example of poset without atoms is the real unit interval [0,1] endowed with the usual order on numbers. If to [0,1] we add an element $0 < a < 1$ such that a is incomparable with all elements in (0,1) (see the Hasse diagram of Fig. 3), we obtain a poset with an atom a that is nonatomic.

Let A be a subset of a poset $\langle \mathcal{P}, \leq\rangle$. Then the ordering $\leq$ of $\mathcal{P}$ induces an order on A in the following way: If $a, b \in A$, then $a \leq b$ as elements of A iff $a \leq b$ as elements of $\mathcal{P}$. In this case, we shall say that the pair $\langle A, \leq\rangle$ is a subposet of $\langle \mathcal{P}, \leq\rangle$. Note that a subposet of a poset $\mathcal{P}$ may be totally ordered, also if $\mathcal{P}$ is not. For instance, in the nonlinear poset represented by the diagram of Fig. 1, the two subposets {*b, c, e*} and {*a, d*} are totally ordered, i.e., chains, whereas the subposet {*a, b, c*} is not. Any way, a subposet of a chain is always a chain.

Definition 3. *Let* $\mathcal{S} \neq \emptyset$ *be a nonempty set, subset of the poset* $\mathcal{P}$.

i) An element $x \in \mathcal{P}$ *is said to be the* least upper bound *(l.u.b.) of* $\mathcal{S}$, *written* $x = sup(\mathcal{S})$(also $x = \vee \mathcal{S}$), *iff:*

a) $\forall\ s \in \mathcal{S}, s \leq x$

b) if $c \in \mathcal{P}$ satisfies $s \leq c$ for any $s \in S$, then $x \leq c$

If only condition i) holds, then the element s is said to be an upper bound *of S*.

ii) An element $y \in \mathcal{P}$ *is said to be the* greatest lower bound *(g.l.b.) of S, written* $y = in\ f(S)$ *(also* $y = \wedge S$*), iff:*

a) $\forall\ s \in S; y \leq s$

b) if $d \in \mathcal{P}$ satisfies $d \leq s$ for any $s \in S$, then $d \leq y$

If only condition i) holds, the element s is said to be a lower bound of $\mathcal{S}$.

Hence, an upper bound lies “above” every element of the set $\mathcal{S}$, whereas a lower bound lies “below” each element with respect to the involved partial ordering. The l.u.b. and the g.l.b. are the “smallest” upper bound and the “largest” lower bound of the set, respectively. If $\mathcal{S}$ is a finite set, say $\mathcal{S} = \{a_1, \ldots, a_n\}$, it is customary to write (if they exist) $\wedge_{i=1}^{n} a_i$ or $a_1 \wedge \ldots \wedge a_n$ and $\vee_{i=1}^{n} a_i$ or $a_1 \vee \ldots \vee a_n$ instead of $\wedge \mathcal{S}$ and $\vee \mathcal{S}$. In particular, we denote by $a \vee b$ and $a \wedge b$, if they exist, the l.u.b. and the g.l.b. of the pair $a, b \in \mathcal{P}$ respectively. Let us note that in the case of a bounded poset $\mathcal{P}$ we have that $x \vee 0 = x, x \wedge 0 = 0$ and $x \wedge 1 = x, x \wedge 1 = x$ for any element $x \in \mathcal{P}$.

Example 6. In Fig. 1, the set {*d, e*} has no l.u.b., whereas its g.l.b. is $c = d \wedge e$; the set {*a, b*} has g.l.b. $c = a \vee b$, but the l.u.b. of this set does not exist. Moreover, the whole poset has neither the minimum element 0 nor the maximum element 1.

Example 7. In the set of all integer numbers $\mathbb{Z}$, with the standard partial ordering, number 8 is an upper bound of the two subsets $\mathcal{S} = \{-2,0,1,3,4\}$ and $Q = \{-4,-1,2,4,8\}$, which is the l.u.b of Q but not of $\mathcal{S}$. Similarly, -4 is a lower bound of both $\mathcal{S}$ and Q, which is the g.l.b of Q.

In the poset of all real numbers $\mathbb{R}$, number 8 is an upper bound of the open intervals (–2,6) and (6,8) and also of the closed intervals $[-5, -2]$ and $[-1,8]$. Moreover, it is the l.u.b of both (6, 8) and $[-1,8]$; in this second case, 8 belongs to the interval, contrary to the former one. In an analogous way, -5 is a lower bound of all these intervals, which is the g.l.b. of $[-5,-2]$.

There cannot be a number $\underline{\vee}$(resp., $\overline{\wedge}$) that is an upper (resp., lower) bound of the whole set $\mathbb{Z}$ of all integer numbers, or of all real numbers $\mathbb{R}$ for that matter, Because $\underline{\vee}$ (resp., $\overline{\wedge}$) is evidently strictly less (resp., great) than the number $\underline{\vee} + 1$ (resp., $\overline{\wedge} - 1$).

In the following proposition, for a pair of subsets Y_1 and Y_2 of a poset $\mathcal{P}$ we adopt the convention to denote by $Y_1 \leq Y_2$ the fact that $y_1 \leq y_2$ for every $y_1 \in Y_1$ and every $y_2 \in Y_2$.

Proposition 1. *Let* $\langle \mathcal{P}, \leq\rangle$ *be a poset.*

1) Let Y be a subset of $\mathcal{P}$; *then the elements* sup(Y) *and* inf(Y), *if they exist, are unique.*

2) Let Y_1 *and* Y_2 *be subsets of* $\mathcal{P}$ *such that* $Y_1 \leq Y_2$ *and both* $\inf(Y_1)$ *and* $\sup(Y_2)$ *exist; then* $\inf(Y_1) \leq \sup(Y_2)$.

3) $x \leq y$ iff $x \wedge y = x$ and $x \vee y = y$

Proof 1. Let $x_1 = \sup(Y)$ and $x_2 = \sup(Y)$. Then from (i-b) of Definition 3 applied to x_1, we have in particular that $y \le h$ for any $y \in Y$ implies $x_1 \le h$. On the other hand, from (i-a) of the same definition applied to x_2, we have that $y \le x_2$ for any $y \in Y$; hence $x_1 \le x_2$. Reversing the role of these two elements, we arrive at the result that $x_2 \le x_1$ and so $x_1 = x_1$.

A similar proof can be given in the case of inf(Y). The proofs of points 2 and 3 are trivial.

Proposition 2. *A poset $\mathcal{P}$ that has the l.u.b. $\vee\mathcal{P}$ and the g.l.b. $\wedge\mathcal{P}$ is* bounded; *indeed*, $\wedge\mathcal{P} \le x \le \vee\mathcal{P}$ *for every* $x \in \mathcal{P}$.

Given the poset $\langle \mathcal{P}, \le \rangle$, it is possible to construct the binary relation $\le_c$ on $\mathcal{P}$ defined as $x \le_c y$ iff $y \le x$, which trivially is itself a partial ordering on $\mathcal{P}$ called the *converse* of $\le$.

Other interesting notions involving posets are the following.

Definition 4. *A* meet *(resp., join)-semilattice is a poset for which the g.l.b.* $a \wedge b$ *(resp., l.u.b.* $a \vee b$*)exists for any pair a, b of elements from the poset.*

Example 8. Let us consider a finite set $\mathcal{A} = \{a_1, a_2. \ldots, a_n\}$, the *alphabet* of this example, whose elements a_i are symbols, also called *letters*. A *word* of the alphabet $\mathcal{A}$ is any finite sequence (also called *string*) of elements from $\mathcal{A}$, whose collection will be denoted by $\mathcal{A}^*$. The string with no letters at all, denoted by $\in$, is a special word of $\mathcal{A}^*$ called the *empty* or *null word*. The length of a word $w \in \mathcal{A}^*$, denoted by $|w|$, is the number of letters forming w, counting each appearance of a letter; in particular $|\in| = 0$. For instance, in the case of a Boolean alphabet $\mathcal{A} = \{0,1\}$, the finite sequence 0010101 is a word, i.e., an element from $\{0,1\}^*$, of length 7.

A standard inner operation on $\mathcal{A}^*$ is the so-called *concatenation* (also *juxtaposition*): Let $u = u_1u_2 \ldots u_h$ and $w = w_1w_2 \ldots w_k$ be two words in $\mathcal{A}^*$ (of length h and k, respectively); then one can construct the new word $uw = u_1u_2 \ldots u_hw_1w_2 \ldots w_k$, which is another element of $\mathcal{A}^*$ (of length $h + k$). Concatenation with the empty word leaves the word unchanged: $w\epsilon = \epsilon w =$ w whatever be $w \in \mathcal{A}^*$. In the case of a nontrivial alphabet (i.e., containing two or more letters), this operation is not commutative; for instance, in $\{0,1\}^*$ the two words $u = 00$ and $w = 100$ are such that $uw = 00100$ and $wu = 10000$, which are different: $uw \ne wu$.

On $\mathcal{A}^*$, the binary relation $u \le w$ means that the word $u \in \mathcal{A}^*$ is the initial segment of the word $w \in \mathcal{A}^*$, formally that there exists a word $v \in \mathcal{A}^*$ such that $uv = w$; in this case, u is said to be a *prefix* of w. For instance, in the case of the Boolean alphabet, the two words 010 and 010011 are comparable, with $010 \le 010011$, because there exists the word 011 such that 010011 = (010)(011). The poset $\{0,1\}^*$ has the empty word $\in$ as its minimum element; indeed, for any word $w \in \mathcal{A}^*$ the property $\in w = w$ means that $\in \le w$. Moreover, it is a meet-semilattice; for instance, with respect to the meet, we have $110 \wedge 10 = 1$, $010 \wedge 1 = \in$, but the join of the words 10 and 01 does not exists because there is no upper bound w such that $10 \le w$ and $01 \le w$, i.e., two words u_1 and u_2 such that $10u_1 = 01u_2 = w$.

From the computational point of view, strings of an alphabet, for instance, the Boolean one, "may be thought of as information encoded in binary form: the longer the string the greater the information content. Further, given any string v, we may think of elements u with $u < v$ as providing approximations to v" 2. Extending in an obvious way these considerations to the collection $\mathcal{A}^{**}$ of all finite and infinite sequences of letters from the alphabet $\mathcal{A}$, "any infinite string is, in an [intuitive] sense, the limit of its finite initial substrings" (2).

LATTICES

In the following, if they exist, we shall denote by $\vee a_j$ and $\wedge a_j$ the l.u.b. and the g.l.b. of any family $\{a_j : j \in J\}$ of elements from a poset $\mathcal{P}$, respectively.

Definition 5. *A* lattice *is a poset* $\langle \mathcal{L}, \le \rangle$ *such that both* $x \wedge y := in\ f\{x,y\}$ *(the lattice* meet *or* conjunction*)* $andx \vee y := sup\{x,y\}$ *(the lattice* join *or* disjunction*) exist for every pair of elements x, y* $\in \mathcal{L}$. *To be precise, the lattice meet satisfies the conditions*

$$\begin{array}{lll} \text{(LM1)} & a \wedge b \le \{a,b\} & \\ \text{(LM2)} & x \le \{a,b\} & \text{implies} \quad x \le a \wedge b \end{array}$$

whereas the lattice join satisfies the conditions

$$\begin{array}{lll} \text{(LJ1)} & \{a,b\} \le a \vee b & \\ \text{(LJ2)} & \{a,b\} \le x & \text{implies} \quad a \vee b \le x \end{array}$$

Example 9. Let $\mathcal{P}(X)$ be the poset of all subsets of a given universe X with respect to the usual set theoretic inclusion relation $\subseteq$. A subset A of the universe X can be defined by a property $P(x)$ characterizing just the elements of the universe that satisfies this property, and, in this case, we write $A = \{x \in X : P(x)\}$ where the symbol ":" has the meaning of *such that*. If the element x belongs to the set A, then we write $x \in A$.

If the universe is finite, then a set can be given by listing all its elements: $A = \{x_1, x_2, \ldots, x_n\}$. For instance, in the universe $[0, 10]_{\mathbb{N}}$ that consist of all integer numbers between 0 and 10, the set of even numbers less than 10 can be specified as $\{x$: x is integer, $x < 10$, $x \bmod 2 = 0\}$ or equivalently as {2, 4, 6, 8}.

A special set is the *empty set*, which is a set with no elements, denoted by the symbol $\emptyset$. On the opposite site is the *universe* X, which is the set of all the elements of a given domain of interest. With respect to the poset $\langle \mathcal{P}(X), \subseteq \rangle$ of all subsets of a universe X, the set $\emptyset$ is the least element and X the greatest one:

$$\forall A \in \mathcal{P}(X) : \emptyset \subseteq A \subseteq X$$

Given two subsets $A = \{a \in X : P(a)\}$ and $B = \{b \in X : Q(b)\}$ of X defined by the properties P and Q, respectively, their set intersection and union are given by:

$$\begin{array}{c} A \cap B = \{x \in X : P(x) \text{ and } Q(x)\} = \{x \in X : x \in A \text{ and } x \in B\} \\ A \cup B = \{y \in X : P(y) \text{ or } Q(y)\} = \{y \in X : y \in A \text{ or } y \in B\} \end{array}$$

It is easy to verify that the poset $\langle \mathcal{P}(X), \subseteq \emptyset, X\rangle$ is a lattice because inf{*A, B*} and sup{*A, B*} coincide with their set intersection and union, respectively:

$$A \wedge B = A \cap B \quad \text{and} \quad A \vee B = A \cup B \tag{2}$$

Indeed, the set union $A \cup B$ contains both A and B; hence, it is an upper bound of A and B. However, every subset C of X that contains both A and B (that is, every upper bound of A and B) contains their union too; thus, $A \cup B$ is the smallest upper bound of A and B. We apply the same procedure to the intersection.

To stress the set operations of intersection $\cap$ and union $\cup$ as lattice meet and join, respectively, the power set-bounded lattice based on the universe X will be denoted in the sequel as the system $\langle \mathcal{P}(X), \cap, \cup, \emptyset, X\rangle$.

Remark 1. The formal definition of set union (resp., intersection) of two elements A and B of $\mathcal{P}(X)$ [as collection of all points from X that belong to at least (resp., both) the sets A and B] is very different from the formal definition of least upper (resp., greatest lower) bound of the same elements according to Definition 3. We have just proved that these two formal definitions coincide on $\mathcal{P}(X)$ in the sense that they produce the same subsets of X.

Example 10. The poset 2^X of all Boolean-valued functionals on X is a lattice where the meet and join of any pair of such functionals χ_1, χ_2 are defined for every point $x \in X$ as follows:

$$(\chi_1 \wedge \chi_2)(x) = \inf\{\chi_1(x), \chi_2(x)\} \quad \text{and} \quad (\chi_1 \vee \chi_2)(x) = \sup\{\chi_1(x), \chi_2(x)\} \tag{3}$$

Now, the following result is very important because it assures that the operators of meet $\wedge$ and join $\vee$ of a lattice $\mathcal{L}$ are interchangeable, i.e., it validates the dualization of all theorems with respect to $\mathcal{L}$ in the sense that the join may be replaced by the meet, and conversely.

Proposition 3 (Duality Principle). *Let us consider a lattice* $\langle \mathcal{L}, \leq\rangle$, *with associated converse lattice* $\langle \mathcal{L}, \leq_c\rangle$. Then, for any pair *x,y* $\in \mathcal{L}$ *the following holds:*

(ia) If $x \wedge y$ is the lattice meet of the pair x, y with respect to the ordering $\leq$, then $x \wedge y$ is the lattice join of the same pair with respect to the converse *ordering $\leq_c$; formally, $x \vee_c y$ exists and $x \vee_c y = x \wedge y$.*

(ib) If $x \vee y$ is the lattice join of the pair x, y with respect to the ordering $\leq$, then $x \vee y$ is the lattice meet of the same pair with respect to the converse *ordering $\leq_c$; formally, $x \wedge_c y$ exists and $x \wedge_c y = x \vee y$.*

(iia) If $x \wedge_c y$ is the lattice meet of the pair x,y with respect to the ordering $\leq_c$, then $x \wedge_c y$ is the lattice join of the same pair with respect to the converse *ordering $\leq$; formally, $x \vee y$ exists and $x \vee y = x \wedge_c y$.*

(iib) If $x \wedge_c y$ is the lattice join of the pair x, y with respect to the ordering $\leq_c$, then $x \vee_c y$ is the lattice meet of the same pair with respect to the converse *ordering $\leq$; formally, $x \wedge y$ exists and $x \wedge y = x \wedge_c y$.*

The results (ia) and (iib) can be summarized in the unique formulation:

$$x \wedge y \quad \text{under} \quad \leq \quad \text{iff} \quad x \vee y \quad \text{under} \quad \leq_c$$

Similarly, the results (ib) and (iia) can be summarized in the unique formulation:

$$x \vee y \quad \text{under} \quad \leq \quad \text{iff} \quad x \wedge y \quad \text{under} \quad \leq_c$$

Proof. Let us set $h = x \wedge y$. Then, the condition $h \leq \{x, y\}$ (lower bound of the pair *x, y*) translates in the "converse" condition $\{x, y\} \leq_c h$ (upper bound of the same pair). Moreover, the fact that $d \leq \{x,y\}$ implies $d \leq h$ (l.u.b. of *x, y*) leads to the fact that $\{x, y\} \leq_c d$ implies $h \leq_c d$. So we have proved that if $x \wedge y$ is the lattice meet of the pair *x, y* with respect to the ordering $\leq$, then the same element $x \wedge y$ is the lattice join of the same pair with respect to the ordering $\leq_c$; formally, $x \vee_c y$ exists and $x \vee_c y = x \wedge y$.

The case of the join (ib), and the corresponding dual, can be proved likewise.

Definition 6. *An* σ*-lattice is a poset for which the* sup *and* inf *exist for any countable collection of its elements.*

Example 11. Let us recall that a *measurable space* is a pair $(X, \mathcal{A}(X))$ consisting of a nonempty set X and a family $\mathcal{A}(X)$ of its subsets, called *measurable sets*, that satisfy the following conditions:

(M1) The empty set is measurable: $\emptyset \in \mathcal{A}(x)$.

(M2) The set union of countable measurable sets is measurable: Let $M_n \in \mathcal{A}(x)$: $n \in \mathbb{N}$, then $\cup_{n \in \mathbb{N}} M_n \in \mathcal{A}(x)$.

(M3) The set complement of a measurable set is measurable: $M^c \in \mathcal{A}(X)$ for every $M \in \mathcal{A}(X)$.

The σ-algebra $(\mathcal{A}(X), \subseteq)$ equipped with the set inclusion is an σ-lattice, bounded by the least element $\emptyset \in \mathcal{A}(X)$ and the greatest one $X = \emptyset^c \in \mathcal{A}(X)$. Indeed, for any countable family of measurable sets, from (M2) $\vee M_n = \cup M_n$ follows, and because $\cap M_n = (\cup (M_n)^c)^c \in \mathcal{A}(X)$ from (M2) and (M3), also $\wedge M_n = \cap M_n$. Note that in this example, the meet is the set intersection and the join the set union.

Definition 7. *A lattice $\mathcal{L}$ is* complete *iff both* inf(*S*) *and* sup(*S*) *exist for every nonempty subset, i.e., any $S \subseteq \mathcal{L}$ with $S \neq \emptyset$.*

Every finite lattice is obviously complete. Every complete lattice $\mathcal{L}$ is bounded because 0:= inf($\mathcal{L}$) is the least element and 1 := sup($\mathcal{L}$) is the greatest one.

Example 12. Given a universe X, the two lattices $\mathcal{P}(X)$ (of all subsets of X) and 2^X (of all Boolean-valued functionals defined on X) are complete lattices because the operations introduced in Equations (2) and (3) can be extended to arbitrary families.

Example 13. Let us recall that a *topological space* is a pair *(X,C(X))* that consists of a set X and a family *C(X)* of its subsets, called *closed sets*, which satisfies the following conditions:

(T1) The empty set and the whole space X are closed $\emptyset, X \in$ *C(X)*.

(T2) The set intersection of *any* family of closed sets is closed: Let $C_i \in C(X)$, with i running on an arbitrary index set I, then $\cap_{i \in I} C_i \in C(X)$.

(T3) The *finite* union of closed sets is closed: $\cup_{j=1,\dots k} C_k \in C(X)$ for any finite collection $C_1, \dots, C_k$ of closed sets.

Then *(C(X)*, $\subseteq$), being $\subseteq$ the usual set inclusion, is a complete lattice in which the meet is the set intersection $(\wedge C_i = \cap_i C_i)$, whereas the join is the closure of the set union $[\vee C_j = \overline{\cup_j C_j}$, obtained as the set intersection of all closed sets containing $\cup C_j$, formally $\vee C_j = \cap \{K \in C(X) : \cup C_j \subseteq K\}$, which is closed owing to (T2)]. Therefore, if we exclude the case of a finite number of closed sets, then the join is, in general, different from the set union.

For instance, if one considers the real line $\mathbb{R}$. with the standard Euclidean topology, then the sequence of closed sets $I_n = [-\frac{1}{n}, 1 + \frac{1}{n}]$ (for $n \in \mathbb{N}$) has the set union

$$\bigcup_{n \in N} I_n = (0,1)$$

which is an open set whose closure is the interval [0,1], the intersection of all closed sets containing the union $\cup I_n =$ (0,1):

$$\bigvee_{n \in \mathbb{N}} I_n = [0,1] \neq (0,1) = \bigcup_{n \in \mathbb{N}} I_n$$

This is not the unique situation in which some lattice operation does not coincide with the set theoretic ones.

Example 14. Let $\mathcal{V}(X) = \langle X, +, \mathbb{K} \rangle$ be a *linear space*, i.e., a nonempty set of *vectors* containing a privileged element $\underline{0}$ and equipped with a binary operation of sum $+ : X \times X \to X$ (of abelian group with $\underline{0}$ as the zero vector) and an external operation:$\mathbb{K} \times X \to X$ of multiplication of *scalars* from a field $\mathbb{K}$ (usually $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$) with vectors from X to produce a new vector in X.

A *linear subspace*, simply subspace in the following, is a nonempty subset M of X (in general $\underline{0} \in M$), which is closed with respect to the inner sum (for any pair of vectors *x, y* $\in$ *M*, it is $x+y \in M$) and the external product of scalars with vectors ($\alpha \in \mathbb{K}$ and $x \in M$ imply $\alpha \cdot x \in M$). The collection of all subspaces of X will be denoted by $\mathcal{M}(X)$, which is a poset with respect to the set inclusion, bounded by the least element $\{\underline{0}\}$, the subspace consisting of the unique zero vector, and the greatest element X, the whole space; formally, for any subspace $M \in \mathcal{M}(X)$, it is $\{\underline{0}\} \subseteq M \subseteq X$ with both $\{\underline{0}\}, X \in \mathcal{M}(X)$. Then it is easy to show that the set intersection of *any* family of subspaces is a subspace too:

$$\bigcap_{i \in I} M_i \in \mathcal{M}(X) \quad \text{for any family of subspaces} \quad M_i \in \mathcal{M}(X)$$

In this way, the lattice meet of any family of subspaces is their set intersection:

$$\bigwedge_{i \in I} M_i = \bigcap_{i \in I} M_i$$

whereas, the lattice join of the same family is the subspace *generated* by the set union, which is the intersection of all subspaces that contain the union $\cup M_i$:

$$\bigvee_{i \in I} M_i = \bigcap \{H \in \mathcal{M}(X) : \bigcup_{i \in I} M_i \subseteq H\}$$

For instance, in the linear space $\mathbb{R}^2$, the two subspaces $M_x = \{(x,0) \in \mathbb{R}^2 : x \in \mathbb{R}\}$ and $M_{\pi/4} = \{(x,x) \in \mathbb{R}^2 : x \in \mathbb{R}\}$ are both (one-dimensional) subspaces for which $M_x \wedge M_{\pi/4} = M_x \cap M_{\pi/4} = \{\underline{0}\}$ and $M_x \vee M_{\pi/4} = \mathbb{R}^2$ (because $\mathbb{R}^2$ is the unique subspace that contains both M_x and $M_{\pi/4}$), this latter being strictly greater than the set union of these two subspaces.

If one represents any one-dimensional subspace of $\mathbb{R}^2$ by the angle $\alpha \in [0, \pi)$ between this subspace and the x-axis, written as M_α, then the complete lattice $\mathcal{M}(\mathbb{R}^2)$ can be represented by the Hasse diagram of the right side of Fig. 4, where it is clear that for any pair of different one-dimensional subspaces M_α and M_β, it is $M_\alpha \wedge M_\beta = \{\underline{0}\}$ and $M_\alpha \vee M_\beta = \mathbb{R}^2$.

Note that in this lattice, the distributivity conditions of the two lattice meet and join operations do not hold. For instance, if one considers two different one-dimensional subspaces $\alpha \neq \beta$, which are both different from the x-axis, too, then $M_\alpha \wedge (M_\beta \vee M_x) = M_\alpha \wedge \mathbb{R}^2 = M_\alpha$, whereas $(M_\alpha \wedge M_\beta) \vee (M_\alpha \wedge M_x) = \{\underline{0}\} \vee \{\underline{0}\} = \{\underline{0}\}$. Similarly, $M_\alpha \vee (M_\beta \wedge M_x) = M_\alpha \vee \{\underline{0}\} = M_\alpha$, whereas $(M_\alpha \vee M_\beta) \wedge (M_\alpha \vee M_x) = \mathbb{R}^2 \wedge \mathbb{R}^2 = \mathbb{R}^2$.

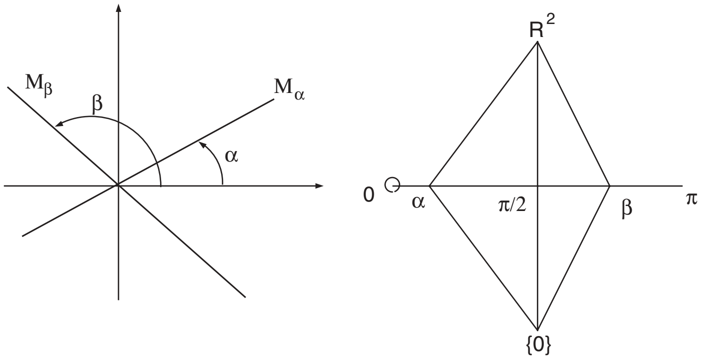


Figure 4. The two-dimensional linear space $\mathbb{R}^2$ with two one-dimensional subspaces M_α, M_β (left) and the corresponding Hasse diagram representation of the (non-distributive) lattice (right).

Lattices and Related Algebras

As we have seen before, a partially ordered set $\langle \mathcal{L}, \leq \rangle$ is a lattice iff both the g.l.b. and the l.u.b exist for every pair of elements x, $y \in \mathcal{L}$ [and are unique according to point 1 of Proposition 1 applied to the subset $Y = \{x,y\}$]. Because of their properties of existence and uniqueness, we can think of inf(x, y) and sup(x, y), with x and y running on the lattice $\mathcal{L}$, as the results of two binary operations on $\mathcal{L}$. In this way, a lattice can be considered as a particular algebraic structure $\langle \mathcal{L}, \wedge, \vee \rangle$, based on a set $\mathcal{L}$ equipped with two inner composition laws.

$$\wedge : \mathcal{L} \times \mathcal{L} \mapsto \mathcal{L}, (x,y) \to x \wedge y := \inf\{x,y\} \qquad (4)$$

$$\vee : \mathcal{L} \times \mathcal{L} \mapsto \mathcal{L}, (x,y) \to x \vee y := \sup\{x,y\} \qquad (5)$$

It is possible to make a link between the lattice $\langle \mathcal{L}, \leq \rangle$ and the algebra $\langle \mathcal{L}, \wedge, \vee \rangle$, whose proof can be found in Ref. 1 on page 8.

Theorem 1. *Let$\langle \mathcal{L}, \leq \rangle$be a lattice. Then, the algebraic structure$\langle \mathcal{L}, \wedge, \vee \rangle$, where $\wedge$ and $\vee$ are the binary operations on $\mathcal{L}$ of meet in Equation*(4) *and join in Equation*(5), *satisfies the following laws for every x, y, z* $\in \mathcal{L}$:

$(L1)$ $x \wedge x = x,$ and $x \vee x = x$ (idempotence)

$(L2)$ $x \wedge y = y \wedge x$ and $x \vee y = y \vee x$ (commutativity)

$(L3)$ $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ and $x \vee (y \vee z)$
$= (x \vee y) \vee z$ (associativity)

$(L4)$ $x \wedge (x \vee y) = x$ and $x \vee (x \wedge y) = x$ (absorpton)

Moreover the two consistence properties hold:

$(L5a)$ $x \leq y$ is equivalent to $x = x \wedge y$

$(L5b)$ $x \leq y$ is equivalent to $y = x \vee y$

Let us notice that according to this result, the algebraic version $\langle \mathcal{L}, \wedge, \vee \rangle$ of the lattice is defined in an entirely *equational* way. Moreover, the peculiar form of the properties (L1)–(L4) gives a corroboration of the duality principle expressed in Proposition 3 because if we consider one of the properties that defines a lattice and we replace each symbol $\wedge$ by $\vee$ and each $\vee$ with $\wedge$, then we obtain the dual property. Thus, it follows that any theorem, result, or identity that holds for a lattice will remain true if we dualize it, that is, if we replace each $\wedge$ by $\vee$ and each $\vee$ by $\wedge$.

We recall the following result whose proof can be found in Ref. 1 on page 22.

Theorem 2. *Postulates (L2)–(L4) for lattices imply (L1), but the six identities of (L2)–(L4) are independent.*

Properties (L2)–(L4) characterize lattices completely, in a sense expressed by the following theorem.

Theorem 3. *Let* $\langle \mathcal{L}, \wedge, \vee \rangle$ *be an algebraic structure where:*

i) $\mathcal{L}$ *is a nonempty set,*

ii) $\wedge$ *and* $\vee$ *are two binary operations on* $\mathcal{L}$ *satisfying properties (L2)–(L4).*

If we define the following binary relation on $\mathcal{L}$:

$$x \leq y \quad \text{iff} \quad y = x \vee y \quad (\text{equivalently, iff} \quad x = x \wedge y) \qquad (6)$$

then $\langle \mathcal{L}, \leq \rangle$ *is a lattice such that*

$$x \vee y = \sup\{x,y\} \quad \text{and} \quad x \wedge y = \inf\{x,y\}$$

Moreover, the following result assures the "categorical equivalence" between the two versions of the lattice notions, the poset one and the algebraic one.

Proposition 4. *Starting with a poset version of a lattice* $\langle \mathcal{L}, \leq \rangle$, *forming its algebraic version* $\langle \mathcal{L}, \wedge, \vee \rangle$, *and then forming its corresponding poset version, denoted by* $\langle \mathcal{L}, \preceq \rangle$, *then this latter gives us back our original poset structure, i.e.,* $\preceq$ *is the partial order* $\leq$ *we started with.*

Vice versa, starting from an algebraic version of a lattice $\langle \mathcal{L}, \wedge, \vee \rangle$, *forming its poset version* $\langle \mathcal{L}, \leq \rangle$, *and then forming its corresponding algebraic version, denoted by* $\langle \mathcal{L}, \sqcap, \vee \rangle$, *then one obtains the original algebraic structure, i.e.,* $\wedge = \sqcap$ *and* $\vee = \sqcup$.

Until now we have considered *inf* and *sup* as two binary operations holding between two generic elements of a set. It is time for us to ask if we can find neutral elements for these operations.

Proposition 5. *Let*$\langle \mathcal{L}, \wedge, \vee \rangle$*be the algebraic formulation of a lattice.*

i) If a neutral element exists for the binary operation $\vee$, *called the algebraic* zero, *then this element is unique and turns out to be the minimum element* 0 *with respect to partial ordering in Equation* (6) *induced from the algebra.*

ii) If the neutral element exists for the binary operation $\wedge$, *called the algebraic* unity, *then this element is unique and turns out to be the maximum element* 1 *with respect to partial ordering in Equation* (6) *induced from the algebra.*

Proof. By definition of $\vee$–neutral element we have that $\forall x \in \mathcal{L} : x \vee 0 = x$, that is, $0 \leq x$ for every element x, according to Equation (6). This result means that if the zero element exists in the algebraic version of a lattice $\langle \mathcal{L}, \wedge, \vee \rangle$, then it is the minimum element relative to the induced poset $\langle \mathcal{L}, \leq \rangle$, and it is unique at the same time.

On the other side, always according to Equation (6), $x \wedge 1 = x$ for every $x \in \mathcal{L}$ is equivalent to $x \leq 1$; so if the algebraic unity 1 exists in $\mathcal{L}$, then this unity is unique and is the maximum element in the induced poset structure $\langle \mathcal{L}, \leq \rangle$.

The case of a bounded lattice has an interesting peculiar algebraic characterization, at least in terms of the join operation.

Proposition 6. *Let* $\langle \mathcal{L}, \leq, 0 \rangle$ *be a bounded lattice; then, the binary operation of join* $\vee$ *satisfies the equations for all x, y, and z*:

$(L1)$ $x \vee x = x$ (idempotence)
$(L2)$ $x \vee y = y \vee x$ (commutativity)
$(L3)$ $x \vee (y \vee z) = (x \vee y) \vee z$ (associativity)
$(L0)$ $x \vee 0 = x$ (unit element)

Briefly, the structure $\langle \mathcal{L}, \vee, 0 \rangle$ *is a* commutative monoid *(semigroup with unity) in which any element is idempotent.*

Conversely, we have that if $\langle \mathcal{L}, \vee, 0 \rangle$ *is a commutative monoid in which every element is idempotent, then there exists a unique partial ordering on* $\mathcal{L}$ *such that* $x \vee y$ *is the lattice join of any pair x and y, and* 0 *is the least element.*

This situation means that this structure is a join-semilattice, and so a semilattice can be defined either in terms of the order relation or in terms of join operation.

Definition 8. *Two lattices* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *are said to be* isomorphic *if there exists a bijective mapping of* $\mathcal{L}_1$ *into* $\mathcal{L}_2$, *say* ϕ: $\mathcal{L}_1 \mapsto \mathcal{L}_2$, such that for arbitrary $x, y \in \mathcal{L}_1$

$$\phi(x \wedge y) = \phi(x) \wedge \phi(y) \quad \text{and} \quad \phi(x \vee y) = \phi(x) \vee \phi(y) \qquad (7)$$

Of course, if $\phi : \mathcal{L}_1 \mapsto \mathcal{L}_2$ is a lattice isomorphism, then its inverse $\phi^{-1} : \mathcal{L}_2 \mapsto \mathcal{L}_1$ is a lattice isomorphism, too. Trivially, a lattice isomorphism preserves the ordering, i.e., it is a poset isomorphism:

$$\begin{array}{lll} x \leq y & \text{implies} & \phi(x) \leq \phi(y) \\ \phi(x) \leq \phi(y) & \text{implies} & x \leq y \end{array}$$

Indeed, $x \leq y$ is equivalent to $x = a \wedge y$, and so $\phi(x) = \phi(x) \wedge \phi(y)$, which means that $\phi(x) \leq \phi(y)$. The second condition can be proved in a similar, if a little bit complicated, way.

Proposition 7. *If two bounded lattices are* isomorphic, *then the additional conditions are satisfied*:

$$\phi(0) = 0 \quad \text{and} \quad \phi(1) = 1$$

Proof. If we apply the first part of Equation (7) to the particular case $y = 0$, then we obtain that for every x it is $\phi(0) = \phi(x) \wedge \phi(0)$, i.e., $\phi(0) \leq \phi(x)$. Because this inequality holds for every element x of the lattice $\mathcal{L}_1$, let us denote by $x_0 = \phi^{-1}(0)$ the inverse imagine of the $0 \in \mathcal{L}_2$; then, we have that $\phi(0) \leq 0$, i.e., $\phi(0) = 0$. The second statement can be proved analogously.

Example 15. Let us consider the two bounded lattices $\mathcal{P}(X)$ and 2^X of Examples 1 and 2, respectively. The lattice structure of the power lattice $\mathcal{P}(X)$ of the universe X has been widely discussed in Example 9. Now, let us consider the mapping

$$\chi : \mathcal{P}(X) \mapsto 2^X$$

associating with any subset A of X its characteristic functional $\chi_A : X \mapsto \{0, 1\}$, i.e., the Boolean-valued functional defined as $\chi_A(x) = 1$ if $x \in A$ and $= 0$ otherwise.

This mapping is a bijection because *(1)* it is injective: $A \neq B$ trivially implies that $\chi_A \neq \chi_B$, and *(2)* it is surjective because for any $\psi : X \mapsto \{0, 1\}$ in 2^X, constructed the subset $A(\psi) := \{x \in X : \psi(x) = 1\}$, the corresponding characteristic functional is such that $\chi_{A(\psi)} = \psi$.

With respect to the meet and join operations, recalling that the lattice meet and join of two characteristic functionals $\chi_A, \chi_B \in 2^X$ with respect to the partial ordering in Equation (1) both exist (Example 10), we have the following correspondences whatever be A, $B \in \mathcal{P}(X)$:

$$\begin{array}{l} A \cap B \xrightarrow{\chi} \chi_{A \cap B} = \chi_A \wedge \chi_B \\ A \cup B \xrightarrow{\chi} \chi_{A \cup B} = \chi_A \vee \chi_B \end{array}$$

i.e., χ is a lattice isomorphism, which allows one to identify the subset A of the universe X with its characteristic functional χ_A. Let us note that if one interprets the Boolean values 0 and 1 as two *membership degrees*, with 1 (resp., 0) the membership certainty (resp., impossibility), in fuzzy set theory the characteristic functional χ_A is considered as the *crisp* representation of the set A (see Refs. 3 and 4. In this context, $\chi_\emptyset$ is the identical zero function, also denoted by 0, that assigns to any x the value 0; similarly, χ_X is the identical one function, also denoted by 1, that assigns to any x the value 1. Of course, the lattice 2^X is bounded because for any $\chi \in 2^X$, it is $0 \leq \chi \leq 1$.

DISTRIBUTIVE AND MODULAR LATTICES

In the context of lattices, we can distinguish at least two different classes: *distributive lattices* and *modular lattices*.

Definition 9. *A lattice* $\mathcal{L}$ *is said to be* distributive *if and only if the following equalities hold:*

(d1) for every $x, y, z \in \mathcal{L}, x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
(d2) for every $x, y, z \in \mathcal{L}, x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

Remark 2. As remarked in Ref. 1 on page 11, conditions (d1) and (d2) are mutually equivalent, and so only one of these equalities is enough to define a distributive lattice [either (d1) or (d2)]. Notice that in the proof of this equivalence property, a determinant role is played by the universal quantifier in the sense that these two conditions are equivalent if they hold for *any* triple of elements from the lattice $(\forall x, y, z \in \mathcal{L})$.

If the lattice is not distributive, then there might be particular triples of individual elements $(\exists x_0, y_0, z_0 \in \mathcal{L})$ such that $x_0 \vee (y_0 \wedge z_0) = (x_0 \vee y_0) \wedge (x_0 \vee z_0)$, but with respect to which $x_0 \wedge (y_0 \vee z_0) \neq (x_0 \wedge y_0) \vee (x_0 \wedge z_0)$, and vice versa.

Example 16. The lattice N5 shown in Fig. 5 is not distributive.

Indeed, one has that $b \wedge (a \vee c) = (b \wedge a) \vee (b \wedge c) = b$, i.e., the (ordered) triple *(b, a, c)* is (d2)-distributive, but $b \vee (a \wedge c) = b \neq a = (b \vee a) \wedge (b \vee c)$, i.e., this triple is not (d1)-distributive.

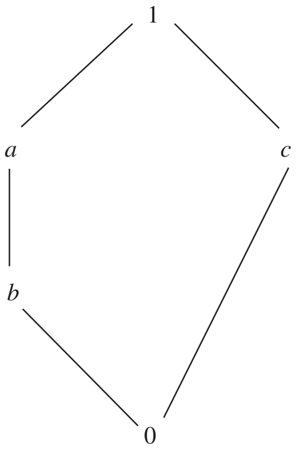


Figure 5. The nondistributive lattice N5.

In Ref. 5, one can find the following result, which is the distributive analogous of Theorem 3 that links lattices and algebraic structures.

Theorem 4. *Let $\langle \mathcal{L}, \wedge, \vee \rangle$ be an algebraic structure where:*

1) *$\mathcal{L}$ is a nonempty set, and*
2) *$\wedge$ and $\vee$ are two binary operations on $\mathcal{L}$ satisfying properties:*
   (DL1) $x = x \wedge (x \vee y)$
   (DL2) $x \wedge (y \vee z) = (z \wedge x) \vee (y \wedge x)$

Then, $\mathcal{L}$ is a distributive lattice.

Definition 10. *A lattice $\mathcal{L}$ is said to be* modular *if and only if the following equality holds:*

(m1) for every *x,y,z* $\in \mathcal{L}$, $x \leq z$ implies $x \vee (y \wedge z) = (x \vee y) \wedge z$

Remark 3. As (d1) and (d2) are dual forms, one is suggested to consider the dual of (m1) whose form is

(m2) $z \leq x$ implies $x \wedge (y \vee z) = (x \wedge y) \vee z$

However, if one makes the substitutions $\hat{x} := z$ and $\hat{z} := x$, then the (m2) assumes the form

(m2) $\hat{x} \leq \hat{z}$ implies $\hat{z} \wedge (y \vee \hat{x}) = (\hat{z} \wedge y) \vee \hat{x}$

which is nothing else than the original (m1).

From Definitions 9 and 10, it is straightforward to prove that every distributive lattice is modular too; indeed, from (d1) $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ the condition $x \leq z$, i.e., $z = x \vee z$, leads to (m1). The vice versa is in general not true because it is possible to find modular lattices that are not distributive.

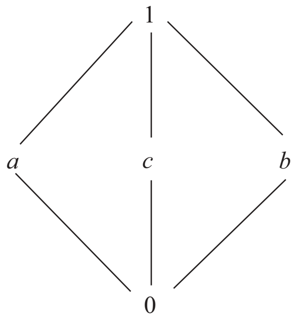


Figure 6. The modular lattice M5.

Example 17. This situation is exactly the case of the finite lattice M5 which has its Hasse diagram represented in Fig. 6.

Indeed, we have $a \wedge (b \vee c) = a \wedge 1 = a$ and $(a \wedge b) \vee (a \wedge c) = 0 \vee 0 = 0$, from which it follows that $a \wedge (b \vee c) \neq (a \wedge b) \vee (a \wedge c)$, i.e., this lattice is not distributive.

We prove now that this lattice is modular. Let $x \leq z$ be two nontrivial ($\neq$ 0,1) elements in M5; then, from the particular structure of M5 it follows that $x \vee (y \wedge z) = x \vee (y \wedge x) = x$ and $(y \vee z) \wedge z = (x \vee y) \wedge x = x$, by absorption (this latter does not hold in the lattice of 5 where, for instance, $b \leq a$, but $(b \vee c) \wedge a \neq (b \vee c) \wedge b$). In the nontrivial cases, we have to consider two subcases $0 < a$ and $a < 1$ by symmetry; in the first, we obtain $0 \vee (y \wedge a) = y \wedge a$ and $(0 \vee y) \wedge a = y \wedge a$, whereas in the latter, we have $a \vee (y \wedge 1) = a \vee y$ and $(a \vee y) \wedge 1 = a \vee y$.

As a concrete example of this behavior, let us consider the complete lattice $\mathcal{M}(\mathbb{R}^2)$ of all subspaces of the two-dimensional linear space $\mathbb{R}^2$. Then, if one set $0 := \{\underline{0}\}$ (the subspace consisting of the zero vector) and $1 = \mathbb{R}^2$, the following subspaces realize the Hasse diagram of Fig. 6:

$$a := M_x = \{(x,0) : x \in \mathbb{R}\} \quad b := M_y\{(0,y) : y \in \mathcal{R}\}$$
$$c := M_{\pi/4} = \{(x,x) : x \in \mathcal{R}\}$$

In general, it can be proved that

> any *finite dimensional* (real or complex) linear space is such that the complete lattice of all its subspaces is modular: "In particular, set–products and straight linear sums are known to satisfy the so-called modular identity" 6.

This result is very simple to prove in the two-dimensional case, whereas the proof of the general finite-dimensional case can be found in Ref. 6 (Footnote 23 of section 11).

Example 18. The lattice $\mathcal{M}(\mathbb{R}^2)$ of all subspaces of the real vector space $\mathbb{R}^2$ has been represented by the Hasse diagram in the right side of Fig. 4. To verify the condition (m1) of Definition 10, we first consider the trivial case of $\alpha = \gamma$. Because for any β it is $\alpha \leq \beta \vee \alpha$, we have $\alpha = \alpha \wedge (\beta \vee \alpha)$. If β represents any subspace different from α and $\mathbb{R}^2$, then $\beta \wedge \alpha = \{\underline{0}\}$, and so $\alpha = \alpha \vee \{\underline{0}\} = \alpha \vee (\beta \wedge \alpha)$: if $\beta = \mathbb{R}^2$, then $\alpha = \mathbb{R}^2 \wedge \alpha$, and so $\alpha = \alpha \vee \alpha = \alpha \vee (\mathbb{R}^2 \wedge \alpha)$. Thus, we have obtained that when α = γ, it is $\alpha \wedge (\beta \vee \alpha) = \alpha = \alpha \vee (\beta \wedge \alpha)$.

Let us now consider the strict ordering $\alpha < \gamma$. It is obvious that this condition implies that $\alpha = \{\underline{0}\}$ or $\gamma = \mathbb{R}^2$. But, if $\alpha = \{\underline{0}\}$, then $\{\underline{0}\} \vee (\gamma \wedge \beta) = \gamma \wedge \beta = (\{\underline{0}\} \vee \gamma) \wedge \beta$, and if $\beta = \mathbb{R}^2$, then it is $\alpha \vee (\gamma \wedge \mathbb{R}^2) = \alpha \vee \gamma = (\alpha \vee \gamma) \wedge \mathbb{R}^2$.

Lastly, we have lattices like the one shown in Fig. 5, which does not belong to the class of modular lattices. Indeed, for this lattice $b \leq a$, but $b \vee (c \wedge a) = b \vee 0 = b$ and $(b \vee c) \wedge a = 1 \wedge a = a$; from $a \neq b$ we can deduce $b \vee (c \wedge a) \neq (b \vee c) \wedge a$.

BOOLEAN LATTICES AND ALGEBRAS

The concept of complement has a great importance in lattice theory.

Definition 11. *Let $\mathcal{L}$ be a bounded lattice. We identify the element $y \in \mathcal{L}$ as the* complement *of a given element $x \in \mathcal{L}$ if it is such that*

$$x \vee y = 1 \quad \text{and} \quad x \wedge y = 0$$

It is easy to see that if y is the x complement, then x is the y complement, as well. However, the following relations $0 \vee 1 = 1$ and $0 \wedge 1 = 0$ imply that these two elements are complements of each other. Moreover, it is easy to prove the following result.

Proposition 8. *If in a distributive lattice $\mathcal{L}$ the complement of a given element exists, then it is unique.*

Proof. Suppose that $\mathcal{L}$ is a distributive lattice and that a is an element with two complements a_1 and a_2. Then, from $a \vee a_2 = 1$ and $a \wedge a_1 = 0$ it follows that

$$a_1 = a_1 \wedge (a \vee a_2) = \text{distributivity} = (a_1 \wedge a) \vee (a_1 \wedge a_2)$$
$$= a_1 \wedge a_2 \leq a_2$$

Similarly, $a_2 \leq a_1$, so $a_1 = a_2$.

In a nondistributive lattice, this uniqueness property, in general, does not hold.

Example 19. In the modular lattice M5 of Fig. 6, both c and b are complements of the element a because $a \wedge c = a \wedge b = 0$ and $a \vee c = a \vee b = 1$. In general, for any $x \in \{a, b, c\}$, the two elements $y \in \{a, b, c\}$ such that $y \neq x$ are both complements of x.

But, there could exist distributive bounded lattice in which only the elements 0 and 1 have the complement (which is 1 and 0, respectively).

Example 20. Let $\mathcal{L}$ be a totally ordered bounded lattice that contain more than two elements. Then it is easy to see that this lattice is distributive, but no other element than 0 and 1 can have a complement.

Definition 12. *We have a* complementable *bounded lattice when every element of the lattice has* at least *one complement. We have a bounded* complemented *lattice when every element a of the lattice has just one complement, denoted by a'.*

A *Boolean lattice $\mathcal{B}$* is a distributive bounded lattice equipped with a *complementation* unary mapping, i.e., every element x has a well-defined (unique) complement, denoted by x'. The operation of assigning the complement x' to every element x of the lattice defines a unary operation $'$ from $\mathcal{B}$ onto itself, called *complementation*. These very important structures were introduced by George Boole in 1854. At that time, Boole was trying to define a logical calculus that could coincide with the propositional (enunciative) calculus. Later on, it was possible to recognize as another important concrete Boolean lattice class, the power set $\mathcal{P}(X)$ of a *universe* X equipped with the set theoretic union and intersection, and the set theoretic complementation operations. Formally, a Boolean lattice is defined in the following way:

Definition 13. *A* Boolean lattice *is a structure$\langle \mathcal{B}, \wedge, \vee, \prime, 0, 1\rangle$where*

- *(b1) $\mathcal{B}$ is a set that contains two distinct elements* 0 *and* 1,
- *(b2) $\wedge$ and $\vee$ are two binary operations on$\mathcal{B}$ with respect to which the structure$\langle \mathcal{B}, \wedge, \vee, 0, 1\rangle$is a distributive lattice bounded by the least element* 0 *and the greatest element 1: $\forall x \in B, 0 \leq x \leq 1$,* and
- *(b3) For everyx $\in$ $\mathcal{B}$there existsx$'$ $\in$ $\mathcal{B}$such thatx $\wedge$ x$'$* = 0 *andx $\vee$ x$'$ = 1; x$'$ is the complement of x.*

The following are very important examples of Boolean lattice:

Example 21. Let $\langle\{0, 1\}, \leq\rangle$ be the set consisting of the two elements 0,1 equipped with the usual order relation $0 \leq 1$. This poset is a Boolean lattice with respect to the operations presented in the tables below (at the left the lattice operations and at the right the complementation):

a	b	$a \wedge b$	$a \vee b$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

x	x'
0	1
1	0

This part is usually known as the *two-valued* or *two-elements Boolean lattice*, denoted by $\mathbb{B} = \langle\{0, 1\}, \wedge, \vee, \prime, 0, 1\rangle$.

Example 22. The power set $P(X)$ of a universe X discussed in Example 9 is a Boolean lattice if we choose the set theoretic complement $A^c = X \backslash A := \{x \in X : x \in X \text{ and } x \notin A\}$ as the complement of a given set A in the universe X. Such a Boolean lattice is $\mathcal{P} = \langle \mathcal{P}(X), \cap, \cup, ^c, \emptyset, X\rangle$.

Example 23. The collection 2^X of all two-valued functionals on the universe X introduced in Example 10 is a Boolean lattice if we choose the functional $\chi' := 1 - \chi$ as the complement of a given functional χ. Such a lattice is $\mathbb{E} = \langle 2^X, \wedge, \vee, ', \underline{0}, \underline{1}\rangle$.

Example 24. In example 14, we have investigated the lattice of all closed sets of a topological space X. Taking inspiration from usual set theory, a natural way to introduce a complement of a generic closed set K could be its set complement K^c. The problem is that in general this latter is not a closed set. For instance, in the usual topology of $\mathbb{R}$ by closed sets, the closed interval $[a,b] := \{x, \in \mathbb{R} : a \leq x \leq b\}$ (for $a \leq b$) has set complement $[a,b]^c = (-\infty, a) \cup (b, +\infty)$ that is not closed.

In general, if $\mathcal{C}(X)$ is the collection of all closed sets (as primitive notion of a topology), then the collection of all set complement of closed sets determines (as derived notion in this context) the family $\mathcal{O}(X)$ of all *open* sets; formally, $O \in \mathcal{O}(X)$ iff there exists a closed set $K \in \mathcal{C}(X)$ such that $O = K^c$. On the basis of this definition and the properties (T1)–(T3) of Example 13, it is easy to prove that the family $\mathcal{O}(X)$ satisfies the following conditions:

(O1) The empty set and the whole space X are open $\emptyset, X \in \mathcal{O}(X)$.
(O2) The set union of *any* family of open sets is open, and
(O3) The *finite* intersection of open sets is open.

Now if one considers the collection $\mathcal{CO}(X)$ of all subsets of X that are both closed and open (the so-called *clopen* sets), then according to (T2) and (O2), the induced partial order structure is the one of complete lattice; moreover, for any clopen set U, its set complement U^c is obviously clopen, too, and so one obtains that $\langle \mathcal{CO}(X), \cap, \cup, ^c, \emptyset, X \rangle$ is a Boolean algebra.

In the topological space $\mathbb{R}$, the unique clopen sets are the trivial ones $\emptyset$ and $\mathbb{R}$, a very poor collection. A canonical way to obtain a sufficiently rich topology of clopen is the following one. Let us consider a *partition space* (X,π), which consists of a nonempty universe X equipped with a partition π, i.e., a collection of nonempty subsets of X that are pairwise disjoint and whose set union covers the universe. Let us denote by $\mathcal{CO}(X,\pi)$ the collection of all subsets of X that are union of equivalence classes from π (i.e., $E \in \mathcal{CO}(X,\pi)$ iff there exists a family $\{A_i\} \subseteq \pi$ such that $E = \cup A_i$) plus the empty set. Trivially, for any pair of sets, $E, F \in \mathcal{CO}(X,\pi)$, also, $E \cup F \in \mathcal{CO}(X,\pi)$ and $E \cap F \in \mathcal{CO}(X,\pi)$; moreover, if $E \in \mathcal{CO}(X,\pi)$, then its set complement $E^c \in \mathcal{CO}(X,\pi)$, too (it is also a set union of equivalence classes from π). Then, the structure $\langle \mathcal{CO}(X,\pi), \cap, \cup, ^c, \emptyset, X \rangle$ is the Boolean lattice induced from the partition π of X that, from the topological point of view, is a topological space of clopen sets.

This Boolean lattice is the basic structure of the so-called approach to *rough sets* as introduced by Pawlak in Ref. 7 (see Ref. 8 for the abstract approach to this argument). In this framework, one considers an information system formalized as a mapping $F : X \times Att \mapsto val$ defined on pairs $(x, a) \in X \times Att$ consisting of an object x and an attribute a, and furnishing the value $F(x, a) \in val$ assumed by the object x relatively to the attribute a. For any family of attributes $\mathcal{A} \subseteq Att$, it is possible to introduce the equivalence relation of *indistinguishability* on objects: $x(\text{Ind})_{\mathcal{A}}y$ iff for any $a \in \mathcal{A} : F(x,a) = F(y,a)$, i.e., two objects are equivalent if they cannot be distinguished relatively to the knowledge supported by all the attributes of $\mathcal{A}$.

Example 25. Let $D = \{1, 2, 5, 7, 10, 14, 35, 70\}$ be the set of all divisors of 70. Once defined $x \wedge y = \text{mcd}(x, y)$ $(5 \wedge 14 = 1,$ $10 \wedge 35 = 5)$, $x \vee y = \text{mcm}(x, y)$ $(5 \vee 14 = 70,\ 10 \vee 35 = 70)$ and $x' = \frac{70}{x}(5' = 14)$, the structure $\langle D, \wedge, \vee, ', 1, 70 \rangle$ is a Boolean lattice.

Summarizing, a Boolean lattice is a distributive, complementable lattice. But, we have seen that in a distributive lattice, the uniqueness property of the complement of each element holds, and so in a complementable distributive lattice $\mathcal{B}$, we can introduce a unary operation$'$: $\mathcal{B} \mapsto \mathcal{B}, x \to x'$, the *complementation*, which associates with each element x its unique complement x'. In this way, we can have the following equivalent, but equational, algebraic definition.

Definition 14. *A* Boolean algebra *is a sextuple* $\langle \mathcal{B}, \wedge, \vee, ', 0, 1 \rangle$, *where:*

i) *$\mathcal{B}$ is a set that contains two distinct elements,* 0 *and* 1,
ii) *$\wedge$ and $\vee$ are binary operations on $\mathcal{B}$*, and
(iii) *$' : \mathcal{B} \mapsto \mathcal{B}$ is a unary operation from $\mathcal{B}$ onto $\mathcal{B}$,*

such that for every x,y,z $\in \mathcal{B}$, *the following axioms hold:*

1) $x \wedge y = y \wedge x$ and $x \vee y = y \vee x$
2) $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ and $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
3) $x \vee 0 = x$ and $x \wedge 1 = x$
4) $x \vee x' = 1$ and $x \wedge x' = 0$

An alternative system of axioms for Boolean algebras is presented in Ref. 1.

Definition 15. *A structure* $\langle \mathcal{B}, \wedge, \vee, ', 0, 1 \rangle$ *with* $\wedge$ *and* $\vee$ *binary operations on*$\mathcal{B}$*and*$'$ *a unary operation on the same set is a Boolean algebra iff the following properties are satisfied for all* $a, b \in \mathcal{B}$:

(B1) $a \wedge b = (a' \vee b')'$
(B2) $a \vee b = b \vee a$
(B3) $a \vee (b \vee c) = (a \vee b) \vee c$
(B4) $(a \wedge b) \vee (a \wedge b') = a$

In this case, axioms (B1)–(B4) are independent, that is, no one of them can be derived from the other; if one's eliminates then the structure $\langle \mathcal{B}, \wedge, \vee, ', 0, 1 \rangle$ is no more a Boolean algebra. Independence is an important property of an algebra, because it enables us to determine which are the necessary properties to define it. Given a set of axioms, it is possible to prove the independence of one of them by finding a model (a concrete example) of the structure that satisfies all the axioms but this chosen one.

Example 26. Let us consider axioms (B1)–(B4). We show that (B2) is independent from the others. To do this, we give a structure, that satisfies (B1), (B3), (B4) and not (B2). The structure is the following: $\langle\{0,1\}, \wedge, \wedge, i\rangle$ (this means that $\vee := \wedge$), where $i(x) = x$ and $0 \wedge x = 0, 1 \wedge x = 1$. As can be easily seen, (B2) is not satisfied: $0 \vee 1 = 0 \neq 1 = 1 \vee 0$.

Proposition 9. 1. *In any Boolean algebra, it is possible to prove that the following properties hold:*

1) The lattice conditions

$$\begin{array}{ll} a \vee a =, a \wedge a = a & \text{(idempotency)} \\ a \vee b = b \vee a, a \wedge b = b \wedge a & \text{(commutativity)} \\ a \vee (b \vee c) = (a \vee b) \vee c & (\vee \text{ associativity}) \\ a \wedge (b \wedge c) = (a \wedge b) \wedge c & (\wedge \text{ associativity}) \\ a \vee (a \wedge b) = a, a \wedge (a \vee b) = a & \text{(absorption)} \end{array}$$

2) The distributivity conditions

$$\begin{array}{ll} a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) & (\vee \text{ distributivity}) \\ a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) & (\wedge \text{ distributivity}) \end{array}$$

3) The lattice zero and unit elements

$$\begin{array}{ll} a \vee 0 = a, a \vee 1 = 1 & (\vee \text{ boundry conditions}) \\ a \wedge 0 = 0, a \wedge 1 = a & (\wedge \text{ boundry conditions}) \end{array}$$

4) The orthocomplementation conditions

$$\begin{array}{ll} (a')' = a & \text{(involution)} \\ (a \wedge b)' = a' \vee b' & (\vee \text{ de Morgan law}) \\ (a \vee b)' = a' \wedge b' & (\wedge \text{ de Morgan law}) \\ a \vee a' = 1 & \text{(excluded middle law)} \\ a \wedge a' = 0 & \text{(noncontradiction law)} \end{array}$$

Also, for the case of Boolean lattices, in Ref. 5 it is possible to find a result that represents the analog of Theorem 3 (for lattices) and Theorem 4 (for distributive lattices).

Theorem 5. *Let $\langle \mathcal{B}, \wedge, \vee \rangle$ be an algebraic structure where:*

1) $\mathcal{L}$ is a nonempty set, and

2) $\wedge$ and $\vee$ are two binary operations on $\mathcal{L}$ satisfying properties:

(DL1) $x = x \wedge (x \vee y)$

(DL2) $x \wedge (y \vee z) = (z \wedge x) \vee (y \wedge x)$

(BL) to each y there corresponds some y' such that

$$x \wedge (y \vee y') = x \vee (y \wedge y')$$

Then, $\mathcal{L}$ is a Boolean lattice.

Now we can extend the notion of lattice isomorphism to the case of Boolean algebras.

Definition 16. *Two Boolean algebras $\mathcal{B}_1$ and $\mathcal{B}_2$ are said to be* isomorphic *iff there exists a lattice isomorphism $\phi : \mathcal{B}_1 \mapsto \mathcal{B}_2$ that preserves the complementation,*

$$\forall x \in \mathcal{B}_1, \quad \phi(x') = \phi(x)'$$

Example 27. The lattice isomorphism seen in Example 15 between the power set $\mathcal{P}(X)$ of a universe X and the collection 2^X of all the Boolean–valued functionals on $X, \chi : \mathcal{P}(X) \mapsto 2^X, A \in \mathcal{P}(X) \to \chi_A \in 2^X$, is, indeed, a Boolean lattice isomorphism because trivially for any subset A of X

$$\chi_{A^c} = 1 - \chi_A$$

As we have seen in Example 22, the collection of all subsets of a universe X is a Boolean algebra under set inclusion (with associated set union and intersection) and set complementation. The following is an interesting *representation theorem* of finite Boolean algebras.

Theorem 6 (Stone's Representation Theorem). *Every finite Boolean algebra $\mathcal{B}$ is isomorphic to the algebra of parts of the set of its atoms.*

The preceding theorem causes the following important consequences:

1) The cardinality of every finite Boolean algebra is a power of 2.

2) Two finite Boolean algebras with the same cardinality are isomorphic to each other.

Let us explain this procedure with one example.

Example 28. In Fig. 7, the left side is the Hasse diagram of the (finite) Boolean lattice B8, whose atoms are the elements

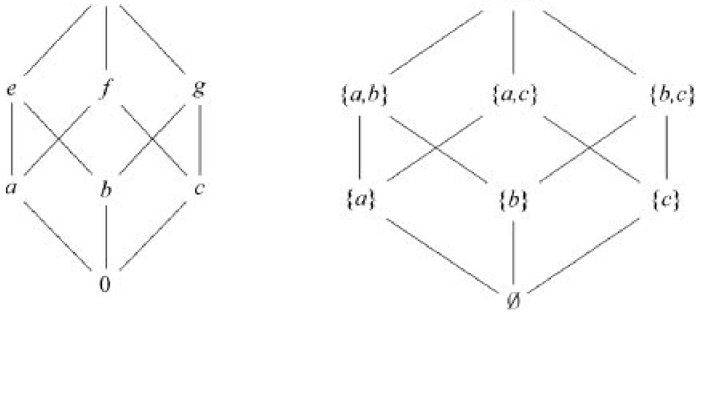


Figure 7. The finite Boolean lattice B8 with its Stone representation 2^3 of all subsets of the universe $\{a, b, n\}$.

a, b, and *c*. If one then considers the universe $X = \{a,\ b,\ c\}$ formed by all atoms of B8, the power set 2^3 of this universe ordered with the standard set theoretic ordering (the right side of the figure) is the Stone representation of B8.

The general procedure is the following one. Let $\mathcal{B}$ be a finite Boolean lattice, and let us denote by $A(\mathcal{B})$ the collection of all its atoms, assumed as the universe of the Stone representation of $\mathcal{B}$. Then, let us construct the mapping $\phi : \mathcal{B} \mapsto \mathcal{P}(A(B))$ that assigns to any element $x \in \mathcal{B}$ of the Boolean lattice the subset $\phi(x) = \{a \in A(\mathcal{B})) : a \leq x\}$, element of the power set $\mathcal{P}(A(\mathcal{B}))$ obtained as the collection of all atoms a that precede x. For instance, in the example below, $\phi(e) = \{a,\ b\}$ because a and b are the two atoms that precede e. Trivially, the mapping ϕ is a Boolean lattice isomorphism. For instance, $\phi(e \vee g) = \phi(1) = \{a,b,c\} = \{a,b\} \cup \{b,c\} = \phi(e) \vee \phi(g)$.

The extension of this representation procedure to the case of an infinite Boolean lattice presents some problems because it is easy to give examples either of nonatomic or of noncomplete Boolean algebras and any Boolean algebra $\mathcal{P}(X)$ of all subsets of a universe X has both these properties. The best results of this situation are given by the following.

Proposition 10. *Let $\mathcal{B}$ be an abstract Boolean algebra, and let $A(\mathcal{B})$ denote the set of all its atoms. Then, the mapping $\phi : \mathcal{B} \mapsto \mathcal{P}(A(\mathcal{B}))$ associating with any element $x \in \mathcal{B}$ the subset of atoms $\phi(x) := \{a \in A(\mathcal{B}) : a \leq x\}$ is a homomorphism of Boolean lattices, i.e., both conditions in Equation* (7) *of Definition 8 are satisfied by the mapping ϕ without the certainty that the bijective condition is assured. Moreover,*

i) if $\mathcal{B}$ is atomic, then the mapping ϕ is injective, and
ii) if $\mathcal{B}$ is complete, then the mapping ϕ is surjective.

Proof. The fact that ϕ is a Boolean algebras homomorphism follows from the remark that an atom a satisfies the condition $a \leq x \vee y$ iff either $a \leq x$ or $a \leq y$. Analogously, an atom a satisfies the condition $a \leq x \wedge y$ iff both $a \leq x$ and $a \leq y$.

If $\mathcal{B}$ is atomic, then for any pair of its elements $x \neq y$, their symmetric difference $x\Delta y := (x \wedge y') \vee (y \wedge x')$ admits an atom $a \leq (x\Delta y)$; this implies that either $a \leq x \wedge y'$ or $a \leq y \wedge x'$ but not both (otherwise, for instance, $a \leq x \wedge x' = 0$). Hence, a will be in just one, and only one, of $\phi(x)$ and $\phi(y)$.

If $\mathcal{B}$ is complete, then for any subset Y of $A(\mathcal{B})$ there exists the element $z = \vee\{y : y \in Y\}$ in $\mathcal{B}$ such that $\phi(z) = Y$.

As an immediate consequence, we have:

Theorem 7. *A Boolean algebra $\mathcal{B}$ is isomorphic to the algebra $\mathcal{P}(X)$of all subsets of some universe X iff it is atomic and complete.*

This result states the problem of what happens in all the other cases of Boolean algebras for which at least one of the above conditions does not hold. In Example 24, we have seen that given a topological space X, the clopen subsets of X are closed under intersection, union, and complementation forming a Boolean algebra. Note that, in general, the topology of clopen set $\mathcal{CO}(X)$ of a space X is contained in the power set of this latter, $\mathcal{CO}(X) \subseteq \mathcal{P}(X)$, in other words, it is a subBoolean algebra of the power set. A well-known theorem of Stone 9 asserts that *every* Boolean algebra is represented isomorphically by one of such and that if one requires the topological space to be compact Hausdorff as well as totally disconnected (what is now called *Stone space*, whereas Stone originally called these spaces with the name of *Boolean spaces*), then this representation is essentially uniquely determined by the Boolean algebra. The meaning of this isomorphism is that any algebraic fact about Boolean algebras could be translated into a topological fact about Stone spaces, and vice versa.

Ortholattices

Point 4 of Proposition 9 lists the conditions that are satisfied by the complement of a Boolean algebra. Relative to this point, and as the consequence of the uniqueness of the complement a' of a given element a from the Boolean algebra $\mathcal{B}$, it is possible to introduce a unary operation $' : \mathcal{B} \mapsto \mathcal{B}, a \to a'$ that satisfies the condition of involution, both the de Morgan laws, the excluded middle, and the noncontradiction laws. We can formalize this situation in the following way:

Definition 17. *An orthocomplemented lattice*, ortholattice *for short, is a structure $\langle \mathcal{L}, \wedge, \vee, ', 0\rangle$ of lower-bounded lattice $\langle \mathcal{L}, \wedge, \vee, 0\rangle$ equipped with an ortho-complementation mapping$' : \mathcal{L} \mapsto \mathcal{L}$that satisfies the equational conditions:*

$$\begin{array}{lll}
\text{(oc1)} & x = x'' & \text{(double negation)} \\
\text{(oc2a)} & x' \wedge y' = (x \vee y)' & \text{(de Morgan)} \\
\text{(oc3a)} & x \wedge x' = 0 & \text{(noncontradiction)}
\end{array}$$

With respect to the properties listed in point 4 of Proposition 9, in the now-introduced definition, it involves only one de Morgan property. Also, the two properties of point 3 in the same proposition are simplified by the unique (oc3). But the following results hold:

Proposition 11. *In any lattice, under condition (oc1), the following are equivalent:*

$$\begin{array}{lll}
\text{(oc2a)} & x' \wedge y' = (a \vee y)' & \text{(first de Morgan law)} \\
\text{(oc2b)} & x' \vee y' = (a \wedge y)' & \text{(second de Morgan law)} \\
\text{(oc2c)} & x \leq y \text{ implies } y' \leq x' & \text{(contraposition law)} \\
\text{(oc2d)} & x' \leq y' \text{ implies } y \leq x & \text{(dual contraposition law)}
\end{array}$$

In a similar way, in any bounded lattice under conditions (oc1) and (oc2), the following two are equivalent:

$$\begin{array}{lll}
\text{(oc3a)} & x \wedge x' = 0 & \text{(noncontradiction law)} \\
\text{(oc3b)} & x \vee y' = 1 & \text{(excluded middle law)}
\end{array}$$

Let us stress that the just-introduced notion of ortholattice can be applied to lattice that are not necessarily distributive, but with the loss of the uniqueness of the orthocomplementation mapping that characterizes Boolean

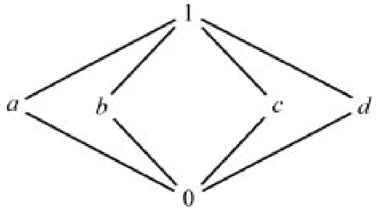


Figure 8. The six elements modular lattice M6.

algebras. In this way, the complementation mapping assumes an important role, outside the "restricted" class of Boolean algebras. We clarify this fact with an example from the larger class of modular lattices.

Example 29. Let us consider the (six-elements) modular lattice M6 represented by the Hasse diagram of Fig. 8.

Then, any nontrivial element $x \neq 0{,}1$ has three complements, precisely every nontrivial $y \neq 0{,}1$ such that $y \neq x$. So, this lattice is complementable, but it is possible to assign more than one orthocomplementation mapping, two of which are represented by the diagrams drawn in Fig. 9.

Example 30. Let us consider the two-dimensional linear space $\mathbb{R}^2$. We have seen in Example 14 that the collection $\mathcal{M}(\mathbb{R}^2)$ of all its subspaces is a modular (nondistributive) lattice, bounded by the minimum subspace $0 = \{\underline{0}\}$ and the maximum subspace $1 = \mathbb{R}^2$. In this lattice any one-dimensional subspace α–has an infinite collection of complements, i.e., any one-dimensional subspace $\beta \neq \alpha$ is a complement of α. At a first glance, it seems that there could be no intuitive criterium to select for each α one of these infinite subspaces, say α', as its *unique* orthocomplement.

But, $\mathbb{R}^2$ can be equipped with the inner product defined for any pair of its vectors $\underline{x} = (x_1, x_2)$ and $\underline{y} = (y_1, y_2)$ as the real number

$$\langle \underline{x} | \underline{y} \rangle := x_1 y_1 + x_2 y_2$$

The standard notion of *orthogonality* is the following one:

$$\underline{x} \perp \underline{y} \quad \text{iff} \quad \langle \underline{x} | \underline{y} \rangle = 0$$

Now, for any subspace M of $\mathbb{R}^2$ (i.e., the trivial subspaces $\{\underline{0}\}$ and $\mathbb{R}^2$, and all one-dimensional subspaces) one can define its complement, denoted by $M^\perp$ as the subspace that is orthogonal to M. Formally,

$$M^\perp := \{\underline{h} \in \mathbb{R}^2 : \forall \underline{m} \in M, \langle \underline{h} | \underline{m} \rangle = 0\}$$

This is a subspace, and the mapping $\perp : \mathcal{M}(\mathbb{R}^2) \mapsto \mathcal{M}(\mathbb{R}^2), M \mapsto M^\perp$ is an orthocomplementation acting on a modular lattice. Probably, this example has been the cause of the term *ortho*complementation given to any mapping that satisfies the above conditions (oc1)–(oc3).

For instance, if one considers the subspace $M_{\pi/4} = \{(x,x) \in \mathbb{R}^2; x \in \mathbb{R}\}$ then its orthocomplement is $(M_{\pi/4})^\perp = \{(h,-h) \in \mathbb{R}^2 : h \in \mathbb{R}\} = M_{(3/4)\pi}$ because a fixed generic vector $(h_0, -h_0) \in (M_{\pi/4})^\perp$ is such that for any $(x,x) \in M_{\pi/4}$ the orthogonality condition $\langle (h_0, -h_0) | (x,x) \rangle = 0$ holds. However, $M_x = \{(x,0) : x \in \mathbb{R}\}$ and $M_y = \{(0,y) : y \in \mathbb{R}\}$ are such that trivially $(M_x)^\perp = M_y$ and $(M_y)^\perp = M_x$. If one considers the modular sublattice of $\mathcal{M}(\mathbb{R}^2)$ consisting of the six subspaces $\{\{\underline{0}\}, M_x, M_{\pi/2}, M_y, M_{(3/4)\pi}, \mathbb{R}^2\}$, then the corresponding Hasse diagram has the form of the one depicted in the right side of Fig. 9 by the identifications $a = M_x, b = M_{\pi/2}, c = M_y$ and $d = M_{(3/4)\pi}$.

Let us make now a brief discussion about the semantical interpretation of the algebraic operations of an orthocomplemented lattice with respect to the ordinary propositional logic, the so–called *algebraic semantic* of a logic. First of all, let us stress that "when interpreting [lattice] structures from the viewpoint of logic, it is customary to interpret the lattice elements [a, b,. . .] as *propositions,* the meet operation [$\wedge$] as *conjunction* [logical And], the join operation [$\vee$] as *disjunction* [logical Or], and the orthocomplement [$\neg$] as *negation* [logical Not]" (10). Hence, from the point of view of *algebraic semantic,* "clearly L1–L4 [of a lattice, according to Theorem 1] are well-known properties of And and Or in ordinary logic. This gives an algebraic reason for admitting as a *postulate* (if necessary) the statement that a given calculus of proposition is a lattice" 6. In this lattice context, "it is generally agreed that the partial ordering $\leq$ of a lattice [. . .] can be logically interpreted as a *relation* of implication, more specifically *semantic entailment"*(6), which must not be confused with an implication logical *connective* that, like conjunction and disjunction, forms propositions out of propositions remaining at the same linguistic level.

In particular, Boolean algebras have a significant role in classical logic. Indeed, once interpreted the elements of the algebra "as representing [. . .] *propositions* or *statements* or *sentences"* (11), then all the postulates (axioms) of a Boolean algebra represent axioms of Boolean logic.

However, the modular distributive lattice of all subspaces of a finite-dimensional (complex) linear space (or, better, the orthomodular version of any infinite-dimensional complex separable Hilbert space) has a significant role in the so–called quantum logic or logic of quantum mechanics, as described by Birkhoff and von Neumann in their seminal paper about this subject 6. Quoting from the latter: "The object of the present paper is to discover what

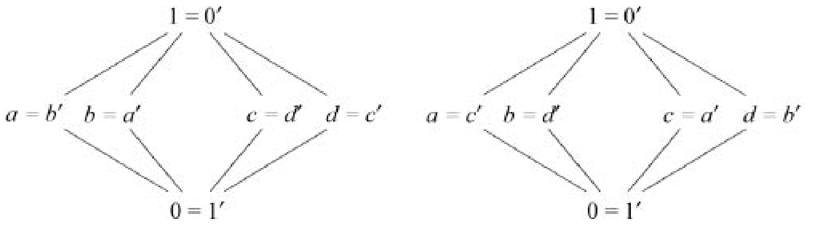


Figure 9. Two different orthocomplementations of the modular lattice M6 of Fig. 8.

logical structure one may hope to find in physical theories which, like quantum mechanics, do not conform to classical logic. Our main conclusion ... is that one can reasonably expected to find a calculus of propositions which is formally indistinguishable from the calculus of linear subspaces with respect to *set products* [i.e., intersection], *linear sums* [i.e., linear subspace generated by the set union], and *orthogonal complements* – and resembles the usual calculus of propositions with respect to *and*, *or*, and *not*."

Coming back to the general situation of an orthocomplementation in a generic lattice, condition (oc1) is the algebraic counterpart of the *strong double negation law* of a negation. Conditions (oc2a,b) are the *de Morgan* laws, whereas (oc2c) is the *contraposition* law and (oc2d) its dual. Finally, condition (oc3a) is the lattice realization of the *noncontradiction* law and (oc3b) of the *excluded middle* law.

Let us stress that conditions (oc3a) and (oc3b) define the mapping $'$ as a *complementation*, whereas conditions (oc1) and (oc2a–d) define $'$ as an *orthogonality* operator because one can introduce the *orthogonality* binary relation $a \perp b$ iff $a \leq b'$ or equivalently $b \leq a'$. For these reasons, the mapping $'$ has been called an *orthocomplementation*. Sometimes, for the sake of simplicity, we shall also name this mapping as *complementation* and the lattice as *complemented lattice,* if no confusion is likely.

CHARACTERISTIC FUNCTIONS ON A UNIVERSE AS SHARP SETS

As we have seen in Example 15, in the power set $\mathcal{P}(X)$ based on a universe X, any subset A of X can be represented by the corresponding *characteristic functional*$\chi_A : X \mapsto \{0,1\}$, which is the *crisp* (or *exact*, also *sharp*) realization of the set A, i.e., a mapping defined as follows:

$$\chi_A(x) := \begin{cases} 1, & \text{iff} \quad x \in A \\ 0, & \text{iff} \quad x \notin A \end{cases}$$

In this way, the set $\varepsilon(X) := \{0,1\}^X$ of all characteristic functionals on X has the structure $\langle \varepsilon(X), \leq, \prime, 0, 1\rangle$ of Boolean complete lattice ($0 = \chi_\emptyset$ and $1 = \chi_X$ being the characteristic functionals of the empty set and of the whole space respectively) with respect to the usual pointwise ordering:

$$\text{(or)} \quad \chi_A \leq \chi_B \quad \text{iff} \quad \text{for all} \quad x \in X, \quad \chi_A(x) \leq \chi_B(x)$$

The lattice meet of any two sharp sets $\chi_A, \chi_B \in \varepsilon(X)$ is the sharp set $\chi_A \wedge \chi_B \in \varepsilon(X)$ defined $\forall x \in X$ in the following way:

$$(\chi_A \wedge \chi_B)(x) = \min\{\chi_A(x), \chi_B(x)\} \tag{8a}$$

$$= \max\{0, \chi_A(x) + \chi_B(x) - 1\} \tag{8b}$$

$$= \chi_A(x) \cdot \chi_B(x) \tag{8c}$$

The lattice join of any two sharp sets $\chi_A, \chi_B \in \varepsilon(X)$ is the sharp set $\chi_A \vee \chi_B \in \varepsilon(X)$ defined $\forall x \in X$ by the law

$$(\chi_A \vee \chi_B)(x) = \max\{\chi_A(x), \chi_B(x)\} \tag{9a}$$

$$= \min\{1, \chi_A(x) + \chi_B(x)\} \tag{9b}$$

$$= \chi_A(x) + \chi_B(x) - \chi_A(x) \cdot \chi_B(x) \tag{9c}$$

The orthocomplementation mapping associates to any characteristic functional χ_A the characteristic functional

$$(\chi_A)'(x) = (1 - \chi_A)(x) \tag{10a}$$

$$= \begin{cases} 1, & \text{iff} \quad \chi_A(x) = 0 \\ 0, & \text{iff} \quad \chi_A(x) = 1 \end{cases} \tag{10b}$$

Of course, the mapping $\chi : \mathcal{P}(X) \mapsto \varepsilon(X), A \mapsto \chi_A$ is an isomorphism between the two Boolean lattice structures $\langle \mathcal{P}(X), \cap, \cup, {}^c, \emptyset, X\rangle$ and $\langle \varepsilon(X), \wedge, \vee, \prime, 0, 1\rangle$ because besides the identities $\chi_{A \cap B} = \chi_A \wedge \chi_B$ and $\chi_{A \cup B} = \chi_A \vee \chi_B$, one has that $\chi_{A^c} = (\chi_A)' = 1 - \chi_A$. This isomorphism preserves also the partial ordering:

$$A \subseteq B \quad \text{iff} \quad \chi_A \leq \chi_B$$

According to the classic logic approach, the implication connective is algebraically realized by the characteristic function of the set $A \to B := A^c \cup B$

$$\chi_{A \to B} = \chi_{A^c \cup B} = \max\{1 - \chi_A, \chi_B\} \tag{11}$$

from which trivially the minimal implication condition $\chi_{A \to B} = 1 \quad \text{iff} \quad A \subseteq B$ follows.

GENERALIZED CHARACTERISTIC FUNCTIONS AS FUZZY (UNSHARP) SETS

The more natural generalization of the notion of characteristic (or, equivalently, Boolean-valued) functional on the reference space X is the notion of *generalized characteristic functional* or *fuzzy set*, which is defined as a functional $f : X \mapsto [0,1]$ whose range may assume all the real values of the interval [0,1], not only the extreme ones, either 0 or 1. Let us stress that in fuzzy theory, any value $f(x) \in [0,1]$ is not interpreted as a possible probability value assumed by the object $x \in X$ but, on the contrary, as a *degree of membership* of this object relative to the fuzzy set f under consideration.

Example 31. Let us define the fuzzy set of *tall* people. So, we have that x is a man/woman belonging to a certain universe X, and *h(x)* is a real positive number, which

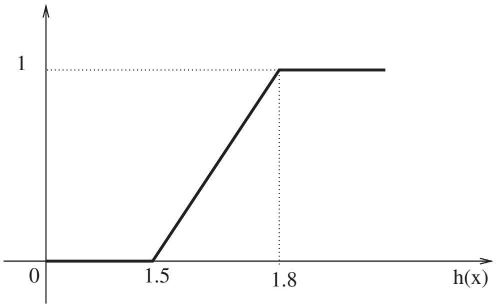


Figure 10. Fuzzy set of tall people.

represents the height of x in meters (see Fig. 10). The tall fuzzy set is defined as:

$$t(x) := \begin{cases} 1 & h(x) \geq 1.80 \\ \dfrac{h(x) - 1.50}{0.30} & 1.50 < h(x) < 1.80 \\ 0 & h(x) \leq 1.50 \end{cases}$$

The set $\mathcal{F}(X) := [0,1]^X$ of all generalized characteristic functionals is a partially ordered set with respect to the ordering

$$(\text{or} - \text{F}) \quad f_1 \leq f_2 \quad \text{iff} \quad f_1(x) \leq f_2(x), \quad \text{for all} \quad x \in X$$

which is an extension of the ordering (or) previously introduced on the set $\varepsilon(X)$ of all crisp characteristic functionals. With respect to this ordering, $\mathcal{F}(X)$ turns out to be a distributive complete lattice. In particular, we have that the g.l.b. $f \wedge g$ and the l.u.b. $f \vee g$ of two fuzzy sets with respect to this ordering are given, for any $x \in X$, respectively as

$$(f \wedge g)(x) = \min\{f(x), g(x)\} = f(x) \wedge g(x) \qquad (12)$$

$$(f \vee g)(x) = \max\{f(x), g(x)\} = f(x) \vee g(x) \qquad (13)$$

These operations are just the operation of intersection and union that are customarily introduced in the usual theory of fuzzy set (see Ref. 3).

Example 32. Let us define the fuzzy set of short people, s for all $x \in X$, as

$$s(x) := \begin{cases} 1 & h(x) \leq 1.45 \\ \dfrac{1.70 - h(x)}{0.25} & 1.45 < h(x) < 1.70 \\ 0 & h(x) \geq 1.70 \end{cases}$$

Considering also the fuzzy set of tall people defined in Example 31, the set of tall *or* short people, i.e., $t \vee s$ is defined $\forall x \in X$ as

$$(t \vee s)(x) := \begin{cases} 1 & h(x) \leq 1.45 \text{ or } h(x) \geq 1.80 \\ \dfrac{1.70 - h(x)}{0.25} & 1.45 < h(x) < 1.61 \\ \dfrac{h(x) - 1.50}{0.30} & 1.61 \leq h(x) < 1.70 \end{cases}$$

The set of tall *and* short people i.e., $t \wedge s$, is defined for all x

$$(t \wedge s)(x) := \begin{cases} 0 & h(x) \leq 1.50 \text{ or } h(x) \geq 1.70 \\ \dfrac{h(x) - 1.50}{0.30} & 1.50 < h(x) < 1.61 \\ \dfrac{1.70 - h(x)}{0.25} & 1.61 \leq h(x) < 1.80 \end{cases}$$

In Fig. 11, both fuzzy set intersection $t \wedge s$ and union $t \vee s$ of tall and short people are drawn.

Let us remark that a person can be at the same time and with a certain degree tall and short. Let us consider for example that x is a person who has an eighth of $h(x) = 1.55$ meters. Then, x is “tall *and* short” with a degree $\{t \wedge s)(x) = 0.17$.

Negations in Fuzzy Set Theory

The generalization of the orthocomplement of a fuzzy set f, starting from the equivalent forms in Equation (10) of the crisp case, presents some trouble because it is not so univocal as it could seem, and, moreover, all the here-analyzed

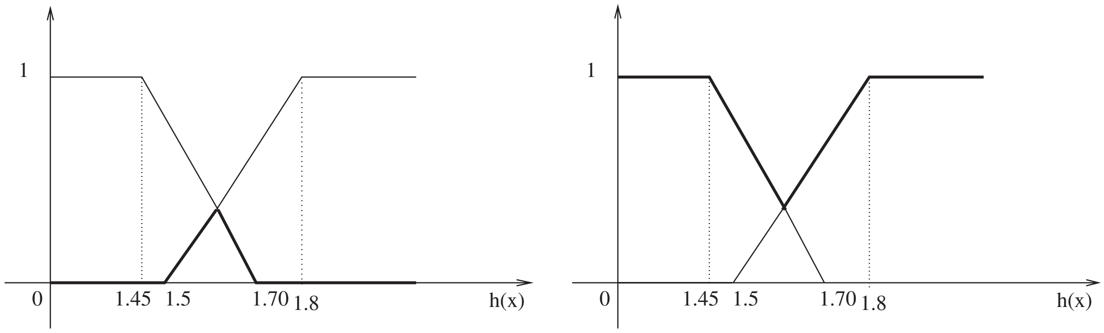


Figure 11. Fuzzy set of intersection (at left) and union (at right) of tall and short people.

versions present some pathological behavior with respect to the standard properties that characterize the standard version of negation (i.e., orthocomplementation) as introduced in Definition 17, with the Proposition 11. First of all, we can have at least these three distinct possibilities of fuzzy negation:

1) The *diametrical* orthocomplement

$$f'(x) := (1 - f)(x) \tag{14a}$$

which is an extension of Equation (10a), and the following, which are two different extensions of Equation (10b)

2) The *intuitionistic* orthocomplement

$$f^{\sim}(x) := \begin{cases} 1, & \text{if} \quad f(x) = 0 \\ 0, & \text{if} \quad f(x) \neq 0 \end{cases} \tag{14ba}$$

3) The *anti-intuitionistic* orthocomplement

$$f^{\flat}(x) := \begin{cases} 1, & \text{if} \quad f(x) \neq 1 \\ 0, & \text{if} \quad f(x) = 1 \end{cases} \tag{14ca}$$

Remark 4. If one introduces the subset of the universe

$$A_0(f) := \{x \in X : f(x) = 0\}$$

collection of all objects from the universe X in which the fuzzy set f is *impossible* (the membership degree is 0), called the *certainly not* or *impossibility domain* of f, then $f^{\sim}$ is the crisp set $f^{\sim} = \chi A_0(f)$. Similarly, if one defines as *contingency domain* of f the subset

$$A_c(f) := \{x \in X : f(x) \neq 1\}$$

collection of all objects in which the membership degree of f is not certain, then f^b is the crisp set $f^b = \chi A_c(f)$.

Trivially, for the ordering point of view

$$f^{\sim} \leq f' \leq f^b$$

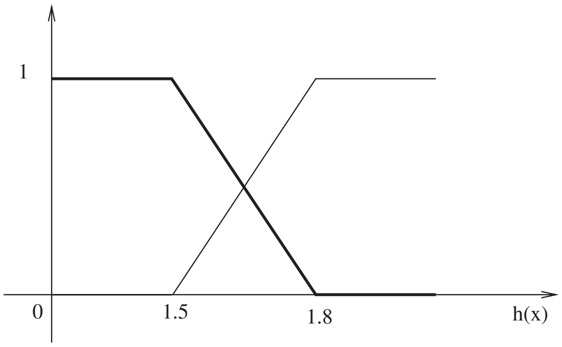


Figure 12. Fuzzy set of not tall people with respect to diametrical complementation.

Example 33. Let us consider the fuzzy set of tall people. Using the diametrical negation (see Fig. 12), the set of *not tall* people is thus defined as:

$$(t')(x) = \begin{cases} 0 & h(x) \geq 1.80 \\ \dfrac{1.80 - h(x)}{0.30} & 1.50 \quad < h(x) \quad < 1.80 \\ 1 & h(x) \leq 1.50 \end{cases}$$

The *impossible tall* people with respect to the intuitionistic negation is:

$$(t^{\sim})(x) = \begin{cases} 1 & \text{if} \quad h(x) \leq 1.50 \\ 0 & \text{otherwise} \end{cases}$$

Finally, using the anti–intuitionistic negation (see Fig. 13) we obtain that the *contingent tall* people is:

$$(t^b)(x) = \begin{cases} 1 & \text{if} \quad h(x) \leq 1.80 \\ 0 & \text{otherwise} \end{cases}$$

Hence, if we consider a person whose height is $h(x) = 1.70$, we have that he/she is not tall with values: $(t')(1.70) = \frac{1}{3}$, $(t^{\sim})(1.70) = 0$, and $(t^b)(1.70) = 1$.

Diametrical Complementation

The diametrical complementation allows one to introduce the mapping $' : \mathcal{F}(X) \to \mathcal{F}(X)$, $f \to f'$, which is a unusual

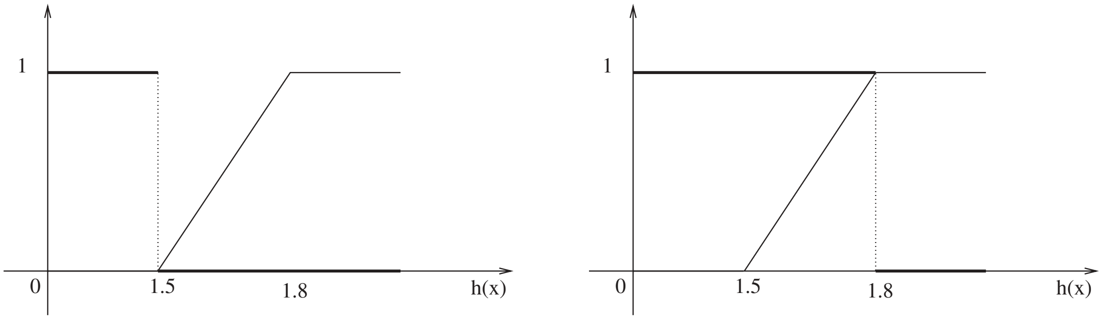


Figure 13. Fuzzy set of not tall people with respect to intuitonistic (at left) and anti-intuitionistic (at right) complementation.

orthocomplementation because it satisfies only the following conditions:

$$\text{(ocl)}\quad f'' = f, \text{for all } f \in \mathcal{F}(X)$$

$$\begin{array}{ll} \text{(oc2a)} & \text{Let } f, g \in \mathcal{F}(X), \text{ then } f' \wedge g' = (f \vee g)' \\ \text{(oc2b)} & \text{Let } f, g \in \mathcal{F}(X), \text{ then } f' \vee g' = (f \wedge g)' \\ \text{(oc2c)} & \text{Let } f, g \in \mathcal{F}(X), \text{ then } f \leq g \text{ implies } g' \leq f' \\ \text{(oc2d)} & \text{Let } f, g \in \mathcal{F}(X), \text{ then } f' \leq g' \text{ implies } g \leq f \end{array}$$

Moreover, the diametrical orthocomplementation satisfies the following *Kleene* condition:

$$\text{(KL)}\quad \text{for every } f, g \in \mathcal{F}(X),\ f \wedge f' \leq g \vee g'$$

Remark 5. As stressed before, under condition (oc1), all the conditions (oc2a–d) are mutually equivalent among them.

Let us denote by $\mathbf{k}$, for any fixed $k \in [0, 1]$, the fuzzy set $\forall x \in X, \mathrm{k}(x) = k$. We observe that the fuzzy set $\mathbf{1/2}$, called the *half* fuzzy set, satisfies $\mathbf{1/2} = \mathbf{(1/2)}'$. Making use of the half fuzzy set, condition (KL) can be strengthened more in the form:

$$\text{(KLa)}\quad \text{For every } f, g \in \mathcal{F}(X),\ f \wedge f' \leq 1/2 \leq g \vee g'$$

The unusual behavior of this weak form of negation with respect to the standard orthocomplementation is that the *contradiction* law $'\forall\ f,\ f \wedge f' = \mathbf{0}'$, and the *excluded middle* law $'\forall\ f,\ f \vee f' = \mathbf{1}'$, in general, do not hold for the diametrical orthocomplementation. In particular, we have that

$$(\mathbf{1/2}) \wedge (\mathbf{1/2})' = (\mathbf{1/2}) \vee (\mathbf{1/2})' = (\mathbf{1/2}) \neq \mathbf{0}, \mathbf{1}$$

The Intuitionistic Complementation

The intuitionistic complementation allows one to introduce the mapping $^{\sim} : \mathcal{F}(X) \to \mathcal{F}(X),\ f \to f^{\sim}$, which also, in this case, turns out to be an unusual orthocomplementation because the following conditions are fulfilled:

$$\text{(woc1)}\quad f \leq f^{\sim\sim}, \text{ for all } f.$$

$$\begin{array}{ll} \text{(woc2a)} & \text{Let } f, g \in \mathcal{F}(X); \text{ then } f^{\sim} \wedge g^{\sim} = (f \vee g)^{\sim}. \\ \text{(woc2b)} & \text{Let } f, g \in \mathcal{F}(X); \text{ then } f \leq g \text{ implies } g^{\sim} \leq f^{\sim}. \end{array}$$

$$\text{(woc3)}\quad f \wedge f^{\sim} = \mathbf{0}, \text{ for all f.}$$

The anti-intuitionistic complementation can be derived from the other two complementations according to the formula:

$$f^{b} = f'^{\sim\prime}$$

Remark 6. From the general point of view, in an abstract lattice, the condition (woc1) implies only the equivalence between (woc2a) and (woc2b); the de Morgan law "dual" of (woc2a) $f^{\sim} \vee g^{\sim} = (f \wedge g)^{\sim}$ in general is not true.

A dual behavior can be stated for the anti–intuitionistic complementation. For these reasons the following result is quite important because it shows a behavior that is peculiar of classic fuzzy set theory.

Proposition 12. *The distributive complete lattice of all fuzzy sets on the universe X satisfies the "dual" de Morgan laws:*

$$\begin{array}{ll} (woc2c) & \text{Let } f, g \in \mathcal{F}(X); \text{ then } f^{\sim} \vee g^{\sim} = (f \wedge g)^{\sim}. \\ (aoc2c) & \text{Let } f, g \in \mathcal{F}(X); \text{ then } f^{b} \wedge g^{b} = (f \vee g)^{b}. \end{array}$$

Proof. For any fixed point $x \in X$, without losing in generality, we can assume that $f(x) \leq g(x)$. Then, owing to (woc2b), $g^{\sim}(x) \leq f^{\sim}(x)$, from which we get $f^{\sim}(x) \vee g^{\sim}(x) = f^{\sim}(x)$. However $(f \wedge g)(x) = f(x)$, and so $(f \wedge g)^{\sim}(x) = f^{\sim}(x)$. Therefore, $\forall x \in X, (f^{\sim} \vee g^{\sim})(x) = (f \wedge g)^{\sim}(x)$.

The anti-intuitionistic case can be proved in a similar way.

The intuitionistic-like orthocomplementation satisfies the *weak double negation* law (woc1), which algebraically expresses the fact that any proposition f "implies" its double negation $f^{\sim\sim}$, but, in general, the vice versa does not hold. In particular, $(\mathbf{1/2})^{\sim} = \mathbf{0}$, from which it follows that $(\mathbf{1/2})^{\sim\sim} = \mathbf{0}^{\sim} = \mathbf{1}$, concluding that $(\mathbf{1/2}) < (\mathbf{1/2})^{\sim\sim}$, with $(\mathbf{1/2}) \neq (\mathbf{1/2})^{\sim\sim}$.

The *excluded middle* law for the intuitionistic-like orthocomplementation $'\forall\ f,\ f \vee f^{\sim} = \mathbf{1}'$, does not hold. Indeed, as a simple example, we have that $(\mathbf{1/2}) \vee (\mathbf{1/2})^{\sim} = (\mathbf{1/2}) \vee \mathbf{0} = (\mathbf{1/2}) \neq \mathbf{1}$. As another, less-trivial example, let us consider the fuzzy set on the universe $X = \mathbb{R}$

$$f(x) = \begin{cases} 1 & \text{if } x \in [0, 1] \\ 1/2 & \text{if } x \in (1, +\infty) \\ 0 & \text{otherwise} \end{cases} \text{then } f^{\sim}(x) = \begin{cases} 1 & \text{if } x \in (-\infty, 0) \\ 0 & \text{otherwise} \end{cases}$$

from which we have

$$(f \vee f^{\sim})(x) = \begin{cases} 1 & \text{if } x \in (-\infty, 0) \\ 1/2 & \text{otherwise} \end{cases}$$

and so $f \vee f^{\sim} \neq \mathbf{1}$.

Summarizing, both the intuitionistic-like and the fuzzy-like orthocomplementation do not satisfy the "excluded middle" law. Moreover, the intuitionistic-like negation does not satisfy the strong "double negation" law, and the "contradiction" law, too, which are substituted by the weaker Kleene condition (K). Indeed, if for a certain fuzzy set $f \wedge f' = \mathbf{0}$ and $f \vee f' = \mathbf{1}$, then trivially $f \wedge f' \leq f \vee f'$. For a complete discussion of the role of negation in intuitionistic logic or better, in its algebraic semantic, let us refer to Ref. 12.

Let us note that the negations of impossibility (i.e., the *possibility*) and contingency (i.e., the *necessity*) assume the form:

$$(4a)\quad ((f)(x) := f^{\sim\prime}(x) = \begin{cases} 1, & \text{if} \quad f(x) \neq 0 \\ 0, & \text{if} \quad f(x) = 0 \end{cases}$$

$$(4b)\quad \square(f)(x) := f^{b'}(x) = \begin{cases} 1, & \text{if} \quad f(x) = 1 \\ 0, & \text{if} \quad f(x) \neq 1 \end{cases}$$

Remark 7. If one associates with any fuzzy set f the *possibility* domain $A_p(f) := \{x \in X : f(x) \neq 0\}$ and the *necessity* domain $A_1(f) := \{x \in X : f(x) = 1\}$, then, as in the two cases of Remark 5, we have to do with two crisp sets:

$$\Box(f) = \chi A_1(f) \quad \text{and} \quad ((f) = \chi A_p(f)$$

The Lattice of Ordered-Pairs of Crisp Sets (OCS)

Note that the necessity $\Box(f)$ (resp., possibility $((f)$) is the best crisp approximation of the fuzzy set f from the bottom (resp., top): $\Box(f) \leq f \leq \ ((f)$. For this reason, in the application of roughness theory to fuzzy sets (see Ref. 12), the *rough approximation* of the fuzzy set f by crisp sets is defined as the pair of crisp sets

$$r(f) := \Box((f),((f)) = (\chi A_1(f), \chi A_p(f)) \quad \text{with} \quad \chi A_1(f) \leq \chi A_p(f)$$

This rough approximation can be identified with the pair $r(f) = (A_1(f), A_p(f))$ by the bijective correspondence $r : f \in \mathcal{F}(X) \to r(f) = (A_1(f), A_p(f)) \in \mathcal{P}(X) \times \mathcal{P}(X)$, with $A_1(f) \subseteq A_p(f)$.

These considerations lead one to treat as interesting the so-called *ordered pairs of crisp sets* (OCS) defined as pair (A_1, A_p) of subsets of the universe X under the condition that $A_1 \subseteq A_p$ whose collection will be denoted by $OC(X)$. To the best of our knowledge, the concept of OCS was introduced for the first time by M. Yves Gentilhomme in Ref. 13. (see also Ref. 14). Let us now investigate the lattice behavior of an OCS. First of all, let us introduce on $OC(X)$ the following Gentilhomme binary operations:

$$\begin{aligned}(A_1, A_p) \sqcap (B_1, B_p) &= (A_1 \cap B_1, A_p \cap B_p)\\ (A_1, A_p) \sqcup (B_1, B_p) &= (A_1 \cup B_1, A_p \cup B_p)\end{aligned}$$

Now, the structure $\langle OC(X), \sqcap, \sqcup \rangle$ is a distributive lattice whose induced partial order relation is

$$(A_1, A_p) \sqsubseteq (B_1, B_p) \quad \text{iff} \quad A_1 \subseteq B_1 \,\text{and}\, A_p \subseteq B_p$$

This lattice is bounded by the least element $\mathbf{0} = (\theta, \theta)$ and the greatest element $\mathbf{1} = (X, X)$.

As to the negation, we have the two cases

$$\begin{aligned}-(A_1, A_p) &= ((A_p)^c, (A_1)^c)\\ \sim (A_1, A_p) &= ((A_p)^c, (A_p)^c)\end{aligned}$$

The operation – is a Kleene complementation because (oc1) $-(-(A_1, A_p)) = (A_1, A_p)$; (oc2c) $(A_1, A_p) \sqsubseteq (B_1, B_p)$ implies $-(B_1, B_p) = ((B_p)^c, (B_1)^c) \sqsubseteq ((A_p)^c, (A_1)^c) = -(A_1, A_p)$. Moreover, this lattice admits the half element $\frac{1}{2} = (\emptyset, X)$ with respect to the negation —, because $-\frac{1}{2} = -(\emptyset, X) = \frac{1}{2}$. As a consequence, with respect to this complement, the lattice $OC(X)$ is not Boolean. As to the Kleene condition, let us observe that $(A_1, A_p) \sqcap -(A_1, A_p) = (\emptyset, (A_1)^c \cap A_p)$ (with $(A_1)^c \cap A_p$ the *boundary* of the OCS (A_1, A_p)), $(B_1, B_p) \sqcup -(B_1, B_p) = (B_1 \cup (B_p)^c, X)$, and so

$$(A_1, A_0) \sqcap -(A_1, A_0) \sqsubseteq (\theta, X) \sqsubseteq (B_1, B_0) \sqcup -(B_1, B_0).$$

In a similar way, it is easy to prove that the operation $\sim$ is an intuitionistic complementation, i.e., conditions (woc1)-(woc3) are all satisfied.

BIBLIOGRAPHY

1. G. Birkhoff, *Lattice Theory*, 3rd ed., American Mathematical Society Colloquium Publication, Vol. **25**. American Mathematical Society, 1967. Providence, Rhode Island; (first edition, 1940; second (revisited) edition, 1948).
2. B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, Cambridge: Cambridge University Press, 1990.
3. L. A. Zadeh, Fuzzy sets, *Information and Control* **8**: 338–353, 1965.
4. D. Dubois and H. Prade, *Fuzzy Sets and Systems. Theory and Applications*, New York: Academic Press, 1980.
5. M. Sholander, Postulates for distributive lattices, *Canadian J. Mathematics* **3**: 29–30, 1951.
6. G. Birkhoff and J. von Neumann, The logic of quantum mechanics, Annals of Mathematics, **37**: 823–843, 1936.
7. Z. Pawlak, Rough sets, *Int. J. of Computer and Information Sciences***11**: 341–356, 1982.
8. G. Cattaneo, *Abstract approximation spaces for rough theories*, in L. Polkowski and A. Skowron Rough Sets in Knowledge Discovery 1. Heidelberg: Physica-Verlag, 1998, pp. 59–98.
9. M. H. H, The theory of representation for Boolean algebras, *Trans. American Mathematical Society* **40**: 37–111, 1936.
10. G. M. M, Material implication in orthomodular (and Boolean) lattices, *Notre Dame J. Modal Logic* **22**: 163–182, 1981.
11. O. Frink, New algebras of logic, *American Mathematical Monthly* **45**: 210–219, 1938.
12. G. Cattaneo and D. Ciucci, Basic intuitionistic principles in fuzzy set theories and its extensions (a terminological debate on Atanassov IFS), *Fuzzy Sets and Systems* **157**: 3198–3219, 2006.
13. M. Y. Gentilhomme, Les ensembles flous en linguistique, *Cahiers de linguistique theoretique et applique, Bucarest* **47**: 47–65, 1968.
14. G. C. Moisil, *Les ensembles flous et la logique a trois valeurs (texte inedit)*, in *Essais sur les Logiques non Chrysippiennes*, Edition de l'Académie de la République Socialiste de Roumanie, 99–103, 1972.

Gianpiero Cattaneo
Davide Ciucci
Università di Milano–Bicocca
Milano, Italy

PROBABILITY AND STATISTICS

As an introductory definition, we consider probability to be a collection of concepts, methods, and interpretations that facilitate the investigation of nondeterministic phenomena. Although more mathematically precise definitions will appear in due course, this broadly inclusive description certainly captures the nonspecialist's meaning of probability, and it aligns closely with the intuitive appreciation shared by engineering and computer science professionals. That is, probability is concerned with chance. In a general context, probability represents a systematic attempt to cope with fluctuations of the stock market, winning and losing streaks at the gaming table, and trends announced by economists, sociologists, and weather forecasters. In more scientific contexts, these examples extend to traffic flow patterns on computer networks, request queue lengths at a database server, and charge distributions near a semiconductor junction.

Modern life, despite the predictability afforded by scientific advances and technological accomplishments, offers frequent exposure to chance happenings. It is a safe conclusion that this situation has been part of human existence from prehistoric times. Indeed, despite the associated anxiety, it seems that humans actively seek chance encounters and that they have done so from earliest times. For example, archeological evidence confirms that gambling, typically in the form of dice games, has been with us for thousands of years. Given this long history, we might expect some accumulation of knowledge, at least at the level of folklore, that purports to improve the outcomes of chance encounters. In Europe, however, no evidence of a science of gambling appears before the fifteenth century, and it is only in the middle of the seventeenth century that such knowledge starts to encompass chance phenomena outside the gaming sphere.

A science of chance in this context means a set of concepts and methods that quantifies the unpredictability of random events. For example, frequency of occurrence provides a useful guideline when betting on a roll of the dice. Suppose your science tells you, from observation or calculation, that a seven occurs about once in every 6 throws and a two occurs about once in every 36 throws. You would then demand a greater payoff for correctly predicting the next throw to be a two than for predicting a seven. Although the dice rolls may be random, a pattern applies to repeated rolls. That pattern governs the relative abundance of sevens and twos among the outcomes.

Probability as a science of chance originated in efforts to quantify such patterns and to construct interpretations useful to the gambler. Many historians associate the birth of probability with a particular event in mid-seventeenth-century France. In 1654, the Chevalier de Mère, a gambler, asked Blaise Pascal and Pierre Fermat, leading mathematicians of the time, to calculate a fair disposition of stakes when a game suffers a premature termination. They reasoned that it was unfair to award all stakes to the party who was leading at the point of interruption. Rather, they computed the outcomes for all possible continuations of the game. Each party is allocated that fraction of the outcomes that result in a win for him. This fraction constitutes the probability that the party would win if the game were completed. Hence, the party's "expectation" is that same fraction of the stakes and such should be his fair allocation in the event of an interrupted game. We note that this resolution accords with the intuition that the party who is ahead at the time of interruption should receive a larger share of the stakes. Indeed, he is "ahead" precisely because the fraction of game continuations in his favor is noticeably larger than that of his opponent.

For his role in gambling problems, such as the one described above, and for his concept of dominating strategies in decision theory, Pascal is sometimes called the father of probability theory. The title is contested, however, as there are earlier contributers. In the 1560s, Girolamo Cardano wrote a book on games of chance, and Fra Luca Paciuoli described the division of stakes problem in 1494. In any case, it is not realistic to attribute such a broad subject to a single parental figure. The mid-seventeenth-century date of birth, however, is appropriate. Earlier investigations are frequently erroneous and tend to be anecdotal collections, whereas Pascal's work is correct by modern standards and marks the beginning of a systematic study that quickly attracted other talented contributors.

In 1657, Christian Huygens introduced the concept of expectation in the first printed probability textbook. About the same time John Wallis published an algebra text that featured a discussion of combinatorics. Appearing posthumously in 1713, Jacques Bernoulli's *Ars Conjectandi* (The Art of Conjecture) presented the first limit theorem, which is today known as the Weak Law of Large Numbers. Soon thereafter, Abraham de Moivre demonstrated the normal distribution as the limiting form of binomial distributions, a special case of the Central Limit Theorem. In marked contrast with its earlier focus on gambling, the new science now found application in a variety of fields. In the 1660s, John Hudde and John de Witt provided an actuarial foundation for the annuities used by Dutch towns to finance public works. In 1662, John Graunt published the first set of statistical inferences from mortality records, a subject of morbid interest in those times when the plague was devastating London. Around 1665, Gottfried Leibniz brought probability to the law as he attempted to assess the credibility of judicial evidence.

Excepting Leibniz's concern with the credibility of evidence, all concepts and applications discussed to this point have involved patterns that appear over repeated trials of some nondeterministic process. This is the frequency-of-occurrence interpretation of probability, and it presents some dificulty when contemplating a single trial. Suppose a medical test reveals that there is a 44% chance of your having a particular disease. The frequency-of-occurrence

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

interpretation is that 44% of a large number of persons with the same test results have actually had the disease. What does this information tell you? At best, it seems to be a rough guideline. If the number were low, say 0.01%, then you might conclude that you have no medical problem. If it were high, say 99.9%, then you would likely conclude that you should undergo treatment. There is a rather large gap between the two extremes for which it is dificult to reach any conclusion. Even more tenuous are statements such as "There is a 30% chance of war if country X acquires nuclear weapons." In this case, we cannot even envision the "large number of trials" that might underlie a frequency-of-occurrence interpretation. These philosophical issues have long contributed to the uneasiness associated with probabilistic solutions. Fortunately for engineering and computer science, these questions do not typically arise. That is, probability applications in these disciplines normally admit a frequency-of-occurrence interpretation. If, for example, we use probability concepts to model query arrivals at a database input queue, we envision an operating environment in which a large number of such queries are generated with random time spacings between them. We want to design the system to accommodate most of the queries in an economic manner, and we wish to limit the consequences of the relatively rare decision to reject a request.

In any case, as probability developed a sophisticated mathematical foundation, the mathematics community took the opportunity to sidestep the entire issue of interpretations. A movement, led primarily by Andrei Kolmogorov, developed *axiomatic* probability theory. This theory defines the elements of interest, such as probability spaces, random variables, and distributions and their transforms, as abstract mathematical objects. In this setting, the theorems have specific meanings, and their proofs are the timeless proofs of general mathematics. The theorems easily admit frequency-of-occurrence interpretations, and other interpretations are simply left to the observer without further comment. Modern probability theory, meaning probability as understood by mathematicians since the first third of the twentieth century, is grounded in this approach, and it is along this path that we now begin a technical overview of the subject.

PROBABILITY SPACES

Formally, a probability space is a triple $(\Omega, \mathcal{F}, \mathcal{P})$. The first component Ω is simply a set of outcomes. Examples are $\Omega = \{\text{heads, tails}\}$ for a coin-flipping scenario, $\Omega = \{1, 2, 3, 4, 5, 6\}$ for a single-die experiment, or $\Omega = \{x : 0 \leq x < 1\}$ for a spinning pointer that comes to rest at some fraction of a full circle. The members of Ω are the occurrences over which we wish to observe quantifiable patterns. That is, we envision that the various members will appear nondeterministically over the course of many trials, but that the relative frequencies of these appearances will tend to have established values. These are values assigned by the function P to outcomes or to collections of outcomes. These values are known as *probabilities*. In the single-die experiment, for example, we might speak of $P(3) = 1/6$ as the probability of obtaining a three on a single roll. A composite outcome, such as three or four on a single roll, is the probability $P\{3, 4\} = 1/3$.

How are these values determined? In axiomatic probability theory, probabilities are externally specified. For example, in a coin-tossing context, we might know from external considerations that heads appears with relative frequency 0.55. We then simply declare $P(\text{heads}) = 0.55$ as the probability of heads. The probability of tails is then $1.0 - 0.55 = 0.45$. In axiomatic probability theory, these values are assigned by the function P, the third component of the probability space triple. That is, the theory develops under the assumption that the assignments are arbitrary (within certain constraints to be discussed shortly), and therefore any derived results are immediately applicable to any situation where the user can place meaningful frequency assignments on the outcomes.

There is, however, one intervening technical dificulty. The function P does not assign probabilities directly to the outcomes. Rather, it assigns probabilities to *events*, which are subsets of Ω. These events constitute $\mathcal{F}$, the second element of the probability triple. In the frequency-of-occurrence interpretation, the probability of an event is the fraction of trials that produces an outcome in the event. A subset in $\mathcal{F}$ may contain a single outcome from Ω, in which case this outcome receives a specific probability assignment. However, some outcomes may not occur as singleton subsets in $\mathcal{F}$. Such an outcome appears only within subsets (events) that receive overall probability assignments. Also, all outcomes of an event $E \in \mathcal{F}$ may constitute singleton events, each with probability assignment zero, whereas E itself receives a nonzero assignment. Countability considerations, to be discussed shortly, force these subtleties in the general case to avoid internal contradictions. In simple cases where Ω is a finite set, such as the outcomes of a dice roll, we can take $\mathcal{F}$ to be all subsets of Ω, including the singletons. In this case, the probability assignment to an event must be the sum of the assignments to its constituent outcomes.

Here are the official rules for constructing a probability space $(\Omega, \mathcal{F}, P)$. First, Ω may be any set whatsoever. For engineering and computer science applications, the most convenient choice is often the real numbers, but any set will do. $\mathcal{F}$ is a collection of subsets chosen from Ω. $\mathcal{F}$ must constitute a σ-*field*, which is a collection containing ϕ, the empty subset, and closed under the operations of complement and countable union. That is,

- $\phi \in \mathcal{F}$.
- $A \in \mathcal{F}$ forces $\overline{A} = \Omega - A \in \mathcal{F}$.
- $A_n \in \mathcal{F}$ for $n = 1, 2, \ldots$ forces $\cup_{n=1}^{\infty} A_n \in \mathcal{F}$.

Finally, the function P maps subsets in $\mathcal{F}$ to real numbers in the range zero to one in a countably additive manner. That is,

- $P : \mathcal{F} \rightarrow [0, 1]$.
- $P(\phi) = 0$.
- $P(\Omega) = 1$.
- $P(\cup_{n=1}^{\infty} A_n) = \sum_{n=1}^{\infty} P(A_n)$ for pairwise disjoint subsets $A_1, A_2, \ldots$.

The events in $\mathcal{F}$ are called *measurable* sets because the function P specifies their sizes as probability allotments. You might well wonder about the necessity of the σ-field $\mathcal{F}$. As noted, when Ω is a finite outcome set, $\mathcal{F}$ is normally taken to be the collection of all subsets of Ω. This maximal collection clearly satisfies the requirements. It contains all singleton outcomes as well as all possible groupings. Moreover, the rule that P must be countably additive forces the probability of any subset to be the sum of the probabilities of its members. This happy circumstance occurs in experiments with dice, coins, and cards, and subsequent sections will investigate some typical probability assignments in those cases.

A countable set is one that can be placed in one-to-one correspondence with the positive integers. The expedient introduced for finite Ω extends to include outcome spaces that are countable. That is, we can still choose $\mathcal{F}$ to be all subsets of the countably infinite Ω. The probability of a subset remains the sum of the probabilities of its members, although this sum now contains countably many terms. However, the appropriate Ω for many scientific purposes is the set of real numbers. In the nineteenth century, Georg Cantor proved that the real numbers are uncountable, as is any nonempty interval of real numbers.

When Ω is an uncountable set, set-theoretic conundrums beyond the scope of this article force $\mathcal{F}$ to be smaller than the collection of all subsets. In particular, the countable additivity requirement on the function P cannot always be achieved if $\mathcal{F}$ is the collection of all subsets of Ω. On most occasions associated with engineering or computer science applications, the uncountable Ω is the real numbers or some restricted span of real numbers. In this case, we take $\mathcal{F}$ to be the *Borel* sets, which is the smallest σ-field containing all open intervals of the form (a, b).

Consider again the example where our outcomes are interarrival times between requests in a database input queue. These outcomes can be any positive real numbers, but our instrumentation cannot measure them in infinite detail. Consequently, our interest normally lies with events of the form (a, b). The probability assigned to this interval should reflect the relative frequency with which the interarrival time falls between a and b. Hence, we do not need all subsets of the positive real numbers among our measurable sets. We need the intervals and interval combinations achieved through intersections and unions. Because $\mathcal{F}$ must be a σ-field, we must therefore take some σ-field containing the intervals and their combinations. By definition the collection of Borel sets satisfies this condition, and this choice has happy consequences in the later development of random variables.

Although not specified explicitly in the defining constraints, closure under countable intersections is also a property of a σ-field. Moreover, we may interpret countable as either finite or countably infinite. Thus, every σ-field is closed under finite unions and finite intersections. From set-theoretic arguments, we obtain the following properties of the probability assignment function P:

- $P(\overline{A}) = 1 - P(A)$.
- $P(\cup_{n=1}^{\infty} A_n) \leq \sum_{n=1}^{\infty} P(A_n)$, without regard to disjointness among the A_n.
- If $A \subseteq B$, then $P(A) \leq P(B)$.
- If $A_1 \subseteq A_2 \subseteq \ldots$, then $P(\cup_{n=1}^{\infty} A_n) = \lim_{n\to\infty} P(A_n)$.
- If $A_1 \supseteq A_2 \supseteq \ldots$, then $P(\cap_{n=1}^{\infty} A_n) = \lim_{n\to\infty} P(A_n)$.

The final two entries above are collectively known as the Continuity of Measure Theorem. A more precise continuity of measure result is possible with more refined limits. For a sequence $a_1, a_2, \ldots$ of real numbers and a sequence $A_1, A_2, \ldots$ of elements from $\mathcal{F}$, we define the limit supremum and the limit infimum as follows:

$$\begin{aligned}
\operatorname*{limsup}_{n\to\infty} a_n &= \operatorname*{limsup}_{k\to\infty n\geq k} a_n \\
\operatorname*{limsup}_{n\to\infty} A_n &= \cap_{k=1}^{\infty} \cup_{n=k}^{\infty} A_n \\
\operatorname*{liminf}_{n\to\infty} a_n &= \operatorname*{liminf}_{k\to\infty n\geq k} a_n \\
\operatorname*{liminf}_{n\to\infty} A_n &= \cup_{k=1}^{\infty} \cap_{n=k}^{\infty} A_n
\end{aligned}$$

We adopt the term extended limits for the limit supremum and the limit infimum. Although a sequence of bounded real numbers, such as probabilities, may oscillate indefinitely and therefore fail to achieve a well-defined limiting value, every such sequence nevertheless achieves extended limits. We find that $\operatorname{liminf} a_n \leq \operatorname{limsup} a_n$ in all cases, and if the sequence converges, then $\operatorname{liminf} a_n = \lim a_n = \operatorname{limsup} a_n$. For a sequence of subsets in $\mathcal{F}$, we adopt equality of the extended limits as the definition of convergence. That is, $\lim A_n = A$ if $\operatorname{liminf} A_n = \operatorname{limsup} A_n = A$.

When applied to subsets, we find $\operatorname{liminf} A_n \subseteq \operatorname{limsup} A_n$. Also, for increasing sequences, $A_1 \subseteq A_2 \subseteq \ldots$, or decreasing sequences, $A_1 \supseteq A_2 \supseteq \ldots$, it is evident that the sequences converge to the countable union and countable intersection, respectively. Consequently, the Continuity of Measure Theorem above is actually a statement about convergent sequences of sets: $P(\lim A_n) = \lim P(A_n)$. However, even if the sequence does not converge, we can derive a relationship among the probabilities: $P(\operatorname{liminf} A_n) \leq \operatorname{liminf} P(A_n) \leq \operatorname{limsup} P(A_n) \leq P(\operatorname{limsup} A_n)$.

We conclude this section with two more advanced results that are useful in analyzing sequences of events. Suppose A_n, for $n = 1, 2, \ldots$, is a sequence of events in $\mathcal{F}$ for which $\sum_{n=1}^{\infty} P(A_n) < \infty$. The Borel Lemma states that, under these circumstances, $P(\operatorname{liminf} A_n) = P(\operatorname{limsup} A_n) = 0$.

The second result is the Borel–Cantelli Lemma, for which we need the definition of independent subsets. Two subsets A, B in $\mathcal{F}$ are *independent* if $P(A \cap B) = P(A)P(B)$. Now suppose that A_i and A_j are independent for any two distinct subsets in the sequence. Then the lemma asserts that if $\sum_{n=1}^{\infty} P(A_n) = \infty$, then $P(\operatorname{limsup} A_n) = 1$. A more general result, the Kochen–Stone Lemma, provides a lower bound on $P(\operatorname{limsup} A_n)$ under slightly more relaxed conditions.

COMBINATORICS

Suppose a probability space $(\Omega, \mathcal{F}, P)$ features a finite outcome set Ω. If we use the notation $|A|$ to denote the number of elements in a set A, then this condition is $|\Omega| = n$

for some positive integer n. In this case, we take $\mathcal{F}$ to be the collection of all subsets of Ω. We find that $|\mathcal{F}| = 2^n$, and a feasible probability assignment allots probability $1/n$ to each event containing a singleton outcome. The countable additivity constraint then forces each composite event to receive probability equal to the sum of the probabilities of its constituents, which amounts to the size of the event divided by n. In this context, combinatorics refers to methods for calculating the size of Ω and for determining the number of constituents in a composite event.

In the simplest cases, this computation is mere enumeration. If Ω contains the six possible outcomes from the roll of a single die, then $\Omega = 1, 2, 3, 4, 5, 6$. We observe that $n = 6$. If an event E is described as "the outcome is odd or it is greater than 4," then we note that outcomes 1, 3, 5, 6 conform to the description, and we calculate $P(E) = 4/6$. In more complicated circumstances, neither n nor the size of E is so easily available. For example, suppose we receive 5 cards from a shuffled deck of 52 standard playing cards. What is the probability that we receive five cards of the same suit with consecutive numerical values (a straight flush)? How many possible hands exist? How many of those constitute a straight flush?

A systematic approach to such problems considers sequences or subsets obtained by choosing from a common pool of candidates. A further distinction appears when we consider two choice protocols: choosing with replacement and choosing without replacement. Two sequences differ if they differ in any position. For example, the sequence 1, 2, 3 is different from the sequence 2, 1, 3. However, two sets differ if one contains an element that is not in the other. Consequently, the sets 1, 2, 3 and 2, 1, 3 are the same set.

Suppose we are choosing k items from a pool of n without replacement. That is, each choice reduces that size of the pool available for subsequent choices. This constraint forces $k \leq n$. Let $P_{k,n}$ be the number of distincts sequences that might be chosen, and let $C_{k,n}$ denote the number of possible sets. We have

$$P_{k,n} = (n)_k^{\downarrow} \equiv n(n-1)(n-2)\cdots(n-k+1) = \frac{n!}{k!}$$
$$C_{k,n} = \binom{n}{k} \equiv \frac{n!}{k!(n-k)!}$$

Consider again the scenario mentioned above in which we receive five cards from a shuffled deck. We receive one of $(52)_5^{\downarrow} = 311875200$ sequences. To determine whether we have received a straight flush, however, we are allowed to reorder the cards in our hand. Consequently, the size of the outcome space is the number of possible sets, rather than the number of sequences. As there are $\binom{52}{5} = 2598960$ such sets, we conclude that the size of the outcome space is $n = 2598960$. Now, how many of these possible hands constitute a straight flush?

For this calculation, it is convenient to introduce another useful counting tool. If we can undertake a choice as a succession of subchoices, then the number of candidate choices is the product of the number available at each stage. A straight flush, for example, results from choosing one of four suits and then one of nine low cards to anchor the ascending numerical values. That is, the first subchoice has candidates: clubs, diamonds, hearts, spades. The second subchoice has candidates: 2, 3,. . ., 10. The number of candidate hands for a straight flush and the corresponding probability of a straight flush are then

$$\begin{aligned} N(\text{straight flush}) &= \binom{4}{1}\binom{9}{1} = 36 \\ P(\text{straight flush}) &= \frac{36}{2598960} = 0.0000138517 \end{aligned}$$

The multiplicative approach that determines the number of straight flush hands amounts to laying out the hands in four columns, one for each suit, and nine rows, one for each low card anchor value. That is, each candidate from the first subchoice admits the same number of subsequent choices, nine in this example. If the number of subsequent subchoices is not uniform, we resort to summing the values. For example, how many hands exhibit either one or two aces? One-ace hands involve a choice of suit for the ace, followed by a choice of any 4 cards from the 48 non-aces. Two-ace hands require a choice of two suits for the aces, followed by a selection of any 3 cards from the 48 non-aces. The computation is

$$\begin{aligned} N(\text{one or two aces}) &= \binom{4}{1}\binom{48}{4} + \binom{4}{2}\binom{48}{3} = 882096 \\ P(\text{one or two aces}) &= \frac{882096}{2598960} = 0.3394 \end{aligned}$$

When the selections are performed with replacement, the resulting sequences or sets may contain duplicate entries. In this case, a set is more accurately described as a multiset, which is a set that admits duplicate members. Moreover, the size of the selection k may be larger than the size of the candidate pool n. If we let $\overline{P}_{k,n}$ and $\overline{C}_{k,n}$ denote the number of sequences and multisets, respectively, we obtain

$$\begin{aligned} \overline{P}_{k,n} &= n^k \\ \overline{C}_{k,n} &= \binom{n+k-1}{k} \end{aligned}$$

These formulas are useful in occupancy problems. For example, suppose we have n bins into which we must distribute k balls. As we place each ball, we choose one of the n bins for it. The chosen bin remains available for subsequent balls, so we are choosing with replacement. A generic outcome is $(n_1, n_2,\ldots, n_k)$, where n_i denotes the bin selected for ball i. There are n^k such outcomes corresponding to the number of such sequences.

If we collect birthdays from a group of 10 persons, we obtain a sequence $n_1, n_2,\ldots, n_{10}$, in which each entry is an integer in the range 1 to 365. As each such sequence represents one choice from a field of $\overline{P}_{365,10} = 365^{10}$, we can calculate the probability that there will be at least one repetition among the birthdays by computing the number of such sequences with at least one repetition and dividing by the size of the field. We can construct a sequence with *no* repetitions by selecting, without replacement, 10 integers from the range 1 to 365. There are $P_{365,10}$ such sequences, and the remaining sequences must all correspond to least one repeated birthday among the 10 people. The probability of a repeated birthday is then

$$\begin{aligned} P(\text{repeated birthday}) &= \frac{365^{10} - P_{365,10}}{365^{10}} \\ &= 1 - \frac{(365)(364)\ldots(365)}{365^{10}} = 0.117 \end{aligned}$$

As we consider larger groups, the probability of a repeated birthday rises, although many people are surprised by how quickly it becomes larger than 0.5. For example, if we redo the above calculation with 23 persons, we obtain 0.5073 for the probability of a repeated birthday.

Multisets differ from sequences in that a multiset is not concerned with the order of its elements. In the bin-choosing experiment above, a generic multiset outcome is $k_1, k_2, \ldots, k_n$, where k_i counts the number of times bin i was chosen to receive a ball. That is, k_i is the number of occurrences of i in the generic sequence outcome $n_1, n_2, \ldots, n_k$, with a zero entered when a bin does not appear at all. In the birthday example, there are $\overline{C}_{365,10}$ such day-count vectors, but we would not consider them equally likely outcomes. Doing so would imply that the probability of all 10 birthdays coinciding is the same as the probability of them dispersing across several days, a conclusion that does not accord with experience.

As an example of where the collection of multisets correctly specifies the equally likely outcomes, consider the ways of writing the positive integer k as a sum of n nonnegative integers. A particular sum $k_1 + k_2 + \ldots + k_n = k$ is called a *partition* of k into nonnegative components. We can construct such a sum by tossing k ones at n bins. The first bin accumulates summand k_1, which is equal to the number of times that bin is hit by an incoming one. The second bin plays a similar role for the summand k_2 and so forth. There are $\overline{C}_{3,4} = 15$ ways to partition 4 into 3 components:

$$\begin{array}{lllll} 0+0+4 & 0+1+3 & 0+2+2 & 0+3+1 & 0+4+0 \\ 1+0+3 & 1+1+2 & 1+2+1 & 1+3+0 & \\ 2+0+2 & 2+1+1 & 2+2+0 & & \\ 3+0+1 & 3+1+0 & & & \\ 4+0+0 & & & & \end{array}$$

We can turn the set of partitions into a probability space by assigning probability 1/15 to each partition. We would then speak of a random partition as 1 of these 15 equally likely decompositions.

When the bin-choosing experiment is performed with distinguishable balls, then it is possible to observe the outcome $n_1, n_2, \ldots, n_k$, where n_i is the bin chosen for ball i. There are $\overline{P}_{n,k}$ such observable vectors. If the balls are not distinguishable, the outcome will not contain enough information for us to know the numbers n_i. After the experiment, we cannot locate ball i, and hence, we cannot specify its bin. In this case, we know only the multiset outcome $k_1, k_2, \ldots, k_n$, where k_i is the number of balls in bin i. There are only $\overline{C}_{n,k}$observable vectors of this latter type. In certain physics contexts, probability models with $\overline{P}_{n,k}$ equally likely outcomes accurately describe experiments with distinguishable particles across a range of energy levels (bins). These systems are said to obey *Maxwell–Boltzmann* statistics. On the other hand, if the experiment involves indistinguishable particles, the more realistic model use $\overline{C}_{n,k}$ outcomes, and the system is said to obey *Bose–Einstein* statistics.

The discussion above presents only an introduction to the vast literature of counting methods and their interrelationships that is known as combinatorics. For our purposes, we take these methods as one approach to establishing a probability space over a finite collection of outcomes.

RANDOM VARIABLES AND THEIR DISTRIBUTIONS

A random variable is a function that maps a probability space into the real numbers $\mathcal{R}$. There is a subtle constraint. Suppose $(\Omega, \mathcal{F}, P)$ is a probability space. Then $X : \Omega \to \mathcal{R}$ is a random variable if $X^{-1}(B) \in \mathcal{F}$ for all Borel sets $B \subset \mathcal{R}$. This constraint ensures that all events of the form $\{\omega \in \Omega | a < X(\omega) < b\}$ do indeed receive a probability assignment. Such events are typically abbreviated $(a < X \quad < b)$ and are interpreted to mean that the random variable (for the implicit outcome ω) lies in the interval (a, b). The laws of σ-fields then guarantee that related events, those obtained by unions, intersections, and complements from the open intervals, also receive probability assignments. In other words, X constitutes a *measurable* function from $(\Omega, \mathcal{F})$ to $\mathcal{R}$.

If the probability space is the real numbers, then the identity function is a random variable. However, for any probability space, we can use a random variable to transfer probability to the Borel sets $\mathcal{B}$ via the prescription $P'(B) = P(\{\omega \in \Omega | X(\omega) \in B\})$ and thereby obtain a new probability space $(\mathcal{R}, \mathcal{B}, P')$. Frequently, all subsequent analysis takes place in the real number setting, and the underlying outcome space plays no further role.

For any $x \in \mathcal{R}$, the function $F_X(x) = P'(X \leq x)$ is called the *cumulative distribution* of random variable X. It is frequently written $F(x)$ when the underlying random variable is clear from context. Distribution functions have the following properties:

- $F(x)$ is monotone nondecreasing.
- $\lim_{x \to -\infty} F(x) = 0$; $\lim_{x \to +\infty} F(x) = 1$.
- At each point x, F is continuous from the right and possesses a limit from the left. That is, $\lim_{y \to x^-} F(y) \leq F(x) = \lim_{y \to x^+} F(y)$.
- The set of discontinuities of F is countable.

Indeed, any function F with these properties is the distribution of some random variable over some probability space. If a nonnegative function f exists such that $F(x) = \int_{-\infty}^{x} f(t)dt$, then f is called the *density* of the underlying random variable X. Of course, there are actually many densities, if there is one, because f can be changed arbitrarily at isolated points without disturbing the integral.

Random variables and their distributions provide the opening wedge into a broad spectrum of analytical results because at this point the concepts have been quantified. In working with distributions, we are working with real-valued functions. The first step is to enumerate some distributions that prove useful in computer science and engineering applications. In each case, the underlying probability space is scarcely mentioned. After transfering probability to the Borel sets within the real numbers, all analysis takes place in a real-number setting. When the random variable takes on only a countable number of discrete values, it is traditionally described by giving the probability assigned to each of these values. When the random variable assumes a continuum of values, it is described by its density or distribution.

The *Bernoulli* random variable models experiments with two outcomes. It is an abstraction of the coin-tossing experiment, and it carries a parameter that denotes the probability of “heads.” Formally, a Bernoulli random variable X takes on only two values: 0 or 1. We say that X is a Bernoulli random variable with parameter p if $P(X = 1) = p$ and (necessarily) $P(X = 0) = 1 - p$.

The *expected value* of a discrete random variable X, denoted $E[X]$, is

$$E[X] \quad = \quad \sum_{n=1}^{\infty} t_n \cdot P(X = t_n)$$

where t_1, t_2,... enumerates the discrete values that X may assume. The expected value represents a weighted average across the possible outcomes. The *variance* of a discrete random variable is

$$\mathrm{Var}[X] = \sum_{n=1}^{\infty} (t_n - E[X])^2$$

The variance provides a summary measure of the dispersion of the X values about the expected value with low variance corresponding to a tighter clustering. For a Bernoulli random variable X with parameter p, we have $E[X] = p$ and $\mathrm{Var}[X] = p(1 - p)$.

An *indicator* random variable is a Bernoulli random variable that takes on the value 1 precisely when some other random variable falls in a prescribed set. For example, suppose we have a random variable X that measures the service time (seconds) of a customer with a bank teller. We might be particularly interested in lengthy service times, say those that exceed 120 seconds. The indicator

$$I_{(120,\infty)} = \begin{cases} 1, X > 120 \\ 0, X \leq 120 \end{cases}$$

is a Bernoulli random variable with parameter $p = P(X > 120)$.

Random variables X_1, X_2,..., X_n are *independent* if, for any Borel sets B_1, B_2,..., B_n, the probability that all n random variables lie in their respective sets is the product of the individual occupancy probabilities. That is,

$$P\left(\bigcap_{i=1}^{n} (X_i \in B_i)\right) = \prod_{i=1}^{n} P(X_i \in B_i)$$

This definition is a restatement of the concept of independent events introduced earlier; the events are now expressed in terms of the random variables. Because the Borel sets constitute a σ-field, it suffices to check the above condition on Borel sets of the form $(X \leq t)$. That is, X_1, X_2,..., X_n are independent if, for any n-tuple of real numbers t_1, t_2,...,t_n, we have

$$P\left(\bigcap_{i=1}^{n} (X_i \leq t_i)\right) = \prod_{i=1}^{n} P(X_i \leq t_i) = \prod_{i=1}^{n} F_X(t_i)$$

The sum of n independent Bernoulli random variables, each with parameter p, exhibits a *binomial* distribution. That is, if X_1, X_2,..., X_n are Bernoulli with parameter p and $Y = \sum_{i=1}^{n} X_i$, then

$$P(Y = i) \quad = \quad \binom{n}{i} p^i (1 - p)^{n-i}$$

for $i = 0, 1, 2, \ldots, n$. This random variable models, for example, the number of heads in n tosses of a coin for which the probability of a head on any given toss is p. For linear combinations of independent random variables, expected values and variances are simple functions of the component values.

$$\begin{aligned} E[a_1X_1 + a_2X_2 + \ldots] &= a_1E[X_1] + a_2E[X_2] + \ldots \\ \mathrm{Var}[a_1X_1 + a_2X_2 + \ldots] &= a_1^2\mathrm{Var}[X_1] + a_2^2\mathrm{Var}[X_2] + \ldots \end{aligned}$$

For the binomial random variable Y above, therefore, we have $E[Y] = np$ and $\mathrm{Var}[Y] = np(1 - p)$.

A *Poisson* random variable X with parameter λ has $P(X = k) = e^{-\lambda}\lambda^k/k!$. This random variable assumes all nonnegative integer values, and it is useful for modeling the number of events occurring in a specified interval when it is plausible to assume that the count is proportional to the interval width in the limit of very small widths. Specifically, the following context gives rise to a Poisson random variable X with parameter λ. Suppose, as time progresses, some random process is generating events. Let $X_{t,\Delta}$ count the number of events that occur during the time interval $[t, t + \Delta]$. Now, suppose further that the generating process obeys three assumptions. The first is a homogeneity constraint:

- $P(X_{t_1,\,\Delta} = k) = P(X_{t_2,\,\Delta} = k)$ for all integers $k \geq 0$

That is, the probabilities associated with an interval of width do not depend on the location of the interval. This constraint allows a notational simplification, and we can now speak of X_Δ because the various random variables associated with different anchor positions t all have the same distribution. The remaining assumptions are as follows:

- $P(X_\Delta = 1) = \lambda\Delta + o_1(\Delta)$
- $P(X_\Delta > 1) = o_2(\Delta)$

where the $o_i(\Delta)$ denote anonymous functions with the property that $o_i(\Delta)/\Delta \rightarrow 0$ as $\Delta \rightarrow 0$. Then the assignment $P(X = k) = \lim_{\Delta\rightarrow 0} P(X_\Delta = k)$ produces a Poisson random variable.

This model accurately describes such diverse phenomena as particles emitted in radioactive decay, customer arrivals at an input queue, flaws in magnetic recording media, airline accidents, and spectator coughing during a concert. The expected value and variance are both λ. If we consider a sequence of binomial random variables $B_{n,p}$, where the parameters n and p are constrained such that $n \rightarrow \infty$ and $p \rightarrow 0$ in a manner that allows $np \rightarrow \lambda > 0$, then the distributions approach that of a Poisson random variable Y with parameter λ. That is, $P(B_{n,p} = k) \rightarrow P(Y = k) = e^{-\lambda}\lambda^k/k!$.

A *geometric* random variable X with parameter p exhibits $P(X = k) = p(1-p)^k$ for $k = 0, 1, 2, \ldots$. It models, for example, the number of tails before the first head in repeated tosses of a coin for which the probability of heads is p. We have $E[X] = (1-p)/p$ and $\mathrm{Var}[X] = (1-p)/p^2$. Suppose, for example, that we have a hash table in which j of the N addresses are unoccupied. If we generate random address probes in search of an unoccupied slot, the probability of success is j/N for each probe. The number of failures prior to the first success then follows a geometric distribution with parameter j/N.

The sum of n independent geometric random variables displays a *negative binomial* distribution. That is, if X_1, $X_2, \ldots, X_n$ are all geometric with parameter p, then $Y = X_1 + X_2 + \ldots + X_n$ is negative binomial with parameters (n, p). We have

$$P(Y = k) = C_{n+k-1,k}\, p^n (1-p)^k$$

$$E[Y] = \frac{n(1-p)}{p}$$

$$\mathrm{Var}[Y] = \frac{n(1-p)}{p^2}$$

where $C_{n+k-1,k}$ is the number of distinct multisets available when choosing k from a field of n with replacement. This random variable models, for example, the number of tails before the nth head in repeated coin tosses, the number of successful fights prior to the nth accident in an airline history, or the number of defective parts chosen (with replacement) from a bin prior to the nth functional one. For the hash table example above, if we are trying to fill n unoccupied slots, the number of unsuccessful probes in the process will follow a negative binomial distribution with parameters $n, j/N$. In this example, we assume that n is significantly smaller than N, so that insertions do not materially change the probability j/N of success for each address probe.

Moving on to random variables that assume a continuum of values, we describe each by giving its density function. The summation formulas for the expected value and variance become integrals involving this density. That is, if random variable X has density f, then

$$E[X] = \int_{-\infty}^{\infty} t\, f(t)dt$$

$$\mathrm{Var}[X] = \int_{-\infty}^{\infty} [t - E[X]]^2\, f(t)dt$$

In truth, precise work in mathematical probability uses a generalization of the familiar Riemann integral known as a *measure-theoretic* integral. The separate formulas, summation for discrete random variables and Riemann integration against a density for continuous random variables, are then subsumed under a common notation. This more general integral also enables computations in cases where a density does not exist. When the measure in question corresponds to the traditional notion of length on the real line, the measure-theoretic integral is known as the *Lebesgue* integral. In other cases, it corresponds to a notion of length accorded by the probability distribution: $P(a < X < t)$ for real a and b. In most instances of interest in engineering and computer science, the form involving ordinary integration against a density suffices.

The *uniform* random variable U on [0, 1] is described by the constant density $f(t) = 1$ for $0 \leq t \leq 1$. The probability that U falls in a subinterval (a, b) within [0, 1] is simply $b - a$, the length of that subinterval. We have

$$E[U] = \int_0^1 t dt = \frac{1}{2}$$

$$\mathrm{Var}[U] = \int_0^1 \left(t - \frac{1}{2}\right)^2 dt = \frac{1}{12}$$

The uniform distribution is the continuous analog of the equally likely outcomes discussed in the combinatorics section above. It assumes that any outcome in the interval [0, 1] is equally likely to the extent possible under the constraint that probability must now be assigned to Borel sets. In this case, all individual outcomes receive zero probability, but intervals receive probability in proportion to their lengths. This random variable models situations such as the final resting position of a spinning pointer, where no particular location has any apparent advantage.

The most famous continuous random variable is the *Gaussian* or *normal* random variable $Z_{\mu,\sigma}$. It is characterized by two parameters, μ and σ, and has density, expected value, and variance as follows:

$$f_{z_{\mu,\sigma}}(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{(t-\mu)^2/2\sigma^2}$$

$$E[Z_{\mu,\sigma}] = \mu$$

$$\mathrm{Var}[\mathrm{Z}_{\mu,\sigma}] = \sigma^2$$

The well-known *Central Limit Theorem* states that the average of a large number of independent observations behaves like a Gaussian random variable. Specifically, if $X_1, X_2, \ldots$ are independent random variables with identical finite-variance distributions, say $E[X_i] = a$ and $\mathrm{Var}[X_i] = c^2$, then for any t,

$$\lim_{n\to\infty} P\left(\frac{1}{\sqrt{nc^2}}\sum_{i=1}^{n}(X_i - a) \le t\right) = P(Z_{0,1} \le t)$$

For example, if we toss a fair coin 100 times, what is the probability that we will see 40 or fewer heads? To use the Central Limit Theorem, we let $X_i = 1$ if heads occurs on the ith toss and zero otherwise. With this definition, we have $E[X_i] = 0.5$ and $\mathrm{Var}[X_i] = 0.25$, which yields

$$P\left(\sum_{i=1}^{100} X_i \le 40\right) = P\left(\frac{1}{\sqrt{100(0.25)}}\sum_{i=1}^{100}(X_i - 0.5) \le \frac{40 - 100(0.5)}{\sqrt{100(0.25)}}\right)$$
$$\approx P(Z_{0,1} \le -2) = 0.0288$$

The last equality was obtained from a tabulation of such values for the *standard* normal random variable with expected value 0 and variance 1.

Because it represents a common limiting distribution for an average of independent observations, the Gaussian random variable is heavily used to calculate *confidence intervals* that describe the chance of correctly estimating a parameter from multiple trials. We will return to this matter in a subsequent section.

The *Gamma* function is $\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx$, defined for $t < 0$. The Gamma random variable X with parameters (γ, λ) (both positive) is described by the density

$$f(x) = \begin{cases} \lambda^\gamma x^{\gamma-1}e^{-\lambda x}/\Gamma(\gamma), & \text{for } x \ge 0 \\ 0, & \text{for } x < 0 \end{cases}$$

It has $E[X] = \gamma/\lambda$ and $\mathrm{Var}[X] = \gamma/\lambda^2$. For certain specific values of γ, the random variable is known by other names. If $\gamma = 1$, the density reduces to $f(x) = \lambda e^{-\lambda x}$ for $x \le 0$, and X is then called an *exponential* random variable. The exponential random variable models the interarrival times associated with events such as radioactive decay and customer queues, which were discussed in connnection with Poisson random variables above. Specifically, if a Poisson random variable with parameter λT models the number of events in interval T, then an exponential random variable with parameter λ models their interarrival times. Consequently, the exponential random variable features prominently in queueing theory.

Exponential random variables possess a remarkable feature; they are *memoryless*. To understand this concept, we must first define the notion of *conditional probability*. We will use the exponential random variable as an example, although the discussion applies equally well to random variables in general. Notationally, we have a probability space $(\Omega, \mathcal{F}, P)$ and a random variable X, for which

$$P\{\omega \in \Omega : X(\omega) > t\} = \int_t^\infty \lambda e^{-\lambda x}dx = e^{-\lambda t}$$

for $t \le 0$. Let t_1 be a fixed positive real number, and consider a related probability space $(\hat{\Omega}, \hat{\mathcal{F}}, \hat{P})$, obtained as follows:

$$\hat{\Omega} = \{\omega \in \Omega : X(\omega) > t_1\}$$
$$\hat{\mathcal{F}} = \{A \cap \hat{\Omega} : A \in \mathcal{F}\}$$
$$\hat{P}(B) = P(B)/P(\hat{\Omega}), \text{ for all } \quad B \in \hat{\mathcal{F}}$$

By restricting its domain, we can consider X to be a random variable on $\hat{\Omega}$. For any $\omega \in \hat{\Omega}$, we have $X(\omega) > t_1$, but we can legitimately ask the probability, using the new measure $\hat{P}$, that $X(\omega)$ exceeds t_1 by more than t.

$$\begin{aligned}\hat{P}(X > t_1 + t) &= \frac{P(X > t_1 + t_2)}{P(X > t_1)} = \frac{e^{-\lambda(t_1+t)}}{e^{-\lambda t_1}} \\ &= e^{-\lambda t} = P(X > t)\end{aligned}$$

The probability $\hat{P}(B)$ is known as the *conditional probability* of B, given $\hat{\Omega}$. From the calculation above, we see that the conditional probability that X exceeds t_1 by t or more, given that $X > t_1$ is equal to the unconditional probability that $X > t$. This is the memoryless property. If X is an exponential random variable representing the time between query arrivals to a database input queue, then the probability that 6 microseconds or more elapses before the next arrival is the same as the probability that an additional 6 microseconds or more elapses before the next arrival, given that we have already waited in vain for 10 microseconds.

In general, we can renormalize our probability assignments by restricting the outcome space to some particular event, such as the $\hat{\Omega}$ in the example. The more general notation is $P(B \mid A)$ for the conditional probability of B given A. Also, we normally allow B to be any event in the original $\mathcal{F}$ with the understanding that only that part of B that intersects A carries nonzero probability under the new measure. The definition requires that the conditioning event A have nonzero probability. In that case,

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

specifies the revised probabilities for all events B. Note that

$$P(B|A) = \frac{P(A \cap B)}{P(A)} = \frac{P(A \cap B)}{P(A \cap B) + P(A \cap \overline{B})}$$
$$= \frac{P(A|B)P(B)}{P(A|B)P(B) + P(A|\overline{B})P(\overline{B})}$$

This formula, a simple form of *Bayes' Law*, relates the conditional probability of B given A to that of A given B. It finds frequent use in updating probability assignments to reflect new information. Specifically, suppose we know $P(B)$ and therefore $P(\overline{B}) = 1 - P(B)$. Such probabilities are called *prior* probabilities because they reflect the chances of a B occurrence in the absence of further knowledge about the underlying random process. If the actual outcome remains unknown to us, but we are told that event A has occurred, we may want to update our probability assignment to reflect more accurately the chances that B has also occurred. That is, we are interested in the *posterior* probability $P(B\,|\,A)$. Bayes' Law allows us to compute this new value, provided we also have the reverse conditional probabilities.

For example, suppose a medical test for a specific disease is applied to a large population of persons known to have the disease. In 99% of the cases, the disease is detected. This is a conditional probability. If we let S be the event "person is sick" and "+" be the event "medical test was positive," we have $P(+|S) = 0.99$ as an empirical estimate. Applying the test to a population of persons known not to have the disease might reveal $P(+|\overline{S}) = 0.01$ as a false alarm rate. Suppose further that the fraction $P(S) = 0.001$ of the general population is sick with the disease. Now, if you take the test with positive results, what is the chance that you have the disease? That is, what is $P(S|+)$? Applying Bayes' Law, we have

$$\begin{aligned} P(S|+) &= \frac{P(+|S)P(S)}{P(+|S)P(S) + P(+|\overline{S})P(\overline{S})} \\ &= \frac{0.99(0.001)}{0.99(0.001) + 0.01(0.99)} = 0.0909 \end{aligned}$$

You have only a 9% chance of being sick, despite having scored positive on a test with an apparent 1% error rate. Nevertheless, your chance of being sick has risen from a prior value of 0.001 to a posterior value of 0.0909. This is nearly a hundredfold increase, which is commensurate with the error rate of the test.

The full form of Bayes' Law uses an arbitrary partition of the outcome space, rather than a simple two-event decomposition, such as "sick" and "not sick." Suppose the event collection $\{A_i : 1 \leq i \leq n\}$ is a partition of the outcome space Ω. That is, the A_i are disjoint, each has nonzero probability, and their union comprises all of Ω. We are interested in which A_i has occurred, given knowledge of another event B. If we know the reverse conditional probabilities, that is if we know the probability of B given each A_i, then Bayes' Law enables the computation

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{n} P(B|A_j)P(A_j)}.$$

Returning to the Gamma random variable with parameters (γ, λ), we can distinguish additional special cases. If $\gamma = n$, a positive integer, then the corresponding distribution is called an *Erlang* distribution. It models the time necessary to accumulate n events in a process that follows a Poisson distribution for the number of events in a specified interval. An Erlang distribution, for example, describes the time span covering the next n calls arriving at a telephone exchange.

If $\gamma = n/2$ for a positive integer n and $\lambda = 1/2$, then the corresponding Gamma random variable is called a chi-square random variable. It exhibits the distribution of the sum of n independent squares, $Y = \sum_{i=1}^{n} Z_i^2$, where each Z_i is a Gaussian random variable with $(\mu, \sigma^2) = (0, 1)$. These distributions are useful in computing confidence intervals for statistical estimates.

Gamma distributions are the limiting forms of negative binomial random variables in the same manner that the Poisson distribution is the limit of binomials. That is, suppose we have a sequence C_n of negative binomial random variables. The parameters of C_n are (m, p_n). As $n \to \infty$, we assume that $p_n \to 0$ in a manner that allows $n p_n \to \lambda > 0$. Then the limiting distribution of C_n/n is the Gamma (Erlang) distribution with parameters (m, γ). In particular, if $m = 1$, the C_n are geometric and the limit is exponential.

Figure 1 summarizes the relationships among the random variables discussed in this section.

The *renewal count* arrow from exponential to Poisson refers to the fact that a phenomenon in which the event interarrival time is exponential (λ) will accumulate events in an interval T according to a Poisson distribution with parameter λT . That is, if the sequence X_1, X_2,... of random variables measures time between successive events, then the random variable

$$N_T = \max\left\{ k | \sum_{i=1}^{k} X_i \leq T \right\}$$

is called a renewal count for the sequence. If the X_i are independent exponentials with parameter λ, then N_T has a Poisson distribution with parameter λT.

A similar relationship holds between a sequence $G_1 + 1$, $G_2 + 1$,... of geometric random variables with a common parameter p. The difference is that the observation interval T is now a positive integer. The renewal count N_T then exhibits a binomial distribution with parameters (T, p).

CONVERGENCE MODES

For a sequence of real numbers, there is a single mode of convergence: A tail of the sequence must enter and remain within any given neighborhood of the limit. This property either holds for some (possibly infinite) limiting value or it does not. Sequences of random variables exhibit more variety in this respect. There are three modes of convergence.

A sequence X_1, X_2,... of random variables converges *pointwise* to a random variable Y if $X_n(\omega) \to Y(\omega)$ as a sequence of real numbers for every point ω in the underlying probability space. We may also have pointwise convergence on sets smaller than the full probability space. If pointwise convergence occurs on a set of probability one, then we say that the sequence converges *almost surely*. In this case, we use the notation $X_n \to Ya.s.$

The sequence converges *in probability* if, for every positive ϵ, the measure of the misbehaving sets approaches zero. That is, as $n \to \infty$,

$$P(\{\omega : |X_n(\omega) - Y(\omega)| > \epsilon\}) \to 0$$

If $X_n \to Y$ *a.s*, then it also converges in probability. However, it is possible for a sequence to converge in probability and at the same time have no pointwise limits.

The final convergence mode concerns distribution functions. The sequence converges *in distribution* if the corresponding cumulative distribution functions of the X_n converge pointwise to the distribution function of Y at all points where the cumulative distribution function of Y is continuous.

The *Weak Law of Large Numbers* states that the average of a large number of independent, identically distributed random variables tends in probability to a constant, the expected value of the common distribution. That is, if X_1, X_2,... is an independent sequence with a common distribution such that $E[X_n] = \mu$ and $\mathrm{Var}[X_n] = \sigma^2 < \infty$, then for every positive ϵ,

$$P\left(\left\{\omega : \left|\frac{\sum_{i=1}^{n} X_i}{n} - \mu\right| > \epsilon\right\}\right) \to 0$$

as $n \to \infty$.

Suppose, for example, that a random variable T measures the time between query requests arriving at a database server. This random variable is likely to exhibit an exponential distribution, as described in the previous section, with some rate parameter λ. The expected value and variance are $1/\lambda$ and $1/\lambda^2$, respectively. We take n observations of T and label them T_1, T_2,... , T_n. The weak law suggests that the number $\sum_{i=1}^{n} T_i/n$ will be close to $1/\lambda$. The precise meaning is more subtle. As an exponential random variable can assume any nonnegative value, we can imagine a sequence of observations that are all larger than, say, twice the expected value. In that case, the average would also be much larger than $1/\lambda$. It is then clear that not all sequences of observations will produce averages close to $1/\lambda$. The weak law states that the set of sequences that misbehave in this fashion is not large, when measured in terms of probability.

We envision an infinite sequence of independent database servers, each with its separate network of clients. Our probability space now consists of outcomes of the form $\omega = (t_1, t_2, \ldots)$, which occurs when server 1 waits t_1 seconds for its next arrival, server 2 waits t_2 seconds, and so forth. Any event of the form $(t_1 \le x_1, t_2 \le x_2, \ldots, t_p \le x_p)$ has probability equal to the product of the factors $P(t_1 \le x_i)$, which are in turn determined by the common exponential distribution of the T_i. By taking unions, complements, and intersections of events of this type, we arrive at a σ-field that supports the probability measure. The random variables $\sum_{i=1}^{n} T_i/n$ are well defined on this new probability space, and the weak law asserts that, for large n, the set of sequences $(t_1, t_2, \ldots)$ with misbehaving prefixes $(t_1, t_2, \ldots, t_n)$ has small probability.

A given sequence can drift into and out of the misbehaving set as n increases. Suppose the average of the first 100 entries is close to $1/\lambda$, but the next 1000 entries are all larger than twice $1/\lambda$. The sequence is then excluded from the misbehaving set at $n = 100$ but enters that set before $n = 1100$. Subsequent patterns of good and bad behavior can migrate the sequence into and out of the exceptional set. With this additional insight, we can interpret more precisely the meaning of the weak law.

Suppose $1/\lambda = 0.4$. We can choose $\epsilon = 0.04$ and let $Y_n = \sum_{i=1}^{n} T_i/n$. The weak law asserts that $P(|Y_n - 0.4| > 0.04) \to 0$, which is the same as $P(0.36 \le Y_n \le 0.44) \to 1$. Although the law does not inform us about the actual size of n required, it does say that eventually this latter probability exceeds 0.99. Intuitively, this means that if we choose a large n, there is a scant 1% chance that our average will fail to fall close to 0.4. Moreover, as we choose larger and larger values for n, that chance decreases.

The *Strong Law of Large Numbers* states that the average converges pointwise to the common expected value, except perhaps on a set of probability zero. specifically, if X_1, X_2,... is is an independent sequence with a common distribution such that $E[X_n] = \mu$ (possibly infinite), then

$$\frac{\sum_{i=1}^{n} X_i}{n} \to \mu \quad a.s.$$

as $n \to \infty$.

Applied in the above example, the strong law asserts that essentially all outcome sequences exhibit averages that draw closer and closer to the expected value as n increases. The issue of a given sequence forever drifting into and out of the misbehaving set is placed in a pleasant perspective. Such sequences must belong to the set with probability measure zero. This reassurance does not mean that the exceptional set is empty, because individual outcomes $(t_1, t_2, \ldots)$ have zero probability. It does mean that we can expect, with virtual certainty, that our average of n observations of the arrival time will draw ever closer to the expected $1/\lambda$.

Although the above convergence results can be obtained with set-theoretic arguments, further progress is greatly facilitated with the concept of characteristic functions, which are essentially Fourier transforms in a probability space setting. For a random variable X, the *characteristic function* of X is the complex-valued function $\beta_X(u) = E[e^{iuX}]$. The exceptional utility of this device follows because there is a one-to-one relationship between characteristic functions and their generating random variables (distributions). For example, X is Gaussian with parameters $\mu = 0$ and $\sigma^2 = 1$ if and only if $\beta_X(u) = e^{u^2/2}$. X is Poisson with parameter λ if and only if $\beta_X(u) = \exp(-\lambda(1 - e^{iu}))$.

If X has a density $f(t)$, the computation of β_X is a common integration: $\beta_X(u) = \int_{-\infty}^{\infty} e^{iut} f(t)\mathrm{d}t$. Conversely, if β_X is absolutely integrable, then X has a density, which can be recovered by an inversion formula. That is, if $\int_{-\infty}^{\infty} |\beta(u)|\, du < \infty$, then the density of X is

$$f_X(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-iut} \beta(u) du$$

These remarks have parallel versions if X is integer-valued. The calculation of β_X is a sum: $\beta_X(u) = \sum_{n=-\infty}^{\infty} e^{iun}P(X = n)$. Also, if β_X is periodic with period 2π, then the corresponding X is integer-valued and the point probabilities are recovered with a similar inversion formula:

$$P(X = n) \quad = \quad \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{-iun}\beta(u)du$$

In more general cases, the β_X computation requires the measure-theoretic integral referenced earlier, and the recovery of the distribution of X requires more complex operations on β_X. Nevertheless, it is theoretically possible to translate in both directions between distributions and their characteristic functions.

Some useful properties of characteristic functions are as follows:

- (Linear combinations) If $Y = aX + b$, then $\beta_Y(u) = e^{iub}\beta_X(au)$.
- (Independent sums) If $Y = X_1 + X_2 + \ldots + X_n$, where the X_i are independent, then $\beta_Y(u) = \prod_{i=1}^{n} \beta_{X_i}(u)$.
- (Continuity Theorem) A sequence of random variables $X_1, X_2, \ldots$ converges in distribution to a random variable X if and only if the corresponding characteristic functions converge pointwise to a function that is continuous at zero; in which case, the limiting function is the characteristic function of X.
- (Moment Theorem) If $E[|X|^n] < \infty$, then β_X has derivatives through order n and $E[X^n] = (-i)^n \beta_X^{(n)}(0)$, where $\beta_X^{(n)}(u)$ is the n^{th} derivative of β_X.

These features allow us to study convergence in distribution of random variables by investigating the more tractable pointwise convergence of their characteristic functions. In the case of independent, identically distributed random variables with finite variance, this method leads quickly to the Central Limit Theorem cited earlier.

For a nonnegative random variable X, the *moment generating function* $\phi_X(u)$ is less difficult to manipulate. Here $\phi_X(u) = E[e^{-uX}]$. For a random variable X that assumes only nonnegative integer values, the *probability generating function* $\rho_X(u)$ is another appropriate transform. It is defined by $\rho_X(u) = \sum_{n=0}^{\infty} P(X = n)u^n$. Both moment and probability generating functions admit versions of the moment theorem and the continuity theorem and are therefore useful for studying convergence in the special cases where they apply.

COMPUTER SIMULATIONS

In various programming contexts, particularly with simulations, the need arises to generate samples from some particular distribution. For example, if we know that $P(X = 1) = 0.4$ and $P(X = 0) = 0.6$, we may want to *realize* this random variable as a sequence of numbers $x_1, x_2, \ldots$. This sequence should exhibit the same variability as would the original X if outcomes were directly observed. That is, we expect a thorough mixing of ones and zeros, with about 40% ones. Notice that we can readily achieve this result if we have a method for generating samples from a uniform distribution U on [0, 1]. In particular, each time we need a new sample of X, we generate an observation U and report $X = 1$ if $U \leq 0.4$ and $X = 0$ otherwise.

This argument generalizes in various ways, but the gist of the extension is that essentially any random variable can be sampled by first sampling a uniform random variable and then resorting to some calculations on the observed value. Although this reduction simplifies the problem, the necessity remains of simulating observations from a uniform distribution on [0, 1]. Here we encounter two dificulties. First, the computer operates with a finite register length, say 32 bits, which means that the values returned are patterns from the 2^{32} possible arrangements of 32 bits. Second, a computer is a deterministic device.

To circumvent the first problem, we put a binary point at the left end of each such pattern, obtaining 2^{32} evenly spaced numbers in the range [0, 1). The most uniform probability assignment allots probability $1/2^{32}$ to each such point. Let U be the random variable that operates on this probability space as the identity function. If we calculate $P(a < U < b)$ for subintervals (a, b) that are appreciably wider than $1/2^{32}$, we discover that these probabilities are nearly $b - a$, which is the value required for a true uniform random variable. The second dificulty is overcome by arranging that the values returned on successive calls exhaust, or very nearly exhaust, the full range of patterns before repeating. In this manner, any deterministic behavior is not observable under normal use. Some modern supercomputer computations may involve more than 2^{32} random samples, an escalation that has forced the use of 64-bit registers to maintain the appearance of nondeterminism.

After accepting an approximation based on 2^{32} (or more) closely spaced numbers in [0, 1), we still face the problem of simulating a discrete probability distribution on this finite set. This problem remains an area of active research today. One popular approach is the *linear congruential* method. We start with a seed sample x_0, which is typically obtained in some nonreproducible manner, such as extracting a 32-bit string from the computer real-time clock. Subsequent samples are obtained with the recurrence $x_{n+1} = (ax_n + b) \bmod c$, where the parameters a, b, c are chosen to optimize the criteria of a long period before repetition and a fast computer implementation. For example, c is frequently chosen to be 2^{32} because the $(ax_n + b) \bmod 2^{32}$ operation involves retaining only the least significant 32 bits of $(ax_n + b)$. Knuth (1) discusses the mathematics involved in choosing these parameters.

On many systems, the resulting generator is called rand(). A program assignment statement, such as $x = \mathrm{rand}()$, places a new sample in the variable x. From this point, we manipulate the returned value to simulate samples from other distributions. As noted, if we wish to sample B, a Bernoulli random variable with parameter p, we continue by setting $B = 1$ if $x \leq p$ and $B = 0$ otherwise. If we need an random variable $U_{a,b}$, uniform on the interval $[a, b]$, we calculate $U_{a,b} = a + (b - a)x$.

If the desired distribution has a continuous cumulative distribution function, a general technique, called distribution inversion, provides a simple computation of samples. Suppose X is a random variable for which the cumulative distribution $F(t) = P(X \leq t)$ is continuous and strictly increasing. The inverse $F^{-1}(u)$ then exists for $0 < u < 1$, and it can be shown that the derived random variable $Y = F(X)$ has a uniform distribution on (0, 1). It follows that the distribution of $F^{-1}(U)$ is the same as that of X, where U is the uniform random variable approximated by rand(). To obtain samples from X, we sample U instead and return the values $F^{-1}(U)$.

For example, the exponential random variable X with parameter λ has the cumulative distribution function $F(t) = 1 - e^{-\lambda t}$, for $t \geq 0$, which satisfies the required conditions. The inverse is $\mathcal{F}^{-1}(u) = -[\log(1-u)]/\lambda$. If U is uniformly distributed, so is $1 - U$. Therefore, the samples obtained from successive $-[\log(\text{rand}())]/\lambda$ values exhibit the desired exponential distribution.

A variation is necessary to accommodate discrete random variables, such as those that assume integer values. Suppose we have a random variable X that assumes nonnegative integer values n with probabilities p_n. Because the cumulative distribution now exhibits a discrete jump at each integer, it no longer possesses an inverse. Nevertheless, we can salvage the idea by acquiring a rand() sample, say x, and then summing the p_n until the accumulation exceeds x. We return the largest n such that $\sum_{i=0}^{n} p_i \leq x$. A moment's reflection will show that this is precisely the method we used to obtain samples from a Bernoulli random variable above.

For certain cases, we can solve for the required n. For example, suppose X is a geometric random variable with parameter p. In this case, $p_n = p(1-p)^n$. Therefore if x is the value obtained from rand(), we find

$$\max\left\{ n : \sum_{k=0}^{n} p_k \leq x \right\} = \left\lfloor \frac{\log_x}{\log(1-p)} \right\rfloor$$

For more irregular cases, we may need to perform the summation. Suppose we want to sample a Poisson random variable X with parameter λ. In this case, we have $p_n = e^{-\lambda}\lambda^n/n!$, and the following pseudocode illustrates the technique. We exploit the fact that $p_0 = e^{-\lambda}$ and $p_{n+1} = p_n\lambda/(n+1)$.

```
$x$ = rand();
$p = \exp(-\lambda)$;
cum = $p$;
$n = 0$;
while ($x >$ cum){
    $n = n + 1$;
    $p = p * \lambda/(n + 1)$;
    cum = cum + $p$;}
return $n$;
```

Various enhancements are available to reduce the number of iterations necessary to locate the desired n to return. In the above example, we could start the search near $n = \lfloor \lambda \rfloor$, because values near this expected value are most frequently returned.

Another method for dealing with irregular discrete distributions is the *rejection filter*. If we have an algorithm to simulate distribution X, we can, under certain conditions, systematically withhold some of the returns to simulate a related distribution Y. Suppose X assumes nonnegative integer values with probabilities $p_0, p_1, \ldots$, Y assumes the same values but with different probabilities $q_0, q_1, \ldots$. The required condition is that a positive K exists such that $q_n \leq K p_n$ for all n. The following pseudocode shows how to reject just the right number of X returns so as to correctly adjust the return distribution to that of Y. Here the routine X() refers to the existing algorithm that returns nonnegative integers according to the X distribution. We also require that the p_n be nonzero.

```
while (true) {
    $n$ = X();
    $x$ = rand();
    if ($x < q_n/(K * p_n)$)
        return $n$;}
```

STATISTICAL INFERENCE

Suppose we have several random variables $X, Y, \ldots$ of interest. For example, X might be the systolic blood pressure of a person who has taken a certain drug, whereas Y is the blood pressure of an individual who has not taken it. In this case, X and Y are defined on different probability spaces. Each probability space is a collection of persons who either have or have not used the drug in question. X and Y then have distributions in a certain range, say [50, 250], but it is not feasible to measure X or Y at each outcome (person) to determine the detailed distributions. Consequently, we resort to samples. That is, we observe X for various outcomes by measuring blood pressure for a subset of the X population. We call the observations $X_1, X_2, \ldots, X_n$. We follow a similar procedure for Y if we are interested in comparing the two distributions. Here, we concentrate on samples from a single distribution.

A sample from a random variable X is actually another random variable. Of course, after taking the sample, we observe that it is a specific number, which hardly seems to merit the status of a random variable. However, we can envision that our choice is just one of many parallel observations that deliver a range of results. We can then speak of events such as $P(X_1 \leq t)$ as they relate to the disparate values obtained across the many parallel experiments as they make their first observations. We refer to the distribution of X as the *population* distribution and to that of X_n as the nth *sample* distribution. Of course, $P(X_n \leq t) = P(X \leq t)$ for all n and t, but the term sample typically carries the implicit understanding that the various X_n are independent. That is, $P(X_n \leq t_1, \ldots, X_n \leq t_n) = \prod_{i=1}^{n} P(X_i \leq t_i)$. In this case, we say that the sample is a *random* sample.

With a random sample, the X_n are independent, identically distributed random variables. Indeed, each has the same distribution as the underlying population X. In practice, this property is assured by taking precautions to avoid any selection bias during the sampling. In the blood pressure application, for example, we attempt to choose persons in a manner that gives every individual the same chance of being observed.

Armed with a random sample, we now attempt to infer features of the unknown distribution for the population X. Ideally, we want the cumulative distribution of $F_X(t)$, which announces the fraction of the population with blood pressures less than or equal to t. Less complete, but still valuable, information lies with certain summary features, such as the expected value and variance of X.

A *statistic* is simply a function of a sample. Given the sample $X_1, X_2, \ldots, X_n$,, the new random variables

$$\overline{X} = \frac{1}{n}\sum_{k=1}^{n} X_k$$

$$S^2 = \frac{1}{n-1}\sum_{k=1}^{n}(X_k - \overline{X})^2$$

are statistics known as the *sample mean* and *sample variance* respectively. If the population has $E[X] = \mu$ and $\mathrm{Var}[X] = \sigma^2$, then $E[\overline{X}] = \mu$ and $E[S^2] = \sigma^2$. The expected value and variance are called*parameters* of the population, and a central problem in statistical inference is to estimate such unknown parameters through calculations on samples. At any point we can declare a particular statistic to be an *estimator* of some parameter. Typically we only do so when the value realized through samples is indeed an accurate estimate.

Suppose θ is some parameter of a population distribution X. We say that a statistic Y is an *unbiased* estimator of θ if $E[Y] = \theta$. We then have that the sample mean and sample variance are unbiased estimators of the population mean and variance. The quantity

$$\hat{S}^2 = \frac{1}{n}\sum_{k=1}^{n}(X_k - \overline{X})^2$$

is also called the sample variance, but it is a *biased* estimator of the population variance σ^2. If context is not clear, we need to refer to the biased or unbiased sample variance. In particular $E[\hat{S}^2] = \sigma^2(1 - 1/n)$, which introduces a bias of $b = -\sigma^2/n$. Evidently, the bias decays to zero with increasing sample size n. A sequence of biased estimators with this property is termed *asymptotically unbiased*.

A statistic can be a vector-valued quantity. Consequently, the entire sample $(X_1, X_2, \ldots, X_n)$ is a statistic. For any given t, we can compute the fraction of the sample values that is less than or equal to t. For a given set of t values, these computation produce a *sample distribution function*:

$$F_{n}(t) = \frac{\#\{k : X_k \le t\}}{n}$$

Here we use #{. . .} to denote the size of a set. For each t, the Glivenko–Cantelli Theorem states that the $F_n(t)$ constitute an asymptotically unbiased sequence of estimators for $F(t) = P(X \le t)$.

Suppose $X_1, X_2, \ldots, X_n$ is a random sample of the population random variable X, which has $E[X] = \mu$ and $\mathrm{Var}[X] = \sigma^2 < \infty$. The Central Limit Theorem gives the limiting distribution for $\sqrt{n}(\overline{X} - \mu)/\sigma$ as the standard Gaussian $Z_{0,1}$. Let us assume (unrealistically) for the moment that we know σ^2. Then, we can announce $\overline{X}$ as our estimate of μ, and we can provide some credibility for this estimate in the form of a *confidence interval*. Suppose we want a 90% confidence interval. From tables for the standard Gaussian, we discover that $P(|Z_{0,1}| \le 1.645) = 0.9$. For large n, we have

$$\begin{aligned} 0.9 &= P(|Z_{0,1}| \le 1.645) \approx P\left(\left|\frac{\sqrt{n}(\overline{X} - \mu)}{\sigma}\right| \le 1.645\right) \\ &= P\left(|\overline{X} - \mu| \le \frac{1.645\sigma}{\sqrt{n}}\right) \end{aligned}$$

If we let $\delta = 1.645\sigma/\sqrt{n}$, we can assert that, for large n, there is a 90% chance that the estimate $\overline{X}$ will lie within δ of the population parameter μ. We can further manipulate the equation above to obtain $P(\overline{X} - \delta \le \mu \le \overline{X} + \delta) \approx 0.9$. The specific interval obtained by substituting the observed value of $\overline{X}$ into the generic form $[\overline{X} - \delta, \overline{X} + \delta]$ is known as the (90%) confidence interval. It must be properly interpreted. The parameter μ is an unknown constant, not a random variable. Consequently, either μ lies in the specified confidence interval or it does not. The random variable is the interval itself, which changes endpoints when new values of $\overline{X}$ are observed. The width of the interval remains constant at δ. The proper interpretation is that 90% of these nondeterministic intervals will bracket the parameter μ.

Under more realistic conditions, neither the mean μ nor the variance σ^2 of the population is known. In this case, we can make further progress if we assume that the individual X_i samples are normal random variables. Various devices, such as composing each X_i as a sum of a subset of the samples, render this assumption more viable. In any case, under this constraint, we can show that $n(\overline{X} - \mu)^2/\sigma^2$ and $(n-1)\mathcal{S}^2/\sigma^2$ are independent random variables with known distributions. These random variables have chi-squared distributions.

A *chi-squared* random variable with m *degrees of freedom* is the sum of the squares of m independent standard normal random variables. It is actually a special case of the gamma distributions discussed previously; it occurs when the parameters are $\gamma = m$ and $\lambda = 1/2$. If Y_1 is chi-squared with m_1 degrees of freedom and Y_2 is chi-squared with m_2 degrees of freedom, then the ratio $m_2Y_1/(m_1Y_2)$ has an F distribution with (n,m) degree of freedom. A symmetric random variable is said to follow a t distribution with m_2 degrees of freedom if its square has an F distribution with $(1,m_2)$ degrees of freedom. For a given random variable R and a given value p in the range $(0, 1)$, the point r_p for which $P(R \le r_p) = p$ is called the pth *percentile* of the random variable. Percentiles for F and t distributions are available in tables.

Returning to our sample $X_1, X_2, \ldots, X_n$, we find that under the normal inference constraint, the two statistics mentioned above have independent chi-squared distributions with 1 and $n-1$ degrees of freedom, respectively. Therefore the quantity $\sqrt{n}|\overline{X} - \mu|/\sqrt{S^2}$ has a t distribution with $n-1$ degrees of freedom. Given a confidence level, say

90%, we consult a table of percentiles for the t distribution with $n-1$ degrees of freedom. We obtain a symmetric interval $[-r, r]$ such that

$$0.9 = P\left(\frac{\sqrt{n}|\overline{X}-\mu|}{\sqrt{S}^2} \le r\right) = P\left(|\overline{X}-\mu| \le \frac{r\sqrt{S}^2}{\sqrt{n}}\right)$$

Letting $\delta = r\sqrt{S}^2/\sqrt{n}$, we obtain the 90% confidence interval $[\overline{X}-\delta, \overline{X}+\delta]$ for our estimate $\overline{X}$ of the population parameter μ. The interpretation of this interval remains as discussed above.

This discussion above is an exceedingly abbreviated introduction to a vast literature on statistical inference. The references below provide a starting point for further study.

FURTHER READING

B. Fristedt and L. Gray, *A Modern Approach to Probability Theory*. Cambridge, MA: Birkhuser, 1997.

A. Gut, *Probability: A Graduate Course*. New York: Springer, 2006.

I. Hacking, *The Emergence of Probability*. Cambridge, MA: Cambridge University Press, 1975.

I. Hacking, *The Taming of Chance*. Cambridge, MA: Cambridge University Press, 1990.

J. L. Johnson, *Probability and Statistics for Computer Science*. New York: Wiley, 2003.

O. Ore, *Cardano, the Gambling Scholar*. Princeton, NJ: Princeton University Press, 1953.

O. Ore, Pascal and the invention of probability theory, *Amer. Math. Monthly*, **67**: 409–419, 1960.

C. A. Pickover, *Computers and the Imagination*. St. Martin's Press, 1991.

S. M. Ross, *Probability Models for Computer Science*. New York: Academic Press, 2002.

H. Royden, *Real Analysis*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1988.

BIBLIOGRAPHY

1. D. E. Knuth, *The Art of Computer Programming*, Vol. **2** 3rd ed. Reading, MA: Addison-Welsey, 1998.

James Johnson
Western Washington University
Bellingham, Washington

P

PROOFS OF CORRECTNESS IN MATHEMATICS AND INDUSTRY

THE QUALITY PROBLEM

Buying a product from a craftsman requires some care. For example, in the Stone Age, an arrow, used for hunting and hence for survival, needed to be inspected for its sharpness and proper fixation of the stone head to the wood. Complex products of more modern times cannot be checked in such a simple way and the idea of warranty was born: A nonsatisfactory product will be repaired or replaced, or else you get your money back. This puts the responsibility for quality on the shoulders of the manufacturer, who has to test the product before selling. In contemporary IT products, however, testing for proper functioning in general becomes impossible. If we have an array of 17×17 switches in a device, the number of possible positions is $2^{17^2} = 2^{289} \sim 10^{87}$, more than the estimated number of elementary particles in the universe. Modern chips have billions of switches on them, hence, a state space of a size that is truly dwarfing astronomical numbers. Therefore, in most cases, simple-minded testing is out of the question because the required time would surpass by far the lifetime expectancy of the universe. As these chips are used in strategic applications, like airplanes, medical equipment, and banking systems, there is a problem with how to warrant correct functioning.

Therefore, the need for special attention to the quality of complex products is obvious, both from a user's point of view and that of a producer. This concern is not just academic. In 1994 the computational number theorist T. R. Nicely discovered by chance a bug[1] in a widely distributed Pentium chip. After an initial denial, the manufacturer eventually had to publicly announce a recall, replacement, and destruction of the flawed chip with a budgeted cost of US $475 million.

Fortunately, mathematics has found a way to handle within a finite amount of time a supra-astronomical number of cases, in fact, an infinity of them. The notion of proof provides a way to handle all possible cases with certainty. The notion of mathematical induction is one proof method that can deal with an infinity of cases: If a property P is valid for the first natural number 0 (or if you prefer 1) and if validity of P for n implies that for $n+1$, then P is valid for all natural numbers. For example, for all n one has

$$\sum_{k=0}^{n} k^2 = \frac{1}{6} n(n+1)(2n+1). \qquad P(n)$$

This can be proved by showing it for $n = 0$; and then showing that if $P(n)$ holds, then also $p(n+1)$. Indeed $P(0)$ holds: $\sum_{k=0}^{0} k^2 = 0$. If $P(n)$ holds, then

$$\begin{aligned} \sum_{k=0}^{n+1} k^2 &= \left(\sum_{k=0}^{n} k^2\right) + (n+1)^2 \\ &= \frac{1}{6} n(n+1)(2n+1) + (n+1)^2 \\ &= \frac{1}{6}(n+1)(n+2)(2n+3), \end{aligned}$$

hence $P(n+1)$. Therefore $P(n)$ holds for all natural numbers n.

Another method to prove statements valid for an infinite number of instances is to use symbolic rewriting: From the usual properties of addition and multiplication over the natural numbers (proved by induction), one can derive equationally that $(x+1)(x-1) = x^2 - 1$, for all instances of x.

Proofs have been for more than two millennia the essence of mathematics. For more than two decades, proofs have become essential for warranting quality of complex IT products. Moreover, by the end of the twentieth century, proofs in mathematics have become highly complex. Three results deserve mention: the Four Color Theorem, the Classification of the Finite Simple Groups, and the correctness of the Kepler Conjecture (about optimal packing of equal three-dimensional spheres). Part of the complexity of these proofs is that they rely on large computations by a computer (involving up to a billion cases). A new technology for showing correctness has emerged: automated verification of large proofs.

Two methodological problems arise (1). How do proofs in mathematics relate to the physical world of processors and other products? (2) How can we be sure that complex proofs are correct? The first question will be addressed in the next section, and the second in the following section. Finally, the technology is predicted to have a major impact on the way mathematics will be done in the future.

SPECIFICATION, DESIGN, AND PROOFS OF CORRECTNESS

The Rationality Square

The ideas in this section come from Ref. 2 and make explicit what is known intuitively by designers of systems that use proofs. The first thing to realize is that if we want quality of a product, then we need to specify what we want as its behavior. Both the product and its (desired) behavior are in "reality", whereas the specification is written in some precise language. Then we make a design with the intention to realize it as the intended product. Also the design is a formal (mathematical) object. If one can prove that the designed object satisfies the formal specification, then it is expected that the realization has the desired behavior

[1] It took Dr. Nicely several months to realize that the inconsistency he noted in some of his output was not due to his algorithms, but caused by the (microcode on the) chip. See Ref. 1 for a description of the mathematics behind the error.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

$$\begin{tikzcd}
\text{Design} \arrow[r, dash, "\text{proof}"] \arrow[d, "\text{realization}"'] & \text{Specification} \\
\text{Product} \arrow[r, dash, "\text{warranty}"'] & \text{Behavior} \arrow[u, "\text{requirement}"']
\end{tikzcd}$$

Figure 1. Wupper's rationality square.

(see Fig. 1). For this it is necessary that the informal (desired) behavior and the specification are close to each other and can be inspected in a clearly understandable way. The same holds for the design and realization. Then the role of proofs is in its place: They do not apply to an object and desired behavior in reality but to a mathematical descriptions of these.

In this setup, the specification language should be close enough to the informal specification of the desired behavior. Similarly, the technology of realization should also be reliable. The latter again may depend on tools that are constructed component wise and realize some design (e.g., silicon compilers that take as input the design of a chip and have as output the instructions to realize them). Hence, the rationality square may have to be used in an earlier phase.

This raises, however, two questions. Proofs should be based on some axioms. Which ones? Moreover, how do we know that provability of a formal (mathematical) property implies that we get what we want? The answers to these questions come together. The proofs are based on some axioms that hold for the objects of which the product is composed. Based on the empirical facts that the axioms hold, the quality of the realized product will follow.

Products as Chinese Boxes of Components

Now we need to enter some of the details of how the languages for the design and specification of the products should look. The intuitive idea is that a complex product consists of components $b_1, \ldots, b_k$ put together in a specific way yielding $F^{(k)}(b_1, \ldots, b_k)$. The superscript "$(k)$" indicates the number of arguments that F needs. The components are constructed in a similar way, until one hits the basic components $O_0, O_1, \ldots$ that no longer are composed. Think of a playing music installation B. It consists of a CD, CD-player, amplifier, boxes, wires and an electric outlet, all put together in the right way. So

$$B = F^{(6)}(\text{CD}, \text{CD-player, amplifyer, boxes, wires, outlet}),$$

where $F^{(6)}$ is the action that makes the right connections. Similarly the amplifier and other components can be described as a composition of their parts. A convenient way to depict this idea in general is the so-called *Chinese box* (see Fig. 2). This is a box with a lid. After opening the lid one finds a (finite) set of "neatly arranged" boxes that either are open and contain a basic object or are again other Chinese boxes (with lid). Eventually one will find something in a decreasing chain of boxes. This corresponds to the component-wise construction of anything, in particular of hardware, but also of software[2]. It is easy to construct a grammar for expressions denoting these Chinese boxes. The basic objects are denoted by $o_0, o_1, \ldots$. Then there are "constructors" that turn expressions into new expressions. Each constructor has an "arity" that indicates how many arguments it has. There may be unary, binary, ternary, and so on constructors. Such constructors are denoted by

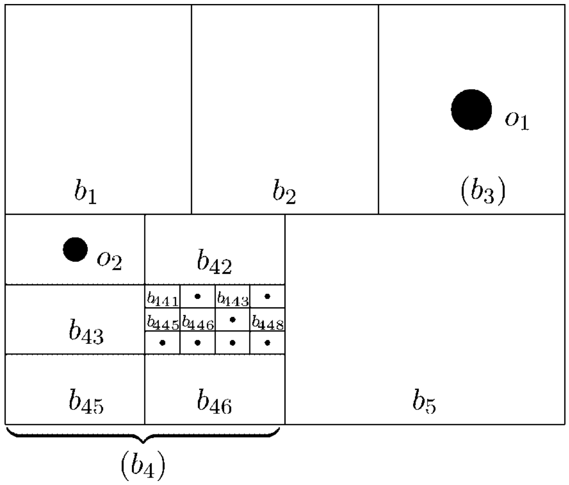


Figure 2. Partially opened Chinese box.

$$f_0^{(k)}, f_1^{(k)}, \ldots,$$

where k denotes the arity of the constructor. If $b_1, \ldots, b_k$ are expressions and $f_i^{(k)}$ is a constructor of arity k, then

$$f_i^{(k)}(b_1, \ldots, b_k)$$

is an expression. A precise grammar for such expressions is as follows.

Definition

1. Consider the following alphabet:

$$\sum = \{o_i | i \in \mathbb{N}\} \cup \{f_i^k | i, k \in \mathbb{N}\} \cup \{,, (,)\}.$$

2. Expressions $\mathcal{E}$ form the smallest set of words over Σ satisfying

$$o_i \in \mathcal{E};$$

$$b_1, \ldots, b_k \in \mathcal{E} \Rightarrow f_i^{(k)} \ (b_1, \ldots, b_k) \in \mathcal{E}.$$

An example of a fully specified expression is

$$f_1^{(2)}(o_0, f_3^1(o_1, o_2, o_0)).$$

[2] In order that the sketched design method works well for software, it is preferable to have *declarative* software, i.e., in the *functional* or *logic* programming style, in particular without side effects.

The partially opened Chinese box in Fig. 2 can be denoted by

$$f_1^{(5)}(b_1, b_2, o_1, b_4, b_5),$$

where

$$\begin{aligned} b_4 &= f_2^{(6)}(o_2, b_{4,2}, b_{4,3}, b_{4,4}, b_{4,5}, b_{4,6}),\\ b_{4,4} &= f_4^{(12)}(b_{4,4,1}, o_3, b_{4,4,3}, o_4, b_{4,3,5}, b_{4,4,6}, o_5,\\ &\quad b_{4,4,8}, o_6, o_7, o_8, o_9), \end{aligned}$$

and the other b_k still have to be specified.

Definition. A *design* is an expression $b \in \mathcal{E}$.

Specification and Correctness of Design

Following the rationality square, one now can explain the role of mathematical proofs in industrial design.

Some mathematical language is needed to state in a precise way the requirements of the products. We suppose that we have such a specification language $\mathcal{L}$, in which the expressions in $\mathcal{E}$ are terms. We will not enter into the details of such a language, but we will mention that for IT products, it often is convenient to be able to express relationships between the states before and after the execution of a command or to express temporal relationships. Temporal statements include "eventually the machine halts" or "there will be always a later moment in which the system receives input". See Refs. 3–6 for possible specification languages, notably for reactive systems, and Ref. 7 for a general introduction to the syntax and semantics of logical languages used in computer science.

Definition. A *specification* is a unary formula[3] $S(\cdot)$ in $\mathcal{L}$.

Suppose we have the specification S and a candidate design b as given. The task is to prove in a mathematical way $S(b)$, i.e., that S holds of b. We did not yet discuss any axioms, or a way to warrant that the proved property is relevant. For this we need the following.

Definition. A *valid interpretation* for $\mathcal{L}$ consists of the following.

1. For basic component expressions o, there is an interpretation O in the "reality" of products.
2. For constructors $f^{(k)}$, there is a way to put together k products $p_1, \ldots, p_k$ to form $F^{(k)}(p_1, \ldots, p_k)$.
3. By (1) and (2), all designs have a realization. For example, the design $f_1^{(2)}(o_0, f_3^{(1)}(o_1, o_2, o_0))$ is interpreted as $F_1^{(2)}(O_0, F_3^{(1)}(O_1, O_2, O_0))$.
4. (4) There are axioms of the form

$$P(c)$$
$$\forall x_1 \ldots x_k [Q(x_1, \ldots, x_k) \Rightarrow R(f^{(k)}(x_1, \ldots, x_k))].$$

   Here P, Q, and R are formulas (formal statements) about designs: P and R about one design and Q about k designs.
5. The formulas of $\mathcal{L}$ have a physical interpretation.
6. By the laws of physics, it is known that the interpretation given by (5) of the axioms holds for the interpretation described in the basic components and constructors. The soundness of logic then implies that statements proved from the axioms will also hold after interpretation.

This all may sound a bit complex, but the idea is simple and can be found in any book on predicate logic and its semantics (see Refs. 7 and 8). Proving starts from the axioms using logical steps; validity of the axioms and soundness of logic implies that the proved formulas are also valid.

The industrial task of constructing a product with a desired behavior can be fulfilled as follows.

Design Method (I)

1. *Find a language $\mathcal{L}$ with a valid interpretation.*
2. *Formulate a specification S, such that the desired behavior becomes the interpretation of S.*
3. *Construct an expression b, intended to solve the task.*
4. *Prove S(b) from the axioms of the interpretation mentioned in (1).*
5. *The realization of b is the required product.*

Of course the last step of realizing designs may be nontrivial. For example, transforming a chip design to an actual chip is an industry by itself. But that is not the concern now. Moreover, such a realization process can be performed by a tool that is the outcome of a similar specification-design-proof procedure.

The needed proofs have to be given from the axioms in the interpretation. Design method I builds up products from "scratch". In order not to reinvent the wheel all the time, one can base new products on previously designed ones.

Design Method (II). *Suppose one wants to construct b satisfying S.*

1. *Find subspecifications $S_1, \ldots, S_k$ and a constructor $f^{(k)}$ such that*

$$S_1(x_i) \;\&\; \ldots \;\&\; S_k(x_k) \Rightarrow S(f^{(k)}(x_1, \ldots, x_k)).$$

2. *Find (on-the-shelf) designs $b_1, \ldots, b_k$ such that for $1 \le i \le k$, one has*

$$S_i(b_i).$$

3. Then the design $b = f^{(k)}(b_1, \ldots, b_k)$ solves the task.

Again this is done in a context of a language $\mathcal{L}$ with a valid interpretation and the proofs are from the axioms in the interpretation.

[3] Better: A formula $S = S(x) = S(\cdot)$ with one free variable x in S.

After having explained proofs of correctness, the correctness of proofs becomes an issue. In an actual nontrivial industrial design, a software system controlling metro-trains in Paris without a driver, one needed to prove about 25,000 propositions in order to get reliability. These proofs were provided by a theorem prover. Derivation rules were added to enhance the proving power of the system. It turned out that if no care was taken, 2% to 5% of these added derivation rules were flawed and led to incorrect statements; see Ref. 9. The next section deals with the problem of getting proofs right.

CORRECTNESS OF PROOFS

Methodology

Both in computer science and in mathematics proofs can become large. In computer science, this is the case because the proofs that products satisfy certain specifications, as explained earlier, may depend on a large number of cases that need to be analyzed. In mathematics, large proofs occur as well, in this case because of the depth of the subject. The example of the Four Color Theorem in which billions of cases need to be checked is well known. Then there is the proof of the classification theorem for simple finite groups needing thousands of pages (in the usual style of informal rigor).

That there are long proofs of short statements is not an accident, but a consequence of a famous undecidability result.

Theorem (Turing). *Provability in predicate logic is undecidable.*

Proof. See, for example, Ref. 10. ■

Corollary. *For predicate logic, there is a number n and a theorem of length n, with the smallest proof of length at least $n^{n!}$.*

Proof. Suppose that for every n theorems of length at least n, a proof of length $< n^{n!}$ exists. Then checking all possible proofs of such length provides a decision method for theoremhood, contradicting the undecidablity result. ■

Of course this does not imply that there are *interesting* theorems with essentially long proofs.

The question now arises, how one can verify long proofs and large numbers of shorter ones? This question is both of importance for pure mathematics and for the industrial applications mentioned before.

The answer is that the state of the foundations of mathematics is such that proofs can be written in full detail, making it possible for a computer to check their correctness. Currently, it still requires considerable effort to make such "formalizations" of proofs, but there is good hope that in the future this will become easier. Anyway, industrial design, as explained earlier, already has proved the viability and value of formal proofs. For example, the Itanium, a successor of the Pentium chip, has a provably correct arithmetical unit; see Ref. 11.

Still one may wonder how one can assure the correctness of mathematical proofs via machine verification, if such proofs need to assure the correctness of machines. It seems that there is here a vicious circle of the chicken-and-the-egg type. The principal founder of machine verification of formalized proofs is the Dutch mathematician N. G. de Bruijn[4]; see Ref. 13. He emphasized the following criterion for reliable automated proof-checkers: Their programs must be small, so small that a human can (easily) verify the code by hand. In the next subsection, we will explain why it is possible to satisfy this so-called de Bruijn criterion.

Foundations of Mathematics

The reason that fully formalized proofs are possible is that for all mathematical activities, there is a solid foundation that has been laid in a precise formal system. The reason that automated proof-checkers exist that satisfy the de Bruijn criterion is that these formal systems are simple enough, allowing a logician to write them down from memory in a couple of pages.

Mathematics is created by three mental activities: structuring, computing, and reasoning. It is an art and craftsmanship "with a power, precision and certainty, that is unequalled elsewhere in life[5]." The three activities, respectively, provide definitions and structures, algorithms and computations, and proofs and theorems. These activities are taken as a subject of study by themselves, yielding ontology (consisting either of set, type, or category theory), computability theory, and logic.

Activity	Tools	Results	Meta study
Structuring	Axioms Definitions	Structures	Ontology
Computing	Algorithms	Answers	Computability[6]
Reasoning	Proofs	Theorems	Logic

Figure 3. Mathematical activity: tools, results, and meta study.

During the history of mathematics these activities enjoyed attention in different degrees. Mathematics started with the structures of the numbers and planar geometry. Babylonian–Chinese–Egyptian mathematics, was mainly occupied with computing. In ancient Greek mathematics, reasoning was introduced. These two activities came together in the work of Archimedes, al-Kwarizmi, and Newton. For a long time only occasional extensions of the number systems was all that was done as structuring activity. The art of defining more and more structures started in the ninteenth century with the introduction of groups by Galois and non-Euclidean spaces by Lobachevsky and Bolyai. Then mathematics flourished as never before.

[4] McCarthy described machine proof-checking some years earlier (see, Ref. 12), but did not come up with a formal system that had a sufficiently powerful and convenient implementation.

[5] From: *The man without qualities*, R. Musil, Rohwolt.

[6] Formerly called "Recursion Theory".

Logic. The quest for finding a foundation for the three activities started with Aristotle. This search for "foundation" does not imply that one was uncertain how to prove theorems. Plato had already emphasized that any human being of normal intelligence had the capacity to reason that was required for mathematics. What Aristotle wanted was a survey and an understanding of that capacity. He started the quest for logic. At the same time Aristotle introduced the "synthetic way" of introducing new structures: the axiomatic method. Mathematics consists of concepts and of valid statements. Concepts can be defined from other concepts. Valid statements can be proved from other such statements. To prevent an infinite regress, one had to start somewhere. For concepts one starts with the primitive notions and for valid statements with the axioms. Not long after this description, Euclid described geometry using the axiomatic method in a way that was only improved by Hilbert, more than 2000 years later. Also Hilbert gave the right view on the axiomatic method: The axioms form an implicit definition of the primitive notions.

Frege completed the quest of Aristotle by giving a precise description of predicate logic. Gödel proved that his system was complete, i.e., sufficiently strong to derive all valid statements within a given axiomatic system. Brouwer and Heyting refined predicate logic into the so-called *intuitionistic* version. In their system, one can make a distinction between a weak existence ("there exists a solution, but it is not clear how to find it") and a constructive one ("there exists a solution and from the proof of this fact one can construct it") (see Ref. 14).

Ontology. An early contribution to ontology came from Descartes, who introduced what is now called Cartesian products (pairs or more generally tuples of entities), thereby relating geometrical structures to arithmetical (in the sense of algebraic) ones. When in the nineteenth century, there was a need for systematic ontology, Cantor introduced set theory in which sets are the fundamental building-blocks of mathematics. His system turned out to be inconsistent, but Zermelo and Fraenkel removed the inconsistency and improved the theory so that it could act as an ontological foundation for large parts of mathematics, (see Ref 15).

Computability. As soon as the set of consequences of an axiom system had become a precise mathematical object, results about this collection started to appear. From the work of Gödel, it followed that the axioms of arithmetic are essentially incomplete (for any consistent extension of arithmetic, there is an independent statement A that is neither provable nor refutable). An important part of the reasoning of Gödel was that the notion "p is a proof of A" is after coding a computable relation. Turing showed that predicate logic is undecidable (it cannot be predicted by machine whether a given statement can be derived or not). To prove undecidability results, the notion of computation needed to be formalized. To this end, Church came with a system of lambda-calculus (see Ref. 16), later leading to the notion of functional programming with languages such as Lisp, ML, and Haskell. Turing came with the notion of the Turing machine, later leading to imperative programming with languages such as Fortran and C and showed that it gave the same notion of computability as Church's. If we assume the so-called Church–Turing thesis that humans and machines can compute the same class of mathematical functions, something that most logicians and computer scientists are willing to do, then it follows that provability in predicate logic is also undecidable by humans.

Mechanical Proof Verification

As soon as logic was fully described, one started to formalize mathematics. In this endeavor, Frege was unfortunate enough to base mathematics on the inconsistent version of Cantorian set theory. Then Russell and Whitehead came with an alternative ontology, type theory, and started to formalize very elementary parts of mathematics. In type theory, that currently exists in various forms, functions are the basic elements of mathematics and the types form a way to classify these. The formal development of mathematics, initiated by Russell and Whitehead, lay at the basis of the theoretical results of Gödel and Turing. On the other hand, for practical applications, the formal proofs become so elaborate that it is almost undoable for a human to produce them, let alone to check that they are correct.

It was realized by J. McCarthy and independently by N. G. de Bruijn that this verification should not be done by humans but by machines. The formal systems describing logic, ontology, and computability have an amazingly small number of axioms and rules. This makes it possible to construct relatively small mathematical assistants. These computer systems help the mathematician to verify whether the definitions and proofs provided by the human are well founded and correct.

Based on an extended form of type theory, de Bruijn introduced the system AUTOMATH (see Ref. 17), in which this idea was first realized, although somewhat painfully, because of the level of detail in which the proofs needed to be presented. Nevertheless, proof-checking by mathematical assistants based on type theory is feasible and promising. For some modern versions of type theory and assistants based on these, see Refs. 17–21.

Soon after the introduction of AUTOMATH, other mathematical assistants were developed, based on different foundational systems. There is the system MIZAR based on set theory; the system HOL(-light) based on higher order logic; and ACL_2 based on the computational model "primitive recursive arithmetic." See Ref. 22 for an introduction and references and Ref. 23 for resulting differences of views in the philosophy of mathematics. To obtain a feel of the different styles of formalization, see Ref. 24.

In Ref. 25, an impressive full development of the Four Color Theorem is described. Tom Hales of the University of Pittsburgh, assisted by a group of computer scientists, specializing in formalized proof-verification, is well on his way to verifying his proof of the Kepler conjecture (26); see Ref. 27. The *Annals of Mathematics* published that proof and considered adding—but finally did not do so– a proviso, that the referees became exhausted (after 5 years) from checking all of the details by hand; therefore, the full correctness depends on a (perhaps not so reliable) computer computation. If Hales and his group succeed in formalizing and verifying the entire proof, then that will be of a reliability higher than most mathematical proofs, one third of which is estimated to contain real errors, not just typos.[7]

The possibility of formalizing mathematics is not in contradiction with Gödel's theorem, which only states the limitations of the axiomatic method, informal or formal alike. The proof of Gödel's incompleteness theorem does in fact heavily rely on the fact that proof-checking is decidable and uses this by reflecting over the notion of provability (the Gödel sentence states: "This sentence is not provable").

One particular technology to verify that statements are valid is the use of model-checking. In IT applications the request "statement A can be proved from assumptions Γ (the 'situation')" often boils down to "A is valid in a model $\mathcal{A} = \mathcal{A}_\Gamma$ depending on Γ". (In logical notation

$$\Gamma \vdash A \Leftrightarrow \mathcal{A}_\Gamma \models A.$$

This is so because of the completeness theorem of logic and because of the fact that the IT situation is related to models of digital hardware that are finite by its nature.) Now, despite the usual huge size of the model, using some cleverness the validity in several models in some industrially relevant cases is decidable within a feasible ammount of time. One of these methods uses the so-called *binary decision diagrams* (BDDs). Another ingredient is that universal properties are checked via some rewriting rules, like $(x+1)(x-1) = x^2 - 1$.

For an introduction to model-checkers, see Ref. 20. For successful applications, see Ref. 29. The method of model-checking is often somewhat ad hoc, but nevertheless important. Using "automated abstraction" that works in many cases (see Refs. 30 and 31), the method becomes more streamlined.

SCALING-UP THROUGH REFLECTION

As to the question of whether fully formalized proofs are practically possible, the opinions have been divided. Indeed, it seems too much work to work out intuitive steps in full detail. Because of industrial pressure, however, full developments have been given for correctness of hardware and frequently used protocols. Formalizations of substantial parts of mathematics have been lagging behind.

There is a method that helps in tackling larger proofs. Suppose we want to prove statement A. Then it helps if we can write $A \leftrightarrow B(f(t))$, where t belongs to some collection X of objects, and we also can see that the truth of this is independent of t; i.e., one has a proof $\forall x \in X.B(f(x))$. Then $B(f(t))$, hence A.

An easy example of this was conveyed to me by A. Mostowski in 1968. Consider the following formula as proof obligation in propositional logic:

$$A = p \leftrightarrow (p \leftrightarrow (p \leftrightarrow (p \leftrightarrow (p \leftrightarrow (p \leftrightarrow (p \leftrightarrow (p \leftrightarrow (p \leftrightarrow (p \leftrightarrow (p \leftrightarrow)))))))))).$$

Then $A \leftrightarrow B(12)$, with $B(1) = p, B(n+1) = (p \leftrightarrow B(n))$. By induction on n one can show that for all natural numbers $n \geq 1$, one has $B(2 \times n)$. Therefore, B(12) and hence A, because $2 \times 6 = 12$. A direct proof from the axioms of propositonal logic would be long. Much more sophisticated examples exist, but this is the essence of the method of reflection. It needs some form of computational reasoning inside proofs. Therefore, the modern mathematical assistants contain a model of computation for which equalities like $2 \times 6 = 12$ and much more complex ones become provable. There are two ways to do this. One possibility is that there is a deduction rule of the form

$$\frac{A(s)}{A(t)}\ s \twoheadrightarrow_R t.$$

This so-called *Poincaré Principle* should be interpreted as follows: From the assumption $A(s)$ and the side condition that s computationally reduces in several steps to t according to the rewrite system R, it follows that $A(t)$. The alternative is that the transition from $A(s)$ to $A(t)$ is only allowed if $s = t$ has been proved first. These two ways of dealing with proving computational statements can be compared with the styles of, respectively, functional and logical programming. In the first style, one obtains proofs that can be recorded as *proof-objects*. In the second style, these full proofs become too large to record as one object, because computations may take giga steps. Nevertheless the proof exists, but appearing line by line over time, and one speaks about an *ephemeral* proof-object.

In the technology of proof-verification, general statements are about mathematical objects *and* algorithms, proofs show the correctness of statements *and* computations, and computations are dealing with objects *and* proofs.

RESULTS

The state-of-the-art of computer-verified proofs is as follows. To formalize one page of informal mathematics, one needs four pages in a fully formalized style and it takes about five working days to produce these four pages (see Ref. 22). It is expected that both numbers will go down. There have been formalized several nontrivial statements, like the fundamental theorem of algebra (also in a constructive fashion; it states that every non-constant polynomial over the complex numbers has a root), the prime number theorem (giving an asymptotic estimation of the number of primes below a given number), and the Jordan curve theorem (every closed curve divides the plane into two regions that cannot be reached without crossing this curve; on the torus surface, this is not true). One of the great success stories is the full formalization of the Four Color Theorem by Gonthier (see Ref. 25). The original proof of

[7] It is interesting to note that, although informal mathematics often contains bugs, the intuition of mathematicians is strong enough that most of these bugs usually can be repaired.

this result was not completely trustable for its correctness, as a large number of cases needed to be examined by computer. Gonthier's proof still needs a computer-aided computation, but all steps have been formally verified by an assistant satisfying the de Bruijn principle.

BIBLIOGRAPHY

1. Alan Edelman, The mathematics of the Pentium division bug, *SIAM Review*, **37**: 54–67, 1997.
2. H. Wupper, Design as the discovery of a mathematical theorem – What designers should know about the art of mathematics, in Ertas, et al., (eds.), *Proc. Third Biennial World Conf. on Integrated Design and Process Technology (IDPT)*, 1998, pp. 86–94; *J. Integrated Des. Process Sci.*, **4** (2): 1–13, 2000.
3. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, New York: Springer, 1992.
4. K. R. Apt and Ernst-Rüdiger Olderog, *Verification of Sequential and Concurrent Programs*, Texts and Monographs in Computer Science, 2nd ed. New York: Springer-Verlag, 1997.
5. C.A.P Hoare and H. Jifeng, *Unifying Theories of Programming*, Englewood Cliffs, N.J.: Prentice Hall, 1998.
6. J. A. Bergstra, A. Ponse, and S. A. Smolka, eds., *Handbook of Process Algebra*, Amsterdam: North-Holland Publishing Co., 2001.
7. B.-A. Mordechai, *Mathematical Logic for Computer Science*, New York: Springer, 2001.
8. W. Hodges, *A Shorter Model Theory*, Cambridge, U.K.: Cambridge University Press, 1997.
9. J.-R. Abrial, On B, in D. Bert, (ed.), *B'98: Recent Advances in the Development and Use of the B Method: Second International B Conference Montpellier*, in Vol. 1393 of *LNCS*, Berlin: Springer, 1998, pp. 1–8.
10. M. Davis, ed., *The Undecidable*, Mineola, NY: Dover Publications Inc., 2004.
11. B. Greer, J. Harrison, G. Henry, W. Li, and P. Tang, Scientific computing on the Itanium^r processor, *Scientific Prog.*, **10** (4): 329–337, 2002,
12. J. McCarthy, Computer programs for checking the correctness of mathematical proofs, in *Proc. of a Symposium in Pure Mathematics, vol. V.*, Providence, RI, 1962, pp. 219–227.
13. N. G. de Bruijn, The mathematical language AUTOMATH, its usage, and some of its extensions, in *Symposium on Automatic Demonstration, Versailles, 1968, Mathematics*, **125**: 29–61, Berlin: Springer, 1970.
14. D. van Dalen, *Logic and Structure*, Universitext, 4th ed. Berlin: Springer-Verlag, 2004.
15. P. R. Halmos, *Naive Set Theory*, New York: Springer-Verlag, 1974.
16. H. P. Barendregt, Lambda calculi with types, in *Handbook of Logic in Computer Science*, Vol. 2, New York: Oxford Univ. Press, 1992, pp. 117–309.
17. R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer, Twenty-five years of Automath research, in *Selected Papers on Automath*, volume 133 of *Stud. Logic Found, Math.*, North-Holland, Amsterdam, 1994, pp. 3–54.
18. P. Martin-Löf, *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory, Lecture Notes*, Bibliopolis, Naples, Italy, 1984.
19. R. L. Constable, The structure of Nuprl's type theory, in *Logic of Computation (Marktoberdorf, 1995)*, vol. 157 of *NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci.*, Berlin: Springer, 1997, pp. 123–155.
20. H. P. Barendregt and H. Geuvers, Proof-assistants using dependent type systems, in A. Robinson and A. Voronkov, (eds.), *Handbook of Automated Reasoning*, Elsevier Science Publishers B.V., 2001, pp. 1149–1238.
21. Y. Bertot and P. Castéran, *Coq'Art: The Calculus of Inductive Constructions*, Texts in Theoretical Computer Science, Berlin: Springer, 2004.
22. H. P. Barendregt and F. Wiedijk, The challenge of computer mathematics, *Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci.*, **363** (1835): 2351–2375, 2005.
23. H. P. Barendregt, Foundations of mathematics from the perspective of computer verification, in *Mathematics, Computer Science, Logic – A Never Ending Story*. New York: Springer-Verlag, 2006. To appear. Available www.cs.ru.nl/˜henk/papers.html.
24. F. Wiedijk, *The Seventeen Provers of the World*, vol. 3600 of *LNCS*, New York: Springer, 2006.
25. G. Gonthier, A computer checked proof of the Four Colour Theorem, 2005. Available: ⟨research.microsoft.com/˜gonthier/4colproof.pdf⟩.
26. T. C. Hales, A proof of the Kepler conjecture, *Ann. of Math. (2)*, **162** (3): 1065–1185, 2005.
27. T. C. Hales, The flyspeck project fact sheet. Available: www.math.pitt.edu/~thales/flyspeck/index.html.
28. E. M. Clarke Jr., O. Grumberg, and D. A. Peled, *Model Checking*, Cambridge, MA: MIT Press, 1999.
29. G. J. Holzmann, *The SPIN model checker, primer and reference manual*, Reading, MA: Addison-Wesley, 2003.
30. S. Bensalem, V. Ganesh, Y. Lakhnech, C. Mu noz, S. Owre, H. Rue, J. Rushby, V. Rusu, H. Sadi, N. Shankar, E. Singerman, and A. Tiwari, An overview of SAL, in C. M. Holloway, (ed.), *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, 2000, pp. 187–196.
31. F. W. Vaandrager, Does it pay off? Model-based verification and validation of embedded systems! in F. A. Karelse, (ed.), *PROGRESS White papers 2006*. STW, the Netherlands, 2006. Available: www.cs.ru.nl/ita/publications/papers/fvaan/whitepaper.

HENK BARENDREGT
Radboud University Nijmegen
Nijmegen, The Netherlands

REGRESSION ANALYSIS

In statistics, regression is the study of *dependence*. The goal of regression is to study how the distribution of a response variable changes as values of one or more predictors are changed. For example, regression can be used to study changes in automobile stopping distance as speed is varied. In another example, the response could be the total profitability of a product as characteristics of it like selling price, advertising budget, placement in stores, and so on, are varied. Key uses for regression methods include prediction of future values and assessing dependence of one variable on another.

The study of conditional distributions dates at least to the beginning of the nineteenth century and the work of A. Legendre and C. F. Gauss. The use of the term *regression* is somewhat newer, dating to the work of F. Galton at the end of the nineteenth century; see Ref. 1 for more history.

GENERAL SETUP

For the general univariate regression problem, we use the symbol Y for a response variable, which is sometimes called the dependent variable. The response can be a continuous variable like a distance or a profit, or it could be discrete, like success or failure, or some other categorical outcome. The predictors, also called independent variables, carriers, or features, can also be continuous or categorical; in the latter case they are often called *factors* or *class variables*. For now, we assume only one predictor and use the symbol X for it, but we will generalize to many predictors shortly. The goal is to learn about the conditional distribution of Y given that X has a particular value x, written symbolically as $F(Y|X = x)$.

For example, Fig. 1 displays the heights of $n = 1375$ mother–daughter pairs, with $X =$ mother's height on the horizontal axis and $Y =$ daughter's height on the vertical axis, in inches. The conditional distributions $F(Y|X = x)$ correspond to the vertical spread of points in strips in this plot. In Fig. 1, three of these conditional distributions are highlighted, corresponding to mother's heights of 58, 61, and 65 inches. The conditional distributions almost certainly differ in mean, with shorter mothers on average having shorter daughters than do taller mothers, but there is substantial overlap between the distributions.

Most regression problems center on the study of the *mean function*, to learn about the average value of Y given $X = x$. We write the most general mean function as $\mu(Y|X = x)$, the mean of Y when $X = x$. The mean function for the heights data would be a smooth curve, with $\mu(Y|X = x)$ increasing as x increases. Other characteristics of the conditional distributions, such as conditional variances $\mathrm{var}(Y|X = x)$ may well be constant across the range of values for mother's height, but in general the variance or indeed any other moment or percentile function can depend on X.

Most regression models are *parametric*, so the mean function $\mu(Y|X = x)$ depends only on a few unknown parameters collected into a vector β. We write $\mu(Y|X = x) = g(x, \beta)$, where g is completely known apart from the unknown value of β.

In the heights data described above, data are generated obtaining a sample of units, here mother–daughter pairs, and measuring the values of height for each of the pairs. Study of the conditional distribution of daughter's height given the mother's height makes more sense than the study of the mother's height given the daughter's height because the mother precedes the daughter, but in principle either conditional distribution could be studied via regression. In other problems, the values of the predictor or the predictors may be set by an experimenter. For example, in a laboratory setting, samples of homogeneous material could be assigned to get different levels of stress, and then a response variable is measured with the goal of determining the effect of stress on the outcome. This latter scenario will usually include random assignment of units to levels of predictors and can lead to more meaningful inferences. Considerations for allocating levels of treatments to experimental units are part of the *design of experiments*; see Ref. 3. Both cases of predictors determined by the experimenter and predictors measured on a sample of units can often be analyzed using regression analysis.

SIMPLE LINEAR REGRESSION

Model

Linear regression is the most familiar and widely used method for regression analysis; see, for example, Ref. 4 for book-length treatment of simple and multiple regression. This method concentrates almost exclusively on the mean function. Data consist of n independent pairs $(x_1, y_1), \ldots, (x_n, y_n)$ as with the heights data in Fig. 1. The independence assumption might be violated if, for example, a mother were included several times in the data, each with a different daughter, or if the mothers formed several groups of sisters.

The simple linear regression model requires the following mean and variance functions:

$$\begin{aligned} \mu(Y|X = x) = g(x, \boldsymbol{\beta}) &= \beta_0 + \beta_1 x \\ \mathrm{Var}(Y|X = x) &= \sigma^2 \end{aligned} \tag{1}$$

so for this model $\beta = (\beta_0, \beta_1)'$. β_1 is *slope*, which is the expected change in Y when X is increased by one unit. The intercept β_0 is the mean value of Y when $X = 0$, although that interpretation may not make sense if X cannot equal zero. The line shown on Fig. 1 is an estimate of the simple regression mean function, computed using least squares, to be described below. For the heights data, the simple regression mean function seems plausible, as it matches the data in the graph.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

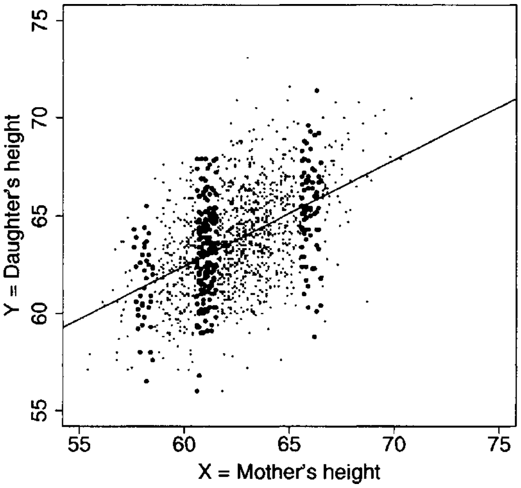


Figure 1. Heights of a sample of $n = 1375$ mothers and daughters as reported by Ref. 2. The line shown on the plot is the *ordinary least-squares* regression line, assuming a simple linear regression model. The darker points display all pairs with mother's height that would round to 58, 61, or 65 inches.

The simple regression model also assumes a constant variance function, with $\sigma^2 > 0$ generally unknown. This assumption is not a prerequisite for all regression models, but it is a feature of the simple regression model.

Estimation

We can obtain estimates of the unknown parameters, and thus of the mean and the variance functions, without any further assumptions. The most common method of estimation is via *least squares*, which chooses the estimates $\mathbf{b} = (b_0, b_1)'$ of $\beta = (\beta_0, \beta_1)'$ via a minimization problem:

$$b = \arg\min_{b^*} \sum_{i=1}^{n} \{y_i - g(x, b^*)\}^2 \qquad (2)$$

A generic notation is used in Equation (2) because this same objective function can be used for other parametric mean functions.

The solution to this minimization problem is easily found by differentiating Equation (2) with respect to each element of b* and setting the resulting equations to zero, and solving. If we write m_x and m_y as the sample means of the x_i and the y_i respectively, SD_x and SD_y as the sample standard deviations, and r_{xy} as the sample correlation, then

$$b_1 = r_{xy}\frac{\mathrm{SD}_y}{\mathrm{SD}_x} \qquad b_0 = m_y - b_1 m_x \qquad (3)$$

These are *linear* estimators because both m_y and r_{xy} are linear combinations of the y_i. They are also unbiased, $\mathrm{E}(b_0) = \beta_0$ and $\mathrm{E}(b_1) = \beta_1$. According to the Gauss-Markov theorem, the least-squares estimates have minimum variance among all possible linear unbiased estimates. Details are given in Refs. 4 and 5, the latter reference at a higher mathematical level.

As σ^2 is the mean-squared difference between each data point and its mean, it should be no surprise that the estimate of σ^2 is similar to the average of the squared fitting errors. Let d be the *degrees of freedom* for error, which in linear regression is the number of observations minus the number of parameters in the mean function, or $n - 2$ in simple regression. Then

$$s^2 = \frac{1}{d}\sum_{i=1}^{n}(y - g(x_i, \mathbf{b}))^2 \qquad (4)$$

The quantity $\sum(y_i - g(x_i, \mathbf{b}))^2$ is called the *residual sum of squares*. We divide by d rather than the more intuitive sample size n because this results in an unbiased estimate, $\mathrm{E}(s^2) = \sigma^2$. Many computing formulas for the residuals sum of squares depend only on summary statistics. One that is particularly revealing is

$$\sum_{i=1}^{n}(y - g(x_i, \mathbf{b}))^2 = (n-1)\mathrm{SD}_y^2(1 - R^2) \qquad (5)$$

In both simple and multiple linear regression, the quantity R^2 is the square of the *sample correlation between the observed response, the y_i, and the fitted values $g(x_i, \mathbf{b})$*. In simple regression $R^2 = r_{xy}^2$.

Distribution of Estimates

The estimates (b_0, b_1) are random variables with variances

$$\begin{aligned} \mathrm{Var}(b_0) &= \sigma^2\left(\frac{1}{n} + \frac{m_x^2}{(n-1)\mathrm{SD}_x^2}\right), \\ \mathrm{Var}(b_1) &= \sigma^2\frac{1}{(n-1)\mathrm{SD}_x^2} \end{aligned} \qquad (6)$$

The estimates are correlated, with covariance $\mathrm{Cov}(b_0, b_1) = -\sigma^2 m_x/\{(n-1)\mathrm{SD}_x^2\}$. The estimates are uncorrelated if the predictor is rescaled to have sample mean $m_x = 0$; that is, replace X by a new predictor $X^* = X - m_x$. This will also change the meaning of the intercept parameter from the value of $\mathrm{E}(Y|X = 0)$ to the value of $\mathrm{E}(Y|X = m_x)$. Estimates of the variances and covariances are obtained by substituting s^2 for σ^2. For example, the square root of the estimated variance of b_1 is called its *standard error*, and is given by

$$\mathrm{se}(b_1) = s\frac{1}{\{(n-1)\mathrm{SD}_x^2\}^{1/2}} \qquad (7)$$

Tests and confidence statements concerning the parameters require the *sampling distribution* of the statistic (b_0, b_1). This information can come about in three ways. First, we might assume normality for the conditional distributions, $F(Y|X = x) = \mathrm{N}(g(x, \beta), \sigma^2)$. Since the least squares estimates are linear functions of the y_i, this leads to normal sampling distributions for b_0 and b_1. Alternatively, by the central limit theorem b_0 and b_1 will be approximately normal regardless of the true F, assuming only mild regularity conditions and a large enough sample. A third approach uses the data itself to estimate the sampling distribution of b, and thereby get approximate inference. This last method is generally called the *bootstrap*, and is discussed briefly in Ref. 4 and more completely in Ref. 6.

Regardless of distributional assumptions, the estimate s^2 has a distribution that is independent of $\mathbf{b}$. If we add the normality assumption, then $ds^2 \sim \sigma^2\chi^2(d)$, a chi-squared distribution with d df. The ratio $(b_1 - \beta_1)/\text{se}(b_1)$ has a t-distribution with d df, written $t(d)$. Most tests in linear regression models under normality or in large samples are based either on t-distributions or on the related F-distributions.

Suppose we write $t_\gamma(d)$ to be the quantile of the t-distribution with d df that cuts off probability γ in its upper tail. Based on normal theory, either a normality assumption or large samples, a test of $\beta_1 = \beta_1^*$ versus the alternative $\beta_1 \neq \beta_1^*$ is rejected at level α if $t = (b_1 - \beta_1^*)/\text{se}(b_1)$ exceeds $t_{1-\alpha/2}(d)$, where d is the number of df used to estimate σ^2. Similarly, a $(1-\alpha) \times 100\%$ confidence interval for β_1 is given by the set

$$\{\beta_1 \in (b_1 - t_{1-\alpha/2}(d)\text{se}(b_1), b_1 + t_{1-\alpha/2}(d)\text{se}(b_1))\}$$

Computer Output from Heights Data

Typical computer output from a packaged regression program is shown in Table 1 for the heights data. The usual output includes the estimates (3) and their standard errors (7). The fitted mean function is $g(x, \text{b}) = 29.9174 + 0.5417x$; this function is the straight line drawn on Fig. 1. The column marked "t-value" displays the ratio of each estimate to its standard error, which is an appropriate statistic for testing the hypotheses that each corresponding coefficient is equal to zero against either a one-tailed or two-tailed alternative. The column marked "Pr(> |t|)" is the significance level of this test assuming a two-sided alternative, based on the $t(n-2)$ distribution. In this example the p-values are zero to four decimals, and strong evidence is present that the intercept is nonzero given the slope, and that the slope is nonzero given the intercept. The estimated slope of about 0.54 suggests that each inch increase on mother's height corresponds to an increase in daughter's height of only about 0.54 inches, which indicates that tall mothers have tall daughters but not as tall as themselves. This could have been anticipated from (3): Assuming that heights of daughters and mothers are equally variable, we will have $\text{SD}_x \approx \text{SD}_y$ and so $b_1 \approx r_{xy}$, the correlation. As the scale-free correlation coefficient is always in $[-1, 1]$, the slope must also be in the range. This observation of *regression toward the mean* is the origin of the term *regression* for the study of conditional distributions.

Also included in Table 1 are the estimate s of σ, the degrees of freedom associated with s^2, and $R^2 = r_{xy}^2$. This latter value is usually interpreted as a summary of the comparison of the fit of model (1) with the fit of the "null" mean function

$$\mu_0(Y|X = x) = \beta_0 \tag{8}$$

Mean function (8) asserts that the mean of $Y|X$ is the same for all values of X. Under this mean function the least squares estimate of β_0 is just the sample mean m_y and the residual sum of squares is $(n-1)\text{SD}_y^2$. Under mean function (1), the simple linear regression mean function, the residual sum of squares is given by Equation (5). The proportion of variability unexplained by regression on X is just the ratio of these two residual sums of squares:

$$\begin{aligned}\text{Unexplained variability} &= \frac{(n-1)\text{SD}_y^2(1-R^2)}{(n-1)\text{SD}_y^2} \\ &= 1 - R^2\end{aligned}$$

and so R^2 is the proportion of variability in Y that is explained by the linear regression on X. This same interpretation also applies to multiple linear regression.

An important use of regression is the prediction of future values. Consider predicting the height of a daughter whose mother's height is $X = 61$. Whether data collected on English mother–daughter pairs over 100 years ago is relevant to contemporary mother–daughter pairs is questionable, but if it were, the point prediction would be the estimated mean, $g(61, \text{b}) = 29.9174 + 0.5417 \times 61 \approx 63$ inches. From Fig. 1, even if we knew the mean function exactly, we would not expect the prediction to be prefect because mothers of height 61 inches have daughters of a variety of heights. We therefore expect predictions to have two sources of error: a prediction error of magnitude σ due to the new observation, and an error from estimating the mean function,

$$\text{Var}(\text{Prediction}|X = x^*) = \sigma^2 + \text{Var}(g(x, \text{b}))$$

For simple regression $\text{Var}(g(x, \text{b})) = \text{Var}(b_0 + b_1 x) = \text{Var}(b_0) + x^2\text{Var}(b_1) + x\text{Cov}(b_0, b_1)$. Simplifying and replacing σ^2 by s^2 and taking square roots, we get

$$\text{se}(\text{Prediction}|X = x^*) = s\left(1 + \frac{1}{n} + \frac{(x^* - m_x)^2}{(n-1)\text{SD}_x^2}\right)^{1/2}$$

where the sample size n, sample mean m_x, and sample standard deviation SD_x are all from the data used to estimate β. For the heights data, this standard error at $x^* = 61$ is about 2.3 inches. A 95% prediction interval, based on the $t(n-2)$ distribution, is from 58.5 to 67.4 inches.

Table 1. Typical simple regression computer output, for the heights data

	Estimate	Std. Error	t-value	Pr(> |t|)
b_0	29.9174	1.6225	18.44	0.0000
b_1	0.5417	0.0260	20.87	0.0000

$s = 2.27$, 1373 df, $R^2 = 0.241$.

MULTIPLE LINEAR REGRESSION

The *multiple linear regression model* is an elaboration of the simple linear regression model. We now have a predictor $\mathbf{X}$ with $p \geq 1$ components, $\mathbf{X} = (1, X_1, \ldots, X_p)$. Also, let $\mathbf{x} = (1, x_1, \ldots, x_p)$ be a vector of possible observed values for **X**; the "1" is appended to the left of these quantities to allow for an intercept. Then the mean function in Equation (1) is replaced by

$$\begin{aligned}\mu(Y|\mathbf{X}=\mathbf{x}) &= g(\mathbf{x},\boldsymbol{\beta})\\ &= \beta_0+\beta_1x_1+\cdots+\beta_px_p\\ &= \boldsymbol{\beta}'\mathbf{x}\\ \mathrm{Var}(Y|\mathbf{X}=\mathbf{x}) &= \sigma^2\end{aligned} \tag{9}$$

The parameter vector $\boldsymbol{\beta}=(\beta_0,\ldots,\beta_p)'$ now has $p+1$ components. Equation (9) describes a plane in $p+1$-dimensional space. Each β_j for $j>0$ is called a *partial slope*, and gives the expected change in Y when X_j is increased by one unit, *assuming all other X_k, $k\neq j$, are fixed*. This interpretation can be problematical if changing X_j would require that one or more of the other X_k be changed as well. For example, if X_j was tax rate and X_k was savings rate, changing X_j may necessarily change X_k as well.

Linear models are not really restricted to fitting straight lines and planes because we are free to define **X** as we wish. For example, if the elements of X are different powers or other functions of the same base variables, then when viewed as a function of the base variables the mean function will be curved. Similarly, by including dummy variables, which have values of zero and one only, denoting two possible categories, we can fit separate mean functions to subpopulations in the data (see Ref. 4., Chapter 6).

Estimation

Given data $(y_i,x_{i1},\ldots,x_{ip})$ for $i=1,\ldots,n$, we assume that each case in the data is independent of each other case. This may exclude, for example, time-ordered observations on the same case, or other sampling plans with correlated cases. The least-squares estimates minimize Equation 2, but with $g(\mathbf{x},\mathbf{b}^*)$ from Equation 9 substituting for the simple regression mean function. The estimate s^2 of σ^2 is obtained from Equation 4 but with $d=n-p-1$ df rather than the $n-2$ for simple regression. Numerical methods for least-squares computations are discussed in Ref. 7. High-quality subroutines for least squares are provided by Ref. 8. As with simple regression, the standard least-squares calculations are performed by virtually all statistical computing packages.

For the multiple linear regression model, there is a closed-form solution for **b** available in compact form in matrix notation. Suppose we write $\mathcal{Y}$ to be the $n\times 1$ vector of the response variable and $\mathcal{X}$ to be the $n\times(p+1)$ matrix of the predictors, including a column of ones. The order of rows of $\mathcal{Y}$ and $\mathcal{X}$ must be the same. Then

$$\mathbf{b}=(\mathcal{X}'\mathcal{X})^{-1}\mathcal{X}'\mathcal{Y} \tag{10}$$

provided that the inverse exists. If the inverse does not exist, then there is not a unique least-squares estimator. If the matrix $\mathcal{X}$ is of rank $r\leq p$, then most statistical computing packages resolve the indeterminacy by finding r linearly independent columns of $\mathcal{X}$, resulting in a matrix $\mathcal{X}_1$, and then computing the estimator (10) with $\mathcal{X}_1$ replacing $\mathcal{X}$. This will change interpretation of parameters but not change predictions: All least-squares estimates produce the same predictions.

Equation 10 should never be used in computations, and methods based on decompositions such as the QR decomposition are more numerically stable; see "Linear systems of equation" and Ref. 8.

Distribution

If the $F(Y|\mathbf{X})$ are normal distributions, or if the sample size is large enough, then we will have, assuming $\mathcal{X}$ of full rank,

$$\mathbf{b}\sim \mathrm{N}(\boldsymbol{\beta},\sigma^2(\mathcal{X}'\mathcal{X})^{-1}) \tag{11}$$

The standard error of any of the estimates is given by s times the square root of the corresponding diagonal element of $(\mathcal{X}'\mathcal{X})^{-1}$. Similarly, if $\mathbf{a}'\mathbf{b}$ is any linear combination of the elements of **b**, then

$$\mathbf{a}'\mathbf{b}\sim \mathrm{N}(\mathbf{a}'\boldsymbol{\beta},\sigma^2\mathbf{a}'(\mathcal{X}'\mathcal{X})^{-1}\mathbf{a})$$

In particular, the fitted value at $\mathbf{X}=\mathbf{x}^*$ is given by $\mathbf{x}^{*\prime}\mathbf{b}$, and its variance is $\sigma^2\mathbf{x}^{*\prime}(\mathcal{X}'\mathcal{X})^{-1}\mathbf{x}^*$. A prediction of a future value at $\mathbf{X}=\mathbf{x}^*$ is also given by $\mathbf{x}^{*\prime}\mathbf{b}$, and its variance is given by $\sigma^2+\sigma^2\mathbf{x}^{*\prime}(\mathcal{X}'\mathcal{X})^{-1}\mathbf{x}^*$. Both of these variances are estimated by replacing σ^2 by s^2.

Prescription Drug Cost

As an example, we will use data collected on 29 health plans with pharmacies managed by the same insurance company in the United States in the mid-1990s. The response variable is *Cost*, the average cost to the health plan for one prescription for one day, in dollars. Three aspects of the drug plan under the control of the health plan are *GS*, the usage of generic substitute drugs by the plan, an index between 0, for no substitution and 100, for complete substitution, *RI*, a restrictiveness index, also between 0 and 100, describing the extent to which the plan requires physicians to prescribe drugs from a limited formulary, and *Copay*, the cost to the patient per prescription. Other characteristics of the plan that might influence costs are the average *Age* of patients in the plan, and *RXPM*, the number of prescriptions per year per patient, as a proxy measure of the overall health of the members in the plan. Although primary interest is in the first three predictors, the last two are included to adjust for demographic differences in the plans. The data are from Ref. (4).

Figure 2 is a *scatterplot matrix*. Except for the diagonal, a scatterplot matrix is a two-dimensional array of scatterplots. The variable names on the diagonal label the axes. In Fig. 2, the variable *Age* appears on the horizontal axis of all plots in the fifth column from the left and on the vertical axis of all plots in the fifth row from the top. Each plot in a scatterplot matrix is relevant to a particular one-predictor regression of the variable on the vertical axis given the variable on the horizontal axis. For example, the plot of *Cost* versus *GS* in the first plot in the second column of the scatterplot matrix is relevant for the regression of *Cost* on *GS* ignoring the other variables. From the first row of plots, the mean of *Cost* generally decreases as predictors increase,

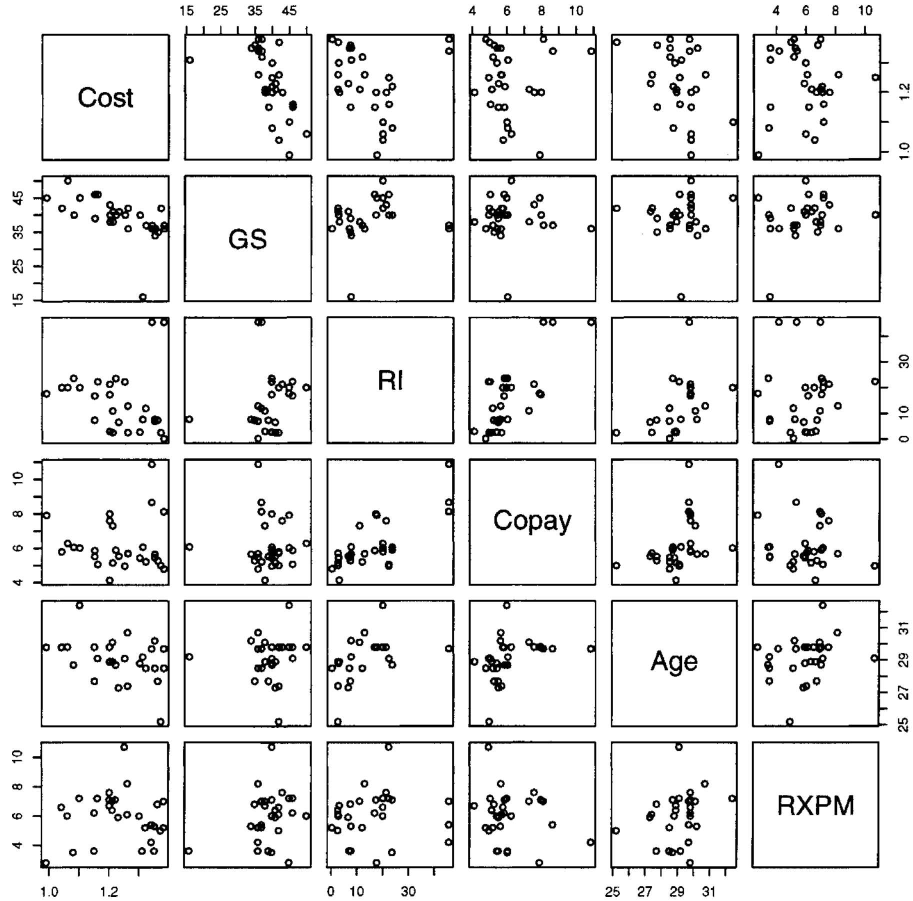


Figure 2. Scatterplot matrix for the drug cost example.

except perhaps *RXPM* where there is not any obvious dependence. This summary is clouded, however, by a few unusual points, in particular one health plan with a very low value for *GS* and three plans with large values of *RI* that have relatively high costs. The scatterplot matrix can be very effective in helping the analyst focus on possibly unusual data early in an analysis.

The pairwise relationships between the *predictors* are displayed in most other frames of this plot. Predictors that have nonlinear joint distributions, or outlying or separated points, may complicate a regression problem; Refs. 4 and 9 present methodology for using the scatterplot matrix to choose transformations of predictors for which a linear regression model is more likely to provide a good approximation.

Table 2 gives standard computer output for the fit of a multiple linear regression model with five predictors. As in simple regression, the value of R^2 gives the proportion of variability in the response explained by the predictors; about half the variability in *Cost* is explained by this regression. The estimated coefficient for *GS* is about -0.012, which suggests that, if all other variables could be held fixed, increasing GI by 10 units is expected to change *Cost* by $10 \times -.012 = \$-0.12$, which is a relatively large change. The t-test for the coefficient for *GS* equal to zero has a very small p-value, which suggests that this coefficient may indeed by nonzero. The coefficient for *Age* also has a small p-value and plans with older members have lower cost per prescription per day. Adjusted for the other predictors, *RI* appears to be unimportant, whereas the coefficient for *Copay* appears to be of the wrong sign.

Table 2. Regression output for the drug cost data.

	Estimate	Std. Error	t-value	Pr(> |t|)
(Intercept)	2.6829	0.4010	6.69	0.0000
GS	−0.0117	0.0028	−4.23	0.0003
RI	0.0004	0.0021	0.19	0.8483
Copay	0.0154	0.0187	0.82	0.4193
Age	−0.0420	0.0141	−2.98	0.0068
RXPM	0.0223	0.0110	2.03	0.0543

$s = 0.0828$, df $= 23$, $R^2 = 0.535$.

Model Comparison. In some regression problems, we may wish to test the null hypothesis NH that a subset of the β_j are simultaneously zero versus the alternative AH that at least one in the subset is nonzero. The usual procedure is to do a *likelihood ratio test*: (*1*) Fit both the NH and the AH models and save the residual sum of squares and the residual df; (*2*) compute the statistic

$$F = \frac{\mathrm{RSS}_{\mathrm{NH}} - \mathrm{RSS}_{\mathrm{AH}}/(\mathrm{df}_{\mathrm{NH}} - \mathrm{df}_{\mathrm{AH}})}{\mathrm{RSS}_{\mathrm{AH}}/\mathrm{df}_{\mathrm{AH}}}$$

Under the normality assumption, the numerator and denominator are independent multiples of χ^2 random variables, and F has an $F(\mathrm{df}_{\mathrm{NH}} - \mathrm{df}_{\mathrm{AH}}\,\mathrm{df}_{\mathrm{AH}})$ distribution, which can be used to get significance levels. For example, consider testing the null hypothesis that the mean function is given by Equation 8, which asserts that the mean function does not vary with the predictors versus the alternative given by Equation 9. For the drug data, $F = 5.29$ with $(5, 23)$ df, $p = 0.002$, which suggests that at least one of the $\beta_j, j \geq 1$ is nonzero.

Model Selection/Variable Selection. Although some regression models are dictated by a theory that specifies which predictors are needed and how they should be used in the problem, many problems are not so well specified. In the drug cost example, *Cost* may depend on the predictors as given, on some subset of them, or on some other functional form other than a linear combination. Many regression problems will therefore include a *model selection* phase in which several competing specifications for the mean function are to be considered. In the drug cost example, we might consider all $2^5 = 32$ possible mean functions obtained using subsets of the five base predictors, although this is clearly only a small fraction of all possible sensible models. Comparing models two at a time is at best inefficient and at worst impossible because the likelihood ratio tests can only be used to compare models if the null model is a special case of the alternative model.

One important method for comparing models is based on estimating a criterion function that depends on both lack of fit and complexity of the model (see also “Information theory.”) The most commonly used method is the *Akaike information criterion*, or *AIC*, given for linear regression by

$$AIC = n\log(\text{Residual sum of squares}) + 2(p+1)$$

where $p + 1$ is the number of estimated coefficients in the mean function. The model that minimizes *AIC* is selected, even if the difference in *AIC* between two models is trivially small; see Ref. 10. For the drug cost data, the mean function with all five predictors has $AIC = -139.21$. The mean function with minimum *AIC* excludes only *RI*, with $AIC = -141.16$. The fitted mean function for this mean function is $m(Y|\mathbf{X} = \mathbf{x}) = 2.6572 - 0.0117\,GS + 0.0181\,Copay - 0.0417Age + 0.0229\,RXPM$. Assuming the multiple linear regression model is appropriate for these data, this suggests that the restrictiveness of the formulary is not related to cost after adjusting for the other variables, plans with more *GS* are associated with lower costs. Both *Copay* and *Age* seem to have the wrong sign.

An alternative approach to model selection is *model aggregation*, in which a probability or weight is estimated for each candidate model, and the “final” model is a weighted combination of the individual models; see Ref. 11 for a Bayesian approach and Ref. 12 for a frequentist approach.

Parameter Interpretation. If the results in Table 2 or the fitted model after selection were a reasonable summary of the conditional mean of *Cost* given the predictors, how can we interpret the parameters? For example, can we infer than increasing *GS* would decrease *Cost*? Or, should we be more cautious and only infer that plans with higher *GS* are *associated* with lower values of *Cost*? The answer to this question depends on the way that the data were generated. If *GS* were assigned to medical plans using a random mechanism, and then we observed *Cost* after the random assignment, then inference of causation could be justified. The lack of randomization in these data could explain the wrong sign for *Copay*, as it is quite plausible that plans raise the copayment in response to higher costs. For observational studies like this one, causal inference based on regression coefficients is problematical, but a substantial literature exists on methods for making causal inference from observational data; see Ref. 13.

Diagnostics. Fitting regression models is predicated upon several assumptions about $F(Y|\mathbf{X})$. Should any of these assumptions fail, then a fitted regression model may not provide a useful summary of the regression problem. For example, if the true mean function were $\mathrm{E}(Y|X = x) = \beta_0 + \beta_1 x + \beta_2 x^2$, then the fit of the simple linear regression model (1) could provide a misleading summary if β_2 were substantially different from zero. Similarly, if the assumed mean function were correct but the variance function was not constant, then estimates would no longer be efficient, and tests and confidence intervals could be badly in error.

Regression diagnostics are a collection of graphical and numerical methods for checking assumptions concerning the mean function and the variance function. In addition, these methods can be used to detect *outliers*, a small fraction of the cases for which the assumed model is incorrect, and *influential cases* (14), which are cases that if deleted would substantially change the estimates and inferences. Diagnostics can also be used to suggest remedial action like transforming predictors or the response, or adding interactions to a mean function, that could improve the match of the model to the data. Much of the theory for diagnostics is laid out in Ref. 15 see also Refs. 4 and 9.

Many diagnostic methods are based on examining the *residuals*, which for linear models are simply the differences $r_i = y_i - g(\mathbf{x}_i, \mathbf{b}), i = 1, \ldots, n$. The key idea is that, if a fitted model is correct, then the residuals should be unrelated to the fitted values, to any function of the predictors, or indeed to any function of data that was not used in the modeling. This suggests examining plots of the residuals versus functions of the predictors, such as the predictors themselves and also versus fitted values. If these graphs show any pattern, such as a curved mean function or nonconstant variance, we have evidence that the model used does not match the data. Lack of patterns in all plots is consistent with an acceptable model, but not definitive.

Figure 3 shows six plots, the residuals versus each of the predictors, and also the residuals versus the fitted values based on the model with all predictors. Diagnostic analysis should generally be done before any model selection based on the largest sensible mean function. In each plot, the dashed line is a reference horizontal line at zero. The dotted line is the fitted least-squares regression line for a quadratic regression with the response given by the residuals and the predictor given by the horizontal axis. The t-test that the coefficient for the quadratic term when added to the original mean function is zero is a numeric diagnostic

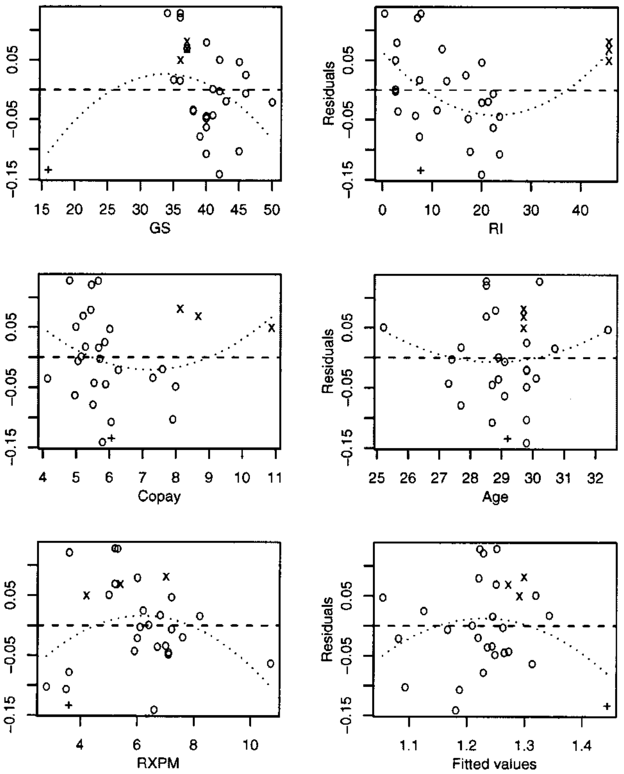


Figure 3. Residual plots for the drug cost data. The "+" symbol indicates the plan with very small *GS*, "x" indicates plans with very high *RI*, and all other plans are indicated with a "o."

that can help interpret the plot. In the case of the plot versus fitted value, this test is called *Tukey's test for non-additivity*, and p-values are obtained by comparing with a normal distribution rather than a t-distribution that is used for all other plots.

In this example, the residual plots display patterns that indicate that the linear model that was fit does not match the data well. The plot for *GS* suggests that the case with a very small value of *GS* might be quite different than the others; the p-value for the lack-of-fit test is about 0.02. Similarly, curvature is evident for *RI* due to the three plans with very high values of *RI* but very high costs. No other plot is particularly troubling, particularly in view of the small sample size. For example, the p-value for Tukey's test corresponding to the plot versus fitted values is about $p = 0.10$. The seemingly contradictory result that the mean function matches acceptably overall but not with regard to *GS* or *RI* is plausible because the overall test will necessarily be less powerful than a test for a more specific type of model failure.

This analysis suggests that the four plans, one with very low *GS* and the other three with very high *RI*, may be cases that should be treated separately from the remaining cases. If we refit without these cases, the resulting residual plots do not exhibit any particular problems. After using *AIC* to select a subset, we end up with the fitted model $g(\mathbf{x}, \mathbf{b}) = 2.394 - 0.014\,GS - 0.004\,RI - 0.024\,Age + 0.020\,RXPM$. In this fitted model, *Copay* is deleted. The coefficient estimate for *GS* is somewhat larger, and the remaining estimates are of the appropriate sign. This seems to provide a useful summary for the data. We would call the four points that were omitted *influential observations*, (14) because their exclusion markedly changes conclusions in the analysis.

In this example, as in many examples, we end up with a fitted model that depends on choices made about the data. The estimated model ignores 4 of 29 data points, so we are admitting that the mean function is not appropriate for all data.

OTHER REGRESSION MODELS

The linear model given by Equation 9 has surprising generality, given that so few assumptions are required. For some problems, these methods will certainly not be useful, for example if the response is not continuous, if the variance depends on the mean, or if additional information about the conditional distributions is available. For these cases, methods are available to take advantage of the additional information.

Logistic Regression

Suppose that the response variable Y can only take on two values, say 1, corresponding perhaps to "success," or 0, corresponding to "failure." For example, in a manufacturing plant where all output is inspected, Y could indicate items that either pass ($Y = 1$) or fail ($Y = 1$) inspection. We may want to study how the probability of passing depends on characteristics such as operator, time of day, quality of input materials, and so on.

We build the logistic regression model in pieces. First, as each Y can only equal 0 or 1, each Y has a Bernoulli distribution, and

$$\begin{aligned}\mu(Y|\mathbf{X} = \mathbf{x}) &= \text{Prob}(Y = 1|\mathbf{X} = \mathbf{x})\\ &= g(\mathbf{x}, \boldsymbol{\beta})\\ \text{Var}(Y|\mathbf{X} = \mathbf{x}) &= g(\mathbf{x}, \boldsymbol{\beta})(1 - g(\mathbf{x}, \boldsymbol{\beta}))\end{aligned} \tag{12}$$

Each observation can have its own probability of success $g(\mathbf{x}, \boldsymbol{\beta})$ and its own variance.

Next, assume that Y depends on $\mathbf{X} = \mathbf{x}$ only through a linear combination $\eta(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p = \boldsymbol{\beta}'\mathbf{x}$. The quantity $\eta(\mathbf{x})$ is called a *linear predictor*. For the multiple linear regression model, we have $g(\mathbf{x}, \boldsymbol{\beta}) = \eta(\mathbf{x})$, but for a binary response, this does not make any sense because a probability is bounded between zero and one. We can make a connection between $g(\mathbf{x}, \boldsymbol{\beta})$ and $\eta(\mathbf{x})$ by assuming that

$$g(\mathbf{x}, \boldsymbol{\beta}) = \frac{1}{1 + \exp(-\eta(\mathbf{x}))} \tag{13}$$

This is called logistic regression because the right side of Equation 13 is the logistic function. Other choices for g are possible, using any function that maps from $(-\infty, \infty)$ to (0, 1), but the logistic is adequate for many applications.

To make the analogy with linear regression clearer, Equation 13 is often inverted to have just the linear predictor on the right side of the equation,

$$\log(\frac{g(\mathbf{x},\boldsymbol{\beta})}{1-g(\mathbf{x},\boldsymbol{\beta})}) = \eta(\mathbf{x}) = \boldsymbol{\beta}'\mathbf{x} \tag{14}$$

In this context, the *logit function*, $\log(g(\mathbf{x},\boldsymbol{\beta})/(1-g(\mathbf{x},\boldsymbol{\beta})))$, is called a *link function* that links the parameter of the Bernoulli distribution, $g(\mathbf{x},\boldsymbol{\beta})$, to the linear predictor $\eta(\mathbf{x})$.

Estimation. If we have data $(y_i, \mathbf{x}_i)$, $i = 1,\ldots,n$ that are mutually independent, then we can write the *log-likelihood function* as

$$\begin{aligned}L(\boldsymbol{\beta}) &= \log\left\{\prod_{i=1}^{n}(g(\mathbf{x}_i,\boldsymbol{\beta}))^{y_i}(1-g(\mathbf{x}_i,\boldsymbol{\beta}))^{1-y_i}\right\}\\ &= \log\left\{\prod_{i=1}^{n}\left(\frac{g(\mathbf{x}_i,\boldsymbol{\beta})}{(1-g(\mathbf{x}_i,\boldsymbol{\beta})}\right)^{y_i}(1-g(\mathbf{x}_i,\boldsymbol{\beta}))\right\}\\ &= \sum_{i=1}^{n}\left\{y_i(\mathbf{x}_i'\boldsymbol{\beta}) + \log\left(1-\frac{1}{(1+\exp(-\mathbf{x}_i'\boldsymbol{\beta}))}\right)\right\}\end{aligned}$$

Maximum likelihood estimates are obtained to be the values of $\boldsymbol{\beta}$ that maximize this last equation. Computations are generally done using Newton–Raphson iteration or using a variant called Fisher scoring; see Ref. 16; for book-length treatments of this topic, see Refs. 17 and 18.

Poisson Regression

When the response is the count of the number of independent events in a fixed time period, Poisson regression models are often used. The development is similar to the Bernoulli case. We first assume that $Y|X = \mathbf{x}$ is distributed as a Poisson random variable with mean $\mu(Y|\mathbf{X} = \mathbf{x}) = g(\mathbf{x},\boldsymbol{\beta})$,

$$\text{Prob}(Y = y|\mathbf{X} = \mathbf{x}) = \frac{g(\mathbf{x},\boldsymbol{\beta})^y}{y!}\exp(-g(\mathbf{x},\boldsymbol{\beta}))$$

For the Poisson, $0 < \mu(Y|\mathbf{X} = \mathbf{x}) = \text{Var}(Y|\mathbf{X} = \mathbf{x}) = g(\mathbf{x},\boldsymbol{\beta})$. The connection between Y and $\mathbf{X}$ is assumed to be through the linear predictor $\eta(\mathbf{x})$, and for a log-linear model, we assume that

$$g(\mathbf{x},\boldsymbol{\beta}) = \exp(\eta(\mathbf{x}))$$

giving the exponential mean function, or inverting we get the log-link,

$$\eta(\mathbf{x}) = \log(g(\mathbf{x},\boldsymbol{\beta}))$$

Assuming independence, the log-likelihood function can be shown to be equal to

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n}\{y_i(\mathbf{x}_i{'}\boldsymbol{\beta}) - \exp(\mathbf{x}_i'\boldsymbol{\beta})\}$$

Log-linear Poisson models are discussed in Ref. 19.

There are obvious connections between the logistic and the Poisson models briefly described here. Both of these models as well as the multiple linear regression model assuming normal errors, are examples of *generalized linear models*, described in Ref. 16.

Nonlinear Regression

Nonlinear regression refers in general to any regression problem for which the linear regression model does not hold. Thus, for example, the logistic and log-linear Poisson models are nonlinear models; indeed nearly *all* regression problems are nonlinear.

However, it is traditional to use a narrower definition for nonlinear regression that matches the multiple linear regression model except that the mean function $\mu(Y|\mathbf{X} = \mathbf{x}) = g(\mathbf{x},\boldsymbol{\beta})$ is a *nonlinear* function of the parameters $\boldsymbol{\beta}$. For example, the mean relationship between X = age of a fish and Y = length of the fish is commonly described using the *von Bertalanffy* function,

$$\text{E}(Y|X = x) = L_\infty(1-\exp(-K(x-x_0)))$$

The parameters $\boldsymbol{\beta} = (L_\infty, K, x_0)'$ to be estimated are the maximum length L_∞ for very old fish; the growth rate K, and $x_0 < 0$, which allows fish to have positive length at birth.

As with the linear model, a normality assumption for $Y|\mathbf{X}$ is not required to obtain estimates. An estimator $\mathbf{b}$ of $\boldsymbol{\beta}$ can be obtained by minimizing Equation (2), and the estimate of σ^2 assuming constant variance from Equation (4). Computations for a nonlinear mean function are much more difficult; see "Least squares approximation". The nonlinear regression problem generally requires an iterative computational method for solution (7) and requires reasonable starting values for the computations. In addition, the objective function (2) may be multimodal and programs can converge to a local rather than a global minimum. Although software is generally available in statistical packages and in mathematical programming languages, the quality of the routines available is more variable and different packages may give different answers. Additionally, even if normality is assumed, the estimate of $\boldsymbol{\beta}$ is normally distributed only in large samples, so inferences are approximate and particularly in small samples may be in error. See Ref. 20 for book-length treatment.

Nonparametric Regression

For the limited and important goal of learning about the mean function, several newer approaches to regression have been proposed in the last few years. These methods either weaken assumptions or are designed to meet particular goals while sacrificing other goals.

The central idea behind *nonparametric regression* is to estimate the mean function $\mu(Y|\mathbf{X} = \mathbf{x})$ without assuming any particular parametric form for the mean function. In the special case of one predictor, the Naradaya–Watson kernel regression estimator is the fundamental method. It estimates $\mu(Y|X = x)$ at any particular x by a weighed average of the y_i with weights determined by $|x_i - x|$, so points close to x have higher weight than do points far away. In particular, If $H(u)$ is a symmetric unimodal function,

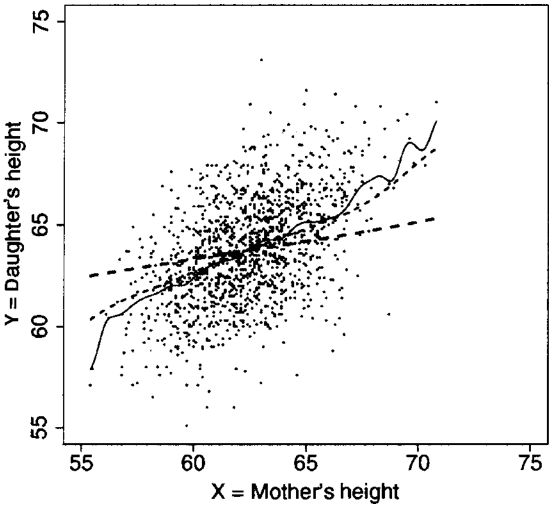


Figure 4. Three Naradaya–Watson kernel smoothing estimates of the mean for the heights data, with $h = 1$ for the solid line and $h = 3$ for the dashed line and $h = 9$ for the dotted line.

then the estimated mean function is

$$m(Y|X = x) = \sum_{i=1}^{n} w_i(h) y_i / \sum_{j=1}^{n} w_j(h)$$
$$w_i(h) = \frac{1}{h} H\left(\frac{x_i - x}{h}\right)$$

One choice for H is the standard normal density function, but other choices can have somewhat better properties. The *bandwidth h* is selected by the analyst; small values of h weigh cases with $|x_i - x|$ small heavily while ignoring other cases, giving a very rough estimate. Choosing h large weighs all cases nearly equally, giving a very smooth, but possibly biased, estimate, as shown in Fig. 4. The bandwidth must be selected to balance bias and smoothness. Other methods for nonparametric regression include smoothing splines, local polynomial regression, and wavelets, among others; see Ref. 21.

Semiparametric Regression

A key feature of nonparametric regression is using nearby observations to estimate the mean at a given point. If the predictor is in many dimensions, then for most points $\mathbf{x}$, there may be either no points or at best just a few points that are nearby. As a result, nonparametric regression does not scale well because of this *curse of dimensionality*, Ref. 22.

This has led to the proposal of *semiparametric* regression models. For example, the *additive regression model*, Refs. 23, 24, suggests modeling the mean function as

$$\mu(Y|\mathbf{X} = \mathbf{x}) = \sum_{j=1}^{p} g_j(x_j)$$

where each g_j is a function of just one predictor x_j that can be estimated nonparametrically. Estimates can be obtained by an iterative procedure that sequentially estimates each of the g_j, continuing until convergence is obtained. This type of model can also be used in the generalized linear model framework, where it is called a *generalized additive model*.

Robust Regression

Robust regression was developed to address the concern that standard estimates such as least-squares or maximum likelihood estimates may be highly unstable in the presence of outliers or other very large errors. For example, the least-squares criterion (2) may be replaced by

$$\mathbf{b} = \arg\min_{\mathbf{b}*} \sum_{i=1}^{n} \rho\{|y_i - g(x, \mathbf{b}^*)|\}$$

where ρ is symmetric about zero and may downweight observations for which $|y_i - g(x, \mathbf{b}^*)|$ is large. The methodology is presented in Ref. 25, although these methods seem to be rarely used in practice, perhaps because they give protection against outliers but not necessarily against model misspecifications, Ref. 26.

Regression Trees

With one predictor, a *regression tree* would seek to replace the predictor by a discrete predictor, such that the predicted value of Y would be the same for all X in the same discrete category. With two predictors, each category created by discretizing the first variable could be subdivided again according to a discrete version of the second predictor, which leads to a tree-like structure for the predictions. Basic methods for regression trees are outlined in Refs. 27 and 28. The exact methodology for implementing regression trees is constantly changing and is an active area of research; see "Machine learning."

Dimension Reduction

Virtually all regression methods described so far require assumptions concerning some aspect of the conditional distributions $F(Y|\mathbf{X})$, either about the mean function on some other characteristic. *Dimension reduction regression* seeks to learn about $F(Y|\mathbf{X})$ but with minimal assumptions. For example, suppose $\mathbf{X}$ is a p-dimensional predictor, now not including a "1" for the intercept. Suppose we could find a $r \times p$ matrix $\mathbf{B}$ of minimal rank r such that $F(Y|\mathbf{X}) = F(Y|\mathbf{BX})$, which means that all dependence of the response on the predictor is through r combinations of the predictors. If $r \ll p$, then the resulting regression problem is of much lower dimensionality and can be much easier to study. Methodology for finding $\mathbf{B}$ and r with no assumptions about $F(Y|\mathbf{X})$ is a very active area of research; see Ref. 29 for the foundations; and references to this work for more recent results.

BIBLIOGRAPHY

1. S. M. Stigler, *The History of Statistics: the Measurement of Uncertainly before 1900*. Cambridge MA: Harvard University Press, 1986.

2. K. Pearson and S. Lee, One the laws of inheritance in man. *Biometrika*, **2**: 357–463, 1903.
3. G. Oehlert, *A First Course in Design and Analysis of Experiments*. New York: Freeman, 2000.
4. S. Weisberg, *Applied Linear Regression, Third Edition*. New York: John Wiley & Sons, 2005.
5. R. Christensen, *Plane Answers to Complex Questions: The Theory of Linear Models*. New York: Sparinger-Verlag Inc, 2002.
6. B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*. Boca Raton, FL: Chapman & Hall Ltd, 1993.
7. C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. SIAM [Society for Industrial and Applied Mathematics], 1995.
8. LAPACK Linear Algebra PACKAGE. http://www.netlib.org/lapack/. 1995.
9. R. Dennis Cook and S. Weisberg, *Applied Regression Including Computing and Graphics*. New York: John Wiley & Sons, 1999.
10. C.-L Tsai and Allan D. R. McQuarrie. *Regression and Time Series Model Selection*. Singapore: World Scientific, 1998.
11. J. A. Hoeting, D. Madigan, A. E. Reftery, and C. T. Volinsky, Bayesian model averaging: A tutorial. *Statistical Science*, **14**(4): 382–401, 1999.
12. Z. Yuan and Y. Yang, Combining linear regression models: When and how? *Journal of the Amerian Statistical Association*, **100**(472): 1202–1214, 2005.
13. P. R. Rosenbaum, *Observational Studies*. New York: Springer-Verlag Inc, 2002.
14. R. D. Cook, Detection of influential observation in linear regression. *Technometrics*, **19**: 15–18, 1977.
15. R. D. Cook and S. Weisberg, *Residuals and Influence in Regression*. Boca Raton, FL: Chapman & Hall Ltd, available online at www.stat.umn.edu/rir, 1982.
16. P. McCullagh and J. A. Nelder, *Generalized Linear Models, Second Edition*. Boca Raton, FL: Chapman & Hall Ltd, 1989.
17. D. W. Hosmer and S. Lemeshow, *Applied Logistic Regression, Second Edition*. New York: John Wiley & Sons, 2000.
18. D. Collett, *Modelling Binary Data, Second Edition*. Boca Raton, FL: Chapman & Hall Ltd, 2003.
19. A. Agresti, *Categorical Data Analysis, Second Edition*. New York: John Wiley & Sons, 2002.
20. D. M. Bates and D. G. Watts, *Nonlinear Regression Analysis and Its Applications*. New York: John Wiley & Sons, 1988.
21. J. S. Simonoff, *Smoothing Methods in Statistics*. New York: Springer-Verlag Inc., 1996.
22. R. E. Bellman, *Adaptive Control Processes*. Princeton NJ: Princeton University Press, 1961.
23. T. Hastie and R. Tibshirani, *Generalized Additive Models*. Boca Raton, FL: Chapman & Hall Ltd, 1999.
24. P. J. Green and B. W. Silverman, *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*. Boca Raton, FL: Chapman & Hall Ltd, 1994.
25. R. G. Staudte and S. J. Sheather, *Robust Estimation and Testing*. New York: John Wiley & Sons, 1990.
26. R. D. Cook, D. M. Hawkins, and S. Weisberg, Comparison of model misspecification diagnostics using residuals from least mean of squares and least median of squares fits, *Journal of the American Statistical Association*, **87**: 419–424, 1992.
27. D. M. Hawkins, FIRM: Formal inference-based recursive modeling, *The American Statistician*, **45**: 155, 1991.
28. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Boca Raton, FL: Wadsworth Adv. Book Prog., 1984.
29. R. Dennis Cook, *Regression Graphics: Ideas for Studying Regressions through Graphics*. New York: John Wiley & Sons, 1998.

SANFORD WEISBERG
University of Minnesota,
School of Statistics
Minneapolis, Minnesota

ROUNDING ERRORS

INTRODUCTION

Human beings are in constant need of making bigger and faster computations. Over the past four centuries, many machines were created for this purpose, and 50 years ago, actual electronic computers were developed specifically to perform scientific computations. The first mechanical calculating machines were *Schikard's machine* (1623, Germany), the *Pascaline* (1642, France), followed by *Leibniz's machine* (1694, Germany). *Babbage's analytical machine* (1833, England) was the first attempt at a mechanical computer, and the first mainframe computer was the *Z4 computer of K. Zuse* (1938, Germany).

Until the beginning of the twentieth century, computations were only done on integer numbers. To perform efficient real numbers computations, it was necessary to wait until the birth of the famous BIT *(BInary digiT)*, which was introduced by C. Shannon (1937, USA) in his PhD thesis. Shannon's work imposed electronics for the building of computers and, then, the base 2 for coding integer or real numbers, although other bases have been tried. It has now been established that the base 2 is the most efficient base on computers for numerical computations, although the base 10 may still be used on pocket calculators.

For coding real numbers, one also has to determine the kind of coding they want to use. The decimal fixed-point notation was introduced at the end of the sixteenth century consecutively by S. Stévin (1582, France), J. Bürgi (1592, Switzerland), and G. Magini (1592, Italy). It remains the notation used worldwide today. Although it is the most natural notation for mental calculations, it is not very efficient for automatic computations. In fact, on this subject, one can say that nothing has changed since J. Napier's logarithm (1614, Scotland) and W. Oughtred's slide rule (1622, England). Logarithms were introduced by J. Napier to make multiplication easier (using logarithm, multiplication becomes addition). Three centuries later, the same idea was kept for the coding of real numbers on computers and led to the floating-point representation (see the next section).

But whatever the representation is on computer, it is a finite representation, like for computations by hand. So, at each operation, because the result needs to be truncated (but is in general), an error may appear that is called the rounding error. Scientists have been well aware of this for four centuries. In the nineteenth century, when numerical computations were presented in an article, they were systematically followed by errors computations to justify the validity of the results. In 1950, in his famous article on eigenvalue computation with his new algorithm, C. Lanczos devoted 30% of his paper to error computation. Unfortunately, this use has completely disappeared since the beginning of the 1960s because of the improvement of computers. When eight billion floating-point operations are performed in one second on a processor, it seems impossible to quantify the rounding error even though neglecting rounding errors may lead to catastrophic consequences.

For instance, for real-time applications, the discretization step may be $h = 10^{-1}$ second. One can compute the absolute time by performing $t_{abs} = t_{abs} + h$ at each step or performing $icount = icount + 1$; $t_{abs} = h * icount$, where *icount* is correctly initialized at the beginning of the process. Because the real-number representation is finite on computers, only a finite number of them can be exactly coded. They are called floating-point numbers. The others are approximated by a floating-point number. Unfortunately, $h = 10^{-1}$ is not a floating-point number. Therefore, each operation $t_{abs} = t_{abs} + h$ generates a small but nonzero error. One hundred hours later, this error has grown to about 0.34 second. It really happened during the first Gulf war (1991) in the control programs of Patriot missiles, which were to intercept Scud missiles (1). At 1600 km/h, 0.34 second corresponds to approximatively 500 meters, the interception failed and 28 people were killed. With the second formulation, whatever the absolute time is, if no overflow occurs for *icount*, then the relative rounding error remains below 10^{-15} using the IEEE double precision arithmetic. A good knowledge of the floating-point arithmetic should be required of all computer scientists (2).

The second section is devoted to the description of the computer arithmetic. The third section presents approaches to study: to bound or to estimate rounding errors. The last section describes methods to improve the accuracy of computed results. A goal of this paper is to answer the question in numerical computing, *"What is the computing error due to floating-point arithmetic on the results produced by a program?"*

COMPUTER ARITHMETIC

Representation of Numbers

In a numeral system, numbers are represented by a sequence of symbols. The number of distinct symbols that can be used is called the radix (or the base). For instance, in the decimal system, where the radix is 10, the 10 symbols used are the digits 0,1, . . . ,9. In the binary system, which is used on most computers, the radix is 2; hence, numbers are represented with sequences of 0s and 1s.

Several formats exist to represent numbers on a computer. The representation of integer numbers differs from the one of real numbers. Using a radix b, if unsigned integers are encoded on n digits, they can range from 0 to $b^n - 1$. Hence, an unsigned integer X is represented by a sequence $a_{n-1}a_{n-2} \ldots a_1a_0$ with

$$X = \sum_{i=0}^{n-1} a_i b^i \quad \text{and} \quad a_i \in \{0, \ldots, b-1\}.$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

With a radix 2 representation, signed integers are usually represented using two's complement. With this rule, signed integers range from $-b^{n-1}$ to $b^{n-1}-1$ and the sequence $a_{n-1}a_{n-2}\ldots a_1a_0$ with $a_i \in \{0,\ldots,b-1\}$ represents the number

$$X = -a_{n-1}b^{n-1} + \sum_{i=0}^{n-2} a_i b^i.$$

The opposite of a number in two's complement format can be obtained by inverting each bit and adding 1.

In numerical computations, most real numbers are not exactly represented because only a finite number of digits can be saved in memory. Two representations exist for real numbers:

- the fixed-point format, available on most embedded systems
- the floating-point format, available on classical computers

In fixed-point arithmetic, a number is represented with a fixed number of digits before and after the radix point. Using a radix b, a number X that is encoded on m digits for its magnitude (e.g., its integer part) and f digits for its fractional part is represented by $a_{m-1}\ldots a_0 \cdot a_{-1}\ldots a_{-f}$, with

$$X = \sum_{i=-f}^{m-1} a_i b^i \quad \text{and} \quad a_i \in \{0,\ldots,b-1\}.$$

If $b = 2$, then unsigned values range from 0 to $2^m - 2^{-f}$ and signed values, which are usually represented with the two's complement format, range from -2^{m-1} to $2^{m-1} - 2^{-f}$.

In a floating-point arithmetic using the radix b, a number X is represented by:

- its sign ε_X which is encoded on one digit that equals 0 if $\varepsilon_X = 1$ and 1 if $\varepsilon_X = -1$,
- its exponent E_X, a k digit integer,
- its mantissa M_X, encoded on p digits.

Therefore, $X = \varepsilon_X M_X b^{E_X}$ with

$$M_X = \sum_{i=0}^{p-1} a_i b^{-i} \quad \text{and} \quad a_i \in \{0,\ldots,b-1\}.$$

The mantissa M_X can be written as $M_X = a_0 \,.\, a_1 \ldots a_{p-1}$. Floating-point numbers are usually normalized. In this case, $a_0 \neq 0$, $M_X \in [1, b)$ and the number zero has a special representation. Normalization presents several advantages, such as the uniqueness of the representation (there is exactly one way to write a number in such a form) and the easiness of comparisons (the signs, exponents, and mantissas of two normalized numbers can be tested separately).

The IEEE 754 Standard

The poor definition of the floating-point arithmetic on most computers created the need for a unified standard in floating-point numbers. Indeed, the bad quality of arithmetic operators could heavily affect some results. Furthermore, simulation programs could provide different results from one computer to another, because of different floating-point representations. Different values could be used for the radix, the length of the exponent, the length of the mantissa, and so on. So, in 1985, the IEEE 754 standard (3) was elaborated to define floating-point formats and rounding modes. It specifies two basic formats, both using the radix 2.

- With the single precision format, numbers are stored on 32 bits: 1 for the sign, 8 for the exponent, and 23 for the mantissa.
- With the double precision format, numbers are stored on 64 bits: 1 for the sign, 11 for the exponent, and 52 for the mantissa.

Extended floating-point formats also exist; the standard does not specify their exact size but gives a minimum number of bits for their storage.

Because of the normalization, the first bit in the mantissa must be 1. As this implicit bit is not stored, the precision of the mantissa is actually 24 bits in single precision and 53 bits in double precision.

The exponent E is a k digit signed integer. Let us denote its bounds by E_{min} and E_{max}. The exponent that is actually stored is a biased exponent E_Δ such that $E_\Delta = E + \Delta$, Δ being the bias. Table 1 specifies how the exponent is encoded.

The number zero is encoded by setting to 0 all the bits of the (biased) exponent and all the bits of the mantissa. Two representations actually exist for zero: $+0$ if the sign bit is 0, and -0 if the sign bit is 1. This distinction is consistent with the existence of two infinities. Indeed $1/(+0) = +\infty$ and $1/(-0) = -\infty$. These two infinities are encoded by setting to 1 all the bits of the (biased) exponent and to 0 all the bits from the mantissa. The corresponding nonbiased exponent is therefore $E_{max} + 1$.

NaN (Not a Number) is a special value that represents the result of an invalid operation such as 0/0, $\sqrt{-1}$, or $0 \times \infty$. NaN is encoded by setting all the bits of the (biased) exponent to 1 and the fractional part of the mantissa to any nonzero value.

Denormalized numbers (also called subnormal numbers) represent values close to zero. Without them, as the integer part of the mantissa is implicitly set to 1, there would be no representable number between 0 and $2^{E_{min}}$ but

Table 1. Exponent Coding in Single and Double Precision

			nonbiased		biased	
precision	length k	bias Δ	E_{min}	E_{max}	$E_{min}+\Delta$	$E_{max}+\Delta$
single	8	127	−126	127	1	254
double	11	1023	−1022	1023	1	2046

2^{p-1} representable numbers between $2^{E_{min}}$ and $2^{E_{min}+1}$. Denormalized numbers have a biased exponent set to 0. The corresponding values are:

$$X = \varepsilon_X M_X 2^{E_{min}} \text{ with } \varepsilon_X = \pm 1,$$

$$M_X = \sum_{i=1}^{p-1} a_i 2^{-i}$$

and

$$a_i \in \{0, 1\}.$$

The mantissa M_X can be written as $M_X = 0.a_1 \ldots a_{p-1}$. Therefore, the lowest positive denormalized number is $\underline{u} = 2^{E_{min}+1-p}$. Moreover, denormalized numbers and gradual underflow imply the nice equivalence $a = b \Leftrightarrow a - b = 0$.

Let us denote by $\mathbb{F}$ the set of all floating-point numbers, (i.e., the set of all machine representable numbers). This set, which depends on the chosen precision, is bounded and discrete. Let us denote its bounds by X_{min} and X_{max}. Let x be a real number that is not machine representable. If $x \in (X_{min}, X_{max})$, then $\{X^-, X^+\} \subset \mathbb{F}^2$ exists such as $X^- < x < X^+$ and $(X^-, X^+) \cap \mathbb{F} = \emptyset$. A rounding mode is a rule that, from x, provides X^- or X^+. This rounding occurs at each assignment and at each arithmetic operation. The IEEE 754 standard imposes a correct rounding for all arithmetic operations (+, −, ×, /) and also for the square root. The result must be the same as the one obtained with infinite precision and then rounded. The IEEE 754 standard defines four rounding modes:

- rounding toward $+\infty$ (or upward rounding), x is represented by X^+,
- rounding toward $-\infty$ (or downward rounding), x is represented by X^-,
- rounding toward 0, if x is negative, then it is represented by X^+, if x is positive, then it is represented by X^-,
- rounding to the nearest, x is represented by its nearest machine number. If x is at the same distance of X^- and X^+, then it is represented by the machine number that has a mantissa ending with a zero. With this rule, rounding is said to be *tie to even*.

Let us denote by X the number obtained by applying one of these rounding modes to x. By definition, an *overflow* occurs if $|X| > \max\{|Y| : Y \in \mathbb{F}\}$ and an *underflow* occurs if $0 < |X| < \min\{|Y| : 0 \neq Y \in \mathbb{F}\}$. Gradual underflow denotes the situation in which a number is not representable as a normalized number, but still as a denormalized one.

Rounding Error Formalization

Notion of Exact Significant Digits. To quantify the accuracy of a computed result correctly, the notion of exact significant digits must be formalized. Let R be a computed result and r the corresponding exact result. The number $C_{R,r}$ of exact significant decimal digits of R is defined as the number of significant digits that are in common with r:

$$C_{R,r} = \log_{10}\left|\frac{R+r}{2(R-r)}\right| \qquad (1)$$

This mathematical definition is in accordance with the intuitive idea of decimal significant digits in common between two numbers. Indeed Equation (1) is equivalent to

$$|R - r| = \left|\frac{R+r}{2}\right| 10^{-C_{R,r}} \qquad (2)$$

If $C_{R,r} = 3$, then the relative error between R and r is of the order of 10^{-3}. R and r have therefore three common decimal digits.

However, the value of $C_{R,r}$ may seem surprising if one considers the decimal notations of R and r. For example, if $R = 2.4599976$ and $r = 2.4600012$, then $C_{R,r} \approx 5.8$. The difference caused by the sequences of "0" or "9" is illusive. The significant decimal digits of R and r are really different from the sixth position.

Rounding Error that Occurs at Each Operation. A formalization of rounding errors generated by assignments and arithmetic operations is proposed below. Let X be the representation of a real number x in a floating-point arithmetic respecting the IEEE 754 standard. This floating-point representation of X may be written as $X = \text{fl}(x)$. Adopting the same notations as in Equation (1)

$$X = \varepsilon_X M_X 2^{E_X} \qquad (3)$$

and

$$X = x - \varepsilon_X 2^{E_X - p} \alpha_X \qquad (4)$$

where α_X represents the normalized rounding error.

- with rounding to the nearest, $\alpha_X \in [-0.5, 0.5)$
- with rounding toward zero, $\alpha_X \in [0, 1)$
- with rounding toward $+\infty$ or $-\infty$, $\alpha_X \in [-1, 1)$

Equivalent models for X are given below. The *machine epsilon* is the distance ε from 1.0 to the next larger floating-point number. Clearly, $\varepsilon 2^{1-p}$, p being the length of the mantissa that includes the implicit bit. The relative error on X is no larger than the *unit round-off u:*

$$X = x(1+\delta) \text{ with } |\delta| \leq u \qquad (5)$$

where $u = \varepsilon/2$ with rounding to the nearest and $u = \varepsilon$ with the other rounding modes. The model associated with Equation (5) ignores the possibility of underflow. To take underflow into account, one must modify it to

$$X = x(1+\delta) + \eta \text{ with } |\delta| \leq u \qquad (6)$$

and $|\eta| \leq \underline{u}/2$ with rounding to the nearest and $|\eta| \leq \underline{u}$ with the other rounding modes, $\underline{u}$ being the lowest positive denormalized number.

Let X_1 (respectively X_2) be the floating-point representation of a real number x_1 (respectively x_2)

$$X_i = x_i - \varepsilon_i 2^{E_i - p}\alpha_i \quad \text{for} \quad i = 1, 2 \tag{7}$$

The errors caused by arithmetic operations that have X_1 and X_2 as operands are given below. For each operation, let us denote by E_3 and ε_3 the exponent and the sign of the computed result. α_3 represents the rounding error performed on the result. Let us denote by $\oplus$, $\ominus$, $\otimes$, $\oslash$ the arithmetic operators on a computer.

$$\begin{aligned} X_1 \oplus X_2 = x_1 + x_2 - \varepsilon_1 2^{E_1 - p}\alpha_1 - \varepsilon_2 2^{E_2 - p}\alpha_2 \\ - \varepsilon_3 2^{E_3 - p}\alpha_3 \end{aligned} \tag{8}$$

Similarly

$$\begin{aligned} X_1 \ominus X_2 = x_1 - x_2 - \varepsilon_1 2^{E_1 - p}\alpha_1 + \varepsilon_2 2^{E_2 - p}\alpha_2 \\ - \varepsilon_3 2^{E_3 - p}\alpha_3 \end{aligned} \tag{9}$$

$$\begin{aligned} X_1 \otimes X_2 = x_1 x_2 - \varepsilon_1 2^{E_1 - p}\alpha_1 x_2 - \varepsilon_2 2^{E_2 - p}\alpha_2 x_1 \\ + \varepsilon_1 \varepsilon_2 2^{E_1 + E_2 - 2p}\alpha_1\alpha_2 - \varepsilon_3 2^{E_3 - p}\alpha_3 \end{aligned} \tag{10}$$

By neglecting the fourth term, which is of the second order in 2^{-p}, one obtains

$$\begin{aligned} X_1 \otimes X_2 = x_1 x_2 - \varepsilon_1 2^{E_1 - p}\alpha_1 x_2 - \varepsilon_2 2^{E_2 - p}\alpha_2 x_1 \\ - \varepsilon_3 2^{E_3 - p}\alpha_3 \end{aligned} \tag{11}$$

By neglecting terms of an order greater than or equal to 2^{-2p}, one obtains

$$\begin{aligned} X_1 \oslash X_2 = \frac{x_1}{x_2} - \varepsilon_1 2^{E_1 - p}\frac{\alpha_1}{x_2} + \varepsilon_2 2^{E_2 - p}\alpha_2 \frac{x_1}{x_2^2} \\ - \varepsilon_3 2^{E_3 - p}\alpha_3 \end{aligned} \tag{12}$$

In the case of an addition with operands of the same sign,

$$E_3 = \max(E_1, E_2) + \delta \text{ with } \delta = 0 \text{ or } \delta = 1$$

The order of magnitude of the two terms that result from the rounding errors on X_1 and X_2 is at most $2^{E_3 - p}$. The relative error on $X_1 \oplus X_2$ remains of the order of 2^{-p}. This operation is therefore relatively stable: It does not induce any brutal loss of accuracy.

The same conclusions are valid in the case of a multiplication, because

$$E_3 = E_1 + E_2 + \delta, \text{ with } \delta = 0 \text{ or } \delta = -1$$

and in the case of a division, because

$$E_3 = E_1 - E_2 + \delta, \text{ with } \delta = 0 \text{ or } \delta = 1$$

In the case of a subtraction with operands of the same sign, $E_3 = \max(E_1, E_2) - k$. If X_1 and X_2 are very close, then k may be large. The order of magnitude of the absolute error remains $2^{\max(E_1,E_2)-p}$, but the order of magnitude of the relative error is $2^{\max(E_1,E_2)-p-E_3} = 2^{-p+k}$. In one operation, k exact significant bits have been lost: It is the so-called *catastrophic cancellation*.

Rounding Error Propagation. A numerical program is a sequence of arithmetic operations. The result R provided by a program after n operations or assignments can be modeled to the first order in 2^{-p} as:

$$R \approx r + \sum_{i=1}^{n} g_i(d) 2^{-p}\alpha_i \tag{13}$$

where r is the exact result, p is the number of bits in the mantissa, α_i are independent uniformly distributed random variables on $[-1, 1]$ and $g_i(d)$ are coefficients depending exclusively on the data and on the code. For instance, in Equation (12), $g_i(d)$ are $\frac{1}{x_2}$ and $\frac{x_1}{x_2^2}$.

The number $C_{R,r}$ of exact significant bits of the computed result R is

$$C_{R,r} = \log_2\left|\frac{R + r}{2(R - r)}\right| \tag{14}$$

$$C_{R,r} \approx -\log_2\left|\frac{R - r}{r}\right| = p - \log_2\left|\sum_{i=1}^{n} g_i(d)\frac{\alpha_i}{x}\right| \tag{15}$$

The last term in Equation (15) represents the loss of accuracy in the computation of R. This term is independent of p. Therefore, assuming that the model at the first order established in Equation (13) is valid, the loss of accuracy in a computation is independent of the precision used.

Impact of Rounding Errors on Numerical Programs

With floating-point arithmetic, rounding errors occur in numerical programs and lead to a loss of accuracy, which is difficult to estimate. Another consequence of floating-point arithmetic is the loss of algebraic properties. The floating-point addition and the floating-point multiplication are commutative, but not associative. Therefore the same formula may generate different results depending on the order in which arithmetic operations are executed. For instance, in IEEE single precision arithmetic with rounding to the nearest,

$$(-10^{20} \oplus 10^{20}) \oplus 1 = 1 \tag{16}$$

but

$$-10^{20} \oplus (10^{20} \oplus 1) = 0 \tag{17}$$

Equation (17) causes a so-called *absorption*. Indeed, an absorption may occur during the addition of numbers with very different orders of magnitude: The smallest number may be lost.

Furthermore, with floating-point arithmetic, the multiplication is not distributive with respect to the addition. Let A, B, and C be floating-point numbers, $A \otimes (B \oplus C)$ may not be equal to $(A \otimes B) \oplus (A \otimes C)$. For instance, in IEEE single precision arithmetic with rounding to the nearest, if A, B and C are respectively assigned to 3.3333333, 12345679 and 1.2345678, for $A \otimes (B \oplus C)$ and $(A \otimes B) \oplus (A \otimes C)$, one obtains 41152264 and 41152268, respectively.

Impact on Direct Methods. The particularity of a direct method is to provide the solution to a problem in a finite number of steps. In infinite precision arithmetic, a direct method would compute the exact result. In finite precision arithmetic, rounding error propagation induces a loss of accuracy and may cause problems in branching statements. The general form of a branching statement in a program is

IF condition THEN sequence 1 ELSE sequence 2.

If the condition is satisfied, then a sequence of instructions is executed, otherwise another sequence is performed. Such a condition can be for instance $A \geq B$. In the case when A and B are intermediate results already affected by rounding errors, the difference between A and B may have no exact significant digit. The choice of the sequence that is executed may depend on rounding error propagation. The sequence chosen may be the wrong one: It may be different from the one that would have been chosen in exact arithmetic.

For instance, depending on the value of the discriminant, a second degree polynomial has one (double) real root, two real roots, or two conjugate complex roots. The discriminant and the roots of the polynomial $0.3x^2 - 2.1x + 3.675$ obtained using IEEE single precision arithmetic with rounding to the nearest are $D = -5.185604\text{E-}07$, $x = 3.4999998 \pm 1.2001855\text{E}03\ i$. Two conjugate complex roots are computed. But the exact values are $D = 0$, $x = 3.5$. The polynomial actually has one double real root. In floating-point arithmetic, rounding errors occur because of both assignments and arithmetic operations. Indeed the coefficients of the polynomial are not floating-point numbers. Therefore, the computed discriminant has no exact significant digit, and the wrong sequence of instructions is executed.

Impact on Iterative Methods. The result of an iterative method is defined as the limit L of a first-order recurrent sequence:

$$L = \operatorname*{Lim}_{n \to \infty} U_n \quad \text{with} \quad U_{n+1} = \mathcal{F}(U_n) \quad \mathbb{R}^m \xrightarrow{\mathcal{F}} \mathbb{R}^m \tag{18}$$

Because of rounding error propagation, the same problems as in a direct method may occur. But another difficulty is caused by the loss of the notion of limit on a computer. Computations are performed until a stopping criterion is satisfied. Such a stopping criterion may involve the absolute error:

$$\|U_n - U_{n-1}\| \leq \varepsilon \tag{19}$$

or the relative error:

$$\|U_n - U_{n-1}\| \leq \varepsilon \, \|U_{n-1}\| \tag{20}$$

It may be difficult to choose a suitable value for ε. If ε is too high, then computations stop too early and the result is very approximative. If ε is too low, useless iterations are performed without improving the accuracy of the result, because of rounding errors. In this case, the stopping criterion may never be satisfied because the chosen accuracy is illusive. The impact of ε on the quality of the result is shown in the numerical experiment described below.

Newton's method is used to compute a root of

$$f(x) = x^4 - 1002x^3 + 252001x^2 - 501000x + 250000 \tag{21}$$

The following sequence is computed:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{with} \quad x_0 = 1100 \tag{22}$$

The exact limit is $L = 500$, which is a double root of f. The stopping criterion is $|x_n - x_{n-1}| \leq \varepsilon |x_{n-1}|$, and the maximum number of iterations is set to 1000. Table 2 shows for several values of ε the last value of n and the error $|x_n - L|$ computed using IEEE double precision arithmetic with rounding to the nearest.

It is noticeable that the optimal order of magnitude for ε is 10^{-11}. The stopping criterion can not be satisfied if $\varepsilon \leq 10^{-12}$: The maximum number of iterations is reached. Furthermore, the error is slightly higher than for $\varepsilon = 10^{-11}$.

Impact on Approximation Methods. These methods provide an approximation of a limit $L = \lim_{h \to 0} L(h)$. This approximation is affected by a global error $E_g(h)$, which consists in a truncation error $E_t(h)$, inherent to the method, and a rounding error $E_r(h)$. If the step h decreases, then the truncation error $E_t(h)$ also decreases, but the rounding error $E_r(h)$ usually increases, as shown in Fig. 1. It may therefore seem difficult to choose the optimal step h_{opt}. The rounding error should be evaluated, because the global error is minimal if the truncation error and the rounding error have the same order of magnitude.

The numerical experiment described below (4) shows the impact of the step h on the quality of the approximation. The second derivative at $x = 1$ of the following function

$$f(x) = \frac{4970x - 4923}{4970x^2 - 9799x + 4830} \tag{23}$$

Table 2. Number of Iterations and Error Obtained Using Newton's Method in Double Precision

ε	n	$\lvert x_n - L \rvert$
10^{-7}	26	3.368976E-05
10^{-8}	29	4.211986E-06
10^{-9}	33	2.525668E-07
10^{-10}	35	1.405326E-07
10^{-11}	127	1.273870E-07
10^{-12}	1000	1.573727E-07
10^{-13}	1000	1.573727E-07

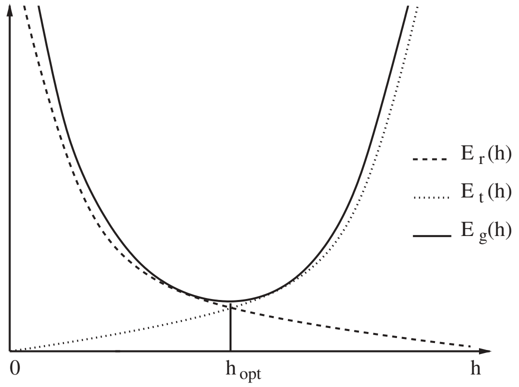


Figure 1. Evolution of the rounding error *$E_r(h)$*, the truncation error *$E_t(h)$* and the global error *$E_g(h)$* with respect to the step h.

is approximated by

$$L(h) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \qquad (24)$$

The exact result is $f''(1) = 94$. Table 3 shows for several steps h the result $L(h)$, and the absolute error $|L(h) - L|$ computed using IEEE double precision arithmetic with rounding to the nearest.

It is noticeable that the optimal order of magnitude for h is 10^{-6}. If h is too low, then the rounding error prevails and invalidates the computed result.

METHODS FOR ROUNDING ERROR ANALYSIS

In this section, different methods of analyzing rounding errors are reviewed.

Forward/Backward Analysis

This subsection is heavily inspired from Refs. 5 and 6. Other good references are Refs. 7–9.

Let X be an approximation to a real number x. The two common measures of the accuracy of X are its *absolute error*

$$E_a(X) = |x - X| \qquad (25)$$

and its *relative error*

$$E_r(X) = \frac{|x - X|}{|x|} \qquad (26)$$

(which is undefined if $x = 0$). When x and X are vectors, the relative error is usually defined with a norm as $\|x - X\|/\|x\|$. This is a *normwise relative error*. A more widely used relative error is the *componentwise relative error* defined by

$$\max_i \frac{|x_i - X_i|}{|x_i|}.$$

It makes it possible to put the individual relative errors on an equal footing.

Table 3. Second Order Approximation of $f''(1) = 94$ Computed in Double Precision

h	$L(h)$	$\|L(h) - L\|$
10^{-3}	−2.250198E+03	2.344198E+03
10^{-4}	7.078819E+01	2.321181E+01
10^{-5}	9.376629E+01	2.337145E−01
4.10^{-6}	9.397453E+01	2.546980E−02
3.10^{-6}	9.397742E+01	2.257732E−02
10^{-6}	9.418052E+01	1.805210E−01
10^{-7}	7.607526E+01	1.792474E+01
10^{-8}	1.720360E+03	1.626360E+03
10^{-9}	−1.700411E+05	1.701351E+05
10^{-10}	4.111295E+05	4.110355E+05

Well-Posed Problems. Let us consider the following mathematical problem *(P)*

$$(P) : \text{given}\, y,\ \text{find}\, x \text{ such that}\, F(x) = y$$

where F is a continuous mapping between two linear spaces (in general $\mathbb{R}^n$ or $\mathbb{C}^n$). One will say that the problem (P) is *well posed* in the sense of Hadamard if the solution $x = F^{-1}(y)$ exists, is unique and F^{-1} is continuous in the neighborhood of y. If it is not the case, then one says that the problem is *ill posed*. An example of ill-posed problem is the solution of a linear system $Ax = b$, where A is singular. It is difficult to deal numerically with ill-posed problems (this is generally done via regularization techniques). That is why we will focus only on well-posed problems in the sequel.

Conditioning. Given a well-posed problem (P), one wants now to know how to measure the difficulty of solving this problem. This measurement will be done via the notion of *condition number*. Roughly speaking, the condition number measures the sensitivity of the solution to perturbation in the data. Because the problem (P) is well posed, one can write it as $x = G(y)$ with $G = F^{-1}$.

The input space (data) and the output space (result) are denoted by $\mathcal{D}$ and $\mathcal{R}$, respectively the norms on these spaces will be denoted $\|\cdot\|_{\mathcal{D}}$ and $\|\cdot\|_{\mathcal{R}}$. Given $\varepsilon > 0$ and let $\mathcal{P}(\varepsilon) \subset \mathcal{D}$ be a set of perturbation Δy of the data y that satisfies $\|\Delta y\|_{\mathcal{D}} \le \varepsilon$, the perturbed problem associated with problem (P) is defined by

$$\text{Find}\, \Delta x \in \mathcal{R} \text{ such that}\, F(x + \Delta x) = y + \Delta y \text{ for a given}\, \Delta y \in \mathcal{P}(\varepsilon)$$

x and y are assumed to be nonzero. The *condition number* of the problem (P) in the data y is defined by

$$\text{cond}(P, y) := \lim_{\varepsilon \to 0} \sup_{\Delta y \in \mathcal{P}(\varepsilon), \Delta y \neq 0} \left\{ \frac{\|\Delta x\|_{\mathcal{R}}}{\|\Delta y\|_{\mathcal{D}}} \right\} \qquad (27)$$

Example 3.1. (summation). Let us consider the problem of computing the sum

$$x = \sum_{i=1}^{n} y_i$$

assuming that $y_i \neq 0$ for all i. One will take into account the perturbation of the input data that are the coefficients y_i. Let $\Delta y = (\Delta y_1, \ldots, \Delta y_n)$ be the perturbation on $y = (y_1, \ldots, y_n)$. It follows that $\Delta x = \sum_{i=1}^{n} \Delta y_i$. Let us endow $\mathcal{D} = \mathbb{R}^n$ with the relative norm $\|\Delta y\|_{\mathcal{D}} = \max_{i=1,\ldots,n} |\Delta y_i|/|y_i|$ and $\mathcal{R} = \mathbb{R}$ with the relative norm $\|\Delta x\|_{\mathcal{R}} = |\Delta x|/|x|$. Because

$$|\Delta x| = |\sum_{i=1}^{n} \Delta y_i| \leq \|\Delta y\|_{\mathcal{D}} \sum_{i=1}^{n} |y_i|,$$

one has[1]

$$\frac{\|\Delta x\|_{\mathcal{R}}}{\|\Delta y\|_{\mathcal{D}}} \leq \frac{\sum_{i=1}^{n} |y_i|}{|\sum_{i=1}^{n} y_i|} \tag{28}$$

This bound is reached for the perturbation Δy such that $\Delta y_i / y_i = \text{sign}(y_i)\|\Delta y\|_{\mathcal{D}}$ where sign is the sign of a real number. As a consequence,

$$\text{cond}\left(\sum_{i=1}^{n} y_i\right) = \frac{\sum_{i=1}^{n} |y_i|}{|\sum_{i=1}^{n} y_i|} \tag{29}$$

Now one has to interpret this condition number. A problem is considered as ill conditioned if it has a large condition number. Otherwise, it is well conditioned. It is difficult to give a precise frontier between well conditioned and ill-conditioned problems. This statement will be clarified in a later section thanks to the rule of thumb. The larger the condition number is, the more a small perturbation on the data can imply a greater error on the result. Nevertheless, the condition number measures the *worst case* implied by a small perturbation. As a consequence, it is possible for an ill-conditioned problem that a small perturbation on the data also implies a small perturbation on the result. Sometimes, such a behavior is even typical.

Remark 1. It is important to note that the condition number is independent of the algorithm used to solve the problem. It is only a characteristic of the problem.

Stability of an Algorithm. Problems are generally solved using an *algorithm*, which is a set of operations and tests that one can consider as the function G defined above given the solution of our problem. Because of the rounding errors, the algorithm is not the function G but rather a function $\hat{G}$. Therefore, the algorithm does not compute $x = G(y)$ but $\hat{x} = \hat{G}(y)$.

The *forward analysis* tries to study the execution of the algorithm $\hat{G}$ on the data y. Following the propagation of the rounding errors in each intermediate variables, the forward analysis tries to estimate or to bound the difference between x and $\hat{x}$. This difference between the exact solution x and the computed solution $\hat{x}$ is called the *forward error*.

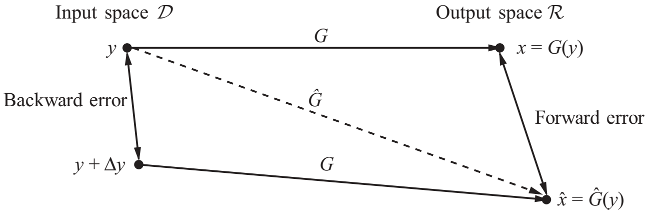


Figure 2. Forward and backward error for the computation of $x = G(y)$.

It is easy to recognize that it is pretty difficult to follow the propagation of all the intermediate rounding errors. The *backward analysis* makes it possible to avoid this problem by working with the function G itself. The idea is to seek for a problem that is actually solved and to check if this problem is "close to" the initial one. Basically, one tries to put the error on the result as an error on the data. More theoretically, one seeks for Δy such that $\hat{x} = G(y + \Delta y)$. Δy is said to be the *backward error* associated with $\hat{x}$. A backward error measures the distance between the problem that is solved and the initial problem. As $\hat{x}$ and G are known, it is often possible to obtain a good upper bound for Δy (generally, it is easier than for the forward error). Figure 2 sums up the principle of the forward and backward analysis.

Sometimes, it is not possible to have $\hat{x} = G(y + \Delta y)$ for some Δy but it is often possible to get Δx and Δy such that $\hat{x} + \Delta x = G(y + \Delta y)$. Such a relation is called a *mixed forward-backward error*.

The *stability* of an algorithm describes the influence of the computation in finite precision on the quality of the result. The backward error associated with $\hat{x} = \hat{G}(y)$ is the scalar $\eta(\hat{x})$ defined by, when it exists,

$$\eta(\hat{x}) = \min_{\Delta y \in \mathcal{D}} \{\|\Delta y\|_{\mathcal{D}} : \hat{x} = G(y + \Delta y)\} \tag{30}$$

If it does not exist, then $\eta(\hat{x})$ is set to $+\infty$. An algorithm is said to be *backward-stable* for the problem (P) if the computed solution $\hat{x}$ has a "small" backward error $\eta(\hat{x})$. In general, in finite precision, "small" means of the order of the rounding unit u.

Example 3.2. (summation). The addition is supposed to satisfy the following property:

$$\hat{z} = z(1 + \delta) = (a + b)(1 + \delta) \text{ with } |\delta| \leq u \tag{31}$$

It should be noticed that this assumption is satisfied by the IEEE arithmetic. The following algorithm to compute the sum $\sum y_i$ will be used.

Algorithm 3.1. Computation of the sum of floating-point numbers

```
function res = Sum($y$)
  $s_1 = y_1$
  for $i = 2 : n$
    $s_{\mathrm{i}} = s_{\mathrm{i}-1} \oplus y_{\mathrm{i}}$
  res = $s_n$
```

Thanks to Equation (31), one can write

$$s_i = (s_{i-1} + y_i)(1 + \delta_i) \text{ with } |\delta_i| \leq u \tag{32}$$

[1] The Cauchy-Schwarz inequality $|\sum_{i=1}^{n} x_i y_i| \leq \max_{i=1,\ldots,n} |x_i| \times \sum_{i=1}^{n} |y_i|$ is used.

For convenience, $1+\theta_j=\prod_{i=1}^{j}(1+\varepsilon_i)$ is written, for $|\varepsilon_i|\leq u$ and $j\in\mathbb{N}$. Iterating the previous equation yields

$$\begin{aligned}\texttt{res} &= y_1(1+\theta_{n-1})+y_2(1+\theta_{n-1})+y_3(1+\theta_{n-2})\\ &\quad+\cdots+y_{n-1}(1+\theta_2)+y_n(1+\theta_1)\end{aligned}\qquad(33)$$

One can interpret the computed sum as the exact sum of the vector z with $z_i=y_i(1+\theta_{n+1-i})$ for $i = 2:n$ and $z_1=y_1(1+\theta_{n-1})$.

As $|\varepsilon_i|\leq u$ for all i and assuming $nu<1$, it can be proved that $|\theta_i|\leq iu/(1-iu)$ for all i. Consequently, one can conclude that the backward error satisfies

$$\eta(\hat{x})=|\theta_{n-1}|\lesssim nu\qquad(34)$$

Because the backward error is of the order of u, one concludes that the classic summation algorithm is backward-stable.

Accuracy of the Solution. How is the accuracy of the computed solution estimated? The accuracy of the computed solution actually depends on the condition number of the problem and on the stability of the algorithm used. The condition number measures the effect of the perturbation of the data on the result. The backward error simulates the errors introduced by the algorithm as errors on the data. As a consequence, at the first order, one has the following *rule of thumb*:

$$\text{forward error}\lesssim\text{condition number}\times\text{backward error}\qquad(35)$$

If the algorithm is backward-stable (that is to say the backward error is of the order of the rounding unit u), then the rule of thumb can be written as follows

$$\text{forward error}\lesssim\text{condition number}\times u\qquad(36)$$

In general, the condition number is hard to compute (as hard as the problem itself). As a consequence, some estimators make it possible to compute an approximation of the condition number with a reasonable complexity.

The rule of thumb makes it possible to be more precise about what were called ill-conditioned and well-conditioned problems. A problem will be said to be ill conditioned if the condition number is greater than $1/u$. It means that the relative forward error is greater than 1 just saying that one has no accuracy at all for the computed solution.

In fact, in some cases, the rule of thumb can be proved. For the summation, if one denotes by $\hat{s}$ the computed sum of the vector y_i, $1\leq i\leq n$ and

$$s=\sum_{i=1}^{n}y_i$$

the real sum, then Equation (33) implies

$$\frac{|\hat{s}-s|}{|s|}\leq\gamma_{n-1}\,\mathrm{cond}\left(\sum_{i=1}^{n}y_i\right)\qquad(37)$$

with γ_n defined by

$$\gamma_n:=\frac{nu}{1-nu}\ \text{for}\ n\in\mathbb{N}\qquad(38)$$

Because $\gamma_{n-1}\approx(n-1)u$, it is almost the rule of thumb with just a small factor $n-1$ before u.

The LAPACK Library. The LAPACK library (10) is a collection of subroutines in Fortran 77 designed to solve major problems in linear algebra: linear systems, least square systems, eigenvalues, and singular values problems.

One of the most important advantages of LAPACK is that it provides error bounds for all the computed quantities. These error bounds are not rigorous but are mostly reliable. To do this, LAPACK uses the principles of backward analysis. In general, LAPACK provides both componentwise and normwise relative error bounds using the rule of thumb established in Equation (35).

In fact, the major part of the algorithms implemented in LAPACK are backward stable, which means that the rule of thumb [Equation (36)] is satisfied. As the condition number is generally very hard to compute, LAPACK uses estimators. It may happen that the estimator is far from the right condition number. In fact, the estimation can arbitrarily be far from the true condition number. The error bounds in LAPACK are only qualitative markers of the accuracy of the computed results.

Linear algebra problems are central in current scientific computing. Getting some good error bounds is therefore very important and is still a challenge.

Interval Arithmetic

Interval arithmetic (11, 12) is not defined on real numbers but on closed bounded intervals. The result of an arithmetic operation between two intervals, $X=[\underline{x},\overline{x}]$ and $Y=[\underline{y},\overline{y}]$, contains all values that can be obtained by performing this operation on elements from each interval. The arithmetic operations are defined below.

$$X+Y = [\underline{x}+\underline{y},\ \overline{x}+\overline{y}]\qquad(39)$$

$$X-Y = [\underline{x}-\overline{y},\ \overline{x}-\underline{y}]\qquad(40)$$

$$\begin{aligned}X\times Y = {}&[\min(\underline{x}\times\underline{y},\ \underline{x}\times\overline{y},\ \overline{x}\times\underline{y},\ \overline{x}\times\overline{y})\\ &\max(\underline{x}\times\underline{y},\ \underline{x}\times\overline{y},\ \overline{x}\times\underline{y},\ \overline{x}\times\overline{y})]\end{aligned}\qquad(41)$$

$$X^2 = \begin{cases}[\min(\underline{x}^2,\ \overline{x}^2),\max(\underline{x}^2,\ \overline{x}^2)]\ \text{if}\ 0\notin[\underline{x},\ \overline{x}]\\ [0,\max(\underline{x}^2,\ \overline{x}^2)]\ \text{otherwise}\end{cases}\qquad(42)$$

$$1/Y = [\min(1/\underline{y},1/\overline{y}),\max(1/\underline{y},1/\overline{y})]\ \text{if}\ 0\notin[\underline{y},\ \overline{y}]\qquad(43)$$

$$X/Y = [\underline{x},\ \overline{x}]\times(1/[\underline{y},\ \overline{y}])\ \text{if}\ 0\notin[\underline{y},\ \overline{y}]\qquad(44)$$

Arithmetic operations can also be applied to interval vectors and interval matrices by performing scalar interval operations componentwise.

Table 4. Determinant of the Hilbert Matrix H of Dimension 8

	det(H)	#exact digits
IEEE double precision	2.73705030017821E-33	7.17
interval Gaussian elimination	[2.717163073713011E-33, 2.756937028322111E-33]	1.84
interval specific algorithm	[2.737038183754026E-33, 2.737061910503125E-33]	5.06

An interval extension of a function f must provide all values that can be obtained by applying the function to any element of the interval argument X:

$$\forall x \in X,\ f(x) \in f(X) \tag{45}$$

For instance, $\exp[\underline{x}, \overline{x}] = [\exp \underline{x}, \exp \overline{x}]$ and $\sin[\pi/6, 2\pi/3] = [1/2, 1]$.

The interval obtained may depend on the formula chosen for mathematically equivalent expressions. For instance, let $f_1(x) = x^2 - x + 1$. $f_1([-2, 1]) = [-2, 7]$. Let $f_2(x) = (x - 1/2)^2 + 3/4$. The function f_2 is mathematically equivalent to f_1, but $f_2([-2, 1]) = [3/4\,, 7] \neq f_1([-2, 1])$. One can notice that $f_2([-2, 1]) \subseteq f_1([-2, 1])$. Indeed a power set evaluation is always contained in the intervals that result from other mathematically equivalent formulas.

Interval arithmetic enables one to control rounding errors automatically. On a computer, a real value that is not machine representable can be approximated to a floating-point number. It can also be enclosed by two floating-point numbers. Real numbers can therefore be replaced by intervals with machine-representable bounds. An interval operation can be performed using directed rounding modes, in such a way that the rounding error is taken into account and the exact result is necessarily contained in the computed interval. For instance, the computed results, with guaranteed bounds, of the addition and the subtraction between two intervals $X = [\underline{x}, \overline{x}]$ and $Y = [\underline{y}, \overline{y}]$ are

$$X + Y \;=\; [\nabla(\underline{x} + \underline{y}), \Delta(\overline{x} + \overline{y})] \supseteq \{x + y | x \in X, y \in Y\} \tag{46}$$

$$X - Y \;=\; [\nabla(\underline{x} - \overline{y}), \Delta(\overline{x} - \underline{y})] \supseteq \{x - y | x \in X, y \in Y\} \tag{47}$$

where ∇ (respectively Δ) denotes the downward (respectively upward) rounding mode.

Interval arithmetic has been implemented in several libraries or softwares. For instance, a C++ class library, C-XSC,[2] and a Matlab toolbox, INTLAB,[3] are freely available.

The main advantage of interval arithmetic is its reliability. But the intervals obtained may be too large. The intervals width regularly increases with respect to the intervals that would have been obtained in exact arithmetic. With interval arithmetic, rounding error compensation is not taken into account.

The overestimation of the error can be caused by the loss of variable dependency. In interval arithmetic, several occurrences of the same variable are considered as different variables. For instance, let $X = [1,2]$,

$$\forall x \in X,\ x - x = 0 \tag{48}$$

but

$$X - X = [-1, 1] \tag{49}$$

Another source of overestimation is the "wrapping effect" because of the enclosure of a noninterval shape range into an interval. For instance, the image of the square $[0, \sqrt{2}] \times [0, \sqrt{2}]$ by the function

$$f(x,y) = \frac{\sqrt{2}}{2}(x + y, y - x) \tag{50}$$

is the rotated square S_1 with corners (0, 0), (1, −1), (2, 0), (1, 1). The square S_2 provided by interval arithmetic operations is: $f([0, \sqrt{2}], [0, \sqrt{2}]) = ([0, 2], [-1, 1])$. The area obtained with interval arithmetic is twice the one of the rotated square S_1.

As the classic numerical algorithms can lead to overpessimistic results in interval arithmetic, specific algorithms, suited for interval arithmetic, have been proposed. Table 4 presents the results obtained for the determinant of Hilbert matrix H of dimension 8 defined by

$$H_{ij} = \frac{1}{i + j - 1} \quad \text{for} \quad i = 1, \ldots, 8 \quad \text{and} \quad j = 1, \ldots 8 \tag{51}$$

computed:

- using the Gaussian elimination in IEEE double precision arithmetic with rounding to the nearest
- using the Gaussian elimination in interval arithmetic
- using a specific interval algorithm for the inclusion of the determinant of a matrix, which is described in Ref. 8, p. 214.

Results obtained in interval arithmetic have been computed using the INTLAB toolbox.

The exact value of the determinant is

$$\mathrm{det(H)} = \prod_{\mathrm{k}=0}^{7} \frac{(\mathrm{k}!)^3}{(8 + \mathrm{k})!} \tag{52}$$

Its 15 first exact significant digits are:

$$\mathrm{det(H)} = 2.73705011379151\mathrm{E} - 33 \tag{53}$$

The number of exact significant decimal digits of each computed result has been reported in Table 4.

One can verify the main feature of interval arithmetic: The exact value of the determinant is enclosed in the computed intervals. Table 4 points out the overestimation of the

[2] http://www.xsc.de.

[3] http://www.ti3.tu-harburg.de/rump/intlab.

error with naive implementations of classic numerical algorithms in interval arithmetic. The algorithm for the inclusion of a determinant that is specific to interval arithmetic leads to a much thinner interval. Such interval algorithms exist in most areas of numerical analysis. Interval analysis can be used not only for reliable numerical simulations but also for computer assisted proofs (cf., for example, Ref. 8).

Probabilistic Approach

Here, a method for estimating rounding errors is presented without taking into account the model errors or the discretization errors.

Let us go back to the question *"What is the computing error due to floating-point arithmetic on the results produced by a program?"* From the physical point of view, in large numerical simulations, the final rounding error is the result of billions and billions of elementary rounding errors. In the general case, it is impossible to describe each elementary error carefully and, then to compute the right value of the final rounding error. It is usual, in physics, when a deterministic approach is not possible, to apply a probabilistic model. Of course, one loses the exact description of the phenomena, but one may hope to get some global information like order of magnitude, frequency, and so on. It is exactly what is hoped for when using a probabilistic model of rounding errors.

For the mathematical model, remember the formula at the first order [Equation (13)]. Concretely, the rounding mode of the computer is replaced by a random rounding mode (i.e., at each elementary operation, the result is rounded toward $-\infty$ or $+\infty$ with the probability 0.5.) The main interest of this new rounding mode is to run a same binary code with different rounding error propagations because one generates for different runs different random draws. If rounding errors affect the result, even slightly, then one obtains for N different runs, N different results on which a statistical test may be applied. This strategy is the basic idea of the CESTAC method (Contrôle et Estimation STochastique des Arrondis de Calcul). Briefly, the part of the N mantissas that is common to the N results is assumed to be not affected by rounding errors, contrary to the part of the N mantissas that is different from one result to another.

The implementation of the CESTAC method in a code providing a result R consists in:

- executing N times this code with the random rounding mode, which is obtained by using randomly the rounding mode toward $-\infty$ or $+\infty$; then, an N-sample (R_i) of R is obtained,
- choosing as the computed result the mean value $\overline{R}$ of R_i, $i = 1, \ldots, N$,
- estimating the number of exact decimal significant digits of $\overline{R}$ with

$$C_{\overline{R}} = \log_{10}\left(\frac{\sqrt{N}|\overline{R}|}{\sigma\tau_\beta}\right) \tag{54}$$

where

$$\overline{R} = \frac{1}{N}\sum_{i=1}^{N} R_i \quad \text{and} \quad \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(R_i - \overline{R})^2 \tag{55}$$

τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a probability level $1-\beta$.

From Equation (13), if the first order approximation is valid, one may deduce that:

1. The mean value of the random variable R is the exact result r,
2. Under some assumptions, the distribution of R is a quasi-Gaussian distribution.

It has been shown that $N = 3$ is the optimal value. The estimation with $N = 3$ is more reliable than with $N = 2$ and increasing the size of the sample does not improve the quality of the estimation. The complete theory can be found in Refs. 13 and 14. The approximation at the first order in Equation (13) is essential for the validation of the CESTAC method. It has been shown that this approximation may be wrong only if multiplications or divisions involve nonsignificant values. A nonsignificant value is a computed result for which all the significant digits are affected by rounding errors. Therefore, one needs a dynamical control of multiplication and division during the execution of the code. This step leads to the synchronous implementation of the method (i.e., to the parallel computation of the N results R_i.) In this approach, a classic floating-point number is replaced by a 3-sample $X = (X_1, X_2, X_3)$, and an elementary operation $\Omega \in \{+, -, \times, /\}$ is defined by $X\Omega Y = (X_1\omega Y_1, X_2\omega Y_2, X_3\omega Y_3)$, where ω represents the corresponding floating-point operation followed by a random rounding. A new important concept has also been introduced: the computational zero.

Definition 3.1. During the run of a code using the CESTAC method, an intermediate or a final result R is a computational zero, denoted by @.0, if one of the two following conditions holds:

- $\forall i$, $R_i = 0$,
- $C_{\overline{R}} \leq 0$.

Any computed result R is a computational zero if either $R = 0$, R being significant, or R is nonsignificant. In other words, a computational zero is a value that cannot be differentiated from the mathematical zero because of its rounding error. From this new concept of zero, one can deduce new order relationships that take into account the accuracy of intermediate results. For instance,

Definition 3.2. X is stochastically strictly greater than Y if and only if:

$$\overline{X} > \overline{Y} \quad \text{and} \quad X - Y \neq @.0$$

or

Definition 3.3. X is stochastically greater than or equal to Y if and only if:

$$\overline{X} \geq \overline{Y} \quad \text{or} \quad X - Y = @.0$$

The joint use of the CESTAC method and these new definitions is called Discrete Stochastic Arithmetic (DSA). DSA enables to estimate the impact of rounding errors on any result of a scientific code and also to check that no anomaly occurred during the run, especially in branching statements. DSA is implemented in the Control of Accuracy and Debugging for Numerical Applications (CADNA) library.[4] The CADNA library allows, during the execution of any code:

- the estimation of the error caused by rounding error propagation,
- the detection of numerical instabilities,
- the checking of the sequencing of the program (tests and branchings),
- the estimation of the accuracy of all intermediate computations.

METHODS FOR ACCURATE COMPUTATIONS

In this section, different methods to increase the accuracy of the computed result of an algorithm are presented. Far from being exhaustive, two classes of methods will be presented. The first class is the class of compensated methods. These methods consist in estimating the rounding error and then adding it to the computed result. The second class are algorithms that use multiprecision arithmetic.

Compensated Methods

Throughout this subsection, one assumes that the floating-point arithmetic adhers to IEEE 754 floating-point standard in rounding to the nearest. One also assume that no overflow nor underflow occurs. The material presented in this section heavily relies on Ref. (15).

Error-Free Transformations (EFT). One can notice that $a \circ b \in \mathbb{R}$ and $a \circledcirc b \in \mathbb{F}$, but in general $a \circ b \in \mathbb{F}$ does not hold. It is known that for the basic operations $+, -, \times, \sqrt{}$ the approximation error of a floating-point operation is still a floating-point number:

$$\begin{array}{lcll} x = a \oplus b & \Rightarrow & a + b = x + y & \text{with} \, y \in \mathbb{F}, \\ x = a \ominus b & \Rightarrow & a - b = x + y & \text{with} \, y \in \mathbb{F}, \\ x = a \otimes b & \Rightarrow & a \times b = x + y & \text{with} \, y \in \mathbb{F}, \\ x = a \oslash b & \Rightarrow & a = x \times b + y & \text{with} \, y \in \mathbb{F}, \\ x = \textcircled{\checkmark}\ (a) & \Rightarrow & a = x^2 + y & \text{with} \, y \in \mathbb{F} \end{array} \tag{56}$$

These example are *error-free* transformations of the pair (a, b) into the pair (x, y). The floating-point number x is the result of the floating-point operation and y is the rounding term. Fortunately, the quantities x and y in Equation (56) can be computed exactly in floating-point arithmetic. For the algorithms, Matlab-like notations are used. For addition, one can use the following algorithm by Knuth.

Algorithm 4.1. (16). Error-free transformation of the sum of two floating-point numbers

```
function [$x$, $y$] = TwoSum($a$, $b$)
  $x = a \oplus b$
  $z = x \ominus a$
  $y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$
```

Another algorithm to compute an error-free transformation is the following algorithm from Dekker (17). The drawback of this algorithm is that $x + y = a + b$ provided that $|a| \geq |b|$. Generally, on modern computers, a comparison followed by a branching and three operations costs more than six operations. As a consequence, TwoSum is generally more efficient than FastTwoSum plus a branching.

Algorithm 4.2. (17). Error-free transformation of the sum of two floating-point numbers.

```
function [$x$, $y$] = FastTwoSum($a$, $b$)
  $x = a \oplus b$
  $y = (a \ominus x) \oplus b$
```

For the error-free transformation of a product, one first needs to split the input argument into two parts. Let p be given by $u = 2^{-p}$, and let us define $s = \lceil p/2 \rceil$. For example, if the working precision is IEEE 754 double precision, then $p = 53$ and $s = 27$. The following algorithm by Dekker (17) splits a floating-point number $a \in \mathbb{F}$ into two parts x and y such that

$$a = x + y \quad \text{with} \, |y| \leq |x| \tag{57}$$

Both parts x and y have at most $s - 1$ non-zero bits.

Algorithm 4.3. (17) Error-free split of a floating-point number into two parts

```
function [$x$, $y$] = Split($a$)
  factor $= 2^s \oplus 1$
  $c =$ factor $\otimes a$
  $x = c \ominus (c \ominus a)$
  $y = a \ominus x$
```

The main point of Split is that both parts can be multiplied in the same precision without error. With this function, an algorithm attributed to Veltkamp by Dekker enables to compute an error-free transformation for the product of two floating-point numbers. This algorithm returns two floating-point numbers x and y such that

$$a \times b = x + y \quad \text{with} \ x = a \otimes b \tag{58}$$

Algorithm 4.4. (17). Error-free transformation of the product of two floating-point numbers

```
function [$x$, $y$] = TwoProduct($a$, $b$)
  $x = a \otimes b$
  $[a_1, a_2] =$ Split($a$)
  $[b_1, b_2] =$ Split($b$)
  $y = a_2 \otimes b_2 \ominus (((x \ominus a_1 \otimes b_1) \ominus a2 \otimes b_1) \ominus a_1 \otimes b_2)$
```

The performance of the algorithms is interpreted in terms of floating-point operations (flops). The following

[4] http://www.lip6.fr/cadna/.

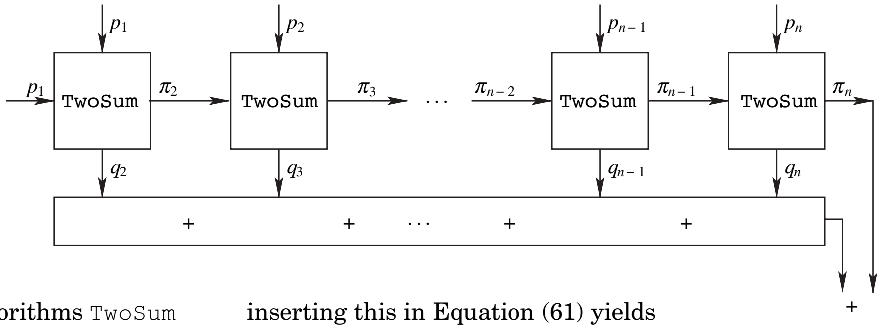


Figure 3. Compensated summation algorithm.

theorem summarizes the properties of algorithms `TwoSum` and `TwoProduct`.

Theorem 4.1. Let $a, b \in \mathbb{F}$ and let $x, y \in \mathbb{F}$ such that $[x, y] =$ `TwoSum`(a, b) (Algorithm 4.1). Then,

$$a + b = x + y, \quad x = a \oplus b, \quad |y| \le u|x|, \quad |y| \le u|a + b|. \tag{59}$$

The algorithm `TwoSum` requires 6 flops.

Let $a, b \in \mathbb{F}$ and let $x, y \in \mathbb{F}$ such that $[x, y] =$ `TwoProduct`(a, b) (Algorithm 4.4). Then,

$$a \times b = x + y, \quad x = a \otimes b, \quad |y| \le u|x|, \quad |y| \le u|a \times b|\,. \tag{60}$$

The algorithm `TwoProduct` requires 17 flops.

A Compensated Summation Algorithm. Hereafter, a compensated scheme to evaluate the sum of floating-point numbers is presented, (i.e., the error of individual summation is somehow corrected).

Indeed, with Algorithm 4.1 (`TwoSum`), one can compute the rounding error. This algorithm can be cascaded and sum up the errors to the ordinary computed summation. For a summary, see Fig. 3 and Algorithm 4.5.

Algorithm 4.5. Compensated summation algorithm

```
function res = CompSum($p$)
  $\pi_1 = p_1$; $\sigma_1 = 0$;
for $i = 2 : n$
  $[\pi_i, q_i]$ = TwoSum($\pi_{i-1}, p_i$)
  $\sigma_i = \sigma_{i-1} \oplus qi$
res = $\pi_n \oplus \sigma_n$
```

The following proposition gives a bound on the accuracy of the result. The notation γ_n defined by Equation (38) will be used. When using $\gamma_n, nu \le 1$ is implicitly assumed.

Proposition 4.2. (15). Suppose Algorithm `CompSum` is applied to floating-point number $p_i \in \mathbb{F}, 1 \le i \le n$. Let $s := \sum p_i, S := \sum |p_i|$ and nu < 1. Then, one has

$$|\mathtt{res} - s| \le u|s| + \gamma_{n-1}^2 S \tag{61}$$

In fact, the assertions of Proposition 4.2 are also valid in the presence of underflow.

One can interpret Equation (61) in terms of the condition number for the summation (29). Because

$$\mathrm{cond}\Big(\sum p_i\Big) = \frac{\sum |p_i|}{|\sum p_i|} = \frac{S}{|s|} \tag{62}$$

inserting this in Equation (61) yields

$$\frac{|\mathtt{res} - s|}{|s|} \le u + \gamma_{n-1}^2 \mathrm{cond}\Big(\sum p_i\Big) \tag{63}$$

Basically, the bound for the relative error of the result is essentially $(nu)^2$ times the condition number plus the rounding u because of the working precision. The second term on the right-hand side reflects the computation in twice the working precision (u^2) thanks to the *rule of thumb*. The first term reflects the rounding back in the working precision.

The compensated summation on ill-conditioned sum was tested; the condition number varied from 10^4 to 10^{40}.

Figure 4 shows the relative accuracy $|\mathtt{res} - s|/|s|$ of the computed value by the two algorithms 3.1 and 4.5. The *a priori* error estimations Equations (37) and (63) are also plotted.

As one can see in Fig. 4, the compensated summation algorithm exhibits the expected behavior, that is to say, the compensated rule of thumb Equation (63). As long as the condition number is less than u^{-1}, the compensated summation algorithm produces results with full precision (forward relative error of the order of u). For condition numbers greater than u^{-1}, the accuracy decreases and there is no accuracy at all for condition numbers greater than u^{-2}.

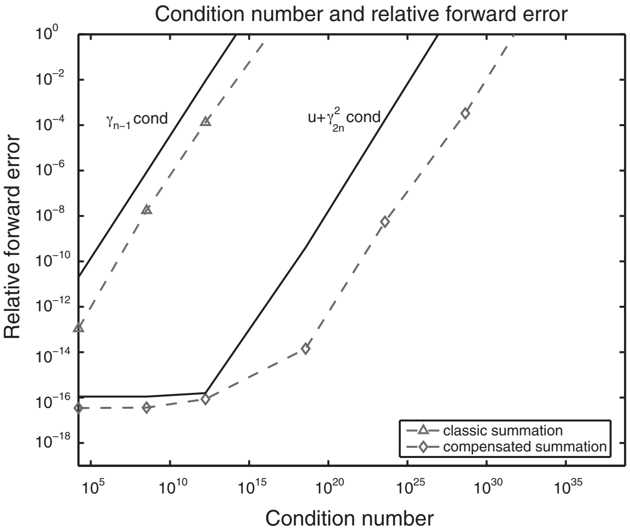


Figure 4. Compensated summation algorithm.

Multiple Precision Arithmetic

Compensated methods are a possible way to improve accuracy. Another possibility is to increase the working precision. For this purpose, some multiprecision libraries have been developed. One can divide the libraries into three categories.

- Arbitrary precision libraries using a *multiple-digit* format in which a number is expressed as a sequence of digits coupled with a single exponent. Examples of this format are Bailey's MPFUN/ARPREC,[5] Brent's MP,[6] or MPFR.[7]
- Arbitrary precision libraries using a *multiple-component* format where a number is expressed as unevaluated sums of ordinary floating-point words. Examples using this format are Priest's[8] and Shewchuk's[9] libraries. Such a format is also introduced in Ref. 18.
- Extended fixed-precision libraries using the *multiple-component* format but with a limited number of components. Examples of this format are Bailey's *double-double*[5] (double-double numbers are represented as an unevaluated sum of a leading double and a trailing double) and *quad-double*.[5]

The double-double library will be now presented. For our purpose, it suffices to know that a double-double number a is the pair (a_h, a_l) of IEEE-754 floating-point numbers with $a = a_h + a_l$ and $|a_l| \leq u\,|a_h|$. In the sequel, algorithms for

- the addition of a double number to a double-double number;
- the product of a double-double number by a double number;
- the addition of a double-double number to a double-double number

will only be presented. Of course, it is also possible to implement the product of a double-double by a double-double as well as the division of a double-double by a double, and so on.

Algorithm 4.6. Addition of the double number b to the double-double number (a_h, a_l)

```
function $[c_h, c_l]$ = add_dd_d($a_h, a_l, b$)
    $[t_h, t_l]$ = TwoSum($a_h, b$)
    $[c_h, c_l]$ = FastTwoSum($t_h, (t_l \oplus a_l)$)
```

Algorithm 4.7. Product of the double-double number (a_h,a_l) by the double number b

```
function $[c_h, c_l]$ = prod_dd_d($a_h, a_l, b$)
    $[s_h, s_l]$ = TwoProduct($a_h, b$)
    $[t_h, t_l]$ = FastTwoSum($s_h, (a_l \otimes b)$)
    $[c_h, c_l]$ = FastTwoSum($t_h, (t_l \oplus s_l)$)
```

Algorithm 4.8. Addition of the double-double number (a_h, a_l) to the double-double number (b_h, b_l)

```
function $[c_h, c_l]$ = add_dd_dd ($a_h, a_l, b_h, b_l$)
    $[s_h, s_l]$ = TwoSum($a_h, b_h$)
    $[t_h, t_l]$ = TwoSum($a_l, b_l$)
    $[t_h, s_l]$ = FastTwoSum($s_h, (s_l \oplus t_h)$)
    $[c_h, c_l]$ = FastTwoSum($t_h, (t_l \oplus s_l)$)
```

Algorithms 4.6 to 4.8 use error-free transformations and are very similar to compensated algorithms. The difference lies in the step of renormalization. This step is the last one in each algorithm and makes it possible to ensure that $|c_l| \leq u|c_h|$.

Several implementations can be used for the double-double library. The difference is that the lower-order terms are treated in a different way. If a, b are double-double numbers and $\circledcirc \in \{+, \times\}$, then one can show (19) that

$$\mathrm{fl}(a \circledcirc b) = (1+\delta)(a \circledcirc b)$$

with $|\delta| \leq 4 \cdot 2^{-106}$.

One might also note that when keeping $[\pi_n, \sigma_n]$ as a pair the first summand u disappears in [Equation (63)] (see Ref. 15), so it is an example for a double-double result.

Let us now briefly describe the MPFR library. This library is written in C language based on the GNU MP library (GMP for short). The internal representation of a floating-point number x by MPFR is

- a mantissa m;
- a sign s;
- a signed exponent e.

If the precision of x is p, then the mantissa m has p significant bits. The mantissa m is represented by an array of GMP unsigned machine-integer type and is interpreted as $1/2 \leq m < 1$. As a consequence, MPFR does not allow denormalized numbers.

MPFR provides the four IEEE rounding modes as well as some elementary functions (e.g., exp, log, cos, sin), all correctly rounded. The semantic in MPFR is as follows: For each instruction $a = b + c$ or $a = f(b, c)$ the variables may have different precisions. In MPFR, the data b and c are considered with their full precision and a correct rounding to the full precision of a is computed.

Unlike compensated methods that need to modify the algorithms, multiprecision libraries are convenient ways to increase the precision without too many efforts.

ACKNOWLEDGMENT

The authors sincerely wish to thank the reviewers for their careful reading and their constructive comments.

BIBLIOGRAPHY

1. report of the General Accounting office, GAO/IMTEC-92-26.
2. D. Goldberg, What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surve.*, **23**(1): 5–48, 1991.

[5] http://crd.lbl.gov/~dhbailey/mpdist/.

[6] http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub043.html.

[7] http://www.mpfr.org/.

[8] ftp://ftp.icsi.berkeley.edu/pub/theory/priest-thesis.ps.Z.

[9] http://www.cs.cmu.edu/~quake/robust.html.

3. IEEE Computer Society, *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*, 1985. Reprinted in SIGPLAN Notices, **22**(2): 9–25, 1987.
4. S. M. Rump, How reliable are results of computers? *Jahrbuch Überblicke Mathematik*, pp. 163–168, 1983.
5. N. J. Higham, *Accuracy and stability of numerical algorithms*, Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2nd ed. 2002.
6. P. Langlois, Analyse d'erreur en precision finie. In A. Barraud (ed.), *Outils d'Analyse Numérique pour l'Automatique*, Traité IC2, Cachan, France: Hermes Science, 2002, pp. 19–52.
7. F. Chaitin-Chatelin and V. Frayssé, *Lectures on Finite Precision Computations*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1996.
8. S. M. Rump, Computer-assisted proofs and self-validating methods. In B. Einarsson (ed.), *Accuracy and Reliability in Scientific Computing*, Software-Environments-Tools, Philadelphia, PA: SIAM, 2005, pp. 195–240.
9. J. H. Wilkinson, Rounding errors in algebraic processes. (32), 1963. Also published by Englewood Cliffs, NJ: Prentice-Hall, and New York: Dover, 1994.
10. E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.
11. G. Alefeld and J. Herzberger, *Introduction to Interval Analysis*. New York: Academic Press, 1983.
12. U. W. Kulisch, *Advanced Arithmetic for the Digital Computer*. Wien: Springer-Verlag, 2002.
13. J.-M. Chesneaux. *L'Arithmétique Stochastique et le Logiciel CADNA*. Paris: Habilitation à diriger des recherches, Université Pierre et Marie Curie, 1995.
14. J. Vignes, A stochastic arithmetic for reliable scientific computation. *Math. Comput. Simulation*, **35**: 233–261, 1993.
15. T. Ogita, S. M. Rump, and S. Oishi, Accurate sum and dot product. *SIAM J. Sci. Comput.*, **26**(6): 1955–1988, 2005.
16. D. E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 3rd ed. Reading, MA: Addison-Wesley, 1998.
17. T. J. Dekker, A floating-point technique for extending the available precision. *Numer. Math.*, **18**: 224–242, 1971.
18. S. M. Rump, T. Ogita, and S. Oishi, Accurate Floating-point Summation II: Sign, K-fold Faithful and Rounding to Nearest. Technical Report 07.2, Faculty for Information and Communication Sciences, Hamburg, Germany: Hamburg University of Technology, 2007.
19. X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, B. J. Thompson, T. Tung, and D. J. Yoo, Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Softw.*, **28**(2): 152–205, 2002.

JEAN-MARIE CHESNEAUX
STEF GRAILLAT
FABIENNE JÉZÉQUEL
Laboratoire d'Informatique de
Paris, France

ADDRESSING: DIRECT AND INDIRECT

This article focuses on addressing modes—a mechanism machine instructions use to specify operands. Addressing modes are an important aspect of instruction set architecture (ISA). ISA refers to a portion of the computer that is visible to a compiler writer or an assembly language programmer, and it encompasses components such as (*1*) class of ISA, (*2*) memory, (*3*) addressing modes, (*4*) types and sizes of operands, (*5*) data processing and control flow operations supported by machine instructions, and (*6*) instruction encoding. As it is difficult to separate addressing modes from other components of instruction set architecture, we will first give a brief overview of basic ISA components. For a detailed treatment of the remaining ISA components, readers are directed to the article dedicated to instruction set architectures.

Class of ISA. Virtually all recent instruction set architectures have a set of general-purpose registers visible to programmers. These architectures are known as *general-purpose register architectures*. Machine instructions in these architectures specify all operands in memory or general-purpose registers explicitly. In older architectures, machine instructions specified one or more operands implicitly on the stack—so-called *stack architectures*—or in the accumulator—so-called *accumulator architectures*. There are many reasons why general-purpose register architectures dominate in today's computers. Allocating frequently used variables, pointers, and intermediate results of calculations in *registers* reduces memory traffic; improves processor performance since registers are much faster than memory; and reduces code size since naming registers requires fewer bits than naming memory locations directly. A general trend in recent architectures is to increase the number of general-purpose registers.

General-purpose register architectures can be classified into *register-memory* and *load-store architectures,* depending on the location of operands used in typical arithmetic and logical instructions. In register-memory architectures, arithmetic and logical machine instructions can have one or more operands in memory. In load-store architectures, only load and store instructions can access memory, and common arithmetic and logical instructions are performed on operands in registers. Depending on the number of operands that can be specified by an instruction, ISAs can be classified into *2-operand* or *3-operand architectures*. With 2-operand architectures, typical arithmetic and logical instructions specify one operand that is both a source and a destination for the operation result, and another operand is a source. For example, the arithmetic instruction *ADD R1, R2* adds the operands from the registers *R1* and *R2* and writes the result back to the register *R2*. With 3-operand architectures, instructions can specify two source operands and the result operand. For example, the arithmetic instruction *ADD R1, R2, R3* adds the operands from the registers *R1* and *R2* and writes the result to the register *R3*. In this text, we consider general-purpose register architectures that can be either register-memory or load-store.

Memory. Program instructions and data are stored in memory during program execution. Programmers see memory as a linear array of addressable locations as shown in Fig. 1. In nearly all memory systems, the smallest addressable location in memory is a single byte (8 bits). The range of memory that can be addressed by the processor is called an *address space*. For example, any program running on a 32-bit processor can address up to 4 GB (2^{32} bytes) of the address space. Although the smallest addressable object is a byte, bytes are generally grouped into multibyte objects. For example, in a 32-bit architecture, we refer to 2-byte objects as half-words, 4-byte objects as words, and 8-byte objects as double words. Machine instructions can directly reference and operate on words, half-words, or bytes. When referencing a multibyte object in memory, its given address is the address of its first byte. For example, a half-word located in memory at the address 8 will occupy two byte addresses 8 and 9.

Many instruction set architectures require multibyte objects to be *aligned* to their natural boundaries. For example, if we assume a 4-byte wide memory (Fig. 1), half-words must begin at even addresses, whereas words and double words must begin at addresses divisible by 4. This kind of alignment requirement is often referred to as *hard alignment*. It should be noted that hard alignment is not an architectural requirement; rather it makes hardware implementation more practical. Even architectures that do not require hard alignment may benefit from having multibyte objects aligned. Access to unaligned objects may require multiple accesses to memory, resulting in performance penalty. Another important issue related to memory is ordering the bytes within a larger object. There are two different conventions for byte ordering: *little-endian* and *big-endian* (Fig. 2). With little-endian byte ordering, the least significant byte in a word is located at the lowest byte address, and with big-endian, the most significant byte in a word is located at the lowest byte address. For example, let us consider a 32-bit integer variable with a hexadecimal value of 0x1234ABCD stored in memory at word address 8. For both big-endian and little-endian byte ordering, the most significant byte of the variable is 0x12 and the least significant byte is 0xCD. However, with the big-endian scheme, the most significant byte is at address 8, whereas with the little-endian scheme, the most significant byte is at address 11 (Fig. 2).

Types and Sizes of Operands. Machine instructions operate on operands of certain types. Common types supported by ISAs include character (e.g., 8-bit ASCII or 16-bit Unicode), signed and unsigned integers, and single- and double-precision floating-point numbers. ISAs typically support several sizes for integer numbers. For example,

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

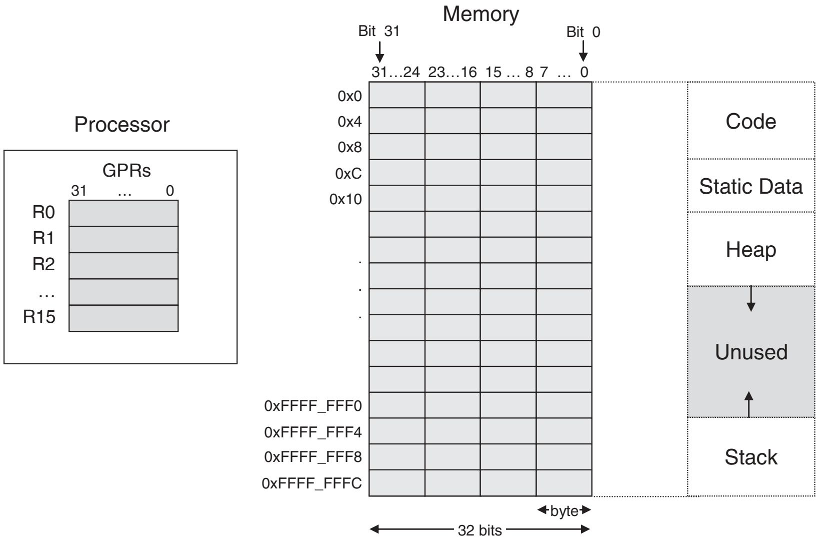


Figure 1. Programming model: general-purpose registers (GPRs) and memory. (This figure is available in full color at http://www.interscience.wiley.com/reference/ecse.)

a 32-bit architecture may include arithmetic instructions that operate on 8-bit integers, 16-bit integers (short integers), and 32-bit integers. Signed integers are represented using two's complement binary representation, whereas floating-point numbers rely on IEEE Standard 754. Some ISAs support less frequently used data types, such as character strings and packed decimal or binary-coded decimal numbers (a decimal digit requires 4 bits, and two decimal digits are packed into a byte).

Instructions. Machine instructions can be broadly classified into data processing and control-flow instructions. Data processing instructions manipulate operands in registers and memory locations. Common data processing instructions support integer arithmetic operations (e.g., add, subtract, compare, multiply, and divide), logic operations (e.g., bitwise and, or, xor, nor, and not); shift operations (e.g., shift to the right or left, and rotate), and data transfer operations *(load* that moves a specified operand from memory to a register, *store* that moves a specified operand from register to a memory location, and *move* that transfers data between registers). If a computer is intended for applications that extensively use floating-point numbers, the ISA may support floating-point arithmetic (e.g., floating-point add, subtract, compare, multiply, and divide). Several older ISAs support instructions that manipulate decimal operands and character string operands. In media and signal processing architectures, we may encounter instructions that operate on more complex data types (e.g., pixels).

Machine instructions are fetched from memory and executed sequentially. Control-flow or branch instructions allow us to make decisions and to change the execution flow to an instruction other than the next one in sequence. These instructions can be classified into conditional (often referred to as branches) and unconditional (often referred to as jumps), procedure calls, and procedure returns. A conditional branch instruction is defined by its outcome that determines whether the branch is taken or not taken; and by its target address that specifies the address of the following instruction in sequence to be executed, if the branch is taken. A jump instruction is defined by its target address only. Branch target addresses can be known at compile time (direct branches) or be determined during program execution (indirect branches).

Binary Encoding of Instructions. Instruction encoding defines the binary representation of machine instructions.

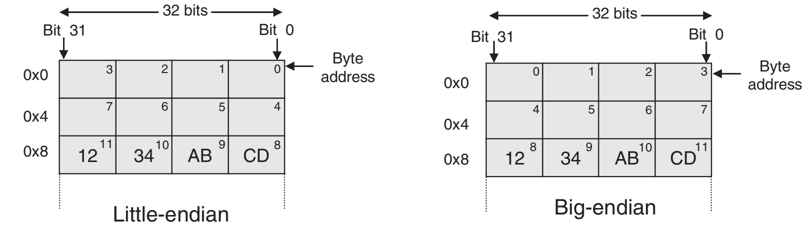


Figure 2. Little-endian and big-endian byte ordering. (This figure is available in full color at http://www. interscience.wiley.com/reference/ecse.)

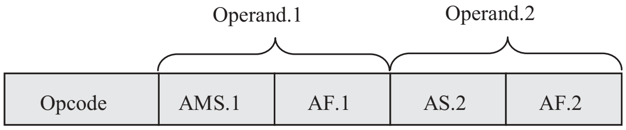


Figure 3. A generalized 2-operand instruction format (AMS—Address Mode Specifier, AF—Address Field). (This figure is available in full color at http://www. interscience.wiley.com/reference/ecse.)

Exact encoding depends on many parameters, such as architecture type, the number of operands, the number and type of instructions, the number of general-purpose registers, operand types, and the size of address space. This representation affects not only the size of the program, but also the processor implementation. The operation and possibly the number of operands are typically specified by one instruction field called the *opcode*. For each operand, the machine instruction includes an *addressing mode specifier*—a field that tells what addressing mode is used to access the operand, and one or more *address fields* that specify the operand address. Figure 3 shows a generalized instruction format for a 2-operand instruction. This approach to instruction encoding is often referred to as *variable length* — each operation can work with virtually all addressing modes that are supported by the ISA. An alternative approach *is fixed-length* instruction encoding where the *opcode* is combined with addressing mode specifiers. Typically a single size is used for all instructions and this approach is used when there are a few addressing modes and operations. A third approach called *hybrid* is somewhere in between. It reduces variability in instruction encoding but allows multiple instruction lengths. In load/store architectures, all instructions except loads and stores find their operands in general-purpose registers; hence, the addressing mode specifiers are not needed. Here we will assume that information about the number of operands and the corresponding addressing mode specifiers are all merged with the opcode field. Fixed-length instruction formats require less complex decoding logic, resulting in faster decoding, but they tend to increase the number of bits needed to encode an instruction, resulting in poor code density. Code density is an important characteristic of an instruction set, and it can be measured by the size of a program needed to complete a particular task.

Addressing Modes. A machine instruction can find its operand in one of three places: (*1*) as a part of the instruction, (*2*) in a general-purpose register, and (*3*) in memory. Operands in registers and memory can be specified directly or indirectly. Consequently, addressing modes can be broadly classified into (*1*) *direct*— the address field specifies the operand address—and (*2*) *indirect*—the address field specifies a location that contains the operand address. A wide variety of addressing modes is used in instruction set architectures, such as *immediate, register direct, register indirect, register indirect with displacement, memory direct,* and *memory indirect,* to name just a few. Table 1 gives a list of the most common addressing modes with examples and usage. Each addressing mode is illustrated by a LOAD instruction that moves the specified operand into a general-purpose register. Figure 4 gives a graphical illustration of these addressing modes.

BASIC ADDRESSING MODES

Three basic addressing modes are (*1*) *immediate or literal* where the operand is a part of the instruction; (*2*) *register direct* where the operand is in a general-purpose register; and (3) *memory direct* where the operand is in memory at a location specified directly by the instruction.

Immediate or *literal* addressing means that the operand is located in the instruction itself (Fig. 4a). Immediately upon instruction decoding, the operand is extracted from the instruction and possibly zero-or sign-extended before it is used in the instruction execution. This addressing mode specifies a data or an address constant, and it is often used in arithmetic operations, in initializing registers and memory locations, and in comparisons. The number of bits used to encode an immediate operand affects the instruction length. Data constants in programs tend to be small and can be encoded by a relatively small number of bits, improving code density. Unlike data constants, address constants tend to be large and require all address bits. An analysis of integer and floating-point programs from the SPEC benchmark suite shows that a 16-bit field captures over 80% of all immediate operands and an 8-bit field captures over 50% of all immediate operands (1). To exploit this characteristic of programs, many architectures support so-called short constants that require just a few bits to encode the most frequently used constants, such as 0, ±1, ±2, and ±4.

Register direct or just *register* addressing means that the operand is located in a general-purpose register and the instruction specifies the register by its name (Fig. 4b). For example, only a 4-bit address field is needed to specify a register in a processor with 16 general-purpose registers (0001 encodes R1 and 0011 encodes R3). This addressing mode is preferred for keeping frequently used operands because it is compact—only a register identifier is stored in the instruction, and access to a register is much faster than access to a memory location. In load/store architectures, this mode is used for all instructions except when an operand is moved from memory into a register or from a register to memory. Many compiler optimizations are employed to determine which variables are the most frequently accessed and to put them in general-purpose registers.

Memory direct addressing (often referred to as *absolute)* means that the operand is located in memory and the address field contains the actual address of the operand (Fig. 4c). This address is specified at the compile-time and cannot be modified by the processor during program execution. Hence, this addressing mode is used to access global variables whose addresses are known at compile time. The number of bits needed to encode a full memory address is directly proportional to the size of address space, and it tends to be large. Several architectures support so-called short absolute addressing—only a portion of the address space is directly accessible by this addressing mode (2). For example, a short direct address has only 16 bits and

Table 1. Data Addressing Modes, Example Instructions, Description, and Typical Use

Addressing Mode	Example Instruction	RTL Description	Typical Use
Immediate	`LOAD #3, R3`	`[R3] ← 0×00000003`	For constants
Register-direct	`LOAD R1, R3`	`[R3] ←[R1]`	When a value is in a register

Memory direct or Absolute	`LOAD $8000, R3`	`EA ← $00008000`
		`[R3] ←[Mem(EA)]`
			Access to static variables in memory

Register indirect	`LOAD (R1), R3`	`EA ←[R1]`
		`[R3] ←[Mem(EA)]`
			Access to variables in memory using a pointer

Register indirect with index	`LOAD (R1+R2), R3`	`EA ←[R1] + [R2]`
		`[R3] ←[Mem(EA)]`
			Access to elements in an array of complex data structures (R1 points to the base, R2 is stride)

Register indirect with scaled index	`LOAD (R1+R2*4),R3`	`EA ←[R1] + [R2] *4`
		`[R3] ←[Mem(EA)]`
			Access to elements in an array of complex data structures (R3 points to the base, R2 is index)

Autoincrement	`LOAD (R1)+, R3`	`EA ←[R1] ; [R1] ←[R1] + 4`
		`[R3] ←[Mem(EA)]`
			Access to elements of an array in a loop; Access to stack (push/pop)

Autodecrement	`LOAD −(R1), R3`	`[R1] ←[R1] − 4; EA ←[R1]`
		`[R3] ←[Mem(EA)]`
			Access to elements of an array in a loop; Access to stack (push/pop)

Register indirect with displacement	`LOAD 0×100(R1), R3`	`EA ←[R1] + 0×0100`
		`[R3] ←[Mem(EA)]`
			Access to local variables

Register indirect with scaled index and displacement	`LOAD 0×100(R1+R2*4), R3`	`EA ← 0×0100 +[R1] +[R2] *4`
		`[R3] ←[Mem(EA)]`
			Access to arrays allocated on the stack

PC relative	`LOAD 0×100(PC), R3`	`EA ← 0×0100 +[PC]`
		`[R3] ←[Mem(EA)]`
			Branches; Jumps; Procedure calls; Static data

PC relative with index	`LOAD (PC+R2),R3`	`EA ←[PC]+[R2]`
		`[R3] ←[Mem(EA)]`
			Branches; Jumps; Procedure calls; Static data

PC relative with scaled index and displacement	`LOAD 0×100(PC+R2*4), R3`	`EA ← 0×0100 +[PC]+[R2] *4`
		`[R3] ←[Mem(EA)]`
			Branches; Jumps; Procedure calls; Static data

Register transfer language (RTL) is used to describe data transfers and operations. Square brackets [ ] indicate content of registers and memory locations, and backward arrows indicate data transfers from the source specified on the right-hand side of the expression to the destination specified on the left-hand side of the expression.

allows access to the first 64-KB of address space, instead of the full 4 GB.

INDIRECT ADDRESSING

An important class of addressing modes is indirect addressing, namely, *register indirect* and *memory indirect*. These modes are called indirect because they do not specify an operand directly. Rather, machine instructions specify the operand indirectly by providing information from which the *effective address* (EA) of the operand can be determined.

Register Indirect and Relative Addressing

Instruction set architectures often support several variations of register indirect addressing, such as *register indirect, register indirect with postincrement* (often referred to as *autoincrement), register indirect with predecrement (autodecrement), register indirect with displacement, register indirect with (scaled) index,* and *register indirect with (scaled) index and displacement*.

With *register indirect* addressing, the instruction specifies a general-purpose register (referred to as an address or a pointer register) that contains the effective address of the memory location where the operand is located (the register R1 in Fig. 4d). If we assume that the content of the register R1 is 0x0000CA00, the operand is fetched from a memory location at address 0x0000CA00. An advantage of this addressing mode is that the instruction can reference an operand in memory without specifying a full memory address. Rather, just several bits are used to specify a general-purpose register. Several ISAs have separate address and data registers. For example, the Motorola 68000's ISA has 8 address registers and 8 data registers (2). This approach may further improve code density because only three bits are needed to specify an address or data register. However, this may require a more complex decoding logic.

Register indirect with index addressing means that the instruction specifies two registers used for computation of the effective address of an operand residing in memory. One register typically acts as a base register pointing to the beginning of an array, and the other register, named an index register, provides an offset from the beginning of the array to the current element (Fig. 4e). For example, let us assume that we want to increment elements of an integer array located in memory at the addresses A, A + 4, A + 8, ... up to the last element in the array. The starting address of the array is placed in the base register, and the index register is initialized to 0. A program loop traverses elements of the integer array, and a separate instruction increments the index register by the size of a single element in each loop iteration (by 4 in this example).

Often the value of the index register is equal to the array index no matter what is the size of one element. The index register is simply incremented or decremented by 1 and can also serve as a loop iteration counter. Hence, in address computations, the content of the index register has to be implicitly multiplied by a scale factor whose value is 1, 2, 4, or 8. The scale factor is inferred from the *opcode* and depends on the size of the operand. This variant is often referred to as *register indirect with scaled index* addressing or *scaled index* or just *scaled.*

Incrementing or decrementing of the index register is such a common operation that it is desirable to be done automatically using so-called *autoindexing (autoincrement* and *autodecrement)* addressing modes. With *autoincrement* addressing, the effective address of an operand is generated in the same way as in register indirect—the effective address is the content of the specified address register. An additional step is carried out—the content of the specified address register is incremented by the size of the operand (by 1 for byte size operands, by 2 for half-word size operands, by 4 for word size operands, and by 8 for double word size operands). With *autodecerement* addressing, the content of the address register is decremented by the size of the operand. Generally, this incrementing/decrementing step can be done before *(preincrement or*

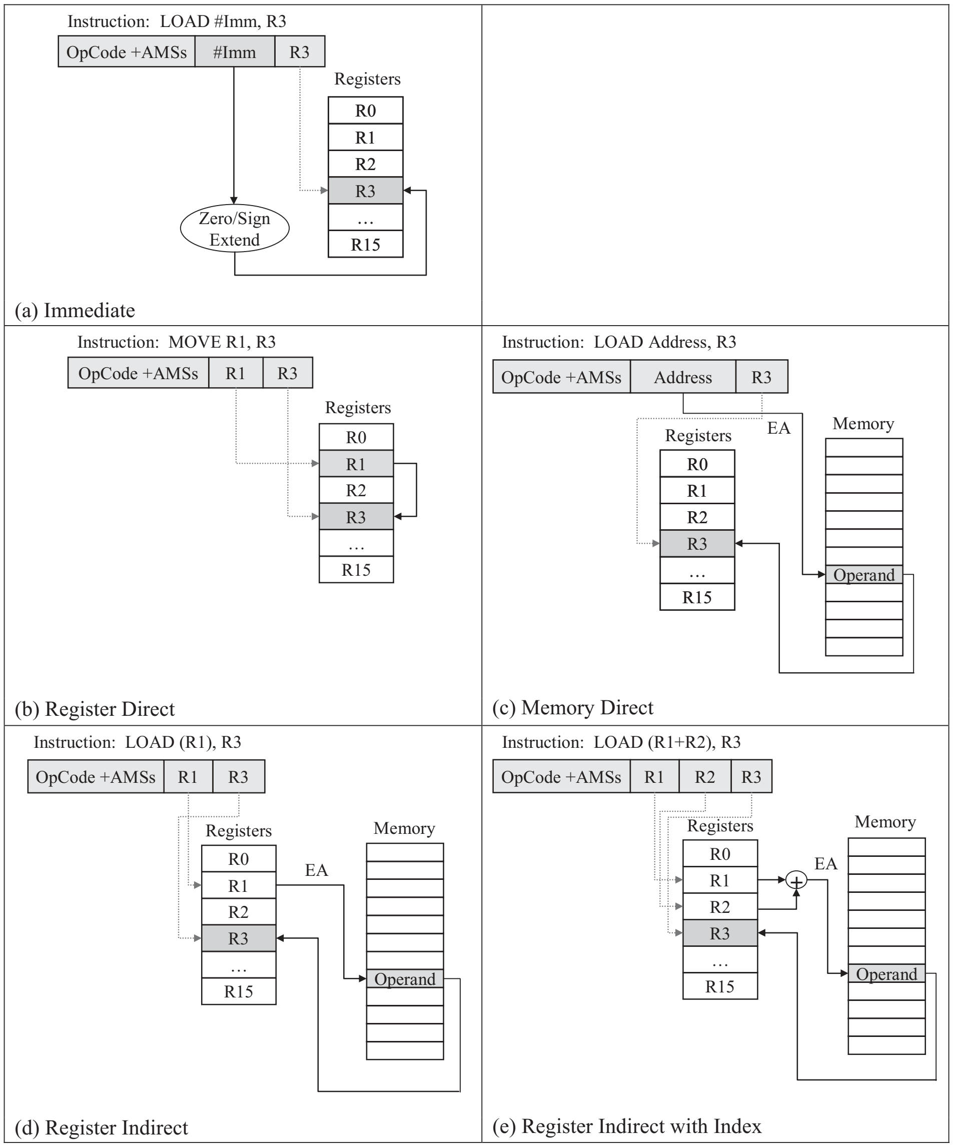
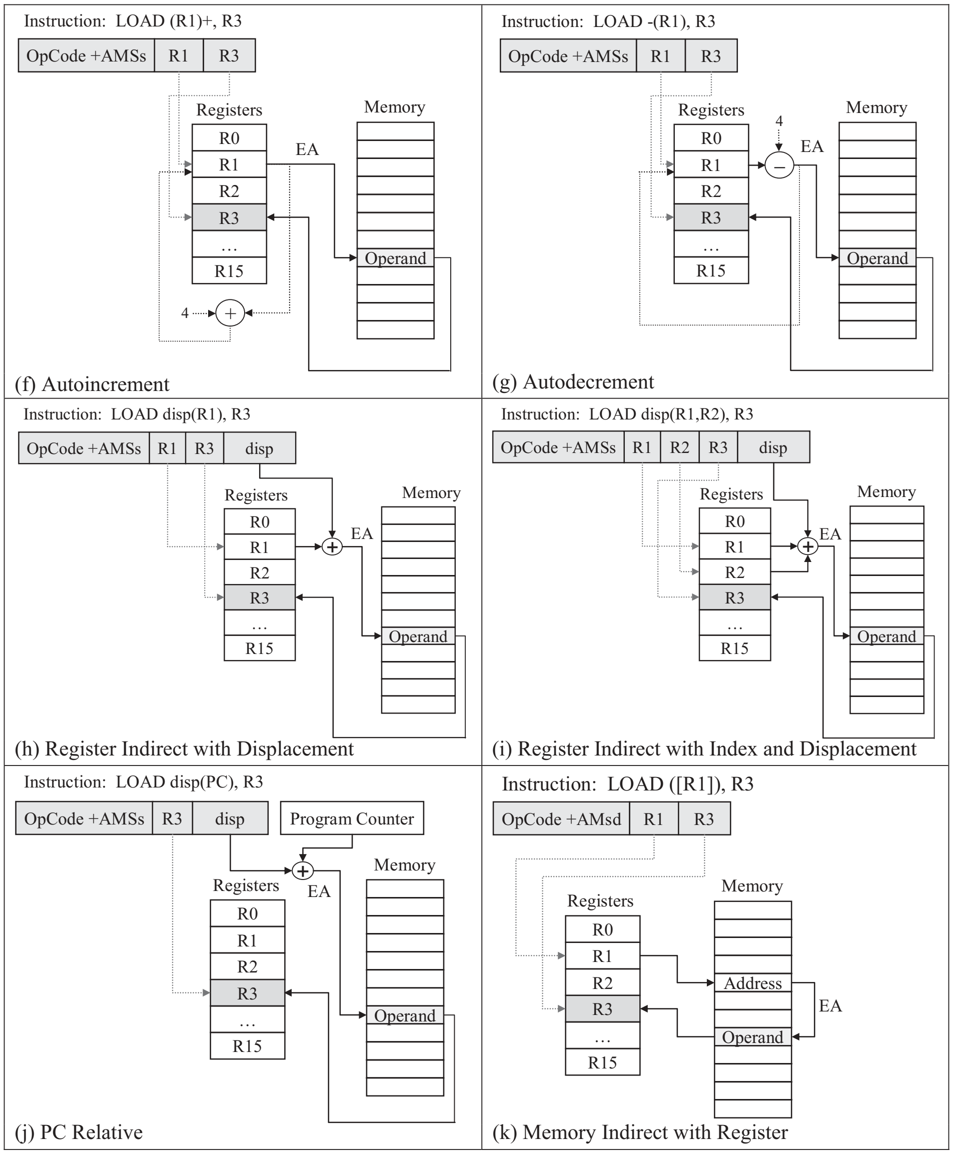


Figure 4. Illustration of addressing modes. (This figure is available in full color at http://www.interscience.wiley.com/reference/ecse.)



(f) Autoincrement

(g) Autodecrement

(h) Register Indirect with Displacement

(i) Register Indirect with Index and Displacement

(j) PC Relative

(k) Memory Indirect with Register

Figure 4. (*Continued*)

predecrement) or after *(postincrement or postdecrement)* the effective address computation. Instruction set architectures may support some or all of the four combinations pre/post auto-(increment/decrement). The most frequently used are post-autoincrement (Fig. 4f) and pre-autodecrement (Fig. 4g). These addressing modes are a very useful tool in accessing elements in regular arrays. The advantages of using autoindexing addressing modes compared with register indirect with index are as follows: (*1*) The machine instruction needs only one address field (the base register) leading to shorter instructions, (*2*) the program loops are shorter because we do not need an instruction to increment or decrement the index register, and (*3*) we use only one address register instead of two allowing more variables and pointers to be kept in general-purpose registers.

One of the most powerful addressing modes is *register indirect addressing with displacement,* also known as *base plus offset* addressing. It combines the capabilities of memory direct with register indirect addressing. The instruction specifies two address fields for the operand residing in memory, a register field and a displacement field (Fig. 4h). The effective address of the operand is calculated by adding the content of the specified address register (or base) to the content of the displacement field. The displacement field is typically shorter than the full address length; hence, the displacement field is zero- or sign-extended to the number of address bits before adding it to the content of the base register. This addressing mode is commonly used for accessing local variables allocated on the program stack. The base register is pointing to the beginning of the stack frame (also known as frame pointer), and the instruction specifies a displacement (or an offset) from this location to the location where the operand is located. Several architectures support a variant of this mode where the base register is defined implicitly, so instructions specify only the displacement field, thus reducing the number of bits needed to encode an instruction.

A natural extension of the previous addressing modes is *register indirect with index and displacement* addressing (Fig. 4i). An instruction using this mode includes three address fields for the memory operand: the base register, the index register, and the displacement field. The effective address is calculated as a sum of the contents in the base register, the index register, and the zero- or sign-extended displacement field. This addressing mode is a convenient tool in managing access to multidimensional data structures and for accessing local arrays on the stack. If the index register can be scaled, the addressing mode is *register indirect with scaled index and displacement*.

Another important class of addressing modes is so-called *relative addressing*. This class is similar to the register indirect addressing and its derivatives, except that the address of an operand is determined with respect to the content of the program counter (PC), rather than with respect to the content of the explicitly specified address register (Fig. 4j). For *PC relative* addressing, the instruction specifies only the displacement field, and the effective address is calculated as a sum of the contents of the program counter and the zero- or sign-extended displacement field.

Similarly, with *PC relative with index* addressing, the instruction word specifies an index register, and the effective address is calculated as a sum of the program counter and the content of the index register. Finally, for *PC relative with index and displacement* addressing, the instruction specifies the index register and the displacement, and the effective address is equal to the sum of the contents of the program counter, the index register, and the zero- or sign-extended displacement field.

An important question related to all addressing modes with displacement is what should be the size of the displacement field in the instruction. The size of this field directly influences the instruction length; hence, it is desirable to have a short displacement field. However, a shorter displacement field may limit its usefulness. An experimental evaluation of displacement length finds that a 16-bit displacement field is suitable for 32-bit processors (1). Several instruction set architectures with variable instructions lengths support both short and long displacements (d8—8-bit displacement, d16—16-bit displacement, and d32—32-bit displacement) (3).

Memory Indirect Addressing

With *memory indirect* addressing, both the operand and its address are in memory. The instruction specifies the location in memory where the operand address (pointer) is located. First the operand address is read from memory, and then an additional memory read or write is performed to access the operand. There are many variants of memory indirect addressing depending on how we calculate the address of the pointer and how that pointer is used in determining the address of the operand.

The *memory indirect with register* mode means that the instruction specifies a general-purpose register pointing to a location in memory that keeps the pointer to the operand. For example, let us consider the following instruction from Table 2: LOAD ([R1]), R3. The effective address of the operand is in memory at the address determined by the content of the register R1 (Fig. 4k). Once the effective address is fetched from memory, another read from memory is employed to fetch the operand. The *memory indirect absolute* means that the instruction contains the direct address of the pointer location in memory.

The Motorola 68020 instruction set architecture supports a range of memory indirect addressing modes known as *preindexed* and *postindexed memory indirect* and *PC*

Table 2. Memory Indirect Data Addressing Modes, Example Instructions, Description, and Typical Use

Addressing Mode	Example Instruction	RTL Description	Typical Use
Memory indirect with register	`LOAD ([R1] ), R3`	EA ← Mem([R1]) [R3] ←[Mem(EA)]	Jump tables; Linked lists
Memory indirect absolute	`LOAD ([$8000] ), R3`	EA ← Mem($8000) [R3] ←[Mem(EA)]	Access to structures through a pointer

Memory indirect, postindexed	`LOAD ([$100, R1] , R2*4, $200), R3`	Temp ← Mem($100 + R1)
		EA ← Temp + R4*4 + $200
		[R3] ←[Mem(EA)]
			Jump tables; Linked lists; Access to complex data structures through array of pointers

Memory indirect, preindexed	`LOAD ([$100, R1, R2*4] , $200), R3`	Temp ← Mem($100 + R1 + R2*4)
		EA ← Temp + $200
		[R3] ←[Mem(EA)]
			Jump tables; Linked lists; Access to complex data structures through array of pointers

PC memory indirect, postindexed	`LOAD ([$100, PC] , R2*4, $200), R3`	Temp ← Mem($100 + PC)
		EA ← Temp + R2*4 + $200
		[R3] ←[Mem(EA)]
			Jump tables; Linked lists; Access to complex data structures through array of pointers

PC memory indirect, preindexed	`LOAD ([$100, PC, R2*4] , $200), R3`	Temp ← Mem($100 + PC + R2*4)
		EA ← Temp + $200
		[R3] ←[Mem(EA)]
			Jump tables; Linked lists; Access to complex data structures through array of pointers

memory indirect addressing as shown in Table 2. The *memory indirect postindexed* addressing can be specified as follows: ([*bd,* Rx], Ry*sc, *od*). The instruction has 4 address fields for the operand in memory, and they specify a base register *Rx,* an index register *Ry,* a base displacement *bd,* and an outer displacement *od.* The scale factor *sc* depends on the operand size and can be 1, 2, 4, or 8. The processor calculates an intermediate memory address using the base address register and the base displacement *(Rx + bd)* and reads the pointer from this location. The address read from memory is added to the index portion (Ry*sc) and the outer displacement to create the effective address of the operand as follows: $EA = Mem\ [Rx + bd] + Ry^{*}sc + od$. The displacements and the index register contents are sign-extended to 32 bits. In the syntax for this mode, brackets enclose the values used to calculate the intermediate memory address. All four user-specified values are optional and can be suppressed. Both the base and outer displacements may be null, 16-bit, or 32-bit long. When suppressing a displacement or a register, its value is zero in the effective address calculation. A large number of addressing options can be created using suppressing. For example, by suppressing *bd, Ry,* and *od,* the resulting addressing mode is equivalent to the *memory indirect with register* addressing. By suppressing all but the 32-bit outer displacement *od,* the resulting addressing mode is equivalent to the *memory indirect absolute* addressing.

The *memory indirect preindexed* mode is described with the following specification: (*[bd, Rx, Ry*sc], od*). The intermediate indirect address is the sum of the base displacement, the base address register, and the scaled index register. The address fetched from memory at this location is added to the outer displacement to create the effective address of the operand: $EA = Mem[Rx + bd + Ry^{*}sc] + od$. The *PC indirect memory preindexed and postindexed* modes are equivalent to the memory indirect preindexed and postindexed modes, except that the program counter is used instead of the base address register *Rx*.

ADDRESSING MODES FOR CONTROL FLOW INSTRUCTIONS

Addressing modes are important not only for instructions dealing with data, but also for instructions that change the program control flow. The most frequently used addressing mode for branch instructions is *PC relative with displacement* and its derivatives, such as *PC relative with index,* and *PC relative with index and displacement.* The branch instruction specifies an offset or displacement relative to the current program counter. The *PC-relative* addressing is preferred because the target address is often near the current instruction, and hence, an offset field with just a few bits would suffice. The PC-relative addressing also allows *position independence* of the code, meaning that the code can run independently of where it is loaded in memory. This property is useful because it reduces the number of tasks for linkers and makes dynamically linked programs easier to implement.

Other addressing modes are also applicable to indirect branches and jumps. The target address is equal to the effective address specified by the instruction. For example, let us consider an unconditional indirect branch instruction JMP (R3) that specifies the register indirect addressing mode. The branch target address is the content of the register R3 and, hence, determined at runtime. It should be noted that branch instructions operate on addresses rather than data, and hence, we do not fetch the data from memory location pointed to by the register R3. These branches are useful in implementing high-level language constructs such as switch statements, virtual functions and methods, and dynamically shared libraries.

One important design decision made by ISA designers is the size of the offset field for PC-relative addressing. The distributions of branch offsets can be measured on a set of representative benchmarks in order to determine the optimal size of the offset field. One such measurement indicates that shorter offsets that can be encoded using up to 8 bits dominate (1).

A CASE STUDY

Instruction set architectures support only a subset of all addressing modes discussed so far. Which addressing modes are supported depends on many parameters. In selecting addressing modes, computer architects would like to be able to address a large number of locations in memory with maximum flexibility. Flexibility means that each machine instruction can be combined with any addressing mode, allowing compilers to produce more optimized code with maximal code density. These requirements favor a rich set of addressing modes and variable instruction lengths. On the other hand, it is desirable to have fixed and uniform instruction encodings that reduce the complexity of decoding and address calculations as well as latencies for these steps. These often conflicting requirements are carefully evaluated during the design process.

Today almost all programming is done in high-level languages and instructions executed are the output of compilers. Hence, compilers play a critical role and should not be omitted when deciding about the ISA design and addressing modes. Having sophisticated instruction sets including a wide range of powerful addressing modes does not guarantee efficient code if compilers are not able to use them effectively. What memory addressing modes are most frequently used in compiler-generated code? A study based on a few programs indicates that three most frequently used modes account for 88% of all memory accesses: the *register indirect with displacement* accounts on average for 42%, *immediate* accounts for 33%, and *register indirect* accounts for 13% (1). Current technology trends favor *Reduced Instruction Set Computers* (RISC) with load/store architectures, fixed instruction lengths, a rich set of general-purpose registers, and a relatively small set of most frequently used addressing modes. In embedded computers where code density is of utmost importance, hybrid instruction encodings are used with a somewhat richer set of addressing modes.

In this subsection, we give a short overview of addressing modes found in three different instruction set architectures: Intel's IA32 (3), Motorola's 68000 (2), and

Table 3. Summary of Addressing Modes Supported by Intel's IA32, Motorola 68000, and ARM ISA for Data and Control-Flow Instruction

Addressing modes	IA32		M68000		ARM	
	Data	Control	Data	Control	Data	Control
Immediate	X	–	X	–	X	–
Register direct	X	–	X	–	X	–
Absolute (Direct addressing in IA32)	X	X	X	X	–	–
Register indirect	X	X	X	X	X	–
Autoincrement	X	–	X	–	X	–
Autodecrement	X	–	X	–	X	–
Register indirect with displacement (Register relative in IA32)	X	–	X	X	X	–
Register indirect with (scaled) index (Base plus index/(Scaled) in IA32)	X	–	X	X	X	–
Register indirect with (scaled) index and displacement (Base relative plus index in IA32)	X	–	X	X	X	–
PC relative	–	X	X	X	X	X
PC relative with index	–	–	X	X	X	–
PC relative with index and displacement	–	–	X	X	X	–
Memory indirect	–	X	X	X	–	–
PC memory indirect	–	–	X	X	–	–

ARM (4). IA32 and Motorola 68000 are representative examples of *Complex Instruction Set Computer* (CISC) instruction sets, with variable instruction encoding, register-memory architectures, a small set of general-purpose registers, 2-operand instructions, and a rich set of addressing modes. The ARM is a RISC processor with load/store architecture, fixed instruction encodings, 16 general-purpose registers, 3-operand instructions, and yet with a very sophisticated addressing. Table 3 gives a summary of addressing modes for data processing and control-flow instructions supported by these three instruction set architectures. Figure 5 shows code produced by the GCC compiler for these ISAs for a simple code snippet that summarizes elements of an integer array. This example demonstrates superior code density achieved by CISC ISAs: Motorola 68000 requires 6 bytes, IA32 13 bytes, and ARM 20 bytes of code.

The text below discusses the instruction encoding for the ARM data transfer instructions and how it achieves many addressing options derived from a single mode, the *register indirect with displacement* addressing (Table 4). Figure 6 shows the binary encoding for single-word (B=0) and unsigned byte (B=1) data transfer instructions, namely for loads (L=1) or stores (L=0). When the program counter is specified as the base register, this addressing mode corresponds to the traditional *PC relative* addressing and its derivatives. The effective address for a memory operand is calculated by adding or subtracting an offset to the content of a base register. The offset is either (*1*) an unsigned immediate directly encoded in

```
for (i=0; i<100; i++) {
  sum += a[i];
 }

IA32:
Address     Binary encoding       Label   Assembly code
                                  .L9:
0024        03948568 FEFFFF               addl    -408(%ebp,%eax,4), %edx
002b        40                            incl    %eax
002c        83F863                        cmpl    $99, %eax
002f        7EF3                          jle     .L9

M68000:
Address     Binary encoding       Label   Assembly code
                                  .L9:
002a        D498                          add.l (%a0)+,%d2
002c        5380                          subq.l #1,%d0
002e        6AFA                          jbpl .L9

ARM:
Address     Binary encoding       Label   Assembly code
                                  .L26:
002c        023190E7                      ldr     r3, [r0, r2, asl #2]
0030        012082E2                      add     r2, r2, #1
0034        630052E3                      cmp     r2, #99
0038        031081E0                      add     r1, r1, r3
003c        090000DA                      ble     .L26
```

Figure 5. IA32, M68000, and ARM code produced by the GCC compiler for the program loop on the top.

Table 4. ARM Addressing Modes. Note: PC Relative Addressing Modes are Inferred From the Preindexed with the PC Serving as the Base Register

Addressing modes	Example	RTL Description
Immediate offset		

Preindexed	`LD r0,[r1, #±offset12]`	`EA ←[r1] ± offset`
		`[r0] ←[Mem(EA)]`

Preindexed with writeback	`LD r0,[r1, #±offset12] !`	`EA ← r1 ± offset`
		`[r0] ←[Mem(EA)]`
		`[r1] ←[r1] ± offset`

Postindexed	`LD r0,[r1], #±offset12`	`EA ←[r1];`
		`[r0] ←[Mem(EA)]`
		`[r1] ←[r1] ± offset`

(Scaled) register offset		

Preindexed	`LD r0,[r1, ±r2, Sh, #Shift]`	`EA ←[r1] ±([r2] Sh #Shift)`
		`[r0] ←[Mem(EA)]`

Preindexed with writeback	`LD r0,[r1, ±r2, Sh, #Shift] !`	`EA ←[r1] ±([r2] Sh #Shift)`
		`[r0] ←[Mem(EA)]`
		`[r1] ←[r1] ±([r2] Sh #Shift)`

Post-indexed	`LD r0,[r1], ±r2, Sh, #Shift`	`EA ←[r1]`
		`[r0] ←[Mem(EA)]`
		`[r1] ←[r1] ±([r2] Sh #Shift)`

the instruction, (*2*) the content of a register, or (*3*) a scaled value of a register. Bit 25 of the instruction word determines the interpretation of the 12 least significant bits (Fig. 6). When bit 25 is a "0", the effective address of the operand is computed as follows: The 12-bit immediate field is added to (U=1) or subtracted from (U=0) the content of the base register *Rn*. When bit 25 is a "1", the effective address is computed as follows: The content of the scaled index register *Rm* is added to or subtracted from the content of the base register *Rn*. Scaling type is defined by the 2-bit *Sh* field, and the shifting magnitude is defined by a 5-bit *Shift* field. A pre-indexed (P=1) addressing uses the computed address for the load or store operation. If a write-back is requested (W=1), the base register is updated to the computed value. A post-indexed (P=0) addressing uses the unmodified base register for the effective address calculation, and then it updates the base register independently of the W field.

Several addressing modes can be inferred from this encoding. For example, the post-indexed or pre-indexed register indirect with the immediate field equal to zero corresponds to the classic *register indirect* addressing mode. When the immediate value is nonzero, this mode corresponds to *the post- or pre-indexed autoincrement* or *autodecrement* addressing modes. It should be noted that a base register could be incremented or decremented by any value that can be encoded in the 12-bit immediate value, providing an increased flexibility compared with the classic implementations of the autoindexing modes. If the offset address comes from an index register, the addressing mode corresponds to the *register indirect with index mode.* When the value in the index register is scaled, this addressing mode becomes equivalent to the *register indirect with scaled index mode,* but with increased flexibility in selecting a scale factor. Thus, thanks to very clever encodings, a rich set of addressing modes is supported.

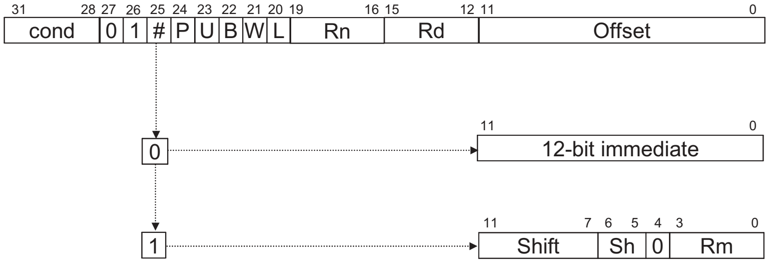


Figure 6. Binary encoding of word-size and unsigned byte-szie data transfer instructions. Legend: Rn - base register, Rd - source/destination register, L-load or store, W - write-back (autoindex), B-unsigned byte or word, U - up/down, P - pre or post index, cond - condition field enabling conditional execution.

BIBLIOGRAPHY

1. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 4th ed., San Maleo, CA: Morgan Kaufmann, 2007.
2. A. Clements, *Microprocessor Systems Design: 68000 Hardware, Software, and Interfacing*, 3rd ed., New York: PWS Publishing Company, 1997.
3. Intel(®) 64 and IA-32 *Architectures Software Developer's Manual Volume 1: Basic Architecture*, Available: http://www.intel.-com/products/processor/manuals/index.htm.
4. S. Furber, *ARM Systems-on-Chip Architecture* Reading, MA: Addison-Wesley, 2000.

READING LIST

J. L. Hennessy, VLSI processor architecture, *IEEE Trans. Comput.*, **33** (12): 1221–1246, 1984.

D. Patterson, Reduced instruction set computers, *Commun. ACM*, **28** (1): 8–21, 1985.

Aleksandar Milenkovic
University of Alabama, Huntsville
Huntsville, Alabama

A

ANALOG-TO-DIGITAL CONVERSION IN THE EARLY TWENTY-FIRST CENTURY

Analog-to-digital converters (ADCs) continue to be important components of signal-processing systems, such as those for mobile communications, software radio, radar, satellite communications, and others. This article revisits the state-of-the-art of ADCs and includes recent data on experimental converters and commercially available parts. Converter performances have improved significantly since previous surveys were published (1999–2005). Specifically, aperture uncertainty (jitter) and power dissipation have both decreased substantially during the early 2000s. The lowest jitter value has fallen from approximately 1 picosecond in 1999 to <100 femtoseconds for the very best of current ADCs. In addition, the lowest values for the IEEE Figure of Merit (which is proportional to the product of jitter and power dissipation) have also decreased by an order of magnitude. For converters that operate at multi-GSPS rates, the speed of the fastest ADC IC device technologies (e.g., InP, GaAs) is the main limitation to performance; as measured by device transit-time frequency, f_T, has roughly tripled since 1999. ADC architectures used in high-performance broadband circuits include pipelined (successive approximation, multistage flash) and parallel (time-interleaved, filter-bank) with the former leading to lower power operation and the latter being applied to high-sample rate converters. Bandpass ADCs based on delta-sigma modulation are being applied to narrow band applications with ever increasing center frequencies. CMOS has become a mainstream ADC IC technology because (*1*) it enables designs with low power dissipation and (*2*) it allows for significant amounts of digital-signal processing to be included on-chip. DSP enables correction of conversion errors, improved channel matching in parallel structures, and provides filtering required for delta-sigma converters. Finally, a performance projection based on a trend in aperture jitter predicts 25 fs in approximately 10 years, which would imply performance of 12 ENOB at nearly 1-GHz bandwidth.

INTRODUCTION

During the past three decades, especially the past 7–10 years, the increasingly rapid evolution of digital integrated circuit technologies, as predicted by Moore's Law, has led to ever more sophisticated signal-processing systems. These systems operate on a wide variety of continuous-time signals, which include speech, medical imaging, sonar, radar, electronic warfare, instrumentation, consumer electronics, telecommunications (terrestrial and satellite), and mobile telecommunications (cell phones and associated networks).

Analog-to-digital converters (ADCs) are the circuits that convert these and other continuous-time signals to discrete-time, binary-coded form, that is, from human-recognizable form to computer-recognizable form. Such a conversion can be thought of as a two-step process. First, the input signal is sampled in time, usually at regularly spaced intervals; $f_{samp} = 1/T$, where T = sampling interval (e.g., for $f_{samp} = 1$ gigasample per second, T = 1 ns). The second step is to quantize (or digitize) the samples (usually a voltage) in amplitude. The maximum signal range (full-scale input voltage) is divided into 2^N subranges, where N = the ADC's resolution (number of output leads), (e.g., for $N = 12$ bits, a 1-Volt full-scale range is divided into $2^N = 4096$ levels). The least-significant bit (LSB) is $1\,V / 2^N = 244$ µV. Two potential purposes for these conversions are (*1*) to enable computer analysis of the signal and (*2*) to enable digital transmission of the signal.

The limitations of ADCs in terms of both resolution and sampling rate are determined by the capability of the integrated circuit (IC) process, as well as chip design techniques, used to manufacture them, and, were perceived as a limiting factor (1) to system performances as recently as 1999. However, more recent developments have demonstrated that significant progress has occurred with respect to converter performances, especially for sampling rates in the 100 megasamples/s (MSPS) range as well as in the neighborhood of 1 gigasample/s. The purpose of this article is to provide an update to previous ADC surveys (1–3) and to analyze the new results.

The next section of this article summarizes the characterization and limitations of ADCs, and the section on ADC performance update discusses how ADC performances have changed (improved) during the past 8 years. The next section covers Figures of Merit, and the section on high-performance ADC architectures discusses architectures that are presently in use along with the advantages conferred by increased ADC IC complexity. Then, performance trends and projections are discussed, and finally, the last section gives a summary and provides conclusions. Appendix 1 contains a list of the more than 175 ADCs covered in this work.

PERFORMANCE CHARACTERIZATION AND LIMITATIONS

One of the key parameters describing ADC performances is signal-to-noise plus distortion ratio, (SNDR), which is defined as the ratio of the root mean square signal amplitude to the square root of the integral of the noise power spectrum (including spurious tones) over the frequency band of interest [e.g., for a Nyquist converter, the pertinent band extends from 0 to one-half the sampling rate ($f_{samp}/2$)]. Another way of expressing this ratio is in terms of the effective number of bits (ENOB) which is obtained from SNDR(dB)= $20^{*}\log_{10}$(SNDR) by Ref. 4.

$$\mathrm{ENOB} = \frac{\mathrm{SNDR(dB)} - 1.76}{6.02} \qquad (1)$$

This expression includes the effects of all losses associated with the subject ADC, which include equivalent

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

input-referred thermal noise, $ENOB_{thermal}$, aperture uncertainty (jitter) noise, $ENOB_{aperture}$, and comparator ambiguity, $ENOB_{ambig}$, (1), as well as distortion induced by spurious tones. If we were dealing with a loss-free, distortion-free ADC, then the value of ENOB obtained from Equation (1) would equal the number of output leads *N*, and it would correspond to the (intrinsic) quantization noise case. Expressions for ENOB for thermal noise, jitter, and comparator ambiguity losses (each acting separately) are given below:

$$ENOB_{\text{thermal}} = \log_2\left(\frac{V_{pp}^2}{12kTR_{\text{eff}}f_{\text{sig}}}\right)^{1/2} - 1 \qquad (2)$$

$$ENOB_{\text{aperture}} = \log_2\left(\frac{1}{2\pi f_{\text{sig}}\tau_a}\right) - 1 \qquad (3)$$

$$ENOB_{\text{ambiguity}} = \frac{\pi f_T}{13.9 f_{\text{sig}}} - 1.1 \qquad (4)$$

Here, V_{pp} is the maximum peak-to-peak input voltage presented to the ADC, k is Boltzmann's constant, T is the semiconductor substrate temperature in kelvins, R_{eff} is the effective input resistance, f_{sig} is the analog input frequency, τ_a is the rms aperture jitter, and f_T is the transit-time frequency associated with the transistors used in the given ADC IC process technology.

It is noted that for Nyquist ADCs, f_{sig} in Equations (2)–(4) can be replaced by $f_{samp}/2$, and the resulting expressions would correspond closely, apart from one small numerical difference in Equation (3), to those presented in Ref. 1 with $ENOB_x$ replacing B_x, where *x* is thermal or aperture or ambiguity. However, for non-Nyquist operation, (e.g., under-sampling), Equations (2)–(4) should be used as written above because f_{sig} can be different (greater) than $f_{samp}/2$. Hence, these expressions are more general than those in Ref. 1.

Another important ADC characteristic is spurious-free dynamic range (SFDR), which when expressed in dBc, is the difference of the input signal magnitude and the largest spurious tone in the frequency-band of interest. The quantity SFDR bits is analogous to ENOB and is defined by

$$\text{SFDR bits} = \text{SFDR(dBc)}/6.02 \qquad (5)$$

A complete characterization of an ADC includes measuring the values of SNDR and SFDR as a function of input signal frequency f_{sig} with f_{samp} as a parameter. For a baseband ADC, at low values of f_{sig}, the SNDR is constant with respect to f_{sig} and then decreases as f_{sig} increases. The value of f_{sig} at which the SNDR decreases to 3 dB below the low-frequency value is the effective resolution bandwidth (ERBW). This important characteristic implies the range of frequencies over which the converter may be used. If ERBW $\geq f_{samp}/2$, then the ADC is a Nyquist converter, which is the design goal of many ADCs. Not all published reports on alleged Nyquist ADCs satisfy the conditions for Nyquist conversion. However, some still achieve noteworthy sampling speed, SNDR or SFDR. To include these examples, the values for f_{sig}, f_{samp}, as well as associated SNDR and SFDR are stated clearly herein.

Some converters have ERBW values well beyond $f_{samp}/2$, and, as noted above, imply they can be used in undersampling mode. Because this type of operation involves sampling upper Nyquist zones, an antialiasing filter (AAF) must be placed in the signal path and in front of the converter to prevent contamination by unwanted out-of-zone signals. This Process results in using the ADC as a bandpass converter in which the passband of the AAF defines the band of interest, and, through the natural mixing property of the sampling process, the passband is shifted to baseband. The appropriate value of ENOB for the bandpass case is the midband value. An example would be sampling at 100 MSPS while using an AAF that covers the band 100 MHz $+ \delta$ to 150 MHz - δ (third zone), where δ is determined from the AAF skirts and is large enough to keep out unwanted signals and/or tones.

In addition, delta-sigma ($\Delta\Sigma$) converters (modulator followed by digital decimation filter) are coming more into use for bandpass [intermediate frequency (IF) and radio frequency (RF) sampling] conversion as well as (well-established) lowpass (baseband sampling). These ADCs are each characterized, in this study, as having an effective sampling rate = 2 × $\Delta\Sigma$ passband (determined by the modulator 3-dB bandwidth), and an ERBW = $\Delta\Sigma$ passband. Continuous time $\Delta\Sigma$ ADCs do not usually require AAFs (it is built-into the $\Delta\Sigma$ architecture) whereas discrete-time $\Delta\Sigma$ ADCs do. Again, the midband value of ENOB applies here.

Two-tone intermodulation distortion (IMD) of ADCs is particularly relevant for receiver applications. One excites an ADC with two sinusoids of equal amplitude but with different frequencies, f_1 and f_2, and observes spurious tones in the FFT spectrum of the ADC output. The strongest such tone is usually second- ($\pm f_1 \pm f_2$) or third-order ($\pm f_1 \pm 2f_2$, or $\pm 2f_1 \pm f_2$). Unfortunately, whereas IMD data are reported in the literature, no standard set of conditions are available for IMD evaluation, which makes comparisons between ADCs somewhat difficult. Hence, IMDs must be evaluated by the prospective user for the intended application.

ADC PERFORMANCE UPDATE

Figure 1 shows ENOB as a function of analog input frequency (Fig. 1a) and of sampling rate (Fig. 1b) for state-of-the-art ADCs (as of late 2007). The use of f_{sig} rather than f_{samp} as abscissa in Figs. 1a and 2, and as independent variable in Equations (2)–(4), reflects the fact that many modern converters can sample in upper Nyquist zones. As discussed above, this finding means that the analog input bandwidth for such ADCs extends beyond $f_{samp}/2$ and is indicative of the attainment of lower aperture uncertainty values, and hence higher ENOB via Equation (3). The downward sloping lines, with slopes equal to −3.33 ENOB/decade (−20 dB/decade) and labeled 1999 (jitter = 1 ps) and 2007 (jitter = 100 fs) in the figure, represent the state-of-the-art for those years. It is evident that typical best values for ENOB (Fig. 1a) have increased by more than 3 effective bits, which corresponds to an order of magnitude improvement in resolution. In fact, as will be shown in the next section, most of this advance has

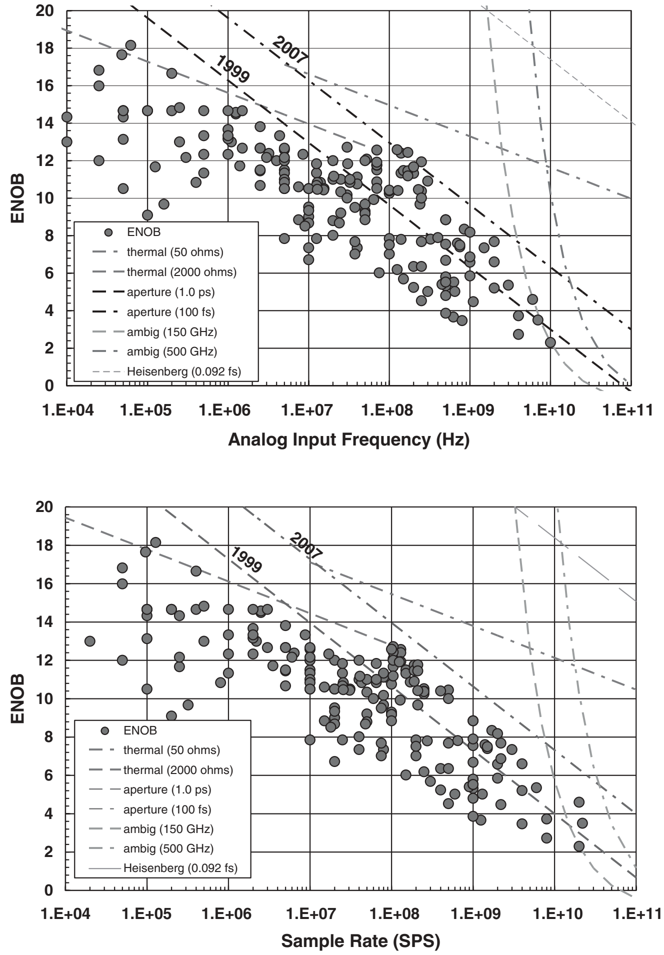


Figure 1. (a) Updated survey of state-of-the-art analog-to-digital converters. The graph shows the effective number of bits (*ENOB*) as a function of analog input frequency for each converter. The downward sloping lines labeled 1999 and 2007 represent the state-of-the-art for those years, and these values correspond to aperture jitter values of 1 ps and 100 fs, respectively. The data indicate an improvement of more than 3 *ENOB* during the past 8 years. The other curves represent the limitations caused by input-referred noise (thermal), comparator ambiguity (ambig), and ultimately the Heisenberg Uncertainty Principle. (b) In this graph, the effective number of bits (*ENOB*) is shown as a function of sample rate for each converter. The downward sloping lines labeled 1999 and 2007 are repeated from Fig. 1a and represent the jitter state-of-the-art for those years. Here, the apparent improvement is approximately 2 ENOB instead of three, which occures because the converters that operate in undersampling mode are not as apparent in this graph.

occurred within the past 4 years. In Fig. 1b, the two jitter lines are repeated using the correspondance $f_{sig} = f_{samp}/2$. Here, the overall improvement seems to be only a little more than 2 ENOB rather than 3 ENOB, because the true performance of converters that operate in undersampling mode is not evident when f_{samp} is used as abscissa. One might interpret this comparison as indicating two thirds of the SNDR increase is caused by improved sampling and one third is caused by an improved capability to undersample.

The thermal noise limitation is indicated by the more gently sloping lines (−10 dB/decade) in Fig. 1a and 1b. At values of $f_{sig} \leq 3$ MHz, the effective input-referred resistance for most ADCs in this range is on the order of a few thousand ohms (lower line is for 2000 ohms) and seems to limit the attained ENOB. For systems that operate at higher input frequencies, the input-referred resistance shifts toward 50 ohms (upper line is for 50 ohms) and is less of a limitation compared with aperture uncertainty. At very high values of f_{sig}, (i.e., a few GHz), ADC performance is limited abruptly by comparator ambiguity (IC technology presently at $f_T \sim 150$ GHz). However, this limitation has been pushed out by about a factor of three since 1999 (1). It is noted that the ultimate limit on ADC performance is suggested by the Heisenberg Uncertainty Principle, [e.g., 16 ENOB at 30 GHz (jitter ∼ 0.1 fs)].

The SFDR for the converters covered in this article is shown in Fig. 2. Here, an improvement of 3 SFDR bits has occurred (essentially the same change as in Fig. 1a). Finally, it is pointed out that the recent improvements in the state-of-the-art have occurred in the frequency range of ∼50 MHz to ∼300 MHz; it is felt that this change was caused by an emphasis on communications applications. Over 175 converters (Appendix 1), which include experimental ADCs and commercially available parts, are represented in Figs. 1 and 2.

It is worthwhile to compare ENOB and SFDR bits with the stated number of bits (number of output pins) because confusion often occurs over the interpretation of these quantities. These comparisons are shown in Fig. 3 in which the differences (stated bits–ENOB) and (stated bits − SFDR bits) are shown as functions of f_{sig}. As can be observed from the graphs in Fig. 3, the differences are not too dissimilar to those reported in 1999 (1), whereas the spreads in the data remain rather broad (stated bits − ENOB = 1.53 ± 1.5 bits; stated bits − SFDR bits = 0.05 ± 3.1 bits). The reasons for such diversity are complex, but most likely include (*1*) the variety of design objectives

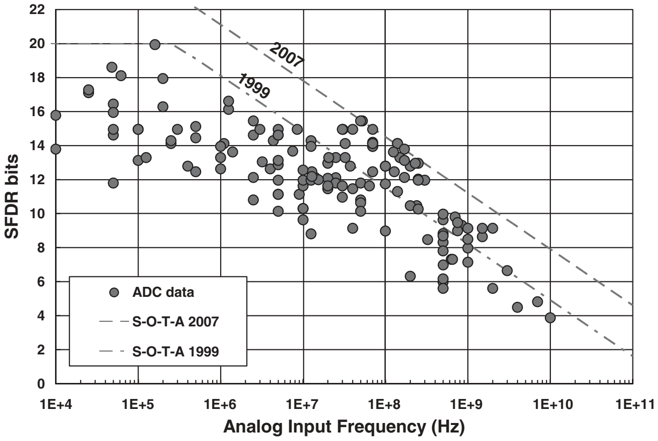


Figure 2. Spur-free dynamic range expressed as SFDR bits plotted against input frequency for each converter. The downward sloping lines labeled 1999 and 2007 represent the state-of-the-art for those years. These data indicate an improvement of approximately 3 SFDR bits during the past 8 years.

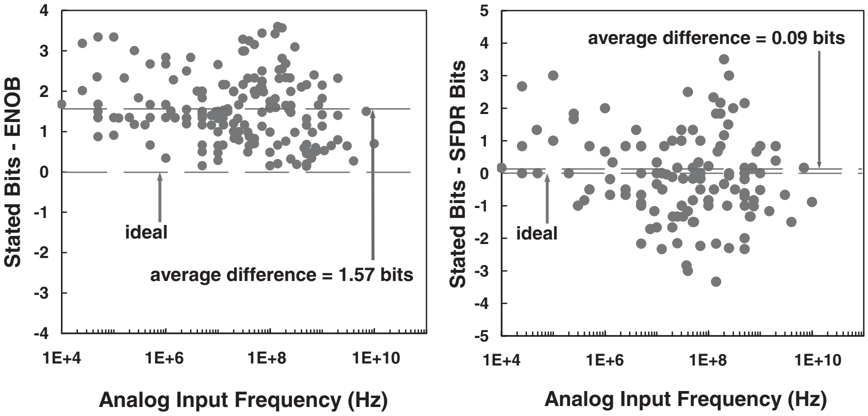


Figure 3. The differences between stated bits (number of outputs) and ENOB and SFDR bits for the ADC population considered in this study are shown here. For ENOB, the average difference is about 1.5 bits, whereas for SFDR bits it is nearly zero. However, a wide range exists in the data: > ± 1.5 bits for ENOB and > ± 3 bits for SFDR.

(high SNDR, or high SFDR, and/or low power dissipation, etc.) and (*2*) the variety of IC technologies (many flavors of CMOS, Si bipolar, SiGe, InP, GaAs, hybrids, etc.) applied to the realization of these circuits.

FIGURES OF MERIT

Figures of Merit are useful for assessing the performance of ADCs. Four examples are presented below, one for performance regardless of power and three that account for it.

The expression for aperture uncertainty,

$$\tau_a = \frac{1}{4 \cdot \pi \cdot \ f_{sig} \cdot 2^{ENOB}} \quad \text{(seconds)} \tag{6}$$

can be used as a performance measure because it contains the product of the quantities f_{sig} and 2^{ENOB} (similar to a gain-bandwidth product), and it is indicative of ADC state-of-the-art (e.g., $\tau_a = 1$ ps and 100 fs in Fig. 1a).

When it is desired to include power dissipation, P_{diss}, an "old" figure of merit (1) could be used,

$$F = \frac{2^{ENOB} \cdot f_{samp}}{p_{diss}} \quad \text{(effective LSBs per Joule)} \tag{7}$$

This expression does not account for the possibility that f_{sig} can be greater than $f_{samp}/2$(i.e., undersampling mode). A "new(er)" figure of merit, which does include this case, is obtained from

$$F_a = \frac{1}{2 \cdot \pi \cdot \tau_a \cdot P_{diss}} = \frac{2 \cdot f_{sig} \cdot 2^{ENOB}}{P_{diss}} \quad \text{(effective LSBs per Joule)} \tag{8}$$

Note that for a Nyquist ADC, F_a reduces to F. Taking this equation a step further, F_a is inverted and then becomes a more general form of the IEEE Figure of Merit,

$$FOM_a = \frac{1}{F_a} = \frac{P_{diss}}{2 \cdot f_{sig} \cdot 2^{ENOB}} \quad \text{(Joules per effective LSB)} \tag{9}$$

Another way of looking at equation (8) is as the product,

$$F_a \cdot \tau_a = \frac{1}{2 \cdot \pi \cdot P_{diss}} \tag{10}$$

which is a function only of power dissipation. Figure 4 shows F_a versus τ_a for the ADC population. A wide variation in P_{diss} is evident nearly six orders of magnitude.

HIGH-PERFORMANCE ADC ARCHITECTURES

The highest-performing ADCs are listed in Table 1(5–13). They are sorted primarily by aperture uncertainty and secondarily by the IEEE Figure of Merit, FOM_a. The list

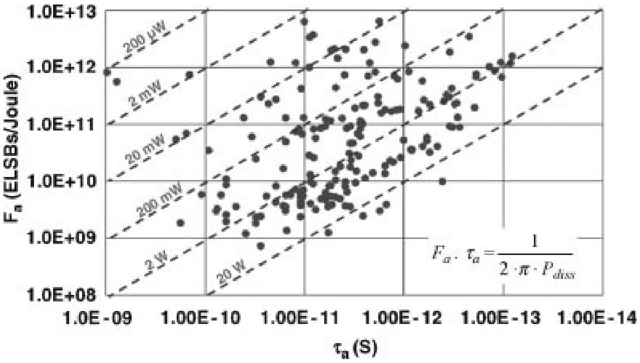


Figure 4. The relationship of F_a and τ_a for the ADCs studied herein. Nearly six orders of magnitude in ADC power dissipation are evident.

Table 1. A list of high-performance ADCs sorted first by aperture uncertainty, t_a, and secondarily by IEEE Figure of Merit, FOMa, Column 1 indexes the converters

Index	Institution	Author/ Part No.	Year	technology	archit	fsamp Hz	fsig Hz	bits	SNDR	ENOB	ta	SFDR	SFDRbits	Vin	BWin Hz	Pdis W	F	Fa	FOMa
1	Linear Technology	LTC2208	2006	CMOS	pipelined+ dig error corr	1.30E+08	2.50E+08	16	73.6	11.9	8.14E-14	78.0	13.0	2.25	7.00E+08	1.250	4.07E+11	1.56E+12	6.39E
2	Analog Devices	AD9460	2006	CMOS	pipelined	1.05E+08	1.70E+08	16	76.6	12.4	8.47E-14	83.0	13.8	3.40	8.00E+08	1.600	3.63E+11	1.17E+12	8.52E
3	Analog Devices	AD9446	2005	CMOS	pipelined	1.00E+08	1.25E+08	16	77.5	12.6	1.04E-13	82.0	13.7	3.20	2.25E+08	2.300	2.66E+11	5.46E+08	1.50E
4	Linear Technology	LTC2208	2006	CMOS	pipelined+ dig error corr	1.30E+08	1.40E+08	16	76.4	12.4	1.05E-13	85.0	14.2	2.25	7.00E+08	1.250	5.62E+11	1.21E+12	8.27E
5	Analog Devices	AD6645	2004	Cbipolar	3-stage. dig error corr:	1.05E+08	2.00E+08	14	72.0	11.7	1.22E-13	63.0	10.5	2.20	2.70E+08	1.500	2.28E+11	8.68E+11	1.15E
6	Texas Instruments	ADS5546	2006	CMOS	pipeline 1B/ stage SHA DSP	1.90E+08	3.00E+08	14	67.4	10.9	1.38E-13	72.0	12.0	2.00	5.00E+08	1.130	3.22E+11	1.02E+12	9.83E
7	Analog Devices	AD9640	2006	CMOS	dual SHA+ pipelined+DSP	1.50E+08	1.70E+08	14	69.8	11.3	1.85E-13	80.0	13.3	1.00	6.50E+08	0.780	4.86E+11	1.10E+12	9.08E
8	Texas Instruments	ADS5546	2006	CMOS	pipeline 1B/ stage SHA. DSP	1.90E+08	1.50E+08	14	70.8	11.5	1.87E-13	80.0	13.3	2.00	5.00E+08	1.130	4.76E+11	7.52E+11	1.33E
9	Linear Technology	LTC2208	2006	CMOS	pipelined+ DSP	1.30E+08	7.00E+07	16	77.4	12.6	1.88E-13	90.0	15.0	2.25	7.00E+08	1.250	6.30E+11	6.79E+11	1.47E
10	Atmel	AT84AS008	2005	npn Bipolar	flash, dig error corr	2.20E+09	2.00E+09	10	48.0	7.7	1.94E-13	55.0	9.2	0.50	3.30E+09	4.200	1.07E+11	1.95E+11	5.12E
11	Linear Technology	LTC2294	2005	CMOS	pipelined	8.00E+07	1.40E+08	12	70.0	11.3	2.20E-13	85.0	14.2	2.00	5.75E+08	0.211	9.80E+11	3.43E+12	2.92E
12	Texas Instruments	ADS5463	2006	BiCMOS	pipelined	5.00E+08	2.30E+08	12	64.7	10.5	2.46E-13	78.0	13.0	2.20	1.30E+09	2.200	3.19E+11	2.94E+11	3.41E
13	Atmel	AT84AS004	2005	npn Bipolar	flash, dig error corr	2.00E+09	1.00E+09	10	51.0	8.2	2.75E-13	55.0	9.2	0.50	3.00E+09	6.500	8.92E+10	8.92E+10	1.12E
14	Atmel	AT84AS008	2005	npn Bipolar	flash, dig error corr	1.70E+09	8.50E+08	10	52.0	8.3	2.88E-13	56.0	9.3	0.50	3.30E+09	4.200	1.32E+11	1.32E+11	7.60E
15	Analog Devices	AD9445	2005	CMOS	pipelined	1.25E+08	7.00E+07	14	73.5	11.9	2.94E-13	85.0	14.2	3.20	3.00E+09	2.300	2.10E+11	2.35E+11	4.25E
16	Atmel	AT84AS001	2006	npn Bipolar	SHA+ pipeline	5.00E+08	2.50E+08	12	62.0	10.0	3.09E-13	72.0	12.0	1.10	1.50E+09	2.300	2.24E+11	2.24E+11	4.47E
17	Rockwell Scientific	RAD008	2006	GaAs HBT	folding, interpolating	3.00E+09	1.50E+09	8	46.0	7.3	3.25E-13	55.0	9.2	2.00	1.00E+10	5.500	8.89E+10	8.89E+10	1.12E
18	Texas Instruments	ADS5546	2006	CMOS	pipelined 1B/ state SHA DSP	1.90E+08	7.00E+07	14	72.5	11.8	3.30E-13	84.0	14.0	2.00	5.00E+08	1.130	5.79E+11	4.27E+11	2.34E
19	Teranetics	S. Gupta et al. (5)	2006	0.13 um CMOS	time-interleaved	1.00E+09	5.00E+08	11	55.0	8.8	3.46E-13	53.1	8.9		5.00E+08	0.250	1.84E+12	1.84E+12	5.44E
20	Rockwell Scientific	RAD010	2006	GaAs HBT	2-stage	1.00E+09	5.00E+08	10	55.0	8.8	3.46E-13	60.0	10.0	1.00	6.00E+09	5.000	9.19E+10	9.19E+10	1.09E
21	Analog Devices	AD6645	2004	Cbipolar	3-stage	1.05E+08	5.25E+07	14	74.5	12.1	3.49E-13	93.0	15.5	2.20	2.70E+08	1.500	3.04E+11	3.04E+11	3.29E
22	Analog Devices	AD9430	2005	BiCMOS	pipelined	2.10E+08	2.40E+08	12	60.0	9.8	3.72E-13	63.0	10.5	1.50	7.00E+08	1.300	1.44E+11	3.29E+11	3.04E
23	Analog Devices	AD9460	2006	CMOS	pipelined	1.05E+08	3.00E+07	16	78.3	12.7	3.95E-13	90.0	15.0	3.40	8.00E+08	1.600	4.41E+11	2.52E+11	3.97E
24	Hewlett Packard	Schiller and Byrne (6)	1991	Bipolar hybrid	time interleaved	4.00E+09	2.00E+09	8	41.5	6.6	4.10E-13					39.000	9.96E+09	9.96E+09	1.00E
25	Analog Devices	AD9640	2006	CMOS	dual, SHA+ pipelined+DSP	1.50E+08	7.00E+07	14	70.6	11.4	4.11E-13	85.0	14.2	1.00	6.50E+08	0.780	5.33E+11	4.97E+11	2.01E
26	Analog Devices	AD12401	2005	Hybird	parallel, V-Corp, DSP	4.00E+08	1.28E+08	12	64.4	10.4	4.58E-13	75.0	12.5	3.20	1.80E+08	8.500	6.38E+10	4.08E+10	2.45E
27	Agilent Labs	Poulton et al. (7)	2003	0.18 um CMOS	time-interleaved	2.00E+10	6.00E+09	8	29.5	4.6	5.47E-13			0.25	6.60E+09	9.000	5.39E+10	3.23E+10	3.09E
28	Analog Devices	AD9430	2005	BiCMOS	pipelined	2.10E+08	1.00E+08	12	64.5	10.5	5.50E-13	77.0	12.8	1.50	7.00E+08	1.300	2.34E+11	2.23E+11	4.49E

Table 1. (*continued*)

Index	Institution	Author/ Part No.	Year	technology	archit	fsamp Hz	fsig Hz	bits	SNDR	ENOB	ta	SFDR	SFDRbits	Vin	BWin Hz	Pdis W	F	Fa	FOMa
29	Atmel	TS83102	2004	npn Bipolar	flash, dig error corr	1.40E+09	7.00E+08	10	47.5	7.6	5.87E-13	59.0	9.8	0.50	3.30E+09	4.600	5.90E+10	5.90E+10	1.70E
30	Maxim Integrated Prod.	Max 108	2004	Bipolar	flash	1.50E+09	7.50E+08	8	46.9	7.5	5.87E-13	54.1	9.0	0.50	2.20E+09	5.250	5.17E+10	5.17E+10	1.94E
31	National	ADC08D1500	2005	CMOS	folding, interpolating	1.50E+09	7.50E+08	8	46.3	7.4	6.29E-13	53.0	8.8	0.87	1.70E+09	0.950	2.66E+11	2.66E+11	3.75E
32	Rockwell Scientific	RAD006	2006	GaAs HBT	folding, interpolating	6.00E+09	3.00E+08	6	34.0	5.4	6.48E-13	40.0	6.7	2.00	1.00E+10	6.000	4.09E+10	4.09E+10	2.44E
33	Analog Devices	AD9230	2006	CMOS	pipeline, SHA. DSP	2.50E+08	1.00E+08	12	63.5	10.3	6.51E-13	77.0	12.8	1.25	9.00E+08	0.425	7.19E+11	5.75E+11	1.74E
34	Analog Devices	AD6644	2004	Cbipolar	3-stage, dig error corr	6.50E+07	3.05E+08	14	73.0	11.8	7.15E-13	90.0	15.0	2.20	2.50E+07	1.300	1.83E+11	1.71E+11	5.8E
35	Rockwell	Nary et al. (8)	1995	GaAs HBT	folded flash	2.00E+09	1.00E+08	8	41.4	6.6	8.29E-13	48.0	8.0	0.64	3.00E+09	5.300	3.62E+10	3.62E+10	2.76E
36	TelAsic	TC1200	2002	SiGe	folded flash	1.00E+09	4.00E+08	10	49.3	7.9	8.35E-13				4.00E+08	5.500	4.33E+10	3.47E+10	2.88E
37	Maxim Integrated Prod.	Max 104	1999	Bipolar	flash	1.00E+09	5.00E+08	8	47.2	7.5	8.50E-13	52.3	8.7	0.50	2.20E+09	5.250	3.57E+10	3.57E+10	2.81E
38	Linear Technology	LTC2294	2005	CMOS	pipelined	8.00E+07	4.00E+08	12	68.7	11.1	8.94E-13	90.0	15.0	2.00	5.75E+08	0.211	8.44E+11	8.44E+11	1.19E
39	Xignal Techn	Mitteregger et al. (9)	2006	0.18 um CMOS	delta-sigma 3rd order CT BP, 48 Q fc	4.00E+07	2.00E+08	14	74.0	12.0	9.71E-13	78.0	13.0	1.00	2.00E+07	0.069	2.37E+12	2.37E+12	4.21E
40	Nortel	P Schvan et al. (10)	2006	0.13 um SiGe BiCMOS	flash	2.20E+10	7.00E+08	5	22.8	3.5	1.00E-12	29.0	4.8	0.64		3.000	8.30E+10	5.28E+10	1.89E
41	HP & Rockwell	Poulton(HP), (14) Wang(R)	1994	GaAs HBT	flash	4.00E+09	2.00E+08	6	33.1	5.2	1.08E-12	33.7	5.6		1.80E+09	5.700	2.59E+10	2.59E+10	3.86E
42	Philips, Netherl	van Valburg, et al. (11)	1992	Bipolar	folding, interpolating	6.50E+08	3.25E+08	8	48.8	7.8	1.09E-12	51.0	8.5	2.00	1.50E+08	0.810	1.81E+11	1.81E+11	5.54E
43	Infineon	P Bogner et al. (12)	2006	0.13 um Bipolar	pipelined multibit per stage wi DSP cal	1.00E+08	4.00E+08	14	66.5	10.8	1.15E-12	69.0	11.5	1.50	4.00E+078	0.224	7.71E+11	6.17E+11	1.62E
44	Lucent Technologies	CSP1152A	1998	CMOS	dithered	6.50E+07	3.25E+08	14	68.0	11.0	1.19E-12	85.0	14.2	1.60	1.00E+09	0.750	1.78E+11	1.78E+11	5.62E
45	Hewlett Packard	Jewett et al. (13)	1997	npn Bipolar	2-stage, folding	1.28E+08	6.40E+08	12	61.5	9.9	1.28E-12	70.0	11.7	0.50	2.50E+08	5.700	2.18E+10	2.18E+10	4.59E
46	Analog Devices	AD6640	1997	Cbipolar	2-stage, dig error corr	6.50E+07	3.25E+08	12	67.0	10.8	1.34E-12	80.0	13.3	2.00	2.50E+07	0.710	1.67E+11	1.67E+11	5.97E
47	Rockwell	RSC-ADC080S	1998	GaAs HBT	folded flash	2.00E+09	1.00E+08	8	37.0	5.9	1.38E-12	43.0	7.2	0.80	5.00E+08	5.000	2.31E+10	2.31E+10	4.32E

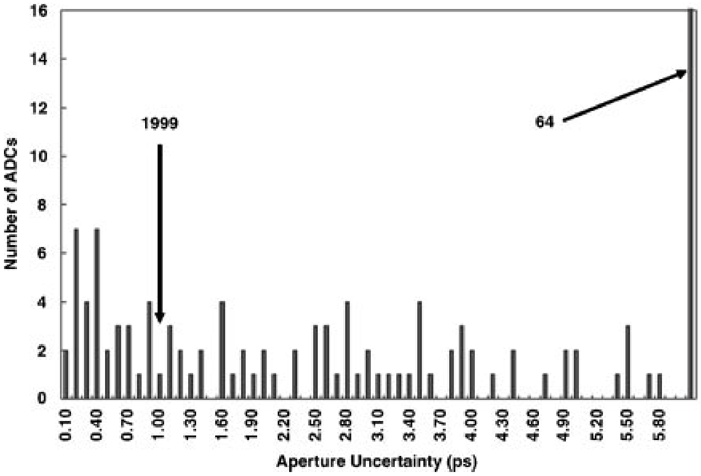


Figure 5. Histogram for aperture jitter. The relatively large number of ADCs situated to the left of the arrow (smallest jitter values) represents the post-1999 activity in the development of high-performance converters.

is dominated by pipelined and multistage flash architectures. Also, time-interleaved, folded flash, and flash ADCs are presented. The two converters at the top of the list, LTC2208 and AD9460, are pipelined and have jitter values of ~80–85 fs, which is an order of magnitude below the best of 1999.

There are eight multiple entries: LTC2208 (#1, #4, #9), AD9460 (#2, #23), AD6645 (#5, #21), ADS5546 (#6, #8, #18), AD9640 (#7, #25), AT84AS008 (#10, #14), LTC2294 (#11, #38), AD9430 (#22, #28). Each of these ADCs has one entry for Nyquist operation and one or more for undersampling mode. Thirty-seven separate converters are represented in the table.

Of the ADC architectures listed in Table 1, flash [# 10, 13, 14, 29, 30, 37, 40 (10), 41, (14), (15)], a parallel technique, is the fastest. It uses 2^N-1 comparators, where N is the stated resolution, but, often includes one or two additional comparators to measure overflow conditions. All comparators sample the analog input voltage simultaneously; hence, it is inherently fast.

The parallelism of the flash architecture has drawbacks for high-resolution applications as the number of comparators grows exponentially with N. In addition, the separation of adjacent reference voltages grows smaller exponentially. Consequently, for large N, this architecture would require very large, power-hungry ICs, and it is difficult to match components in the parallel comparator channels. Finally, increasingly large input capacitance reduces analog input bandwidth. Most flash converters available today have $N \leq 8$. These problems are overcome by using variations on the flash architecture that use relatively few comparators yet retain good speed albeit at the expense of increased latency. They are the pipelined/multistage flash [# 1–9, 11, 12, 15, 16, 20, 21, 22, 28, 33, 34, 43(12), 46], and folded-flash [# 17, 31, 32, 35, 36, 42, 45(13), 47] architectures.

Parallel configurations such as time-interleaved and filter bank offer other ways to attain high-speed conversion. The parallel ADCs listed in Table 1 [# 19(5), 24(6), 26, 27(7)], achieve < 1.5 ps aperture jitter, but some require substantial power, as much as an order of magnitude larger than single-chip converters [e.g., # 24(6)].

Sixteen of the ADCs in Table 1 (#1, 5, 6, 7, 8, 10, 12, 13, 14, 18, 25, 26, 29, 33, 34, 43 and 46) employ some type of on-chip digital signal processing (DSP). All but one of these were introduced after 2003, which implies that increased IC complexity (Moore's Law) is at least partially responsible for the addition of DSP for error correction.

An architecture that trades speed for resolution is delta-sigma ($\Delta\Sigma$) modulation (17); when combined with digital decimation filtering, it forms a complete ADC. For band-pass RF applications, a small geometry Si process [# 39(9)] is beneficial. This example is another instance of the impact of Moore's Law on ADC design and fabrication. On the other hand, III-V technologies (15,18,19) that are intrinsically high bandwidth are still competitors for at least the $\Delta\Sigma$ loop portion of these converters. When IC technologies mature, which include heterogeneous integration (20–22), a single-chip $\Delta\Sigma$ ADC that employs InP and CMOS may eventually be realized.[1] Such techniques may also be applicable to other ADC architectures as well.

PERFORMANCE TRENDS AND PROJECTIONS

Distributions of ADCs as functions of jitter (Fig. 5) and of FOM_a (Fig. 6) show how converter performance and efficiency have progressed over the past 7 years. In Fig. 5, only three ADCs that were announced prior to 2000 (compared to 31 after 2000) had $\tau_a < 1$ ps (6,8), (Table 1 #35). Similarly, in Fig. 6, only two of the ADCs demonstrated prior to 2000 (compared with 32 after 2000) had FOM_a values < 2.5 pJ/

[1] Many heterogeneous integration efforts are aimed at adding photonic capabilities to Si-based circuits, but there is no reason why the addition of very high-speed electronic circuits (e.g., InP) could not be used with high-density Si circuitry as well.

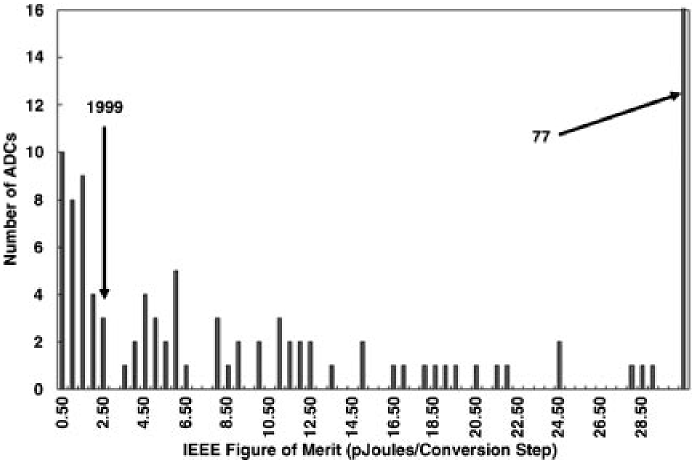


Figure 6. Histogram for IEEE Figure of Merit. The relatively large number of ADCs situated to the left of the arrow (smallest FOMa values) represents the post-1999 activity in the development of high-efficiency converters.

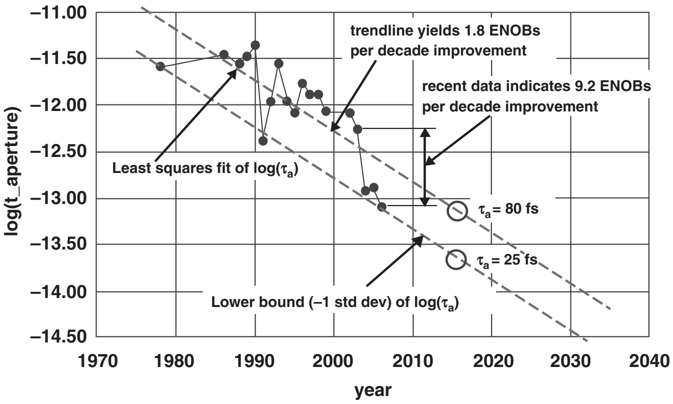


Figure 7. Derived aperture jitter [Equation (3)] for the best ADC performances as a function of the year of introduction. Converter performances are improving, gradually although actual progress is sporadic.

ELSB (23,24). The upsurge in quality of demonstrated converter designs is self-evident.

Another way of probing the data[2] in Figs. 5 and 6 is to examine the performance Figure of Merit (aperture jitter), τ_a, for ADCs over the past three decades (especially the past 4 years). This is done in Fig. 7 where the $\log_{10}$ of the best (lowest) value on an annual basis is graphed. The overall trend over the past 30 years is an improvement of 1.8 ENOB per decade. However, the progress over the past 4 years would correspond to an astounding 9.2 ENOB per decade. Like the similar jump in the early 1990s, this increase is most likely not a sustainable trend. However, at face value it shows an order of magnitude improvement in less than 4 years (not including design and development time). A more-or-less simultaneous occurrence of factors has contributed to this relatively sudden improvement: (*1*) designs that focus on lowering aperture jitter and increasing analog input bandwidth; (*2*) progress in ADC IC technology which includes the addition of on-chip DSP; and (*3*) improved quality of ADC testing.

If the lower trend line in Fig. 7 is used, then a somewhat conservative projection indicates that a jitter value of 25 fs may be reached in the year 2015 or so. This estimation is equivalent to achieving, say, 12 ENOB at an 800 MHz input frequency or 10 ENOB at an input frequency of 3 GHz. The aperture jitter line labeled 2015 in Fig. 8 indicates the possibilities for 25 fs jitter, whereas the comparator ambiguity curve labeled 2015 indicates the limitations imposed by $f_T = 500$ GHz. At lower frequencies (10–100 MHz), thermal noise will limit converter performance, whereas at 10–12 GHz, comparator ambiguity is a limiting factor. Given that f_T, which scales comparator ambiguity, has increased by a factor of three in roughly 7 years for the fastest IC technologies, it is reasonable to assume that

[2] A complete listing of the ADCs discussed in this article is given in the Appendix 1.

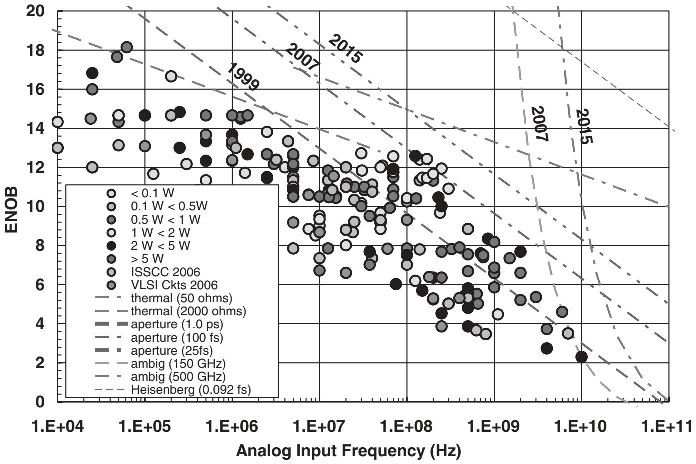


Figure 8. ENOB versus analog input frequency (same data as in Fig. 1a). Here, the ADCs are differentiated according to power dissipation. In addition, converters presented at two conferences (ISSCC, VLSI Circuits Symposium) held in 2006 are indicated. The intersection of the jitter and ambiguity curves, each labeled 2007, indicates 8 ENOB at 3 GHz, close to the current state-of-the-art. The 2015 intersection implies 8 ENOB at 10 GHz.

another factor of about three will come about by the year 2015. Hence in 9–10 years, the state-of-the-art may be defined by the boundaries of the 25 fs jitter line and the 500 GHz ambiguity curve. The intersection of these two curves implies 8 ENOB at 10 GHz.

The data points in Fig. 8 are the same as in Fig. 1a; here, they are sorted by power dissipation. Converters presented at two conferences held in 2006 (ISSCC, VLSI Circuits Symposium) are also delineated.

An examination of how converter performance has progressed with respect to achieving low-power dissipation is shown in Fig. 9. The trend seems to be at the rate of about -1 decade per decade, and the most efficient ADCs have achieved just above 100 fs per conversion step. Using the lower trend line (-1 standard deviation), one might anticipate 10 fs per conversion step by the year 2015. However, the simultaneous achievement of very low τ_a and FOM_a is not necessarily imminent, as implied in Fig. 10. As an example, a design target of 8 ENOB at 3 GHz input frequency (current performance capability) with a dissipation of 0.5 W corresponds to τ_a $\sim$100 fs and $\mathrm{FOM}_a \sim 0.3$ pJ per conversion step. This is beyond the current state-of-the-art.

To improve the present state-of-the-art in ADC performance, significant technical challenges must be met. Specifically, (*1*) a reduction in aperture uncertainty to well below 100 fs, (*2*) an increase in the comparator ambiguity limit to beyond 15 GHz ($f_T > 500$ GHz), and (*3*) accomplishing both (*1*) and (*2*) while maintaining low power consumption (e.g., < 0.5 W). The attainment of these characteristics may involve employment of such technologies as heterogeneous integration (20), photonic clock generation, and distribution (25), and perhaps superconducting circuitry (26) where appropriate. The continued advance of electronic ADC IC techologies is also anticipated.

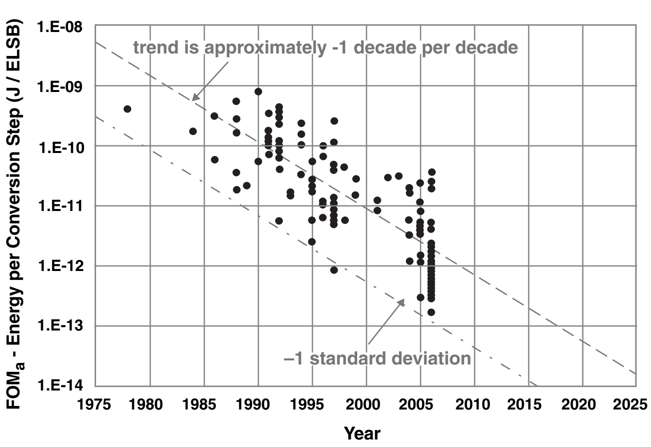


Figure 9. The evolution of the IEEE Figure of Merit for the ADCs studied herein. The rate of decline is approximately -1 decade per decade. If the current trend continues, then converters that operate at 10 fs per conversion step may be realized by the year 2015.

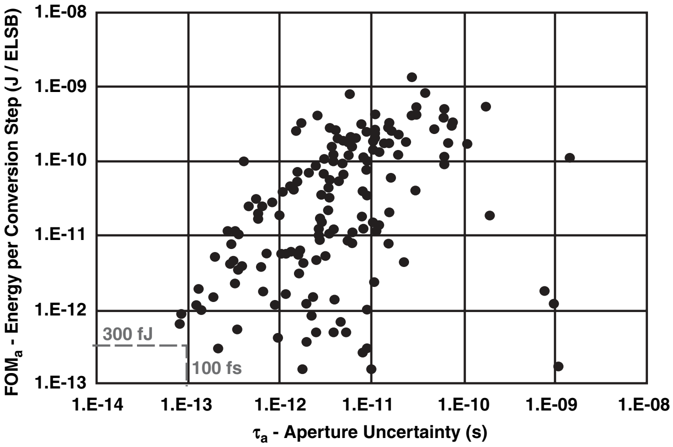


Figure 10. FOM_a versus τ_a for the ADCs studied in this work. The simultaneous achievement of very low τ_a (< 100 fs) and very low FOM_a (< 300 fs per conversion step) seems to be on the horizon, but it has not yet occurred.

Appendix 1. Table of ADCs covered in this work

Institution	Author/Part No.	Year	archit	fsamp Hz	fsig Hz	bits	SNDR	ENOB	τa	SFDR	SFDR bits	Vin	BWin Hz	Pdis W	F	Fa	FOMa
MIT	Venna et al.	2006	SA	1.00E+05	5.00E+04	12	65.0	10.50	1.10E-09	71.0	11.83			2.50E+05	5.81E+12	5.81E+12	1.72E-13
Univ Nova de Lisboa	J Goes et al.	2006	delta-sigma 2nd order	2.00E+04	1.00E+04		80.0	10.00	9.74E-10	83.0	13.83		1.00E+04	2.00E+04	8.17E+11	8.17E+11	1.22E-12
Chinese University of Hong Kong	K-P Pun	2006	delta-sigma 3rd order CT, 1B Q	5.00E+04	2.50E+04		74.0	12.00	7.77E-10			0.5	2.50E+04	3.70E+04	5.54E+11	5.54E+11	1.61E-12
MIT	T Sepke et al.	2006	pipelined 1 5b/stage	8.00E+06	4.00E+06	10		-0.29					4.00E+06	2.50E+03			
IMEC Leuven	G. Van der Plan et al.	2006	flash	1.25E+09	8.25E+08	4	23.8	3.65	1.01E-11			0.2	3.30E+09	2.50E+03	8.33E+12	8.33E+12	1.58E-13
UCB	S-W Chen et al.	2006	time-interleaved SAR	6.00E+08	3.00E+08	6	32.0	5.02	8.16E-12				4.00E+09	5.30E+03	3.68E+12	3.68E+12	2.72E-13
Unversity of Texas	K-S Lee et al.	2006	delta-sigma 2nd order, 2-channel, time interleaved	2.20E+06	1.10E+06		80.0	13.00	8.85E-12	85.0	14.17	1.6	1.10E+06	5.40E+03	3.33E+12	3.33E+12	3.00E-13
Texas Instruments (Burr-Brown)	ADSB3.26	2007	successive approximation register	2.50E+05	1.00E+04	16	88.0	14.33	3.88E-10	95.0	15.83	5	5.00E+05	1.00E+02	5.13E+11	4.11E+10	2.44E-11
Swiss Federal Institute	Hammerschmied, Huang	1997	SA	2.00E+05	1.00E+06	10	56.5	9.09	1.46E-09	79.0	13.17			1.20E+02	9.10E+09	9.10E+09	1.10E-10
University of Texas	Kwon et al.	2006	delta-sigma 2nd order, 4B Q	8.73E+06	4.36E+06		82.0	13.33	1.77E-12	86.0	14.33		2.20E+06	1.40E+02	6.41E+12	6.41E+12	1.56E-13
UC Berkeley	Suterja, et al.	1988	pipelined	2.50E+05	1.25E+05	13	72.0	11.67	1.96E-10	80.0	13.33			1.50E+02	5.42E+10	5.42E+10	1.84E-11
Samsung	H-C Chol et al.	2006	pipelined	5.00E+07	2.50E+07	10	57.2	9.21	5.38E-12	72.9	12.15	2	2.50E+07	1.50E+02	1.92E+12	1.97E+12	5.07E-13
Linear Technology	2351-14	2007	parallel-MUM-serial out	2.50E+05	3.00E+05	14	75.0	12.17	5.77E-11	90.0	16.00	2.5	5.00E+07	1.65E+02	6.96E+10	1.62E+11	5.98E-12
Linear Technology	2356-14	2007	serial out	3.50E+06	1.40E+06	14	72.3	11.72	1.69E-11	82.0	13.67	2.5	5.00E+07	1.80E+02	6.55E+11	5.24E+11	1.91E-12
UCSD	S-T Ryu	2006	pipelined	5.00E+07	2.00E+07	10	54.7	8.80	8.93E-12	69.0	11.50		2.00E+07	1.80E+02	1.24E+12	9.90E+11	1.01E-12
Analog Devices	AD7690	2006	SA TH	4.00E+05	2.00E+05	18	102.0	16.65	3.87E-12	108.0	18.00		2.00E+05	2.00E+02	2.06E+12	2.06E+12	4.86E-13
SHARP, Japan	Y Fujimoto et al.	2006	delta-sigma 4th order CS 4B O tc=	6.40E+06	3.20E+06		76.3	12.38	4.66E-12	78.5	13.08		3.20E+06	2.38E+02	1.44E+12	1.44E+12	6.96E-13
Metaushita	Kusumota, et al.	1993	pipelined, interpolating	1.50E+07	7.50E+08	10	65.0	8.84	2.81E-11				3.00E+06	3.00E+02	2.30E+11	2.30E+11	4.35E-12
Sony	Y Shirmzy	2006	2-stage flash	4.00E+07	2.00E+07	12	68.0	11.00	1.94E-12	72.5	12.12		2.00E+07	3.00E+02	2.73E+12	2.73E+12	3.65E-13
Philips Eindhoven	G Geelan et al.	2006	pipelined	1.00E+08	5.00E+07	10	57.7	9.30	2.52E-12	65.0	10.83		1.00E+08	3.13E+02	2.02E+12	2.02E+12	4.96E-13
Univ of Cal. Berk.	Cho & Gray	1998	pipeline	2.00E+07	1.00E+07	10	59.1	9.52	1.08E-11			2		3.50E+02	4.21E+11	4.21E+11	2.08E-12
Chipidea, Portugal	P Frgueirede et al.	2006	2-stage flash	1.00E+09	5.02E+08	6	30.0	5.32	3.96E-12	42.0	7.00		5.00E+08	5.50E+02	7.27E+11	7.30E+11	1.37E-12
Univ of Illinais, Harris	Kwak et al.	1997	4-stage cal	5.00E+06	2.50E+06	15	84.9	13.81	2.22E-12	93.0	15.50			6.00E+02	1.20E+12	1.20E+12	8.35E-13
Xignal Techno	Mitteregger et al.	2008	delta-sigma 3rd order CT BP 4B Q, tc=640 MHz	4.00E+07	2.00E+07	14	74.0	12.00	9.71E-13	78.0	10.00	1	2.00E+07	6.90E+02	2.37E+12	2.37E+12	4.21E-13
Xignal Techn	XT11401	2006	delta-sigma 3rd order CT BP 4B Q, tc=640 MHz	2.00E+07	4.00E+06	14	74.0	12.00	4.08E-12	76.0	12.67	4	2.00E+07	7.50E+02	1.09E+12	4.37E+11	2.20E-12
Analog Devices	AD9200	1997	4-stage flash, digital corr	2.00E+07	1.00E+07	10	58.0	9.01	1.54E-11	62.0	10.33	2	9.00E+07	8.00E+02	1.29E+11	1.29E+11	7.76E-12
Nordic Semiconductor	nAD12110-18a	2005	pipeline multi-stage wi error corr	8.00E+07	3.00E+07	12	63.0	10.17	2.30E-12	68.0	11.00	1.5	8.00E+08	9.90E+02	9.33E+11	7.00E+11	1.43E-12
Nordic Semiconductor	nAD12110-18a	2005	pipeline multi-stage wi error corr	8.00E+07	5.00E+07	12	60.0	9.67	1.95E-12	64.0	10.67	1.5	6.00E+08	9.90E+02	6.60E+11	8.25E+11	1.21E-12
UC Berkeley	Cline & Gray	1996	pipelined self-cal	5.00E+06	2.50E+06	13	70.7	11.45	1.14E-11	73.0	12.17	6.6	5.00E+07	1.66E+01	8.44E+10	8.44E+10	1.19E-11
Analog Devices	AD9057-80	1997	flash	8.00E+07	4.00E+07	8	46.0	7.35	1.22E-11	55.0	9.17	2.5	1.20E+08	1.75E+01	7.45E+10	7.45E+10	1.34E-11
Burr-Brown	PCM1750	1992	SA	2.00E+05	1.00E+05	18	90.0	14.66	3.08E-11	90.0	15.00	5.5	5.00E+05	2.10E+01	2.46E+10	2.46E+10	4.06E-11
Delft University	Silva et al.	2006	delta-sigma 5th order CT BP, 1B Q	4.00E+05	2.00E+05		90.0	14.66	1.54E-11	98.0	16.33	0.5	2.00E+05	2.10E+01	4.92E+10	4.92E+10	2.03E-11
National	ADC08200	2005	2-stage + DSP	2.00E+08	1.00E+08	8	44.0	7.02	6.15E-12	54.0	9.00	1.6	5.00E+08	2.10E+01	1.23E+11	1.23E+11	8.11E-12
Linear Technology	LTC-2294	2005	pipelined	8.00E+07	4.00E+07	12	68.7	11.12	8.94E-13	90.0	15.00	2	5.75E+08	2.11E+01	8.44E+11	8.44E+11	1.19E-12
Linear Technology	LTC-2294	2005	pipelined	8.00E+07	1.40E+08	12	78.0	11.34	2.20E-13	85.0	14.17	2	5.75E+08	2.11E+01	9.80E+11	3.43E+12	2.92E-13
Infineon	P Bogner et al.	2006	pipelined multibit per stage wi DSP cal	1.00E+08	4.00E+07	14	66.5	10.75	1.15E-12	69.0	11.50	1.5	4.00E+07	2.24E+01	7.71E+11	6.17E+11	1.62E-12
Analog Devices	Mehr & Daiton	1999	flash	4.00E+08	2.00E+08	6	33.3	5.24	1.05E-11	38.0	6.33	1	2.00E+08	2.25E+01	6.71E+10	6.71E+10	1.49E-11
Analog Devices	AD1876		ch. redis	1.00E+05	5.00E+04	16	90.0	14.66	6.16E-11	99.0	16.50	10	1.00E+06	2.35E+01	1.10E+10	1.10E+10	9.09E-11
Broadcom Corp, UCLA	Bult, et al.	1997	averaging, folding	4.80E+07	2.40E+07	10	54.0	8.68	8.10E-12			2	3.20E+07	2.40E+01	8.19E+10	8.19E+10	1.22E-11
Analog Devices	AD7886		3-stage	8.00E+05	4.00E+05	12	67.0	10.84	1.09E-10	77.0	12.83	5	1.00E+06	2.50E+01	5.85E+09	5.85E+09	1.71E-10
Analog Devices	AD9220	1997	4-stage flash, digital corr	1.00E+07	5.00E+06	12	71.0	11.50	5.49E-12	88.0	14.67		6.00E+07	2.50E+01	1.16E+11	1.16E+11	6.62E-12

Teranetics	S Gupta et al.	2006	time-interleaved	1.00E+09	5.00E+08	11	55.0	8.84	3.46E-13	53.1	8.85		5.00E+08	2.50E+01	1.84E+12	1.84E+12	5.44E-13
UCSD	S. Ray et al.	2006	pipelined	4.30E+07	1.40E+07	13	67.0	10.84	3.11E-12	73.0	12.17	1.6	1.40E+07	2.68E+01	2.94E+11	1.91E+11	5.23E-12
Kenet	KAD2208L	2007	pipelined	2.75E+08	1.40E+08	8	48.7	7.80	2.56E-12	68.0	11.33	1.5	6.00E+08	2.75E+01	2.22E+11	2.26E+11	4.42E-12
Analog Devices	AD9220	1996	3-stage	1.00E+07	5.00E+06	12	70.0	11.34	6.16E-12	77.5	12.92		3.50E+07	2.80E+01	9.23E+10	9.23E+10	1.08E-11
Analog Devices	AD9240	1997	4-stage flash, digital corr	1.00E+07	5.00E+06	14	77.5	12.58	2.60E-12	90.0	15.00		7.00E+07	2.85E+01	2.15E+11	2.15E+11	4.65E-12
Motorola	DSP56ADC16	1992	delta-sigma, OSR 64, DSP	1.00E+05	5.00E+04	16	90.0	14.66	6.16E-11	88.0	14.67	3.5	5.00E+04	3.00E+01	8.62E+09	8.62E+09	1.16E-10
Analog Devices	AD9225	1997	4-stage flash, digital corr	2.50E+07	1.25E+07	12	70.0	11.34	2.46E-12	86.0	14.33		2.00E+08	3.00E+01	2.15E+11	2.15E+11	4.64E-12
Philips Components, FR	Vorenkamp, Roovers	1997	folding interp	5.00E+07	2.50E+07	12	64.0	10.34	2.46E-12	71.0	11.83		2.60E+05	3.00E+01	2.16E+11	2.16E+11	4.63E-12
Philips Semiconductors	G Geelan	2001	flash/interpolating/averaging	9.00E+08	4.50E+08	6	34.3	5.41	4.17E-12			1.5	4.50E+08	3.00E+01	1.27E+11	1.27E+11	7.87E-12
Univ. of Illinois	Shu et al.	1995	pipelined, ds corr	1.00E+07	5.00E+06	13	68.0	11.00	7.75E-12					3.60E+01	5.70E+10	5.70E+10	1.75E-11
Analog Devices	Schreier et al.	2006	delta-sigma 4th order CT BP, 48 Q fc=44 MHz	1.70E+07	8.50E+06		76.0	12.33	1.82E-12	90.0	1500		8.50E+06	3.75E+01	2.34E+11	2.34E+11	4.28E-12
Analog Devices	AD9058	2003	7b flash with 8th bit interpolated	4.00E+07	1.00E+07	8	46.0	7.35	4.88E-11	58.0	9.67	2	1.75E+08	3.85E+01	1.69E+10	8.47E+09	1.10E-10
Hughes	ACTC	1984		1.00E+07	5.00E+06	8	49.0	7.85	6.91E-11	61.0	10.17			4.00E+01	5.76E+09	5.76E+09	1.74E-10
Analog Devices	AD9230	2006	pipeline, SHA DSP	2.50E+08	1.00E+08	12	63.5	10.26	6.51E-13	77.0	12.83	1.25	9.00E+08	4.25E+01	7.19E+11	5.75E+11	1.74E-12
Analog Devices	Fernandes, et al.	1988	pipeline	1.00E+05	5.00E+04	14	80.8	13.13	1.78E-10	90.0	15.00	10	6.00E+05	4.80E+01	1.87E+09	1.87E+09	5.36E-10
Analog Devices	AD3054-200	1987	flash	2.00E+09	1.00E+08	8	49.0	7.85	3.48E-12				3.80E+08	6.00E+01	3.21E+10	8.21E+10	1.09E-11
UCLA EE Dept	Choi & Abidi	2001	flash array averaginng, T/H	1.00E+09	6.30E+08	6	35.0	5.52	2.75E-12	44.0	7.33	1.60	6.30E+08	5.00E+01	9.19E+10	1.16E+11	8.64E-12
UCLA EE Dept	Choi & Abidi	2001	flash array averaging, T/H	1.30E+09	6.50E+08	6	32.0	5.02	3.76E-12	44.0	7.33	1.60	6.50E+08	5.00E+01	8.46E+10	8.46E+10	1.18E-11
TI/Burr-Brown	ADS1274/ADS1273	2007	delta-sigma	1.28E+05	6.20E+04	24	111.0	18.15	4.42E-12	109.0	18.17	5.00	6.20E+04	6.00E+01	7.01E+10	8.79E+10	1.47E-11
Analog Devices	Brooks, et al.	1997	delta-sigma OSR & DSP	2.50E+06	1.25E+06	16	89.0	14.49	2.76E-12	97.1	18.18			5.50E+01	1.05E+11	1.05E+11	9.56E-12
Analog Devices	Murden & Geaser	1996	2-stage	5.00E+07	2.60E+07	12	68.0	11.00	1.65E-12	80.0	13.33	1.00	2.00E+08	6.75E+01	1.78E+11	1.78E+11	6.60E-12
Sony	CXA1386P		flash	7.50E+07	3.75E+07	8	44.0	7.02	1.64E-11				1.50E+08	5.80E+01	1.67E+10	1.67E+10	5.92E-11
Analog Devices	AD9042	1996	2-stage	4.10E+07	2.05E+07	12	69.0	11.17	1.69E-12	80.0	13.33		1.40E+09	5.95E+01	1.50E+11	1.59E+11	6.90E-12
Analog Devices	AD9260	1997	delta-sigma OSR8, DSP	2.50E+06	1.25E+06	16	89.5	14.57	2.61E-12	100.0	16.67		1.25E+08	6.00E+01	1.02E+11	1.02E+11	9.80E-12
University of Michigan, Intel	S Parl et al.	2006	flash	4.00E+09	8.00E+08	4	22.6	3.47	8.98E-12			0.46	8.00E+08	6.19E+01	7.16E+10	2.86E+10	3.49E-11
Hughes	ACTC	1998	n-stage	2.00E+07	1.00E+07	7	42.2	6.72	7.66E-11					6.40E+01	3.20E+09	3.29E+09	3.04E-10
Micro Networks	MN6500	1992	SA	1.00E+05	5.00E+04	16	68.0	14.33	7.75E-11	36.0	16.00	10.00	5.00E+04	6.05E+01	3.00E+09	3.00E+09	3.34E-10
Analog Devices	AD3070	1997	2-stage flash	1.00E+08	6.00E+07	10	57.0	9.18	2.75E-12				2.30E+08	7.00E+01	8.26E+10	8.26E+10	1.21E-11
Analog Devices	AD6640	1997	2-stage	5.50E+07	9.25E+07	12	67.0	10.84	1.34E-12	80.0	13.33	2.00	2.50E+07	7.10E+01	1.67E+11	1.67E+11	5.97E-12
National	ADC12C170	2007	pipelined, DSP	1.70E+08	7.00E+07	12	67.2	10.87	6.07E-13	85.4	14.23	2.00	1.10E+09	7.15E+01	4.46E+11	3.67E+11	2.73E-12
Linear Technology	LTC2242-12	2006	5-stage flash digital corr	250E+08	7.00E+07	12	65.1	10.52	7.23E-13	75.0	12.50	1.25	1.20E+03	7.40E+01	4.57E+11	2.78E+11	3.60E-12
Lucent Technologies	CSP1152A	1998	dithered	6.50E+07	3.25E+07	14	68.0	11.00	1.10E-12	85.0	14.17	1.80	1.00E+09	7.50E+01	1.78E+11	1.78E+11	5.62E-12
Leung et al.	Crystal Servic	1997	delta sigma OSR 64, 7th order, 3-bit	9.50E+04	4.80E+04	20	108.0	17.65	8.07E-12	112.0	18.67	4.00	2.20E+04	7.50E+01	2.59E+10	2.69E+10	3.85E-11
UCLA	Colluton & Abudi	1993	pipelined folding	7.50E+07	3.75E+07	10	59.8	9.51	2.91E-12	77.0	12.83		5.00E+07	8.00E+01	6.83E+10	6.83E+10	4.46E-11
Phillips, Netheri	van Valburg et al.	1992	folding	8.50E+08	3.25E+08	6	48.8	7.81	1.09E-12	51.0	6.50	2.00	1.50E+08	8.10E+01	1.81E+11	1.81E+11	5.54E-12
Analog Devices	AD9640	2007	dual SHA-pipelined+DSP	1.50E+08	7.00E+07	14	71.0	11.50	3.92E-13	84.0	14.00	1.00	6.50E+08	8.20E+01	5.30E+11	4.95E+11	2.02E-12
	AD9640	2007	dual; SHA-pipelined-DSP	1.50E+08	2.00E+08	14	69.9	11.32	1.56E-13	77.0	12.83	1.00	6.50E+08	8.20E+01	4.67E+11	1.25E+12	8.02E-13
Analog Devices	AD1879		delta-sigma, OSR 54	5.00E+04	2.50E+04	18	98.0	15.99	4.90E-11	103.0	17.17	6.00	2.20E+04	8.00E+01	3.61E+09	3.61E+09	2.77E-10
Analog Devices	Sone, et al	1993	pipelined, auxr	1.00E+08	5.00E+07	10	57.0	9.18	2.75E-12	61.0	10.17		1.50E+07	9.50E+01	6.09E+10	6.09E+10	1.64E-11
NEC	ADC08D1500	2005	folding, interpolating	1.50E+09	7.50E+08	8	46.3	7.40	6.29E-13	53.0	8.83	0.87	1.70E+09	9.50E+01	2.66E+11	2.66E+11	3.75E-12
National	Max1215	2005	pipelined DSF	2.50E+08	1.00E+08	12	64.3	10.39	5.94E-13	70.7	11.73	1.45	7.00E+08	9.75E+01	3.44E+11	2.75E+11	3.64E-12
Maxim integrated Prod.	Max1215	2006	pipelined DSP	2.50E+08	2.50E+08	12	64.2	10.37	2.40E-13	72.4	12.07	1.45	7.00E+08	9.75E+01	3.40E+11	6.00E+11	1.47E-12
Atmel	AT76		SA-2	1.00E+05	5.00E+04	18	90.0	14.66	6.16E-11			1.00		1.00E+00	2.58E+09	2.58E+09	3.87E-10
Analog Devices	Real et al.	1991	pipeline	2.00E+07	1.00E+07	10	54.0	8.68	1.94E-11	62.0	10.33		1.50E+08	1.00E+00	8.19E+09	6.19E+09	1.22E-10
Hughes	Jensen et al.	1995	delta-sigma, OSR 32	1.00E+08	5.00E+07	12	56.0	8.84	3.46E-12	71.0	11.83			1.00E+00	4.59E+10	4.59E+10	2.18E-11
Lob d'Electr & de Phys Appl	DuCourant et al.	1989	flash	2.20E+09	1.10E+08	5	28.7	4.48	3.25E-12				4.00E+08	1.05E+00	4.66E+10	4.68E+10	2.15E-11
Texas Instruments	ADS5546	2006	pipeline 18/stage SHA DSP	1.90E+08	7.00E+07	14	72.5	11.75	3.30E-13	84.0	14.00	2.00	5.00E+08	1.13E+00	5.79E+11	4.27E+11	2.34E-12
Texas Instruments	ADS5546	2006	pipeline 18/stage SHA DSP	1.90E+08	1.50E+08	14	70.8	11.47	1.87E-13	80.0	13.33	2.00	5.00E+08	1.13E+00	4.76E+11	7.62E+11	1.33E-12
Texas Instruments	ADS5546	2006	pipeline 18/stage SHA DSP	1.90E+08	3.00E+08	14	57.4	10.80	1.38E-13	72.0	12.00	2.00	5.00E+08	1.13E+00	3.22E+11	1.02E+12	9.83E-13
Analog Devices	AD773			1.90E+07	9.00E+06	10	53.0	8.51	2.42E-11	67.0	11.17	1.00	1.00E+08	120E+00	5.47E+09	5.47E+09	1.83E-10
	ADS5547	2007	pipelined+DSP	2.10E+08	7.00E+07	14	72.6	11.77	3.26E-13	85.0	14.17	2.00	8.00E+08	1.23E+00	5.59E+11	3.87E+11	2.52E-12

	ADS5547	2007	pipelined+DSP	2.10E+08	1.70E+08	14	70.7	11.45	1.67E-13	79.0	13.17	2.00	8.00E+08	1.23E+00	4.78E+11	7.74E+11	1.28E-12
Lunear Technology	LTC2206	2006	pipelined+DSP	1.30E+08	7.00E+07	16	77.4	12.56	1.68E-10	90.0	15.00	2.25	7.00E+08	1.25E+00	6.30E+11	6.79E+11	1.47E-12
Linear Technology	LTC2208	2006	pipelined+DSP	1.30E+08	1.40E+08	16	76.4	12.40	1.06E-13	85.0	14.17	2.25	7.00E+08	1.25E+00	5.62E+11	1.21E+12	8.27E-13
Linear Technology	LTC2208	2006	pipelined+DSP	1.30E+08	2.50E+08	16	73.8	11.33	8.14E-14	78.0	13.00	2.25	7.00E+08	1.25E+00	4.07E+11	1.58E+12	6.38E-13
Datel	ADS112		2-stage flash SH, DSP	1.00E+06	5.00E+05	12	70.0	11.34	6.16E-11	75.0	12.50	10.00	1.00E+07	1.30E+00	1.99E+09	1.99E+09	5.03E-10
Analog Devices	AD6644	2004	3-stage	6.50E+07	3.05E+07	14	73.0	11.83	7.15E-13	90.0	15.00	2.20	2.50E+07	1.30E+00	8.83E+11	1.71E+11	5.84E-12
Analog Devices	AD9430	2005	pipelined	2.10E+08	1.00E+08	12	84.5	10.42	5.80E-13	77.0	12.83	1.50	7.00E+08	1.30E+00	2.22E+11	2.11E+11	4.74E-12
Analog Devices	AD9430	2005	pipelined	2.10E+08	2.40E+08	12	60.0	9.67	4.05E-13	63.0	10.50	1.50	7.00E+08	1.30E+00	1.32E+11	3.20E+11	3.31E-12
Sony	CXA1176K		flash	2.50E+08	1.25E+08	8	39.0	6.19	8.74E-12				2.50E+08	1.40E+00	1.30E+10	1.30E+11	7.69E-11
Analog Devices	AD9026	1994		3.10E+07	1.55E+07	12	65.0	10.50	3.53E-12			2.00		1.46E+00	3.09E+10	3.09E+10	3.24E-11
Analog Devices	AD6645	2004	3-stage	1.05E+08	5.25E+07	14	74.5	12.08	3.49E-13	93.0	15.50	2.20	2.70E+08	1.50E+00	3.04E+11	3.04E+11	3.29E-12
Analog Devices	AD6645	2004	3-stage	1.05E+08	2.00E+08	14	72.0	11.67	1.22E-13	63.0	10.50	2.20	2.70E+08	1.50E+00	2.28E+11	6.68E+11	1.15E-12
Analog Devices	AD9460	2006	Pipeline	1.05E+08	3.00E+07	15	76.3	12.71	3.95E-13	90.0	15.00	3.40	8.00E+08	1.60E+00	4.41E+11	2.52E+11	3.97E-12
Analog Devices	AD9460	2006	Pipeline	1.05E+08	1.70E+08	15	76.6	12.43	8.47E-14	83.0	13.83	3.40	8.00E+08	1.60E+00	3.63E+11	1.17E+12	8.52E-13
Sig Proc Tech	SPT7312	1991	pipeline	200E+07	1.00E+07	12	58.0	9.34	1.23E-11	70.0	11.67	2.00	1.20E+08	1.60E+00	7.21E+09	7.21E+09	1.39E-10
Sig Proc Tech	SPT7624	1991	pipeline	400E+07	2.00E+07	10	50.0	8.01	1.54E-11	70.0	11.67	1.80	1.20E+08	1.60E+00	5.74E+09	5.74E+09	1.74E-10
Datel	ADS932	1996	2-stage	200E+06	1.00E+06	16	81.0	13.16	8.58E-12	84.0	14.00	5.50	4.00E+08	1.85E+00	9.91E+09	9.91E+09	1.01E-10
Datel	ADS 110		2-stage	5.00E+06	2.50E+06	12	68.0	10.67	1.95E-11	65.0	10.83	2.00	6.50E+07	1.90E+00	4.29E+09	4.29E+09	2.33E-10
Texas instruments	ADS5424	2005	3-stage+DSP	1.05E+08	5.00E+07	14	74.0	12.00	3.89E-13	93.0	15.50	2.20	5.70E+08	1.90E+00	2.26E+11	2.16E+11	4.64E-12
Texas instruments	ADS5424	2005	3-stage+DSP	1.05E+08	1.70E+08	14	69.1	11.19	2.01E-13	73.0	12.17	2.20	5.70E+08	1.90E+00	1.29E+11	4.17E+11	2.40E-12
Comlinear	CLC950	1994	2-stage	2.56E+07	1.28E+07	12	67.0	10.84	3.40E-12	75.0	12.50	2.00	1.75E+08	2.00E+00	2.34E+10	2.34E+10	4.27E-11
Analog Devices	Mangalsdori	1990	flash	2.00E+08	1.00E+08	6	47.0	7.51	4.35E-12				4.00E+08	2.00E+00	1.83E+10	1.83E+10	5.47E-11
Analog Devices	AD9006		flash	5.00E+08	2.50E+08	6	29.0	4.52	1.38E-11			2.00	5.50E+08	2.00E+00	5.76E+09	5.76E+09	1.74E-10
NTT LSI Lab	Wakimoto et al.	1988	flash	1.00E+09	5.00E+08	6	36.7	5.80	2.85E-12	50.0	8.33	2.00	1.70E+09	2.00E+00	2.79E+10	2.79E+10	3.58E-11
Edge Technology	ET1463	1992		3.00E+06	1.50E+06	14	78.0	12.66	8.17E-12			2.00		2.17E+00	8.98E+09	8.98E+09	1.11E-10
Edge Technology	ET1465	1992		5.00E+06	2.50E+06	14	78.0	12.66	4.90E-12			2.00		2.17E+00	1.50E+10	1.50E+10	6.69E-11
Analogic	ADC4357		3-stage	2.00E+05	1.00E+05	16	90.0	14.66	3.08E-11	90.0	15.00	10.00		2.20E+00	2.35E+09	2.35E+09	4.26E-10
Analog Devices	AD9028		flash	3.00E+08	1.50E+08	8	36.0	5.69	1.03E-11			2.00	2.50E+08	2.20E+00	7.03E+09	7.03E+09	1.42E-10
Texas Instruments	ADS5463	2006	pipelined	5.00E+08	2.30E+08	12	64.7	10.46	2.46E-13	78.0	13.00	2.20	1.30E+09	2.20E+00	3.19E+11	2.94E+11	3.41E-12
Analog Devices	AD9445	2005	pipelined	1.25E+08	7.00E+07	14	73.5	11.92	2.94E-13	85.0	14.17	3.20	3.00E+08	2.30E+00	2.10E+11	2.35E+11	4.25E-12
Analog Devices	AD9446	2005	pipelined	1.00E+08	1.25E+08	16	77.5	12.58	1.04E-13	82.0	13.67	3.20	2.25E+08	2.30E+00	2.66E+11	6.66E+11	1.50E-12
Atmel	AT84AS001	2006	SHA+pipeline	5.00E+08	2.50E+08	12	62.0	10.01	3.09E-13	72.0	12.00	1.10	1.50E+09	2.30E+00	2.24E+11	2.24E+11	4.47E-12
Analogic	ADC5120		3-stage	5.00E+04	2.50E+04	20	103.0	16.82	2.76E-11	104.0	17.33	10.00	1.40E+05	2.38E+00	2.43E+09	2.43E+09	4.12E-10
Inst. fur Electr. Ruhr Univ.	Daniel et al.	1988	stacked flash	1.00E+09	5.00E+08	4	25.0	3.86	1.10E-11	36.0	6.00	1.28	8.00E+08	2.40E+00	6.05E+09	6.05E+09	1.65E-10
Sig Proc Tech	HADC77100		flash	1.50E+08	7.50E+07	8	38.0	6.02	1.64E-11			2.00	1.75R+08	2.60E+00	3.74E+09	3.74E+09	2.67E-10
Hughes	ACTC	1986		1.00E+09	5.00E+08	6	35.0	5.52	3.46E-12	37.1	6.18		5.00E+08	2.60E+00	1.77E+10	1.77E+10	5.66E-11
Analog Devices	AD1362		3-stage	5.00E+05	2.50E+05	16	91.0	14.82	1.10E-11	85.0	14.17	10.00	2.00E+05	2.80E+00	5.18E+09	5.18E+09	1.93E-10
Datel	ADS941		2-stage	1.00E+06	5.00E+05	14	76.0	12.33	3.09E-11	87.0	14.50	10.00	6.00E+06	2.80E+00	1.84E+09	1.84E+09	5.43E-10
Analog Devices	AD9060		flash	7.50E+07	3.75E+07	10	48.0	7.68	1.03E-11			4.00	1.75E+08	2.80E+00	5.50E+09	5.50E+09	1.82E-10
Datel	ADS942		2-stage	2.00E+06	1.00E+06	14	76.0	12.33	1.54E-11	80.0	13.33	10.00	6.00E+06	2.90E+00	3.56E+09	3.56E+09	2.81E-10
Nortel	P Schvan et al.	2006	flash	2.20E+10	7.00E+09	5	22.8	3.50	1.00E-12	29.0	4.83	0.54		3.00E+00	8.30E+10	5.28E+10	1.89E-11
Sony	CXA1276K		flash	4.00E+08	2.00E+08	8	40.0	6.35	4.87E-12				2.50E+08	3.10E+00	1.05E+10	1.05E+10	9.49E-11
Datel	ADS944	1994	2-stage flash, SH, DSP	5.00E+06	2.50E+06	14	71.0	11.50	1.10E-11			2.50		3.37E+00	4.30E+09	4.30E+09	2.32E-10
Datel	ADS 930		2-stage	5.00E+05	2.50E+05	16	80.0	13.00	3.89E-11	85.0	14.33	10.00	2.00E+06	3.40E+00	1.20E+09	1.20E+09	8.32E-10
Analogic	ADC 4344		2-stage	1.00E+06	5.00E+05	16	82.0	13.33	1.55E-11	91.0	15.17	5.00	4.00E+06	3.40E+00	3.30E+09	3.03E+09	8.30E-10
Fraunhofer & TnQuint	Hageiauer et al.	1992	flash	1.00E+09	5.00E+08	5	30.7	4.81	5.68E-12	33.7	5.62	1.80	5.00E+08	3.40E+00	8.24E+09	8.24E+09	1.21E-10
HP	Jewett et al.	1992	ripple ch	2.00E+07	1.00E+07	12	65.0	10.50	5.48E-12	72.0	12.00		9.50E+07	3.50E+00	8.30E+09	8.30E+09	1.20E-10
Hughes	Baringer, et al.	1996	flash	8.00E+09	4.00E+09	3	18.2	2.73	3.00E-12	27.0	4.50	0.60	1.20E+10	3.50E+00	1.52E+10	1.52E+10	8.59E-11
NTT Photonics Labs	Nosaka et al.	2004	flash	2.00E+10	1.00E+10	3	15.6	2.30	1.62E-12	23.3	3.88	0.50	2.00E+10	3.84E+00	2.56E+10	2.56E+10	3.90E-11
Datel	ADS130		2-stage	1.00E+07	5.00E+06	12	65.0	10.50	1.10E-11	67.0	11.17	2.50	6.50E+07	3.85E+00	3.77E+09	3.77E+09	2.65E-10
Analogic	ADC3110		2-stage	2.00E+06	1.00E+06	14	84.0	13.66	6.14E-12	76.0	12.67	10.00	2.00E+07	4.10E+00	6.32E+09	6.32E+09	1.58E-10
Datel	ADS945	1994	2-stage	1.00E+07	5.00E+08	14	74.0	12.00	3.89E-12	79.0	13.17	2.50	5.00E+07	4.20E+00	9.75E+09	9.75E+09	1.03E-10
Atmel	AT84AS008	2005	flash, err corr	1.70E+09	8.50E+08	10	52.0	8.35	2.88E-13	56.0	9.33	0.50	3.30E+09	4.20E+00	1.32E+11	1.32E+11	7.60E-12
Atmel	AT84AS008	2005	flash, err corr	2.20E+09	2.00E+09	10	48.0	7.68	194E-13	55.0	9.17	0.50	3.30E+09	4.20E+00	1.07E+11	1.95E+11	5.12E-12
TRW	THC1202		pipeline	1.00E+07	5.00E+06	12	67.0	10.84	8.70E-12	67.0	11.17	2.00	7.00E+07	4.50E+00	4.07E+09	4.07E+09	2.46E-10
Atmel	TS83102	2004	flash err corr	1.40E+09	7.00E+08	10	47.5	7.60	5.87E-13	59.0	9.83	0.50	3.30E+09	4.60E+00	5.90E+10	5.90E+10	1.70E-11
Comlinear	CLC935B	1994	2-stage	1.50E+07	7.50E+06	12	65.6	10.60	6.81E-12	82.3	13.72	2.00	8.00E+07	4.75E+00	4.92E+09	4.92E+09	2.03E-10

Analog Devices	AD9032	1992	2-stage	2.50E+07	1.25E+07	12	66.0	10.67	3.90E-12	72.0	12.00	2.00	1.00E+07	5.00E+00	6.15E+09	8.15E+09	1.23E-10
Rockwell Scientific	RAD010	2006	2-stage flash	1.00E+09	5.00E+08	10	55.0	8.84	3.46E-13	60.0	10.00	1.00	6.00E+09	5.00E+00	9.19E+10	9.19E+10	1.09E-11
Rockwell	RSC-ADC080S	1998	folded flash	2.00E+09	1.00E+09	8	37.0	5.85	1.38E-12	43.0	7.17	0.80	5.00E+08	5.00E+00	2.31E+10	2.31E+10	4.32E-11
Maxim Integrated Prod.	Max104	1999	flash	1.00E+09	5.00E+08	8	47.2	7.55	8.50E-13	52.3	8.72	0.50	2.20E+09	5.25E+00	3.57E+10	3.57E+10	2.81E-11
Maxim Integrated Prod.	Max108	2004	flash	1.50E+09	7.50E+08	8	46.9	7.50	5.87E-13	54.1	9.02	0.50	2.20E+09	5.25E+00	5.17E+10	5.17E+10	1.94E-11
Comlinear	CLC936C	1994	2-stage	2.00E+07	1.00E+07	12	65.0	10.50	5.48E-12	75.6	12.60	2.00	9.00E+07	5.28E+00	5.50E+09	5.50E+09	1.82E-10
Rockwell	Nary et al.	1995	folded flash	2.00E+09	1.00E+09	8	41.4	6.58	8.29E-13	48.0	8.00	0.64	3.00E+09	5.30E+00	3.62E+10	3.62E+10	2.76E-11
Edge Technology	ET1661	1992		1.00E+06	5.00E+05	16	90.0	14.66	6.16E-12			10.00		5.50E+00	4.70E+09	4.70E+09	2.13E-10
Edge Technology	ET1662	1992		2.00E+06	1.00E+06	16	90.0	14.66	3.08E-12			10.00		5.50E+00	9.40E+09	9.40E+09	1.06E-10
Edge Technology	ET1663	1992		3.00E+06	1.50E+06	16	90.0	14.66	2.05E-12			10.00		5.50E+00	1.41E+10	1.41E+10	7.09E-11
TelAsic	TC1200	2002	folded flash	1.00E+09	4.00E+08	10	49.3	7.90	8.35E-13				4.00E+08	5.50E+00	4.33E+10	3.47E+10	2.88E-11
Signal Processing Tech	SPT7760A	1995	flash	1.00E+09	5.00E+08	8	42.0	6.68	1.55E-12	47.0	7.83	2.00	9.00E+08	5.50E+00	1.87E+10	1.87E+10	5.35E-11
Rockwell Scientific	RAD008	2006	folding interpolating	3.00E+09	1.50E+09	8	46.0	7.35	3.25E-13	55.0	9.17	2.00	1.00E+10	5.50E+00	8.89E+10	8.89E+10	1.12E-11
Edge Technology	ET1471	1992		1.00E+07	5.00E+06	14	78.0	12.66	2.45E-12			2.00		5.70E+00	1.14E+10	1.14E+10	8.78E-11
Hewlett Packard	Jewett et al	1997	2-stage folding	1.28E+08	6.40E+07	12	61.5	9.92	1.28E-12	70.0	11.67	0.50	1.80E+09	5.70E+00	2.18E+10	2.18E+10	4.59E-11
HP & Rockwell	Poulton(HP), Wang(R)	1994	flash	4.00E+09	2.00E+09	6	33.1	5.21	1.08E-12	33.7	5.62		1.00E+10	5.70E+00	2.59E+10	2.59E+10	3.86E-11
Rockwell Scientific	RAD006	2006	folding interpolating	6.00E+09	3.00E+09	6	34.0	5.36	6.48E-13	40.0	6.67	2.00	3.00E+07	6.00E+00	4.09E+10	4.09E+10	2.44E-11
Burr Brown	ADC614	1994	2-stage	5.00E+06	2.50E+06	14	78.0	12.66	4.90E-12	88.0	14.67	2.50	4.00E+07	6.10E+00	5.32E+09	5.32E+09	1.88E-10
Burr Brown	ADC603	1992	2-stage	1.00E+07	5.00E+06	12	65.0	10.50	1.10E-11	72.0	12.00	2.50		6.10E+00	2.38E+09	2.38E+09	4.20E-10
Micro Networks	MN5420	1991	auto ranging	3.20E+05	1.60E+05	12	60.0	9.67	6.09E-10	120.0	20.00		9.00E+06	6.50E+00	4.02E+07	4.02E+07	2.49E-08
Analog Devices	AD1388	1992		2.00E+06	1.00E+06	16	82.0	13.33	7.73E-12			10.00	3.00E+09	6.50E+00	3.17E+09	3.17E+09	3.16E-10
Atmel	AT84AS004	2005	flash err corr	2.00E+09	1.00E+09	10	51.0	8.18	2.75E-13	55.0	9.17	0.50	1.00E+08	6.50E+00	8.92E+10	8.92E+10	1.12E-11
Comlinear	CLC938C	1994	2-stage	3.07E+07	1.54E+07	12	64.6	10.44	3.73E-12	72.2	12.03	2.00	2.80E+09	6.57E+00	6.49E+09	6.49E+09	1.54E-10
Maxim Integrated Prod.	Max 109	2007	flash T/H	2.20E+09	1.00E+09	8	43.1	6.87	6.82E-13	51.1	8.52	0.40	1.00E+08	6.80E+00	3.78E+10	3.43E+10	2.91E-11
Comlinear	CLC937B	1994	2-stage	2.56E+07	1.28E+07	12	64.8	10.47	4.38E-12	73.3	12.22	2.00	1.20E+09	7.35E+00	4.95E+09	4.95E+09	2.02E-10
Micro Networks	MN6900	1991	flash	5.00E+08	2.50E+08	8	48.0	7.68	1.55E-12	61.8	10.30	0.54	6.00E+07	7.50E+00	1.37E+10	1.37E+10	7.31E-11
TRW	TAC1025		flash	2.50E+07	1.25E+07	10	49.0	7.85	2.76E-11	53.0	8.83	1.00	1.80E+08	7.80E+00	7.38E+08	7.38E+08	1.35E-09
Analog Devices	AD12401	2005	parallel V-Corp	4.00E+08	1.28E+08	12	64.4	10.41	4.58E-13	75.0	12.50	3.20	6.60E+09	8.50E+00	6.38E+10	4.08E+10	2.45E-11
Agilent Labs	Poulton et al.	2003	time interleaved	2.00E+10	6.00E+09	8	29.5	4.60	5.47E-13			0.25	2.50E+08	9.00E+00	5.39E+10	3.23E+10	3.09E-11
Hughes	ACTC	1988	flash	5.00E+08	2.50E+08	8	40.0	6.35	3.90E-12				6.00E+07	1.10E+01	3.71E+09	3.71E+09	2.69E-10
Analog Devices	AD9014	1992	2-stage	1.00E+07	5.00E+06	14	75.0	12.17	3.46E-12	88.0	14.67	2.00		1.28E+01	3.59E+09	3.59E+09	2.79E-10
Hughes	Boyko, GSG	1990	pipeline	6.00E+06	3.00E+06	15	75.0	12.17	5.77E-12	90.0	15.00		3.00E+07	2.20E+01	1.25E+09	1.25E+09	7.98E-10
Hughes	ACTC	1978	n-stage	6.00E+07	3.00E+07	13	62.0	10.01	2.58E-12	70.0	11.67		4.00E+09	2.50E+01	2.47E+09	2.47E+09	4.05E-10
HP	Poulton et al.	1997	time interleaved	8.00E+09	4.00E+09	8	24.2	3.73	1.50E-12					2.70E+01	3.92E+09	3.92E+09	2.55E-10
Hughes	ACTC	1991	n-stage	2.50E+07	1.25E+07	14	73.0	11.83	1.74E-12	84.0	14.00			3.00E+01	3.04E+09	3.04E+09	3.29E-10
Hewlett Packard	Schiller & Byrne	1991	time interleaved	4.00E+09	2.00E+09	8	41.5	6.60	4.10E-13					3.90E+01	9.96E+09	9.96E+09	1.00E-10

* The entries in this table are color-coded in the same manner as the points in Fig. 8, i.e., according to power dissipation.

SUMMARY

The current state-of-the-art for ADCs has been reviewed and analyzed. Data for SNDR and SFDR as functions of f_{sig} and f_{samp} have been discussed. The SNDR data show that converter performance is limited by input-referred noise, aperture uncertainty, and comparator ambiguity. The best performances have been achieved for pipelined flash (successive approximation, multistage flash) folded flash, and time-interleaved architectures. Many of these converters employ DSP for error correction and channel matching. The best ADCs can operate in undersampling mode.

With respect to aperture uncertainty, about 3 bits of overall improvement has been achieved over the last 4 years in both SNDR and SFDR. The best converters were two pipelined ADCs, which achieved τ_a ~81–85 fs (Table 1 #1, 2). In addition, the best power-efficient converters, as measured by FOM_a, have reached down to nearly 100 fs per conversion step.

It is clear from the data presented above that significant improvements in converter performances have been achieved during the early twenty-first century and that the performance picture is dramatically better than it was in 1999 (1). In addition, ADC power dissipation has also been reduced and is largely caused by the continuing advances in IC technology (Moore's Law).

Although continued progress in ADC IC design and technology will no doubt continue, advancements in converter performance may also be aided by heterogeneous integration, photonic sampling, and/or by superconducting implementations.

ACKNOWLEDGMENT

This work was supported by The Aerospace Corporation's Independent Research and Development Program.

The author gratefully acknowledges G.C. Valley for his encouragement and helpful comments, especially regarding photonic ADCs, and thanks S.C. Moss for his support and helpful suggestions, especially regarding the application of heterogeneous integration to ADCs.

APPENDIX 1 TABLE OF ADCS COVERED IN THIS WORK

The entries in this table are color-coded in the same manner as the points in Figure 8, i.e., according to power dissipation.

BIBLIOGRAPHY

1. R. H. Walden, Analog-to-digital converter survey and analysis, *IEEE J. Sel. Areas Communica.* **17**(4): 539–550, 1999.
2. K. G. Merkel II and A. L. Wilson, A survey of high performance analog-to-digital converters for defense space applications, *IEEE Proc. Aerospace Conf., vol. 5, paper 1344*, 2003, pp. 5-2415–5-2427.
3. B. Le et al., Analog-to-digital converters, *IEEE Signal Proc. Mag.*, 2005, pp. 69–77.
4. See for example, Atmel data sheet for AT84AS004, Nov. 2005, p. 40.
5. S. Gupta et al., A 1GS/s 11b time-interleaved ADC in 0.13μm CMOS, *Internat. Solid-State Circuits Conf. Digest of Tech. Papers*, paper 31.6, 2006.
6. C. Schiller and P. Byrne, A 4-GHz 8-b ADC system, *IEEE J. Solid-State Circuits*, **26**(12): 1781–1789, 1991.
7. K. Poulton et al., A 20 GS/s 8-b ADC with a 1MB memory in 0.18um CMOS, *Internat. Solid-State Circuits Conf. Digest of Tech. Papers*, vol. 496, 2003, pp. 318–319.
8. K. Nary et al., An 8-bit, 2 gigasample per second analog to digital converter, *GaAs IC Symp. Tech. Digest*, **17**: 303–246, 1995.
9. Mitteregger et al., A 14b 20mW 640MHz CMOS CT ΣΔ ADC with 20MHz signal bandwidth and 12b ENOB, *Internat. Solid-State Circuits Conf. Digest of Tech. Papers*, paper 03.1, 2006.
10. P. Schvan et al., A 22GS/s 5b ADC in 0.13um SiGe BiCMOS, *Internat. Solid-State Circuits Conf. Digest of Tech. Papers*, paper 31.4, 2006.
11. J. van Valberg and R. J. van de Plassche, An 8-bit 650 MHz folding ADC, *IEEE J. Solid-State Circuits*, **27**(12): 1662–1666, 1992.
12. P. Bogner et al., A 14b 100MS/s digitally self-calibrated pipelined ADC in 0.13μm CMOS, *Internat. Solid-State Circuits Conf. Digest of Tech. Papers*, paper 12.6, 2006.
13. R. Jewett et al., A 12b 128 MSanples/s ADC with 0.05LSB DNL, *Internat. Solid-State Circuits Conf. Digest of Tech. Papers*, vol. 443, 1997, pp. 439–443.
14. K. Poulton et al., A 6-bit, 4 Gsa/s ADC fabricated in a GaAs HBT process, *GaAs IC Symp. Tech. Digest*, 16: 240–243, 1994.
15. H. Nosaka et al., A 24-Gsps 3-bit Nyquist ADC using InP HBTs for electronic dispersion compensation, *IEEE MTT-S Digest*, 2004, pp. 101–104.
16. K. Poulton et al., An 8-GSa/s 8-bit ADC system, *Tech. Digest of VLSI Circuits Symp.*, 1997, pp. 23–24.
17. J. C. Candy and G. C. Temes, eds., *Oversampling Delta-Sigma Converters*. New York: IEEE Press, 1992.
18. L. Pellon, Military applications of high-speed ADCs, *IEEE MTT Workshop, WMA: Application and Technology of High-Speed Analog-to-Digital Converters*, 2005.
19. L. Luh et al., A 4GHz 4th order passive LC bandpass delta-sigma modulator with IF at 1.4 GHz, *Symp. VLSI Circuits Digest of Technical Papers*, 2006, pp. 208–209.
20. A. W. Fang et al., Electrically pumped hybrid AlGaInAs-silicon evanescent laser, *Optics Expr.*, **14**(20): 9203–9210, 2006.
21. M. Paniccia et al., A hybrid silicon laser: silicon photonics technology for future tera-scale computing, *Technology@Intel Magazine*, 2006, pp. 44–50.
22. Y. Liu, Heterogeneous integration of OE arrays with Si electronics and microoptics, *IEEE Trans. Adv. Packag.* 25(1): 43–49, 2002.
23. T. B. Cho and P. R. Gray, A 10b, 20 Msample/s, 35 mW pipeline A/D converter, *IEEE J. Solid-State Circuits*, **30**(3): 166–172, 1995.
24. S-U. Kwak, B-S. Song and K. Bacrania, A 15b 5Msample/s low-spurious CMOS ADC, *IEEE ISSCC Digest Tech. Papers*, vol. 40, 1997, pp. 146–147.
25. G. C. Valley, Photonic analog-to-digital converters, *Optics Express*, **15**(15): 1955–1982, 2007.
26. D. Gupta et al., Analog-to-digital converter and SiGe output amplifier, *IEEE Trans. Appl. Superconduct.*, **13**(2): 477–483, 2003.

R. H. Walden
The Aerospace Corporation
Electronics & Photonics
Laboratory
Los Angeles, California

AUTOMATIC TEST GENERATION

INTRODUCTION

This article describes the topic of automatic test generation (ATG) for digital circuitsjmd systems. Considered within the scope of ATG are methods and processes that support computer-generated tests and supporting methodologies. Fundamental concepts necessary to understand defect modeling and testing are presented to support later discussions on ATG techniques. In addition, several closely related topics are also presented that affect the ATG process, such as design for test (DFT) methodologies and technologies.

One can test digital systems to achieve one of several goals. First, testing can be used to verify that a system meets its functional specifications. In functional testing, algorithms, capabilities, and functions are verified to ensure correct design and implementation. Once a system has been verified to be correct, it can be manufactured in quantity. Second, one wishes to know whether each manufactured system is defect free. Third, testing can be used to determine whether a system is defect free. Functional tests can provide the basis for defect tests but are ineffective in providing acceptable defect tests. Defect tests can be developed by creating tests in an ad hoc fashion followed by evaluation using a fault simulator. In complex systems, this process can be challenging and time consuming for the test engineer. As a result, many effective techniques have been developed to perform ATG as well as to make ATG more effective.

Technologic trends have continued to offer impressive increases in capability and performance in computing function and capacity, Moore's law noted the annual doubling of circuit complexities in 1966 that continued through 1976 (1).[1] Although the rate of complexity doubling slowed, such increases are both nontrivial and continuous with the increases in complexity comes the added burden of testing these increasingly complex systems.

Supporting the technologic improvements are complementary advances in the many supporting technologies including design tools, design practices, simulation, manufacturing, and testing. Focusing on testing, the technologic advances have impacted testing in several ways. First, the increase in the number of pins for an integrated circuit has not increased at the same rate as the number of devices on the integrated circuit. In the context of testing, the increasing relative scarcity of pins creates a testing bottleneck because more testing stimulus and results must be communicated through relatively fewer pins. Second, design methodologies have changed to reflect the trends in the introduction of increasingly more complex systems. So-called systems on chip (SoC) approaches enable designers to assemble systems using intellectual property (IP) cores purchased from vendors. SoCs present their own challenges in testing. SoC testing is complicated even more by observing that vendors are, understandably, reluctant to provide sufficient detail on the inner workings of their cores to enable the development of a suitable defect test. Indeed, vendors may be unwilling to provide test vectors that can provide hints on the inner workings. As a result, the idea of embedded test has grown out of these challenges. Third, the sheer complexity of the systems can make it prohibitively expensive to develop effective tests manually. As a result, reliance on tools that can generate tests automatically can reduce manufacturing costs. Furthermore, effective testing schemes also rely on the integration of testing structures to improve the coverage, reduce the number of tests required, and complement the ATG process. Fourth, ATG serves as an enabling technology for other testing techniques. For example, synthesis tools remove the burden of implementing systems down to the gate level. At the same time, gate level detail, necessary for assembling a testing regimen, may be hidden from the designer. ATG fills this gap by generating tests for synthesis without requiring the designer to develop test synthesis tools as well.

FUNDAMENTALS OF TESTING

In this section, fundamental concepts from testing are introduced. First, fault modeling is presented todefinelhe target for testing techniques. Second, testing measures are presented to provide a metric for assessing the efficacy of a given testing regimen. Finally, fault simulation is usedto quantify the testing measures. We will use the three universe mode (2) to differentiate the defect from the manifestation of the fault and also the system malfunction. A fault is the modelof the defect that is present physically in the circuit. An error is the manifestation of the fault where a signal will have a value that differs from the desired value. A failure is the malfunction of the system that results from errors.

Fault Modeling

Circuits can fail in many ways. The failures can result from manufacturing defects, infant mortality, random failures, age, or external disturbances (2). The defects qan be localized, which affect function of one circuit element, or distributed, which affect many of all circuit elements. The failures can result in temporary or permanent circuit failure. The fault model provides an analytical target for testing methodologies and strategies. Thus, the fidelity of the fault models, in the context of the implementation technology, can impact the efficacy of the testing (3). For example, the stuck fault model is considered to be an ineffective model for many faults that occur in CMOS

[1] Moore actually stated his trend in terms of "the number of components per integrated circuit for minimum cost" (1)

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

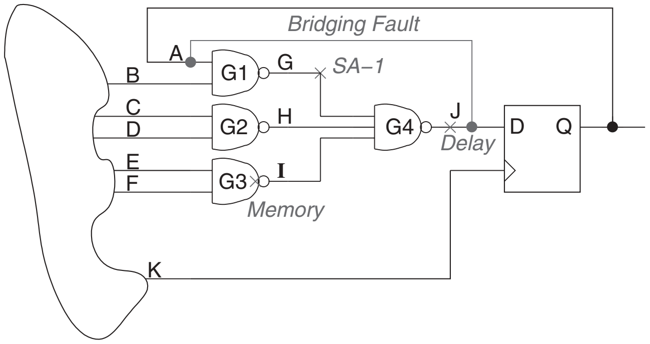


Figure 1. An illustration of fault models.

circuits. In addition, the fault model may influence the overall test strategy.

The fault models selected depend on the technology, used to implement the circuits. Manufacturing defects exist as a consequence of manufacturing the circuit. The introduction and study of manufacturing defects is a heavily studied topic because of its impact on the profitability of the device. Dust or other aerosols in the air can affect the defect statistics of a particular manufacturing run. In addition, mask misalignment and defects in the mask can also increase the defect densities. Other fault models can account for additional failure processes such as transient faults, wear out, and external disturbances. Figure 1 gives some example faults that are discussed in more detail in the following sections.

Stuck-at Fault Models. Stuck-at fault models are the simplest and most widely used fault models. Furthermore, the stuck-at fault model also models a common failure mode in digital circuits. The stuck-at fault model requires the adoption of several fundamental assumptions. First, a stuck-at fault manifests itself as a node being stuck at either of the allowable logic levels, zero or one, regardless of the inputs that are applied to the gate that drive the node. Second, the stuck-at fault model assumes that the faults are permanent. Third, the stuck-at fault model assumes a value for the fault, but otherwise preserves the gate function. The circuit shown in Fig. 1 is used to illustrate the fault model. The output of gate G_1 can be stuck-at 1 (SA-1) as a result of a defect. When the fault is present, the corresponding input to G_4 will always be one. To express the error, a discrepancy with respect to fault-free operation must occur in the circuit as a consequence of the fault. To force the discrepancy, the circuit inputs are manipulated so that $A = B = 1$, and a discrepancy is observed at the output of G1. A second example circuit is shown in Fig. 2, which consists of an OR gate (G_1) that drives one input in each of three AND gates (G_2, G_3, and G_4). Consider the presence of an SA-1 fault on any input to G_1 fault results in the output being 1 as a consequence of the fault. In G_1 input and output SA-1 faults are indistinguishable and for modeling purposes can be “collapsed” into a single fault. A somewhat higher fidelity model can also include gate input stuck faults. For example, in the event gate input I_2 has a stuck-at 1 fault, the situation is somewhat different. In this case, $O_1 = I_3 = I_4$ with G_3 and G_4 not affected directly by the fault.

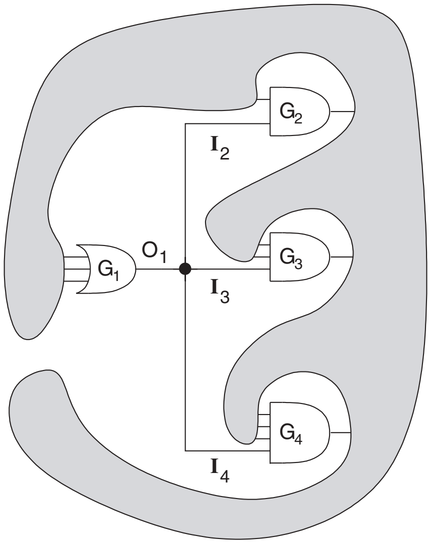


Figure 2. Illustration of input stuck-at faults.

Delay Fault Models. A delay fault is a fault in which a part of the circuit operates slowly compared with a correctly operating circuit. Because normal manufacturing variations result in delay differences, the operation must result in a sufficient delay discrepancy to produce a circuit malfunction. For example, if a delay fault delays a change to a flip-flop excitation input after the expected clock edge, then a fault is manifest. Indeed, when a delay fault is present, the circuit may operate correctly at slower clock rates, but not at speed. Delay faults can be modeled at several levels (4). Gate delay fault models are represented as excessive propagation delay. The transition fault model is slow to transition either a from 0 to 1 or from 1 to 0. A path delay fault is present when the propagation delay through a series of gates is longer than some tolerable worst case delay. Indeed, a current industry practice is to perform statistical timing analysis of parts. The manufacturer can determine that the parts can be run at a higher speed with a certain probability so that higher levels of performance can be delivered to customers. However, this relies on the statistical likelihood that delays will not be worst case (4). By running the device at a higher clock rate indicated by statistical grading, devices and structures that satisfy worst-case timing along the critical path may not meet the timing at the new, higher clock rate. Hence, a delay fault cas seem to be a consequence of the manufacturing decisions. Assuming the indicated delay fault in Fig. 1, Fig. 3 gives a timing diagram that shows the manifestation of the fault. In this circuit, the delay fault causes the flip-flop input, J, to change later, which results in a clock period delay in the flip-flop state change. Because of the nature of delay faults, circuits must be tested at speed to detect the delay fault. As posed here, the delay fault is dynamic, requiring two or more test vectors to detect the fault.

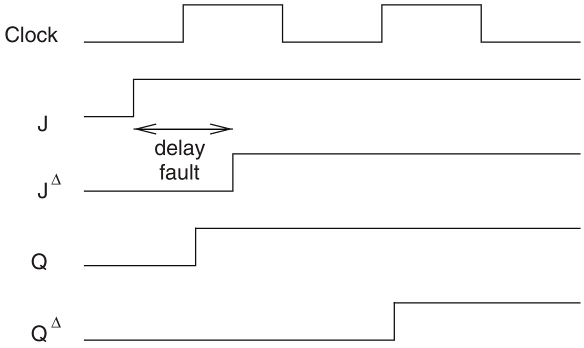


Figure 3. Illustration of a delay faults.

Bridging Faults. Bridging faults exist when an undesirable electrical connection occurs between two nodes resulting in circuit performance degradation or malfunction. Bridging faults between a circuit node and power supply or ground may be manifest as stuck faults. Furthermore, bridging faults may result in behavioral changes such as wired-and, wired-or, and even sequential characteristics when the bridging fault creates a feedback connection (5). Bridging faults require physical proximity between the circuit structures afflicted by the bridging faults. Figure 1 gives an example of a bridging fault that changes the combinational circuit into a sequential circuit.

CMOS Fault Models. CMOS technology has several fault modes that are unique to the technology (5). Furthermore, as a consequence of the properties of the technology, alternative methods for detecting faults in CMOS circuits are necessary. CMOS gates consist of complementary networks of PMOS and NMOS transistors structured such that significant currents are drawn only when signal changes occur. In fault-free operation, when no signal changes occur, the circuit draws very low leakage currents. Understanding the different fault models requires deeper exploration into CMOS circuit structures. CMOS circuits are constructed from complementary pull-up networks of PMOS transistors and pull-down networks of NMOS transistors. In addition, MOS transistors switch based on voltage levels relative to the other transistor terminals. The switching input, or gate, is the input and draws no current other than very low leakage currents. The gate does, however, have significant parasitic capacitance that must be charged and discharged to switch the transistor. Thus, significant currents are drawn when transistors are switched.

Table 1. Input sequence to detect transistor Q_1-stuck-open

A	B	Note
0	0	Set C to 1
0	1	C remains 1 for fault, 0 for no fault

In addition to the stuck faults, CMOS circuits have an interesting failure mode where an ordinary gate can be transformed into a dynamic sequential circuit for certain types of faults. The fault is a consequence of a transistor failure, low quiescent currents, and capacitive gate inputs. In Fig. 4, if transistor Q_1 is stuck open and if $A = 0$, the past value on node C is isolatedelectrically and will act as a storage element through the capacitance on the inverter input. Table 1 summarizes the sequence of inputs necessary to detect the transistor Q_1 stuck open fault. To detect this fault, node C must first be set by assigning $A = B = 0$ followed by setting $B = 1$ to store the value at the input of G_2. Each of the four transistors in the NAND gate will require a similar test.

The CMOS circuit's current draw can be used as a diagnostic for detecting faults. For example, because the CMOS circuit should only draw significant currents when the circuit is switching, any significant deviation from a known current profile suggests faults. Indeed, the CMOS

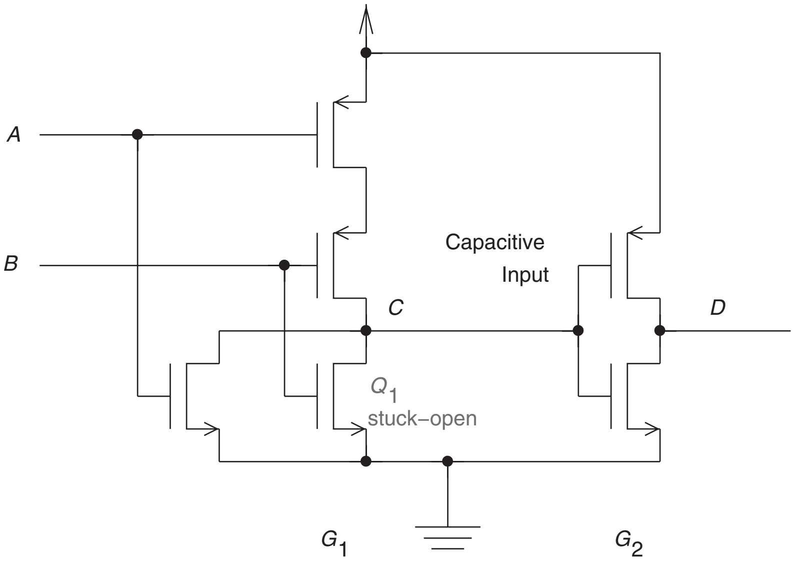


Figure 4. An illustration of a CMOS memory fault.

logic gates may function correctly, but when faults are present, the circuit may draw abnormally large power supply currents. Testing for faults based on this observation is called I_{DDQ} testing. Bridging faults are common in CMOS circuits (6) and are detected effectively with I_{DDQ} testing (7). I_{DDQ} faults can have a significant impact on portable designs where the low current drawn by CMOS circuits is required. Increased I_{DDQ} currents can result from transistors that are degraded because of manufacturing defects such that measurable leakage currents are drawn. In addition, bridging faults can also show increased I_{DDQ} currents.

Memory Faults. Semiconductor memories have structures that are very regular and very dense. As a result, memories can exhibit faults that are not observed ordinarily in other circuits that can complicate the testing process. The faults can affect the memory behavior in unusual ways (8). First, a fault can link two memory cells such that when a value is written into one cell, the value the linked cell toggles. Second, the memory cell can only be written to 0 or 1 but cannot be written the opposite value. Third, the behavior of a memory cell may be sensitive to the contents of neighboring cells. For example, a particular pattern of values stored in surrounding cells may prevent writing into the affected cell. Fourth, the particular pattern of values stored in the cells can result in a change in the value in the affected cell. The nature of these faults make detection challenging because the test must take into account the physical locality of memory cells.

Crosspoint Faults. Crosspoint faults (9) are a type of defect that can occur in programmable logic arrays (PLAs). PLAs consist of AND arrays and OR arrays with functional terms contributing through programming transistors to either include or exclude a term. In field programmable devices, a transistor is programmed to be on or off, respectively, to represent the presence or absence of a connection. A crosspoint fault is the undesired presence or absence of a connection in the PLA. Clearly, because the crosspoint fault can result in a change in the logic function, the stuck fault model cannot model crosspoint defects effectively. A crosspoint fault with a missing connection in the AND array results in a product term of fewer variables, whereas an extra connection results in more variables in the product term. For example, consider function $f(A, B, C, D) = AB + CD$ implemented on a PLA. The existence of a crosspoint fault can change the function to $f^{\mathrm{cpf}}(A, B, C, D) = ABC + CD$. Figure 5 diagrams the structure of the PLA and the functional effect of the crosspoint fault.

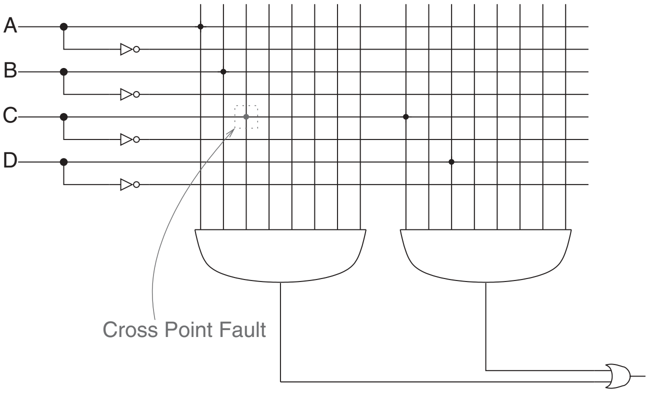


Figure 5. An illustration of a crosspoint fault.

I_{DDQ} Defects. An ideal CMOS circuit draws current only when logic values change. In practice, because transistors are not ideal, a small leakage current is drawn when no circuit changes occur. Many circuit defects result in anomalous significant current that are one type of fault manifestation. Furthermore, many circuits can have a characteristic I_{DDQ} current when switching. Again, this characteristic current can change in response to defects. Detectable I_{DDQ} defects have no relation to the expected correct circuit outputs, which require that testing for I_{DDQ} detectable defects be supplemented with other testing approaches. Furthermore, an integrated circuit is generally integrated with other circuit technologies that draw significant quiescent currents, for example, IC pads, bipolar, and analog circuits. Thus, testing for I_{DDQ} defects requires that the supply for I_{DDQ} testable circuits be isolated from the supply for other parts of the circuit.

Deep Sub-Micron (DSM). Deep sub-micron (DSM) technologies offer the promise of increased circuit densities and speeds. For several reasons, the defect manifestations change with decreasing feature size (3) First, supply voltages are reduced along with an associated reduction in noise margins, which makes circuits more suscepitible to malfunctions caused by noise. Second, higher operating frequencies affect defect manifestations in many ways. Capacitive coupling increases with increasing operating frequency, which increases the likelihood of crosstalk. Furthermore, other errors may be sensitive to operating frequency and may not be detected if testing is conducted at slower frequencies. Third, leakage currents increase with decreasing feature size, which increases the difficulty of using tests to detect current anomalies. Fourth, increasing circuit density has resulted in an increase in the number of interconnect levels, which increases the likelihood of interconnect related defects. Although the classic stuck-at fault model was not conceived with these faults in mind, the stuck-at fault model does focus testing goals on controlling and observing circuits nodes, thereby detecting many interconnect faults that do not conform to the stuck-at fault model.

Measures of Testing. To gauge the success of a test methodology, some metric for assessing the test regimen and any associated overhead is needed. In this section, the measures of test set fault coverage, test set size, hardware overhead, performance impacts, testability, and computational complexity are presented.

Fault Coverage. Fault coverage, sometimes termed test coverage, is the percentage of targeted faults that have been covered by the test regimen. Ideally, 100% fault cover-

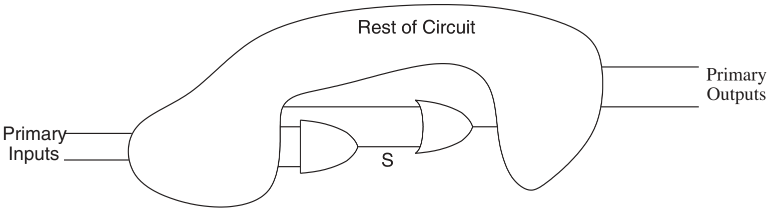


Figure 6. Representative circuit with fault.

age is desired; however, this statistic can be misleading when the fault model does not reflect the types of faults accurately that can be expected to occur (10). As noted earlier, the stuck-at fault model is a simple and popular fault model that works well in many situations. CMOS circuits, however, have several failure modes that are beyond the scope of the simple stuck-at fault model. Fault coverage is determined through fault simulation of the respective circuit. To assess the performance of a test, a fault simulator should model accurately the targeted fault to get a realistic measure of fault coverage.

Size of Test Set. The size of the test set is an indirect measure of the complexity of the test set. Larger test sets increase the testing time, which have a direct a direct impact on the final cost if expensive circuit testers are employed. In addition, the test set size is related to the effort in personnel required computationally to develop the test. The size of the test set depends on many factors including the ease with which the design can be tested as well as integrating DFT methodologies. Use of scan path approaches with flip-flops interconnected as shift registers gives excellent fault coverages, yet the process of scanning into and from the shift register may result in large test sets.

Hardware Overhead. The addition of circuity to improve testability through the integration of built-in test and built-in self test (BIST) capabilities increases the size of a system and can have a significant impact on circuit costs. The ratio of the circuit size with test circuitry to the circuit without test circuitry is a straightforward measure of the hardware overhead. If improved testability is a requirement, then increased hardware overhead can be used as a criterion for evaluating different designs. The additional hardware can simplify the test development process and enable testing for cases that are otherwise impractical, System failure rates are a function of the size and the complexity of the implementation, where ordinarily larger circuits have higher failure rates. As a result, the additional circuitry for test can increase the likelihood of system failure.

Impact on Performance. Likewise, the addition of test circuitry can impact system performance. The impact can be measured in terms of reduced clock rate, higher power requirements, and/or increased cost. For example, scan design methods add circuitry to flip-flops that can switch between normal and test modes and typically with have longer delays compared with circuits not so equipped. For devices with fixed die sizes and PLAs, the addition of test circuitry may displace circuitry that contributes to the functional performance.

Testability. Testability is an analysis and a metric that describes how easily a system may be tested for defects. In circuit defect testing, the goal is to supply inputs to the circuit so that it behaves correctly when no defects are present, but it malfunctions if a single defect is present. In other words, the only way to detect the defect is to force the circuit to malfunction. In general, testability is measured in terms of the specific and the collective observability and controllability of nodes within a design. For example, a circuit that provides the test engineer direct access (setting and reading) to flip-flop contents is estable more easily than one that does not, which gives a corresponding better testability measure. In the test community, testability often is described in the context of controllability and to observability. Controllability of a circuit node is the ability to set the node to a particular value. Observability of a circuit node is the ability to observe the value of the node (either complemented or uncomplemented) at the circuit outputs. Estimating the difficulty to control and to observe circuit nodes forms the basis for testability measures. Figure 6 presents a simple illustration of the problem and the process. The node S is susceptible to many types of faults. The general procedure for testing the correct operation of node S is to control the node to a value complementary to the fault value. Next, the observed value of the signal is propagated to system outputs for observation. Detecting faults in systems that have redundancy of any sort requires special consideration to detect all possible faults. For example, fault-tolerant systems that employ triple modular redundancy will not show any output discrepancies when one masked fault is present (2). To make the modules testable, individual modules must be isolated so that the redundancy does not mask the presence of faults. In addition, redundant gates necessary to remove hazards from combinational circuits result in a circuit where certain faults are untestable. Improved testability can be achieved by making certain internal nodes observable through the addition of test points. The Sandia controllability/observability analysis program is an example application that evaluates the testability of a circuit or system (11).

[2] For example, an exponential time algorithm may double the required resources when the problem size is increased by this result is much like what happens when you double the number of pennies on each successive the square of a chess board.

Computational Complexity. The computational complexity measures both the number of computations and the storage requirements necessary to achieve a particular algorithmic goal. From these measures, bound estimates of the actual amount of time required can be determined. In testing applications, the worst-case computational complexity for many algorithms used to find tests unfortunately is bad. Many algorithms fall in the class of NP-Complete algorithms for which no polynomial time (i.e, good) algorithm exists. Instead, the best known algorithms to solve NP-Complete problems require exponential time.[2] Although devising a perfect test is highly desirable, in practice, 100% coverage generally is not achieved. In most cases, tests are found more quickly than the worst case, and cases taking too long are stopped, which results in a test not being found. Some have noted that most tests are generated in a reasonable amount of time and provide an empirical rationale to support this assertion (12).

Fault Simulation

Fault simulation is a simulation capable of determining whether a set of tests can detect the presence of faults within the circuit. In practice, a fault simulator simulates the fault-free system concurrently with the faulty system. In the event that faults produce circuit responses that differ from the fault-free cases, the fault simulator records the detection of the fault.

To validate a testing approach, fault simulation is employed to determine the efficacy of the test. Fault simulation can be used to validate the success of a test regimen and to give a quantitative measure of fault coverage achieved in the test. In addition, test engineers can use fault simulationfor assessing functional test patterns. By examining the faults covered, the test engineer can identify circuit structures that have not received adequate coverage and can target these structures formore intensive tests.

To assess different fault models, the fault simulator should both model the effect of the faults and also report the faults detected. In the test for bridging faults detectable by I_{DDQ} testing, traditional logic and fault simulators are incapable of detecting such faults because these.faults may not produce a fault value that can differentiate faulty from fault-free instances. In Ref. 13, a fault simulator capable of detecting I_{DDQ} faults is described.

BASIC COMBINATIONAL ATG TECHNIQUES

In ATG, a circuit specification is used to generate a set of tests. In this section, several basic techniques for ATG are presented. The stuck-at fault model described previously provides the test objective for many ATG approaches. The single stuck fault is a fault on a node within the circuit that is either SA-0 or SA-l. Furthermore, only one fault is assumed to be in the circuit at any given time. Presented in detail here are algebraic approaches for ATG, Boolean satisfiability ATG, the D-Algorithm, one of the first ATG algorithms, and PODEM. Subsequent developments in ATG are compared largely with the D-Algorithm and other derived works.

ALGEBRAIC APPROACHES FOR ATG

Algebraic-techniques may be a used to derive tests for faults and can be used in ATG. The Boolean difference (2,14,15) is an algebraic method for finding a test should one exist. Given a Boolean function $F()$ the Boolean difference is defined as

$$\frac{dF(\mathbf{X})}{dx_i} = F(x_1, x_2, , x_{i1}, 0, x_{i+1}, , x_n) \oplus F \times (x_1, x_2, , x_{i1}, 1, x_{i+1}, , x_n) \tag{1}$$

where $\frac{dF(\mathrm{X})}{dx_i}$ is the Boolean difference of the Boolean function $F(\cdot)$, x_i is an input, and $\oplus$ is the exclusive-or. One interpretation of the quantity $\frac{dF(X)}{dx_i}$ is to show the dependence of F(X) on input x_i. If $\frac{dF(\mathrm{X})}{dx_i} = 0$, the function is independent of x_i, which indicates that it is impossible to find a test for a fault on x_i. On the other hand, if $\frac{dF(\mathrm{X})}{dx_i} = 1$, then the output depends on x_i, and a test for a fault on x_i can be found.

The Boolean difference can be used to determine a test because it can be used in an expression that encapsulates both controllability and observability into a Boolean tautology that when satisfied, results in a test for the fault

$$x_i \frac{dF(\mathbf{X})}{dx_i} = 1 \tag{2}$$

for X_i—SA — faults and

$$\bar{x}_i \frac{dF(\mathrm{X})}{dx_i} = 1 \tag{3}$$

for X_i—SA — 1 faults. Note that the Boolean difference represents the observability of input x_i, and the assertion associated with x_i represents its controllability. Equations (2) and (3) can be reduced to SOP or POS forms. A suitable assignment of inputs that satisfies the tautology is the test pattern. Finding a suitable test pattern is intractable computationally if product and sum terms have more than two terms (16).

Boolean Satisfiability ATPG

Boolean satisfiability SAT-ATPG (17) is related to the Boolean difference method for determining test patterns. As in the Boolean difference method, SAT-ATPG constructs the Boolean difference between the fault free and the faulty circuits. Rather than deriving the formula to derive a test, SAT-ATPG creates a satisfiability problem such that the variable assignments to achieve satisfiability are a test for the fault. The satisfiability problem is derived from the combinational circuit by mapping the circuit structure into a directed acyclic graph (DAG). From the DAG, a formula in conjunctive normal form (CNF) is derived that when satisfied, produces a test for a fault in the circuit. Although the SAT problem is NP-Complete (16), the structure of the

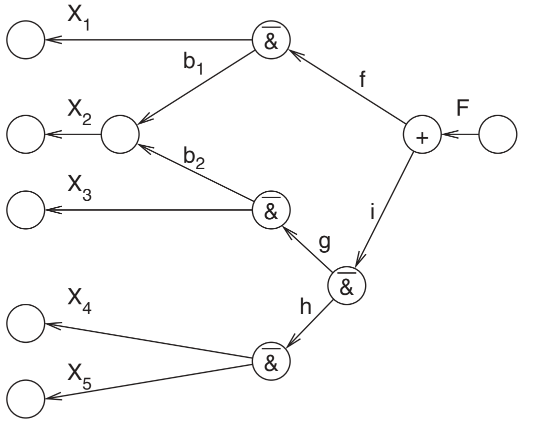


Figure 7. Example combinational circuit.

resulting formula has specific features that tend to reduce the computational effort compared with the general SAT problem. Indeed, the resulting CNF formula is characterized by having a majority of the factors being two terms. Note that satisfiability of a CNF formula with two terms (2SAT) is solvable in linear time. Reference 17 notes that as many as 90% of the factors are two elements. This structure suggests strongly that satisfiability for expressions that results from this construction are solvable in a reasonable amount of time, but the are not guaranteed in polynomial time. The basic SAT-ATPG is described in more detail in the following paragraphs.

As an example, consider the circuit example used to illustrate the D-Algorithm in Fig. 7. The DAG derived is given in Fig. 8. The mapping of the circuit to the DAG is straightforward with each logic gate, input, output, and fanout point mapping to a node in the DAG. Assuming inputs X, Y, and output Z, the conjunctive normal forms for the NAND and OR gates are (17).

$$\begin{array}{l}(Z+X)(Z+Y)(Z+\bar{X}+\bar{Y})\ \text{NAND}\\ (Z+\bar{X})(Z+\bar{Y})(\bar{Z}+X+Y)\ \text{OR}\end{array} \tag{4}$$

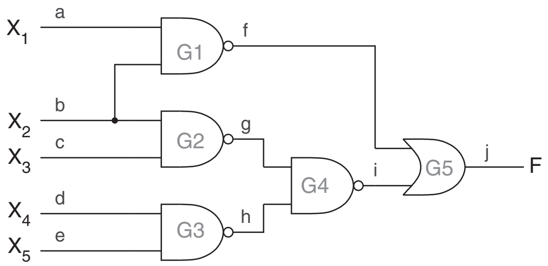


Figure 8. DAG for circuit from Fig. . Logic gate nodes labeled to show original circuit functions.

CNF formulas for other gates and the procedure for handling gates with three or more inputs are also summarized in Ref. 17. The CFN for the fault-free and faulty circuits are derived from the DAG. The exclusive-or of the outputs of fault-free and faulty circuits must be (1) to distinguish between the two expressed as

$$F_{\text{faulty}} \oplus F_{\text{fault - free}} = 1 \tag{5}$$

where satisfication of the tautology results in a test for the desired fault. Starting with the output, the conjunction of all nodes is formed following the edges of the DAG. The fault-free CNF for the circuit is the conjunction of the following conjunctive normal forms for the various circuit structures

$$\begin{array}{ll}(F+\bar{f})(F+\bar{i})(\bar{F}+f+i) & \text{G5}\\ (f+X_1)(f+b_1)(f+\bar{X}_1+\bar{b}_1) & \text{G1}\\ (i+g)(i+h)(i+\bar{g}+\bar{h}) & \text{G4}\\ (g+b_2)(g+X_3)(g+\bar{b}_2+\bar{X}_3) & \text{G2}\\ (\bar{X}_2+b_1)(X_2+\bar{b}_1) & \text{Top fan - out at b}\\ (\bar{X}_2+b_2)(X_2+\bar{b}_2) & \text{Bottom fan - out at b}\\ (h+X_4)(h+X_5)(h+\bar{X}_4+\bar{X}_5) & \text{G3}\end{array} \tag{6}$$

The CNF for the i-SA-0 fault requires a modification to the circuit DAG to represent the presence of the fault. From the DAG structure derived from the circuit and the target fault, a CNF formula is derived. A CNF that represents the fault test is formed by taking the exclusive-or of the fault free circuit with a CNF form that represents the circuit with the fault. The CNF for the faulted circuit is derived by modifying the DAG by breaking the connection at the point of the fault and adding a new variable to represent the fault. The variable $\bar{i}'$ is used to representthe fault i-SA-0. Note that in the faulty circuit, the combinational circuit determining i in the fault-free circuit is redundant, and the CNF formula for the faulted circuit is

$$\begin{array}{ll}(F'+\bar{f})(F'+i')(\bar{F}'+f+\bar{i}') & \text{G5}\\ \bar{i}' & \text{The fault}\\ (f+X_1)(f+b_1)(f+\bar{X}_1+\bar{b}_1) & \text{G1}\\ (\bar{X}_2+b_1)(X_2+\bar{b}_1) & \text{Top fan - out at b}\end{array} \tag{7}$$

Combining Equations (6) and (7), the exclusive-or of the faulty and fault-free formulas, and eliminating redundant terms gives the following formula whose satisfaction is a test for the fault

$$\begin{array}{l}(F+\bar{f})(F+\bar{i})(\bar{F}+f+i)\\ (f+X_1)(f+b_1)(f+\bar{X}_1+\bar{b}_1)\\ (i+g)(i+h)(i+\bar{g}+\bar{h})\\ (g+b_2)(g+X_3)(g+\bar{b}_2+\bar{X}_3)\\ (\bar{X}_2+b_1)(X_2+\bar{b}_1)\\ (\bar{X}_2+b_2)(X_2+\bar{b}_2)\\ (h+X_4)(h+X_5)(h+\bar{X}_4+\bar{X}_5)\\ \bar{i}'\\ (\bar{F}+F'+BD)(F+\bar{F}'+BD)(\bar{X}+\bar{X}'+\bar{B}\bar{D}(X+X'+\bar{B}\bar{D}))\end{array} \tag{8}$$

Note that the last line of Equation (8) represents the exclusive-or for the faulty and the fault-free circuits, and the variable BD is the output that represents the exclusive-

or of the two. Significantly, most terms in Equation (8) have two or fewer terms.

The next step is to determine an assignment that satisfies Equation (8). The problem is broken into two parts where one part represents the satisfaction of the trinary terms and the second the satisfaction of binary terms (solvable in polynomial time) that are consistent with the trinary terms. Efforts that followed (17) concentrated on identifying heuristics that improved the efficiency of finding assignments.

D-Algorithm

The D-Algorithm (18) is an ATG for combinational logic circuits. Furthermore, the D-Algorithm was the first combinational ATG algorithm to guarantee the ability to find a test for a SA-0/1 fault should a test exist. In addition, the D-Algorithm provides a formalism for composing tests for combinational circuits constructed modularly or hierarchically. The D-Algorithm relies on a five-valued Boolean algebra to generate tests, which is summarized in Table 2. Note that the values D and $\bar{D}$ represent a discrepancy between the fault free and faulty signal values where these values can be either the seminal error of the fault or the discrepancy attributable to the fault that has been propagated through several layers of combinational logic. The D-AIgorithm also requires two additional assumptions. First, exactly one stuck-at fault may be present at any given time. Second, other than the faulted node, circuit structures are assumed to operate fault free (i.e., normaly).

To begin the algorithm, the discrepancy that represents the direct manifestation of the fault is assigned to the output of a primitive component. For this component, the input/output combination that forces the manifestation of the fault is called the primitive D cube of failure (PDCF). The PDCF provides a representation of the inputs necessary to result in discrepancies for the faults of interest. The effect of the fault is propagated through logic circuits using the PDC for each circuit. The application of PDCs continues with primitive elements until the discrepancy is propagated to one or more primary outputs. Next, the inputs are justified through a backward propagation step using the singular cover for each component in the backward path. The singular cover is a compressed truth table for the fault-free circuit. Singular covers, PDCFs, and PDCs for several basic gates are shown in Table 3. Note that the PDCFs and PDCs follow straightforwardly from the logic functions and the five valued Boolean logic summarized in Table 2. Theoretic derivation of these terms are presented in Ref. 18.

The D-Algorithm consists principally of two phases. The first phase is the D-drive phase, where the fault is set through the selection of an appropriate PDCF and then propagated to a primary output. Once the D-drive is complete, justification is performed. Justification is the process of determining signal values for internal node and primary inputs consistent with node assignments made during D-drive and intermediate justification steps. In the event a conflict occurs, where at some point a node must be both 0 and 1 to satisfy the algorithm, backtracking occurs to the various points in the algorithm where choices for assignments were possible and an alternate choice is made. An input combination that propagates the fault to the circuit outputs and can be justified at the inputs is a test pattern for the fault. To generate a test for a combinational logic circuit, the D-Algorithm is applied for all faults for which tests are desired.

Table 2. Boolean values

Value	Meaning
1	Logic one
0	Logic zero
D	Discrepancy: expected one, but is zero due to fault
D	Discrepancy: expected zero, but is one due to fault
X	Don’t care, could be either 0 or 1

Table 3. Singular covers, PDCFs, and PDCs for several basic gates

Gate	Singu9lar cover	PDCF	PDC

The D-Algorithm is applied to the circuit given in Fig. 8. For an example demonstration, consider the fault i-SA-0. Figure 9 gives a summary of the algorithmic steps that results in the determination of the test pattern. The resulting test pattern for the example in Fig. 9 is $X_1X_2X_3X_4X_5 = 111XX$, where X is as defined in Table 2. Either fault simulation can be used to identify other faults detected by this test, or the “don’t cares” can be used to combine tests for two or more different faults.

Path-Oriented Decision Making (PODEM)

The D-Algorithm was pioneering in that it provided a complete algorithmic solution to the problem of test pattern generation for combinational circuits. In the years after it was introduced, researchers and practitioners noticed that the D-Algorithm had certain undesirable asymptotic properties and in the general case was found to be NP-Complete (10). This term means the worst case performance is an exponential number of steps in the number of circuit nodes. Despite this finding for many types of problems, the D-Algorithm can find tests in a reasonable amount of time. In Ref. 17, it was noted that the D-Algorithm was particularly inefficient in determining tests for circuit structures typically used in error correcting circuits (ECC). Typically, ECC circuits have a tree of exclusive-OR gates with reconvergent fanout through two separate exclusive-OR trees. The path-oriented decision making (PODEM) test pattern generation algorithm (20) was proposed to speed the search for tests for circuits similar to and used in ECC. In fact, the researchers learned that their approach was in general as effective and more efficient computationally compared with the D-Algorithm.

PODEM is fundamentally different from the D-Algorithm in that test searches are conducted on primary inputs. As a result, the amount of backtracking that might occur is less than the D-Algorithm because fewer places exist where backtracking can occur. Furthermore, the

	Node										
TC	a	b	c	d	e	f	g	h	i	j	Note
1							$\boxed{0}$	$\boxed{X}$	$\boxed{\overline{D}}$		PDCF
2						$\boxed{0}$	0	X	$\overline{D}$	$\boxed{\overline{D}}$	D-Drive
3	$\boxed{1}$	$\boxed{1}$				0	0	X	$\overline{D}$	$\overline{D}$	Justify
4	1	1	$\boxed{1}$			0	0	X	$\overline{D}$	$\overline{D}$	Justify
5	1	1	1	$\boxed{X}$	$\boxed{X}$	0	0	X	$\overline{D}$	$\overline{D}$	Justify

Figure 9. The D-algorithm, step by step. Values in boxes show work for a specific step.

backtracking is simpler computationally. PODEM works by selecting a fault for evaluation and then choosing the inputs one at a time to determine whether the input combination serves as a test pattern. The evaluation process is based on the same five valued logic family used in the D-Algorithm. The algorithm searches for a discrepancy between the good and the faulty circuits. An example decision tree is shown in Fig. 10. The decision tree shows a record of the process for finding a test to detect the i-SA-0 fault. The number of possible nodes in the decision tree are $2^{N+1} - 1$ where each node identifies a possible test. In the worst case, the PODEM algorithm will visit each node in the search tree. A simple search process is employed in this example where each input is assigned trial values in a sequence. The search proceeds for successive inputs from each trial assignment resulting in either the acceptance of this assignment in a test for the desired fault or a rejection because the trial assignment cannot result in a test. The first trial value assigned to an input is 0, and in the event a test is not possible, the trial input is 1. Given this simple structure, the test $X_1X_2X_3X_4X_5 = 11011$ results. Heuristics can be employed to improve the search by taking into account the structure of the circuit (20). For example, if the trial input for $X_3 = 1$, the test $X_1X_2X_3X_4X_5 = 111XX$ results after only three iterations of the search.

SEQUENTIAL ATG TECHNIQUES

Because most interesting digital systems are sequential, sequential ATG is an important aspect of the test generation process. For the purposes of this section, we will assume clocked sequential circuits that conform the structure shown in Fig. 11. The combinational logic in the state machine determines the next state and the output function. A very significant difference between combinational circuits and sequential machines is that the latter have memory elements isolating circuit nodes to be neither directly controllable at the inputs nor observable at the outputs. As a result, the ATG process must be more sophisticated compared with those used in combinational circuits.

A synchronous counter is an example of a simple sequential circuit that also demonstrates some complexities in devejqping defect tests for sequential circuits. Consider a synchronous counter that can be loaded synchronously with an initial count and has one output that is asserted when the counter is at its terminal count. One testing approach is to load the smallest initial count and then clock until the count rolls over after 2^N counts. For long counters, this exhaustive testing of the count function is complete but excessively time consuming. Ad hoc techniques can be employed to devise a test. For example, loading initial counts at selected values that focus on exercising the carry chain within the logic of the counter are very effective because of the regular structure present in most counters. Understanding the intimate structure and the function of the sequential machine produces an effective test. Less-structured sequential circuits are more problematic because tests for specific cases may require more intricate initialization and propagation sequences.

Introduction to Deterministic ATPG for Synchronous Sequential Machines

In this subsection, we will make several observations about ATG in sequential circuits. Clearly, faults will result in incorrect and undesirable behaviors, but the effect of the faulted node may not be observable immediately. In other words, an error is latent for some period of time before it can be detected. Likewise, a faulted node may not be controllable immediately. Consider the simple five state counter, shown in Fig. 12 with an enable input and a terminal count as the output.

After reset, all flip-flop states are set to zero. When enabled, the fault-free counter will cycle through the five

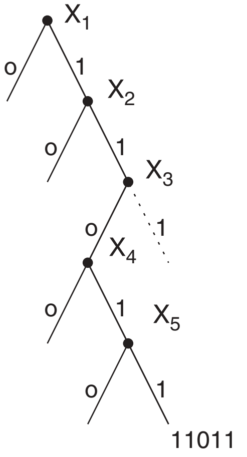


Figure 10. A PODEM decision tree for circuit from Fig. 8.

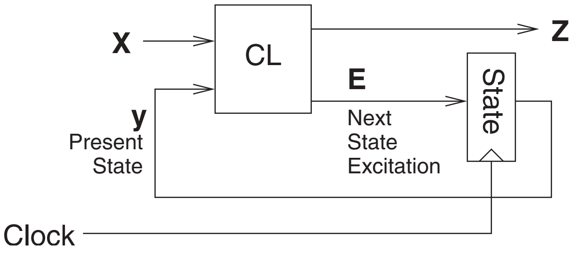


Figure 11. General model of sequential circuit.

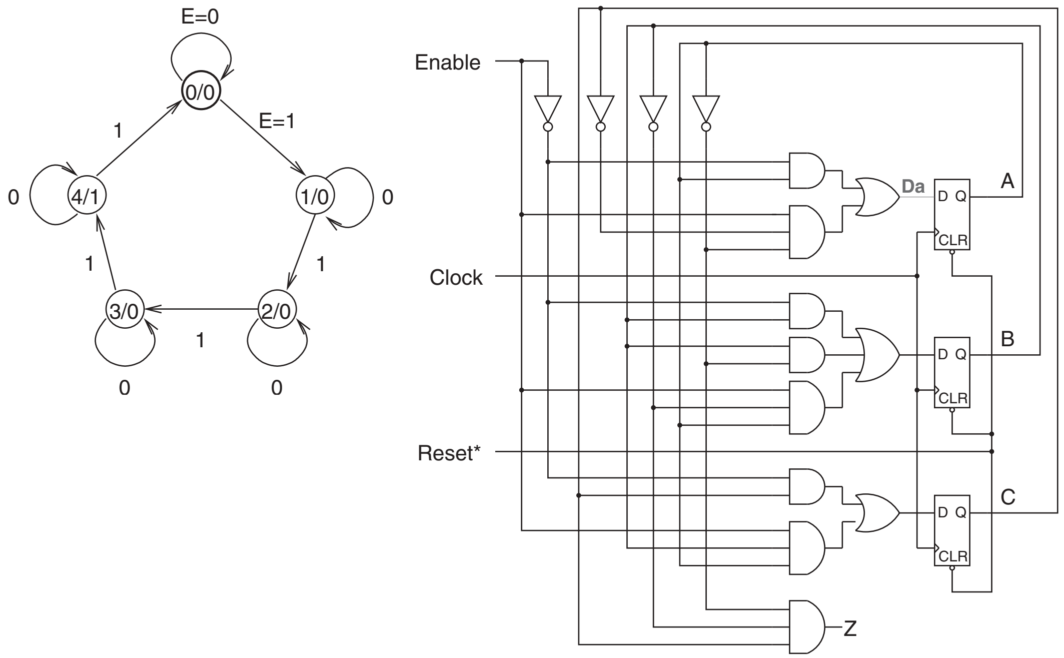
count states. Consider what happens when Da SA-1. With each state update, the fault may cause incorrect operation because discrepancies caused by the fault can be stored in the state. In addition, in subsequent clocks, additional discrepancies can be introduced whereas prior stored discrepancies can be recycled. Indeed, under certain circumstances, faulty behavior may disappear momentarily. As a result of the fault, the state machine operation changes and can be viewed as a change in the state machine operation as shown in Fig. 13.

Using the discrepancy notation from the D Algorithm, the state machine passes through the following states in successive clocks as shown in Fig. 14. Inspecting the state sequencing, $CBA = \{000, 001, 01\bar{D}, \bar{D}D1, D\bar{D}\bar{D}, \bar{D}0\bar{D}, 0\bar{D}1, \bar{D}\bar{D}\bar{D}, 011, 10\bar{D}, 0\bar{D}\,\bar{D}, \bar{D}01, 01\bar{D}\}$ we see that the pattern repeats every eight clocks compared with the expected four clocks. Furthermore, the effect of the fault can be latent because discrepancies are stored in the state and not observable immediately. Because multiple discrepancies can be present simultaneously, the circuit may occasionally show correct operation, for example at T8, despite the occurrence of several discrepancies in prior times. In this example, if the state is observable, the fault can be detected at T2. Suppose that only the inputs and the clock can be controlled and only the output, Z, can be observed. In this case, the detection of the fault is delayed for two clock cycles, until T4.

Synchronous ATG is challenging, but it can be understood and in some cases solved using concepts from combinational ATG methods. Indeed, this leads to one of three approaches. First, tests can be created in an ad hoc fashion. Then, using a technique called fault simulation similar to that presented in Fig. 14, detected faults can be tabulated and reported. Second, the circuit can be transformed in specific ways so that combinational ATG can be applied directly. Such approaches are presented in the following subsections. Third, the circuit operation can be modeled so that the circuit operation gives the illusion of a combinational circuit, and it allows combinational ATG to be used. Indeed, any of these three techniques can be used together as testing needs dictate.

Iterative Array Models for Deterministic ATPG

Clearly, ATG for sequential circuits is more challenging compared with combinational circuits. Many sequential ATG techniques rely on the ability to model a sequential machine in a combina-tional fashion, which makes possible the use of combinational ATG in sequential circuits. Analytical strategies for determining test patterns use iterative array models (19). The iterative array models provide a theoretic framework for making sequential machines seem combinational from the perspective of the ATG process. Iterative arrays follow from unrolling the operation of the



(a) State Diagram

(b) Implementation

Figure 12. Five state counter.

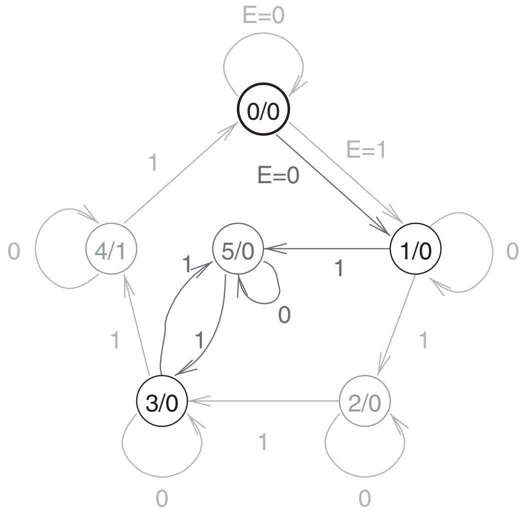


Figure 13. Effect of faults on state diagram.

state machine where each unrolling exposes another time frame or clock edge that can update the state of the state machine. The unrolling process replicates the excitation functions in each time frame, applying the inputs consistent with that particular state. Memory elements are modeled using structures that make them seem combinational. Furthermore, output functions are replicated as well. Each input and output is now represented as a vector of values where each element gives the value within a specific time frame. With this model, combinational ATG methods can be used to determine circuits for specific times.

Iterative array methods complicate the application of sequential ATG methods in our major ways: (*1*) the size of the "combinational circuit" is not known, (*2*) the state contribution is a constrained input, (*3*) multiple iterations express faults multiple times, and (*4*) integration level issues complicate when and how inputs are controlled and outputs are observed. For test pattern generation, the concatenation of the inputs from the different time frames, $\cdots X^{t1}X^{t}X^{t+1}\cdots$, serve as the inputs for the combinational ATG algorithms, where for the actual circuit, the superscript specifies the time frame when that input is set to the required value.. The combinational ATG algorithm outputs are the concatenation of the frame outputs, $\cdots Z^{t1}Z^{t}Z^{t+1}\cdots$, where again, the superscript identifies when an output should be observed to detect a particular fault (Fig. 15).

The size of the circuit is determined by the initial state and the number of array iterations necessary to control and to observe a particular fault. In all iterations, the iterative model replicates combinational circuitry and all inputs and outputs for that time step. The ATG generation process will, by necessity, be required to instantiate, on demand, combinational logic iterations because the required number of iterations necessary to test for a specific fault is not known. For some initial states, it is impossible to determine a test for specific faults because multiple expressions of a fault may mask its presence. Different algorithms will assume either an arbitrary initial state or a fixed state.

From the timing diagram in Fig. 13, the iterative circuit model for the counter example with a Da-SA-1 fault is given in Fig. 16. A discrepancy occurs after the fourth clock cycle, where the test pattern required to detect the fault is $Dc_0Db_0Da_0E_0E_1E_2E_3Z_0Z_1Z_2Z_3 = (00000000001)$, where the input pattern is the concatenation of initial state $Dc_0Db_0Da_0$, the inputs at each time frame $E_0E_1E_2E_3$, and the outputs at each time frame are $Z_0Z_1Z_2Z_3$. Because circuit in Fig. 16 is combinational, algorithms such as the D-Algorithm and PODEM can be employed to determine test patterns. Both algorithms would require modification to handle both variable iterations and multiple faults.

From the iterative array model, two general approaches can be pursued to produce a test for a fault. The first approach identifies a propagation path from the point of the fault and the output. From the selected output, reverse time processing is employed to sensitize a path through the frame iterations to the point of the fault. In the event a path is not found, backtracking and other heuristics are employed. After the propagation path for the fault is established, reverse time processing is used to justify the inputs required to express the fault and to maintain consistency with previous assignments. The second approach employs forward processing from the point of the fault to propagate to the primary outputs and then reverse time processing to justify the conditions to express the fault and consistency (21,22). Additional heuristics are employed to improve the success and the performance of the test pattern generation process, including using hints from fault simulation (22). Several approaches can be applied for determining test patterns.

Genetic Algorithm-Based ATPG

Because of the challenges of devising tests for sequential circuits, many approaches have been studied. Genetic algorithms (GAs) operate by generating and evaluating popu-

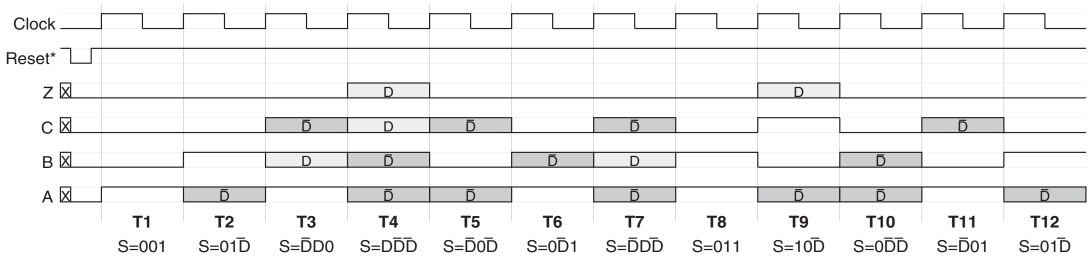


Figure 14. Fault simulation of counter with Da-SA-1.

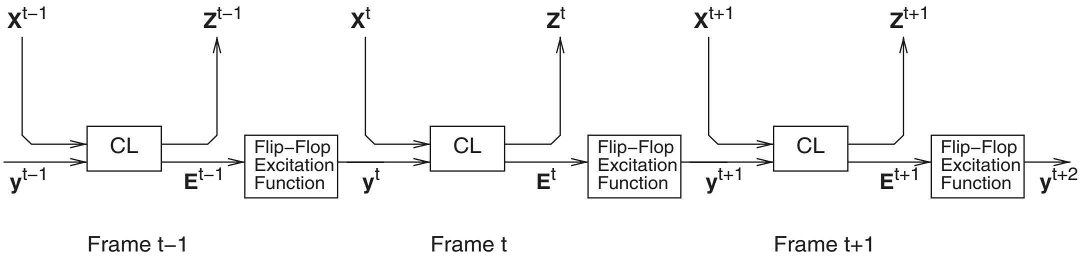


Figure 15. Excerpt from iterative array sequential circuit model.

lations of organisms for fitness to a particular purpose and then using the fitness to identify organisms from which the next generation is created (23). Organism structure and capabilities are defined by a genetic code represented as a string of values that set the configuration of the organism. The organism constructed from the genetic code is then evaluated for fitness. Organisms in the next generation are generated from fit organisms in the current generation using simulated variations of crossover and mutation.

In the context of a DUT, the string of test patterns can serve as the genome for the organism and the fitness can be a function of whether and how well a particular fault is detected from the sequence. The work of Hsiao et al. (24) used GAs to assemble test sequences for faults from promising pieces. In their work, the GA operates in two phases, the first targets controlling the fault and the second targets propagating the faulted node to the primary outputs. Sequences from which a test is assembled are from one of three categories: (*1*) distinguishing sequences, (*2*) set/clear sequences, and (*3*) pseudoregister justification sequences. Distinguishing sequences propagate flip-flop fault effects to the primary outputs. Set/clear sequences justify (i.e., force flip-flop) to specific states. Finally, pseudoregister justification sequences put sets of flip-flops into specific states. These sequences form the basis from which tests are generated for a particular fault. The process is initialized in some random fashion with a set of strings that represent the initial GA generation of test sequences. The likelihood that any given string can detect the fault depends on the complexity and the size of the circuit. Assuming the circuit is sufficiently complex that no organism can detect the fault, the fitness function must include the ability to identify organisms that have good qualities (i.e., have sequencing similar to what one might expect of the actual test sequence). Two fitness functions were formed in Ref. 24 for the justification and the propagation phases. Each was the weighted sum of six components that include the ability to detect the fault, measures of controllability, distinguishing measures, circuit activity measures, and flip-flop justification measures. The GA test generation operates in three stages where the GA is run in each stage until the fault coverage plateaus. In addition, test lengths are allowed to grow in subsequent stages, on the assumption that faults not detectable with shorter sequences may be detected with longer sequences.

DESIGN FOR TESTABILITY

Many ATG techniques provide capabilities that are practical in a variety of situations. DFT is the process and the discipline for designing digital systems to make them easier to test. Furthermore, constructing circuits that facilitate

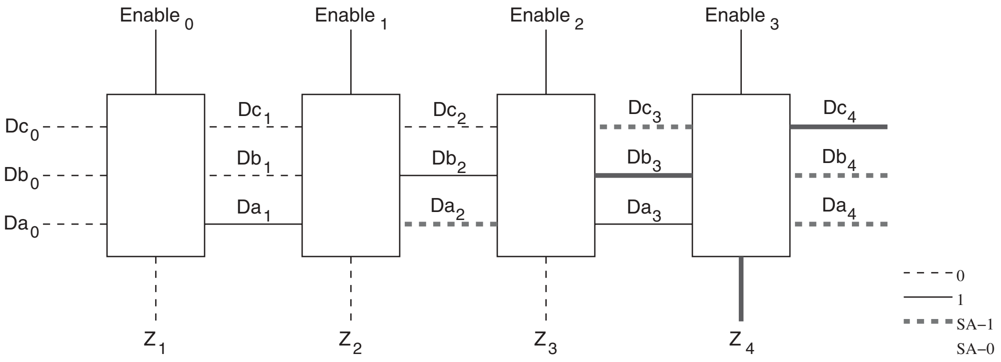


Figure 16. Iterative array model for example circuit for Da-SA-1.

testing also simplifies ATG. In this capacity, DFT can affect the entire testing process, from ATG to testing. DFT techniques apply other concepts and design paradigms including circuit fault models and techniques for detecting faults. Modifying designs themselves to improve and to simplify the testing process can offer many benefits. First, time devoted to test development is reduced with a higher likelihood of guaranteed results. Second, DFT reduces the test set size because individual tests are generally more effective. Third, DFT can reduce the time necessary to test a system. DFT approaches fall into one of two categories. Starting with a traditional design, the first category is exemplified by adding circuit structures to facilitate the testing process. The second category attacks the testing process from basic design principals resulting in systems that are inherently easy to test. In this section, DFT approaches that support ATG are presented.

Circuit Augmentation to Facilitate ATG

Digital systems designed in an ad hoc fashion may be difficult to test. Several basic techniques are available to improve the testability of a given system. These techniques include test point insertion, scan design methods, and boundary scan techniques. These approaches preserve the overall structure of the system.

Test Point Insertion. Test point insertion is a simple and straightforward approach for providing direct controllability and observability of problem circuit nodes. In test point insertion, additional inputs and outputs are provided to serve as inputs and outputs for testing purposes. These additional test point inputs and outputs do not provide any additional functional capabilities. The identification of these points can follow from a testability analysis of the entire system by identifying difficult-to-test internal nodes. Circuit nodes are selected as test points to facilitate testing of difficult-to-test nodes or modules. As test points are identified, the testability analysis can be repeated to determine how well the additional test points improve testability and to determine whether additional test points are necessary. Indeed, the test points enhance clearly the efficacy of ATG (25). Furthermore, test point insertion can provide the basis for structured design for testability approaches. The principal disadvantage of adding test points is the increased expense and reduced performance that results from the addition of test points. One study showed (26) that adding 1% additional test points increases circuit overheads by only 0.5% but can impact system performance by as much as 5%. In the study, test points were internal and inserted by state of the art test point insertion software.

Scan Design Methods. The success of test point insertion relies on the selection of good test points to ease the testing burden. Determining these optimal points for.test point insertion can be difficult. As a result, structured approaches can be integrated to guarantee ATG success. One particularly successful structured design approach is scan design. Scan design methods improve testability by making the internal system state both easily controllable and observable by configuring, in test mode, selected flip-flops as a shift register (27–30), effectively making each flip-flop a test point. Taken to the logical extreme, all flip-flops are part of the shift register and are therefore also test points. The power of this configuration is that the storage elements can be decoupled from the combinational logic enabling the application of combinational ATG to generate fault tests. The shift register organization has the additional benefit that it can be controlled by relatively few inputs and outputs. Because a single shift register might be excessively long, the shift register may be broken into several smaller shift registers.

Historically, scan path approaches follow from techniques incorporated into the IBM System/360 where shift registers were employed to improve testability of the system (27). A typical application of a scan design is given in Fig. 17. Note the switching of the multiplexer at the flip-flop inputs controls whether the circuit is in test or normal operation. Differences between different scan design methods occur in the flip-flop characteristics or in clocking.

Scan Path Design. Relevant aspects of the design include the integration of race free D flip-flops to make the flip-flops fully testabl (29). Level sensitive scan design (28) is formulated similarly. One fundamental difference is the machine state is implemented using special master slave flip-flops clocked with non overlapping clocks to enable testing of all stuck faults in flop-flops.

Boundary Scan Techniques. In many design approaches, the option of applying design for testability to some components is impossible. For example, standard parts that might be used in printed circuit boards that are not typically designed with a full system scandesign in mind. As another example, more and more ASIC designs are integrated from cores, which are subsystems designed by third-party vendors. The core subsystems are typically processors, memories, and other devices that until recently were individual integrated circuits themselves. To enable testing in these situations, boundary scan methods were developed. Boundary scan techniques employ shift registers to achieve controllability and observability for the inputs/outputs to circuit boards, chips, and cores. An important application of boundary scan approaches is to test the interconnect between chips and circuit boards that employ boundary scan techniques. In addition, the boundary scan techniques provide a minimal capability to perform defect testing of the components at the boundary. The interface to the boundary scan is a test access port (TAP) that enables setting and reading of the values at the boundary. In addition, the TAP may also allow internal testing of the components delimited by the boundary scan. Applications of boundary scan approaches include BIST applications (31), test of cores (32), and hierarchical circuits (33). The IEEE (Piscatouoaes NJ) has created and approved the IEEE Std 1149.1 boundary scan standard (34). This standard encourages designers to employ boundary scan techniques by making possible testable designs constructed with subsystems from different companies that conform to the standard.

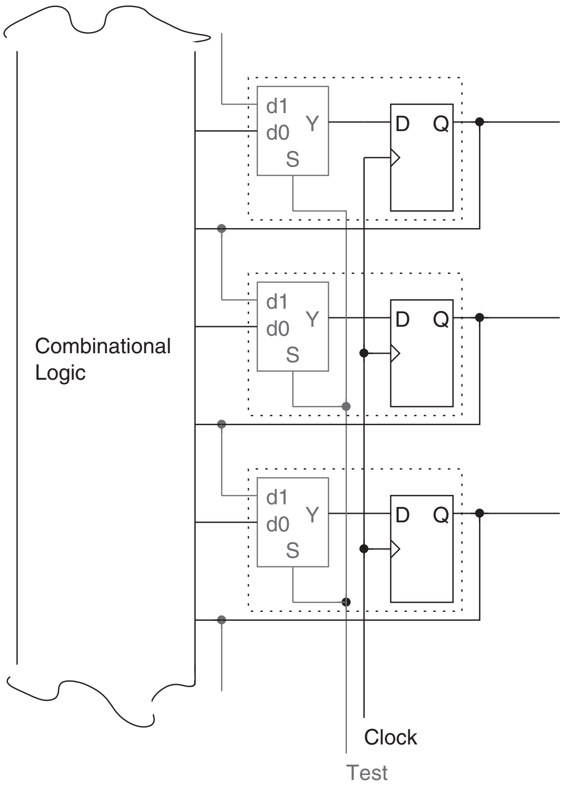


Figure 17. One scan path approach.

ATG and Built-In Test

Background. Requiring built-in tests affects test generation process in two ways. First, the mechanism for generating test patterns must be self-contained within the circuitry itself. Although in theory, circuitry can be designed to produce any desired test pattern sequence, in reality, most are impractical. As a result, simpler circuitry must be employed to generate test patterns. ATG and built-in tests require the circuits to have the ability to generate test sequences and also to determine that the circuits operate correctly after being presented with the test sequence.

Three classes of circuits are typically employed because the patterns have good properties and also have the ability to produce any sequence of test patterns. The first type of circuit is a simple N-bit counter, which can generate all possible assignments to N bits. Counter solutions may be impractical because test sequences are long for circuits with a relatively few inputs and also may be ineffective in producing sequences to test for delay or CMOS faults. Researchers have investigated optimizing count sequences to achieve more reasonable test lengths (35). The second type of circuit generates pseudorandom sequences using linear feedback shift registers (LFSRs). For combinational circuits, as the number of random test patterns applied to the circuit increases, fault coverage increases asymptotically to 100%. Much research has been conducted in the development of efficient pseudorandom sequence generators. An excellent source on many aspects of pseudorandom techniques is Ref. 36. A third type of circuit is constructed to generate specific test patterns efficiently for specific types of circuit structures. In this case, the desired sequence of test patterns is examined and a machine is synthesized to recreate the sequenc. Memory tests have shown some success in using specialized test pattern generator circuits (37).

To determine whether a fault is present, the outputs of the circuit must be monitored and compared with expected fault-free outputs. Test pattern generation equipment solves this by storing the expected circuit outputs for a given sequence of inputs applied by the tester. As noted above, it may be impractical to store or to recreate the exact circuit responses. Alternate approaches employ duplication approaches (several are summarized in Ref. 2) where a duplicate subsystem guarantees the ability to generate correct circuit outputs, assuming a single fault model. A discrepancy between the outputs of the duplicated modules infers the presence of a fault. Although duplication often is used in systems that require tault tolerance or safety, duplication may be an undesirable approach in many situations. An alternative approach, signature analysis, compresses the circuit output responses into a single code word, a signature, which is used to detect the presence of faults. Good circuit responses are taken either from a known good circuit or more frequently from circuit simulations. A fault in a circuit would result in a signature that differs from the expected good signature with high probability.

Signature Analysis. The cost of testing is a function of many influences that include design costs, testing time, and

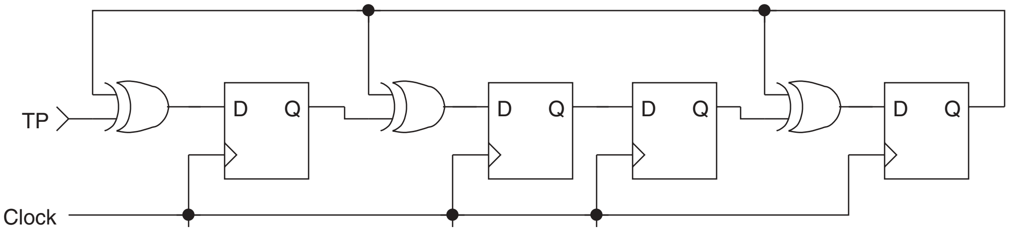
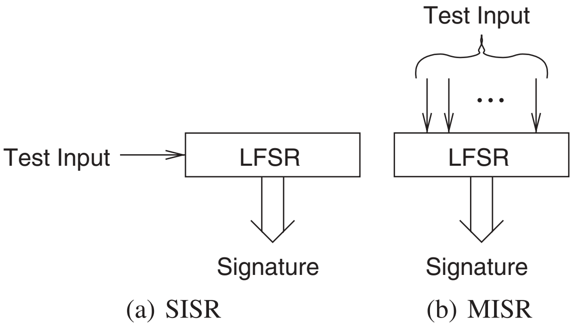


Figure 18. Linear feedback shift register.

test equipment costs. Thus, reducing the test set size and the ease of determining whether a fault is present has a great influence on the success of the system. In signature analysis, the test set is reduced effectively to one representative value, which is termed a signature. The signature comparison can be performed internally, using the same technology as the system proper. In signature analysis, the signature register is implemented as a LFSR as shown in Fig. 18. The LFSR consists of a shift register of length N, and linear connections fed back to stages nearer the beginning of the shift register. With successive clocks, the LFSR combines its current state with the updated test point values. Figure 19 shows two different LFSR configurations: (*1*) a single input shift register (SISR) compresses the results of a single test point into a signature and (*2*) a multiple input shift register (MISR) compresses several test point results. After the test sequence is complete, the contents of the LFSR is compared with the known good signature to determine whether faults are present. A single fault may result in the LFSR contents differing from the good signature, but generally will not provide sufficient information to identify the specific fault. Furthermore, a single fault may result in a final LFSR state that is identical to the good signature, which is termed allasing. This outcome is acceptable if aliasing occurs with low probability. In Ref. 36, the aliasing probability upper bounds were derived for signatures computed with SISRs. In addition in Ref. 30, methods for developing MISRs with no aliasing for single faults were developed. The circuitry necessary to store the signature, to generate a vector to compre with the signature, and to compare the signature is modular and simple enough to be intergrated with circuit functions of reasonable sizes, which makes signature analysis an important BIST technique. LFSRs can be used in signature analysis in several ways.



(a) SISR (b) MISR

Figure 19. Different signature register configurations.

BILBO. The built-in logic block observer (BILBO) approach has gained a fairly wide usage as a result of its modularity and flexibility (40). The BILBO approach can be used in both scan path and signature analysis test applications by encapsulating several important functions. BILBO registers operate in one of four modes. The first mode is used to hold the state for the circuitry as D flip-flops. In the second mode, the BILBO register can be configured as a shift register that can be used to scan values into the register. In the third mode, the register operates as a multiple input signature register (MISR). In the fourth mode, the register operates as a parallel random pattern generator. These four modes make possible several test capabilities. A four-bit BILBO register is shown in Fig. 20. One example application of BILBO registers is shown in Fig. 21. In test mode, two BILBO registers are configured to isolate one combinational logic block. The BILBO register at the input, R1, is configured as a PRPG, whereas the register at the output, R2, is configured as a MISR. In test mode operation, for each random pattern generated, one output is taken and used to compute the next intermediate signature in the MISR. When all tests have been performed, the signature is read and compared with the known good signature. Any deviation indicates the presence of faults in the combinational circuit. To test the other combinational logic block, the functions of R1 and R2 only need to be swapped. Configuration of the data path to support test using BILBO registers is best achtevedby performing register allocation and data path design with testability in mind (41).

Memory BIST

Semiconductor memories are designed to achieve the high storage densities in specific technologies. High storage densities are achieved by developing manufacturing processes that result in the replication, organization, and optimization of a basic storage element. Although in principle the memories can be tested as other sequential storage elements, in reality the overhead associated with using scan path and similar test approaches would impact severely the storage capacity of the devices. Furthermore, the basic function of a memory typically allows straightforward observability and controllability of stored information. On the other hand, the regularity, of the memory's physical structure and the requisite optimizations result in fault manifestations as a linkage between adjacent memory cells. From a testing perspective, the manifestation of the fault is a function of the state of a memory cell and its

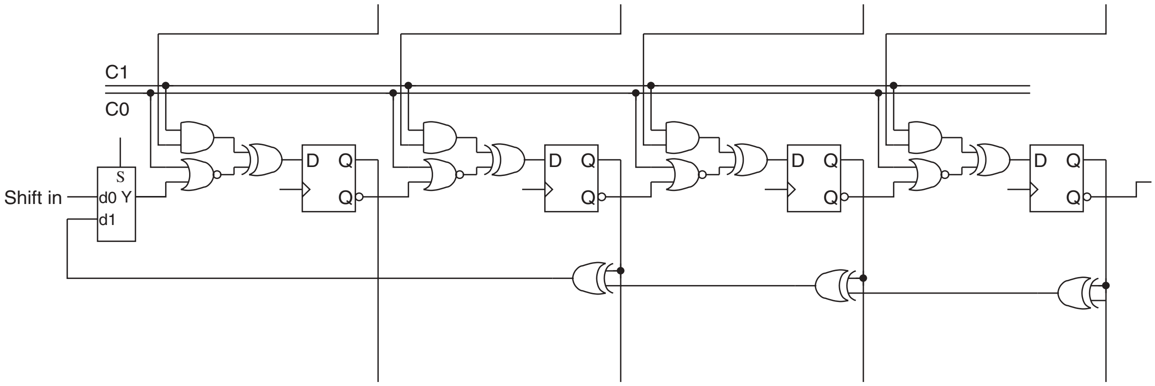


Figure 20. BILBO register.

physically adjacent memory cells. Among the test design considerations of memories is the number of tests as a function of the memory capacity. For example, a test methodology was developed (37) for creating test patterns. These test patterns could be computed using a state machine that is relatively simple and straightforward. The resulting state machine was shown to be implementable in random logic and as a microcode driven sequencer.

Programmable Devices

With seyeral vendors currently (2004) marketing FPGA devices capable of providing in excess of one million usable gates, testing of the programmed FPGA becomes a serious design consideration. Indeed, the capacity and performance of FPGAs makes this class of technology viable in many applications. Furthermore, FPGA devices are an integration of simpler programmable device architectures, each requiring its own testing approach. FPGAs include the ability to integrate memories and programmable logic arrays, which requires ATG approaches most suitable for that component. Summarized previously, PLAs exhibit fault models not observed in other implerneniation technologies. One approach for testing to apply BIST approaches described previously (42). PLA test can be viewed from two perspectives. First, the blank device can be tested and deemed suitable for use in an application. Second, once a device is programmed, the resulting digital system can be tested according to the jpejhods already described. One early and notable ATG methods applied to PLAs is PLATYPUS (43). The method balances random TPG with deterministic TPG to devise tests for both traditional stuck-at faults as well as cross-point faults. Modern FPGAs support standard test interfaces such as JTAG/ IEEE 1149 standards. In this context, ATG techniques can be applied in the context of boundary scan for the programmed device.

Minimizing the Size of the Test Vector Set

In the combinational ATG algorithms presented, specific faults are targeted in the test generation process. To

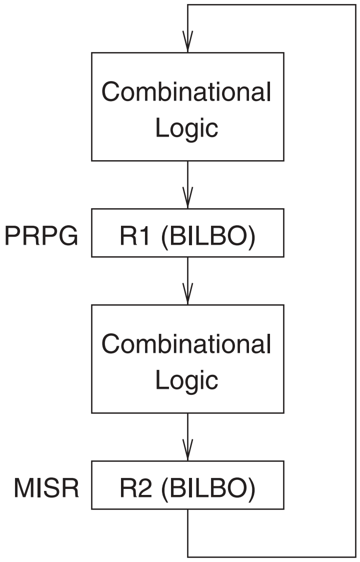


Figure 21. Application of BILBO to testing.

Table 4. Test pattern compaction

Fault	Test
a	00X0
b	001X
c	X00X
d	X011

develop a comprehensive test for a combinational circuit, one may develop a test for each faultlndividually. In most cases, however, the number of tests that results is much larger than necessary to achieve the same fault coverage. Indeed, the techniques of fault collapsing, test compaction, and fault simulation can produce a significant reduction in the test set size.

Fault Collapsing. Distinct faults in a circuit may produce the identical effect when observed at circuit outputs. As a result, the faults cannot be differentiated by any test. For example, if any AND gate input is SA-0, the input fault cannot be differentiated frogi thexiutout SA-0 fault. In this case, the faults can be collapsed into one, output SA-0, which requires a test only for the collapsed fault.

Test Compaction. The D-Algorithm and PODEM can generate test vectors with incompletely specified inputs, providing an opportunity to merge different tests through test compaction. For example, consider a combinational circuit whose tests are given in Table 4. Examining the first two faults in the table, *a*and *b*, shows the test 0010 will test for both faults. Test compaction can be either static or dynamic. In static test compaction, all tests are generated by the desired ATG and then analyzed to determine those tests that can be combined, off-line, thereby creating tests to detect several faults and specifying undetermined input values. In dynamic tests, after each new test is generated, the test is compacted with the cumulative compacted list. For the tests in Table 4, static compaction results in two tests that tests for all faults, 0000, and 0011 for faults {*a*,*c*} and {*b*,*d*} respectively. Dynamic compaction produces a different result, as summarized in Table 5. Note that the number of tests that result from dynamic compaction is more compared with static compaction. From a practical perspective, optimal compaction is computationally expensive and heuristics are often employed (1). Reference 9 also notes that in cases where heuristics are used in static compaction, dynamic compaction generally produces superior results while consuming fewer computational resources. Furthermore, dynamic compaction processes vectors as they come, but more advanced dynamic compaction heuristics may choose to not compact immediately but rather wait until a more opportune time.

Table 5. Simple dynamic test compaction

Sequence	New test	Compacted tests
1	00X0	{00X0}
2	001X	{0010}
3	X00X	{0010,X00X}
4	X011	{0010,X00X,X011}

Compaction Using Fault Simulation. A test that has been found for one fault may also test for several other faults. Those additional tests can be determined experimentally by performing a fault simulation for the test and identifying the additional faults that are also detected. The process is outlined as follows:

1. Initialize fault set
2. Select a fault from the fault set
3. Generate a test pattern for the selected fault
4. Run fault simulation
5. Remove additional detected faults form fault set
6. If fault set empty or fault coverage threshold met then exit, otherwise go to step 2

Test Vector Compression. Test vector compression takes on two forms, lossless and lossy. Lossless compression is necessary in circumstances where the precise input and output values must be known as what might be necessary on an integrated circuit tester. Under certain limited circumstances, lossless compression might make it possible to store test-compressed vectors in the system itself. In lossy compression, the original test vectors cannot be reconstituted. Lossy compression is used in such applications as pseudorandom number generation and signature registers, where input and output vectors are compressed through the inherent structure of the linear feedback shift registers as described previously. Lossy compression is suitable when the probability of not detecting an existing fault is much smaller than the proportion of uncovered faults in the circuit.

ATG CHALLENGES AND RELATED AREAS

As circuit complexity and capabilities evolve, so does the art and science of ATG. Technologic innovations have resulted in the ability to implement increasingly complex and innovative designs. The technologic innovations also drive the evolution of testing. One trend, for example, is that newer CMOS circuits have increased quiescent currents, which impacts the ability to apply I_{DDQ} testing technologies.

ATG in Embedded Systems

The natural evolution of technology advances and design has resulted in systems composed from IP obtained from many sources. Clearly, the use of IP provides developers faster development cycles and quicker entry to market. The use of IP presents several ATG challenges. First, the testing embedded IP components can be difficult. Second, because the internals of the IP components are often not known, the success of ATG techniques that require full knowledge of the circuit structure will be limited. Third, IP developers may be hesitant to provide fault tests for fear that it would give undesired insights into the IP implementation.

Functional ATG

Design tools and rapid prototyping paradigms result in the designer specifying hardware systems in an increasingly abstract fashion. As a result, the modern digital system designer may not get the opportunity develop tests based on gate level implementations. Employing ATG technologies relieves the designer of this task provided the design tools can define and analyze a gate level implementation.

SUMMARY

In this article, many aspects of ATG have been reviewed. ATG is the process of generating tests for a digital system in an automated fashion. The ATG algorithms are grounded in fault models that provide the objective for the test generation process. Building on fault models, ATG for combinational circuits have been shown to be effective. Sequential circuits are more difficult to test as they require the circuits to be unrolled in a symbolic fashion or be the object of specialized test pattern search algorithms. Because of the difficulties encountered in testing sequential circuits, the circuits themselves are occasionally modified to simplify the process of finding test patterns and to improve the overal test fault coverage. The inexorable technology progression provides many challenges in test testing process. As technology advances, new models and techniques must continue to be developed to keep pace.

BIBLIOGRAPHY

1. G. E. Moore, Cramming more components onto integrated circuits, *Electronics*, **38** (8): 1965.
2. B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Reading, MA: Addison-Wesley Publishing Company, 1989.
3. R. C. Aitken, Nanometer technology effects on fault models for IC testing, *Computer*, **32** (11): 47–52, 1999.
4. M. Sivaraman and A. J. Strojwas, *A Unified Approach for Timing Verification and Delay Fault Testing*, Boston: Kluwer Academic Publishers, 1998.
5. N. K. Jha and S. Kindu, *Testing and Reliable Design of CMOS Circuits*, Boston: Kluwer, 1992.
6. J. Gailay, Y. Crouzet, and M. Vergniault, Physical versus logical fault models in MOS LSI circuits: Impact on their testability, *IEEE Trans. Computers*, **29**(6): 286–1293, l980.
7. C. F. Hawkins, J. M. Soden, R. R. Fritzmeier, and L. K. Horning, Quiescent power supply current measurement for CMOS IC defect detection, *IEEE Trans. Industrial Electron.*, **36**(2): 211–218, 1989.
8. R. Dekker, F. Beenker, and L. Thijssen, A realistic fault model and test., algorithms for static random access memories, *IEEE Trans. Comp.-Aided Des.*, **9**(6): 567–572, 1996.
9. M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Ascataway, NJ: IEEE Press, revised printing edition, 1990.
10. N. K. Jha and S. Kindu, *Assessing Fault Model and Test Quality*, Boston: Kluwer, 1992.
11. L. H. Goldstein and E. L. Thigpen, SCOAP: Sandia controllability/observability analysis program, *Proceedings of the 17th Conference on Design Automation*, Minneapolis, MN: 1980, pp. 190–196.
12. V. D. Agrawal, C. R. Kime, and K. K. Saluja, A tutorial on built-in-self-test part 2: Applications, *IEEE Design and Test of Computers*, 69–77, 1993.
13. S. Chakravarty and P. J. Thadikaran, Simulation and generation of IDDQ tests for bridging faults in combinational circuits, *IEEE Trans. Comp.*, **45**(10): 1131–1140, 1996.
14. A. D. Friedman and P. R. Menin, *Fault Detection in Digital Circuits*. Englewood Cliffs, NJ: Prentice-Hall, 1971.
15. Z. Kohavi, *Switching and Finite Automata Theory*, 2nd ed.New York: McGraw-Hill, 1978.
16. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, CA: W.H. Freeman and Company, 1979.
17. T. Larrabee, Test pattern generation using Boolean satisfiability, *IEEE Trans. Comp. Aided Des.*, **5**(1): 4–15, 1992.
18. J. Paul Roth, Diagnosis of automata failures: A calculus and a method, *IBM J. Res. Devel.*, **10**: 277–291, 1966.
19. O. H. Ibarra and S. K. Sahni, Polynomially complete fault detection problems, *IEEE Trans. Computers*, **C-24**(3): 242–249, 1975.
20. P. Goel, An implicit enumeration algorithm to generate tests for combinational logic circuits, *IEEE Trans. Comp.*, **C-30**(3): 2l5–222, 1981.
21. I. Hamzaoglu and J. H. Patel, Deterministic test pattern generation techniques for sequential circuits, *DAC*, 2000, pp. 538–543.
22. T. M. Niermann and J. H. Patel, HITEC: A test generation package for sequential circuits, *Proceedings European Design Automation Conference*, 1990, pp. 214–218.
23. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison Wesley, 1989.
24. M. S. Hsiao, E. M. Rudnick, and J. H. Patel, Application of genetically engineered finite-state-machine sequences to sequential circuit TPGA, *IEEE Trans. Comp.-Aided Design of Integrated Circuits Sys.*, **17**(3): 239–254, 1998.
25. M. J. Geuzebroek, J. Th. van derLinden, and A. J. van deGoor, Test point insertion that facilitates ATPG in reducing test time and data volume, *Proceedings of the 2002 International Test Conference (ITC'2002)*, 2002, pp. 138–147.
26. H. Vranken, F. S. Sapei, and H. Wunderlich, Impact of test point insertion on silicon area and timing during layout, *Proceedings of the Design, Automatin and test in Europe Conference and Exhibition (DATE'04)*, 2004.
27. W. C. Carter, H. C. Montgomery, R. J. Preiss, and J. J. Reinheimer, Design of serviceability features for the IBM system/360, *IBM J. Res. & Devel.*, 115–126, 1964.
28. E. B. Eichelberger and T. W. Williams, A logic design structure for LSI testability, *Proceedings of the Fourteenth Design Automation Conference*, New Orleans, LA: 1977, pp. 462–468.
29. S. Funatsu, N. Wakatsuki, and T. Arima, Test generation systems in Japan, *Proceedings of the Twelfth Design Automation Conference*, 1975, pp. 114–122.
30. M. J. Y. Williams and J. B. Angel, Enhancing testability of large-scale integrated circuits via test points and additional logic, *IEEE Trans. Computers*, **C-22**(l): 46–60, 1973.
31. A. S. M. Hassan, V. K. Agarwal, B. Nadeau-Dostie, and J. Rajski, BIST of PCB interconnects using boundary-scan architecture, *IEEE Trans. Comp.*, **41**(10): 1278–1288, 1992.
32. N. A. Touba and B. Pouya, Using partial isolation rings to test core-based designs, *IEEE Design and Test of Computers*, 1997, pp. 52–57.
33. Y. Zorian, A structured testability approach for multi-chip modules based on BIST and boundary-scan, *IEEE Trans. Compon., Packag. Manufactu. Technol. Part B*, **17**(3): 283–290, 1994.

34. IEEE*IEEE Standard Test Access Port and Boundary-Scan Architecture*, Piscataway, NJ: IEEE, 1990.

35. D. Kagaris, S. Tragoudas, and A. Majumdar, On the use of counters for reproducing deterministic test sets, *IEEE Trans. Comp.*, **45**(12): 1405–1419, 1996.

36. P. H. Bardell, W. H. McAnney, and J. Savir, *Built-in Test for VLSI: Pseudorandom Techniques*, New York: John Wiley & Sons, l987.

37. M. Franklin, K. K. Saluja, and K. Kinoshita, A built-in self-test algorithm for row/column pattern sensitive faults in RAMs, *IEEE J. Solid-State Circuits*, **25**(2): 514–524, 1990.

38. S. Feng, T. Fujiwara, T. Kasami, and K. Iwasaki, On the maximum value of aliasing probabilities for single input signature registers, *IEEE Trans. Comp.*, **44**(11): 1265–1274, 1995.

39. M. Lempel and S. K. Gupta, Zero aliasing for modeled faults, *IEEE Trans. Computers*, **44**(11): 1283–1295, 1995.

40. B. Koenemann, B. J. Mucha, and G. Zwiehoff, Built-in test for complex digital integrated circuits, *IEEE J. Solid State Phys.*, **SC-15**(3): 315–318, 1980.

41. M. Tien-Chien Lee, *High-Level Tests Synthesis of Digital VLSI Circuits*, Boston: Artech House, Inc, 1997.

42. M. R. Prasad, P. Chong, and K. Keutzer, Why is ATPG easy? *Design Automation Conference*, 1999.

43. R. Wei and A. Sangiovanni-Vincentelli, PLATYPUS: A PLA test pattern generation tool, *22nd Design Automation Conference*, 1985, pp. 197–203.

Lee A. Belfore II
Old Dominion University,
Norfolk, Virginia

C

CARRY LOGIC

Addition is the fundamental operation for performing digital arithmetic; subtraction, multiplication, and division rely on it. How computers store numbers and perform arithmetic should be understood by the designers of digital computers. For a given weighted number system, a single digit could represent a maximum value of up to 1 less than the base or radix of the number system. A plurality of number systems exist (1). In the binary system, for instance, the maximum that each digit or bit could represent is 1. Numbers in real applications of computers are multibit and are stored as large collections of 16, 32, 64, or 128 bits. If the addition of multibit numbers in such a number system is considered, the addition of two legal bits could result in the production of a result that cannot fit within one bit. In such cases, a carry is said to have been generated. The generated carry needs to be added to the sum of the next two bits. This process, called carry propagation, continues from the least-significant bit (LSB) or digit, the one that has the least weight and is the rightmost, to the most-significant bit (MSB) or digit, the one with the most weight and is the leftmost. This operation is analogous to the usual manual computation with decimal numbers, where pairs of digits are added with carries being propagated toward the high-order (left) digits. Carry propagation serializes the otherwise parallel process of addition, thus slowing it down.

As a carry can be determined only after the addition of a particular set of bits is complete, it serializes the process of multibit addition. If it takes a finite amount of time, say (Δ_g), to calculate a carry, it will take 64 (Δ_g) to calculate the carries for a 64-bit adder. Several algorithms to reduce the carry propagation overhead have been devised to speed up arithmetic addition. These algorithms are implemented using digital logic gates (2) in computers and are termed *carry logic*. However, the gains in speed afforded by these algorithms come with an additional cost, which is measured in terms of the number of logic gates required to implement them.

In addition to the choice of number systems for representing numbers, they can further be represented as fixed or floating point (3). These representations use different algorithms to calculate a sum, although the carry propagation mechanism remains the same. Hence, throughout this article, carry propagation with respect to fixed-point binary addition will be discussed. As a multitude of 2-input logic gates could be used to implement any algorithm, all measurements are made in terms of the number of 2-input NAND gates throughout this study.

THE MECHANISM OF ADDITION

Currently, most digital computers use the binary number system to represent data. The legal digits, or bits as they are called in the binary number system, are 0 and 1. During addition, a sum S_i and a carryout C_i are produced by adding a set of bits at the ith position. The carryout C_i produced during the process serves as the carry-in C_{i-1} for the succeeding set of bits. Table 1 shows the underlying rules for adding two bits, A_i and B_i, with a carry-in C_i and producing a sum S_i and carry-out, C_i.

FULL ADDER

The logic equations that represent S_i and C_i of Table 1 are shown in Equations (1) and (2). A block of logic that implements these is called full adder, and it is shown in the inset of Fig. 1. The serial path for data through a full adder, hence its delay, is 2 gates, as shown in Fig. 1. A full adder can be implemented using eight gates (2) by sharing terms from Equations (1) and (2):

$$S_i = A_iB_iC_{i-1} + \overline{A}_i\overline{B}_iC_{i-1} + \overline{A}_iB_i\overline{C}_{i-1} + A_i\overline{B}_i\overline{C}_{i-1} \quad (1)$$

$$C_i = A_iB_i + C_{i-1}(A_i + B_i) \quad (2)$$

RIPPLE CARRY ADDER

The obvious implementation of an adder that adds two n-bit numbers A and B, where A is $A_nA_{n-1}A_{n-2}\ldots A_1A_0$ and B is $B_nB_{n-1}B_{n-2}\ldots B_1B_0$, is a ripple carry adder (RCA). By serially connecting n full adders and connecting the carryout C_1 from each full adder as the C_{i-1} of the succeeding full adder, it is possible to propagate the carry from the LSB to the MSB. Figure 1 shows the cascading of n full adder blocks. It is clear that there is no special carry propagation mechanism in the RCA except the serial connection between the adders. Thus, the carry logic has a minimal overhead for the RCA. The number of gates required is $8n$, as each full adder is constructed with eight gates, and there are n such adders. Table 2 shows the typical gate count and speed for RCAs with varying number of bits.

CARRY PROPAGATION MECHANISMS

In a scenario where all carries are available right at the beginning, addition is a parallel process. Each set of inputs A_i, B_i, and C_{i-1} could be added in parallel, and the sum for 2 n-bit numbers could be computed with the delay of a full adder.

The input combinations of Table 1 show that if A_i and B_i are both 0s, then C_i is always 0, irrespective of the value of C_{i-1}. Such a combination is called a carry kill term. For combinations where A_i and B_i are both 1s, C_i is always 1. Such a combination is called a carry generate term. In cases where A_i and B_i are not equal, C_i is equal to C_{i-1}. These are called the propagate terms. Carry propagation originates at a generate term, propagates through

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Table 1. Addition of Bits A_i and B_i with a Carry-in C_{i-1} to Produce Sum S_i and Carry-out C_i

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2. List of Gate Counts and Delay of Various Adders

	Gate Count/Delay		
Adder Type	16-Bit	32-Bit	64-Bit
RCA	144/36	288/68	576/132
CLA	200/10	401/14	808/14
CSA	284/14	597/14	1228/14
CKA	170/17	350/19	695/23

any successive propagate terms, and gets terminated at a carry kill or a new carry generate term. A *carry chain* is a succession of propagate terms that occur for any given input combination of A_i and B_i. For the addition of two n-bit numbers, multiple generates, kills, and propagates could exist. Thus, many carry chains exist. Addition between carry chains can proceed in parallel, as there is no carry propagation necessary over carry generate or kill terms.

Based on the concept of carry generates, propagates, and kills, logic could be designed to predict the carries for each bit of the adder. This mechanism is static in nature. It can be readily seen that different carry chains exist for different sets of inputs. This introduces a dynamic dimension to the process of addition. The dynamic nature of the inputs could also be used and a sum computed after the carry propagation through the longest carry chain is completed. This leads to a classification into static and dynamic carry logic.

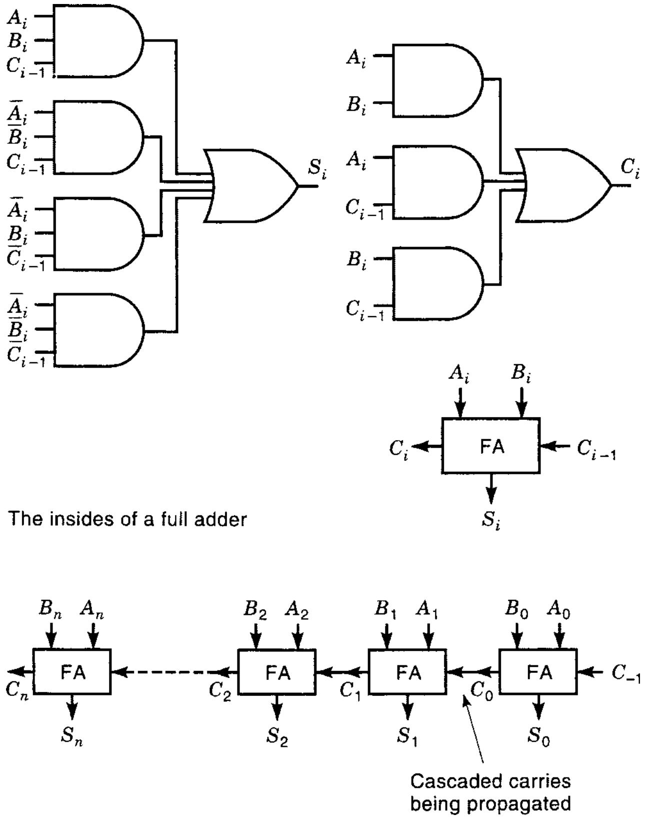


Figure 1. A Ripple Carry Adder ripples the carry from stage to stage using cascaded Full Adders.

An adder that employs static carry propagation always produces a sum after a fixed amount of time, whereas the time taken to compute the sum in a dynamic adder is dependent on the inputs. In general, it is easier to design a digital system with a static adder, as digital systems are predominantly synchronous in nature; i.e., they work in lock step based on a clock that initiates each operation and uses the results after completion of a clock cycle (4).

STATIC CARRY LOGIC

From Equation (1), if A_i and B_i are both true, then C_i is true. If A_i or B_i is true, then C_i depends on C_{i-1}. Thus, the term A_iB_i in Equation (1) is the generate term or g_i, and $A_i + B_i$ is the propagate term or p_i. Equation (1) can be rewritten as in Equation (3):

$$C_i = g_i + p_iC_{i-1} \tag{3}$$

where $g_i = A_iB_i$ and $p_i = A_i + B_i$. Substituting numbers for i in Equation (3) results in Equations (4) and (5):

$$C_1 = g_1 + p_1C_0 \tag{4}$$

$$C_2 = g_2 + p_2C_1 \tag{5}$$

Substituting the value of C_1 from Equation (4) in Equation (5) yields Equation (6):

$$C_2 = g_2 + p_2g_1 + p_2p_1C_0 \tag{6}$$

Generalizing Equation (6) to any carry bit i yields Equation (7):

$$C_i = g_i + p_ig_{i-1} + p_ip_{i-1}g_{i-2} + \cdots + p_ip_{i-1}\cdots p_1g_1 + p_ip_{i-1}p_{i-2}\cdots p_iC_0 \tag{7}$$

By implementing logic for the appropriate value of i in Equation (7), the carry for any set of input bits can be predicted.

Carry Look-Ahead Adder

An adder that uses Equation (7) to generate carries for the various bits, as soon as A and B are available, is called a carry look-ahead adder (CLA). From Equation (7), the carry calculation time for such an adder is two gate delays, and a further two gate delays are required to calculate the sum with bits A_i, B_i, and the generated carry. In general, for a large number of bits n, it is impractical to generate the carries for every bit, as the complexity of Equation (7) increases tremendously. It is commonly practice in such cases to split the addition process into groups of k-bit CLA blocks that are interconnected. A group CLA is shown in Fig. 2. The groups now provide two new output functions G^* and P^*, which are the group generate and propagate terms. Equations (8) and (9) provide examples of how these terms are generated for 4-bit blocks. Equation (10) shows the generation of C_4 using G^*_1 and P^*_1:

$$G^*_1 = g_4 + p_4g_3 + p_4p_3g_2 + p_4p_3p_2g_1 \tag{8}$$

$$P^*_1 = p_4p_3p_2p_1 \tag{9}$$

$$C_4 = G^*_1 + P^*_1C_0 \tag{10}$$

In typical implementations, a CLA computes the sum in $\log_2 n$ time and uses gates to the order of $n\log n$. Table 2 lists the gate count and delay of various CLAs. Thus, with some additional gate investment, considerable speed-up is possible using the CLA carry logic algorithm.

Based on the CLA algorithm, several methods have been devised to speed up carry propagation even further. Three such adders that employ circuit-level optimizations to achieve faster carry propagation are the Manchester Carry Adder (4), Ling Adder (5), and the Modified Ling Adder (6). However, these are specific implementations of the CLA and do not modify carry propagation algorithms.

Carry Select Adder

The discussion in the previous section shows that the hardware investment on CLA logic is severe. Another mechanism to extract parallelism in the addition process is to calculate two sums for each bit, one assuming a carry input of 0 and another assuming a carry input of 1, and choosing one sum based on the real carry generated. The idea is that the selection of one sum is faster than actually propagating carries through all bits of the adder. An adder that employs this mechanism is called a carry select adder (CSA) and is shown in Fig. 3. A CSA works on groups of k-bits, and each group works like an independent RCA. The real carry-in is always known as the LSB, and it is used as C_0. In Fig. 3, C_k is used to select one sum, like $S^1_{3k-2k+1}$ or $S^0_{3k-2k+1}$ from the next group, $gp2$. In general, the selection and the addition time per bit are approximately equal. Thus, for a group that is k bits wide, it approximately takes $2k$ units of time to compute the sums and a further two units of time to select the right sum, based on the actual carry. Thus, the total time for a valid carry to propagate from one group to another is $2(k + 1)$ time units. Thus, for an optimal implementation, the groups in the CSA should be unequal in size, with each succeeding group being 1 bit wider than the preceding group. The gate count and speed of various CSAs is listed in Table 2.

Carry Skip Logic

If an adder is split into groups, gp 0, gp 1, and so on of RCAs of equal width k, and if a carry-in of 0 is forced into each group, then the carry out from each group is its generate term. The propagate term is simple to compute and can be computed by using Equation (9). As the group generate terms and propagate terms are thus available, the real carry-in at each group could be predicted and used to calculate the sum. An adder employing this mechanism for carry propagation is called a carry skip adder (CKA) and is shown in Fig. 3. The logic gates outside of the groups in Fig. 3 implement Equation (11), which is a generalization of Equation (10) for the carry at any position i. Thus, the process of carry propagation takes place at the group level, and it is possible to skip cary propagation over groups of bits:

$$C_i = G^*_{i/k} + P^*_{i/k}C_{ki} \tag{11}$$

It takes $2k$ time units to calculate the carry from any group of size k. Carry propagation across groups takes an additional $n/k - 2$ time units, and it takes another $2k$ time units to calculate the final sum. Thus, the total time is $4k + n/k - 2$ time units. By making the inner blocks larger in size, it is possible to calculate the sum faster, as it is then possible to skip carry propagation over bigger groups. Table 2 lists the gate count and performance of various CKAs.

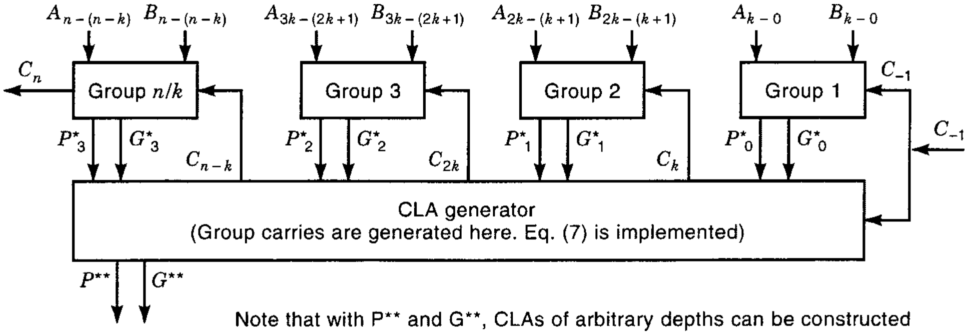


Figure 2. A group Carry Look Ahead Scheme with n/k groups each of size k.

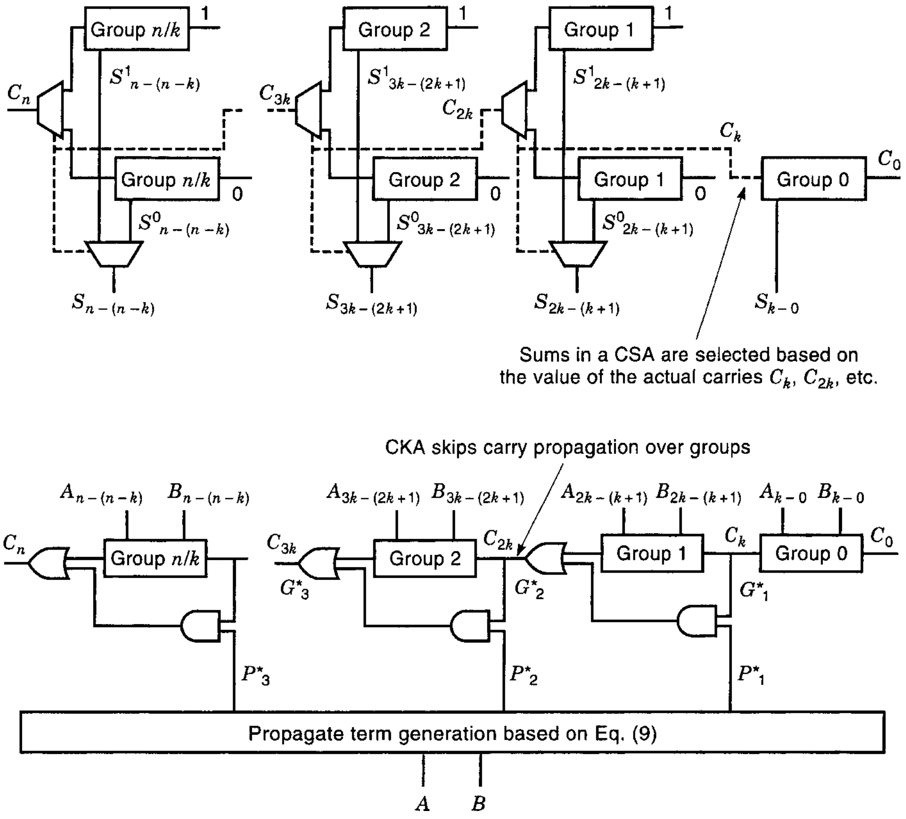


Figure 3. The CSA and CKA propagate carries over groups of k-bits.

Prefix Computation

Binary addition can be viewed as a parallel computation. By introducing an associative operator *, carry propagation and carry generation can be defined recursively. If $C_i = G_i$ in Equation (3), then Equation (12) with * as the concatenation operator holds. P_i is the propagate term, and G_i is the generate term at bit position i at the boundary of a group of size k:

$$(G_i, P_i) = (g_i, p_i) \text{ if } \\ i = 1 \text{ and } (g_i, p_i)^*(G_{i-1}, P_{i-1}) \text{ if } n \geq i > 1 \tag{12}$$

where $(g_t, p_t)^*(g_s, p_s) = (g_t + p_t g_s, p_t p_s)$ by modifying Equation (3). Note that * is NOT commutative. All C_i can be computed in parallel. As * is associative, the recursive Equation (12) can be broken in arbitrary ways. The logic to compute carries can be constructed recursively too. Figure 4 shows an example of carry computation using the prefix computation strategy described in Equation (12), with block size $k = 4$ and how a combination of two 4-bit carry-logic blocks can perform 8-bit carry computation.

The CLA, CKA, CSA, and Prefix computation have been discussed in detail by Swartzlander (7), Henessey (8), and Koren (9).

DYNAMIC CARRY LOGIC

Dynamic carry propagation mechanisms exploit the nature of the input bit patterns to speed up carry propagation and rely on the fact that the carry propagation on an average is of the order of $\log_2 n$. Due to the dynamic nature of this mechanism, valid results from addition are available at different times for different input patterns. Thus, adders that employ this technique have completion signals that flag valid results.

Carry Completion Sensing Adder

The carry-completing sensing adder (CCA) works on the principle of creating two carry vectors, C and D, the primary and secondary carry vectors, respectively. The 1s in C are the generate terms shifted once to the left and are determined by detecting 1s in a pair of A_i and B_i bits, which represent the ith position of the addend and augend, A and B, respectively. The 1s in D are generated by checking the carries triggered by the primary carry vector C, and these are the propagate terms. Figure 5 shows an example for such a carry computation process. The sum can be obtained by adding A, B, C, and D without propagating carries. A n-bit CCA has an approximate gate count of $17n - 1$ and a

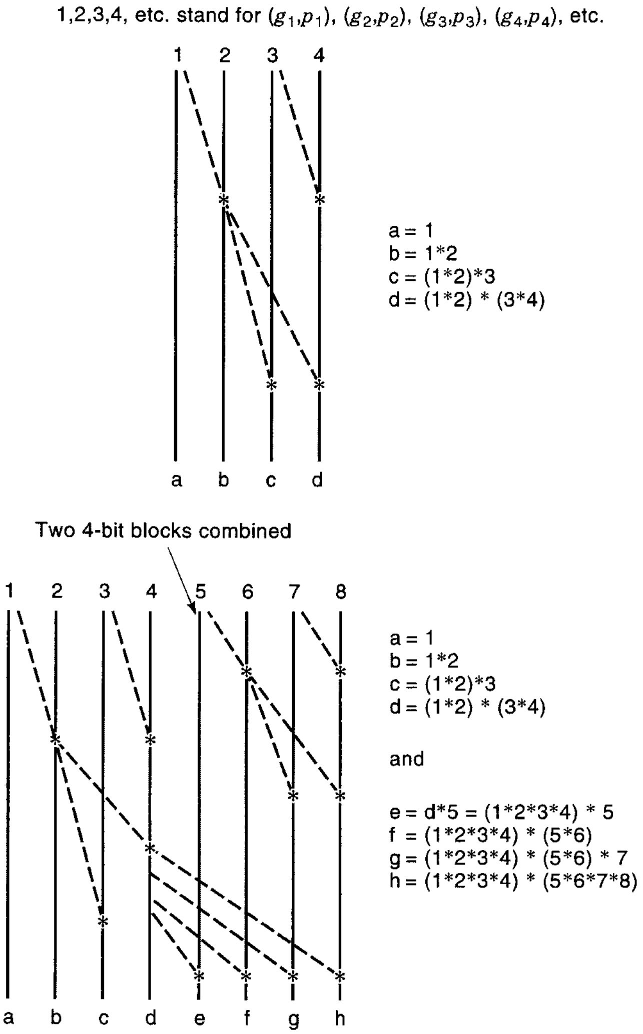


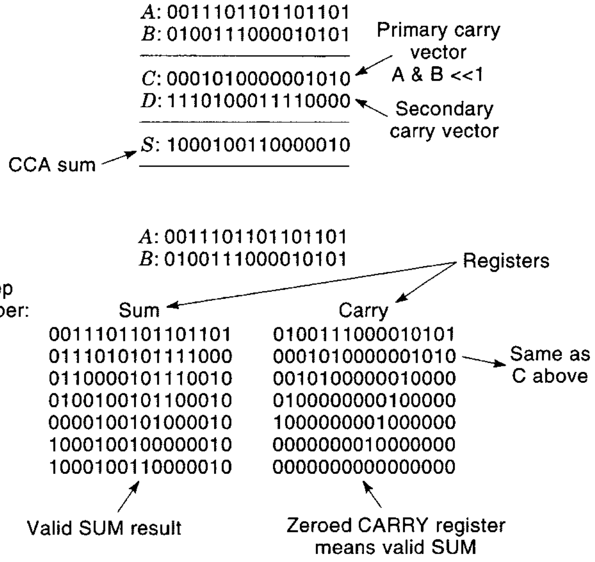
Figure 4. Performing 4-bit prefix computation and extending it to 8-bit numbers.

speed of $n + 4$. Hwang (10) discusses the carry-completion sensing adder in detail. Sklansky (11) provides an evaluation of several two-summand binary adders.

Carry Elimination Adder

Ignoring carry propagation, Equation (1) describes a half-adder, which can be implemented by a single XOR gate. In principle, it is possible to determine the sum of 2 n-bit numbers by performing Half Addition on the input operands at all bit positions in parallel and by iteratively adjusting the result to account for carry propagation. This mechanism is similar to the CCA. However, the difference is that the CCA uses primary and secondary carry vectors to account for carry propagation, whereas the carry elimination adder (CEA) iterates. The CEA algorithm for adding two numbers A and B is formalized by the following steps:

1. Load A and B into two n-bit storage elements called SUM and CARRY.


A: 0011101101101101
B: 0100111000010101
C: 0001010000001010
D: 1110100011110000
S: 1000100110000010

A: 0011101101101101
B: 0100111000010101

Step number:	Sum	Carry
1	0011101101101101	0100111000010101
2	0111010101111000	0001010000001010
3	0110000101110010	0010100000010000
4	0100100101100010	0100000000100000
5	0000100101000010	1000000001000000
6	1000100100000010	0000000010000000
7	1000100110000010	0000000000000000

Figure 5. The CCA and CEA use dynamic carry propagation.

2. Bit-wise XOR and AND SUM and CARRY simultaneously.
3. Route the XORed result back to SUM and left shift the ANDed result and route it back to the CARRY.
4. Repeat the operations until the CARRY register becomes zero. At this point, the result is available in SUM.

The implementation of the algorithm and detailed comparisons with other carry-propagation mechanisms have been discussed by Ramachandran (12).

Figure 5 shows an example of adding two numbers using the CEA algorithm. Note that the Primary carry vector C in the CCA is the same as the CARRY register value after the first iteration. The number of iterations that the CEA performs before converging to a sum is equal to the maximum length of the carry chain for the given inputs A and B. On average, the length of a carry chain for n-bit random patterns is $log_2 n$. The gate count of the CEA is about $8n + 22$ gates. It approaches the CLA in terms of speed and the CRA in terms of gate count.

MATHEMATICAL ESTIMATION OF THE CARRY-CHAIN LENGTH

For a given carry chain of length j, the probability of being in the *propagate* state is $k/k^2 = 1/k$. Define $P_n(j)$ as the probability of the addition of two uniformly random n-bit numbers having a maximal length carry chain of length $\geq j$:

$$P_n(j) = 0 \text{ if } n < j, \quad \text{and} \quad P_n(n) = (1/k)^n \qquad (13)$$

$P_n(j)$ can be computed using dynamic programming if all outcomes contributing to probability $P_n(j)$ are split into suitable disjoint classes of events, which include each contributing outcome exactly once. All outcomes contributing to $P_n(j)$ can be split into two disjoint classes of events:

Class 1: A maximal carry chain of length $\geq j$ does not start at the first position. The events of this class consist precisely of outcomes with initial prefixes having 0 up to $(j-1)$ propagate positions followed by one nonpropagate position and then followed with the probability that a maximal carry chain of length $\geq j$ exists in the remainder of the positions. A probability expression for this class of events is shown in Equation (14):

$$\sum_{t=0}^{j-1}\left(\frac{1}{k}\right)^{t}\cdot\frac{(k-1)}{k}\cdot P_{n-t-1}(t) \qquad (14)$$

In Equation (14), each term represents the condition that the first $(t+1)$ positions are not a part of a maximal carry chain. If a position $k < t$ in some term was instead listed as nonpropagating, it would duplicate the outcomes counted by the earlier case $t = k-1$. Thus, all outcomes with a maximal carry chain beginning after the initial carry chains of length less than$j$ are counted, and none is counted twice. None of the events contributing to $P_n(j)$ in class 1 is contained by any case of class 2 below.

Class 2: A maximal carry chain of length $\geq j$ does begin in the first position. What occurs after the end of each possible maximal carry chain beginning in the first position is of no concern. In particular, it is incorrect to rule out other maximal carry chains in the space following the initial maximal carry chain. Thus, initial carry chains of lengths j through n are considered. Carry chains of length less than n are followed immediately by a nonpropagate position. Equation (15) represents this condition:

$$\left(\frac{1}{k}\right)^{m}+\sum_{t=j}^{m-1}\left(\frac{1}{k}\right)^{t}\cdot\frac{(k-1)}{k} \qquad (15)$$

The term $P_m(m) = (1/k)^m$ handles the case of a carry chain of full length m, and the summand handles the individual cases of maximal length carry chains of length j, $j+1$, $j+2, \ldots, m-1$. Any outcome with a maximal carry chain with length $\geq j$ not belonging to class 2 belongs to 1. In summary, any outcome with a maximal carry chain of length $\geq j$, which contributes to $P_n(j)$, is included once and only once in the disjoint collections of classes 1 and 2.

Adding the probabilities for collections 1 and 2 leads to the dynamic programming solution to $P_n(j)$ provided below, where $P_n(j) = p_n(j) + p_n(j+1) + \cdots + p_n(n-1) + p_n(n)$,, where $P_n(i)$ is the probability of the occurrence of a maximal length carry chain of precisely length i. Thus, the expected value of the carry length [being the sum from $i = 1$ to n of $i^{*}P_n(i)$] becomes simply the sum of the $P_n(j)$ from$j = 1$ to n. Results of dynamic programming indicate that the average carry length in the *2*-ary number system for 8 bits is 2.511718750000; for 16 bits it is 3.425308227539; for 32 bits, 4.379535542335; for 64 bits, 5.356384595083; and for 128 bits, 8.335725789691.

APPROXIMATION ADDITION

To generate the correct final result, the calculation must consider all input bits to obtain the final carry out. However, carry chains are usually much shorter, a design that considers only the previous k inputs instead of all previous input bits for the current carry bit can approximate the result (13). Given that the delay cost of calculating the full carry chain length of N bits is proportional to log (N), if k equals to the square root of$N$, the new approximation adder will perform twice as fast as the fully correct adder. With random inputs, the probability of having a correct result considering only k previous inputs is:

$$P(N,k) = \left(1-\frac{1}{2^{k+2}}\right)^{N-k-1}$$

This is derived with the following steps. First consider why the prediction is incorrect. If we only consider k previous bits to generate the carry, the result will be wrong only if the carry propagation chain is greater than $k+1$. Moreover, the previous bit must be in the carry generate condition. This can only happen with a probability of $1/2^{k+2}$ if we consider a k-segment. Thus, the probability of being correct is one minus the probability of being wrong. Second, there are a total of $N-(k+1)$ segments in an N-bit addition. To produce the final correct result, the segment should not have an error condition. We multiply all probabilities to produce the final product. This equation could determine the risk taken by selecting the value of k. For example, assuming random input data, a 64-bit approximation adder with 8-bit look-ahead ($k = 8$) produces a correct result 95% of the time.

Figure 6 shows a sample approximation adder design with $k = 4$. The top and bottom rows are the usual carry, propagate, and generate circuits. The figure also shows the sum circuits used in other parallel adders. However, the design implements the carry chain with 29, 4-bit carry blocks and three boundary cells. These boundary cells are similar but smaller in size. A Manchester carry chain could implement 4-bit carry blocks. Thus, the critical path delay is asymptotically proportional to constant with this design, and the cost complexity approaches N. In comparison with Kogge Stone or Han Carlson adders, this design is faster and smaller.

It is worthwhile to note that an error indication circuit could be and probably should be implemented because we know exactly what causes a result to be incorrect. Whenever a carry propagation chain is longer than k bits, the result given by the approximation adder circuit will be incorrect. That is, for the ith carry bit, if the logic function - (ai-1 XOR bi-1) AND (ai-2 XOR bi-2) AND... AND (ai-k XOR bi-k) AND (ai-k-1 AND bi-k-1) is true, the prediction will be wrong. We could implement this logic function for each carry bit and perform the logical OR of all

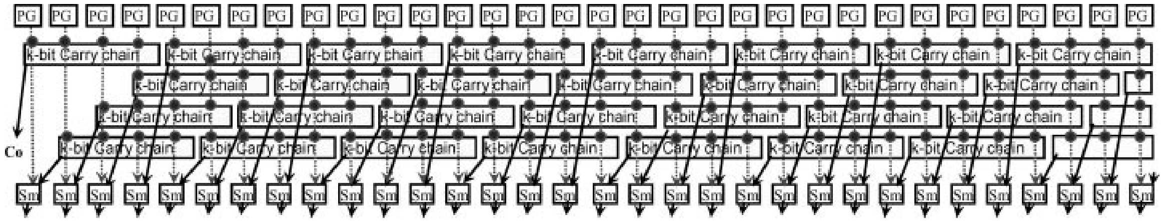


Figure 6. An example 32-bit approximation adder.

these n-4 outputs to signal us if the approximation is incorrect.

HIGH PERFORMANCE IMPLEMENTAION

The most recently reported adder implemented with the state-of-art CMOS technology is (14). The adder style used in that implementation is a variation of the prefix adder previously mentioned. Consideration was given not only to gate delay but also to fan-in, fan-out as well as to wiring delay in the design. Careful tuning was done to make sure the design is balanced, and the critical path is minimized.

BIBLIOGRAPHY

1. H. L. Garner, Number systems and arithmetic, in F. Alt and M. Rubinoff (eds.) *Advances in Computers*. New York: Academic Press, 1965, pp. 131–194.
2. E. J. McCluskey, *Logic Design Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
3. S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital System Designers*. New York: Holt, Reinhart and Winston, 1982.
4. N. H. E. Weste and K. Eshragian, *Principles of CMOS VLSI Design—A Systems Perspective*, 2nd ed. Reading, MA: Addison-Wesley, 1993, chaps. 5–8.
5. H. Ling, High speed binary parallel adder, *IEEE Trans. Comput.*, 799–802, 1966.
6. Milos D. Ercegovac and Tomas Lang, *Digital Arithmetic*. San Mateo, CA: Morgan Kaufmann, 2003.
7. E. E. Swartzlander Jr., *Computer Arithmetic*. Washington, D.C.: IEEE Computer Society Press, 1990, chaps. 5–8.
8. J. L. Henessey and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd ed.San Mateo, CA: Morgan Kauffman, 1996, pp. A-38–A-46.
9. I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. A. K. Peters Ltd., 2001.
10. K. Hwang, *Computer Arithmetic*. New York: Wiley, 1979, chap. 3.
11. J. Sklansky, An evaluation of several two-summand binary adders, *IRE Trans.* **EC-9** (2): 213–226, 1960.
12. R. Ramachandran, Efficient arithmetic using self-timing, Master's Thesis, Corvallis, OR: Oregon State University, 1994.
13. S.-L. Lu, Speeding up processing with approximation circuits, *IEEE Comput.* **37** (3): 67–73, 2004.
14. S. Mathew et al., A 4-GHz 130-nm address generation unit with 32-bit sparse-tree adder core, *IEEE J. Solid-State Circuits* **38** (5): 689–695, 2003.

Shih-Lien Lu
Intel Corporation
Santa Clara, California
Ravichandran Ramachandran
National Semiconductor
Corporation
Santa Clara, California

CD-ROMs AND COMPUTER SYSTEMS

HISTORY OF DEVELOPMENT: CD-ROM AND DVD

To distribute massive amounts of digital audio data, at reasonable cost and high quality, industry giants, such as Philips (Amsterdam, the Netherlands) and Sony (Tokyo, Japan), developed CD-ROM optical storage technology in the early 1980s, when digital "fever" was taking over the analog stereo music industry.

The obvious attractive features of the audio CD versus the vinyl LP are the relatively low cost of production, duplication, and distribution, as well as the robustness of the media and the significantly clearer and better (a feature that is still disputed by some "artistically bent" ears) sound quality that the digital technology offers over that of the analog.

It might be interesting to note that, as with many new, revolutionary technologies, even in the United States, where society accepts technological change at a faster rate than in most other countries, it took approximately 5 years for the audio CD to take over the vinyl phonograph record industry. (Based on this experience, one wonders how long it will take to replace the current combustion automobile engine for clean electric or other type of power. . . .)

For the computer industry, the compact disc digital audio (CD-DA) became an exciting media for storing any data (i.e., not just audio), including computer-controlled interactive multimedia, one of the most interesting technological innovations of the twentieth century. The approximately $1.00 cost of duplicating 650 Mb of data and then selling it as a recorded product for approximately $200.00 (in those days) created a new revolution that became the multimedia CD-ROM as we know it today.

Although there are not just read-only, but read/write CD-ROMs too (see CD-RW below), typically a CD-ROM is an optical read-only media, capable of storing approximately 650 Mb of uncompressed digital data (as an example a Sony CD-ROM stores 656.10 Mb in 335925 blocks, uncompressed), meaning any mixcture of text, digital video, voice, images, and others.

It is important to note that, with the advancement of real-time compression and decompression methods and technologies, CD recording software packages can put on a CD-ROM over 1.3 Gb of data, instead of the usual 650 Mb.

It is expected that, with increasing computer processor speeds and better integration [see what Apple's (Cupertino, CA) backside cache can do to the overall speed of the machine], real-time compression and decompression will be an excellent solution for many applications that need more than 650 Mb on one CD. Obviously this depends on the cost of the competing DVD technology too! This solution makes the CD-ROM and the emerging even higher capacity DVD technology essential to the digitalization and computerization of photography, the animation and the video industry, and the mass archivation and document storage and retrieval business.

To illustrate the breath and the depth of the opportunities of electronic image capture, manipulation, storage, and retrieval, consider Fig. 1(a) and 1(b), a solid model animation sequence of a short battle created by Richard G. Ranky, illustrating 200 high-resolution frame-by-frame rendered complex images, integrated into an Apple QuickTime digital, interactive movie and stored on CD-ROM, and Fig. 2(a)–(d), by Mick. F. Ranky, an interactively navigatable, panable, Apple QuickTime VR virtual reality movie of Budapest by night, allowing user-controlled zoom-in/out and other hot-spot controlled interactivity. (Note that some of these images and sequences are available in full color at www.cimwareukandusa.com and that more interactive demonstrations are available in the electronic version of this encyclopedia.) Note the difference in terms of the approach and methods used between the two figures. The first set was created entirely by computer modeling and imaging, and it illustrates a totally artificial world, whereas the second was first photographed of real, physical objects and then digitized and "pasted" and integrated into an interactive QTVR (see below) movie (1–11).

CD-ROM TECHNOLOGY, MEDIUM, AND THE STORAGE DENSITY OF DATA

The most important differences between the magnetic (hard disk) versus the optical (compact disc) technology include the storage density of data as well as the way data are coded and stored. This difference is because CD-ROMs (and DVDs) use coherent light waves, or laser beams, versus magnetic fields that are spread much wider than laser beams to encode information.

The other major advantage is that the laser beam does not need to be that close to the surface of the media as is the case with the magnetic hard disk read/write heads. Magnetic read/write heads can be as close as 16 μm, or 0.0016 mm, to the surface, increasing the opportunity of the jet-fighter-shaped, literally flying read/write head to crash into the magnetic surface, in most cases, meaning catastrophic data loss to the user.

The principle of the optical technology is that binary data can be encoded by creating a pattern of black-and-white splotches, just as ON/OFF electrical signals do or as the well-known bar code appears in the supermarket. Reading patterns of light and dark requires a photo detector, which changes its resistance depending on the brightness levels it senses through a reflected laser beam.

In terms of manufacturing, i.e., printing/ duplicating the compact disc, the major breakthrough came when engineers found that, by altering the texture of a surface mechanically, its reflectivity could be changed too, which means that the dark pit does not reflect light as well as a bright mirror. Thus, the CD-ROM should be a reflective mirror that should be dotted with dark pits to encode data,

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

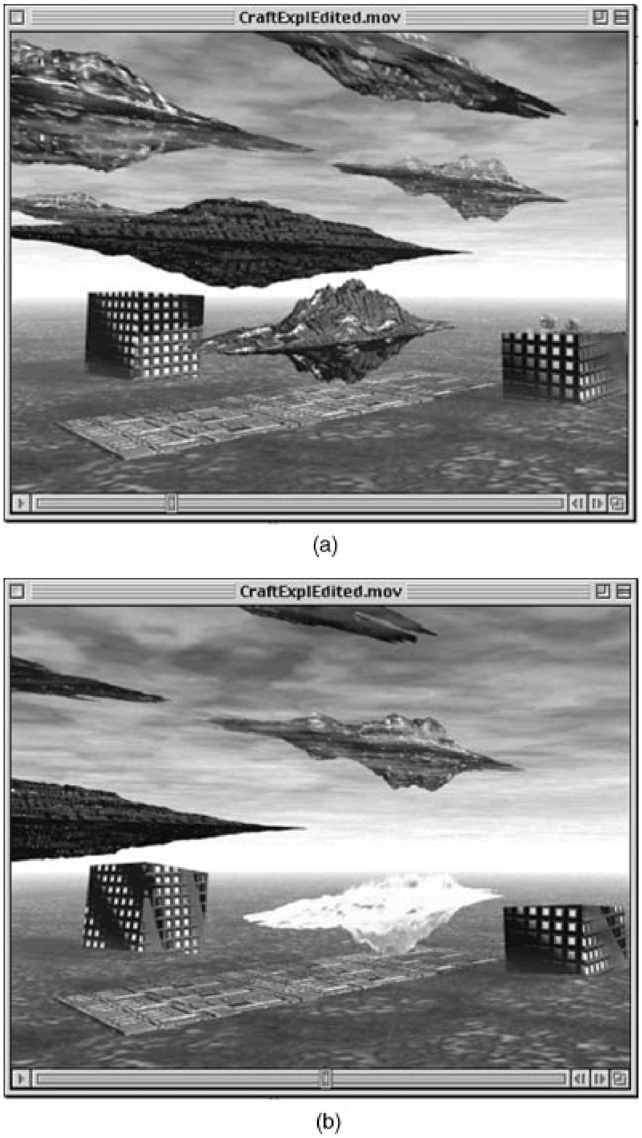


Figure 1. (a) and (b) Example of a 3-D animation scene. Solid model animation sequence of a short battle created by Richard G. Ranky, illustrating 200 high-resolution frame-by-frame rendered complex images, integrated into a QuickTime digital, interactive movie and stored on CD-ROM. (For full-color images, please look up the website: http://www.cimwareukandusa.com.)

by means of a laser beam traveling along a long spiral, just as with the vinyl audio LPs, that blasts pits accurately onto the disc.

The CD is a 80-mm-(i.e., the "minidisc") or a 120-mm-diameter (i.e., the "standard") disc, which is 1.2 mm thick and is spinning, enabling direct data access, just as with the vinyl audio LP, when the needle was dropped to any of the songs in any order. (Note that the more obvious 100 mm diameter would have been too small to provide the approximately 150-Mb-per-square-inch storage density of the optical technology of that of the 1980s, preferred by the classical music industry.) This meant solving the data access problem on a piece of coated plastic disk, in comparison with the magnetic hard disk, mechanically in a much simpler way.

To maximize the data storage capacity of the disc, the linear velocity recording of the compact disc is a constant 1.2 m per second. To achieve this rate both at the inside as well as the outside tracks of the disc, the spin varies between 400 rpm (revolutions per minute) and 200 rpm (at the outside edge). This way the same length of tracks appears to the read/write head in every second.

Furthermore, because of the access time the drive is capable of performing, it is important to note that the track pitch of the CD is 1.6 μm. This is the distance the head moves from the center toward the outside of the disc as it reads/writes data, and that the data bits are at least 0.83 μm long. (In other words, a CD-ROM and its drive are precision electro-mechanical and software instruments.)

It should be noted that, due to such small distances between the tracks, it is extremely important to properly cool CD writers as they cut master CD-ROMs in a professional studio, or inside or outside a PC or Mac on the desktop. Usually fan cooling is adequate; nevertheless, on a warm summer day, air-conditioning even in a desktop environment is advisable! Furthermore, such equipment should not be operated at all in case the inside cooling fan breaks down!

In terms of mass duplication, a master CD (often referred to as the "gold CD") is recorded first, then this master is duplicated by means of a stamping equipment—in principle, it is very similar to the vinyl audio LP production or the photocopying process. A crucial aspect of this process is that the data pits are sealed within layers of the disc and are never reached mechanically, only optically by the laser beam. Therefore, theoretically, quality mass-produced ("silver") CDs never wear out, only when abusing them harshly leaves scratches on the surface of the disc or when exposing them to extreme temperatures (12–17).

CD-ROM BLOCKS AND SECTORS

The recordable part of the compact disc consists of at least three blocks. These are as follows:

- Lead-in-block, holding the directory information, located on the innermost 4 mm of the disc's recording surface.
- Program block, holding the data or audio tracks and fills the next 33 mm of the disc.
- Lead-out-block, which marks the end of the CD at the external 1 mm.

The compact disc is divided into sectors. The actual size that is available is 2352 bytes for each sector. It should be noted that different CD formats use this 2352 bytes in different ways. As an example, an audio CD uses all 2352 bytes for audio data, whereas computer-oriented multimedia data formats need several bytes for error detection and correction.

Figure 2. (a)–(d) An interactively navigatable, 360-degree panable, QuickTime VR virtual reality movie of Budapest by night by Mick F. Ranky, allowing user-controlled zoom-in/out and other hot-spot controlled interactivity. (For full-color images, please look up the website: http://www.cimwareukandusa.com.)

Each sector is then divided further into logical blocks of 512, 1024, or 2048 bytes. These block sizes are part of the definition for each standardized compact disc format.

CD-ROM STANDARDS

As is the case with any successful technology, everyone wants to use CD-ROMs, but in their own way, depending on the applications; therefore, standardization of both the hardware as well as the software has brought at least some order to this "chaos."

Compact disc standards include the following:

- **Red Book:** This is the original compact disc application standardized by the International Standards Organization (ISO 10149) for digital audio data storage that defines digitization and sampling data rates, data transfer rates, and the pulse code modulation used.

  As prescribed by the Red Book, the ISRC-Code holds the serial number for each track in a standardized format.

  Q-Codes contain extra information about sectors such as the ISRC-Code, the Media Catalog Number, and the indices. The Media Catalog Number is a unique identification number (UPC-EAN bar code, Universal Product Code) for the compact disc.

  If required, the ISRC and Q-Codes can be set in specialized CD writing/mastering software, such as in Adaptec's Jam (see the discussion below on CD-ROM software packages).
- **Yellow Book:** Introduced in 1984, it was the first to enable multimedia and it describes the data format standards for CD-ROMs and includes CD-XA, which adds compressed audio data to other CD-ROM data. From a computing, interactive multimedia perspective, this format is the most important.

  The Yellow Book [ISO 10149: 1989(E)] divides the compact disc into two modes, as follows:

  Mode 1, for ordinary computer data.

  Mode 2, for compressed audio and digital video data.

Because Yellow Book CD-ROMs have mixed audio, video, and ordinary computer data, they are often referred to as mixed-mode CDs. (See the discussion on CD-ROM formats below.)

- **The Green Book** is the elaborate extension of the Yellow Book and is a standard for Philips" CD-i, Compact Disc Interactive. It brings together text, video, and sound on a single disc in an interleaved mode as well as extends the amount of digital, stereo audio data that can be put onto a single CD to up to 120 minutes (versus 74 minutes).
- **The Orange Book**, developed jointly by Philips and Sony, defines the hardware as well as the software aspects of the recordable CDs, often referred to as CD-R (Compact Disc Recordable—see below in more detail). Introduced in 1992, the Orange Book enabled multisession technology.

  A session is a collection of one or more tracks. Each recording procedure on a CD-R generates a session that contains all tracks recorded within the same time period, hence the terminology, "session."

  A compact disc recorded in multiple recording sessions is referred to as a multisession CD. In this case, each session has its own lead-in track and table of contents, used by the software. The number of sessions should be minimized, for efficient interactive multimedia playback as well as for saving 13 Mb overhead per session.

  Furthermore, the Orange Book defines the Program Area, which holds the actual data on the disc; a Program Memory Area, which records the track information for the entire disc; including all sessions it contains; the Lead-in Area, which holds the directory information; the Lead-out Area, which marks the end of the CD; and the Power Calibration Area, which is used to calibrate the power of the recording laser beam.
- **The Blue Book** standard was first published in 1995. It introduced stamped multisession compact discs in which the first track is a Red Book audio track. This resolved the "track one compatibility problem." (Formerly this standard was known as CD-Extra. Microsoft calls it CD-Plus.) The Blue Book standard enables compact disc authors to put interactive multimedia data into the unused capacity of music CDs.
- **The White Book** comprises the standards for video CDs. This format is based on CD-i (see the discussion above on the Green Book standard). These CD products are meant to be played on CD-i players.

PROPRIETARY CD-ROM STANDARDS

It should be mentioned that there are other proprietary compact disc standards too, most importantly the following:

- The KODAK Photo CD Sydney, Australia, readable on Macs, PCs, SIGs (computers made by Silicon Graphics Inc.,), and other machines, is a standard for storing high-quality photographic images developed by the Eastman Kodak Company (Rochester, NY).
- MMCD, a multimedia standard for handheld CD players by the Sony Corporation.

CD-ROM TRANSFER RATE

The transfer rate of a compact disc system is a direct function of the revolutions per minute (rpm) at which the disc spins in the player.

Because of the different sizes of blocks and the error correction methods used by different formats, the exact transfer rate at a given spin rate varies from one type of CD to the other.

As an example, in audio mode, the block size of 2362 bytes is transferred using a 1 × drive at 176 Kb per second, and in Mode 1, where the block size is 2048 bytes, the 1 × drive pushes through 153.6 Kb per second.

As with a CD-ROM drive, at 32 × speed, in Mode 1, where the block size is 2048 bytes, the 32 × drive pushes through $32 \times 153.6 = 4915.2$ Kb per second, a value close to a "reasonable" hard disk drive's transfer rate.

CD-ROM ACCESS TIME

In comparison with the magnetic hard disk drives, the CD-ROM's access time is significantly higher, due to the bulkiness of the optical read head, versus the elegant flyweight mechanism of the hard disk. The optical assembly, which moves on a track, carries more mass that translates to longer times for the head to settle into place.

Besides the mass of the optical head, the constant linear velocity recording system of the CD slows further the access of a desired data. With music, for which the CD was originally designed for, this is not a problem, because it is (usually) played back sequentially. As with computer data access, the CD must act as a random access storage device, when the speed (access time, plus read, or write time) becomes crucial.

The typical access time for a modern CD drive is approximately 100 to 200 ms, about ten times longer, than that of a modern magnetic hard disk's access time.

CD-ROM FORMATS

The Yellow Book standard enables multimedia, because it describes the data format standards for CD-ROM discs and includes CD-XA, which adds compressed audio data to other CD-ROM data. However, the Yellow Book does not define how to organize the data into files on the disc. Therefore, the High Sierra Format (HSF) and later the ISO9660 format was developed and standardized.

The only difference between the HFS and the ISO9660 formats is that some CD drives will read HFS CDs only (on old Macs), but the good news is that all recent drives on all platforms (i.e., MacOS, Windows/NT, Unix) should be able to read both.

Note that ISO9660 strictly maintains the 8/3 DOS naming conventions, whereas the HFS format, used on Macs from the very early days, allowed full-length Mac file names. (Long file names are beneficial in particular when a large number of multimedia objects/files has to be named and coded in a meaningful way.)

To fix this problem, for Windows 95, Microsoft (Redmond, WA) has introduced a set of extensions to ISO9660, called the Joliet CD-ROM Recording Specification. These extensions support 128-character-long filenames (not the maximum 255) with a broad character set. Unfortunately, DOS systems before Windows 95 still read according to the 8/3 file naming convention; thus, even some of the latest multimedia CDs are still forced to use the short 8/3 filenames (e.g., for a video clip: 12345678.mov, instead of a more meaningful: JaguarTestDrive_1.mov), to maintain compatibility.

CD-R (CD-RECORDABLE)

The fact that the compact disc is a sequentially recorded, but randomly playable, system (versus the magnetic disk, which is randomly recordable as well as playable) makes writing a CD-R a more complex operation than copying files over to a (magnetic) hard disk.

As CD recorders want to write data (i.e., "burn the CD-R") in a continuous stream, the data files to be recorded onto the CD must be put first into a defragmented magnetic disk folder, often referred to as "writing a virtual image" file. To assure continuous space on a hard drive, the best practice is to reformat the drive or its partition before moving any files into it. This will prevent any interruptions during CD recording (i.e., mastering) that will most likely result in an error in recording. In a normal case, the folder created on the magnetic disk will be copied over/recorded exactly "as is" onto the CD-ROM.

Furthermore, the number of sessions should be minimized too, for efficient interactive multimedia playback (in particular, in the case of several large video files) as well as for saving space (i.e., 13 Mb per session).

For the laser beam to code data onto the CD-R, the CD-R media needs an extra layer of dye. To guide the process even better, in particular in a desktop case, all CD-Rs have a formatting spiral permanently stamped into each disc.

Analyzing the cross section of a CD-R, the outside layer is a Silkscreened Label; then as we move further inside, there is a Protective Layer, and then the Reflective Gold Coating, with the photoreactive Green Layer embedded into a clear polycarbonate base.

As in the case with all CDs, the CD-R has a bottom protective layer, which gives its robustness. On the polycarbonate a thin reflective layer is plated to deflect the CD beam back so that it can be read by the compact disc drive. The dye layer, special to the CD-R, can be found between this reflective layer and the standard protective lacquer layer of the disc. It is photoreactive and therefore changes its reflectivity in response to the laser beam of the CD writer enabling data coding.

CD-RW, CD-ERASABLE, OR CD-E

Ricoh Co. Ltd. (Tokyo, Japan) pioneered the MP6200S CD-ReWritable drive in May 1996. (Note that CD-Erasable or CD-E was the original, confusing terminology.) At that time that was the only solution to a compact disk drive that could read as well as write data onto a CD! Users today enjoy a vastly expanded range of choices, both in terms of manufacturers as of well as of the variety of software bundles and interface options.

CD-RW employ phase-change laser technology to code and decode data. From a user point of view, in operation, CD-RW is similar to that of the magnetic hard disk. The drive can update the disc table of contents any time; thus, files and tracks can be added without additional session overheads. (Note that in the case of the CD-Rs, a session overhead is 13 Mb.)

Where CD-R drives in the past were limited to internal and external small computer system interface (SCSI), today's range of CD-RW/CD-R multifunction drives come with parallel and IDE connections, in addition to SCSI.

Other important aspects of CD-RWs include the following:

- In comparison with the IDE, or parallel-connected drives, SCSI drives can be considerably faster, especially when using a PCI bus-mastering card.
- Most modern PC motherboards support four IDE devices. If two hard drives and two CD-ROM drives are already installed, there is no room for additional IDE devices; thus, something has to be removed to install the CD-RW/CD-R drive.
- At the time of writing, the maximum read-speed of CD-RW drives is 6×; therefore, a faster 12 × to 32 × CD-ROM drive should be installed, in addition to the rewritable drive for fast multimedia playback.

Last, but not least, as with anything as complex as a CD-R, or CD-RW, it is strongly advisable to determine the importance of toll-free technical support, technical support hours of accessibility and availability, and the cost of software, driver, and flash BIOS upgrades.

CD-ROM CARE AND STABILITY

In general, inside the optical disc, there is a data layer on a substrate, which is read by a laser. In the case of CD-ROM, the data layer consists of a reflective layer of aluminum with "pits and plateaus" that selectively reflect and scatter the incident laser beam.

Optical discs are generally constructed from polymers and metallics. The polymers are subject to deformation and degradation. Metallic films are subject to corrosion, delamination, and cracking. Metallic alloys are subject to dealloying.

Optical discs consist of a data layer (pits, bumps, or regions of differing physical or magnetic properties) supported on a much thicker polycarbonate or glass substrate. A reflective layer is also required for CD-ROMs. The data layer/reflective layer is protected with an overcoat.

In optical media, there is a data "pit" that is responsible for reflecting/dispersing of an incident laser beam. Anything that changes the reflectivity or other optical properties for the data "bits" can result in a misread. According to the National Technology Alliance (USA), the optical clarity of the substrate is important in those systems where the laser must pass through this layer. Anything that interferes with the transmission of the beam, such as a scratch, or reduced optical clarity of the substrate, can result in a data error.

CD-ROM technology relies on the difference in reflectivity of "pits" stamped into a polycarbonate substrate and vapor coated with a reflective metallic layer, which is typically aluminum, hence the terminology for the mass-produced CDs, "silver" CDs.

According to the National Technology Alliance (USA), a common cause of CD-ROM failure is a change in the reflectivity of the aluminum coating as a result of oxidation, corrosion, or delamination. Deterioration of the protective overcoat (acrylic or nitrocellulose lacquer) can make the aluminum layer more susceptible to oxidation and corrosion. Some manufacturers use a silver reflecting layer that is subject to tarnishing by sulfur compounds in the environment and CD-ROM packaging.

CD-ROMs can also fail because of deterioration of the polycarbonate substrate. Polycarbonate is subject to crazing, which locally reduces the optical clarity of the substrate. Oils in fingerprints and organic vapors in the environment can contribute to crazing. Scratches in the substrate as a result of mishandling can also cause disk failures.

The relative effectiveness of CD-Recordable media is an issue often bandied about in industry and business circles, where the technology is used and increasingly relied on. Much controversy surrounds finding some useful way of evaluating the blank discs of various brands and types used in CD recorders today.

Several criteria go into evaluating disc usefulness: readability, compatibility with recorders and players, and expected life span. According to the National Technology Alliance (USA), results compiled in a series of tests performed by One-Off CD Shops International between early 1993 and mid-1995 on a variety of disc brands and types shed a great deal of light on the topic, even though the tests were done only to evaluate the readability of recorded discs, and not media longevity or suitability of specific brands or types for use on every system. But the methodological rigor of the narrow focus afforded yielded considerable data that bodes well for the effectiveness of current disc-evaluating mechanisms.

Not every question has been answered by any means, but one finding, according to the National Technology Alliance (USA), is clear: "worry about the quality of CD-R media seems largely unfounded" (18–21). Note that, in reality, the bigger worry is not the disk, but the entire system, in terms of computers, software, as well as CD/DVD readers and writers becoming obsolete within technology periods of approximately 3–5 years, and then after 10–15 years, one might not find a machine (i.e., a system) that can read an "old" CD-ROM or DVD-ROM, even if the data on the media is in good shape. . . .

CD-RECORDABLE VERSUS MASS-REPLICATED ("SILVER") COMPACT DISCS [AN ANALYSIS BY THE NATIONAL TECHNOLOGY ALLIANCE (USA)]

Mass-replicated (i.e., "silver") discs have their data encoded during injection molding, with pits and lands pressed directly into the substrate. The data side of the transparent disc is metalized, usually with aluminum sputtered onto the bumpy surface, which is spincoated with lacquer to protect the metal from corrosion, and then it is usually labeled in some fashion, generally with a silkscreened or offset printed design.

One source of confusion and concern about CD-R discs is their notable physical differences (i.e., “gold/green shine”) from normal (i.e., “silver” shine), pressed compact discs.

Each CD-R blank is designed to meet standards regarding function, but the way each achieves the function of storing digital information in a manner that can be read by standard CD players and drives is distinct. In terms of the top side and bottom side, replicated discs are similar to that of the CD-Rs; it is what comes between the polycarbonate substrate and the top’s lacquer coating that makes the difference.

CD-Rs are polycarbonate underneath, too, but the substrate is molded with a spiral guide groove, not with data pits and lands. This side is then coated with an organic dye, and gold or silver (instead of aluminum as in the case of mass-replicated discs) is layered on the top of the dye as the reflective surface, which in turn is lacquered and sometimes labeled just as replicated discs are.

The dye forms the data layer when the disc is recorded, having a binary information image encoded by a laser controlled from a microcomputer using a pre-mastering and recording program. Where the recording laser hits the dye, the equivalent of a molded “pit” is formed by the laser beam reacting with the photosensitive dye, causing it to become refractive rather than clear or translucent. When read by a CD player or CD-ROM drive, the affected area diffuses the reading laser’s beam, causing it to not reflect back onto the reader’s light-sensor. The alternations between the pickup laser’s reflected light and the refracted light make up the binary signal transmitted to the player’s firmware for unencoding, error detection, and correction, and further transmission to the computer’s processor or the audio player’s digital/analog converter.

According to the National Technology Alliance (USA), the feature that really distinguishes recordable media from replicated discs is the dye layer. The polymer dye formulas used by manufacturers are proprietary or licensed and are one of the distinguishing characteristics between brands.

Two types of dye formulas are in use at the time of writing, cyanine (and metal-stabilized cyanine) and phthalocyanine. One (cyanine) is green, and the other appears gold because the gold metalized reflective layer is seen through the clear dye.

TENETS OF READABILITY TESTING OF CD-ROMS AND CD-RS

At least in theory, however, these differences should have little or no impact on readability, becouse CD-R and CD-ROM media share the “Red Book” standard for CD-DA (Digital Audio). The Red Book specifies several testable measurements that collectively are supposed to determine whether a disc should be readable as an audio CD media. The Yellow Book, or multimedia CD-ROM standard, requires some additional tests.

As CD-Recordable discs, described in the Orange Book, are supposed to be functionally identical to mass-replicated “silver” CD-ROMs, it is logical to assume that the same test equipment and standards should be applied to them as to Yellow Book discs, so no new readability criteria were specified in the Orange Book. According to the National Technology Alliance (USA), several companies have built machines that are used for testing discs during and after the manufacturing process using these criteria, and only recently have new testing devices made specifically for CD-Recordable become available.

ACCELERATED TEST METHODOLOGY BY THE NATIONAL TECHNOLOGY ALLIANCE (USA)

Changes in a physical property involving chemical degradation can usually be modeled by an appropriate Arrhenius model. Error rates can be fit to an appropriate failure time distribution model. Once an appropriate model has been determined and fit to the experimental data, it can be used to estimate media properties or error rates at a future time at a given condition.

In performing accelerated tests, there is a tradeoff between the accuracy and the timeliness of the results. It is impractical to age data storage media at “use” conditions becouse it would take several years to evaluate the product, by which time it would be obsolete. To obtain results in a timely manner, “use” temperatures and humidities are typically exceeded to accelerate the rates of material decomposition.

Severe temperature/humidity aging may allow for a relatively rapid assessment of media stability, but results may not be representative of actual use conditions. Furthermore, samples treated in a laboratory environment may not be in a configuration representative of typical use conditions.

To perform accelerated testing, several media samples are placed in several different temperature/humidity/pollutant environments. The media are removed at periodic intervals, and a key property is measured.

This key property could be a physical characteristic, such as magnetic remanence, or it could be data error rates, if the materials were prerecorded. After a sufficient number of key property versus time data has been collected at each condition, the data can be fit to a predictive model (19,22–31).

ALTERNATIVE, INTERNET/INTRANET-BASED TECHNOLOGIES

With the rapid advancement of the Internet and local, typically much faster and more secure versions of it, often referred to as intranets, mass storage, document archiving, interactive multimedia distribution, and other services, mostly online, will become a reality and to some extent an alternative for data stored and distributed on CD-ROMs and DVDs.

The issue, nevertheless is always the same: online accessible data over a very fast network, under the “network’s control,” or at the desk on a CD-ROM, or DVD disc, under “the user’s/creator’s control.” No doubt there are reasons for both technologies to be viable for a long time, not forgetting the point, that even if it comes online over the fast network, at some point in the system the servers will most likely read

Table 1. Maximum Data Rates of Digital Telecommunications Standards

Standard	Connection type	Downstream rate	Upstream rate
V.34	Analog	33.6 Kbps	33.6 Kbps
SDS 56	Digital	56 Kbps	56 Kbps
ISDN	Digital	128 Kbps	128 Kbps
SDSL	Digital	1.544 Mbps	1.544 Mbps
T1	Digital	1.544 Mbps	1.544 Mbps
E1	Digital	2.048 Mbps	2.048 Mbps
ADSL	Digital	9 Mbps	640 Kbps
VDSL	Digital	52 Mbps	2 Mbps

the data from a CD-ROM or DVD jukebox, or even large-capacity magnetic hard disks.

To understand the importance of the online, networked solution and the areas in which they could, and most likely will, compete with the CD-ROM/ DVD technologies, refer to Table 1. It must be noted that these rates in Table 1 are theoretical maximum data rates, and in practice, unless a direct hired line is used, the actual transfer rates will most likely depend on the actual traffic.

Analyzing Table 1, it is obvious that 128-Kbps ISDN (Integrated Services Digital Network) lines, and upward, such as the T1 lines, representing the bandwidth of 24 voice channel telephone lines combined, provide viable online multimedia solutions. As with anything else, though, simultaneously competing, supporting, and conflicting issues such as speed, ease of use, security, privacy of data, and reliability/robustness will ensure that both the online as well as the, in this sense, offline, CD-ROM, CD-R, and DVD technologies will be used for a very long time.

CD-ROM/DVD-ROM APPLICATIONS

The CD-ROM and DVD-ROM technology is applied in several different areas, but most importantly as audio CDs (note that some rock stars have sold over 100 million CDs), for data and document archiving, for linear and nonlinear (i.e., interactive) video storage and playback, for image compression and storage, for interactive multimedia-based education, marketing, entertainment, and many other fields of interest, where mass storage of data is important.

Since besides the MPEG video standards, Apple's multiplatform as well as Internet-friendly QuickTime and QTVR digital interactive video and virtual reality software tools became the de facto interactive multimedia standards (delivered on CD-ROMs and DVDs as well as usually streamed at lower quality due to the transfer rate and bandwidth over the Internet and intranets), as examples of applications, we introduce these technologies as they are embedded into engineering educational, marketing, or game-oriented CD-ROM and DVD programs. In these examples, one should recognize the importance of accessing a large amount of data (e.g., 5–25-Mb digital, compressed video files), interactively, in a meaningful way, at the time and place the information is needed. (Furthermore, note that many of these interactive examples can be found electronically at the website: http://www.cimwareukandusa.com.)

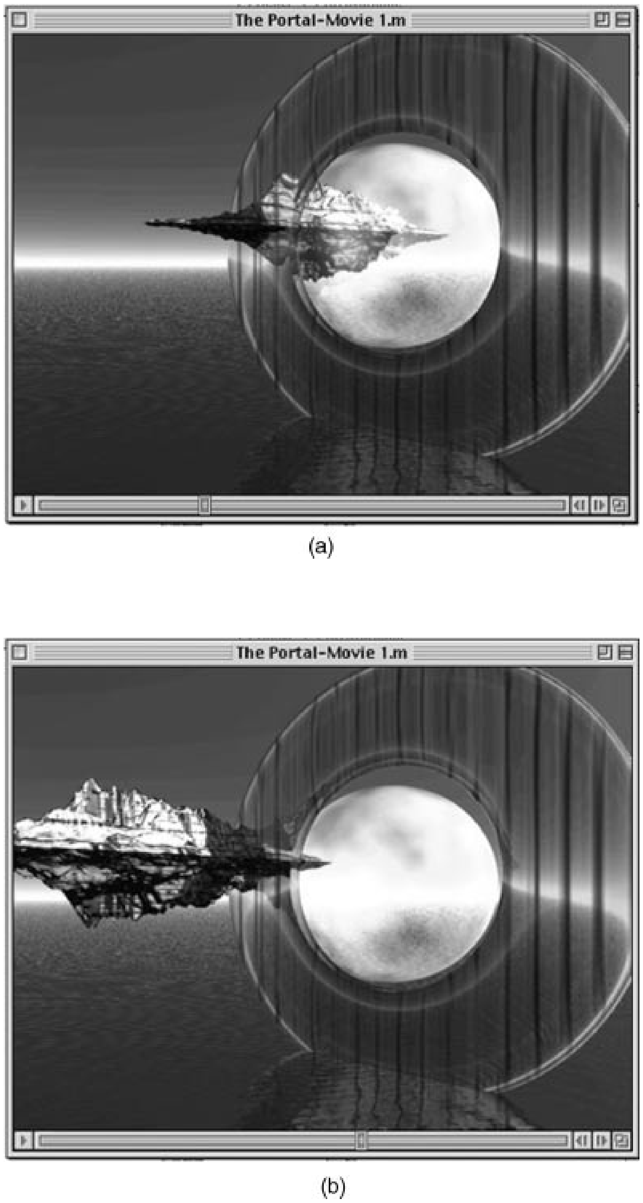


Figure 3. (a) and (b) These screenshots illustrate two frames of an animated space flight (by Gregory N. Ranky) as part of a video-game project on CD-ROM. The individual frames have been computer generated and then rendered and integrated into an interactive QT movie.

As the video-game industry is the prime source for computing and related CD-ROM R&D funding, we felt that we should demonstrate such new developments by showing Fig. 3(a) and (b). These screenshots illustrate two frames of a longer animated space flight (by Gregory N. Ranky) as part of a video-game project on CD-ROM. The individual frames have been computer generated, and then rendered and integrated into an interactive QT movie.

As Apple Computer Inc. defines, QuickTime(QT) is not an application, it is an enabling technology. QuickTime comprises of pieces of software that extend the ability of a Mac's or PC's operating system to handle dynamic media.

Applications then use this technology and turn it into other applications. As an example, many educational titles, games, and reference titles have incorporated QuickTime into their development, including Myst by Broderbund; Microsoft Encarta by Microsoft; DOOM II by Id Software; and Flexible Automation and Manufacturing, Concurrent Engineering, and Total Quality Management by CIMware and others.

QuickTime as a technology became the basis for many of the multimedia/computing industry's most respected digital media tools. QuickTime is much more than just video and sound. It is a true multimedia architecture that allows the integration of text, still graphics, video, animation, 3-D, VR, and sound into a cohesive platform. QuickTime, delivered either on CD-ROMs, DVDs, or in a somewhat less interactive mode over the Internet/intranet makes it easy to bring all of these media types together.

In February 1988, ISO has adopted the QuickTime File Format as a starting point for developing the key component of the MPEG-4 digital video specification, as the next-generation standard. This format is supported by Apple Computer Inc., IBM, Netscape Corp., Oracle Corp., Silicon Graphics Inc., and Sun Microsystems Inc.

"MPEG's decision to utilize the QuickTime file format for the MPEG-4 specification has huge benefits for users and the industry," said Ralph Rogers, Principal Analyst for Multimedia at Dataquest, San Jose, CA. "This strategy will leverage the broad adoption of QuickTime in the professional media space, speed the creation of MPEG-4 tools and content while providing a common target for industry adoption."

At a broader level, interactive multimedia, stored on CD-ROMs, DVDs, and the forthcoming fast Internet and intranets urges the development of anthropocentric systems in which humans and machines work in harmony, each playing the appropriate and affordable (i.e., the best possible) role for the purpose of creating intellectual as well as fiscal wealth. This means creating better educated engineers, managers, and workforce, at all levels, by building on existing skills, ingenuity, and expertise, using new science and technology-based methods and tools, such as interactive multimedia.

Today, and in the forthcoming decade of our information technology revolution, and eventually the Knowledge Age, engineering, science, and technology in combination can create an intellectually exciting environment that molds human creativity, enthusiasm, excitement, and the underlying curiosity and hunger to explore, create, and learn. It is obvious that economic development is not a unidimensional process that can be measured by a narrow view of conventional accounting.

Consequently there is a need to develop new creative and stimulative multimedia-based infrastructures, educational tools, as well as products and means of production that have the embedded intelligence to teach their users about "themselves" and that can meet challenges now faced by many companies and even countries as natural resources become more scarce, the environment becomes more polluted, and major demographic changes and movements of people are taking place.

The fundamental change that has to be recognized is that most existing hi-tech systems were designed with the human operator playing a passive role, and a machine being the "clever" component in the system. This is because accountant-driven management considers the workforce to be a major cost item instead of a major asset!

Anthropocentric technologies, such as flexible, interactive multimedia, make the best use of science and technology, driven by the user at his or her pace and time, enabling the learner to explore and implement concepts further than that of the accountants order-bound fiscal view.

Consequently, interactive multimedia is not war, but a new opportunity to put back humans into harmony with nature and "able" machines, by being better informed, educated, and happier contributors, rather than efficient long-term waste creators and destroyers of nature and the society (32–40).

WHAT IS INTERACTIVE MULTIMEDIA?

Interactive multimedia combines and integrates text, graphics, animation, video, and sound. It enables learners to extend and enhance their skills and knowledge working at a time, pace, and place to suit them as individuals and/or teams and should have a range of choices about the way they might be supported and assessed.

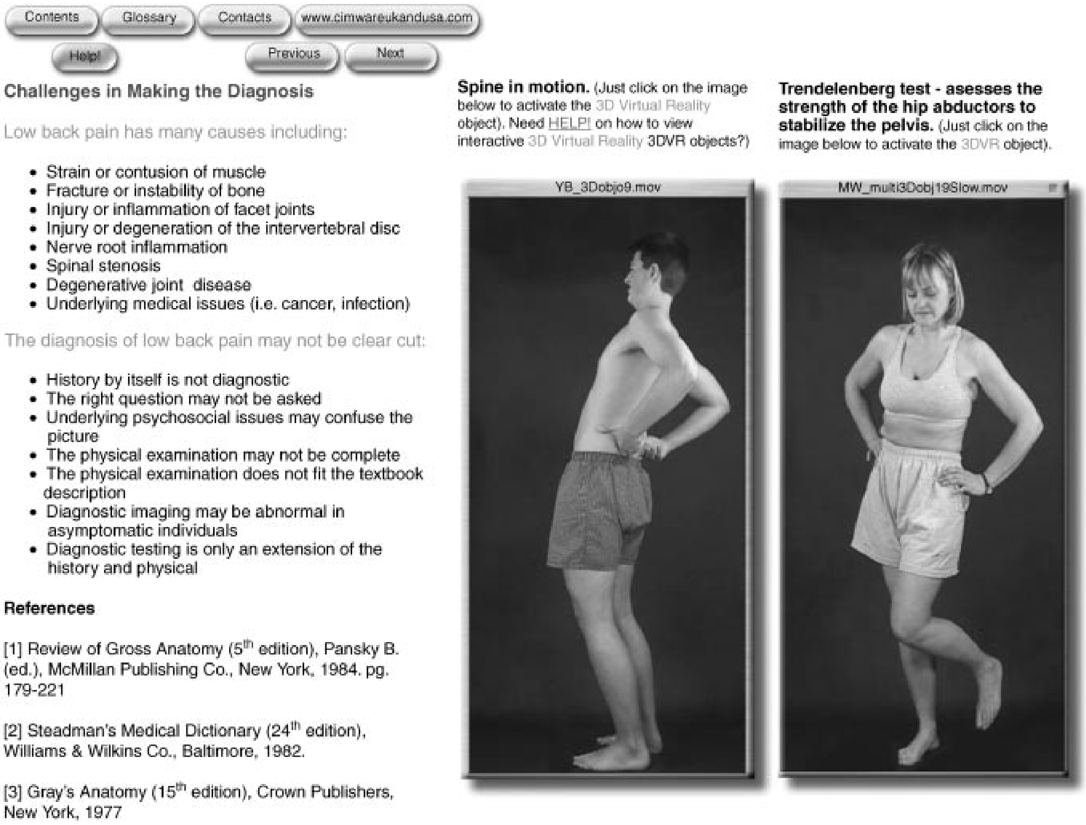
In other words:

- The user has a choice and the freedom to learn.
- He or she is supported by the multimedia-based learning materials and technology.
- The tutors are creating an effective, enjoyable learning environment and infrastructure.
- The learners are simultaneously learners as well as authors.

Figure 4 represents a screen of over 300 interactive screens of an industrial educational program on Servo Pneumatic Positioning, by Flaherty et al. (40) on CD-ROM. The 650 Mb of data includes several hundred color photos and over 45 minutes of interactive digital videos explaining the various aspects of servo pneumatic components, systems, positioning, control, programming, and applications.

Figure 5 is a screen of over 720 interactive screens of an educational multimedia program on Total Quality Control and Management and the ISO 9001 Quality Standard, by Ranky (41) on CD-ROM. The 650 Mb of data includes several hundred color photos and over 45 minutes of interactive digital videos explaining the various aspects of total quality and the international quality standard as applied to design, manufacturing, and assembly in a variety of different industries.

Note the many opportunities we have programmed into these screens to continuously motivate the learners to be responsive and be actively involved in the learning process. To maintain the continuous interactivity not just within the



Challenges in Making the Diagnosis

Low back pain has many causes including:

- Strain or contusion of muscle
- Fracture or instability of bone
- Injury or inflammation of facet joints
- Injury or degeneration of the intervertebral disc
- Nerve root inflammation
- Spinal stenosis
- Degenerative joint disease
- Underlying medical issues (i.e. cancer, infection)

The diagnosis of low back pain may not be clear cut:

- History by itself is not diagnostic
- The right question may not be asked
- Underlying psychosocial issues may confuse the picture
- The physical examination may not be complete
- The physical examination does not fit the textbook description
- Diagnostic imaging may be abnormal in asymptomatic individuals
- Diagnostic testing is only an extension of the history and physical

References

[1] Review of Gross Anatomy (5th edition), Pansky B. (ed.), McMillan Publishing Co., New York, 1984. pg. 179-221

[2] Steadman's Medical Dictionary (24th edition), Williams & Wilkins Co., Baltimore, 1982.

[3] Gray's Anatomy (15th edition), Crown Publishers, New York, 1977

Spine in motion. (Just click on the image below to activate the 3D Virtual Reality object). Need HELP! on how to view interactive 3D Virtual Reality 3DVR objects?)

Trendelenberg test - asesses the strength of the hip abductors to stabilize the pelvis. (Just click on the image below to activate the 3DVR object).

Figure 4. A sample screen of over 300 interactive screens of a 3-D eBook multimedia program for medical education. As can be seen, the screen includes text, images, video clips, and even 3-D objects. The novel feature of this approach is that the human characters are all based on real, living people and illustrated on the screen using photo-accurate, interactive 3-D methods developed by Paul G Ranky and Mick F. Ranky. (For full-color images and 3-D models, please look up the website: http://www.cimwareukandusa.com.)

CD-ROM, but also "outside" the CD, Internet and e-mail support is offered to learners. This enables them to interact with the author(s) and/or the subject area specialists of the particular CD-ROM via e-mail as well as visit the designated WWW domain site for further technical as well as educational support (42).

(Please note that some of these interactive multimedia examples are available in electronic format as executable demo code when this encyclopedia is published electronically. Also note that some of the images and demos illustrated here can be seen in full color at the website: http://www.cimwareukandusa.com.)

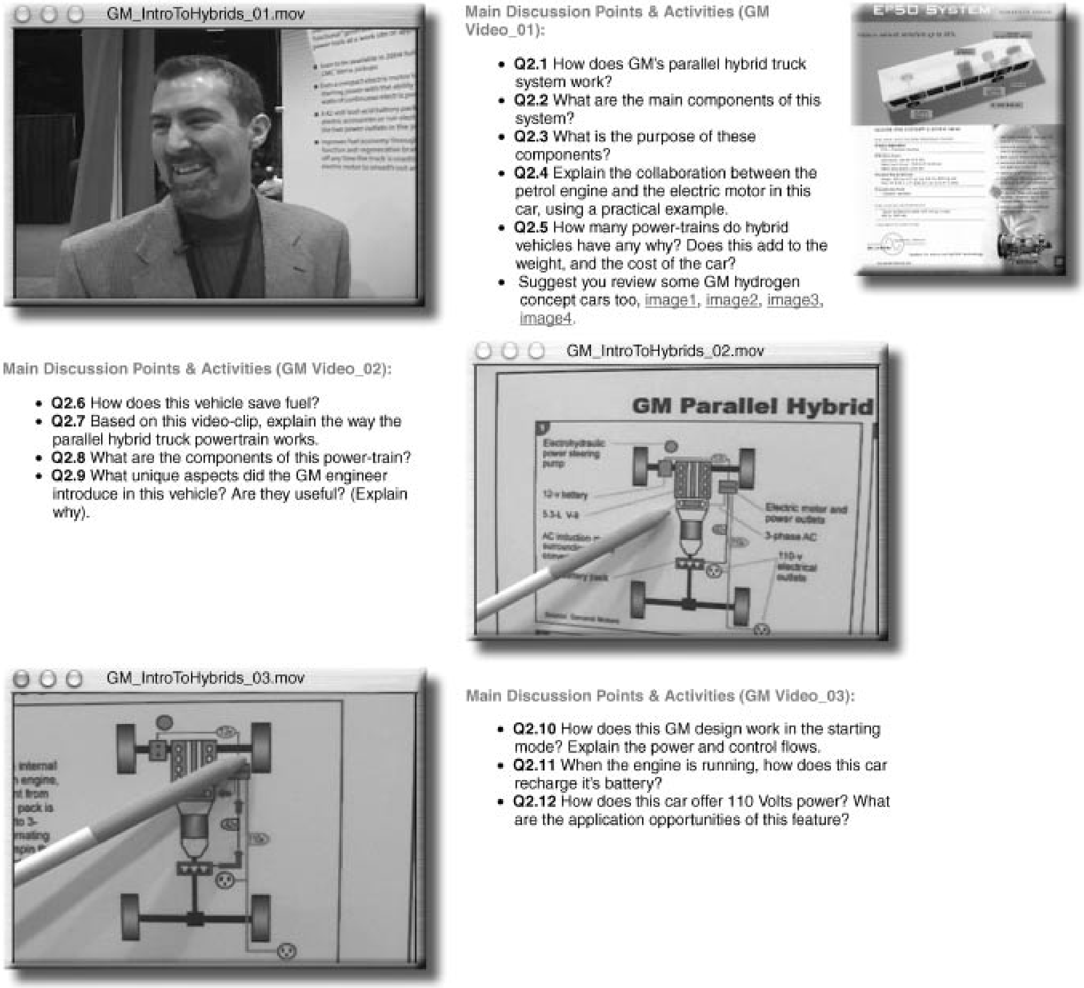
WHAT IS QUICKTIME VR?

As Apple describes, virtual reality describes a range of experiences that enables a person to interact with and explore a spatial environment through a computer. These environments are typically artistic renderings of simple or complex computer models. Until recently, most VR applications required specialized hardware or accessories, such as high-end graphics workstations, stereo displays, or 3-D goggles or gloves. QuickTime VR now does this in software, with real photographic images, versus rendered artificial models.

Apple's QuickTime VR is now an integral part of QuickTime; it allows Macintosh and Windows users to experience these kinds of spatial interactions using only a personal computer. Furthermore, through an innovative use of 360 panoramic photography, QuickTime VR enables these interactions using real-world representations as well as computer simulations.

To illustrate the power of this technology, when applied to interactive knowledge propagation on CD-ROMs, DVD-ROMs, and to some extent on the Internet, refer to Fig. 6(a)–(c), illustrating a few frames of an interactively controllable (Chevy) automobile image, including opening and closing its doors, under user control; Fig. 7(a)–(d), showing a few frames of an interactively navigatable interior of a Mercedes automobile; and Fig. 8(a)–(b), showing a traditional job-shop, again with all those great opportunities of interactive navigation, zoom/in and out, and hotspot controlled exploration of these hyperlinked images.

As can be recognized, the opportunities for interactivity, for learning by exploring under user (versus teacher control) is wasted, not just in education, but also in marketing and general culture, in terms of showing and illustrating



Main Discussion Points & Activities (GM Video_01):

- **Q2.1** How does GM's parallel hybrid truck system work?
- **Q2.2** What are the main components of this system?
- **Q2.3** What is the purpose of these components?
- **Q2.4** Explain the collaboration between the petrol engine and the electric motor in this car, using a practical example.
- **Q2.5** How many power-trains do hybrid vehicles have any why? Does this add to the weight, and the cost of the car?
- Suggest you review some GM hydrogen concept cars too, image1, image2, image3, image4.

Main Discussion Points & Activities (GM Video_02):

- **Q2.6** How does this vehicle save fuel?
- **Q2.7** Based on this video-clip, explain the way the parallel hybrid truck powertrain works.
- **Q2.8** What are the components of this power-train?
- **Q2.9** What unique aspects did the GM engineer introduce in this vehicle? Are they useful? (Explain why).

Main Discussion Points & Activities (GM Video_03):

- **Q2.10** How does this GM design work in the starting mode? Explain the power and control flows.
- **Q2.11** When the engine is running, how does this car recharge it's battery?
- **Q2.12** How does this car offer 110 Volts power? What are the application opportunities of this feature?

Figure 5. An illustration of a screen of over 720 interactive screens of an educational multimedia program on Alternative Energy Sources. The program is stored on CD-ROM (as well as the Web) and includes hundreds of images, video clips, 3-D objects, and 3-D panoramas; all interactive for the users to explore. (For full-color images and samples, please look up the website: http://www.cimwareukandusa.com.)

scenes, people, cultures, lifestyles, business practices, manufacturing, design and maintenance processes, and products even remotely, which have never been explored like this before.

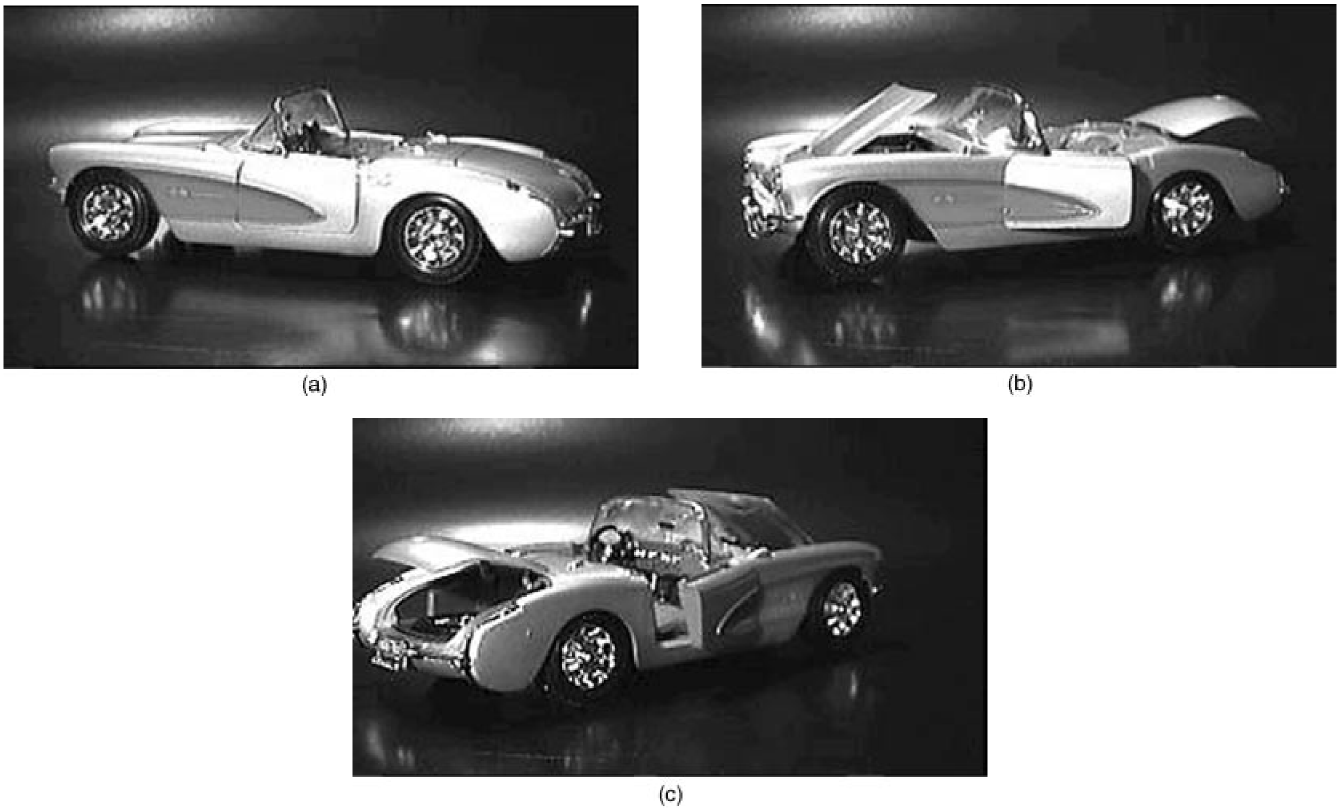
(Please note that some of these interactive multimedia examples are available in electronic format as executable demo code when this encyclopedia is published electronically. Also note that some of the images and demos illustrated here can be seen in full color at the website: http://www.cimwareukandusa.com.)

SMART DART: A SMART DIAGNOSTIC AND REPAIR TOOL IMPLEMENTED IN A VOICE I/O CONTROLLED, INTERACTIVE MULTIMEDIA, MOBILE -WEARABLE COMPUTER-BASED DEVICE FOR THE AUTOMOBILE (AND OTHER) INDUSTRIES

Smart DART is a novel, computer-based prototype mentoring system originally developed at the New Jersey Institute of Technology (NJIT) with industry-partners in 1998 with serious industrial applications in mind, implemented in a voice I/O controlled, interactive multimedia, mobile-wearable device for use by the automobile (and other) industries (see Fig. 9). The Co-Principal Investigators of this R&D project at NJIT were Professor Paul G. Ranky and Professor S. Tricamo and project partners in an R&D Consortium included General Motors, Raytheon, the U.S. National Guard, and Interactive Solutions, Inc.

The system consists of the following components:

- Integrated to the computer diagnostic port of the automobile, or offline, interacting with the technician, *can diagnose* a variety of problems and can communicate the results at the appropriate level, format, and mode, using various multimedia tools and solutions.
- *Can self-tune*, in terms of adjusting to the actual user needs and levels in an "intelligent way."
- Has a *highly interactive* and user-friendly multimedia interface.

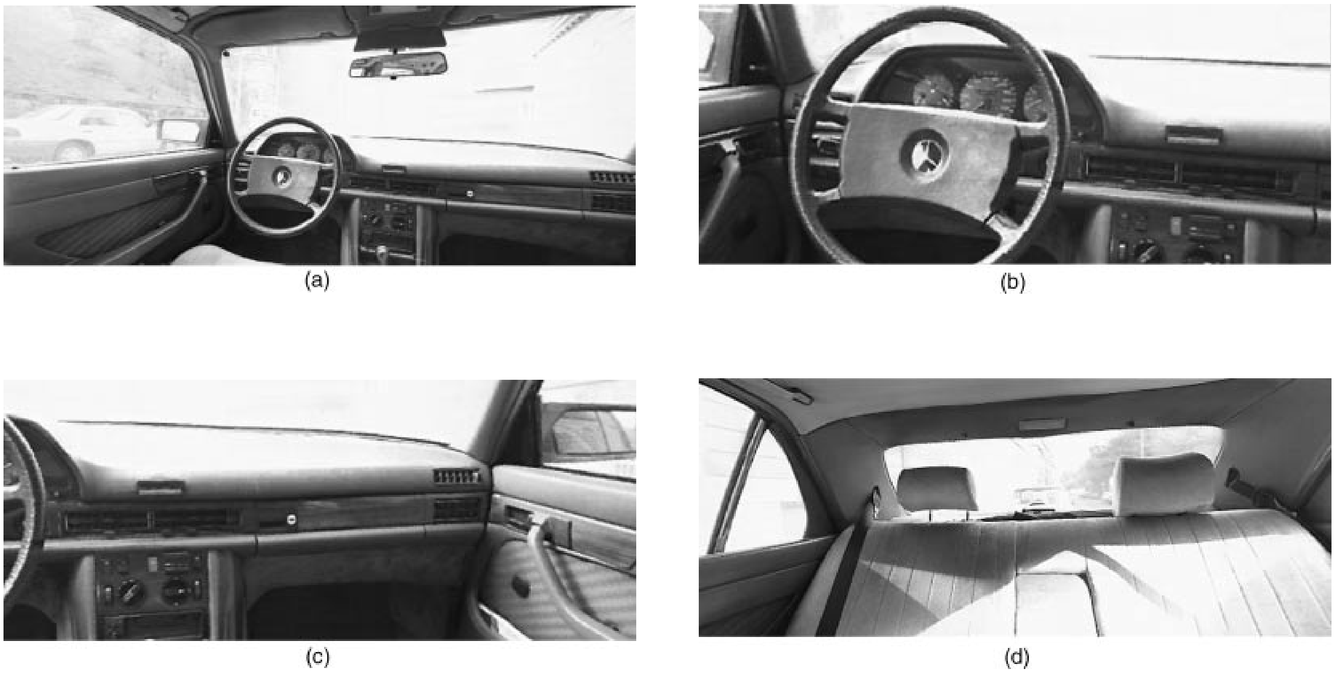


(a)

(b)

(c)

Figure 6. (a)–(c) The figure illustrates a few frames of an interactively controllable (GM Chevy) automobile image, including opening and closing its doors, under user control in QTVR on CD-ROM. (For full-color images, please look up the website: http://www.cimwareukandusa.com.)



(a)

(b)

(c)

(d)

Figure 7. (a)–(d) The figure shows a few frames of the interactively navigatable 3-D interior of a Mercedes automobile in QTVR on CD-ROM. (For full-color images, please look up the website: http://www.cimwareukandusa.com.)

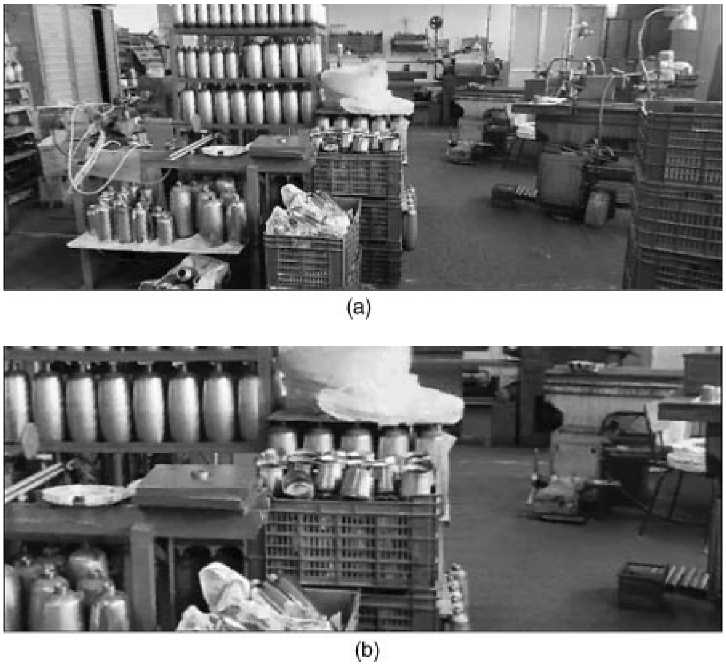


Figure 8. (a) and (b) The figure shows a traditional job-shop, again with all those great opportunities of interactive 3-D navigation, zoom/in and out, and hot-spot controlled exploration of these hyperlinked images in QTVR on CD-ROM. (For full-color images, please look up the website: http://www.cimwareukandusa.com.)

- *Can update itself* (based on either the learned knowledge and/or by means of networked or plugged-in technical fact data).
- Is a *highly parallel, distributed, and networked device*.
- Has command-based *voice recognition*.
- Has a *"hands-free" user interface*.
- Can work in *hazardous environments*.
- *Can automatically generate diagnostic and maintenance reports* and can communicate these reports via its networked communications system to any receiving site or compatible computer.
- To help to improve the next generation of products, the automated mentoring system can feed data as well as learned knowledge in a format and language that is appropriate and understandable to the design, manufacturing, quality control, and so on engineering community and their computer support and design systems (CAD/CAM).
- Smart DART *can diagnose itself* and report its own problems (and possible solutions) as they occur; therefore, it can help to improve the maintenance process as well as the design and the overall quality of the automobile (or other complex product it is trained for).

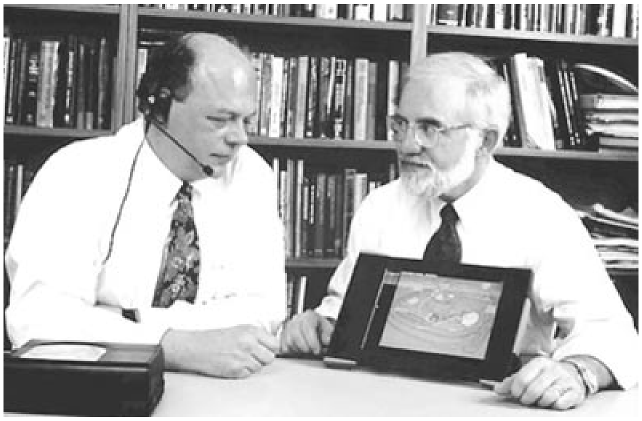


Figure 9. Smart DART is a novel, computer-based prototype mentoring system, originally developed in 1998, with serious industrial applications in mind, implemented in a voice I/O controlled, interactive multimedia, mobile-wearable device for use by the automobile (and other) industries. The R&D Consortium included NJIT, General Motors, Raytheon, the U.S. National Guard, and Interactive Solutions, Inc. (For full color-images, please look up the website: http://www.cimwareukandusa.com.)

About the System Architecture

To achieve the above listed and other functions, Smart DART is implemented as a small, ruggedized, networked mobile-wearable, or desktop networked computer-based device, which runs on a set of core processes, such as:

- The Process Manager.
- The Information Manager.
- The Interface Manager.
- The Team Coordinator.

Smart DART has a set of core modules linked to a fast knowledge-bus, through which various smart cards, or modules, it can execute various processes. These smart cards have embedded various domain expertise and have been integrated following the object-linking methodology.

Smart Dart has an open systems architecture, meaning that as the need arises new smart cards can be developed and plugged-in, in a way enhancing its "field expertise." Due to the well-integrated, object-linked design architecture, these new modules, or smart cards, will automatically integrate with the rest of the system, as well as follow the standard multimedia user-interface design, cutting the learning curve of using a new smart card to minimum.

The Typical Application Scope of Smart DART

To explain the application scope of our system, let us list some broad application areas, with that of the view of the maintenance technician, or engineer, whose job is to diagnose or fix a problem. In general, Smart DART will answer the following questions and resolve the following problems:

- *How does the particular system under test work?* This is explained using highly interactive, multimedia tools and interfaces to a newcomer, or to anybody that wishes to learn about the particular system. Note that a "system" in this sense can be an automobile, a tank, or some other machine, such as a VCR or a medical instrument.
- *What are the subsystems*, *how do they work*, and how do they *interact?*

Furthermore, Smart DART can

- *Diagnose the problem*.
- *Offer Go/No-go reporting*.
- *Provide end-to-end versus fault isolation*.
- *Rehearse the repair/fix scenarios and procedures* by means of highly interactive, and if required by the user, individualized interactive multimedia tools and techniques.
- Be used as an *"expert" tutor*, supporting learners at various levels, following different educational scenarios and techniques, best suited to the variety of different users (i.e., maintenance technicians, design, manufacturing and quality engineers, students, managers, and others).

DVD-ROM (DIGITAL VERSATILITY DISC)

The DVD-ROM, or DVD technology, was created by merging two competing proposals, one by the CD-ROM inventors Philips and Sony and the other one by Toshiba (Tokyo, Japan). The purpose of the DVD is to create up-front a universal, digital storage and playback system, not just for audio, but for video, multimedia, archiving, and general digital mass data storage.

DVDs are capable of storing significantly more data than CD-ROMs and come in different sizes and standards.

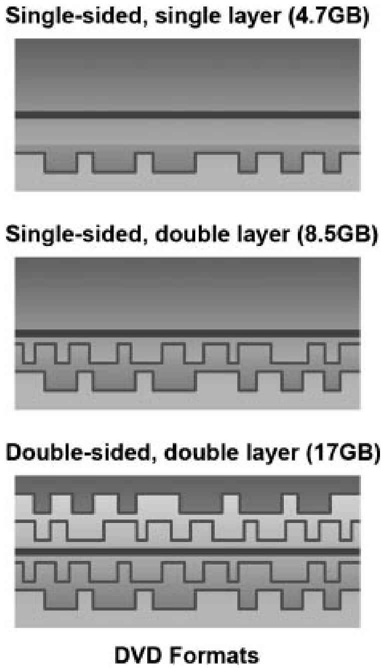

Single-sided, single layer (4.7GB)
Single-sided, double layer (8.5GB)
Double-sided, double layer (17GB)
DVD Formats

Figure 10. Examples of the structure and storage capacity of different DVD formats; single-sided single and double layer, and double-sided, double layer.

DVD is short for digital video (or versatility) disc and is the successor of the CD or compact disc. Because of its greater storage capacity (approximately seven times that of a CD), a DVD can hold 8 hours of music or 133 minutes of high-resolution video per side. This storage capacity varies depending on whether single-or double-layer discs are used and can range between 4.7 Gb and 8.5 Gb for single-sided discs or 17 Gb for double-sided dual-layer discs (see Fig. 10). The capacity does not directly double when a second layer is added because the pits on each layer are made longer to avoid interference. Otherwise they have the same dimensions as a CD, 12 cm in diameter and 1.2 mm in thickness.

The DVD medium resembles that of the CD-ROM technology. Even the size is the same, 120 mm diameter and 1.2 mm thick.

A DVD or CD is created by injection molding several layers of plastic into a circular shape, which creates a continuous stream of bumps arranged in a spiral pattern around the disc. Next, a layer of reflective material, aluminum for the inner layers, gold for the outermost, is spread to cover the indents. Finally, each layer is covered with lacquer and then compressed and cured under infrared light. Because of its composition, it is far more resistant to water absorption than its predecessor, the laser disc, and do not suffer from "laser rot." Each of these layers could act as fully functional disks on both sides. Individual layers are distinguished (i.e., addressed) by the system by focusing the laser beam. The result is a sandwich that has two layers per side, or in other words four different recording surfaces, hence, the significant data capacity increase.

Because the spiral data track begins in the center of the disc, a single-layer DVD can actually be smaller than 12 cm.

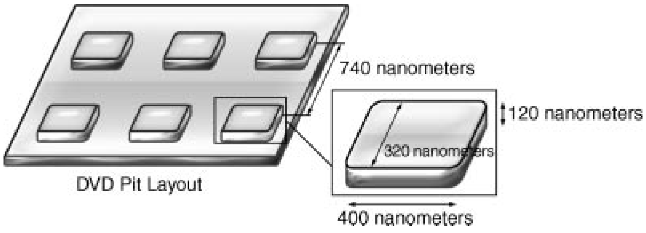


Figure 11. An illustration of the dimensions and spacing of the pits in successive tracks on a DVD.

This is the case for the UMD discs used by the Sony PSP Handheld Console. Each successive spiral is separated by 740 nm (10^{-9}m) of space (see Figs. 11 and 12), with each bump 120 nm in height, 400 nm long, and 320 nm wide; if unrolled, the entire line would be nearly 12 km (12000 m!) long.

These are usually called pits due to their appearance on the aluminum coating, although they are bumps when read by a laser. Because data are stored gradually outward, the speed of the drive is usually 50–70% of the maximum speed. By comparison, the spiral tracks of a CD are separated by 1.6 μm (10^{-6}m), with each bump 100 nm deep, 500 nm wide, and up to 850 nm long. This, combined with a 780 nm wavelength red laser, allows for much less data capacity than a DVD, approximately 700 Mb.

The data are actually stored directly under the label and are read from beneath by the laser. Therefore, if the top surface is scratched, the data can be damaged. If the underside is scratched or smudged, the data will remain, but the laser will have difficulty reading through the distortion.

VIDEO FORMATS

The usual form of data compression for (standard definition or SD) digital video is MPEG-2; the acronym comes from the Moving Picture Experts Group, which establishes video standards. The usual rate is 24 frames per second for video footage, but the display frame depends on the television format. The NTSC format displays footage in 60 fields, whereas PAL displays 50 fields but at a higher resolution. These differences in resolution also entail Pal or NTSC formatting for DVDs. Audio is usually in Dolby Digital formats, although NTSC discs may use PCM as well. Region codes also exist depending on the geographic location, from 1 to 8, with 0 used for universal playability.

Data tracks on a DVD

Figure 12. A simplified illustration of the spiraling layout of the DVD pits.

There are several types of recordable DVD discs; of these, DVD-R for Authoring, DVD-R for General use, DVD + R, and DVD-R are used to record data once like CD-R. The remaining three, DVD + RW, DVD-RW, and DVD-RAM, can all be rewritten multiple times.

As an example, DVD-5 with one side and one layer offers 4.7-Gb storage capacity and 133 minutes of playing time. DVD-9 can store 8.5 Gb on two layers, DVD-10 can store 9.4 Gb, and DVD-18 can store a massive 17.5 Gb and 480 minutes of equivalent playing time. These DVDs will be most likely used in interactive multimedia and digital video applications.

As optical technology has improved significantly since the 1980s when the CD-ROM was created, DVDs (standardized in December 1995) employ more closely spaced tracks and a better focused and high wavelength laser beam (635 to 650 nm, medium red).

The DVD constant linear velocity is 3.49 m per second, and the disc spins between 600 rpm and 1200 rpm, at the inner edge, much faster than the conventional CD-ROM.

DVD raw data transfer rates are high too, 11.08 Mb per second raw and approximately 9.8 Mb per second actual, approximately 7× or 8× in CD-ROM terms, enabling full motion, full-screen video playback.

Besides the computing industry's need to store massive amounts of data, the real commercial driver behind the DVD technology is the mereging, new high-definition (or HD) video industry, because DVDs could replace the old-fashioned, slow, linear, and relatively poor-quality VHS and S-VHS videotape technology.

For videos, DVD uses MPEG-2 encoding that allows a relatively high-quality display with 480 lines of 720 pixels (SD DVD quality), each to fit into a 4-Mb/s datastream. (Note that with MPEG, the actual data rate depends on the complexity of the image, analyzed frame-by-frame at compression stage. Also note that HD DVD video offers 1920 × 1080 or better resolution, meaning approximately 2 megapixels per frame, which is very good quality for most home and even professional users.)

DVD-Audio is excellent too, allowing a 44.1-KHz sampling rate and supporting 24-bit audio as well as several compressed multichannel formats, allowing switchable, multiple-language full-length videos to be stored and played back with additional audio and interactive features.

BLU-RAY

Blu-ray discs are named for the laser wavelength of 405 nm used to encode their data. Their sponsors include Apple Computer Corp., Dell, HP, Panasonic, Walt Disney, and Sun Microsystems.

As DVDs use a longer wavelength red laser, Blu-ray discs have a higher storage capacity. By using a shorter wavelength, as well as using higher quality lenses and a higher numerical aperture, the laser beam can be more tightly focused and therefore used to store more data.

A standard 12-cm Blu-ray disc has a single-layer storage capacity of 23.3, 25, or 27 Gb, equal to approximately 4 hours of high-definition video. They have a dual-layer capacity of 46.6 to 54 GB. Blu-ray discs were initially more vulnerable due to their data being closer to the surface, but with the introduction of a clear polymer coating, they can be cleaned with a tissue or supposedly resist damage by a screwdriver. This makes them more durable than current DVDs, with even fingerprints removable.

Blu-ray DVDs require a much lower rotation speed than HD DVDs to reach a 36 Mbps transfer rate. This results in a 12× BD for a Blu-ray disc but only 9× BD for an HD disc, as the current upper limit for optical drives is 10 rpm.

Unlike the eight DVD region codes, Blu-ray discs have three; region 1 covers the Americas, Japan, and East Asia excluding China; region 2 is for Europe and Africa; and region 3 is for China, Russia, India, and all other countries.

The Blu-ray Disc Association has also added digital watermarking to prevent unofficial distribution, or through HDTVs without an HDCP-enabled interface. Possible codecs used by Blu-ray discs include MPEG-2, H.264, and VC-1 for video and PCM and Dolby Digital for audio.

HD DVD

HD DVD discs, like Blu-ray, use a blue-violet 405-nm laser to encode data. Their promoters include Sanyo, Toshiba, Intel, Microsoft, Paramount Pictures, and Warner Bros.

HD DVDs have storage capacities of 15 Gb and 30 Gb for single-and dual-layer discs, respectively. This allows for approximately 8 hours of high-definition video storage for the 30-Gb model. Unlike Blu-ray, HD DVDs are backward compatible with DVDs, requiring no change in DVD players for the new format. HD DVD discs have a thicker protective coating (0.6 mm compared with 0.1 mm for Blu-ray), which allows greater resistance to damage, but also lower storage capacity, as the laser has more covering to penetrate.

Because HD DVDs use similar manufacturing processes to current DVDs, it is less expensive than having to change facilities to newer systems. A new system by Memory Tech can be adapted to create HD DVDs in 5 minutes. These converted lines will also be able to produce higher quality conventional DVDs, because HD-DVDs require a higher level of manufacturing precision.

CD-ROM/DVD DRIVE MANUFACTURERS AND CURRENT DRIVES

Although companies manufacturing CD-ROM and DVD hardware and software change, this list could be used as a reliable source for searching information and products.

DVS (Synchrome Technology)

Maestro CDR 4x12E,4X/12X, Windows 95, Windows NT, 200ms, SCSI.

Maestro CDR 4x12E,4X/12X, Macintosh, 200ms, SCSI.

Maestro CDR 4x121,4X/12X, Windows 95, Windows NT, 200ms, SCSI.

Japan Computer & Communication

JCD-64RW, 4X/2X/6X, Windows 95, Windows NT, 250ms, E-IDE.

MicroBoards Technology

Playwrite 4000RW, 4X2X/6X, Windows 95, Windows NT, Windows 3.1, UNIX, Macintosh, 250ms, SCSI-2.

Playwrite 400IRW, 4X/2X/6X, Windows 95, Windows NT, Windows 3.1, 250ms, E-IDE.

MicroNet Technology

MCDPLUS4X12, 4X/12X, Macintosh, 165ms, SCSI.

MCDPLUS4X12ADD, 4X/!2X, Windows 95, Windows NT, Windows 3.1, DOS, 165ms, SCSI.

MCDPLUS4X12PC, 4X/12X, Windows 95, Windows NT, Windows 3.1, DOS, 165ms, SCSI.

MCDPLUS4X121, 4X/12X, Windows 95, Windows NT, Windows 3.1, DOS, 165ms, SCSI.

MCDPLUS4X121PC, 4X/12X, Windows 95, Windows NT, Windows 3.1, DOS, 165ms, SCSI.

Microsynergy

CD-R4121, 4X/12X, Windows 95, Windows NT, Windows 3.1 Macintosh, 165ms, SCSI-2.

CD-R412E, 4X/12X, Windows 95, Windows NT, Windows 3.1 Macintosh, 165ms, SCSI-2.

CD-RW4261, 4X/2X/6X, Windows 95, Windows NT, Windows 3.1 Macintosh, 250ms, SCSI-2.

CD-RW426E, 4X/2X/6X, Windows 95, Windows NT, Windows 3.1 Macintosh, 250ms, SCSI-2.

Optima Technology Corp

CDWriter, 4X/2X/6X, Windows 95, Windows NT, 250ms, SCSI-2.

Panasonic

CW-7502-B, 4X/8X, Windows 95, Windows NT, Windows 3.1, Macintosh, 175ms, SCSI-2.

Pinnacle Micro

RCD-4x12e, 4X/12X, Windows 95, Windows NT, Macintosh, 165ms, SCSI-2.

RCD-4x12i, 4X/12X, Windows 95, Windows NT, Macintosh, 165ms, SCSI-2.

Plexor

PX-R412Ce, 4X/12X, Windows 95, Windows NT, Macintosh, 190ms, SCSI.

PX-R412Ci, 4X/12X, Windows 95, Windows NT, Macintosh, 190ms, SCSI.

Smart and Friendly

CD-R 4006 Delux Ext (SAF781), 4X/6X, Windows 95, Windows NT, Macintosh, 250ms, SCSI-2.

CD-R 4006 Delux Int (SAF780), 4X/6X, Windows 95, Windows NT, Macintosh, 250ms, SCSI-2.

CD Speed/Writer Delux Ext (SAF785), 4X/6X, Windows 95, Windows NT, Macintosh, 165ms, SCSI-2.

CD Speed/Writer Int (SAF783), 4X/6X, Windows 95, Windows NT, 165ms, SCSI-2.

CD-RW 426 Delux Ext (SAF782), 4X/2X/6X, Windows 95, Windows NT, Macintosh, 250ms, SCSI-2.

CD-RW 426 Delux Int (SAF779), 4X/2X/6X, Windows 95, Windows NT, 250ms, E-IDE.

TEAC

CD-R555,4X/12X, Windows 95, Windows NT, Windows 3.1, 165ms, SCSI.

CD-RE555,4X/12X, Windows 95, Windows NT, Windows 3.1, 165ms.SCSI.

Yamaha

CDR400t, 4X/6X, Windows 95, Windows NT, Windows 3.1, UNIX, Macintosh, 250ms, SCSI-2.

CDR400tx, 4X/6X, Windows 95, Windows NT, Windows 3.1, UNIX, Macintosh, 250ms, SCSI-2.

CDRW4260t, 4X/2X/6X, Windows 95, Windows NT, Windows 3.1, UNIX, Macintosh, 250ms, SCSI-2.

CDRW4260tx, 4X/2X/6X, Windows 95, Windows NT, Windows 3.1, UNIX, Macintosh, 250ms, SCSI-2.

CD-ROM/DVD SOFTWARE WRITERS/VENDORS AND CD-RECORDING SOFTWARE

Although companies manufacturing CD-ROM and DVD software as well as software version numbers change, this list could be used as a reliable source for searching information and products.

Company	Software
Apple MacOS	
Adaptec	Jam 2.1
Adaptec	Toast 3.54
Adaptec	DirectCD 1.01
Astarte	CD-Copy 2.01
CeQuadrat	Vulkan 1.43
CharisMac Engineering	Backup Mastery 1.00
CharisMac Engineering	Discribe 2.13
Dantz	Retrospect 4.0
Dataware Technologies	CD Record 2.12
Digidesign	Masterlist CD 1.4
Electroson	Gear 3.34
JVC	Personal Archiver Plus 4.10a
Kodac	Build-It 1.5
Microboards	VideoCD Maker1.2.5E
OMI/Microtest	Audiotracer 1.0
OMI/Microtest	Disc-to-disk 1.8
OMI/Microtest	Quick TOPiX2.20
Optima Technology	CD-R Access Pro 3.0
Pinnacle Micro	CD Burner 2.21
Pinnacle Micro	RCD 1.58
Ricoh	CD Print 2.3.1
IBM OS/2	
Citrus Technology	Unite CD-Maker 3.0
Electroson	GEAR 3.3
Young Minds	Makedisc/CD Studio 1.20
Sun SunOS	
Creative Digital Research	CD Publisher HyCD 4.6.5.
Dataware Technologies	CD Record 2.2
Eletroson	GEAR 3.50
JVC	Personal RomMaker Plus UNIX 3.6
Young Minds	Makedisc/CD Studio 1.2
Sun Solaris	
Creative Digital Research	CDR Publisher HyCD 4.6.5
Dataware Technologies	CD Record 2.2
Electroson	GEAR 3.50
JVC	Personal RomMaker Plus UNI 3.6
Kodak	Built-It 1.2
Luminex	Fire Series 1.9
Smart Storage	SmartCD for integrated recording & access 2.00
Young Minds	Makedisc/CD Studio 1.2
HP HP/UX	
Electroson	Gear 3.50
Smart Storage	SmartCD for integrated recording & access 2.00
Young Minds	Makedisc/CD Studio 1.20
JVC	Personal RomMaker Plus UNIX 1.0
Luminex	Fire Series 1.9
SGI IRIX	
Creative Digital Research	CDR Publisher HyCD 4.6.5
Electroson	GEAR 3.50
JVC	Personal RomMaker Plus UNIX 1.0
Luminex	Fire Series 1.9
Young Minds	Makedisc/CD Studio 1.20
DEC OSF	
Electroson	GEAR 3.50
Young Minds	Makedisc/CD Sturdio 1.20
IBM AIX	
Electroson	GEAR 3.50
Luminex	Fire Series 1.9
Smart Storage	SmartCD for integrated recording & access 2.00
Young Minds	Makedisc/CD Studio1.20
SCO SVR/ODT	
Young Minds	Makedisc/CD Studio 1.20

Novell NetWare	
Celerity systems	Virtual CD Writer 2.1
Smart Storage	SmartCD for recording 3.78
Smart Storage	Smart CD for integrated recording & access 3.78
Amiga	
Asimware Innovations	MasterISO 2.0

ACKNOWLEDGMENTS

We hereby would like to express our thanks to NSF (USA), NJIT (in particular co-PIs in major research grants, Professors Steven Tricamo, Don Sebastian, and Richard Hatch, and the students), Professor T. Pato at ISBE, Switzerland (co-PI in our European research grants), the students, and faculty who have helped us a lot with their comments in the United Kingdom, the United States, Switzerland, Sweden, Germany, Hungary, Austria, Hong Kong, China, and Japan. We also thank DARPA in the United States, The U.S. Department of Commerce, The National Council for Educational Technology (NCET, United Kingdom), The University of East London, the Enterprise Project team, the Ford Motor Company, General Motors, Hitachi Seiki (United Kingdom) Ltd, FESTO (United Kingdom and United States), Denford Machine Tools, Rolls-Royce Motor Cars, HP (United Kingdom) Ltd., Siemens Plessey, Marconi Instruments, and Apple Computers Inc. for their continuous support in our research, industrial, and educational multimedia (and other) projects. Furthermore, we would like to express our thanks to our families for their unconditional support and encouragement, including our sons, Gregory, Mick Jr. and Richard, for being our first test engineers and for their valuable contributions to our interactive multimedia projects.

FURTHER READING

A. Kleijhorst, E. T. Van der Velde, M. H. Baljon, M. J. G. M. Gerritsen and H. Oon, Secure and cost-effective exchange of cardiac images over the electronic highway in the Netherlands, computers in cardiology, *Proc. 1997 24th Annual Meeting on Computers in Cardiology*, Lund, Sweden, Sept. 7–10, 1997, pp. 191–194.

P. Laguna, R. G. Mark, A. Goldberg and G. B. Moody, Database for evaluation of algorithms for measurement of qt and other waveform intervals in the ecg, *Proc. 1997 24th Annual Meeting on Computers in Cardiology*, Lund, Sweden, Sept. 7–10, 1997, pp. 673–676.

B. J. Dutson, Outlook for interactivity via digital satellite, *IEE Conference Publication, Proc. 1997 International Broadcasting Convention*, Amsterdam, the Netherlands, Sept. 12–16, 1997, pp. 1–5.

Physical properties of polymers handbook, CD-ROM, *J. Am. Chemi. Soc.*, **119**(46): 1997.

J. Phillips, Roamable imaging gets professional: Putting immersive images to work, *Adv. Imag.*, **12**(10): 47–50, 1997.

H. Yamauchi, H. Miyamoto, T. Sakamoto, T. Watanabe, H. Tsuda and R. Yamamura, 24&Times;-speed circ decoder for a Cd-Dsp/CD-ROM decoder LSI Sanyo Electric Co, Ltd, *Digest of Technical Papers—IEEE International Conference on Consumer Electronics Proc. 1997 16th International Conference on Consumer Electronics*, Rosemont IL, 11–13, 1997, pp. 122–123.

K. Holtz and E. Holtz, Carom: A solid-state replacement for the CD-ROM, *Record Proc. 1997 WESCON Conference*, San Jose, CA, Nov. 4–6, 1997, pp. 478–483.

Anonymous, Trenchless technology research in the UK water industry, *Tunnelling Underground Space Technol.*, **11**(Suppl 2): 61–66, 1996.

J. Larish, IMAGEGATE: Making web image marketing work for the individual photographer, *Adv. Imag.*, **13**(1): 73–75, 1998.

B. C. Lamartine, R. A. Stutz and J. B. Alexander, Long, long-term storage, *IEEE Potentials*, **16**(5): 17–19, 1998.

A. D. Stuart and A. W. Mayers, Two examples of asynchronous learning programs for professional development, *Conference Proc. 1997 27th Annual Conference on Frontiers in Education. Part 1 (of 3)*, Nov. 5–8, Pittsburgh, PA, 1997, pp. 256–260.

P. Jacso, CD-ROM databases with full-page images, *Comput. Libraries*, **18**(2): 1998.

J. Hohle, Computer-assisted teaching and learning in photogrammetry, *ISPRS J. Photogrammetry Remote Sensing*, **52**(6): 266–276, 1997.

Y. Zhao, Q. Zhao, C. Zhu and W. Huang, Laser-induced temperature field distribution in multi-layers of cds and its effect on the stability of the organic record-layer, *Chinese J. Lasers*, **24**(6): 546–550, 1997.

K. Sakamoto and H. Urabe, Standard high precision pictures:SHIPP, *Proc. 1997 5th Color Imaging Conference: Color Science, Systems, and Applications*, Scottsdale, AZ, Nov. 17–20, 1997, pp. 240–244.

S. M. Zhu, F. H. Choo, K. S. Low, C. W. Chan, P. H. Kong and M. Suraj, Servo system control in digital video disc, *Proc. 1997 IEEE International Symposium on Consumer Electronics*, ISCE'97 Singapore, Dec. 2–4, 1997, pp. 114–117.

Robert T. Parkhurst, Pollution prevention in the laboratory, *Proc. Air & Waste Management Association's Annual Meeting & Exhibition Proceedings*, Toronto, Canada, June 8–13, 1997.

V. W. Sparrow and V. S. Williams, CD-ROM development for a certificate program in acoustics, *Engineering Proc. 1997 National Conference on Noise Control Engineering*, June 15–17, 1997, pp. 369–374.

W. H. Abbott, Corrosion of electrical contacts: Review of flowing mixed gas test developments, *Br. Corros. J.*, **24**(2): 153, 1989.

M. Parker, et al., Magnetic and magneto-photoellipsometric evaluation of corrosion in metal-particle media, *IEEE Trans. Magnetics*, **28**(5): 2368, 1992.

P. C. Searson and K. Sieradzki, Corrosion chemistry of magneto-optic data storage media, *Proc. SPIE*, **1663**: 397, 1992.

Y. Gorodetsky, Y. Haibin and R. Heming, Effective use of multimedia for presentations, *Proc. 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, Oct. 12–15, 1997, pp. 2375–2379.

J. Lamont, Latest federal information on CD-ROMs, *Comput. Libraries*, **17**: 1997.

M. F. Iskander, A. Rodriguez-Balcells, O. de losSantos, R. M. Jameson and A. Nielsen, Interactive multimedia CD-ROM for engineering electromagnetics, *Proc. 1997 IEEE Antennas and Propagation Society International Symposium*, Montreal, Quebec, Canada, July 13–18, 1997, pp. 2486–2489.

M. Elphick, Rapid progress seen in chips for optical drives, *Comput. Design*, **36**(9): 46, 48–50, 1997.

H. Iwamoto, H. Kawabe, and N. Mutoh, Telephone directory retrieval technology for CD-ROM, *Telecommun. Res. Lab Source*, **46**(7): 639–646, 1997.

J. Deponte, H. Mueller, G. Pietrek, S. Schlosser and B. Stoltefuss, Design and implementation of a system for multimedial distributed teaching and scientific conferences, *Proc. 1997 3rd Annual Conference on Virtual Systems and Multimedia*, Geneva, Switzerland, Sept. 10–12, 1997, pp. 156–165.

B. K. Das and A. C. Rastogi, Thin films for secondary data storage IETE, *J. Res.*, **43**(2–3): 221–232, 1997.

D. E. Speliotis et al., Corrosion study of metal particle, metal film, and ba-ferrite tape, *IEEE Trans. Magnetics*, **27**(6): 1991.

J. VanBogart et al., Understanding the Battelle Lab accelerated tests, *NML Bits*, **2**(4): 2, 1992.

P. G. Ranky, An Introduction to Concurrent Engineering, an Interactive Multimedia CD-ROM with off-line and on-line Internet support, over 700 interactive screens following an Interactive Multimedia Talking Book format, Design & Programming by P. G. Ranky and M. F. Ranky, CIMware 1996, 97. Available: http://www.cimwareukandusa.com.

P. G. Ranky, An Introduction to Computer Networks, an Interactive Multimedia CD-ROM with off-line and on-line Internet support, over 700 interactive screens following an Interactive Multimedia Talking Book format, Design & Programming by P. G. Ranky and M. F. Ranky, CIMware 1998. Available: http://www.cimwareukandusa.com.

Nice, Karim, Available: http://electronics.howstuffworks.com/dvd1. htm, 2005.

Available: http://en.wikipedia.org/wiki/Blu-Ray, 2006.

Available: http://en.wikipedia.org/wiki/Dvd, Feb. 2006.

Available: http://en.wikipedia.org/wiki/HD_DVD, Feb. 2006.

R. Silva, Available: http://hometheater.about.com/od/dvdrecorderfaqs/f/ dvdrecgfaq5. htm, 2006.

Herbert, Available: http://www.cdfreaks.com/article/186/1, Mar. 2005.

J. Taylor, Available: http://www.dvddemystified.com/dvdfaq.html, Feb. 10, 2005.

B. Greenway, Available: http://www.hometheaterblog.com/hometheater/blu-ray_hd-dvd/, Feb. 14, 2006.

L. Magid, Available: http://www.pcanswer.com/articles/synd_dvds. htm, Oct. 2 2003.

BIBLIOGRAPHY

1. C. F. Quist, L. Lindegren and S. Soderhjelm, Synthesis imaging, *Eur. Space Agency*, **SP-402**: 257–262, 1997.
2. H. Schrijver, Hipparcos/Tycho ASCII CD-ROM and access software, *Eur. Space Agency*, **SP-402**: 69–72, 1997.
3. J. Zedler, and M. Ramadan, I-Media: An integrated media server and media database as a basic component of a cross media publishing system, *Comput. Graph.*, **21**(6): 693–702, 1997.
4. V. Madisetti, A. Gadient, J. Stinson, J. Aylor, R. Klenke, H. Carter, T. Egolf, M. Salinas and T. Taylor, Darpa's digital system design curriculum and peer-reviewed educational infrastructure, *Proc. 1997 ASEE Annual Conference*, Milwaukee, WI, 1997.
5. T. F. Hess, R. F. Rynk, S. Chen, L. G. King and A. L. Kenimer, Natural systems for wastewater treatment: Course material and CD-ROM development, *ASEE Annual Conference Proc.*, Milwaukee, WI, 1997.
6. R. Guensler, P. Chinowsky and C. Conklin, Development of a Web-based environmental, impact, monitoring and assment course, *Proc. 1997 ASEE Annual Conference*, Milwaukee, WI, 1997.
7. M. G. J. M. Gerritsen, F. M. VanRappard, M. H. Baljon, N. V. Putten, W. R. M. Dassen, W. A. Dijk, DICOM CD-R, your guarantee to interchangeability?*Proc. 1997 24th Annual Meeting on Computers in Cardiology*, Lund, Sweden, 1997, pp. 175–178.
8. T. Yoshida, N. Yanagihara, Y. Mii, M. Soma, H. Yamada, Robust control of CD-ROM drives using multirate disturbance observer, *Trans. Jpn. Soc. Mech. Eng. (Part C)*, **63**(615): pp. 3919–3925, 1997.
9. J. Glanville and I. Smith, Evaluating the options for developing databases to support research-based medicine at the NHS Centre for Reviews and Dissemination, *Int. J. Med. Informatics*, **47**(1–2): pp. 83–86, 1997.
10. J.-L. Malleron and A. Juin, with R.-P. Rorer, Database of palladium chemistry: Reactions, catalytic cycles and chemical parameters on CD-ROM Version 1.0 *J. Amer. Chem. Soc.*, **120**(6): p. 1347, 1998.
11. M. S. Park, Y. Chait, M. Steinbuch, Inversion-free design algorithms for multivariable quantitative feedback theory: An application to robust control of a CD-ROM park, *Automatica*, **33**(5): pp. 915–920, 1997.
12. J.-H. Zhang and L. Cai, Profilometry using an optical stylus with interferometric readout, *Proc. IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Tokyo, Japan, p. 62, 1997.
13. E. W. Williams and T. Kubo, Cross-substitutional alloys of insb. for write-once read-many optical media, *Jpn. J. Appl. Phys. (Part 2)*, **37**(2A): pp. L127–L128, 1998.
14. P. Nicholls, Apocalypse now or orderly withdrawal for CD-ROM? *Comput. Libraries*, **18**(4): p. 57, 1998.
15. N. Honda, T. Ishiwaka, T. Takagi, M. Ishikawa, T. Nakajima, Information services for greater driving enjoyment, *SAE Special Publications on ITS Advanced Controls and Vehicle Navigation Systems, Proc. 1998 SAE International Congress & Exposition*, Detroit, MI, 1998, pp. 51–69.
16. J. K. Whitesell, Merck Index, 12th Edition, CD-ROM (Macintosh): An encyclopedia of chemicals, drugs & biologicals, *J. Amer. Chem. Soc.*, **120**(9): 1998.
17. C. Van Nimwegen, C. Zeelenberg, W. Cavens, Medical devices database on CD, *Proc. 1996 18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (Part 5)*, Amsterdam, 1996, pp. 1973–1974.
18. S. G. Stan, H. Van Kempen, G. Leenknegt, T. H. M. Akkermans, Look-ahead seek correction in high-performance CD-ROM drives, *IEEE Transactions on Consumer Electronics*, **44**(1): 178–186, 1998.
19. W. P. Murray, CD-ROM archivability, *NML Bits*, **2**(2): p. 4, 1992.
20. W. P. Murray, Archival life expectancy of 3M magneto-optic media, *J. Magnetic Soci. Japan*, **17**(S1): 309, 1993.
21. F. L. Podio, Research on methods for determining optical disc media life expectancy estimates, *Proc. SPIE*, **1663**: 447, 1992.
22. On CD-ROMs that set a new standard, *Technol. Rev.*, **101**(2): 1998.
23. National Geographic publishes 108 years on CD-ROM, *Imaging Mag.*, **7**(3): 1998.
24. Shareware Boosts CD-ROM performance, tells time, *EDN*, **43**(4): 1998.

25. Ford standardizes training with CD-ROMs, *Industrial Paint Powder*, **74**(2): 1998.

26. O. Widell and E. Egis, Geophysical information services, *Eur. Space Agency*, **SP-397**: 1997.

27. J. Tillinghast, G. Beretta, Structure and navigation for electronic publishing, *HP Laboratories Technical Report*, 97–162, Hewlett Packard Lab Technical Publ Dept, Palo Alto, CA, Dec. 1997.

28. J. Fry, A cornerstone of tomorrow's entertainment economy, *Proc. 1997 WESCON Conference*, San Jose, CA, 1997, pp. 65–73.

29. M. Kageyama, A. Ohba, T. Matsushita, T. Suzuki, H. Tanabe, Y. Kumagai, H. Yoshigi and T. Kinoshita, Free time-shift DVD video recorder, *IEEE Trans. Consumer Electron.*, **43**(3): 469–474, 1997.

30. S. P. Schreiner, M. Gaughan, T. Myint and R. Walentowicz, Exposure models of library and integrated model evaluation system: A modeling information system on a CD-ROM with World-Wide Web links, *Proc. 1997 4th IAWQ International Symposium on Systems Analysis and Computing in Water Quality Management*, Quebec, Canada, June 17–20, 1997, pp. 243–249.

31. Anonymous, software review, *Contr. Eng.*, **44**(15): 1997.

32. M. William, Using multimedia and cooperative learning in and out of class, *Proc. 1997 27th Annual Conference on Frontiers in Education.* Part 1 (of 3), Pittsburgh, PA, Nov. 5–8,1997 pp. 48–52.

33. P. G. Ranky, A methodology for supporting the product innovation process, *Proc. USA/Japan International IEEE Conference on Factory Automation*, Kobe, Japan, 1994, pp. 234–239.

34. P. Ashton and P. G. Ranky, The development and application of an advanced concurrent engineering research tool set at Rolls-Royce Motor Cars Limited, UK, *Proc. USA/Japan International IEEE Conference on Factory Automation*, Kobe, Japan, 1994, pp. 186–190.

35. K. L. Ho and P. G. Ranky, The design and operation control of a reconfigurable flexible material handling system, *Proc. USA/Japan International IEEE Conference on Factory Automation*, Kobe, Japan, 1994, pp. 324–328.

36. P. G. Ranky, The principles, application and research of interactive multimedia and open/distance learning in advanced manufacturing technology, Invited Keynote Presentation, The Fourth International Conference on Modern Industrial Training, Xi'lan, China, 1994, pp. 16–28.

37. D. A. Norman and J. C. Spohner, Learner-centered education, *Commun. ACM*, **39**(4): 24–27, 1996.

38. R. C. Schank and A. Kass, A goal-based scenario for high school students, *Commun. ACM*, **39**(4): 28–29, 1996.

39. B. Woolf, Intelligent multimedia tutoring systems, *Commun. ACM*, **39**(4): 30–31, 1996.

40. M. Flaherty, M. F. Ranky, P. G. Ranky, S. Sands and S. Stratful, FESTO: Servo Pneumatic Positioning, an Interactive Multimedia CD-ROM with off-line and on-line Internet support, Over 330 interactive screens, CIMware & FESTO Automation joint development 1995,96, Design & Programming by P.G. Ranky and M. F. Ranky. Available: http://www.cimwareukandusa.com.

41. P. G. Ranky, An Introduction to Total Quality (including ISO9000x), an Interactive Multimedia CD-ROM with off-line and on-line Internet support, over 700 interactive screens following an Interactive Multimedia Talking Book format, Design & Programming by P. G. Ranky and M. F. Ranky, CIMware 1997. Available: http://www.cimwareukandusa.com.

42. P. G. Ranky, An Introduction to Flexible Manufacturing, Automation & Assembly, an Interactive Multimedia CD-ROM with off-line and on-line Internet support, over 700 interactive screens following an Interactive Multimedia Talking Book format, Design & Programming by P. G. Ranky and M. F. Ranky, CIMware 1997. Available: http://www.cimwareukandusa.com.

Paul G. Ranky
New Jersey Institute of Technology
Newark, New Jersey
Gregory N. Ranky
Mick F. Ranky
Ridgewood, New Jersey

COMMUNICATION PROCESSORS FOR WIRELESS SYSTEMS

INTRODUCTION

In this article, we define the term *communication processor* as a device in a wired or wireless communication system that carries out operations on data in terms of either modifying the data, processing the data, or transporting the data to other parts of the system. A communication processor has certain optimizations built inside its hardware and/or software that enables it to perform its task in an efficient manner. Depending on the application, communication processors may also have additional constraints on area, real-time processing, and power, while providing the software flexibility close to general purpose microprocessors or microcontrollers. Although general purpose microprocessors and microcontrollers are designed to support high processing requirements or low power, the need to process data in real-time is an important distinction for communication processors.

The processing in a communication system is performed in multiple layers, according to the open systems interconnection (OSI) model. (For details on the OSI model, please see Ref. 1). When the communication is via a network of intermediate systems, only the lower three layers of the OSI protocols are used in the intermediate systems. In this chapter, we will focus on these lower, three layers of the OSI model, shown in Fig. 1.

The bottom-most layer is called the physical layer (or layer 1 in the OSI model). This layer serializes the data to be transferred into bits and sends it across a communication circuit to the destination. The form of communication can be wired using a cable or can be wireless using a radio device. In a wireless system, the physical layer is composed of two parts: the radio frequency layer (RF) and the baseband frequency layer. Both layers describe the frequency at which the communication circuits are working to process the transmitted wireless data. The RF layer processes signals at the analog level, whereas the baseband operations are mostly performed after the signal has been downconverted from the radio frequency to the baseband frequency and converted to a digital form for processing using a analog-to-digital converter. All signal processing needed to capture the transmitted signal and error correction is performed in this layer.

Above the physical layer is the data link layer, which is known more commonly as the medium access control (MAC) layer. The MAC layer is one of the two sub-layers in the data link layer of the OSI model. The MAC layer manages and maintains communication between multiple communication devices by coordinating access to a shared medium and by using protocols that enhance communication over that medium.

The third layer in the OSI model is the network layer. The network layer knows the address of the neighboring nodes in the network, packages output with the correct network address information, selects routes and quality of service (QOS), and recognizes and forwards to the transport layer incoming messages for local host domain.

Communication processors primarily have optimizations for the lower three layers of the OSI model. Depending on which layer has the most optimizations, communication processors are classified further into physical layer space (or baseband) processors, medium access control processors, or network processors.

The desire to support higher data rates in wireless communication systems implies meeting cost, area, power, and real-time processing requirements in communication processors. These constraints have the greatest impact on the physical layer design of the communication processor. Hence, although we mention the processing requirements of multiple layers, we focus this article on challenges in designing the physical layer of communication processors.

Evolution of Wireless Communication Systems

Over the past several years, communication systems have evolved from low data-rate systems for voice and data (with data rates of several Kbps, such as dial-up modems, cellular systems, and 802.11b local area networks) to high data-rate systems that support multimedia and video applications with data rates of several Mbps and going toward Gbps, such as DSL, cable modems, 802.11n local area networks (LANs), and ultra-wideband personal area networks (PANs) (2). The first generation systems (1G) came in the 1980s mostly for cellular analog voice using AMPS (advanced mobile phone service). This standard evolved into the second generation standard (2G) in the 1990s to support digital voice and low bit rate data services. An example of such a cellular system is IS-54 (2). At the same time, wireless local area networks began service starting at 1 Mbps for 802.11b standards and extending to 11 Mbps close to the year 2000. In the current generation of the standards (3G), cellular services have progressed to higher data rates in terms of hundreds of Kbps to support voice, data, and multimedia, and wireless LANs have evolved to 802.11a and 802.11g to, support data rates around 100 Mbps. In the future, for the fourth generation systems (4G), the data rates are expected to continue to increase and will provide IP-based services along with QoS (3). Table 1 presents the evolution of wireless communication systems as they have evolved from 1G to 4G systems. A range of data rates is shown in the table to account for both cellular and W-LAN data rates in communication systems.

CHALLENGES FOR COMMUNICATION PROCESSORS

This evolution of communication systems has involved radical changes in processor designs for these systems for multiple reasons. First, the increase in data rates has come at the cost of increased complexity in the system design.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

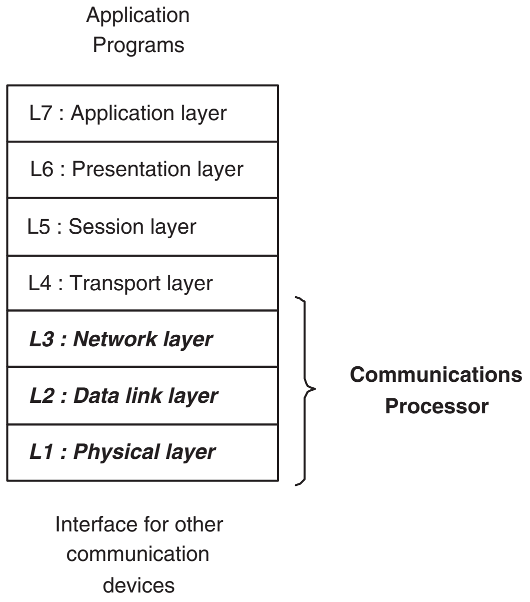


Figure 1. Layers in a OSI model. The communication processors defined in this chapter are processors that have specific optimizations for the lower three layers.

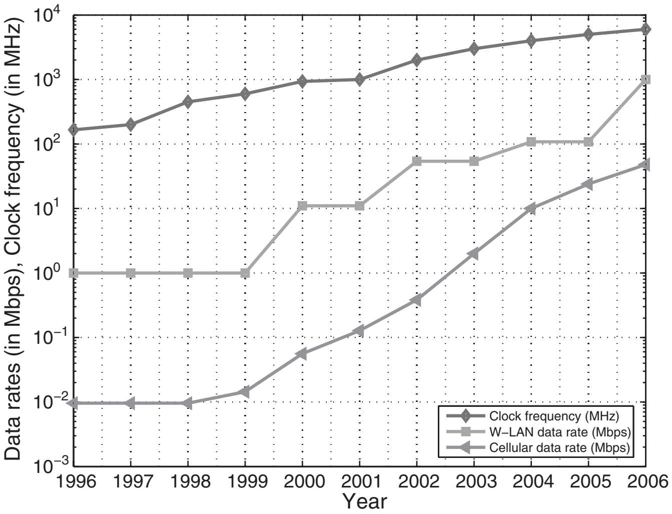


Figure 2. Increase in data rates for communication systems. The data rates in communication systems are increasing at a much greater rate than typical processor clock frequencies, necessitating new processor designs for communication system.

Second, the performance of communication systems have been increasing consistently as communication system designers develop sophisticated signal processing algorithms that enhance the performance of the system at the expense of increased computational complexity. Flexibility is also an important emerging characteristic in communication processors because of the need to support multiple protocols and environments. Also, newer applications have become more complex and they need to be backward-compatible with existing systems. As the number of standards and protocols increase, the demand increases for new standards to be spectrum-efficient, to avoid interference to other systems, and also to mitigate interference from other systems. The flexibility needed in the baseband and radio and regulatory requirements of spectrum and transmit power also add challenges in testing the design of these processors. The interaction and integration between different layers of the communication system also presents interesting challenges. The physical layer is signal processing-based, involving complex, mathematical computations, whereas the MAC layer is data processing-based, involving data movement, scheduling, and control of the physical layer. Finally, the range of consumer applications for communication systems has increased from small low-cost devices, such as RFID tags, to cellular phones, PDAs, laptops, personal computers, and high-end network servers. Processors for different applications have different optimizatio constraints such as the workload characteristics, cost, power, area, and data rate and require significant trade-off analysis. The above changes puts additional constraints on the processor design for communication systems.

Increasing Data Rates

Figure 2 shows the increase in data rates provided by communication systems over time. The figure shows that over the past decade, communication systems have had a 1000× increase in data rate requirements. Systems such as wireless LANs and PANs have evolved from 1 Mbps systems such as 802.11a and Bluetooth to 100 + Mbps 802.11b LANs to now Gbps systems being proposed for ultra-wideband personal area networks. The same has been true even for wired communication systems, going from 10 Mbps ethernet cards to now Gbps ethernet systems. The increase in processor clock frequencies across generations cannot keep up with the increase in raw data rate requirements. During the same period, the processor clock frequencies have only gone up by one order of magnitude. Also, applications (such as multimedia) are demanding more compute resources and more memory than previous processors. This demand implies that silicon process technology advances are insufficient to meet the increase in raw data rate requirements and additional architecture innovations such as exploiting parallelism, pipelining, and algorithm complexity reduction are needed to meet the data rate requirements. We discuss this in more detail in the section on area, time, and power tradeoffs.

Table 1. Evolution of communication systems

Generation	Year	Function	Data rates
1G	1980–1990	Analog voice	Kbps
2G	1990–2000	Voice + low-rate data	10 Kbps–10 Mbps
3G	2000–2010	Voice + data + multimedia	100 Kbps–100 Mbps
4G	2010–2020	Voice + data + multimedia + QoS + IP	10 Mbps–Gbps

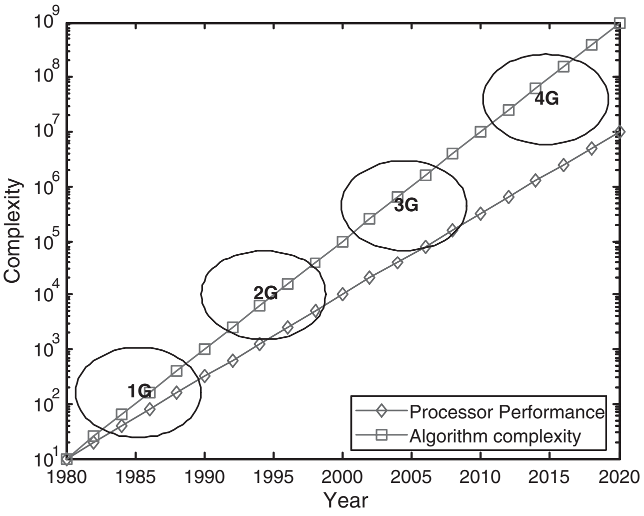


Figure 3. Algorithm complexity increasing faster than silicon process technology advances. (Reprinted with permission from Ref. 4.)

Increasing Algorithm Complexity

Although the data rate requirements of communication processors are increasing, the processor design difficulty is exacerbated by the introduction of more sophisticated algorithms that give significant performance improvements for communication systems. Figure 3 shows the increase in computational complexity as standards have progressed from first generation to second and third generations (4). The figure shows that even if the data rates are assumed constant, the increase in algorithmic complexity cannot be met solely with advances in silicon process technology.

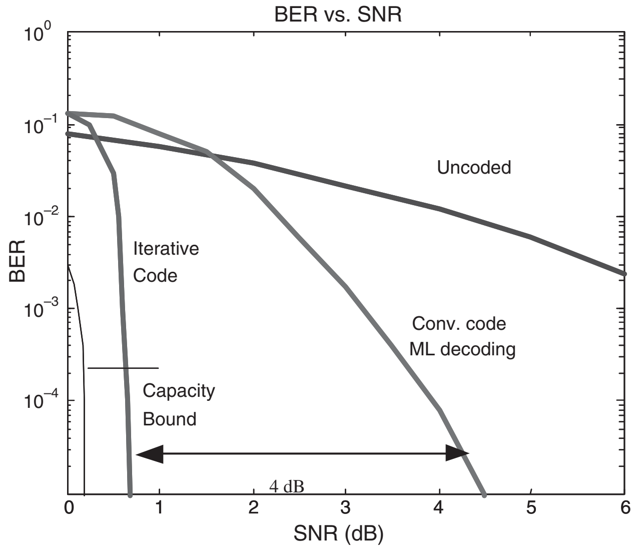


Figure 4. Decoder performance with advanced coding schemes. (Reprinted with permission from Ref. 7.)

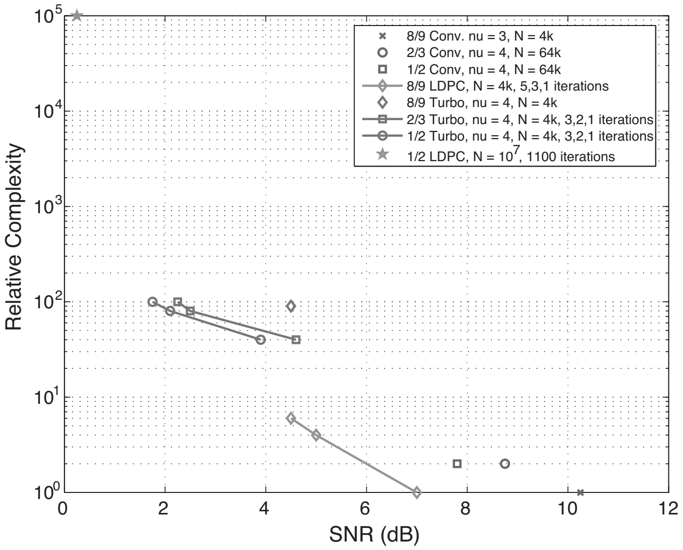


Figure 5. Decoder complexity for various types of coding schemes. (Reprinted with permission from Ref. 7.)

As an example, we consider decoding of error-control codes at the receiver of a communication processor. Figure 4 shows the benefits of coding in a communication system by reducing the bit error rate at a given signal-to-noise ratio. We can see that advanced coding schemes such as low density parity check codes (LDPC) (5) and turbo codes (6) which are iterative decoders that can give 4-dB benefits over conventional convolutional decoders. A 4-dB gain translates into roughly a 60% improvement in communication range of a wireless system. Such advanced coding schemes are proposed and implemented in standards such as HSDPA, VDSL, gigabit ethernet, digital video broadcast, and Wi-Fi. However, this improvement comes at a significant increase in computational complexity. Figure 5 shows the increased complexity of some advanced coding schemes (7). It can be observed that the iterative decoders have 3–5 orders of magnitude increase in computational complexity over convolutional decoders. Thus, to order to implement these algorithms, reduced complexity versions of these algorithms should be investigated for communication processors that allow simpler hardware designs with significant parallelism without significant loss in performance. An example of such a design is presented in Ref. 8.

Flexibility

As communication systems evolve over time, a greater need exists for communication processors to be increasingly flexible. Communication systems are designed to support several parameters such as variable coding rates, variable modulation modes, and variable frequency band. This flexibility allows the communication system to adapt itself better to the environment to maximize data rates over the channel and/or to minimize power. For example, Fig. 6 shows base-station computational requirements

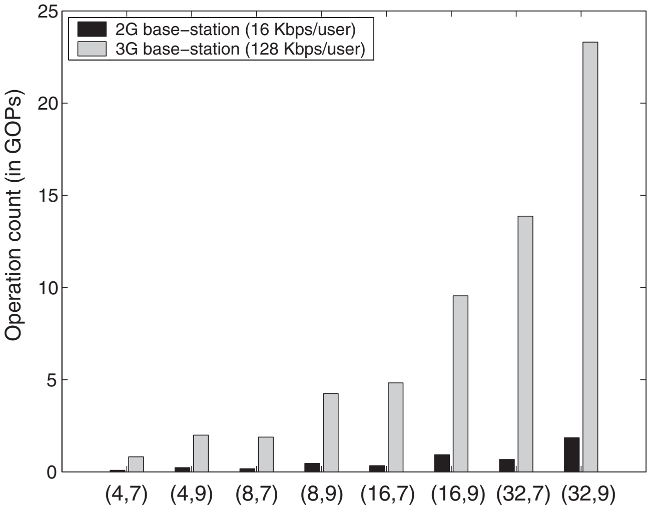


Figure 6. Flexibility needed to support various users, rates (for example), and backward -compatibility to standards. (Reprinted with permission from Ref. 9.)

and the flexibility needed to support several users at variable constraint lengths (9). The figure also shows an example of a 2G station at 16 Kbps/user supporting only voice and a 3G base-station at 128 Kbps/user supporting voice, data, and multimedia. A 3G base-station processor now must be backward-compatible to a 2G base-station processor and hence, must support both the standards as well as adapt its compute resources to save power when the processing requirements are lower. The amount of flexibility provided in communication processors can make the design for test for these systems extremely challenging because of the large number of parameters, algorithms, and radio interfaces that must be tested.

Along with the support for variable standards and protocols, researchers are also investigating the design of a single communication processor that can switch seamlessly between different standards, depending on the availability and cost of that standard. The RENE (Rice Everywhere NEtwork) project (10) demonstrates the design of a multitier network interface card with a communication processor that supports outdoor cellular (CDMA) and indoor wireless (LAN) and changes over the network seamlessly when the user moves from an office environment with wireless LAN into an outdoor environment using cellular services. Figure 7 shows the design of the wireless multitier network interface card concept at Rice University. Thus, flexibility to support various standards is becoming an increasingly desired feature in communication processors.

Spectrum Issues

The wireless spectrum is a scarce resource and is regulated by multiple agencies worldwide. As new standards evolve, they have to coexist with spectrums that are allocated already for existing standards. The regulatory bodies, such as Federal Communications Commission (see www.fcc.gov), demand that new standards meet certain limitations on transmit power and interference avoidance to make sure that the existing services are not degraded by the new standard. Also, because of a plethora of wireless standards in the 1–5 GHz wireless spectrum, new standards are forced to look at much higher RF frequencies, which make the design of radies more challenging as well as increase the need for transmit power because of larger attendation at higher frequencies. Newer standards also need to have interference detection and mitigation techniques to coexist with existing standards. This involves challenges at the radio level, such as to transmit at different frequencies to avoid interference and to develop the need for software-defined radios (11) Spectrum regulations have variations across countries worldwide and devices need to have the flexibility to support different programming to meet regulatory specifications.

Area, Time, and Power Tradeoffs

The design of communication processors is complicated even more by the nature of optimizations needed for the application and for the market segment. A mobile market segment may place greater emphasis on cost (area) and power, whereas a high-data rate market segment may place a greater focus on performance. Thus, even after new algorithms are designed and computationally efficient versions of the algorithms have been developed, tradeoffs between area-time and power consumptions occur for the implementation of the algorithm on the communication processor. Also, other parameters exist that need to be traded off such as the silicon process technology (0.18- vs. 0.13-vs. 0.09-μm CMOS process) and voltage and clock frequencies. For example, the area-time tradeoffs for Viterbi decoding are shown in Fig. 8(12). The curve shows that the area needed for the Viterbi decoder can be traded off at the cost of increasing the execution time for the Viterbi decoder.

In programmable processors, the number of functional units and the clock frequency can be adjusted to meet

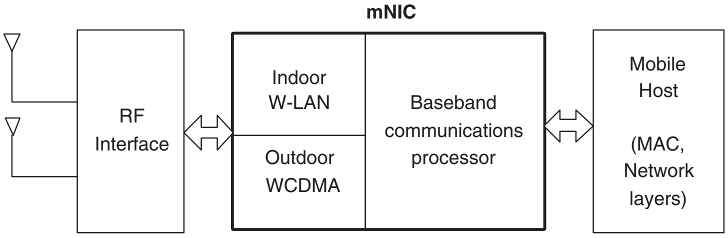


Figure 7. Multi-tier network interface card concept. (Reprinted with permission from Ref. 10.)

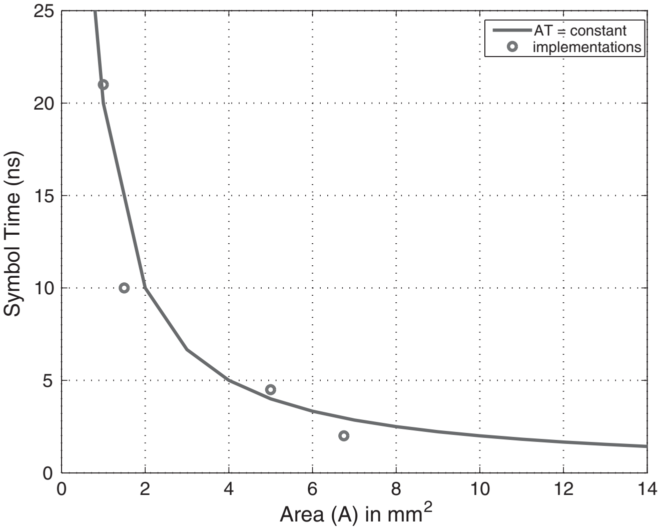


Figure 8. Normalized area-time efficiency for viterbi decoding. (Reprinted with permission from Ref. 12.)

real-time requirements for an application. An example of this application is shown in Fig. 9 (13). The figure shows that as the number of adders and multipliers in a programmable processor are increased, the clock frequency needed to meet real-time for an application decreases until a certain point, at which no more operations can be scheduled on the additional adders and multipliers in the processor. The numbers on the graph indicate the functional unit use of the adders and multipliers in the processor.

Interaction Between Multiple Layers

The interaction between the different layers in a communications system also presents challenges to the processor design. As will be shown in the following sections, the characteristics of the physical layer in a communication system are completely different than the characteristics of the MAC or network layer. The physical layer of a communication system consists of signal processing algorithms that work on estimation of the channel, detection of the received bits, and decoding of the data, and it requires computational resources. The MAC and network layers are more data flow-oriented and have more control and data-grouping operations. The combination of these two divers requirements make the task of design of a single integrated communication processor (which does both the PHY as well as the MAC) difficult. Hence, these layers are implemented typically as separate processors, although they may be present on the same chip.

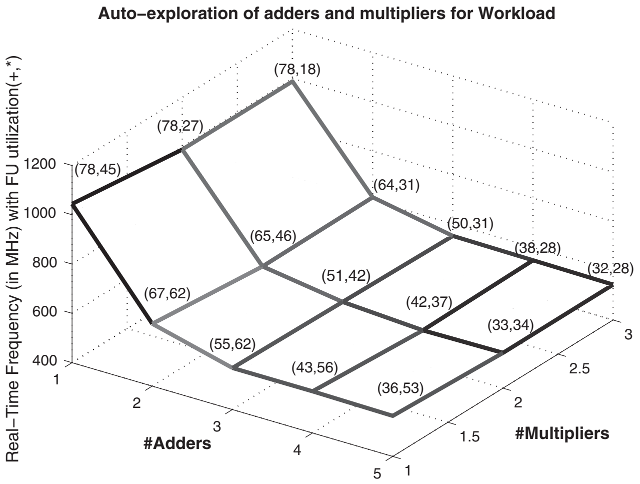


Figure 9. Number of adders and multipliers to meet real-time requirements in a programmable processor. (Reprinted with permission from Ref. 13.)

For example, it is very common in communication processor design to have a coprocessor-based approach for the physical layer that performs sophisticated mathematical operations in real-time, while having a microcontroller that handles the control and data management.

PHYSICAL LAYER OR BASEBAND PROCESSORS

The physical layer of wireless communication systems presents more challenges to the communication processor design than wired communication systems. The nature of the wireless channel implies the need for sophisticated algorithms on the receiver to receive and decode the data. Challenges exist in both the analog/RF radio and the digital baseband of the physical layer in emerging communication processors. The analog and RF radio design challenge is dominated by the need to support multiple communication protocols with varying requirements on the components in the transmitter and receiver chain of the radio. This need has emerged into a stream of research called software defined radios (11). We focus on the challenges in meeting the computational, real-time processing requirements and the flexibility requirements of the physical layer in the communication processor.

Characteristics of Baseband Communication Algorithms

Algorithms for communication systems in the physical layer process signals for transmission and reception of analog signals over the wireless (or the even wired) link. Hence, most algorithms implemented on communication processors are signal-processing algorithms and show certain characteristics that can be exploited in the design of communication processors.

1. Communication processors have stringent real-time requirements that imply the need to process data at a certain throughput rate while also meeting certain latency requirements.
2. Signal processing algorithms are typically compute-bound, which implies that the bottle neck in the processing are the computations (as opposed to memory) and the architectures require a significant number of adders and multipliers.

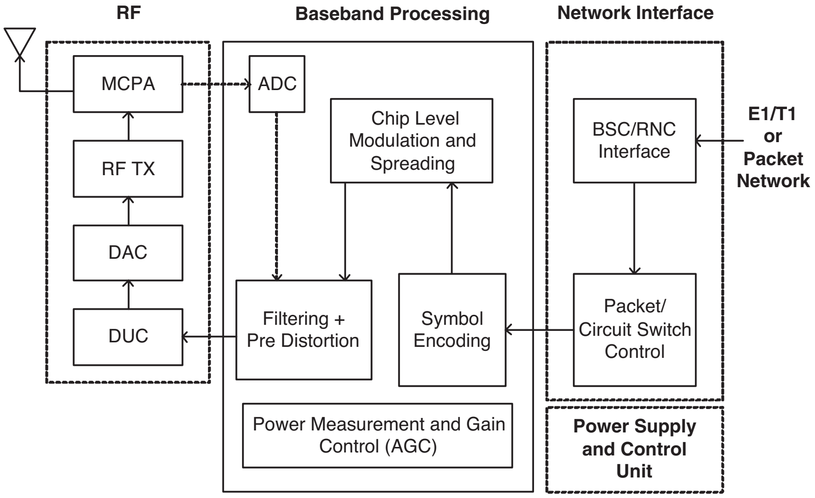

RF
Baseband Processing
Network Interface
MCPA
RF TX
DAC
DUC
ADC
Chip Level Modulation and Spreading
Filtering + Pre Distortion
Symbol Encoding
Power Measurement and Gain Control (AGC)
BSC/RNC Interface
Packet/ Circuit Switch Control
E1/T1 or Packet Network
Power Supply and Control Unit

Figure 10. Typical operations at a transmitter of a baseband processor. (Reprinted with permission from Texas Instruments.)

3. Communication processors require very low fixed-point precision in computations. At the transmitter, the inputs are sent typically as bits. At the receiver, the ADCs reduce the dynamic range of the input signal by quantizing the signal. Quantization in communication processors is acceptable because the quantization errors are typically small compared with the noise added through the wireless channel. This finding is very useful to design low power and high speed arithmetic and to keep the size of memory requirements small in communication processors.
4. Communication algorithms exhibit significant amounts of data parallelism and show regular patterns in computation that can be exploited for hardware design.
5. Communication algorithms have a streaming dataflow in a producer-consumer fashion between blocks with very little data reuse. This dataflow can be exploited to avoid storage of intermediate values and to eliminate hardware in processors such as caches that try to exploit temporal reuse.

Figure 10 shows a typical transmitter in a communication processor. The transmitter, in the physical layer of a communication system is typically much simpler compared with the receiver. The transmitter operations typically consist of taking the data from the MAC layer and then scrambling it to make it look sufficiently random, encoding it for error protection, modulating it on certain frequencies, and then precompensating it for any RF impairments or distortions.

Figure 11 shows a typical receiver in a communications processor. The receiver estimates the channel to compensate for it, and then it demodulates the transmitted data and decodes the data to correct for any errors during transmission. Although not shown in the figure, many other impairments in the channel and the radio, such as fading, interference, I/Q imbalance, frequency offsets, and

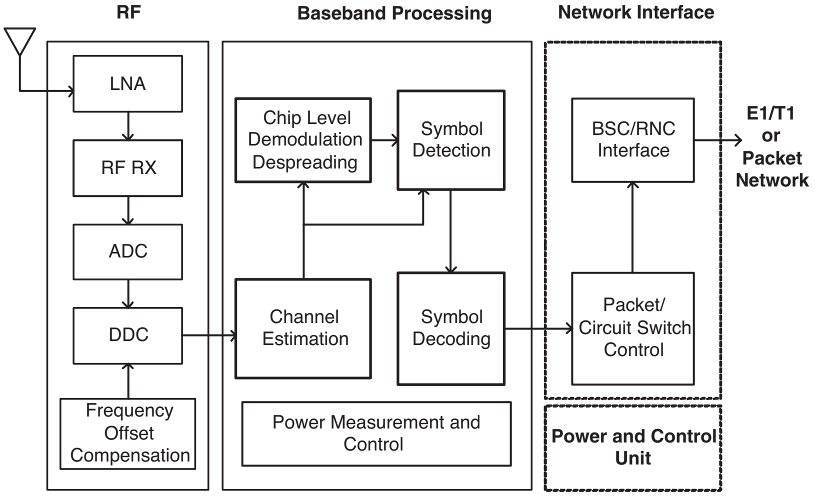


Figure 11. Typical operations at the receiver of a baseband processor (Reprinted with permission from Texas Instruments.)

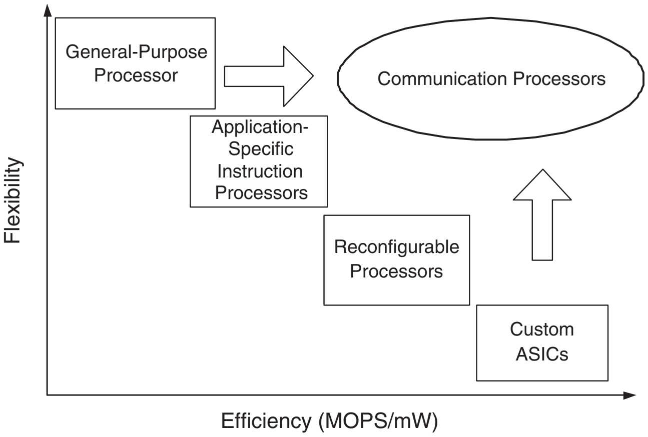


Figure 12. Desired characteristics of communication processors.

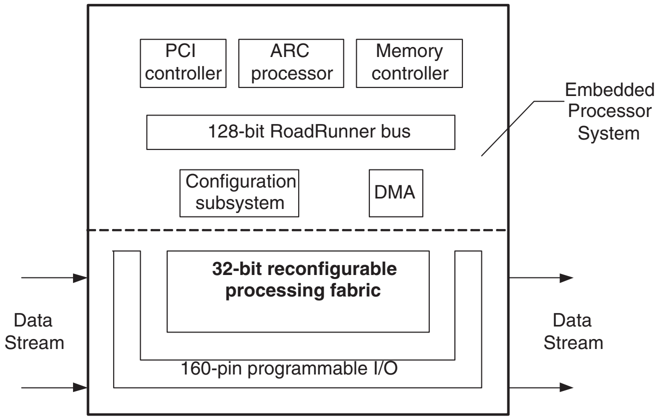


Figure 13. Reconfigurable communication processors. (Reprinted with permission from Ref. 15.)

phase offsets are also corrected at the receiver. The algorithms used at the receiver involve sophisticated signal processing and in general, have increased in complexity over time while providing more reliable and stable communication systems.

Architecture Designs

A wide range of architectures can be used to design a communication processor. Figure 12 shows the desired characteristics in communication processors and shows how different architectures meet the characteristics in terms of performance and flexibility. The efficiency metric on the x-axis is characterized as MOPs/mW (millions of operations performed per mW of power). The architectures shown in the figure trade flexibility with performance/power and are suited for different applications. A custom ASIC has the best efficiency in terms of data rate at unit power consumption at the same time it has the least amount of flexibility (14). On the other hand, a fully programmable processor is extremely flexible but is not area/power/throughput efficient. We discuss the tradeoffs among the different types of architectures to use them as communication processors.

Custom ASICs. Custom ASICs are the solution for communication processors that provide the highest efficiency and the lowest cost in terms of chip area and price. This, however, comes at the expense of a fairly large design and test time and lack of flexibility and scalability with changes in standards and protocols. Another issue with custom ASICs is that fabrication of these ASICs are extremely expensive (millions of dollars), which implies that extreme care needs to be taken in the functional design to ensure first pass success. Also, the volume of shipment for these custom chips must be high to amortize the development cost. A partial amount of flexibility can be provided as register settings for setting transmission or reception parameters or for tuning the chip that can then controlled by the MAC or higher layers in firmware (software). For example, the data rate to be used for transmission can be programmed into a register in the custom ASIC from the MAC and that can be used to set the appropriate controls in the processor.

Reconfigurable Processors. Reconfigurable processors are a relatively new addition to the area of communication processors. Typically, reconfigurable processors consist of a CISC type instruction set processor with a reconfigurable fabric attached to the processor core. This reconfigurable fabric is used to run complex signal processing algorithms that have sufficient parallelism and need a large number of adders and multipliers. The benefits of the reconfigurable fabric compared with FPGAs is that the reconfiguration can be done dynamically during run-time. Figure 13 shows an example of the Chameleon reconfigurable communication processor (15).

The reconfigurable fabric and the instruction set computing seek to provide the flexibility needed for communication processor while providing the dedicated logic in the reconfiguration fabric for efficient computing that can be reprogrammed dynamically. One of the major disadvantages of reconfigurable processors is that the software tools and compilers have not progressed to a state where performance/power benefits are easily visible along with the ease of programming the processor. The Chameleon reconfigurable processor is no longer an active product. However, several researchers in academia, such as GARP at Berkeley (16), RAW at MIT (17). Stallion at Virginia Tech (18), and in industry such as PACT (19) are still pursuing this promising architecture for communication processors.

Application-Specific Instruction Processors. Application-specific instruction processors (ASIPs) are processors with an instruction set for programmability and with customized hardware tailored for a given application (20). The programmability of these processors followed by the customization for a particular application to meet data rate and power requirements make ASIPs a viable candidate for communication processors.

A DSP is an example of such an application-specific instruction processor with specific optimizations to support signal processing operations. Because standards are typi-

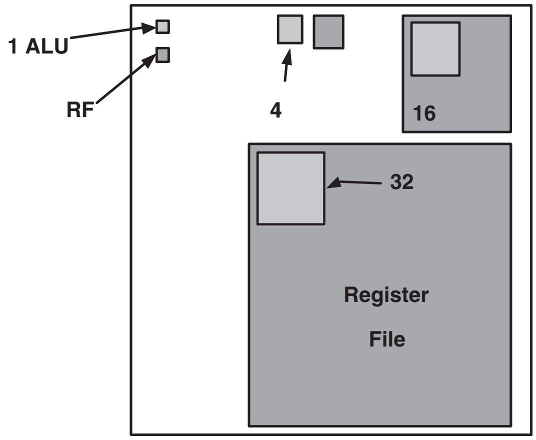


Figure 14. Register file expansion with increasing number of functional units in a processor. (Reprinted with permision from Ref. 23.)

cally driven by what is possible from an ASIC implementation feasibility for cost, performance, and power, it is difficult for a programmable architecture to compete with a fully custom, based ASIC design for wireless communications. DSPs fail to meet real-time requirements for implementing sophisticated algorithms because of the lack of sufficient functional units. However, it is not simple to increase the number of adders and multipliers in a DSP. Traditional single processor DSP architectures such as the C64x DSP by Texas Instruments (Dallas, TX) (21) employ VLIW architectures and exploit instruction level parallelism (ILP) and subword parallelism. Such single-processor DSPs can only have limited arithmetic units (less than 10) and cannot extend directly their architectures to 100s of arithmetic units. This limitation is because as the number of arithmetic units increases in an architecture, the size of the register files increases and the port interconnections start dominating the chip area (21,22). This growth is shown as a cartoon in Fig. 14 (28). Although the use of distributed register files may alleviate the register file explosion at the cost of increased pepalty in register allocation (21), an associated cost exists in exploiting ILP because of the limited size of register files, dependencies in the computations, and the register and functional unit allocation and use efficiency of the compiler. It has been shown that even with extremely good techniques, it is very difficult to exploit ILP beyond 5 (24). The large number of arithmetic and logic units (ALUs) also make the task of compiling and scheduling algorithms on the ALUs and keeping all the ALUs busy difficult.

Another popular approach to designing communication processors is to use a DSP with coprocessors (25–27). The coprocessors are still needed to perform more sophisticated operations that cannot be done real-time on the DSP because of the lack of sufficient adders and multipliers. Coprocessor support in a DSP can be both tightly coupled and loosely coupled (27). In a tightly coupled coprocessor (TCC) approach, the coprocessor interfaces directly to the DSP core and has access for specific registers in the DSP core. The TCC approach is used for algorithms that work with small datasets and require only a few instruction cycles to complete. The DSP processor freezes when the coprocessor is used because the DSP will have to interrupt the coprocessor immediately in the next few cycles. In time, the TCC is integrated into the DSP core with a specific instruction or is replaced with code in a faster or lower-power DSP. An example of such a TCC approach would be the implementation of a Galois field bit manipulation that may not be part of the DSP instruction set (27). The loosely coupled coprocessor approach (LCC) is used for algorithms that work with large datasets and require a significant amount of cycles to complete without interruption from the DSP. The LCC approach allows the DSP and coprocessor to execute in parallel. The coprocessors are loaded with the parameters and data and are initiated through application-specific instructions. The coprocessors sit on an external bus and do not interface directly to the DSP core, which allows the DSP core to execute in parallel. Figure 15 shows an example of the TMS320C6416 processor from Texas Instruments which has Viterbi and Turbo coprocessors for decoding (28) using the LCC approach. The DSP provides the flexibility needed for applications and the coprocessors provide the compute resources for more sophisticated computations that are unable to be met on the DSP.

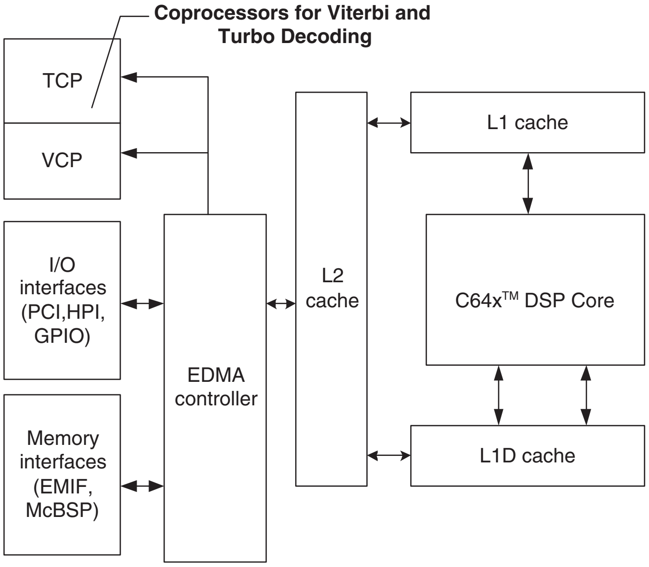

Coprocessors for Viterbi and Turbo Decoding
TCP
VCP
I/O interfaces (PCI,HPI, GPIO)
Memory interfaces (EMIF, McBSP)
EDMA controller
L2 cache
L1 cache
C64x^{TM} DSP Core
L1D cache

Figure 15. DSP with coprocessors for decoding, (Reprinted with permission from Ref. 28.)

Programmable Processors. To be precise with definitions, in this subsection, we consider programmable processors as processors that do not have an application-specific optimization or instruction set. For example, DSPs without coprocessors are considered in this subsection as programmable processors.

Stream processors are programmable processors that have optimizations for media and signal processing. They are able provide hundreds of ALUs in a processor by

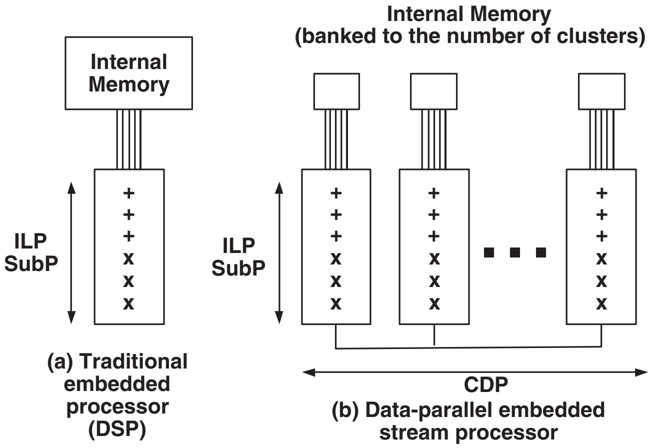

Internal Memory
Internal Memory
(banked to the number of clusters)
ILP
SubP
ILP
SubP
+
+
+
x
x
x
CDP
(a) Traditional embedded processor (DSP)
(b) Data-parallel embedded stream processor

Figure 16. DSP and stream processors. (Reprinted with permission from Ref. 13.)

arranging the ALUs into groups of clusters and by exploiting data parallelism across clusters. Stream processors are able to support giga-operations per second of computation in the processor. Figure 16 shows the distinction between DSPs and stream processors. Although typical DSPs exploit ILP and sub-word parallelism (SubP), stream processors also exploit data-parallelism across clusters to provide the needed computational horsepower.

Streams are stored in a stream register file, which can transfer data efficiently to and from a set of local register files between major computations. Local register files, colocated with the arithmetic units inside the clusters, feed those units directly with their operands. Truly global data, data that is persistent throughout the application, is stored off-chip only when necessary. These three explicit levels of storage form an efficient communication structure to keep hundreds of arithmetic units efficiently fed with data. The Imagine stream processor developed at Stanford is the first implementation of such a stream processor (29). Figure 17 shows the architecture of a stream processor with $C + 1$ arithmetic clusters. Operations in a stream processor all consume and/or produce streams that are stored in the centrally located stream register file (SRF). The two major stream instructions are memory transfers and kernel operations. A stream memory transfer either loads an entire stream into the SRF from external memory or stores an entire stream from the SRF to external memory. Multiple stream memory transfers can occur simultaneously, as hardware resources allow. A kernel operation performs a computation on a set of input streams to produce a set of output streams. Kernel operations are performed within a data parallel array of arithmetic clusters. Each cluster performs the same sequence of operations on independent stream elements. The stream buffers (SBs) allow the single port into the SRF array (limited for area/power/delay reasons) to be time-multiplexed among all the interfaces to the SRF, making it seen that many logical ports exist the array. The SBs also act as prefetch buffers and prefetch the data for kernel operations. Both the SRF and the stream buffers are banked to match the number of clusters. Hence, kernels that need to access data in other SRF banks must use the intercluster communication network for communicating data between the clusters.

The similarity between stream computations and communication processing in the physical layer makes stream-based processors an attractive architecture candidate for communication processors (9).

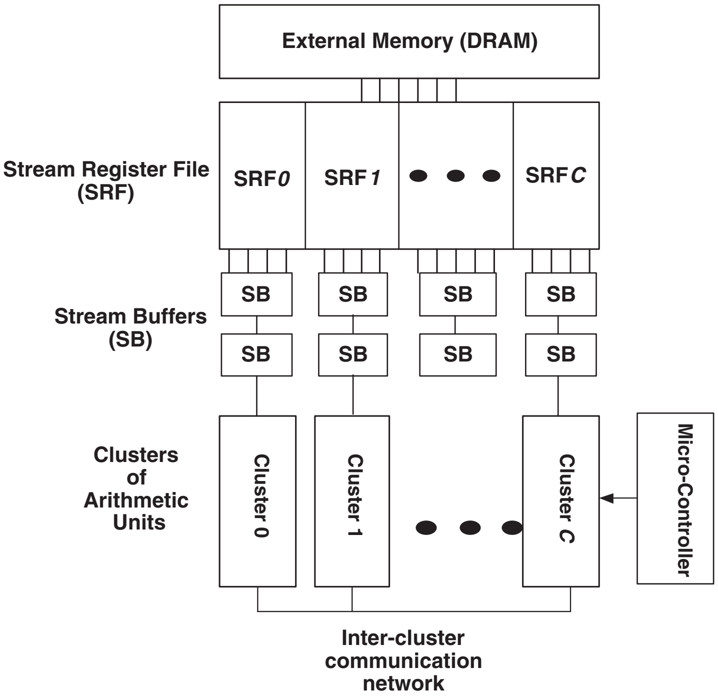


Figure 17. Stream processor architecture. (Reprinted with permission from Ref. 25.)

MAC AND NETWORK PROCESSORS

Although the focus of this article is on the physical layer of the communication processor, the MAC and network layers have a strong interaction with the physical layer especially in wireless networks. In this section, we briefly discuss the challenges and the functionality needed in processors for MAC and network layers (30).

MACs for wireless networks involve greater challenges than MACs for wired networks. The wireless channel necessitates the need for retransmissions when the received data is not decoded correctly in the physical layer. Wireless MACs also need to send out beacons to notify the access point that an active device is present on the network.

Typical functions of a wireless MAC include:

1. Transmissions of beacons in regular intervals to indicate the presence of the device on the network.
2. Buffering frames of data that are received from the physical layer and sending requests for re-transmissions for lost frames.
3. Monitoring radio channels for signals, noise and interference.
4. Monitoring presence of other devices on the network.
5. Encryption of data using AES/DES to provide security over the wireless channel.
6. Rate control of the physical layer to decide what data rates should be used for transmission of the data.

From the above, it can be seen that the MAC layer typically involves significant data management and processing. Typically, MACs are implemented as a combination of a RISC core that provides the control to different parts or the processor and dedicated logic for parts such as encryption for security and host interfaces.

Some functions of the network layer can be implemented on the MAC layer and vice-versa, depending on the actual protocol and application used. Typical functions at the network layer include:

1. Pattern matching and lookup. This involves matching the IP address and TCP port.
2. Computation of checksum to see if the frame is valid and any additional encryption and decryption.
3. Data manipulation that involves extracting and insertion of fields in the IP header and also, fragmentation and reassembly of packets.
4. Queue management for low priority and high priority traffic for QoS.
5. Control processing for updating routing tables and timers to check for retransmissions and backoff and so on.

CONCLUSIONS

Communication processor designs are evolving rapidly as silicon process technology advances have proven unable to keep up with increasing data rates and algorithm complexity. The need for greater flexibility to support multiple protocols and be backward compatible exacerbates the design problem because of the need to design programmable solutions that can provide high throughput and meet real-time requirements while being area and power efficient. The stringent regulatory requirements on spectrum, transmit power, and interference mitigation makes the design of the radio difficult while the complexity, diverse processing characteristics, and interaction between the physical layers and the higher layers complicates the design of the digital part of the communication processor Various tradeoffs can be made in communication processors to optimize throughputs versus area versus power versus cost, and the decisions depend the actual application under consideration. We present a detailed look at the challenges involved in designing these processors and present sample communication processor architectures that are considered for communication processors in the future.

ACKNOWLEDGMENTS

Sridhar Rajagopal and Joseph Cavallaro were supported in part by Nokia Corporation, Texas Instruments, Inc., and by NSF under grants EIA-0224458, and EIA-0321266.

BIBLIOGRAPHY

1. H. Zimmermann, OSI reference model – The ISO model of architecture for open systems interconnection, *IEEE Trans. Communicat.*, **28**: 425–432, 1980.
2. T. Ojanpera and R. Prasad, ed., *Wideband CDMA for Third Generation Mobile Communications*, Norwood, MA: Artech House Publishers, 1998.
3. H. Honkasalo, K. Pehkonen, M. T. Niemi, and A. T. Leino, WCDMA and WLAN for 3G and beyond, *IEEE Wireless Communicat.,* **9**(2): 14–18, 2002.
4. J. M. Rabaey, Low-power silicon architectures for wireless communications, *Design Automation Conference ASP-DAC 2000, Asia and South Pacific Meeting*, Yokohama, Japan, pp. 377–380, 2000.
5. T. Richardson and R. Urbanke, The renaissance of Gallager's low-density parity-check codes, *IEEE Communicat. Mag.*, 126–131, 2003.
6. B. Vucetic and J. Yuan, *Turbo Codes: Principles and Applications*, 1st ed., Dordrecht: Kluwer Academic Publishers, 2000.
7. E. Yeo, Shannon's bound: at what costs? Architectures and implementations of high throughput iterative decoders, *Berkeley Wireless Research Center Winter Retreat*, 2003.
8. S. Rajagopal, S. Bhashyam, J. R. Cavallaro, and B. Aazhang, Real-time algorithms and architectures for multiuser channel estimation and detection in wireless base-station receivers, *IEEE Trans. Wireless Commmunicat.*, **1**(3): 468–479, 2002.
9. S. Rajagopal, S. Rixner, and J. R. Cavallaro, Improving power efficiency in stream processors through dynamic cluster reconfiguration, *Workshop on Media and Streaming Processors*, Portland, OR, 2004.

10. B. Aazhang and J. R. Cavallaro, Multi-tier wireless communications, *Wireless Personal Communications, Special Issue on Future Strategy for the New Millennium Wireless World, Kluwer*, **17**: 323–330, 2001.

11. J. H. Reed, ed., *Software Radio: A Modern Approach to Radio Engineering*, Englewood Cliffs, NJ: Prentice Hall, 2002.

12. T. Gemmeke, M. Gansen, and T. G. Noll, Implementation of scalable and area efficient high throughput viterbi decoders, *IEEE J. Solid-State Circuits*, **37**(7): 941–948, 2002.

13. S. Rajagopal, S. Rixner, and J. R. Cavallaro, Design-space exploration for real-time embedded stream processors, *IEEE Micro* **24**(4): 54–66, 2004.

14. N. Zhang, A. Poon, D. Tse, R. Brodersen, and S. Verdú, Tradeoffs of performance and single chip implementation of indoor wireless multi-access receivers, *IEEE Wireless Communications and Networking Conference (WCNC)*, vol. 1, New Orleans, LA, September 1999, pp. 226–230.

15. B. Salefski and L. Caglar, Re-configurable computing in wireless, *Design Automation Conference*, Las Vegas, NV, 2001, pp. 178–183.

16. T. C. Callahan, J. R. Huser, and J. Wawrzynek, The GARP architecture and C compiler, *IEEE Computer*, 62–69, 2000.

17. A. Agarwal, RAW computation, *Scientific American*, **28l**(2): 60–63, 1999.

18. S. Srikanteswara, R. C. Palat, J. H. Reed, and P. Athanas, An overview of configurable computing machines for software radio handsets, *IEEE Communicat. Mag.*, 2003, pp. 134–141.

19. PACT: eXtreme Processing Platform (XPP) white paper. Available: http://www.pactcorp.com.

20. K. Keutzer, S. Malik, and A. R. Newton, From ASIC to ASIP: The next design discontinuity, *IEEE International Conference on Computer Design*, 2002, pp. 84–90.

21. S. Rixner, W. J. Dally, U. J. Kapasi, B. Khailany, A. Lopez-Lagunas, P. R. Mattson, and J. D. Owens, A bandwidth-efficient architecture for media processing, *31st Annual ACM/IEEE International Symposium on Microarchitecture (Micro-31)*, Dallas, TX, 1998, pp. 3–13.

22. H. Corporaal. *Microprocessor Architectures - from VLIW to TTA*, 1st ed., Wiley International, 1998.

23. S. Rixner, *Stream Processor Architecture*, Dordrecht: Kluwer Academic Publishers, 2002.

24. D. W. Wall, Limits of Instruction-Level Parallelism, *4th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Santa Clara, CA, 199l, pp. 176–188.

25. C-K. Chen, P-C. Tseng, Y-C. Chang, and L-G. Chen, A digital signal processor with programmable correlator architecture for third generation wireless communication system, *IEEE Trans. Circuits Systems-II: Analog Digital Signal Proc.*, **48**(12): 1110–1120, 2001.

26. A. Gatherer and E. Auslander, eds., *The Application of Programmable DSPs in Mobile Communications*, New York: John Wiley and Sons, 2002.

27. A. Gatherer, T. Stetzler, M. McMahan, and E. Auslander, DSP-based architectures for mobile communications: past, present and future, *IEEE Communicat. Mag.*, **38**(1): 84–90, 2000.

28. S. Agarwala, et al., A 600 MHz VLIW DSP, *IEEE J. Solid-State Circuits*, **37**(11): 1532–1544, 2002.

29. U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, Programmable stream processors, *IEEE Computer*, **36**(8): 54–62, 2003.

30. P. Crowley, M. A. Franklin, H. Hadimioglu, and P.Z. Onufryk, *Network Processor Design: Issues and Practices*, vol. l, San Francisco, CA: Morgan Kaufmann Publishers, 2002.

SRIDHAR RAJAGOPAL
WiQuest Communications, Inc.
Allen, Texas
JOSEPH R. CAVALLARO
Rice University
Houston, Texas

C

COMPUTER ARCHITECTURE

The term *computer architecture* was coined in the 1960s by the designers of the IBM System/360 to mean the structure of a computer that a machine language programmer must understand to write a correct program for a machine (1).

The task of a computer architect is to understand the state-of-the-art technologies at each design level and the changing design tradeoffs for their specific applications. The tradeoff of cost, performance, and power consumption is fundamental to a computer system design. Different designs result from the selection of different points on the cost-performance-power continuum, and each application will require a different optimum design point. For high-performance server applications, chip and system costs are less important than performance. Computer speedup can be accomplished by constructing more capable processor units or by integrating many processors units on a die. For cost-sensitive embedded applications, the goal is to minimize processor die size and system power consumption.

Technology Considerations

Modern computer implementations are based on silicon technology. The two driving parameters of this technology are die size and feature size. Die size largely determines cost. Feature size is dependent on the lithography used in wafer processing and is defined as the length of the smallest realizable device. Feature size determines circuit density, circuit delay, and power consumption. Current feature sizes range from 90 nm to 250 nm. Feature sizes below 100 nm are called deep submicron. Deep submicron technology allows microprocessors to be increasingly more complicated. According to the Semiconductor Industry Association (2), the number of transistors (Fig. 1) for high-performance microprocessors will continue to grow exponentially in the next 10 years. However, there are physical and program behavioral constraints that limit the usefulness of this complexity. Physical constraints include interconnect and device limits as well as practical limits on power and cost. Program behavior constraints result from program control and data dependencies and unpredictable events during execution (3).

Much of the improvement in microprocessor performance has been a result of technology scaling that allows increased circuit densities at higher clock frequencies. As feature sizes shrink, device area shrinks roughly as the square of the scaling factor, whereas device speed (under constant field assumptions) improves linearly with feature size.

On the other hand, there are a number of major technical challenges in the deep submicron era, the most important of which is that interconnect delay (especially global interconnect delay) does not scale with the feature size. If all three dimensions of an interconnect wire are scaled down by the same scaling factor, the interconnect delay remains roughly unchanged, because the fringing field component of wire capacitance does not vary with feature size. Consequently, interconnect delay becomes a limiting factor in the deep submicron era.

Another very important technical challenge is the difficulty faced trying to dissipate heat from processor chip packages as chip complexity and clock frequency increases.

Indeed, special cooling techniques are needed for processors that consume more than 100W of power. These cooling techniques are expensive and economically infeasible for most applications (e.g., PC). There are also a number of other technical challenges for high-performance processors. Custom circuit designs are necessary to enable GHz signals to travel in and out of chips. These challenges require that designers provide whole-system solutions rather than treating logic design, circuit design, and packaging as independent phases of the design process.

Performance Considerations

Microprocessor performance has improved by approximately 50% per year for the last 20 years, which can be attributed to higher clock frequencies, deeper pipelines, and improved exploitation of instruction-level parallelism. However, the cycle time at a given technology cannot be too small, or we will sacrifice overall performance by incurring too much clock overhead and suffering long pipeline breaks. Similarly, the instruction-level parallelism is usually limited by the application, which is further diminished by code generation inefficiencies, processor resource limitations, and execution disturbances. The overall system performance may deteriorate if the hardware to exploit the parallelism becomes too complicated.

High-performance server applications, in which chip and system costs are less important than total performance, encompass a wide range of requirements, from computation-intensive to memory-intensive to I/O-intensive. The need to customize implementation to specific applications may even alter manufacturing. Although expensive, high-performance servers may require fabrication microproduction runs to maximize performance.

Power Considerations

Power consumption has received increasingly more attention because both high-performance processors and processors for portable applications are limited by power consumption. For CMOS design, the total power dissipation has three major components as follows:

1. switching loss,
2. leakage current loss, and
3. short circuit current loss.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

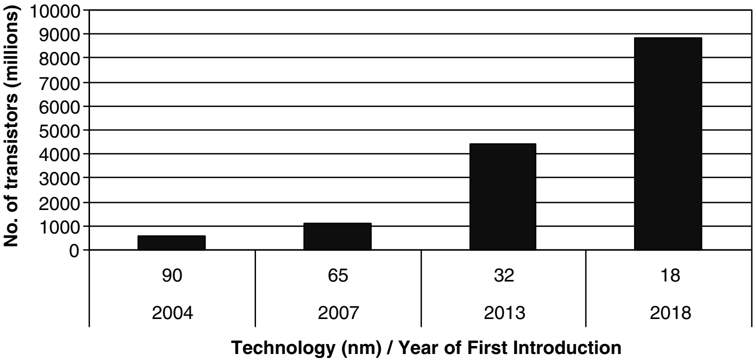


Figure 1. Number of transistors per chip. (Source: National Technology Roadmap for Semiconductors)

Among these three factors, switching loss is usually the most dominant factor. Switching loss is proportional to operating frequency and is also proportional to the square of supply voltage. Thus, lowering supply voltage can effectively reduce switching loss. In general, operating frequency is roughly proportional to supply voltage. If supply voltage is reduced by 50%, operating frequency is also reduced by 50%, and total power consumption becomes one-eighth of the original power. On the other hand, leakage power loss is a function of CMOS threshold voltage. As supply voltage decreases, threshold voltage has to be reduced, which results in an exponential increase in leakage power loss. When feature size goes below 90 nm, leakage power loss can be as high as switching power loss.

For many DSP applications, the acceptable performance can be achieved at a low operating frequency by exploiting the available program parallelism using suitable parallel forms of processor configurations. Improving the battery technology, obviously, can allow processors to run for an extended period of time. Conventional nickel-cadmium battery technology has been replaced by high-energy density batteries such as the NiMH battery. Nevertheless, the energy density of a battery is unlikely to improve drastically for safety reasons. When the energy density is too high, a battery becomes virtually an explosive.

Cost Considerations

Another design tradeoff is to determine the optimum die size. In the high-performance server market, the processor cost may be relatively small compared with the overall system cost. Increasing the processor cost by 10 times may not significantly affect the overall system cost. On the other hand, system-on-chip implementations tend to be very cost sensitive. For these applications, the optimum use of die size is extremely important.

The area available to a designer is largely a function of the manufacturing processing technology, which includes the purity of the silicon crystals, the absence of dust and other impurities, and the overall control of the diffusion and process technology. Improved manufacturing technology allows larger die with higher yields, and thus lower manufacturing costs.

At a given technology, die cost is affected by chip size in two ways. First, as die area increases, fewer die can be realized from a wafer. Second, as the chip size increases, the yield decreases, generally following a Poisson distribution of defects. For certain die sizes, doubling the area can increase the die cost by 10 times.

Other Considerations

As VLSI technology continues to improve, there are new design considerations for computer architects. The simple traditional measures of processor performance—cycle time and cache size—are becoming less relevant in evaluating application performance. Some of the new considerations include:

1. Creating high-performance processors with enabling compiler technology.
2. Designing power-sensitive system-on-chip processors in a very short turnaround time.
3. Improving features that ensure the integrity and reliability of the computer.
4. Increasing the adaptability of processor structures, such as cache and signal processors.

Performance-Cost-Power Tradeoffs

In the era of deep-submicron technology, two classes of microprocessors are evolving: (1) high-performance server processors and (2) embedded client processors. The majority of implementations are commodity system-on-chip processors devoted to end-user applications. These highly cost-sensitive client processors are used extensively in consumer electronics. Individual application may have specific requirements; for example, portable and wireless applications require very low power consumption. The other class consists of high-end server processors, which are performance-driven. Here, other parts of the system dominate cost and power issues.

At a fixed feature size, area can be traded off for performance (expressed in term of execution time, T). VLSI complexity theorists have shown that an $A \cdot T^n$ bound exists for microprocessor designs (1), where n usually falls between 1 and 2. By varying the supply voltage, it is also possible to tradeoff time T for power P with a $P \cdot T^3$ bound.

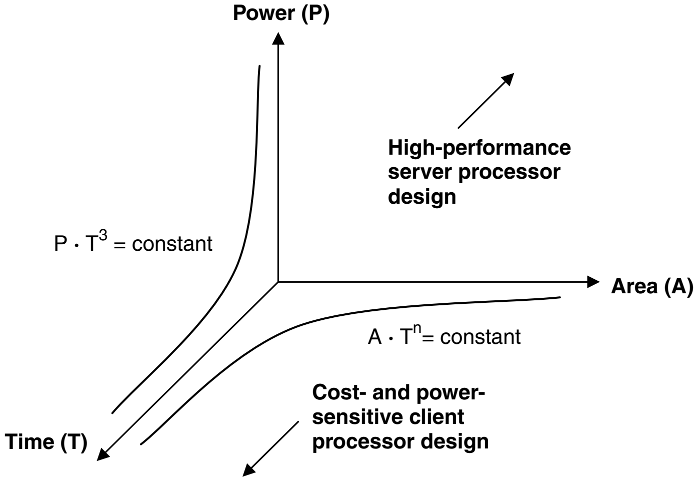

Power (P)
High-performance server processor design
P · T^3 = constant
Area (A)
A · T^n= constant
Cost- and power-sensitive client processor design
Time (T)

Figure 2. Design tradeoffs for high-end and low-end processors.

Figure 2 shows the possible tradeoff involving area, time, and power in a processor design (3). Embedded and high-end processors operate in different design regions of this three-dimensional space. The power and area axes are typically optimized for embedded processors, whereas the time axis is typically optimized for high-end processors.

Alternatives in Computer Architecture

In computer architecture, the designer must understand the technology and the user requirements as well as the available alternatives in configuring a processor. The designer must apply what is known of user program behavior and other requirements to the task of realizing an area-time-power optimized processor. User programs offer differing types and forms of parallelism that can be matched by one or more processor configurations. A primary design goal is to identify the most suitable processor configuration and then scale the concurrency available within that configuration to match cost constraints.

The next section describes the principle functional elements of a processor. Then the various types of parallel and concurrent processor configuration are discussed. Finally, some recent architectures are compared and some concluding remarks are presented.

PROCESSOR ARCHITECTURE

The processor architecture consists of the instruction set, the memory that it operates on, and the control and functional units that implement and interpret the instructions. Although the instruction set implies many implementation details, the resulting implementation is a great deal more than the instruction set. It is the synthesis of the physical device limitations with area-time-power tradeoffs to optimize cost-performance for specified user requirements. As shown in Fig. 3, the processor architecture may be divided into a high-level programming model and a low-level microarchitecture.

Instruction Set

Computers deal with many different kinds of data and data representations. The operations available to perform the requisite data manipulations are determined by the data types and the uses of such data. Processor design issues are closely bound to the instruction set. Instruction set behavior data affects many of these design issues

The *instruction set* for most modern machines is based on a register set to hold operands and addresses. The register set size varies from 8 to 64 words, each word consisting of 32 to 64 bits. An additional set of floating-point registers (16 to 64 bits) is usually also available. A

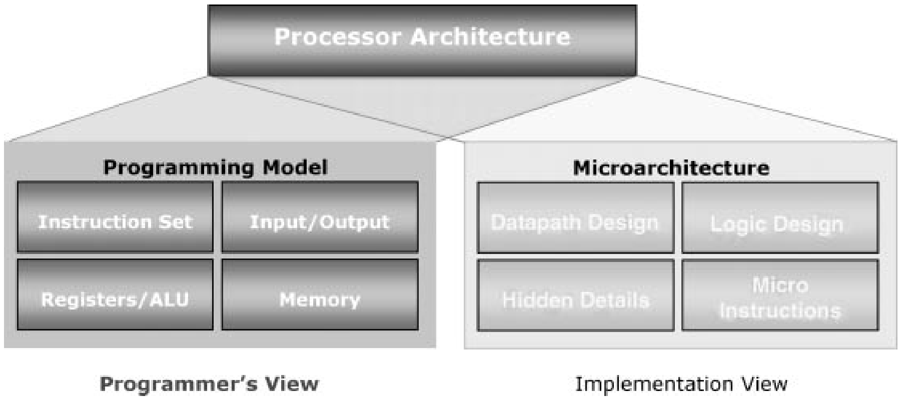


Figure 3. Processor architecture block diagram.

typical instruction set specifies a program status word, which consists of various types of control status information, including condition codes set by the instruction. Common instruction sets can be classified by format differences into three types as follows:

1. the L/S, or Load-Store architecture;
2. the R/M, or Register-Memory architecture; and
3. the R+M, or Register-plus-Memory architecture.

The *L / S* or *Load-Store* instruction set describes many of the RISC (reduced instruction set computer) microprocessors (5). All values must be loaded into registers before an execution can take place. An ALU ADD instruction must have both operands with the result specified as registers (three addresses). The purpose of the RISC architecture is to establish regularity of execution and ease of decoding in an effort to improve overall performance. RISC architects have tried to reduce the amount of complexity in the instruction set itself and regularize the instruction format so as to simplify decoding of the instruction. A simpler instruction set with straightforward timing is more readily implemented. For these reasons, it was assumed that implementations based on the L/S instruction set would always be faster (higher clock rates and performance) than other classes, other parameters being generally the same.

The *R / M* or *Register-Memory* architectures include instruction that can operate both on registers and with one of the operands residing in memory. Thus, for the R/M architecture, an ADD instruction might be defined as the sum of a register value and a value contained in memory, with the result going to a register. The R/M instruction sets generally trace their evolution to the IBM System/360 introduced in 1963. The mainframe computers follow the R/M style (IBM, Amdahl, Hitachi, Fujitsu, etc., which all use the IBM instruction set), as well as the basic Intel x86 series.

The *R+M* or *Register-plus-Memory* architectures allow formats to include operands that are either in memory or in registers. Thus, for example, an ADD may have all of its operands in registers or all of its operands in memory, or any combination thereof. The R+M architecture generalizes the formats of R/M. The classic example of the R+M architecture was Digital Equipment's VAX series of machines. VAX also generalized the use of the register set through the use of register modes. The use of an extended set of formats and register modes allows a powerful and varied specification of operands and operation type within a single instruction. Unfortunately, format and mode variability complicates the decoding process so that the process of interpretation of instructions can be slow (but R+M architectures make excellent use of memory/bus bandwidth).

From the architect's point of view, the tradeoff in instruction sets is an area-time compromise. The register-memory (R/M and R+M) architectures offer a more concise program representation using fewer instructions of variable size compared with L/S. Programs occupy less space in memory and smaller instruction caches can be used effectively. The variable instruction size makes decoding more difficult. The decoding of multiple instructions requires predicting the starting point of each. The register-memory processors require more circuitry (and area) to be devoted to instruction fetch and decode. Generally, the success of Intel-type x86 implementations in achieving high clock rates and performance has shown that the limitations of a register-memory instruction set can be overcome.

Memory

The *memory system* comprises the physical storage elements in the memory hierarchy. These elements include those specified by the instruction set (registers, main memory, and disk sectors) as well as those elements that are largely transparent to the user's program (buffer registers, cache, and page-mapped virtual memory). Registers have the fastest access and, although limited in capacity (32 to 128 bits), in program execution, is the most often referenced type of memory. A processor cycle time is usually defined by the time it takes to access one or more registers, operate their contents, and return the result to a register.

Main memory is the type of storage usually associated with the simple term memory. Most implementations are based on DRAM (e.g., DDR and DDR-2 SDRAM), although SRAM and Flash technologies have also been used. DRAM memory is accessible in order of 10s of cycles (typically 20 to 30) and usually processors have between 128 MB and 4 GB of such storage. The disk contains all the programs and data available to the processor. Its addressable unit (sector) is accessible in 1 to 10 ms, with a typical single-unit disk capacity of 10 to 300 GB. Large server systems may have 100s or more such disk units. As the levels of the memory system have such widely differing access times, additional levels of storage (buffer registers, cache, and paged memory) are added that serve as a buffer between levels attempting to hide the access time differences.

Memory Hierarchy. There are basically three parameters that define the effectiveness of the memory system: latency, bandwidth, and the capacity of each level of the system. Latency is the time for a particular access request to be completed. Bandwidth refers to the number of requests supplied per unit time. To provide large memory spaces with desirable access time latency and bandwidths, modern memory systems use a multiple-level memory hierarchy.

Smaller, faster levels have greater cost per bit than larger, slower levels. The multiple levels in the storage hierarchy can be ordered by their size and access time from the smallest, fastest level to the largest, slowest level. The goal of a good memory system design is to provide the processor with an effective memory capacity of the largest level with an access time close to the fastest. How well this goal is achieved depends on a number of factors—the characteristics of the device used in each level as well as the behavioral properties of the programs being executed.

Suppose we have a memory system hierarchy consisting of a cache, a main memory, and a disk or backing storage. The disk contains the contents of the entire virtual memory space. Typical size (S) and access time ratios (t) are as follows:

Size: memory/cache	≈	1000
Access time: memory/cache	≈	30
Size: disk/memory	≈	100–1000+
Access time: disk/memory	≈	100,000

Associated with both the cache and a paged main memory are corresponding tables that define the localities that are currently available at that level. The page table contains the working set of the disk—those disk localities that have been recently referenced by the program, and are contained in main memory. The cache table is completely managed by the hardware and contains those localities (called lines) of memory that have been recently referenced by the program.

The memory system operates by responding to a virtual effective address generated by a user program, which is translated into a real address in main memory. This real address accesses the cache table to find the entry in the cache for the desired value.

Paging and caching are the mechanisms to support the efficient management of the memory space. Paging is the mechanism by which the operating system brings fixed-size blocks (or pages)—a typical size is 4 to 64 KB—into main memory. Pages are fetched from backing store (usually disk) on demand (or as required) by the processor. When a referenced page is not present, the operating system is called and makes a request for the page, then transfers control to another process, allowing the processor resources to be used while waiting for the return of the requested information. The real address is used to access the cache and main memory. The low-order (least significant) bits address a particular location in a page. The upper bits of a virtual address access a page table (in memory) that:

1. determines whether this particular partial page lies in memory, and
2. translates the upper address bits if the page is present, producing the real address.

Usually, the tables performing address translation are in memory, and a mechanism for the translation called the translation lookaside buffer (TLB) must be used to speed up this translation. The TLB is a simple register system usually consisting of between 64 and 256 entries that save recent address translations for reuse.

Control and Execution

Instruction Execution Sequence. The semantics of the instruction determines that a sequence of actions must be performed to produce the specified result (Fig. 4). These actions can be overlapped (as discussed in the pipelined processor section) but the result must appear in the specified serial order. These actions include the following:

1. fetching the instruction into the instruction register (IF),
2. decoding the op code of the instruction (ID),

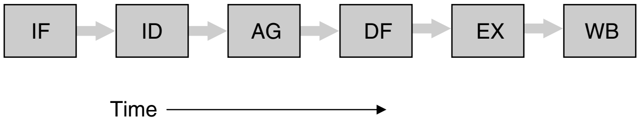


Figure 4. Instruction execution sequence.

3. generating the address in memory of any data item residing there (AG),
4. fetching data operand(s) into executable registers (DF),
5. executing the specified operation (EX), and
6. returning the result to the specified register (WB).

Decode: Hardwired and Microcode. The decoder produces the control signals that enable the functional units to execute the actions that produce the result specified by the instruction. Each cycle, the decoder produces a new set of control values that connect various registers and functional units. The decoder takes as an initial input the op code of the instruction. Using this op code, it generates the sequence of actions, one per cycle, which completes the execution process. The last step of the current instruction's execution is the fetching of the next instruction into the instruction register so that it may be decoded. The implementation of the decoder may be based on Boolean equations that directly implement the specified actions for each instruction. When these equations are implemented with logic gates, the resultant decoder is called a *hardwired decoder*.

For extended instruction sets or complex instructions, another implementation is sometimes used, which is based on the use of a fast storage (or microprogram store). A particular word in the storage (called a *microcode*) contains the control information for a single action or cycle. A sequence of microinstructions implements the instruction execution.

Data Paths: Busses and Functional Units. The data paths of the processor include all the functional units needed to implement the vocabulary (or op codes) of the instruction set. Typical functional units are the ALU (arithmetic logic unit) and the floating-point unit. Busses and other structured interconnections between the registers and the functional units complete the data paths.

PROGRAM PARALLELISM AND PARALLEL ARCHITECTURE

Exploiting program parallelism is one of the most important elements in computer architecture design. Programs written in imperative languages encompass the following four levels of parallelism:

1. parallelism at the instruction level (fine-grained),
2. parallelism at the loop level (middle-grained),

3. parallelism at the procedure level (middle-grained), and
4. parallelism at the program level (coarse-grained).

Instruction-level parallelism (ILP) means that multiple operations can be executed in parallel within a program. ILP may be achieved with hardware, compiler, or operating system techniques. At the loop level, consecutive loop iterations are ideal candidates for parallel execution provided that there is no data dependency between subsequent loop iterations. Next, there is parallelism available at the procedure level, which depends largely on the algorithms used in the program. Finally, multiple independent programs can obviously execute in parallel.

Different computer architectures have been built to exploit this inherent parallelism. In general, a computer architecture consists of one or more interconnected processor elements that operate concurrently, solving a single overall problem. The various architectures can be conveniently described using the stream concept. A stream is simply a sequence of objects or actions. There are both instruction streams and data streams, and there are four simple combinations that describe the most familiar parallel architectures (6):

1. SISD – single instruction, single data stream; The traditional uniprocessor (Fig. 5).
2. SIMD – single instruction, multiple data stream, which includes array processors and vector processors (Fig. 6).
3. MISD – multiple instruction, single data stream, which are typically systolic arrays (Fig. 7).
4. MIMD – multiple instruction, multiple data stream, which includes traditional multiprocessors as well as the newer work of networks of workstations (Fig. 8).

The stream description of computer architectures serves as a programmer's view of the machine. If the processor architecture allows for parallel processing of one sort or another, then this information is also visible to the programmer. As a result, there are limitations to the stream categorization. Although it serves as useful shorthand, it ignores many subtleties of an architecture or an implementation. Even an SISD processor can be highly parallel in its execution of operations. This parallelism is typically not visible to the programmer even at the assembly language level, but becomes visible at execution time with improved performance.

There are many factors that determine the overall effectiveness of a parallel processor organization. Interconnection network, for instance, can affect the overall speedup. The characterizations of both processors and networks are complementary to the stream model and, when coupled with the stream model, enhance the qualitative understanding of a given processor configuration.

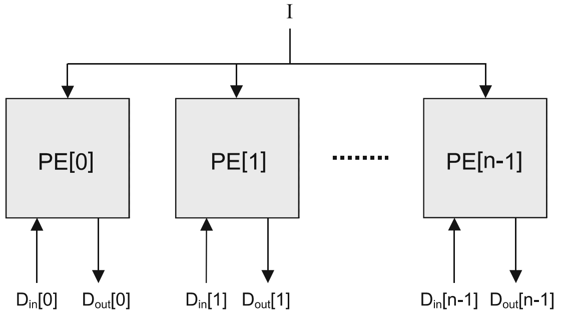

I
PE[0]
PE[1]
....
PE[n-1]
D_{in}[0] D_{out}[0]
D_{in}[1] D_{out}[1]
D_{in}[n-1] D_{out}[n-1]

Figure 6. SIMD – single instruction, multiple data stream.

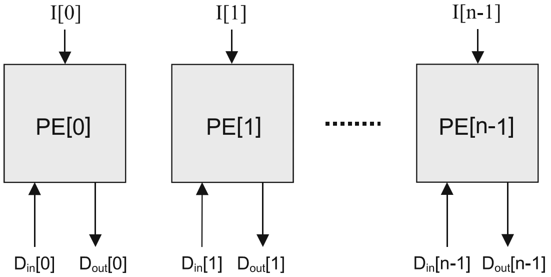


Figure 7. MISD – multiple instruction, single data stream.

SISD – Single Instruction, Single Data Stream

The SISD class of processor architecture includes most commonly available computers. These processors are known as uniprocessors and can be found in millions of embedded processors in video games and home appliances as well as stand-alone processors in home computers, engineering workstations, and mainframe computers. Although

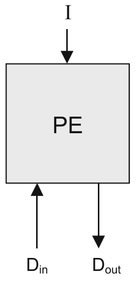


Figure 5. SISD – single instruction, single data stream.

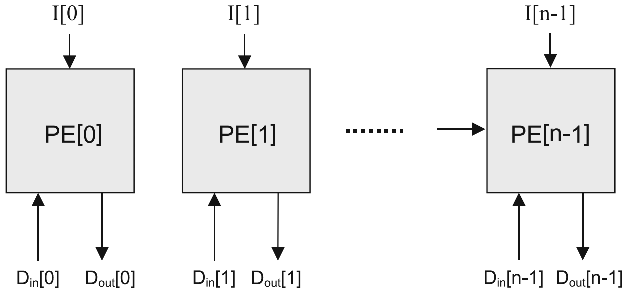


Figure 8. MIMD – multiple instruction, multiple data stream.

Table 1. Typical Scalar Processors (SISD)

Processor	Year of introduction	Number of function unit	Issue width	Scheduling	Number of transistors
Intel 8086	1978	1	1	Dynamic	29K
Intel 80286	1982	1	1	Dynamic	134K
Intel 80486	1989	2	1	Dynamic	1.2M
HP PA-RISC 7000	1991	1	1	Dynamic	580K
Sun SPARC	1992	1	1	Dynamic	1.8M
MIPS R4000	1992	2	1	Dynamic	1.1M
ARM 610	1993	1	1	Dynamic	360K
ARM SA-1100	1997	1	1	Dynamic	2.5M

a programmer may not realize the inherent parallelism within these processors, a good deal of concurrency can be present. Pipelining is a powerful technique that is used in almost all current processor implementations. Other techniques aggressively exploit parallelism in executing code whether it is declared statically or determined dynamically from an analysis of the code stream.

During execution, a SISD processor executes one or more operations per clock cycle from the instruction stream. An instruction is a container that represents the smallest execution packet managed explicitly by the processor. One or more operations are contained within an instruction. The distinction between instructions and operations is crucial to distinguish between processor behaviors. Scalar and superscalar processors consume one or more instructions per cycle where each instruction contains a single operation. VLIW processors, on the other hand, consume a single instruction per cycle where this instruction contains multiple operations.

A SISD processor has four primary characteristics. The first characteristic is whether the processor is capable of executing multiple operations concurrently. The second characteristic is the mechanisms by which operations are scheduled for execution—statically at compile time, dynamically at execution, or possibly both. The third characteristic is the order that operations are issued and retired relative to the original program order—these operations can be in order or out of order. The fourth characteristic is the manner in which exceptions are handled by the processor—precise, imprecise, or a combination. This last condition is not of immediate concern to the applications programmer, although it is certainly important to the compiler writer or operating system programmer who must be able to properly handle exception conditions. Most processors implement precise exceptions, although a few high-performance architectures allow imprecise floating-point exceptions.

Tables 1, 2, and 3 describe some representative scalar, processors, superscalar processors, and VLIW processors.

Scalar Processor. Scalar processors process a maximum of one instruction per cycle and execute a maximum of one operation per cycle. The simplest scalar processors, sequential processors, process instructions atomically one after another. This sequential execution behavior describes the sequential execution model that requires each instruction executed to completion in sequence. In the sequential execution model, execution is *instruction-precise* if the following conditions are met:

1. All instructions (or operations) preceding the current instruction (or operation) have been executed and all results have been committed.
2. All instructions (or operations) after the current instruction (or operation) are unexecuted and no results have been committed.
3. The current instruction (or operation) is in an arbitrary state of execution and may or may not have completed or had its results committed.

For scalar and superscalar processors with only a single operation per instruction, instruction-precise and operation-precise executions are equivalent. The traditional definition of sequential execution requires instruction-precise execution behavior at all times, mimicking the execution of a nonpipelined sequential processor.

Sequential Processor. Sequential processors directly implement the sequential execution model. These processors process instructions sequentially from the instruction stream. The next instruction is not processed until all execution for the current instruction is complete and its results have been committed.

Although conceptually simple, executing each instruction sequentially has significant performance drawbacks—a considerable amount of time is spent in overhead and not in actual execution. Thus, the simplicity of directly implementing the sequential execution model has significant performance costs.

Pipelined Processor. Pipelining is a straightforward approach to exploiting parallelism that is based on concurrently performing different phases (instruction fetch, decode, execution, etc.) of processing an instruction. Pipelining assumes that these phases are independent between different operations and can be overlapped; when this condition does not hold, the processor stalls the downstream phases to enforce the dependency. Thus, multiple operations can be processed simultaneously with each operation at a different phase of its processing. Figure 9 illustrates the instruction timing in a pipelined processors, assuming that the instructions are independent. The

Table 2. Typical Superscalar Processors (SISD)

Processor	Year of introduction	Number of function unit	Issue width	Scheduling	Number of transistors
HP PA-RISC 7100	1992	2	2	Dynamic	850K
Motorola PowerPC 601	1993	4	3	Dynamic	2.8M
MIPS R8000	1994	6	4	Dynamic	3.4M
DEC Alpha 21164	1994	4	4	Dynamic	9.3M
Motorola PowerPC 620	1995	4	2	Dynamic	7M
MIPS R10000	1995	5	4	Dynamic	6.8M
HP PA-RISC 7200	1995	3	2	Dynamic	1.3M
Intel Pentium Pro	1995	5	3/6†	Dynamic	5.5M
DEC Alpha 21064	1992	4	2	Dynamic	1.7M
Sun Ultra I	1995	9	4	Dynamic	5.2M
Sun Ultra II	1996	9	4	Dynamic	5.4M
AMD K5	1996	6	4/46†	Dynamic	4.3M
Intel Pentium II	1997	5	3/66†	Dynamic	7.5M
AMD K6	1997	7	2/66†	Dynamic	8.8M
Motorola PowerPC 740	1997	6	3	Dynamic	6.4M
DEC Alpha 21264	1998	6	4	Dynamic	15.2M
HP PA-RISC 8500	1998	10	4	Dynamic	140M
Motorola PowerPC 7400	1999	10	3	Dynamic	6.5M
AMD K7	1999	9	3/66†	Dynamic	22M
Intel Pentium III	1999	5	3/66†	Dynamic	28M
Sun Ultra III	2000	6	4	Dynamic	29M
DEC Alpha 21364	2000	6	4	Dynamic	100M
AMD Athlon 64 FX51	2003	9	3/66†	Dynamic	105M
Intel Pentium 4 Prescott	2003	5	3/66†	Dynamic	125M

[1] For some Intel ×86 family processors, each instruction is broken into a number of microoperation codes in the decoding stage. In this article, two different issue widths are given for these processors: The first one is the maximum number of instructions issued per cycle, and the second one is the maximum number of microoperation codes issued per cycle.

meaning of each pipeline stage is described in the Instruction Execution Sequence system.

For a simple pipelined machine, only one operation occurs in each phase at any given time. Thus, one operation is being fetched, one operation is being decoded, one operation is accessing operands, one operation is in execution, and one operation is storing results. The most rigid form of a pipeline, sometimes called the static pipeline, requires the processor to go through all stages or phases of the pipeline whether required by a particular instruction or not. Dynamic pipeline allows the bypassing of one or more of the stages of the pipeline depending on the requirements of the instruction. There are at least three levels of sophistication within the category of dynamic pipeline processors as follows:

- *Type 1:* Dynamic pipelines that require instructions to be decoded in sequence and results to be executed and written back in sequence. For these types of simpler dynamic pipeline processors, the advantage over a static pipeline is relatively modest. In-order execution requires the actual change of state to occur in order specified in the instruction sequence.
- *Type 1-Extended:* A popular extension of the Type 1 pipeline is to require the decode to be in order, but the execution stage of ALU operations need not be in order. In these organizations, the address generation stage of the load and store instructions must be completed before any subsequent ALU instruction does a writeback. The reason is that the address generation may cause a page execution and affect the

Table 3. Typical VLIW Processors (SISD)

Processor	Year of introduction	Number of function unit	Issue width	Scheduling	Issue/complete order
Multiflow Trace 7/200	1987	7	7	Static	In-order/in-order
Multiflow Trace 14/200	1987	14	14	Static	In-order/in-order
Multiflow Trace 28/200	1987	28	28	Static	In-order/in-order
Cydrome Cydra 5	1987	7	7	Static	In-order/in-order
Philips TM-1	1996	27	5	Static	In-order/in-order
TI TMS320/C62x	1997	8	8	Static	In-order/in-order
Intel Itanium	2001	9	6	Static	In-order/in-order
Intel Itanium 2	2003	11	6	Static	In-order/in-order

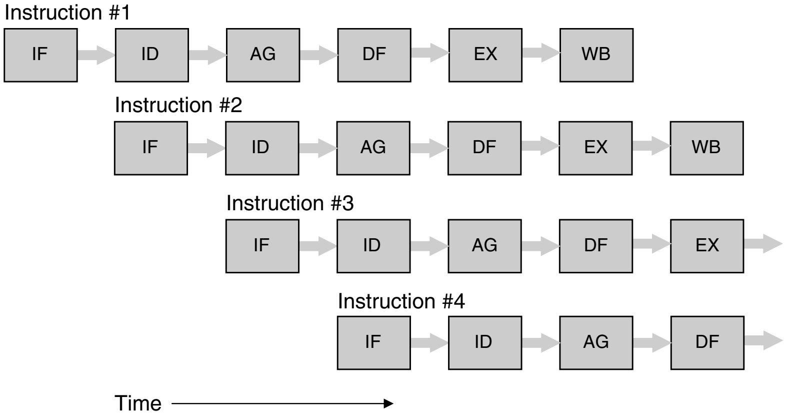


Figure 9. Instruction timing in a pipelined processor.

processor state. As a result of these restrictions and the overall frequency of load and store instructions, the Type 1-Extended pipeline behaves much as the basic Type-1 pipeline.

- *Type 2:* Dynamic pipelined machines that can be configured to allow out-of order execution yet retain in-order instruction decode. For this type of pipelined processor, the execution and writeback of all instructions is a function only of dependencies on prior instruction. If a particular instruction is independent of all preceding instructions, its execution can be completed independently of the successful completion of prior instructions.
- *Type 3:* The third type of dynamic pipeline allows instructions to be issued as well as completed out of order. A group of instruction is analyzed together, and the first instruction that is found to be independent of prior instructions is decoded.

Instruction-level Parallelism. Although pipelining does not necessarily lead to executing multiple instructions at exactly the same time, there are other techniques that do. These techniques may use some combination of static scheduling and dynamic analysis to perform concurrently the actual evaluation phase of several different operations, potentially yielding an execution rate of greater than one operation every cycle. This kind of parallelism exploits concurrency at the computation level. As historically most instructions consist of only a single operation, this kind of parallelism has been named instruction-level parallelism (ILP).

Two architectures that exploit ILP are superscalar and VLIW, which use radically different techniques to achieve greater than one operation per cycle. A superscalar processor dynamically examines the instruction stream to determine which operations are independent and can be executed. A VLIW processor depends on the compiler to analyze the available operations (OP) and to schedule independent operations into wide instruction words, which then executes these operations in parallel with no further analysis.

Figure 10 shows the instruction timing of a pipelined superscalar or VLIW processor executing two instructions per cycle. In this case, all the instructions are independent so that they can be executed in parallel.

Superscalar Processor. Dynamic out-of-order pipelined processors reach the limits of performance for a scalar processor by allowing out-of-order operation execution. Unfortunately, these processors remain limited to executing a single operation per cycle by virtue of their scalar nature. This limitation can be avoided with the addition of multiple functional units and a dynamic scheduler to process more than one instruction per cycle. These resulting superscalar processors can achieve execution rates of more than one instruction per cycle. The most significant advantage of a superscalar processor is that processing multiple instructions per cycle is done transparently to the user, and that it can provide binary compatibility while achieving better performance.

Compared with an out-of-order pipelined processor, a superscalar processor adds a scheduling instruction window that dynamically analyzes multiple instructions from the instruction stream. Although processed in parallel, these instructions are treated in the same manner as in an out-of-order pipelined processor. Before an instruction is issued for execution, dependencies between the instruction and its prior instructions must be checked by hardware.

As a result of the complexity of the dynamic scheduling logic, high-performance superscalar processors are limited to processing four to six instructions per cycle (refer to the Examples of Recent Architecture section). Although superscalar processors can take advantage of dynamic execution behavior and exploit instruction-level parallelism from the dynamic instruction stream, exploiting high degrees of instruction requires a different approach. An alternative approach is to rely on the compiler to perform the dependency analyses and to eliminate the need for complex analyses performed in hardware.

VLIW Processor. In contrast to dynamic analyses in hardware to determine which operations can be executed in parallel, VLIW processors rely on static analyses in the compiler. VLIW processors are, thus, less complex than superscalar processor and have the potential for higher performance. A VLIW processor executes operations from statically scheduled instructions that contain multiple

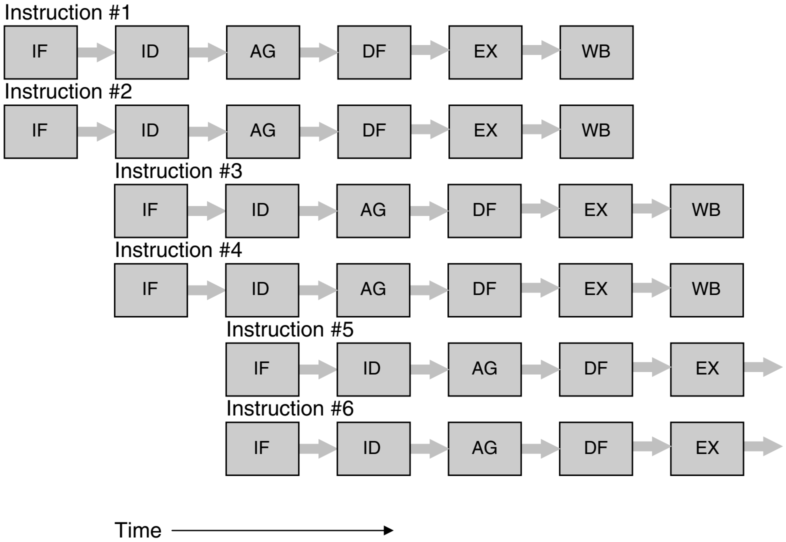


Figure 10. Instruction timing of a pipelined ILP processor.

independent operations. Although it is not required that static processors exploit instruction-level parallelism, most statically scheduled processors use wide instruction words. As the complexity of a VLIW processor is not significantly greater than that of a scalar processor, the improved performance comes without the complexity penalties.

On the other hand, VLIW processors rely on the static analyses performed by the compiler and are unable to take advantage of any dynamic execution characteristics. As issue widths become wider and wider, the avoidance of complex hardware logic will rapidly erode the benefits of out-of-order execution. This benefit becomes more significant as memory latencies increase and the benefits from out-of-order execution become a less significant portion of the total execution time. For applications that can be statically scheduled to use the processor resources effectively, a simple VLIW implementation results in high performance.

Unfortunately, not all applications can be effectively scheduled for VLIW processors. In real systems, execution rarely proceeds exactly along the path defined by the code scheduler in the compiler. These are two classes of execution variations that can develop and affect the scheduled execution behavior:

1. Delayed results from operations whose latency differs from the assumed latency scheduled by the compiler.
2. Interruptions from exceptions or interrupts, which change the execution path to a completely different and unanticipated code schedule.

Although stalling the processor can control delayed results, this solution can result in significant performance penalties from having to distribute the stall signal across the processor. Delays occur from many causes including mismatches between the architecture and an implementation as well as from special-case conditions that require additional cycles to complete an operation. The most common execution delay is a data cache miss; another example is a floating-point operation that requires an additional normalization cycle. For processors without hardware resource management, delayed results can cause resource conflicts and incorrect execution behavior. VLIW processors typically avoid all situations that can result in a delay by not using data caches and by assuming worst-case latencies for operations. However, when there is insufficient parallelism to hide the exposed worst-case operation latency, the instruction schedule will have many incompletely filled or empty instructions that can result in poor performance.

Interruptions are usually more difficult to control than delayed results. Managing interruptions is a significant problem because of their disruptive behavior and because the origins of interruptions are often completely beyond a program's control. Interruptions develop from execution-related internal sources (exceptions) as well as arbitrary external sources (interrupts). The most common interruption is an operation exception resulting from either an error condition during execution or a special-case condition that requires additional corrective action to complete operation execution. Whatever the source, all interruptions require the execution of an appropriate service routine to resolve the problem and to restore normal execution at the point of the interruption.

SIMD – Single Instruction, Multiple Data Stream

The SIMD class of processor architecture includes both array and vector processors. The SIMD processor is a natural response to the use of certain regular data structures, such as vectors and matrices. From the reference point of an assembly-level programmer, programming SIMD architecture appears to be very similar to programming a simple SISD processor except that some operations perform computations on aggregate data. As these

Table 4. Typical Vector Computers (SIMD)

Processor	Year of introduction	Memory- or register-based	Number of processor units	Maximum vector length
Cray 1	1976	Register	1	64
CDC Cyber 205	1981	Memory	1	65535
Cray X-MP	1982	Register	1–4	64
Cray 2	1985	Register	5	64
Fujitsu VP-100/200	1985	Register	3	32–1024
ETA ETA	1987	Memory	2–8	65535
Cray Y-MP/832	1989	Register	1–8	64
Cray Y-MP/C90	1991	Register	16	64
Convex C3	1991	Register	1–8	128
Cray T90	1995	Register	1–32	128
NEC SX-5	1998	Register	1–512	256

regular structures are widely used in scientific programming, the SIMD processor has been very successful in these environments.

The two popular types of SIMD processor are the array processor and the vector processor. They differ both in their implementations and in their data organizations. An array processor consists of many interconnected processor elements that each have their own local memory space. A vector processor consists of a single processor that references a single global memory space and has special function units that operate specifically on vectors. Tables 4 and 5 describe some representative vector processors and array processors.

Array Processors. The array processor is a set of parallel processor elements connected via one or more networks, possibly including local and global interelement communications and control communications. Processor elements operate in lockstep in response to a single broadcast instruction from a control processor. Each processor element has its own private memory and data is distributed across the elements in a regular fashion that is dependent on both the actual structure of the data and also on the computations to be performed on the data. Direct access to global memory or another processor element's local memory is expensive, so intermediate values are propagated through the array through local interprocessor connections, which requires that the data be distributed carefully so that the routing required to propagate these values is simple and regular. It is sometimes easier to duplicate data values and computations than it is to affect a complex or irregular routing of data between processor elements.

As instructions are broadcast, there is no means local to a processor element of altering the flow of the instruction stream; however, individual processor elements can conditionally disable instructions based on local status information—these processor elements are idle when this condition occurs. The actual instruction stream consists of more than a fixed stream of operations; an array processor is typically coupled to a general-purpose control processor that provides both scalar operations as well as array operations that are broadcast to all processor elements in the array. The control processor performs the scalar sections of the application, interfaces with the outside world, and controls the flow of execution; the array processor performs the array sections of the application as directed by the control processor.

A suitable application for use on an array processor has several key characteristics: a significant amount of data that has a regular structure; computations on the data that are uniformly applied to many or all elements of the dataset; and simple and regular patterns relating the computations and the data. An example of an application that has these characteristics is the solution of the Naviér– Stokes equations, although any application that has significant matrix computations is likely to benefit from the concurrent capabilities of an array processor.

The programmer's reference point for an array processor is typically the high-level language level; the programmer is concerned with describing the relationships between the data and the computations but is not directly concerned with the details of scalar and array instruction scheduling or the details of the interprocessor distribution of data within the processor. In fact, in many cases, the programmer is not even concerned with size of the array processor. In general, the programmer specifies the size and any

Table 5. Typical Array Processors (SIMD)

Processor	Year of introduction	Memory model	Processor element	Number of processors
Burroughs BSP	1979	Shared	General purpose	16
Thinking Machine CM-1	1985	Distributed	Bit-serial	Up to 65,536
Thinking Machine CM-2	1987	Distributed	Bit-serial	4,096–65,536
MasPar MP-1	1990	Distributed	Bit-serial	1,024–16,384

specific distribution information for the data and the compiler maps the implied virtual processor array onto the physical processor elements that are available and generates code to perform the required computations. Thus, although the size of the processor is an important factor in determining the performance that the array processor can achieve, it is not a fundamental characteristic of an array processor.

The primary characteristic of a SIMD processor is whether the memory model is shared or distributed. In this section, only processors using a distributed memory model are described as this configuration is used by SIMD processors today and the cost of scaling a shared-memory SIMD processor to a large number of processor elements would be prohibitive. Processor elements and network characteristics are also important in characterizing a SIMD processor.

Vector Processors. A vector processor is a single processor that resembles a traditional SISD processor except that some of the function units (and registers) operate on vectors—sequences of data values that are seemingly operated on as a single entity. These function units are deeply pipelined and have a high clock rate; although the vector pipelines have as long or longer latency than a normal scalar function unit, their high clock rate and the rapid delivery of the input vector data elements results in a significant throughput that cannot be matched by scalar function units.

Early vector processors processed vectors directly from memory. The primary advantage of this approach was that the vectors could be of arbitrary lengths and were not limited by processor resources; however, the high start-up cost, limited memory system bandwidth, and memory system contention proved to be significant limitations. Modern vector processors require that vectors be explicitly loaded into special vector registers and stored back into memory, the same course that modern scalar processors have taken for similar reasons. However, as vector registers can rapidly produce values for or collect results from the vector function units and have low startup costs, modern register-based vector processors achieve significantly higher performance than the earlier memory-based vector processors for the same implementation technology.

Modern processors have several features that enable them to achieve high performance. One feature is the ability to concurrently load and store values between the vector register file and main memory while performing computations on values in the vector register file. This feature is important because the limited length of vector registers requires that vectors that are longer be processed in segments—a technique called strip-mining. Not being able to overlap memory accesses and computations would pose a significant performance bottleneck.

Just like SISD processors, vector processors support a form of result bypassing—in this case called chaining—that allows a follow-on computation to commence as soon as the first value is available from the preceding computation. Thus, instead of waiting for the entire vector to be processed, the follow-on computation can be significantly overlapped with the preceding computation that it is dependent on. Sequential computations can be efficiently compounded and behave as if they were a single operation with a total latency equal to the latency of the first operation with the pipeline and chaining latencies of the remaining operations but none of the startup overhead that would be incurred without chaining. For example, division could be synthesized by chaining a reciprocal with a multiply operation. Chaining typically works for the results of load operations as well as normal computations. Most vector processors implement some form of chaining.

A typical vector processor configuration consists of a vector register file, one vector addition unit, one vector multiplication unit, and one vector reciprocal unit (used in conjunction with the vector multiplication unit to perform division); the vector register file contains multiple vector registers. In addition to the vector registers, there are also a number of auxiliary and control registers, the most important of which is the vector length register. The vector length register contains the length of the vector (or the loaded subvector if the full vector length is longer than the vector register itself) and is used to control the number of elements processed by vector operations. There is no reason to perform computations on non-data that is useless or could cause an exception.

As with the array processor, the programmer's reference point for a vector machine is the high-level language. In most cases, the programmer sees a traditional SISD machine; however, as vector machines excel on vectorizable loops, the programmer can often improve the performance of the application by carefully coding the application, in some cases explicitly writing the code to perform strip-mining, and by providing hints to the compiler that help to locate the vectorizable sections of the code. This situation is purely an artifact of the fact that the programming languages are scalar oriented and do not support the treatment of vectors as an aggregate data type but only as a collection of individual values. As languages are defined (such as Fortran 90 or High-Performance Fortran) that make vectors a fundamental data-type, then the programmer is exposed less to the details of the machine and to its SIMD nature.

The vector processor has one primary characteristic. This characteristic is the location of the vectors; vectors can be memory-or register-based. There are many features that vector processors have that are not included here because of their number and many variations. These features include variations on chaining, masked vector operations based on a boolean mask vector, indirectly addressed vector operations (scatter/gather), compressed/expanded vector operations, reconfigurable register files, multiprocessor support, and soon. Vector processors have developed dramatically from simple memory-based vector processors to modern multiple-processor vector processors that exploit both SIMD vector and MIMD style processing.

MISD – Multiple Instruction, Single Data Stream

Although it is easy to both envision and design MISD processors, there has been little interest in this type of parallel architecture. The reason, so far anyway, is that there are no ready programming constructs that easily map programs into the MISD organization.

Conceptually, MISD architecture can be represented as multiple independently executing function units operating on a single stream of data, forwarding results from one function unit to the next, which, on the microarchitecture level, is exactly what the vector processor does. However, in the vector pipeline, the operations are simply fragments of an assembly-level operation, as distinct from being a complete operation. Surprisingly, some of the earliest attempts at computers in the 1940s could be seen as the MISD concept. They used plug boards for programs, where data in the form of a punched card was introduced into the first stage of a multistage processor. A sequential series of actions was taken where the intermediate results were forwarded from stage to stage until, at the final stage, a result would be punched into a new card.

There are, however, more interesting uses of the MISD organization. Nakamura has pointed out the value of an MISD machine called the SHIFT machine. In the SHIFT machine, all data memory is decomposed into shift registers. Various function units are associated with each shift column. Data is initially introduced into the first column and is shifted across the shift register memory. In the SHIFT machine concept, data is regularly shifted from memory region to memory region (column to column) for processing by various function units. The purpose behind the SHIFT machine is to reduce memory latency. In a traditional organization, any function unit can access any region of memory and the worst-case delay path for accessing memory must be taken into account. In the SHIFT machine, we must only allow for access time to the worst element in a data column. The memory latency in modern machines is becoming a major problem – the SHIFT machine has a natural appeal for its ability to tolerate this latency.

MIMD – Multiple Instruction, Multiple Data Stream

The MIMD class of parallel architecture brings together multiple processors with some form of interconnection. In this configuration, each processor executes completely independently, although most applications require some form of synchronization during execution to pass information and data between processors. Although no requirement exists that all processor elements be identical, most MIMD configurations are homogeneous with all processor elements identical. There have been heterogeneous MIMD configurations that use different kinds of processor elements to perform different kinds of tasks, but these configurations have not yielded to general-purpose applications. We limit ourselves to homogeneous MIMD organizations in the remainder of this section.

MIMD Programming and Implementation Considerations. Up to this point, the MIMD processor with its multiple processor elements interconnected by a network appears to be very similar to a SIMD array processor. This similarity is deceptive because there is a significant difference between these two configurations of processor elements—in the array processor, the instruction stream delivered to each processor element is the same, whereas in the MIMD processor, the instruction stream delivered to each processor element is independent and specific to each processor element. Recall that in the array processor, the control processor generates the instruction stream for each processor element and that the processor elements operate in lock step. In the MIMD processor, the instruction stream for each processor element is generated independently by that processor element as it executes its program. Although it is often the case that each processor element is running pieces the same program, there is no reason that different processor elements should not run different programs.

The interconnection network in both the array processor and the MIMD processor passes data between processor elements; however, in the MIMD processor, it is also used to synchronize the independent execution streams between processor elements. When the memory of the processor is distributed across all processors and only the local processor element has access to it, all data sharing is performed explicitly using messages and all synchronization is handled within the message system. When the memory of the processor is shared across all processor elements, synchronization is more of a problem—certainly messages can be used through the memory system to pass data and information between processor elements, but it is not necessarily the most effective use of the system.

When communications between processor elements is performed through a shared-memory address space, either global or distributed between processor elements (called distributed shared memory to distinguish it from distributed memory), there are two significant problems that develop. The first is maintaining memory consistency; the programmer-visible ordering effects of memory references both within a processor element and between different processor elements. The second is cache coherency; the programmer-invisible mechanism ensures that all processor elements see the same value for a given memory location. Neither of these problems is significant in SISD or SIMD array processors. In a SISD processor, there is only one instruction stream and the amount of reordering is limited so the hardware can easily guarantee the effects of perfect memory reference ordering and thus there is no consistency problem; because a SISD processor has only one processor element, cache coherency is not applicable. In a SIMD array processor (assuming distributed memory), there is still only one instruction stream and typically no instruction reordering; because all interprocessor element communications is via message, there is neither a consistency problem nor a coherency problem.

The memory consistency problem is usually solved through a combination of hardware and software techniques. At the processor element level, the appearance of perfect memory consistency is usually guaranteed for local memory references only, which is usually a feature of the processor element itself. At the MIMD processor level, memory consistency is often only guaranteed through explicit synchronization between processors. In this case, all nonlocal references are only ordered relative to these synchronization points. Although the programmer must be aware of the limitations imposed by the ordering scheme, the added performance achieved using nonsequential ordering can be significant.

The cache coherency problem is usually solved exclusively through hardware techniques. This problem is significant because of the possibility that multiple processor elements will have copies of data in their local caches, each copy of which can have different values. There are two primary techniques to maintain cache coherency. The first is to ensure that all processor elements are informed of any change to the shared-memory state—these changes are broadcast throughout the MIMD processor and each processor element monitors these changes (commonly referred to as snooping). The second is to keep track of all users of a memory address or block in a directory structure and to specifically inform each user when there is a change made to the shared-memory state. In either case, the result of a change can be one of two things, either the new value is provided and the local value is updated or all other copies of the value are invalidated.

As the number of processor elements in a system increases, a directory-based system becomes significantly better as the amount of communications required to maintain coherency is limited to only those processors holding copies of the data. Snooping is frequently used within a small cluster of processor elements to track local changes – here the local interconnection can support the extra traffic used to maintain coherency because each cluster has only a few processor elements in it.

The primary characteristic of a MIMD processor is the nature of the memory address space; it is either separate or shared for all processor elements. The interconnection network is also important in characterizing a MIMD processor and is described in the next section. With a separate address space (distributed memory), the only means of communications between processor elements is through messages and thus these processors force the programmer to use a message-passing paradigm. With a shared address space (shared memory), communications between processor elements is through the memory system—depending on the application needs or programmer preference, either a shared memory or message passing paradigm can be used.

The implementation of a distributed-memory machine is far easier than the implementation of a shared-memory machine when memory consistency and cache coherency is taken into account. However, programming a distributed memory processor can be much more difficult because the applications must be written to exploit and not be limited by the use of message passing as the only form of communications between processor elements. On the other hand, despite the problems associated with maintaining consistency and coherency, programming a shared-memory processor can take advantage of whatever communications paradigm is appropriate for a given communications requirement and can be much easier to program. Both distributed-and shared-memory processors can be extremely scalable and neither approach is significantly more difficult to scale than the other.

MIMD Rationale. MIMD processors usually are designed for at least one of two reasons: fault tolerance or program speedup. Ideally, if we have n identical processors, the failure of one processor should not affect the ability of the multiprocessor to continue program execution. However, this case is not always true. If the operating system is designated to run on a particular processor and that processor fails, the system fails. On the other hand, some multiprocessor ensembles have been built with the sole purpose of high-integrity, fault-tolerant computation. Generally, these systems may not provide any program speedup over a single processor. Systems that duplicate computations or that triplicate and vote on results are examples of designing for fault tolerance.

MIMD Speedup: Partitioning and Scheduling. As multiprocessors simply consist of multiple computing elements, each computing element is subject to the same basic design issues. These elements are slowed down by branch delays, cache misses, and so on. The multiprocessor configuration, however, introduces speedup potential as well as additional sources of delay and performance degradation. The sources of performance bottlenecks in multiprocessors generally relate to the way the program was decomposed to allow concurrent execution on multiple processors. The speedup (*Sp*) of an MIMD processor ensemble is defined as:

$$Sp = T(1)/T(n)$$

or the execution time of a single processor (*T*(*1*)) divided by the execution time for *n* processors executing the same application *(T(n))*. The achievable MIMD speedup depends on the amount of parallelism available in the program (partitioning) and how well the partitioned tasks are scheduled.

Partitioning is the process of dividing a program into tasks, each of which can be assigned to an individual processor for execution at run time. These tasks can be represented as nodes in a control graph. The arcs in the graph specify the order in which execution of the subtasks must occur. The partitioning process occurs at compile time, well before program execution. The goal of the partitioning process is to uncover the maximum amount of parallelism possible without going beyond certain obvious machine limitations.

The program partitioning is usually performed with some a priori notion of program overhead. Program overhead (*o*) is the added time a task takes to be loaded into a processor before beginning execution. The larger the size of the minimum task defined by the partitioning program, the smaller the effect of program overhead. Table 6 gives an instruction count for some various program grain sizes.

The essential difference between multiprocessor concurrency and instruction-level parallelism is the amount of overhead expected to be associated with each task. Overhead affects speedup. If uniprocessor program P_1 does operation W_1, then the parallel version of P_1 does operations W_p, where $W_p \geq W_1$.

For each task T_i, there is an associated number of overhead operations o_{i}, so that if T_i takes W_i operations without

Table 6. Grain Size

Grain Description	Program Construct	Typical number of instructions
Fine grain	Basic block “Instruction-level parallelism”	5 to 10
Medium grain	Loop/Procedure “Loop-level parallelism” “Procedure-level parallelism”	100 to 100,000
Coarse grain	Large task “Program-level parallelism”	100,000 or more

overhead, then

$$W_P = \Sigma(W_i + o_i) \geq W_1$$

where W_p is the total work done by P_p, including overhead. To achieve speedup over a uniprocessor, a multiprocessor system must achieve the maximum degree of parallelism among executing subtasks or control nodes. On the other hand, if we increase the amount of parallelism by using finer- and finer-grain task sizes, we necessarily increase the amount of overhead. Moreover, the overhead depends on the following factors.

- Overhead time is configuration-dependent. Different shared-memory multiprocessors may have significantly different task overheads associated with them, depending on cache size, organization, and the way caches are shared.
- Overhead may be significantly different depending on how tasks are actually assigned (scheduled) at run time. A task returning to a processor whose cache already contains significant pieces of the task code or dataset will have a significantly lower overhead than the same task assigned to an unrelated processor.

Increased parallelism usually corresponds to finer task granularity and larger overhead. Clustering is the grouping together of subtasks into a single assignable task. Clustering is usually performed both at partitioning time and during scheduling run time. The reasons for clustering during partition time might include when

- The available parallelism exceeds the known number of processors that the program is being compiled for.
- The placement of several shorter tasks that share the same instruction or data working set into a single task provides lower overhead.

Scheduling can be performed statically at compile time or dynamically at run time. Static scheduling information can be derived on the basis of the probable critical paths, which alone is insufficient to ensure optimum speedup or even fault tolerance. Suppose, for example, one of the processors scheduled statically was unavailable at run time, having suffered a failure. If only static scheduling had been done, the program would be unable to execute if assignment to all n processors had been made. It is also oftentimes the case that program initiation does not begin with n designated idle processors. Rather, it begins with a smaller number as previously executing tasks complete their work. Thus, the processor availability is difficult to predict and may vary from run to run. Although run-time scheduling has obvious advantages, handling changing systems environments, as well as highly variable program structures, it also has some disadvantages, primarily its run-time overhead.

Run-time scheduling can be performed in a number of different ways. The scheduler itself may run on a particular processor or it may run on any processor. It can be centralized or distributed. It is desirable that the scheduling not be designated to a particular processor, but rather any processor, and then the scheduling process itself can be distributed across all available processors.

Types of MIMD processors. Although all MIMD architectures share the same general programming model, there are many differences in programming detail, hardware configuration, and speedup potential. Most differences develop from the variety of shared hardware, especially the way the processors share memory. For example, processors may share at one of several levels:

- Shared internal logic (floating point, decoders, etc.), shared data cache, and shared memory.
- Shared data cache—shared memory.
- Separate data cache but shared bus—shared memory.
- Separate data cache with separate busses leading to a shared memory.
- Separate processors and separate memory modules interconnected with a multistage interconnection network.
- Separate processor-memory systems cooperatively executing applications via a network.

The basic tradeoff in selecting a type of multiprocessor architecture is between resource limitations and synchronization delay. Simple architectures are generally resource-limited and have rather low synchronization communications delay overhead. More robust processor-memory configurations may offer adequate resources for extensive communications among various processors in memory, but these configurations are limited by

- delay through the communications network and
- multiple accesses of a single synchronization variable.

The simpler and more limited the multiprocessor configuration, the easier it is to provide synchronization communications and memory coherency. Each of these functions requires an access to memory. As long as memory bandwidth is adequate, these functions can be readily handled. As processor speed and the number of processors increase, eventually shared data caches and busses run out of bandwidth and become the bottleneck in the multiprocessor system. Replicating caches or busses to provide additional bandwidth requires management of not only the original traffic, but the coherency traffic also. From the system's point of view, one would expect to find an optimum level of sharing for each of the shared resources—data cache, bus, memory, and so on—fostering a hierarchical view of shared-memory multiprocessing systems.

Multithreaded or shared resource multiprocessing. The simplest and most primitive type of multiprocessor system is what is sometimes called multithreaded or what we call here *shared-resource multiprocessing* (SRMP). In SRMP, each of the processors consists of basically only a register set, which includes a program counter, general registers, instruction counter, and so on. The driving principle behind SRMP is to make the best use of processor silicon area. The functional units and busses are time-shared. The objective is to eliminate context-switching overhead and to reduce the realized effect of branch and cache miss penalties. Each "processor" executes without significant instruction-level concurrency, so it executes more slowly than a more typical SISD, which reduces per instruction effect of processing delays; but the MIMD ensemble can achieve excellent speedup because of the reduced overhead. Note that this speedup is relative to a much slower single processor.

Shared-memory multiprocessing. In the simplest of these configurations, several processors share a common memory via a common bus. They may even share a common data cache or level-2 cache. As bus bandwidth is limited, the number of processors that can be usefully configured in this way is quite limited. Several processors sharing a bus are sometimes referred to as a "*cluster*."

Interconnected multiprocessors. Realizing multiprocessor configurations beyond the cluster requires an interconnection network capable of connecting any one of n processor memory clusters to any other cluster. The interconnection network provides n switched paths, thereby increasing the intercluster bandwidth at the expense of the switch latency in the network and the overall (considerable) cost of the network. Programming such systems may be done either as a shared-memory or message-passing paradigm. The shared-memory approach requires significant additional hardware support to ensure the consistency of data in the memory. Message passing has simpler hardware but is a more complex programming model.

Cooperative computing: networked multiprocessors. Simple processor-memory systems with LAN of even Internet connection can, for particular problems, be quite effective multiprocessors. Such configurations are sometimes called *network of workstations (NOW)*.

Table 7 illustrates some of the tradeoffs possible in configuring multiprocessor systems. Note that the application determines the effectiveness of the system. As architects consider various ways of facilitating interprocessor communication in a shared-memory multiprocessor, they must be constantly aware of the cost required to improve interprocessor communications. In a typical shared-memory multiprocessor, the cost does not scale linearly; each additional processor requires additional network services and facilities. Depending on the type of interconnection, the cost for an additional processor may increase at a greater than linear rate.

For those applications that require rapid communications and have a great deal of interprocessor communications traffic, this added cost is quite acceptable. It is readily justified on a cost-performance basis. However, many other applications, including many naturally parallel applications, may have limited interprocessor communications. In many simulation applications, the various cases to be simulated can be broken down and treated as independent tasks to be run on separate processors with minimum interprocessor communication. For these applications, simple networked systems of workstations provide perfectly adequate communications services. For applications whose program execution time greatly exceeds its interprocessor communication time, it is a quite acceptable message passing time.

The problem for the multiprocessor systems architect is to create a system that can generally satisfy a broad spectrum of applications, which requires a system whose costs scale linearly with the number of processors and whose overall cost effectively competes with the NOW—the simple network of workstations—on the one hand, and satisfies the more aggressive communications requirement for those applications that demand it on the other. As with any systems design, it is impossible to satisfy the requirements of all applications. The designer simply must choose

Table 7. Various Multiprocessor Configurations

Type	Physical sharing	Programmer's model	Remote data access latency	Comments
Multi-threaded	ALU, data cache, memory	Shared memory	No delay	Eliminates context switch overhead but limited possible Sp.
Clustered	Bus and memory	Shared memory	Small delay due to bus congestion	Limited Sp due to bus bandwidth limits.
Interconnection network (1)	Interconnection network and memory	Shared memory	Order of 100 cycles.	Typically 16–64 processors; requires memory consistency support.
Interconnection network (2)	Interconnection network	Message passing	Order of 100 cycles plus message decode overhead	Scalable by application; needs programmer's support.
Cooperative multiprocessors	Only LAN or similar network	Message passing	More than 0.1 ms.	Limited to applications that require minimum communications.

Table 8. Typical MIMD Systems

	Year of	Processor	Number of	Memory	Programming	introduction
System	Interconnection	element	processors	distribution	paradigm	type‡
Alliant FX/2800	1990	Intel i860	4–28	Central	Shared memory	Bus + crossbar
Stanford DASH	1992	MIPS R3000	4–64	Distributed	Shared memory	Bus + mesh
Cray T3D	1993	DEC 21064	128–2048	Distributed	Shared memory	3D torus
MIT Alewife	1994	Sparcle	1–512	Distributed	Message passing	Mesh
Convex C4/XA	1994	Custom	1–4	Global	Shared memory	Crossbar
Thinking Machines CM-500	1995	SuperSPARC	16–2048	Distributed	Message passing	Fat tree
Tera Computers MTA	1995	Custom	16–256	Distributed	Shared memory	3D torus
SGI Power Challenge XL	1995	MIPS R8000	2–18	Global	Shared memory	Bus
Convex SPP1200/XA	1995	PA-RISC 7200	8–128	Global	Shared memory	Crossbar + ring
Cray T3E-1350	2000	DEC 21164	40–2176	Distributed	Shared memory	3D torus

‡An indepth discussion of various interconnection networks may be found in *Parallel Computer Architecture: A Hardware / Software Approach* by David Culler and J.P. Singh with Anoop Gupta.

a broad enough set of applications and design a system robust enough to satisfy those applications. Table 8 shows some representative MIMD computer systems from 1990 to 2000.

COMPARISONS AND CONCLUSIONS

Examples of Recent Architectures

This section describes some recent microprocessors and computer systems, and it illustrates how computer architecture has evolved over time. In the last section, scalar processors are described as the simplest kind of SISD processor, capable of executing only one instruction at a time. Table 1 depicts some commercial scalar processors (7,8).

Intel 8086, which was released in 1978, consists of only 29,000 transistors. In contrast, Pentium III (from the same x86 family) contains more than 28,000,000 transistors. The huge increase in the transistor count is made possible by the phenomenal advancement in VLSI technology. These transistors allow simple scalar processors to emerge to a more complicated architecture and achieve better performance. Many processor families, such as Intel x86, HP PA-RISC, Sun SPARC and MIPS, have evolved from scalar processors to superscalar processors, exploiting a higher level of instruction-level parallelism. In most cases, the migration is transparent to the programmers, as the binary codes running on the scalar processors can continue to run on the superscalar processors. At the same time, simple scalar processors (such as MIPS R4000 and ARM processors) still remain very popular in embedded systems because performance is less important than cost, power consumption, and reliability for most embedded applications.

Table 2 shows some representative superscalar processors from 1992 to 2004 (7,8). In this period of time, the number of transistors in a superscalar processor has escalated from 1,000,000 to more than 100,000,000. Interestingly, most transistors are not used to improve the instruction-level parallelism in the superscalar architectures. Actually, the instruction issue width remains roughly the same (between 2 to 6) because the overhead (such as cycle time penalty) to build a wider machine, in turn, can adversely affect the overall processor performance. In most cases, many of these transistors are used in the on-chip cache to reduce the memory access time. For instance, most of the 140, 000,000 transistors in HP PA-8500 are used in the 1.5MB on-chip cache (512 KB instruction cache and 1MB data cache).

Table 3 presents some representative VLIW processors (7,9). There have been very few commercial VLIW processors in the past, mainly due to poor compiler technology. Recently, there has been major advancement in VLIW compiler technology. In 1997, TI TMS320/C62x became the first DSP chip using VLIW architecture. The simple architecture allows TMS320/C62x to run at a clock frequency (200MHz) much higher than traditional DSPs. After the demise of Multiflow and Cydrome, HP acquired their VLIW technology and co-developed the IA-64 architecture (the first commercial general-purpose VLIW processor) with Intel.

Although SISD processors and computer systems are commonly used for most consumer and business applications, SIMD and MIMD computers are used extensively for scientific and high-end business applications. As described in the previous section, vector processors and array processors are the two different types of SIMD architecture. In the last 25 years, vector processors have developed from a single processor unit (Cray 1) to 512 processor units (NEC SX-5), taking advantage of both SIMD and MIMD processing. Table 4 shows some representative vector processors. On the other hand, there have not been a significant number of array processors due to a limited application base and market requirement. Table 5 shows several representative array processors.

For MIMD computer systems, the primary considerations are the characterization of the memory address space and the interconnection network among the processing elements. The comparison of shared-memory and message-passing programming paradigms was discussed in the last section. At this time, shared-memory programming paradigm is more popular, mainly because of its flexibility and ease of use. As shown in Table 8, the latest Cray supercomputer (Cray T3E-1350), which consists of up to 2176 DEC Alpha 21164 processors with distributed memory modules, adopts the shared-memory programming paradigm.

Concluding Remarks

Computer architecture has evolved greatly over the past decades. It is now much more than the programmer's view of the processor. The process of computer design starts with the implementation technology. As the semiconductor technology changes, so to does the way it is used in a system. At some point in time, cost may be largely determined by transistor count; later as feature sizes shrink, wire density and interconnection may dominate cost. Similarly, the performance of a processor is dependent on delay, but the delay that determines performance changes as the technology changes. Memory access time is only slightly reduced by improvements in feature size because memory implementations stress size and the access delay is largely determined by the wire length across the memory array. As feature sizes shrink, the array simply gets larger.

The computer architect must understand technology, not only today's technology, but the projection of that technology into the future. A design begun today may not be broadly marketable for several years. It is the technology that is actually used in manufacturing, not today's technology that determines the effectiveness of a design.

The foresight of the designer in anticipating changes in user applications is another determinant in design effectiveness. The designer should not be blinded by simple test programs or benchmarks that fail to project the dynamic nature of the future marketplace.

The computer architect must bring together the technology and the application behavior into a system configuration that optimizes the available process concurrency, which must be done in a context of constraints on cost, power, reliability, and usability. Although formidable in objective, a successful design is a design that provides a lasting value to the user community.

FURTHER READING

W. Stallings, *Computer Organization and Architecture*, 5th ed.Englewood Cliffs, NJ: Prentice-Hall, 2000.

K. Hwang, *Advanced Computer Architecture*, New York: McGraw Hill, 1993.

J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, San Francisco, CA: Morgan Kaufman Publishers, 1996.

A. J. Smith, Cache memories, *Comput. Surv.*, **14** (3): 473–530, 1982.

D. Culler and J. P. Singh with A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, San Francisco, CA: Morgan Kaufmann Publishers, 1988.

D. Sima, T. Fountain, and P. Kacsuk, *Advanced Computer Architectures: A Design Space Approach*, Essex, UK: Addison-Wesley, 1997.

K. W. Rudd, *VLIW Processors: Efficiently Exploiting Instruction Level Parallelism*, Ph.D Thesis, Stanford University, 1999.

M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design*, Sudbury, MA: Jones and Bartlett Publishers, 1995.

P. M. Kogge, *The Architecture of Pipelined Computers*, New York: McGraw-Hill, 1981.

S. Kunkel and J. Smith, Optimal pipelining in supercomputers, *Proc. 13th Annual Symposium on Computer Architecture*, 1986, 404–411.

W. M. Johnson, *Superscalar Microprocessor Design*, Englewood Cliffs, NJ: Prentice-Hall, 1991.

BIBLIOGRAPHY

1. G. M. Amdahl, G. H. Blaauw, and F. P. Brooks, Architecture of the IBM System/360. *IBM J. Res. Develop.*, **8** (2): 87–101, 1964.
2. Semiconductor Industry Association, *The National Technology Roadmap for Semiconductors*, San Jose, CA: Semiconductor Industry Association, 1997.
3. M. J. Flynn, P. Hung, and K. W. Rudd, Deep-submicron microprocessor design issues, *IEEE Micro Maga.*, July-August, 11–22, 1999.
4. J. D. Ullman, *Computational Aspects of VLSI*, Rockville, MD: Computer Science Press, 1984.
5. W. Stallings, *Reduced Instruction Set Computers, Tutorial*, 2nd ed.New York: IEEE Comp. Soc. Press, 1989.
6. M. J. Flynn, Very high speed computing systems, *Proc. IEEE*, 54:1901–1909, 1966.
7. MicroDesign Resources, Microprocessor Report, various issues, Sebastopol, CA, 1992–2001.
8. T. Burd, General Processor Information, CPU Info Center, University of California, Berkeley, 2001. Available: http://bwrc.eecs.berkeley.edu/CIC/summary/.
9. M. J. Flynn and K. W. Rudd, Parallel architectues. *ACM Comput. Surv.*, **28** (1): 67–70, 1996.

MICHAEL FLYNN
PATRICK HUNG
Stanford University
Stanford, California

DATAFLOW COMPUTERS: THEIR HISTORY AND FUTURE

INTRODUCTION AND MOTIVATION

As we approach the technological limitations, concurrency will become the major path to increase the computational speed of computers. Conventional parallel/concurrent systems are based mainly on the control-flow paradigm, where a primitive set of operations are performed sequentially on data stored in some storage device. Concurrency in conventional systems is based on instruction level parallelism (ILP), data level parallelism (DLP), and/or thread level parallelism (TLP). These parallelisms are achieved using techniques such as deep pipelining, out-of-order execution, speculative execution, and multithreaded execution of instructions with considerable hardware and software resources.

The dataflow model of computation offers an attractive alternative to control flow in extracting parallelism from programs. The execution of a dataflow instruction is based on the availability of its operand(s); hence, the synchronization of parallel activities is implicit in the dataflow model. Instructions in the dataflow model do not impose any constraints on sequencing except for the data dependencies in the program. The potential for elegant representation of concurrency led to considerable interest in dataflow model over the past three decades. These efforts have led to successively more elaborate architectural implementations of the model. However, studies from past projects have revealed a number of inefficiencies in dataflow computing: the dataflow model incurs more overhead during an instruction cycle compared with its control-flow counterpart, the detection of enabled instructions and the construction of result tokens generally will result in poor performance for applications with low degrees of parallelism, and the execution of an instruction involves consuming tokens on the input arcs and *generating* result token(s) at the output arc(s), which involves communication of tokens among instructions. Recent advances that may address these deficiencies have generated a renewed interest in dataflow. In this article we will survey the various issues and the developments in dataflow computing.

This chapter is organized as follows: the Dataflow Principles section reviews the basic principles of the dataflow model. The discussion includes languages supporting dataflow model. The Dataflow Architectures section provides a general description of the dataflow architecture. The discussion includes a comparison of the architectural characteristics and the evolutionary improvements in dataflow computing, including pioneering pure dataflow architectures, hybrid architectures attempting to overcome the shortcoming of pure dataflow systems, and recent attempts to improve the hybrid systems. The next section outlines research issues in handling data structures, program allocation, and application of cache memories. Several proposed methodologies will be presented and analyzed. Finally, the last section concludes the article.

DATAFLOW PRINCIPLES

The dataflow model of computation deviates from the conventional control-flow method in two fundamental ways: asynchrony and functionality. Dataflow instructions are enabled for execution when all the required operands are available, in contrast to control-flow instructions, which are executed sequentially under the control of a program counter. In dataflow, any two enabled instructions do not interfere with each other and thus can be executed in any order, or even concurrently. In a dataflow environment, conventional concepts such as “variables” and “memory updating” are nonexistent. Instead, objects (data structures or scalar values) are consumed by an actor (instruction) that yields a result object that is passed to the next actor(s). It should be noted that some dataflow languages and architectures, however, use variables and memory locations for the purposes of convenience and of efficiency.

Dataflow Graphs

Dataflow graphs can be viewed as the machine language for dataflow computers. A dataflow graph is a directed graph, $G(N, A)$, where nodes (or actors) in N represent instructions and arcs in A represent data dependencies among the nodes. The operands are conveyed from one node to another in data packets called tokens. The basic primitives of the dataflow graph are shown in Fig. 1. A data value is produced by an operator as a result of some operation f. A true or false control value is generated by a decider (a predicate), depending on its input tokens. Data values are directed by means of either a switch or a merge actor. A switch actor directs an input data token to one of its outputs, depending on the control input. A Merge actor passes one of its input tokens to the output based on the value of the control token. Finally, a copy is an identity operator which duplicates input tokens. Figure 2 depicts the dataflow graph of the following expression:

$$sum = \sum_{i=1}^{\mathrm{N}} f(i)$$

Note the elegance and flexibility of the dataflow graph to describe parallel computation. In this example, the implicit parallelism within an iteration is exposed. Furthermore,

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

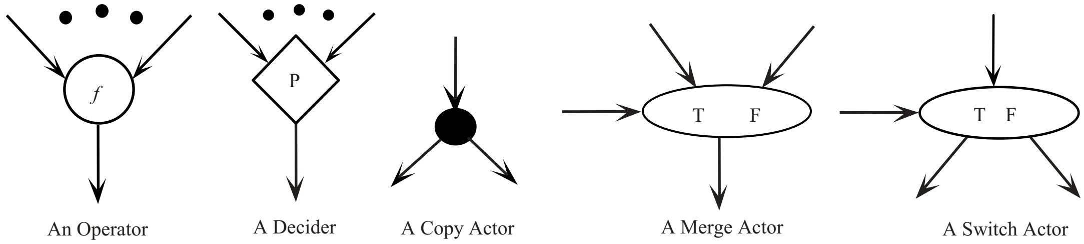


Figure 1. Basic primitives of the dataflow graph.

because of the functional properties of operations, the function *f* can be invoked simultaneously for all values of *i*. Thus, given sufficient amount of resources, *N* iterations of function *f* can be executed concurrently.

Dataflow Languages

Any dataflow language should permit the specification of programs that observe dataflow principles. In terms of programming language semantics, these principles translate into freedom from side-effects (prohibit modification of variables either directly or indirectly), single assignment (values associated with variables cannot be modified), and locality of effect (instructions do not have unnecessary far-reaching data dependencies). In this section we introduce three dataflow languages that received considerable attention in the literature.

VAL: A Value-oriented Algorithmic Language. VAL is a high level programming language developed at MIT (1), and can be viewed as a textual representation of dataflow graphs. VAL relies on pure functional language semantics to exploit implicit concurrency. Since dataflow languages use single assignment semantics, the implementation and the use of arrays present unique challenges (see Research Issues). In VAL, array bounds are not part of the type declarations. Operations are provided to find the range of indices for the declared array. Array construction in VAL is also unusual to improve concurrency in handling arrays. It should be noted that because we must maintain single assignment feature of functional languages, traditional language syntax to accumulate values (for example, the sum in Fig. 2) need some changes. To express such concurrencies, VAL provides parallel expressions in the form of *forall*. Consider the following examples:

```
1. forall i in [array_liml(a), array_limh(a)]

                a[i] := f(i);


2. forall i in [array_liml(a), array_limh(a)]

                eval plus a[i];
```

If one applies imperative semantics, both examples proceed sequentially. In the first case, the elements of the array *a* are constructed sequentially by calling the function *f* with different values of the index *i*. In the second example, we compute a single value that represents the sum of the elements of the array *a*, which represents sequential accumulation of the result. In VAL, the construction of the array elements in example 1 can proceed in parallel because all functions in VAL are side-effect free. Likewise, the accumulation in example 2 also exploits some concurrency because VAL translates such accumulations into a binary tree evaluation.

In addition to loops, VAL provides sequencing operations, *if-then-else* and *tagcase* expressions. When dealing with one of data type, tagcase provides a means of interrogating values with the discriminating unions.

VAL did not provide good support for input/output operation nor for recursion. These limitations allowed for a straightforward translation of programs to dataflow architectures, particularly static dataflow machines (see the earlier Dataflow Architectures section). The dynamic features of VAL can be translated easily if the machine supported dynamic graphs, such as the dynamic dataflow architectures.

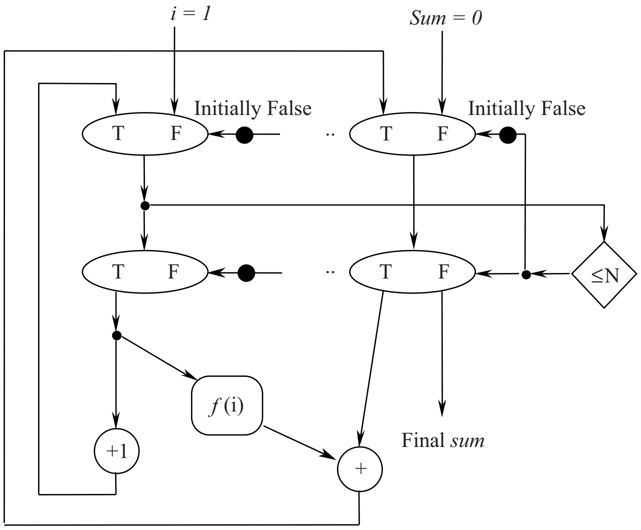


Figure 2. A dataflow graph representation of $sum = \sum_{i=1}^{N} f(i)$.

Id: Irvine Dataflow language. Id is a dataflow language that originated at the University of California-Irvine (2), and was designed to permit high-level programming language for the dynamic dataflow architecture proposed by Arvind (see the Earlier Dataflow Architectures section). Id is a block-structured, expression-oriented, single assignment language. An interpreter was designed to execute Id programs on dynamic dataflow architectures. Data types in Id are associated with values, and variables are typed implicitly by the values they carry. Structures include both arrays and (record) structures; and elements can be accessed using either integer indices or string values that define the name of the element (for example, t[“height”]). Structures are defined with two operators: select and append. Select is used to get the value of an element, whereas append is used to define a new structure by copying the elements of the original structure and adding new values defined by the append operation.

Id programs consist of side-effect free expressions and expressions (or subexpressions) can be executed in any order or concurrently based on the availability of input data. Loops in Id can be understood easily from the following example, which computes $\sum_{i=1}^{N} f(i)$

```
(initial i ← 1; sum ← 0;
while i ≤ N do new i ← i+1;
          new sum ← sum + f(i);
          return sum)
```

Id uses the concept of “new” to define a new value associated with an expression. It should be noted that a variable is not assigned a new value (like in conventional languages), but a new value is generated – variables are used only for the convenience of writing programs. It is also convenient to remember that the expressions in a loop can form recurrence expressions.

Procedures and functions in Id are pure functions and represent value(s) returned by the application of the function on the input values. Recursive procedures can be defined by associating names with procedure declarations. For example:

$$\mathrm{y} \leftarrow \text{procedure } \mathrm{f(n)}(\text{if } \mathrm{n} = 0 \text{ then } 1\text{else } \mathrm{n}^{*}\mathrm{f(n\text{-}1)})$$

defines factorial recursively, and we can invoke the procedure, for example as y(3).

Because no translators to convert Id programs to conventional (control-flow) architectures were developed, Id was used mostly by those with access to dynamic dataflow processors and to Id interpreters.

SISAL: Streams and Iterations in a Single Assignment Language. Sisal is the best-known dataflow language, mostly because of the support provided by the designers. Sisal received a fairly wide acceptance during the 1990s, because Sisal compilers generated optimized C as their intermediate representations and thus could be run on any platform with a C compiler. Although it is not as widely known now, Sisal translator and run-time support software are still available for Unix based systems and can be obtained from the web at http://sisal.sourceforge.net/. Sisal 2.0 provided multitasking (or multithreading) to support dataflow-style parallelism on conventional shared memory multiprocessors (4).

Sisal programs consist of one or more separately compilable units, which include a simple program, modules, and interfaces. A module is similar to a program but is not a starting point of execution. It pairs with an interface to export some of its types and function names. Like Id, Sisal supports scalar data types and structures (records, union, arrays, and streams). A stream is a sequence of values produced in order by one expression (thus it consists of homogeneous typed values), and is consumed in the same order by one or more other expressions. Sisal permits the creation of new values (and associates them with the same name).

```
for i := 1;
while ( i <5) do
          new i := i+2;
          j := i + new i;
returns product (i+j)
end for
```

This program constructs implicitly a stream of values inside the loop and returns the product of the elements of the stream. The values of the stream are the values of (i + j): 7, 13. Thus 91 is returned by the loop.

Sisal expressions can loop over the elements of an array (called array scattering) or over the elements of a stream (stream scattering) (5). As with VAL, Sisal can perform reduction operations concurrently using binary tree evaluations. Sisal has predefined reduction operations to evaluate sum, product, min, and max of a set of values. Catenate is a reduction that returns the concatenation of a sequence of one-dimensional array or a stream. Consider:

```
for i in[ 1..N] do
          return sum f(i);
```

which uses sum as a reduction operation to produce $\sum_{i=1}^{N} f(i)$.

Additional reduction functions can be defined by the programmer. Consider the following example to build histograms.

```
reduction histo(v, N: integer returns array of integer)
initial
          hacc := array[0..N+1:0] ;
 in

          idx := if(v <1) then 0
                    elseif (v > N) then n+1
                    else v
                    end if;
          new hacc := hacc[idx: hacc[ idx] +1]
returns hacc
end reduction
```

Sisal’s popularity also is caused by the concept of modules and interfaces: The interface shows the function templates that are visible publicly and the module defines the implementations of the functions. Sisal implementations also permit foreign code (written in a different language), by associating an interface with the foreign

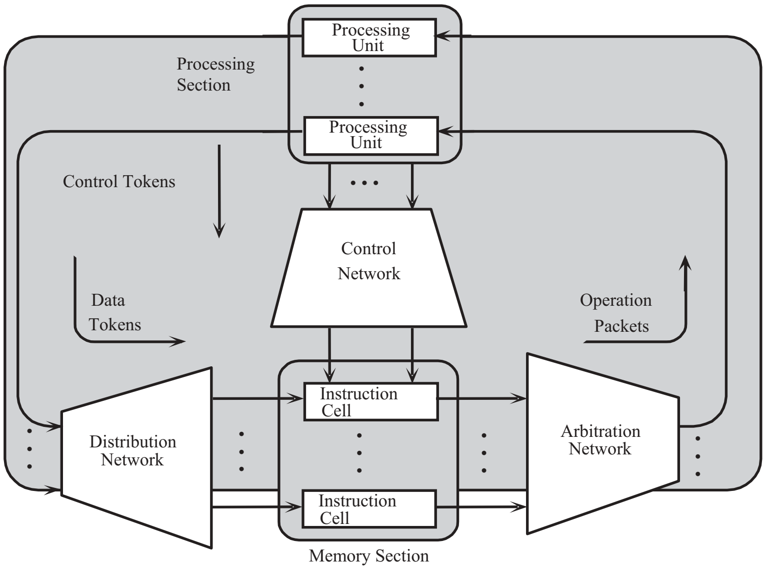


Figure 3. The basic organization of the static dataflow model.

code. Consider for example to import MathLib functions to Sisal programs:

```
interface MathLib in FORTRAN
          function sin (x: real returns real);
          function tan (x: real returns real);
end interface
```

DATAFLOW ARCHITECTURES

Architectural implementations of dataflow traditionally have been classified as either static or dynamic. The static approach allows at most one instance of a node to be enabled for firing, i.e., a dataflow actor can be executed only when all of its input tokens are available and no tokens exist on any of its output arcs. On the other hand, the dynamic approach permits simultaneous activation of several instances of a node during the run-time by viewing arcs as buffers containing multiple data items. To distinguish between different instances of a node (and routing data for different instantiations of the node), a tag is associated with each token that identifies the context in which a particular token was generated. An actor is considered executable when all of its input tokens with identical tags are available.

The static dataflow model has a simplified inherent mechanism to detect enabled nodes, but the model limits the performance because iterations are executed one at a time. The dynamic dataflow allows greater exploitation of parallelism; however, this advantage comes at the expense of the overhead in terms of the generation of tags, larger data tokens, and complexity of the matching tokens. A more subtle problem with the token matching is the complexity involved in allocation of resources (i.e., memory cells). A failure to find a match implicitly allocates memory within the matching hardware. If the matching unit becomes overcommitted, the program may deadlock.

Dataflow architectures have also been classified as pure dataflow architectures, macro dataflow architectures, and hybrid dataflow architectures. Detailed discussion about this classification is beyond the scope of this article and interested reader is referred to Ref. 6.

Earlier Dataflow Architectures

This section discusses three classic dataflow machines: the static dataflow machine, the (dynamic) manchester machine, and the explicit token store. These projects represent the pioneering work in the area of dataflow. The foundation they provide has inspired many other dataflow projects.

Static Model. The general organization of the original (static) dataflow machine is depicted in Fig. 3 (Table 1) (7). The memory section is a collection of memory cells, each cell composed of three memory words that represent an instruction template. The first word of each instruction cell contains op-code and destination address(es), and the next two words represent the operands. The design has envisioned six types of templates that represent binary and unary operators, binary and unary deciders (predicates), and binary and unary Boolean operators. The processing section is composed of five pipelined functional units, which perform the operations, form the result packet(s), and send the result token(s) to the memory section. The arbitration network is intended to establish a smooth flow of enabled instructions (i.e., instruction packet) from the memory section to the processing section. An instruction packet contains the corresponding op-code, operand value(s), and destination address(es). The distribution network is intended to transfer the result packets from the processing section to the memory section. Finally, the control network in designed to reduce the load on the distribution network by transferring the Boolean tokens and the acknowledgement signals from the processing section to the memory section.

Figure 4 shows the dataflow graph and the contents of the memory section for the $Y(t) = A^*X(t) + B^*Y(t\text{-}1)$

Table 1. Earlier Dataflow Architectures

Name	Country	Type
MIT Static Dataflow (1975) (7)	USA	Static
Manchester Dataflow (1977) (8)	England	Dynamic
MIT Tagged Token (1978) (9)	USA	Dynamic
CSIRAC II (1978) (9)	Australia	Dynamic
DDM1 Utah Data Driven (1978) (10)	USA	Static
LAU System (1979) (9)	France	Static
TI Distributed Data Processor (1979) (11)	USA	Static
NEC Image Pipelined Processor (1980) (12)	Japan	Static
NTT Dataflow Processor Array (1983) (13)	Japan	Dynamic
Distributed Data Driven Processor (1983) (14)	Japan	Dynamic
Stateless Dataflow Architecture (1983) (15)	England	Dynamic
SIGMA-1 (1984) (16)	Japan	Dynamic
Parallel Inference Machine (1984) (17)	Japan	Dynamic

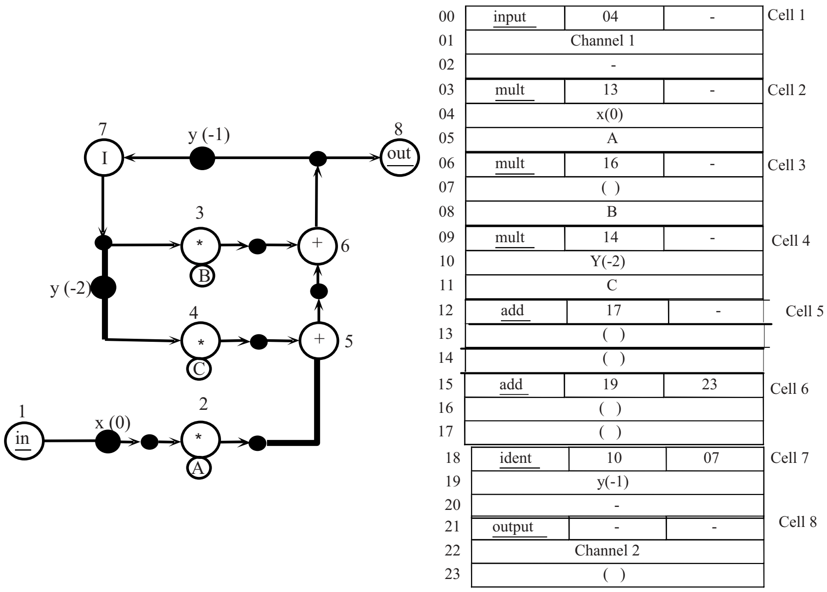
$+C^*Y(t\text{-}2)$. For the sake of simplicity, representation details in the memory are omitted. Note that each memory cell is numbered to correspond to the node number in the dataflow graph.

Manchester Dynamic Model. Figure 5 shows the block diagram of the *dynamic dataflow* system prototyped at the Manchester University (Table 1). It is designed as a back-end, composed of five units organized as a pipeline ring: The switch unit establishes communication between the front-end and back-end processor, and routes the result tokens back to the pipeline ring. The token queue is a First-in-first-out buffer that stores temporarily tokens traversing on the data-flow graph arcs. The basic operation of the matching unit is to bring together tokens with identical tags by pairing associatively tokens with the same destination node address and context. The dataflow program that represents the code for an operation is stored in the node store. The processing unit, a micro-programmed, 2-stage pipeline unit, executes the dataflow operations. The first stage handles the generation of result tokens and the association of tags with tokens. The second pipeline stage consists of 15 functional units to perform the necessary operations.

Explicit Token Store. Despite the potential parallelism promised by the dynamic dataflow model, early experiences have identified the following shortcomings in implementing the model:

- Overhead involved in matching tokens (and the need for associative matching),
- Complex resource allocation,
- The inefficiency of the dataflow instruction cycle (token driven models require multiple cycles through the pipeline to complete execution), and
- Nontrivial mechanisms to handle data structures (see the Research Issues section).

Performance of the dynamic dataflow architecture is related directly to the rate at which the matching unit operates. To facilitate this process while considering the cost, Arvind proposed a pseudo-associative matching mechanism that requires typically several memory accesses (9). A failure in finding a match implicitly allocates



00	input	04	-	Cell 1
01	Channel 1			
02	-			
03	mult	13	-	Cell 2
04	x(0)			
05	A			
06	mult	16	-	Cell 3
07	( )			
08	B			
09	mult	14	-	Cell 4
10	Y(-2)			
11	C			
12	add	17	-	Cell 5
13	( )			
14	( )			
15	add	19	23	Cell 6
16	( )			
17	( )			
18	ident	10	07	Cell 7
19	y(-1)			
20	-			
21	output	-	-	Cell 8
22	Channel 2			
23	( )			

Figure 4. A dataflow graph and its representation (MIT Static Model).

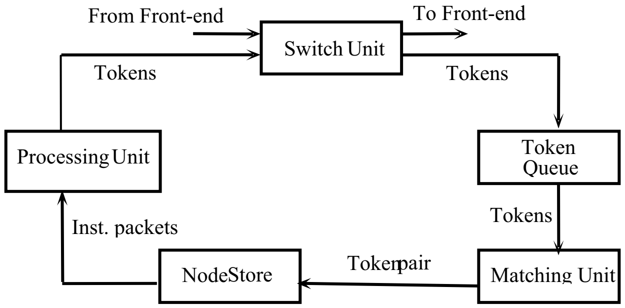


Figure 5. The Manchester Dynamic dataflow Machine.

memory within the matching unit — mapping a code-block to a processor places an unspecified commitment on the processor's matching unit. This can result in a deadlock if this resource becomes overcommitted. Another problem with dataflow processing is caused by the duration of the instruction cycle relative to its control-flow counterpart.

To overcome the inefficient matching of dynamic dataflow model, explicit token store (ETS) proposed a direct matching (9). Storage (called *activation frames*) is allocated dynamically for all the tokens that can be generated by a code block (a code block represents a function or a loop iteration). The usage of memory locations within the activation frame is determined at compile time; however the allocation of storage is determined at run time. A computation is described completely by an instruction pointer (IP) and an activation frame pointer (FP) and the pair <FP. IP> called a *continuation*. A typical instruction specifies an opcode, an offset in the activation frame where a match for its inputs will take place, and one or more displacements that define the destination instructions that will receive the result token(s). Each displacement is also accompanied by an input port (left/right) indicator that specifies the appropriate input arc for a destination actor. Figure 6 shows an example of the ETS code block invocation and its corresponding instruction and frame memory. When a token arrives at an actor (for example, Add), the IP part of the continuation points to the instruction that contains an offset *r* as well as displacements for the destination instructions. The system achieves the actual matching process by checking the disposition of the slot in the frame memory pointed to by $FP + r$. If the slot is empty, the system writes the token's value in the slot and sets its presence bit to indicate that the slot is full. If the slot is already full, the system extracts the value, leaving the slot empty, executes the corresponding instruction, and communicates the result tokens to the destination instructions by updating the IP according to the destinations encoded in the instruction.

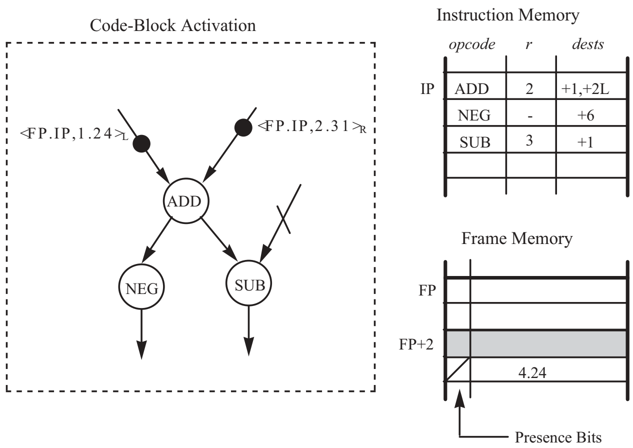


Figure 6. Explicit-token-store representation of a dataflow program.

Table 1 lists early dataflow architectures that have been advanced in the literature. Because of the space constraints, additional discussion about these architectures are beyond the scope of this article and the interested reader is referred to the cited references.

Dataflow Architectures of the 1980s and the 1990s

Relying on the lessons learned from early designs, several dataflow prototypes were designed during the 1980s and the 1990s. These include Monsoon, Epsilon-2, EM-4, P-RISC, and TAM. Table 2 summarizes the architectural characteristics of these designs (18–23).

These prototypes use the dynamic dataflow paradigm, primarily because of the success of direct matching of tokens proposed in ETS. The concept of a code-block in ETS permitted localization and efficient management of tokens. Activation frames can be allocated for different iterations of a loop, thus permitting the "unfolding" of loops.

Another major architectural change was the integration of the control-flow sequencing with the dataflow model. Dataflow architectures that are based on the pure dataflow model, such as the Manchester dataflow machine, provide well-integrated synchronization at the instruction level. However, this process is very inefficient when compared with the synchronization used in control-flow systems. It has been shown that it is more efficient to assign some of these responsibilities to the compiler and to use a simpler control-flow sequencing at run-time. The overhead of constructing and communicating result tokens can be reduced by using processor registers to hold intermediate results (similar to control-flow processors). The hybrid of dataflow flow with control-flow sequencing and usage of registers can be found in EM-4, and the Epsilon-2.

In contrast to these hybrid systems, the threaded abstract machine (TAM) provided a conceptually different perspective on the implementation of the dataflow model of computation. In TAM, the execution model for fine-grain parallelism is supported by an appropriate compilation strategy and program representation rather than by elaborate hardware. By assigning the synchronization, the scheduling, and the storage management tasks to the compiler, the use of processor resources can be optimized for the expected case, rather than for the worst case. In addition, because the scheduling of threads is visible to the compiler, TAM allowed for a more flexible use of registers across thread boundaries.

Recent Dataflow Projects

Since the mid 1980s, computer architecture has expended a considerable effort in exploitation of ILP as a means to improve performance. These efforts manifested in the

Table 2. Dataflow architectures of 80’s and 90’s.

Architecture	Key features
Monsoon (18)	• Joint venture between MIT and Motorola was an outgrowth of the MIT Tagged-Token Dataflow Architecture. • A collection of processing elements communicating with each other and a set of interleaved I-structure memory modules through a multistage packet switching network. • Direct matching of tokens based on the Explicit Token Store concept.
EM-4 (19)	• A highly parallel dataflow multiprocessor based on the SIGMA-1 project. It was an attempt to simplify the architecture by a RISC-based single-chip design, a direct matching scheme, and use of strongly connected arc model. • Use of registers to reduce the instruction cycle time and the communication overhead of transferring tokens. • Integration of a token-based circular pipeline and a register-based advanced control pipeline. • The prototyped processing element consists of a memory module and a single chip processor called EMC-R.
Epsilon-2 (20)	• Epsilon-2 is a multiprocessor dynamic dataflow model evolved from the Epsilon-1 project. It is composed of a set of processing modules contented via a global interconnection network. • Direct matching of tokens. • Repeat fan-out mechanism to reduce the overhead in copying tokens. • Control-flow type of sequencing and use of registers. Register contents are not necessarily preserved across grain boundaries. • Load balancing (adaptive routing).
P-RISC (21)	• P-RISC is a multiprocessor architecture strongly influenced by Iannucci’s dataflow/von Neumann hybrid architecture. It utilizes a RISC-like Instruction set and its generalization for parallel-RISC. Can use both conventional and dataflow compiling technologies. • Application of multithread using a token queue and circulating thread descriptors. • Introduction of Fork and Join instructions to spawn and synchronize multiple threads. • Synchronization of memory accesses by using I-structure semantics.
TAM (22)	• Placing all synchronization, scheduling, and storage management responsibility for execution of fine-grain parallelism explicit and under compiler control to relieve hardware complexity. This allows execution of dataflow languages on conventional control-flow processors. • Providing a basis for scheduling a number of threads within an activation as a quantum while carrying values in registers across threads. • Once an activation is made resident, all enabled threads within the activation execute to completion. • Having the compiler produce specialized message handlers as inlets to each code-block. • A prototype TAM instruction set, TL0 (Threaded Language Version 0), has been developed at the University of California at Berkeley.
*T (23)	• Tokens do not carry data, only continuations. • Provides limited token matching. • Overhead is reduced by off-loading the burden of message handling and synchronization to separate coprocessors.

design and the implementation of the so called superscalar, Very Long Instruction Word, super-speculative, and superpipelined organizations. In these organizations, ILP is exploited through deep pipelining and out-of-order execution of instructions. Aggressive exploitation of ILP is made possible by wide dispatch and issue of instruction, by a large issue buffers, by a large number of physical registers for register renaming, by a large number of functional units, and by a speculative execution of branches. Whether the instruction placement and issue are done statically as in VLIW architecture, or dynamically as in superscalar paradigm, the hardware complexity of these architectures, combined with the diminishing performance gains, renewed an interest in dataflow processing. This interest has resulted in several dataflow based architectures, such as the scheduled dataflow (SDF) (24), the EDGE (Explicit Data Graph Execution) (25), the WaveScalar (26), and the D^{2}NOW (Data-Driven Network of Workstations) (27).

Scheduled Dataflow (SDF). Unlike instruction level dataflow systems, SDF (24) uses dataflow-like synchronization at the thread-level, and control-flow semantics within a thread. A thread is allocated as an activation frame for receiving its inputs, similar to Cilk (28). A thread is enabled for execution when it has received all its inputs, and completes execution without interruption (viz., non-blocking threads). This approach minimizes instruction level communication, and because SDF threads are very fine-grained (typically a basic block), the amount of parallelism lost because of the sequential execution of instructions within a thread is minimal. Additionally, SDF decouples completely all memory accesses from execution pipelines, resulting in overlapped execution of threads. When a thread is enabled, SDF allocates a register set for the thread. Data is *pre-loaded* into the register set context prior to its scheduling on the execution pipeline. After a thread completes execution, the results are *post-stored* from its registers into memory. All memory accesses are performed by synchronization processors (SPs). The execution engines (EPs) rely on in-order execution of instructions within a thread. This architecture exploits two levels of parallelism: Multiple threads can be active simultaneously, permitting thread level parallelism, and the three phases of a thread execution (pre-load, execute, and post-store) can be overlapped with those of other threads. It is also possible to select appropriate number of SPs and EPs to meet application needs. Thread level speculation to improve performance of imperative programs is simplified in SDF system. Similar to the WaveScalar design, epoch numbers are associated with threads along with extended cache coherency protocols to commit (post-store) the results of a speculative thread in program order.

Explicit Data Graph Execution (EDGE). EDGE (25) is a static placement dynamic issue instruction model. It is designed to allow direct instruction communication: hardware delivers a producer's output directly as an input to a consumer instruction, (i.e., fine grained instruction scheduling). The TRIP architecture is an instantiation of an EDGE design. The TRIP prototype is a collection of 16 execution units that communicate with each other via a thin operand routing network. Each processing element includes an integer unit, a floating point unit, an operand router, and an instruction buffer of depth 128 (to hold multiple instructions and their operands). The scheduler determines which instructions to be assigned in each processor buffer (viz., Static placement). However, the availability of operands determines the order of the execution (viz., Dynamic issue).

WaveScalar. Similar to EDGE, WaveScalar (26) is a tiled architecture. It is a tagged-token dynamic dataflow machine composed of processing elements. Instructions are bound dynamically to processing elements during the execution phase. It should be noted that once an instruction is bound to a processing element, it can remain there for many dynamic executions. A processing element is composed of an interface to receive data tokens: a storage medium to store data tokens awaiting their matching partner. Upon the execution of an instruction, the output tokens are routed to the consumer processing element(s). To reduce the communication costs, the processing elements are connected through a hierarchical interconnection infrastructure—pairs of processing elements are coupled into *pods* sharing ALU results via a common bypass network. Four pods are grouped into *domains* that communicate over a set of pipelined busses. Four domains form a cluster supported by conventional memory hierarchy. To build larger machines, multiple clusters can be connected with each other by a grid-based, on-chip network.

Data-Driven Network of Workstations. D^{2}NOW (27) is a collection of off-the-shelf Pantium microprocessors interconnected through a fine-grained interconnection network that supports the thread level synchronization. The design is based on an earlier Decoupled Data Driven model of execution and in principal similar to the SDF model (24). To tolerate the communication latency, D^{2}NOW employs three mechanisms: fine-grained communication, medium-grained communication, and coarse-grained communication. The fine-grained communication is used for consumer identification and for data-token transfer. The medium-grained communication is for the medium size messages that identify a code block. The coarse-grained communication is through the Ethernet to support large data block transfers.

SOME RESEARCH ISSUES

Handling Data Structures

In pure dataflow model, no concept of a variable exists and data is exchanged in the form of tokens flowing between instructions (or if memory is used to store data, then variables can only be assigned a value once – the single assignment principle). To apply this property to arrays and structures, the entire structure must be carried as tokens (or new arrays and structures must be allocated, by copying unchanged items and by assigning new values to the elements that have been modified). In practical dataflow systems, a more efficient treatment of structures and of arrays is needed. The proposed solutions can be classified as either direct access or indirect access methods.

Direct Access Method. The direct access scheme treats each array elements as individual (scalar) data tokens — which eliminates the concept of an array (or structure). The token relabeling scheme proposed by Gaudiot is an example of direct access method (29). In this approach, tokens are identified by tags, associating the values with specific array elements. Although this method is simple, it requires entire data structures be passed from one node to the next or to be duplicated among different nodes. Moreover, in many applications, the notion of array as a single entity cannot be done away with completely, and thus the direct access method is inappropriate for such applications.

Indirect Access Method. In an indirect access scheme, arrays are stored in special (separate) memory units and their elements are accessed through explicit "read" and "write" operations. For example, in MIT, static dataflow machine arrays are represented as a heap forming a tree (30). VAL (see section on Dataflow Languages) provides constructs to generate and to access arrays. Arrays can be appended with new values, and arrays (and elements of arrays) are accessed using pointers. Modified elements can be made inaccessible by using reference counts with pointers (when the count becomes zero the element becomes inaccessible). The disadvantages of this method are: O(log n) time to access successive elements of an array and sequential nature of the append operations (only one element of an array can be modified at a time), which limits the performance of the system. It also becomes necessary to perform garbage collection of inaccessible elements.

I-structure. I-structures are asynchronous array-like structures that include a "presence" bit with each element of the structure, thus preventing access to undefined array elements and enforcing single assignment property (31).

University of Manchester Approach. This approach combines the concept of streams (i.e., a sequence of values communicated between two portions of a code) with conventional arrays (32). However, in contrast to streams, the size of the array structure must be known at the time it is created. Thus, a finite component is defined as a collection of elements, a "unit," on which the basic storage operations are performed. This scheme implies that the modification of any element(s) in an array requires copying the entire array. Sisal language (see the Dataflow Languages section) provides constructs to create and access arrays, as well as streams. An enhanced version permitted "in-place" updates to alleviate the need to copy unaffected elements.

Hybrid Scheme. The basic idea behind the hybrid scheme is to associate a template, called the structure template, with each conceptual array (33). For selective updates, this minimizes copying by allowing only the modified elements to be appended to the new array. Each array is represented by a hybrid structure that consists of a structure template and a vector of array elements. A structure template is subdivided into three fields:

- The reference count field; an integer indicating the number of references to the array,
- The location field; a string of 1's and 0's where the length of the string equals the total number of elements in the array. Each location bit determines whether the desired array element resides in the vector indicated by either the left ("0") or the right ("1") pointer, and
- The status bit (S); when an array is initially created, the status bit (S) is initialized to "0," which indicates that the vector contains the original array. Whenever a modification is made to an array with more than one reference, a new hybrid structure is created (the status bit set to "1") where all the modified elements can be accessed from the vector pointed by the right pointer. The sharing of array elements between the original and the modified array is achieved by linking the left pointer of the modified hybrid structure back to the original hybrid structure.

Program Allocation

To achieve maximum parallelism, programs must be partitioned and assigned to available resources. The goal is to maximize the parallelism (partition program into independent executable units) while minimizing communication among the executable units (by assigning dependent units to the same processing element). It has been shown that obtaining an optimal allocation of a graph with precedence requirements is an NP-complete problem. Two main (heuristic) approaches exist to allocate subtasks of a dataflow graph: static and dynamic. In static allocation, the tasks are allocated at compile-time using global information about the program and system resources. A dynamic allocation uses run-time information on processing loads and on program behavior to distribute tasks.

A number of heuristic algorithms have been developed for the allocation problem based on critical path list schedules. The basic idea behind these approaches is to assign a weight to each node of a directed graph that equals the maximum execution time from that node to an exit node (i.e., critical path). An ordered list of nodes is constructed according to their weights, which is then used to assign dynamically nodes with highest weights to processors as they become available. One major problem with critical path list schedules is the communication among the nodes. Enforcing only critical path scheduling, without considering the communication overhead, will not necessarily minimize the overall execution time.

In response, Ravi et. al. (34) proposed a variation of the critical path list scheduling which takes into account inter processor communication. In this method, rather than simply choosing the topmost node on the list, several top candidates whose critical paths fall within a certain range are considered for allocation. From this set of candidates, a node is selected which maximizes savings in communication time. To determine the compromise between computation and communication costs, the vertically layered (VL) allocation scheme was proposed in Ref. (35). The VL allocation scheme consists of two phases: separation and optimization. The basic idea behind the separation phase is to partition a dataflow graph into vertical layers such that each vertical layer represents a set of data dependent nodes that are executed in sequence (i.e., a thread). A density factor is used to distribute the directed paths among the processors. The optimization phase attempts to minimize the inter-processor communication costs by considering whether the inter-processor communication overhead offsets the advantage gained by overlapping the execution of two subsets of nodes on separate processors (collapsing the vertical layers assigned to different processors). The VL allocation scheme succeeds in balancing the load among the processors, however, by its very nature, it may not always reduce the total execution time.

A more general approach to program allocation was proposed by Sarkar and Hennessy (36). In contrast to the VL allocation scheme, this approach uses a greedy approximation algorithm. The algorithm begins with the trivial partition that places each node in a separate block. A table that represents the decrease in the critical path length obtained from merging a pair of blocks is maintained. It then merges iteratively blocks that result in the largest decrease in the critical path length. The algorithm is terminated when no remaining merger could possibly reduce the critical path length.

Despite the effectiveness of the aforementioned allocation schemes, one major problem still remains unresolved— the issue of handling dynamic parallelism. For example, a dynamic architecture unfolds a loop at run-time by generating multiple instances of the loop body and attempts to execute the instances concurrently. However, a single processor does not allow two simultaneous executions of a node, consequently, mapping the source dataflow graphs to processors, without special provisions to detect dynamic loop unfolding, results in the inability to exploit parallelism fully.

One solution is to provide a code-copying facility, where an instruction within a code block is duplicated among the available resources. Arvind has proposed a mapping scheme in which the instructions within a code block (called the logical domain) are mapped onto available procesors (called the physical domain) based on a hashing scheme (37). For example, if a physical domain consists of n processors, then the destination processor number can be $\text{processor}_{\text{base}} + i \bmod n$, where i is the iteration number. This will distribute the code uniformly over the physical domain. Because each of the n processors has a copy of the code, n iterations may be executed simultaneously. However, because not all program constructs can be unfolded in this manner, the question still remains as to how dynamic parallelism can be detected at compile-time effectively.

Cache in Dataflow

Multithreading can address memory latencies by context-switching to other thread while awaiting a memory access. In pure dataflow, each instruction can be viewed as a thread, causing excessive overheads. ETS based models and hybrid systems (see the Dataflow Architectures of the 1980s and the 1990s section) have created coarser grained threads using code blocks. Within the context of dataflow, threads are nonblocking: A thread is enabled for execution when it receives all inputs, and executes to completion without interruption or context switch. The nonblocking makes it difficult to overcome memory latencies because a thread cannot be context switched during its execution. Cache memories provide a solution, but in pure dataflow, because instruction execution is based on the availability of data, localities of instructions and data cannot be determined easily, making the inclusion of cache memories wasteful. Several innovative proposals for synthesizing localities in the context of dataflow exist. In Ref. (38), the concept of simultaneity of execution is used to define localities with code. A weight that represents the distance from the root is assigned to each dataflow node. The nodes with the same weight are then clustered on the same (cache) page. This strategy partitions the dataflow graph into K horizontal layers, such that the nodes in layer K_i are data independent from each other (hence they can likely be executed in parallel) and are data dependent on nodes in layer K_{i-1} ($1 < \mathrm{i} \leq \mathrm{K}$).

Other approaches to improve localities and cache in dataflow can be found in Ref. (39). Partitioning dataflow programs into threads will have a direct impact on localities. Allocation of threads to processing resources should use “cache affinities” to minimize cache misses and conflicts. An important issue in multithreading is the partitioning of programs into multiple sequential threads (see the Program Allocation section). The costs associated with creating threads and synchronization among threads will impact the granularity of threads and placement of threads. Schauser et al. (40) proposed a partitioning scheme using dual graphs. A dual graph is a directed graph with data, control, and dependence arcs: A data arc represents the data dependence between producer and consumer nodes. A control arc represents the scheduling order between two nodes, and a dependence arc specifies long latency operation caused by the message handlers (i.e., inlets and outlets) sending/receiving messages across code-block boundaries. The actual partitioning is performed using only the control and the dependence edges by first grouping the nodes based on dependence sets. The dependence sets are used to create safe partitions with no cyclic dependencies. A safe partition has the following characteristics: (1) no output of the partition needs to be produced before all inputs to the partition are available, (2) when the inputs to the partition are available, all the nodes in the partition can be executed, and (3) no arc connects a node in the partition to an input node of the same partition. A number of optimization techniques are performed on initial partitions to reduce synchronization costs. The output of the partitioner is a set of threads where the nodes in each thread are executed sequentially and the synchronization requirement determined statically only occurs at the beginning of a thread. SDF and TAM (see the Dataflow Architectures section) use similar ideas for thread generation.

In the static dataflow architecture, localities can be exploited by concentrating on the static order of the dataflow program. The dynamic approach permits the activation of several instances of a node during run-time. To exploit the temporal and the spatial localities in dataflow programs that run on dynamic dataflow models, it is necessary to separate instruction memory from the operand memory. However, asynchrony of the dataflow instructions means frequent context switching, and in general, lack of temporal and spatial localities in accessing instruction and operand memories (41). To cope with these problems, one needs to adopt proper mechanisms to partition the dataflow graphs into subgraphs, to allocate subgraphs among processors, and to control the number of instances of a subgraph in a processor. It should be noted that because of the functional and asynchronous nature of the dataflow instructions, the addresses of the nodes in a dataflow graph can be set as desired without affecting the result of the execution. This property should be the basis of establishing localities in dataflow programs. Moreover, in support of the cache organization, one should study the effectiveness of the traditional statistical replacement algorithms (e.g., LRU) for instruction and operand memories. Therefore, in a processor with the load control mechanism, a sophisticated deterministic algorithm to replace dataflow blocks needs to be developed. Finally, the operand memory plays a dominant role to achieve satisfactory performance in a dataflow machine, and hence the operand cache must be managed effectively. In a dataflow machine, it is not only necessary to maintain spatial locality for the input arguments of a code-block (frame), but also is necessary to maintain spatial locality for the result tokens of the code-block. In the other words, the cache management must keep track of several active frames to avoid cache misses in accessing arguments while storing the results. These design principles motivated the organizations of operand and instruction caches in the literature (41).

CONCLUSION

As modern architects find it difficult to design highly parallel architectures that can exploit high degrees of instruction level parallelism, it may be time to look back to dataflow model of computation. The dataflow model was investigated in 1970s and 1980s but no commercially viable systems were implemented. Nevertheless, several features of the dataflow principle and dataflow computation have found their place in modern processor architectures and compiler technology. Most modern processors use complex hardware techniques to detect data hazards, control hazards, and dynamic parallelism — to bring the execution engine closer to an idealized dataflow engine. Some researchers have proposed hybrid designs in which the dataflow scheduling is applied only at thread level (i.e., macro-dataflow), whereas each thread is comprosed of conventional control-flow instructions.

The advances from the development of dataflow projects indicate potential high performance computation based on the dataflow principles. However, before a successful implementation of dataflow machines is possible, the various issues discussed in this article must be resolved.

It is our contention that a more careful mix of dataflow principles with recent technological advances will pave the way to future tera and peta instructions per second performance. Current multicore and multithreaded systems do not scale well. Instruction level dataflow implementations potentially can scale with processing resources, but they require excessive hardware support for synchronization, distribution, and communication among the instructions. A combination of static (compile time) and dynamic techniques for the creation and distribution of threads (or a unit of concurrent activity) may provide a balance between performance and hardware complexity.

The implementation of imperative memory systems within the context of a dataflow model is yet another issue that is not addressed satisfactorily. In addition to the management of structures, techniques for the management of pointers, dealing with aliasing, and dynamic memory management are needed. Ordering of memory updates (a critical concept in shared memory concurrency) is an alien concept to pure dataflow. However, to be commercially viable, it is essential to provide shared memory based synchronization among concurrent activities. Some ideas such as those presented in Wavescalar and SDF hold some promise in this connection.

We believe that several factors are motivating a renewed interest in the design and implementation of scalable dataflow processors. These include the recent technological advances in increased chip density; the complex interconnections among multiple processing elements on a single chip (Network-on-a-chip); the hardware complexity of the superscalar, super pipeline, and VLIW architectures and the diminishing performance gains of these systems with additional hardware; the large and multi-level caches; the compliers that perform extensive global and interprocedural analyses to extract as much parallelism as possible, and finally, the success of the recent dataflow projects (Recent Dataflow Projects section). At the same time, if dataflow architecture is to address the challenges of future processing architectures containing hundreds, if not thousands, of processing elements, it is necessary to evaluate carefully different forms of dataflow organizations for their suitability for implementation.

BIBLIOGRAPHY

1. W. B. Ackerman, Dataflow languages, *IEEE Computer*, **15** (2): 15–23, 1982.
2. R. S. Nikhil, *Id World Reference Manual*, MIT Laboratory for Computer Science, Cambridge, MA, 1985.
3. J. Feo, D. Cann, and R. Oldehoeft, A report on the Sisal language project, *J. Parallel Distribut. Comput.*, **10**: 249–365, 1990.
4. R. Oldehoeft and D. Cann, Applicative parallelism on a shared-memory multiprocessor, *IEEE Software*, **5** (1): 62–70, 1988.
5. Sisal Lives, Available: http://sisal.sourceforge.net/
6. B. Lee and A. R. Hurson, Dataflow architectures and multithreading, *IEEE Computers*, **27** (8): 27–38, 1994.
7. J. B. Dennis and D. P. Misunas, A preliminary architecture for a basic dataflow processor, *Proc. Symposium on Computer Architecture*, 1975, pp. 126–132.
8. J. R. Gurd, C. C. Kirkham, and I. Watson, The manchester prototype data-flow computer, *Comm. ACM*, **28** (1): 34–52, 1985.
9. Arvind and D. E. Culler, Dataflow Architectures, *Ann. Rev. Comp. Sci.*, **1**: 225–253, 1986.

10. A. L. Davis, The architecture and system Method of DDM1: A recursively structured data driven machine, *Proc. Symposium on Computer Architecture*, 1978, pp. 210–215.
11. M. Cornish, The TI dataflow architecture: The power of concurrency for avionics, *Proceedings of Third Conference on Digital Avionics Systems*, 1979, pp. 19–25.
12. Y. M. Chong, Dataflow chip optimizes image processing, *Computer Design*, 97–103, 1984.
13. N. Takahashi and M. Amamiya, A dataflow processor array system: Design and analysis, *Proc. Symposium on Computer Architecture*, 1983, pp. 243–250.
14. M. Kishi, H. Yasuhara, and Y. Kawamura, DDDP: A distributed data driven processor, *Proc. Symposium on Computer Architecture*, 1983, pp. 236–242.
15. D. F. Snelling, The design and analysis of a Stateless Data-Flow Architecture, *Tech. Report UMCS-93-7-2*, University of Manchester, 1993.
16. T. Shimada et al., Evaluation of a prototype data flow processor of the SIGMA-1 for scientific computations, *Proc. Int. Symposium on Computer Architecture*, 1986, pp. 226–234.
17. N. Ito, M. Kishi, E. Kuno, and K. Rokusawa, The data-flow based parallel inference machine to support two basic languages in KL1, *Proc. IFIP TC-10 Working Conf. Fifth Generation Comp. Arch.*, 1985, pp. 123–145.
18. D. E. Culler and G. M. Papadopoulos, The explicit token store, *J. Parall. Distribut. Comput.*, **10**: 289–308, 1990.
19. M. Sato et al., Thread-based programming for EM-4 hybrid dataflow machine, *Proc. Symposium on Computer Architecture*, 1992, pp. 146–155.
20. V. G. Grafe and J. E. Hoch, The epsilon-2 multiprocessor system, *J. Paral. & Distribut. Comput.*, **10**: 309–318, 1990.
21. R. S. Nikhil and Arvind , Can Dataflow Subsume von Neumann Computing?*Proc. Int. Symposium on Computer Architecture*, 1989, pp. 262–272.
22. D. E. Culler, S. C. Goldstein, K. E. Schauser, and T. Eicken, TAM — A compiler-controlled threaded abstract machine, *J. Paral. Distribut. Comput.*, **18**: 347–370, 1993.
23. R. S. Nikhil, G. M. Papadopoulos, and Arvind , *T: A Multithreaded Massively Parallel Architecture, *Proc. Int. Symposium on Computer Architecture*, 1992, pp. 156–167.
24. K. M. Kavi, R. Giorgi, and J. Arul, Scheduled dataflow: execution paradigm, architecture, and performance evaluation, *IEEE Trans. Comp.*, **50** (8): 834–846, 2001.
25. D. Burger et al., Scaling to the end of silicon with EDGE architectures, *IEEE Computer*, **37** (7): 44–55, 2004.
26. S. Swanson et al., Area-performance trade-offs in tiled dataflow architectures, *IEEE*, **34** (2): 314–326.
27. C. Kyriacou, P. Evipidou, and P. Trancoso, Data-driven multithreading using conventional microprocessors, *IEEE Trans. Parallel Distribut. Sys.*, **17** (10): 1176–1188, 2006.
28. R. D. Blumofe et al., Cilk: An Efficient Multithreaded Run-time System, *ACM Symposium on Principles and Practice of Parallel Programming (PPoP)*, 1995.
29. J.-L. Gaudiot and Y. H. Wei, Token relabeling in a tagged token data-flow architecture, *IEEE Trans. Comp.*, **38** (9):1225–1239, 1989.
30. W. B. Ackerman, A structure processing facility for dataflow computers, *Proc. of the International Conference on Parallel Processing*, 1978, 166–172.
31. Arvind, R. S. Nikhil, and K. K. Pingali, I-structures: Data structures for parallel computing, *Proc. of the Workshop on Graph Reduction*, Los Alamos, NM, 1986.
32. L. M. Patnaik, R. Govindarajan, and N. S. Ramadoss, Design and performance evaluation of EXMAN: an extended manchester dataflow computer, *IEEE Trans. Comp.*, **35** (3): 229–243, 1986.
33. B. Lee, A. R. Hurson, and B. Shirazi, A hybrid scheme for processing data structures in a dataflow environment, *IEEE Trans. Parallel Distribut. Sys.*, **3** (1): 83–96, 1992.
34. T. M. Ravi, M. D. Ercegovac, T. Lang, and R. R. Muntz, Static allocation for a dataflow multiprocessor system, *2^{nd} Int. Conference on Supercomputing*, 1987.
35. B. Lee, A. R. Hurson, and T. Y. Feng, A vertically layered allocation scheme for dataflow computers, *J. Parallel Distribut. Comput.*, **11**: 175–187, 1991.
36. V. Sarkar and J. Hennessy, Compile-time partitioning and scheduling of parallel programs, *Proc. SIGPLAN Symposium on Compiler Construction*, 1986, pp. 17–26.
37. Arvind , Decomposing a program for multiprocessor system, *Proc. of the International Conference on Parallel Processing*, 1980, pp. 7–14.
38. J. T. Lim, A. R. Hurson, and L. D. Pritchett, searching for locality in program graphs, *Paral. Comput. Technol.*, LNCS **2763**: 276–290, 2003.
39. A. R. Hurson, K. Kavi, B. Lee, and B. Shirazi, Cache memories in dataflow architectures: a survey, *IEEE Parall. Distribut. Technol.*, **4** (4): 50–64, 1996.
40. K. E. Schauseret al., Compiler-controlled multithreading for lenient parallel languages, *Proc. of the ACM Conference on Functional Programming Languages and Computer Architecture*, 1991, pp. 50–72.
41. M. Takesau, Cache memories for dataflow machines, *IEEE Trans. Comput.*, **41** (6): 677–687, 1992.

FURTHER READING

G. M. Papadopoulos, *Implementation of a General-Purpose Dataflow Multiprocessor*, Cambridge, MA: *MIT Press*, 1991.

K. Kavi, A. R. Hurson, P. Patadia, E. Abraham, and P. Shanmugam, Design of cache memories for multithreaded dataflow architecture, *Proc. of Symposium on Computer Architecture*, 1995, 253–264.

Ali R. Hurson
University of Missouri-Rolla
Rolla, Missouri
Krishna M. Kavi
The University of North Texas
Denton, Texas

DATA STORAGE ON MAGNETIC DISKS

INTRODUCTION

Data storage can be performed with several devices, such as hard disk drives (HDDs), tape drives, semiconductor memories, optical disks, and magneto-optical disks. However, the primary choice of data recording in computer systems has been based on HDDs. The reasons for this choice include the faster access times at relatively cheaper costs compared with other candidates. Although the semiconductor memories may be faster, they are relatively more expensive than the HDDs. On the other hand, storage on optical disks used to be cheaper than the hard disk storage, but they are slower. In the recent past, HDD technology has grown at such a rapid pace that it is faster than the optical disks but costs almost the same in terms of cost per bit.

HDDs have grown at an average rate of 60% during the last decade. This growth has led to enhanced capacity. The increase in the capacity of hard disks, while maintaining the same manufacturing costs, leads to a lower cost per GB (gigabytes $= 1 \times 10^9$ bytes). At the time of writing (August 2007), HDDs with a capacity of about 1 TB (terabytes $= 1 \times 10^{12}$ bytes) have been released. At the same time, the size of the bits in HDDs has reduced significantly, which has led to an increase of areal density (bits per unit area, usually expressed as Gb/in^2, gigabits per square inch). The increase of areal density has enabled HDDs to be used in portable devices as well as in digital cameras, digital video cameras, and MP3 players.

To fuel this tremendous improvement in technology, significant inventions need to be made in the components as well as in the processes and technologies associated with HDDs. This article will provide a brief background of the magnetic disks that store the information in HDDs. Several articles on the Web are targeted at the novice (1–3). Also, several articles in the research journals, are targeted at the advanced reader (4–7). This article will try to strike a balance between basic and advanced information about the HDDs.

DATA STORAGE PRINCIPLES

Any data storage device needs to satisfy certain criteria. It needs to have a storage medium (or media) to store the data, ways to write the data, ways to read the data, and ways to interpret the data. Let us take the example of a book. In a printed book, paper is the storage medium. Writing information (printing) is done using ink, and reading is carried out with the user's eyes. Interpretation of the data is carried out using the user's brain. Components with similar functions exist in an HDD too.

Figure 1(a) shows the components of a typical HDD, used in desktop PCs. Some key components that make up an HDD are marked. A disk, a head, a spindle motor, an actuator, and several other components are included. The disk is the recording medium that stores the information, which is similar to the paper of a book. The head performs two functions (similar to pen and eye in our example), which are writing and reading information. The spindle motor helps to spin the disk so that the actuator (that moves along the radial direction) can carry the head to any part of the disk and read/write information. The printed circuit board [Fig. 1(b)] is the brain of the HDD, which helps to control the HDD and pass meaningful information to the computer or whatever device that uses the HDD. Before we learn about these components in detail, let us understand the principles of magnetic recording, which is the basis for storage of information in HDDs.

Magnetic recording was demonstrated successfully as a voice signal recorder by Vladimir Poulsen about 100 years ago. Later, it was exploited for storing all kinds of information. In simple terms, magnetic recording relies on two basic principles: (*1*) Magnets have north and south poles. The field from these poles can be sensed by a magnetic field sensor, which is a way of reading information. (*2*) The polarity of the magnets can be changed by applying external magnetic fields, which is a way of writing information. Although earlier systems of magnetic recording such as audio tapes and video tapes are analog devices, HDDs are digital devices, which make use of a string of 1 s and 0 s to store information. In the next few sections, the key technologies that aid in advancing the storage density of HDDs will be discussed briefly. The main focus of the article is, however, on the magnetic disks that store the information.

HEADS

The head is a tiny device [as shown in Fig. 1(a)] that performs the read–write operation in an HDD. The heads have undergone tremendous changes over the years. In the past, both reading and writing operations were carried out using an inductive head. Inductive heads are a kind of transducer that makes use of current-carrying coils wound on a ferromagnetic material to produce magnetic fields (see Fig. 2). The direction of the magnetic field produced by the poles of the ferromagnetic material can be changed by changing the direction of the electric current. This field can be used to change the magnetic polarities of the recording media (writing information).

Inductive heads can also be used for reading information, based on Faraday's law, which states that a voltage will be generated in a coil, if a time-varying flux (magnetic field lines) is in its vicinity. When the magnetic disk (in which information is written) rotates, the field that emanates from the recording media bits will produce a time-varying flux, which will lead to a voltage in the inductive head. This voltage can be used to represent "1"s or "0"s. The inductive head technology was prevalent until the early 1990s. However, to increase the bit density, the bit size had to be decreased, which led to a decrease in the magnetic flux

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

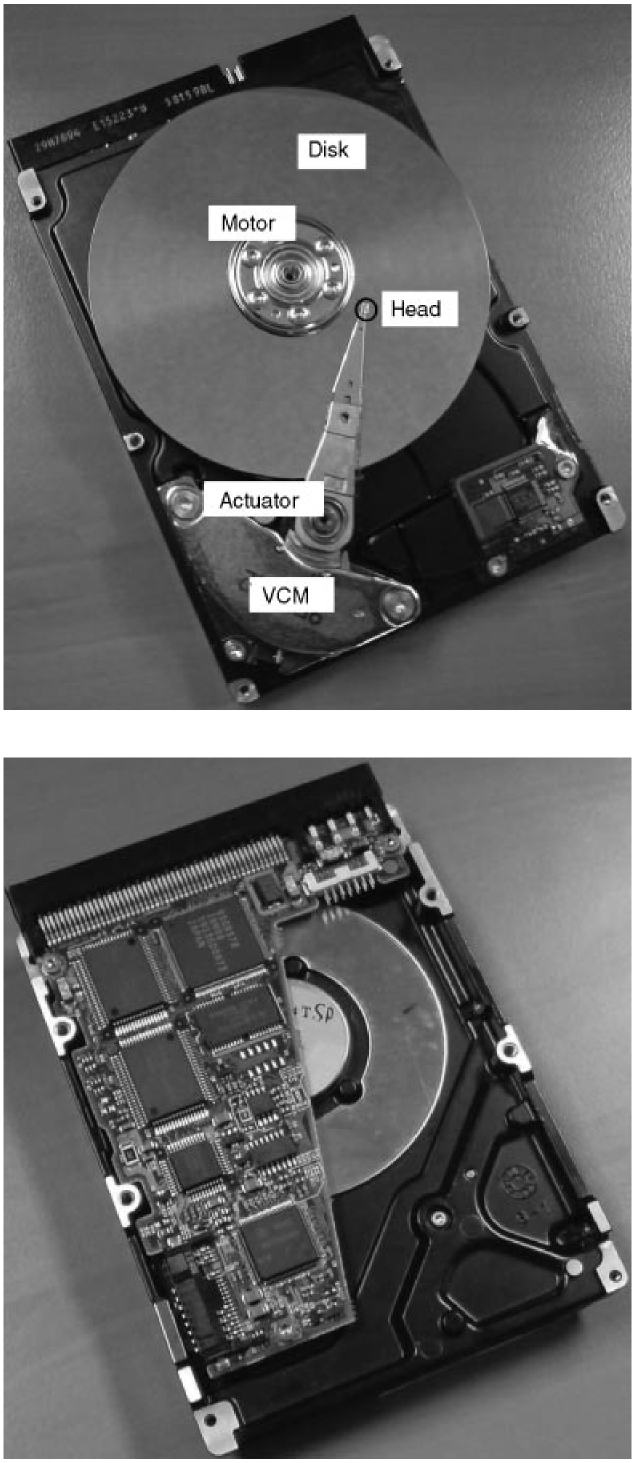


Figure 1. Typical components of an HDD: (a) view of inside parts of the HDD and (b) view of the printed-circuit-board side.

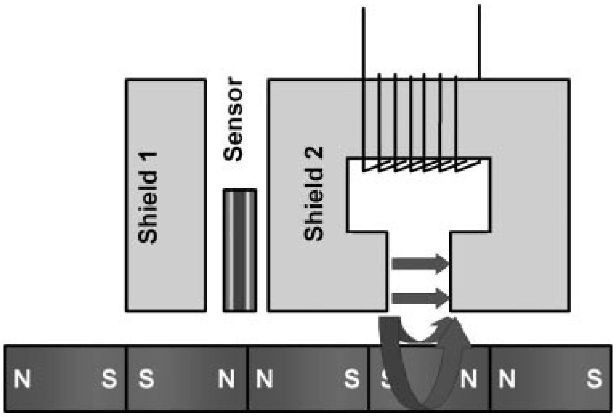


Figure 2. Schematic diagram of a head in an HDD showing the writer (inductive head), reader, and media below.

from the bits. The inductive heads were not sensitive enough to the magnetic field from the smaller bits as the technology progressed. Therefore, more advanced read sensors found their way into the head design. Modern HDDs have two elements in a head: One is a sensor for reading information (analogy: eye), and the other is an inductive writer for writing information (analogy: pen). Such components where the sensor and writer are integrated are called integrated heads or simply heads.

Figure 2 shows the schematic diagram of an integrated head, which consists of a writer and a reader. The inductive head on the right side of the image is the writer that produces the magnetic field to change the magnetic state of the region of the magnetic disk to be in one way or the other. Figure 2 also shows a read-sensor (or reader), sandwiched between two shields. The shield 2, as shown in the figure, is usually substituted with one side of the core of the inductive head. The shields are ferromagnetic materials that would shunt away any field from bits that are away from the region of interest, in a similar way as the blinders prevent a horse from seeing on either side. Because of the shields, only the magnetic field from the bit (that is being read) is sensed by the reader.

The HDDs used magneto-resistive (MR) heads for some time (early to late 1990s) before switching to prevalent giant magneto-resistive (GMR) sensors. Unlike inductive heads, MR and GMR heads work on the basis of change in the resistance in the presence of a magnetic field. The GMR sensor, which is shown in multicolor, is in fact made of several magnetic and nonmagnetic layers. GMR devices make use of the spin-dependent scattering of electrons. Electrons have "up" and "down" spins. When an electric current is passed through a magnetic material, the magnetic orientation of the magnetic material will favor the movement of electrons with a particular spin— "up" or "down". In GMR devices, the magnetic layers can be designed in such a way that the device is "more resistive" or "less resistive" to the flow of electrons depending on the direction of the field sensed by the sensors. Such a change in the resistance can be used to define "1" and "0" needed for digital recording.

RECORDING MEDIA

As mentioned, this article will discuss the recording media—the magnetic disks that store the information—in more detail. Historically, the first recording media for HDD was made by spraying a magnetic paint on 24” aluminium disks. These types of recording media, where fine magnetic particles are embedded in a polymer-based solvent, are called particulate media. However, they became obsolete in the late 1980s. Thin film technology took the place of the old technology to coat the recording media.

Fabrication of Magnetic Disks

In thin film technology, the recording media consists of several layers of thin films deposited by a process called sputtering. Figure 3 shows a typical schematic cross section of the layers that may constitute a perpendicular recording medium, which is an emerging technology at the time of writing. Sputtering, simply said, is a process of playing billiards with atoms to create thin film layers. The sputtering process is carried out in vacuum chambers and is described here briefly. The sputtering chamber, where the thin film deposition would take place, is pumped down to a low pressure at first. The objective of achieving this low pressure, called the base pressure, is to minimize the water vapor or other contaminating elements in the deposition chamber. After the base pressure is achieved, whose value depends on the application, argon (Ar) or some other inert gas is passed into the chamber. When a high negative voltage is applied between the target (the material that needs to be ejected) and the sputtering chamber (ground), the Ar ions that were created by the discharge in the sputtering chamber and accelerated because of the high-voltage knock on the target and release atoms. These atoms are scattered in different directions, and if the sputtering system was optimized well, most of them would arrive at the substrate (the disk that has to be coated).

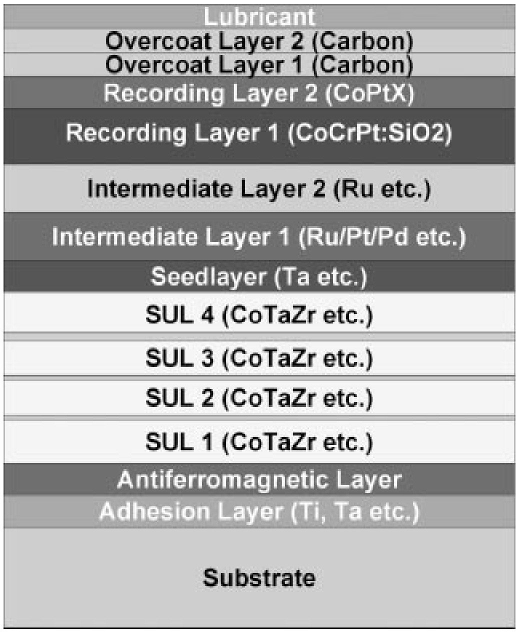

Lubricant
Overcoat Layer 2 (Carbon)
Overcoat Layer 1 (Carbon)
Recording Layer 2 (CoPtX)
Recording Layer 1 (CoCrPt:SiO2)
Intermediate Layer 2 (Ru etc.)
Intermediate Layer 1 (Ru/Pt/Pd etc.)
Seedlayer (Ta etc.)
SUL 4 (CoTaZr etc.)
SUL 3 (CoTaZr etc.)
SUL 2 (CoTaZr etc.)
SUL 1 (CoTaZr etc.)
Antiferromagnetic Layer
Adhesion Layer (Ti, Ta etc.)
Substrate

Figure 3. Typical cross-sectional view of a recording medium that shows different layers.

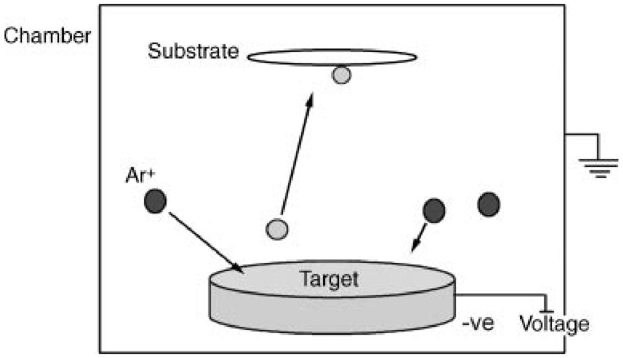


Figure 4. Illustration of the sputtering process inside a sputtering chamber.

Several modifications can be made to the process described above to improve the quality of the films that are obtained. In modern sputtering machines used in the hard disk industry, magnetron sputtering, where magnets are arranged below the targets and can even be rotated, is used to deposit thin film at faster deposition rates with good uniformity over the entire substrate. Moreover, since the recording media have several layers, the modern machines also have several sputtering chambers in a single sputtering machine, so that all layers can be deposited subsequently without exposing the layers to the ambient conditions. Figure 4 illustrates the sputtering process. Figure 5 shows the configuration of chambers in a sputtering system from Oerlikon (8). In the media fabrication process, the disk goes through all the chambers to coat the various layers depicted in Fig. 3.

Magnetic Recording Schemes

Figure 6 shows the way the data are organized on magnetic disks (also called platters). Several platters may be stacked in an HDD to multiply capacity. In almost all HDDs, the information is stored on both sides of the disks. Read/write heads exist on both surfaces, which perform reading and writing of information. For the sake of simplicity, only one side of the disk surface is shown here. As can be shown, the information is stored in circular tracks. Within the tracks, addressed sectors exist, in which the information can be written or read. The randomness in access/storage of information from/in an address provided by the CPU comes from the ability to move the head to the desired sector. In the state-of-the-art hard-disk media, about 150,000 tracks exist running from the inner diameter (ID) of the disk to the outer diameter (OD). The tracks are packed at such a high density that it is equivalent to squeezing about 600 tracks in the width of a typical human hair (considering the average diameter of the human hair to be around 100 μm). In each track, the bits are packed at a density of 1 million bits in an inch (about 4000 bits in the width of a human hair). If a nano-robot has to run on all the tracks to have a look at the bits of a contemporary hard disk, it would have almost completed a marathon race. That is how dense are the bits and tracks in an HDD. This trend of squeezing the tracks and bits in a smaller area will continue. Figure 7

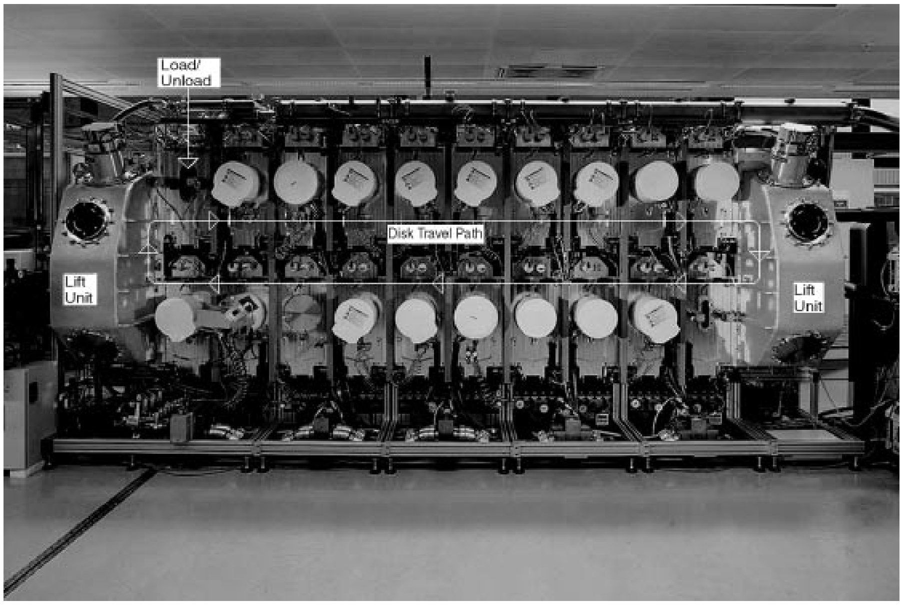


Figure 5. Picture of a sputtering system with different chambers for manufacturing hard disk media. The direction of movement of the disk is marked by arrows.

shows the trend of increase in the areal density (number of bits in a square-inch) as a function of time. It can be noticed that the areal density has increased by about 100 million times in the past five decades.

Figure 8 illustrates the recording process in the longitudinal recording technology. In this technology, the polarities of the magnets are parallel to the surface of the hard disk. When two identical poles (S–S or N–N) are next to each other, a strong magnetic field will emerge from the recording medium, whereas no field will emerge when opposite poles (S–N) are next to each other. Therefore, when a magnetic field sensor (GMR sensor, for example) moves across this surface, a voltage will be produced only when the GMR sensor goes over the transitions (regions where like poles meet). This voltage pulse can be synchronized with a clock pulse. If during the clock window, the GMR sensor produces a voltage, it represents 1. If no voltage is produced during the clock window, it represents 0. This simple illustration shows how "1"s and "0"s are stored in hard disk media.

In our previous discussion, groups of bar magnets were used to illustrate the way bits are organized. However, in real hard disk media, the bits are stored in a collection of nano-magnets called grains for several reasons. Grains (tiny crystallites) are regions of a material within which atoms are in arrangement of a crystal. The grains of current hard disk media have an average size of about 7 nm. Figure 9 shows a close-up view of the bits in a recording medium (bit-boundary is shown by the dark lines). It can be noticed that the bits are not perfectly rectangular. The bit boundaries are not perfectly straight lines, but they are mostly determined by the grain boundaries in modern-day magnetic disks. Also, several grains are between the bit boundaries. In current technology, about 60 grains are included in a bit.

The reasons for using several grains to store information derive mainly from the fabrication process. The traditional deposition methods of making the magnetic disks (such as sputtering) lead to a polycrystalline thin film. If the grains of the polycrystalline material are magnetic, their easy

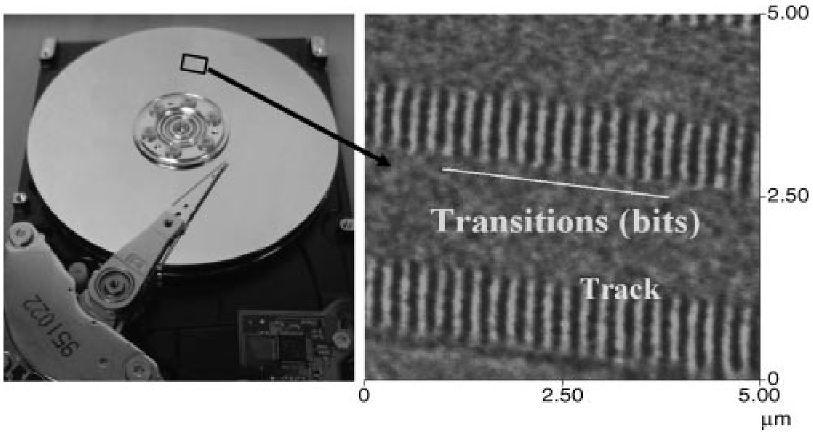


Figure 6. Organization of data in HDDs.

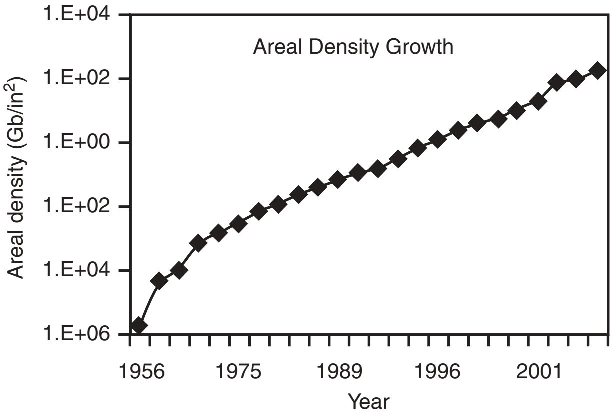


Figure 7. Trend in the increase of areal density of HDDs.

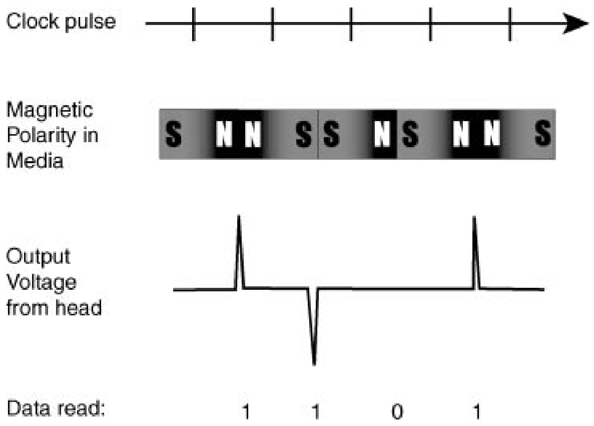


Figure 8. Recording principle of longitudinal recording.

axes (the direction in which north and south poles will naturally align themselves) will be in random directions. Moreover, the grains are also arranged in random positions within the magnetic media so that relying on one grain to store one bit is not possible. In addition, relying on several grains to store information provides us with the convenience of storing information in any part of the recording media.

It can be noticed from Fig. 10 that the easy axes are pointing in different directions. The magnetic field, which produces the signal in the read-sensor, depends on the component of magnetization parallel to the track. If more grains have their magnetization orientated parallel to the track, then the recording media will exhibit a high signal and a low noise. When designing a recording medium, it is important to achieve a high signal-to-noise ratio (SNR) at high densities, as a high SNR will enable the bits to be read reliably. Hard disk media makes use of a mechanical-texturing technology to maximize the number of grains whose easy axes are parallel to the track. However, even by improving this technology, not all grains can be arranged when their easy axis direction is parallel to the track. Because of these reasons, several grains of a magnetic disk are used to store one bit.

Design of a Magnetic Recording Medium

In the previous section, it was highlighted that several grains (tiny crystals) are used for storing information. The SNR, which determines how reliably the bits can be read, depends on the number of grains in a bit. When the areal density needs to be increased in an HDD, the bit size needs to be shrunk. Therefore, to maintain an acceptable SNR, the grain size also needs to be shrunk, so that the number of grains per bit remains more or less the same. Therefore, reduction of grain size is one of the areas, where the attention of researchers is always focused (9,10). Figure 11 shows the reduction of grain size in recording media over several generations. As the technology progressed, the grain size decreased. However, significant improvements in other parts of HDDs can also relax the requirement on grain size. Although the hard disk media of a decade ago had about 600 grains per bit, current media have only about 60 grains per bit. This decrease shows that, along with improvements in recording media technologies, the other components (such as head) or technologies (such as signal processing) have also progressed significantly enough to read a bit reliably from just 60 grains.

It is also essential to achieve sharper bit boundaries to pack bits closer together. The sharpness of the bit boundaries deteriorates in the case of longitudinal recording because identical magnetic poles facing at a bit boundary do not favor a sharp magnetic change at the boundary. On the other hand, in the case of perpendicular recording, which will be described in detail later, where the magnetization is formed in the perpendicular direction to the disk plane, opposite magnetic poles facing at a bit boundary help to from a sharp magnetic change. Therefore, the sharpness

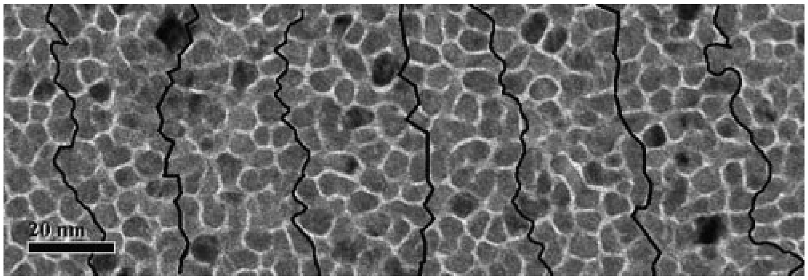


Figure 9. Close-up view of bits in a recording medium.

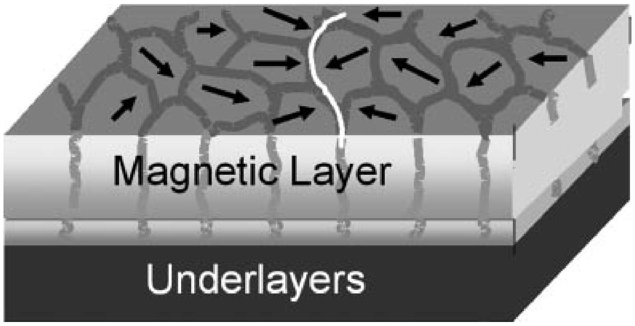


Figure 10. Magnetic orientations of grains in a recorded bit.

of the bit boundaries is determined by the nature of grains in the recording medium.

If the grains are isolated from each other, each grain will act as magnet by itself. In this case, the bit boundaries will be sharper. On the other hand, if the grains are not well isolated, a few grains could switch together. In this case, the bit boundaries will be broader. It is essential to achieve a good inter-grain separation, when the recording medium is designed. In the emerging perpendicular recording, the hard disk media are an alloy of CoCrPt:SiO_2. Significant presence of Co makes this alloy magnetic, and Pt helps to improve the magneto-crystalline anisotropy energy. If this energy is greater, the reversal of magnetic direction becomes difficult, leading to a long-term storage without self-erasure or erasure from small external fields. The addition of Cr and SiO_2 are responsible for improving inter-grain separation. When the magnetic layer is formed from CoCrPt:SiO_2, oxides of Cr and Si are formed in the grain boundary. Because these oxides are nonmagnetic, the magnetic grains are isolated from each other, and this helps to obtain a narrower bit-boundary.

It is also important to achieve a desired crystallographic orientation when a recording medium is designed. The cobalt crystal, which is the most commonly used hard disk media layer, has a hexagonal close-packed structure. In a cobalt crystal, the north and south poles prefer to lie along the C-axis (center of the hexagon). Therefore, when a recording medium is made, it is necessary to orient the C-axis in a desired way. For example, it is necessary to orient the C-axis perpendicular to the disk in the emerging perpendicular recording technology (and parallel to the disk in longitudinal recording technology). If the recording medium is deposited directly on the substrate, it may not be possible to achieve the desired orientation. Therefore, usually several layers are deposited below the recording layer (see Fig. 3) to improve the desired orientation. In recording media design, the choice of materials and the process conditions for the layers play a critical role.

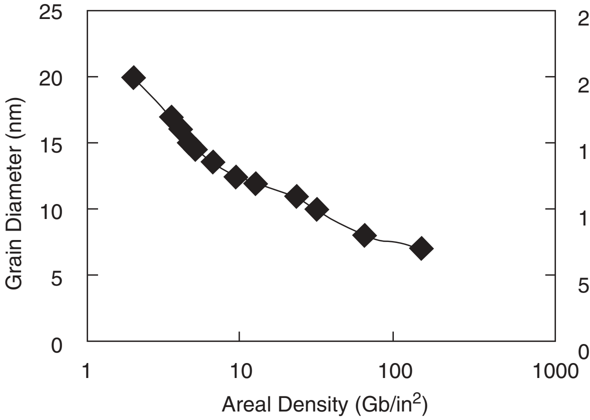


Figure 11. Average size of grains in recording media of different generations.

In addition, several requirements such as corrosion-resistance and a smooth surface must be met when designing recording media. The designed media must not corrode at least for a period of 10 years. The protection layers that have to prevent corrosion cannot be thicker than 4 nm (as of the year 2007). In the future, the protection layers need to be thinned down closer to 1 nm (1/100000th thickness of a human hair). The surface of the media should be very smooth with a roughness of only about 0.2 nm to enable smooth flying of the head over the media surface without crashing. All requirements must be met while maintaining the cost of the hard disk media below about US$5.

OTHER COMPONENTS

The HDD involves several other advanced technologies in its components and mechanisms. For example, the slider that carries the read/write head has to fly at a height of about 7 nm (or lower in future). This flying condition is achieved when the disk below rotates at a linear velocity of 36 kmph. If a comparison is made between the slider and a B747 jet, flying the slider at a height of 7 nm is like flying the jumbo jet at a height of 7 mm above the ground. In some modern HDDs, the flying height of the slider can be lowered by about 2 nm when information needs to be read or written. This decrease will allow the head to fly at safer heights during the idle state and fly closer when needed.

Several challenges are associated with almost all components of an HDD. Because the head needs to fly at a very low height over the media surface, the hard disk needs to be in a very clean environment. Therefore, HDDs are sealed with a rubber gasket and protected from the outside environment. The components inside the HDD (including the screws) should not emit any contaminant particle or gas, as that will contaminate the environment leading to a crash between the head and the media. The motor of the HDDs should provide sufficient torque for the heavy platters to rotate. Yet the motor should be quiet, stable in speed, and free from vibrations. In addition, it should be thin enough to fit into smaller HDDs and consume less power.

OTHER TECHNOLOGIES

HDDs also use several advanced technologies to achieve an ever increasing demand of higher capacity and lower cost per gigabyte. For an Olympic sprinter who is sprinting at a speed of 10 m/s, staying in his track with a width of about 1.25 m is not a serious problem. However, in HDDs, the head that moves at similar speeds has to stay in its track, which is only about 160 nm wide (about 8 million times narrower than the Olympic track). Following the track gets more complicated in the presence of airflow. The airflow inside the HDD causes vibrations in the actuator and the suspension that move and carry the head. Therefore, it is necessary to study the air-flow mechanism and take the necessary steps to minimize the vibrations and stay in the track. These issues will be more challenging in the future, as the tracks will be packed at even higher densities than before. It was also mentioned that the HDDs use fewer grains per bit now than in the recent past, which leads to a reduction in the SNR and makes signal processing more complicated. Therefore, it is very clear that the HDD involves creation of new technologies that should go hand-in-hand. HDD technology is one of the few technologies that needs multidisciplinary research.

RECENT DEVELOPMENTS

In magnetic recording, as mentioned, small grains help to store the information. The energy that helps to make the information stable depends on the volume of the grain. When the grains become smaller (as technology progresses), this energy becomes smaller, which leads to a problem called superparamagnetism. In superparamagnetism, the north and south poles of a magnet fluctuate from thermal agitation, without the application of any external field. Even if 5% of the grains of a bit are superparamagnetic, data will be lost. It was once reported that the areal density of 35 Gb/in^2 would be the limit of magnetic recording (11). However, HDD technology has surpassed this limit, and the current areal density of the products is at least four times larger. Several techniques are employed to overcome the limit of 35 Gb/in^2, but describing those techniques is beyond the scope of this article. One technology, which can carry forward the recording densities to higher values, at least as high as 500 Gb/in^2 (which is roughly three times the areal density in the products of 2006), is called perpendicular recording technology.

Although perpendicular recording technology was proposed in the mid-1970s, successful implementation of perpendicular recording technology has been offered only since 2006. In this technology, the north and south poles of the magnetic grains that store the information lie perpendicular to the plane of the disk (12). In addition, this technology has two main differences in the head and media design. First, the media design includes the presence of a soft magnetic underlayer (a material whose magnetic polarity can be changed with a small field)— marked in Fig. 3 as SUL1 and SUL2. Second, the head has a different architecture in that it produces a strong perpendicular magnetic field that goes through the recording layer and returns back through another pole with a weaker field via the SUL (see Fig. 12). The field becomes weaker at the return pole because of the difference in the geometry of the two poles. This configuration achieves a higher writing field than what is possible with the prevalent longitudinal recording technology. When higher writing fields are possible, it is possible to make use of smaller grains (whose magnetic polarities are more stable) to store information. When smaller grains are used, the bit sizes can be smaller leading to higher areal densities. This increase in the writability is one reason that gives the edge to perpendicular recording technology.

Perpendicular recording technology offers larger M_r.t (which is the product of remanent moment M_r and thickness t or the magnetic moment per unit area) values for the media without degrading the recording resolution. The resolution and the noise are governed by the size of the magnetic clusters (or the grain size in the extreme case). It also offers increased thermal stability at high linear densities. All these characters come from the difference in the demagnetizing effect at the magnetic transitions.

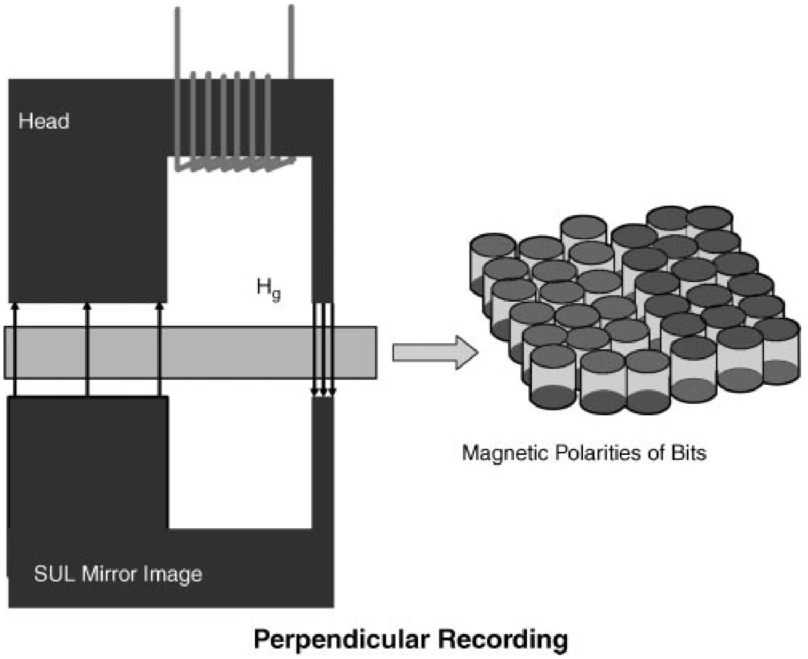


Figure 12. Illustration of perpendicular recording technology.

FUTURE STORAGE: WHAT IS IN STORE?

The recently introduced perpendicular recording technology is expected to continue for the next four to five years without significant hurdles. However, some new technologies would be needed after five years or so, to keep increasing the areal density of the HDDs. One such technology that is investigated intensively is heat-assisted magnetic recording (HAMR). In this technology, a fine spot (about 30 nm) of the recording medium is heated to enable the writing operation. Without heating, writing/erasure cannot be achieved. Once the writing is carried out, the media are cooled to room temperature, at which the data are thermally stable.

Several challenges are associated with this technology, which are being investigated. Another technology that is considered as a competitor to HAMR is bit-patterned media recording. In bit-patterned media recording technology, the media are (lithographically or otherwise) defined to have regions of magnetic and nonmagnetic material. Although the magnetic material will store the information, the nonmagnetic material will define the boundary and help to reduce the inter-bit magnetic interactions and, hence, noise. Since the magnetic material in this technology is larger than that of a single grain in perpendicular recording technology, the information is more stable with bit-patterned media technology. However, to be competitive, feature sizes of about 10 nm need to be achieved lithographically without significantly increasing costs. Although a 10-nm feature size over larger areas at a cheaper costs is a challenge that cannot be tackled so soon, challenges also come from other components to meet the requirement of bit-patterned media recording technology. At this point, it is not clear which technology will take the lead. But, it is foreseen widely that far-future HDDs may combine both technologies. Therefore, research has been going on for both of these technologies.

When the new technologies described above enter into HDDs, in about six to seven years from now, the areal density will be about 1000 Gb/in^2. With this kind of areal densities, a desktop HDD could store about 5 TB or higher. You could carry your laptop with an HDD that stores 1 TB. It is almost certain that all of these high-capacity devices would be available at the same price of a 400-GB desktop drive available now.

SUMMARY

The working principle of an HDD is introduced, with detailed attention given to the magnetic disks that store the information. In an HDD, the head that has a sensor and a writer are used for reading and writing information. The information is stored in magnetic disks. To achieve high densities, it is necessary to reduce the size of the grains (tiny-magnets) that store the information. Recently, perpendicular recording technology has been introduced in the market to continue the growth of HDD technology. Although perpendicular recording technology may last for five to eight more years, alternative technologies are being sought. Heat-assisted magnetic recording and bit-patterned media recording are a few candidates for future.

BIBLIOGRAPHY

1. Available: http://www.phptr.com/content/images/0130130559/samplechapter/0130130559.pdf.
2. Available: http://en.wikipedia.org/wiki/Hard_disk_drive.
3. Available: http: / / computer . howstuffworks . com / hard-disk1.htm.
4. A. Moser, K. Takano, D.T. Margulies, M. Albrecht, Y. Sonobe, Y. Ikeda, S. Sun and E.E. Fullerton, *J. Phys. D: Appl. Phys*. **35**. R157–R167, 2002.
5. D. Weller and M.F. Doerner, *Annu. Rev. Mater. Sci.* **30**: 611–644, 2000.
6. S.N. Piramanayagam, *J. Appl. Phys.,* **102**: 011301, 2007.
7. R. Sbiaa and S.N. Piramanayagam, Recent patents on nanotechnology **1**: 29–40, 2007.
8. Available http://www.oerlikon.com.
9. S.N. Piramanayagam, et al., *Appl. Phys. Lett.* **89**: 162504, 2006.
10. M. Zheng, et al., *IEEE Trans. Magn.* **40** (4): 2498–2500, 2004.
11. S.H. Charap, P.L. Lu, and Y.J. He, *IEEE Trans. Magn.*, **978**: 33, 978, 1997.
12. S. Iwasaki, *IEEE Trans. Magn.*, **20**: 657, 1984.

S. N. Piramanayagam
Data Storage Institute
Singapore

ELECTRONIC CALCULATORS

People have used calculating devices of one type or another throughout history. The abacus, which uses beads to keep track of numbers, was invented over 2000 years ago and is still used today. Blaise Pascal invented a "numerical wheel calculator," a brass box with dials for performing addition, in the seventeenth century. (1) Gottfried Wilhelm von Leibniz soon created a version that could also multiply, but mechanical calculators were not widely used until the early nineteenth century, when Charles Xavier Thomas de Colmar invented a machine that could perform the four basic functions of addition, subtraction, multiplication, and division. Charles Babbage proposed a steam-powered calculating machine around 1822, which included many of the basic concepts of modern computers, but it was never built. A mechanical device that used punched cards to store data was invented in 1889 by Herman Hollerith and then used to mechanically compile the results of the U.S. Census in only 6 weeks instead of 10 years. A bulky mechanical calculator, with gears and shafts, was developed by Vannevar Bush in 1931 for solving differential equations (2).

The first electronic computers used technology based on vacuum tubes, resistors, and soldered joints, and thus they were much too large for use in portable devices. The invention of the transistor (replacing vacuum tubes) followed by the invention of the integrated circuit by Jack Kilby in 1958 led to the shrinking of electronic machinery to the point where it became possible to put simple electronic computer functionality into a package small enough to fit into a hand or a pocket.

Logarithms, developed by John Napier around 1600, can be used to solve multiplication and division problems with the simpler operations of addition and subtraction. Slide rules are mechanical, analog devices that are based on the idea of logarithms and use calibrated sticks or disks to perform multiplication and division to three or four significant figures. Slide rules were an indispensable tool for engineers until they were replaced by handheld scientific calculators starting in the early 1970s (3).

CALCULATOR TYPES AND USES

Electronic calculators come in a variety of types: four-function (addition, subtraction, multiplication, and division), desktop, printing, and scientific. Figure 1 shows various calculators with prices ranging from \$5 to \$150. Scientific calculators can calculate square roots, logarithms and exponents, and trigonometric functions. The scientific category includes business calculators, which have time-value-of-money, amortization, and other money management functions. Graphing calculators are a type of scientific calculator with a display that can show function plots. Advanced scientific and graphing calculators also have user programming capability that allows the user to enter and store programs. These programs can record and automate calculation steps, be used to customize the calculator, or perform complicated or tedious algorithms. Some handheld calculators are solar powered, but most advanced scientific calculators are powered by batteries that last for many months without needing replacement.

Scientific Calculators

Scientific calculators can perform trigonometric functions and inverse trigonometric functions (sin x, cos x, tan x, arcsin x, arccos x, arctan x) as well as hyperbolic and inverse hyperbolic functions (sinh x, cosh x, tanh x, arcsinh x, arccosh x, arctanh x). They can also find natural and common logarithms (ln x, log x), exponential functions ($e^x, y^x, y^{1/x}$), factorials ($n!$), and reciprocals ($1/x$). Scientific calculators contain a representation for the constant π, and they can convert angles between degrees and radians. Most scientific calculators accept numbers with 10 to 12 digits and exponents ranging from -99 to 99, although some allow exponents from -499 to 499.

Graphing Calculators

Graphing calculators were first developed in the late 1980s as larger liquid-crystal displays (LCDs) became available at lower cost. The pixels in an LCD display can be darkened individually and so can be used to plot function graphs. The user keys in a real-valued function of the form $y = f(x)$ and makes some choices about the scale to use for the plot and the set of values for x. Then the calculator evaluates $f(x)$ for each x value specified and displays the resulting $(x,f(x))$ pairs as a function graph. Graphing calculators can also plot polar and parametric functions, three-dimensional (3-D) wireframe plots, differential equations, and statistics graphs such as scatter plots, histograms, and box-and-whisker plots. (See Fig. 2.) Once a graph has been displayed, the user can move a small cursor or crosshairs around the display by pressing the arrow or cursor keys and then obtain information about the graph, such as the coordinates of points, the x-intercepts, or the slope of the graph at a certain point. The user can also select an area of interest to zoom in on, and the calculator will replot the graph using a different scale (4).

Programmable Calculators

If a series of steps is to be repeated using various different inputs, it is convenient to be able to record those steps and replay them automatically. Simple programmable calculators allow the user to store a sequence of keystrokes as a program. More complicated programmable calculators provide programming languages with many of the components of high-level computer languages, such as branching and subroutines.

Given all these types and uses of calculators, what is it that defines a calculator? The basic paradigm of a calculator is key per function. For example, one key is dedicated to the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Figure 1.

square root function on most scientific calculators. All the user has to do is input a number and then press one key, and the calculator performs a complicated series of steps to obtain an answer that a person could not easily calculate on their own. Another way to say this is that there is an asymmetry of information flow: Given a small amount of input, the calculator does something nontrivial and gives you back results that you could not have easily found in your head or with pencil and paper.

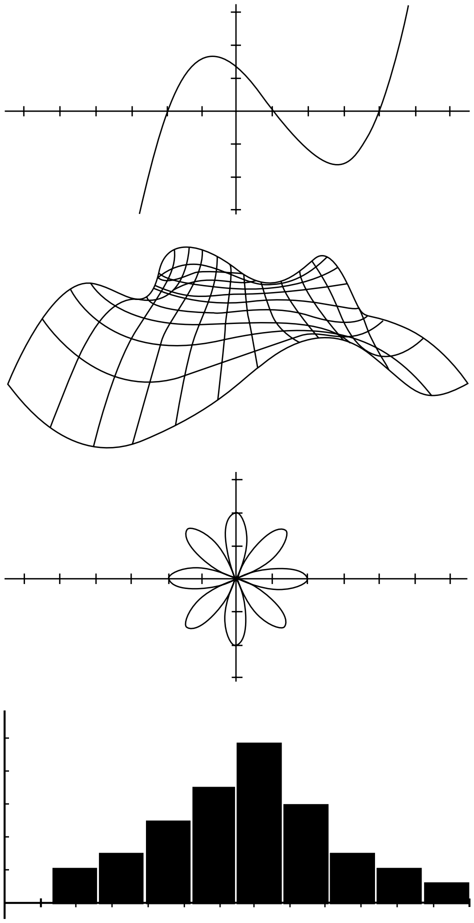


Figure 2.

CALCULATOR HARDWARE COMPONENTS

Today's advanced scientific and graphing calculators have many similarities to computers. The block diagram in Fig. 3 shows the system architecture of an advanced scientific graphing calculator (5). The two main components of a calculator are hardware and software. The hardware includes plastic and metal packaging, display, keypad, optional additional input/output devices (such as infrared, serial ports, card slots, and beeper parts to produce sound), power supply circuit, and an electronic subsystem. The electronic subsystem consists of a printed circuit board with attached electronic components and integrated circuits, including a central processing unit (CPU), display controllers, random access memory (RAM), and the read-only memory (ROM) where software programs are stored permanently.

The mechanical design of a calculator consists of subassemblies such as a top case with display and keypad, a bottom case, and a printed circuit or logic assembly. Figure 4 shows the subassemblies of a graphing calculator. A metal chassis in the top case supports the keypad, protects and frames the glass display, and provides a negative battery contact. The metal chassis is also part of the shielding, which protects the electronic circuitry from electrostatic discharge (ESD). The bottom case may contain additional metal shielding, a piezoelectric beeper part, and circuitry for battery power. The subassemblies are connected electrically with flexible circuits (6).

Display

Early calculators used light-emitting diode (LED) displays, but liquid crystal displays (LCDs) are used in most modern

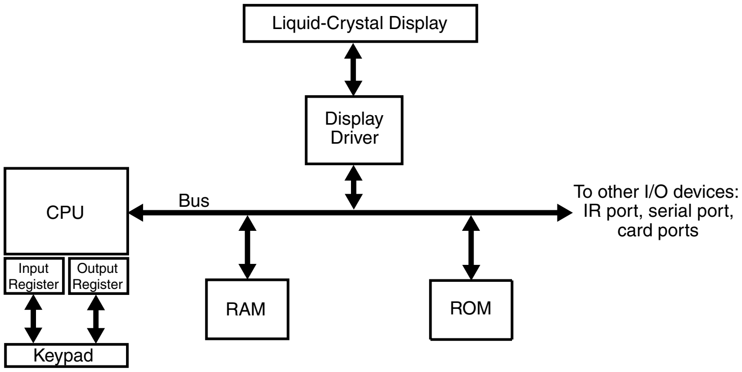


Figure 3.

calculators because they have low voltage requirements, good visibility in high ambient light conditions, and they can produce a variety of character shapes and sizes (7). An LCD consists of two pieces of glass with a layer of liquid crystal in between which will darken in specific areas when a voltage signal is applied. These areas can either be relatively large segments that are combined a few at a time to represent a number or character, or they can be a grid of very small rectangles (also called picture elements or pixels) that can be darkened selectively to produce characters, numbers, and more detailed graphics. Basic calculators have one line displays that show one row of numbers at a time, whereas today's more advanced calculators can display up to eight or more rows of characters with 22 or more characters per row, using a display with as many as 100 rows and 160 columns of pixels.

Keypad

Calculator keypads are made up of the keys that the user presses, an underlying mechanism that allows the keys to be depressed and then to return to their initial state, and circuit traces that allow the system to detect a key press. When a key is pressed, an input register line and an output register line make contact, which causes an interrupt to be generated. This interrupt is a signal to the software to scan the keyboard to see which key is pressed. Keypads have different tactile feel, depending on how they are designed and of what materials they are made. Hinged plastic keys and dome-shaped underlying pads are used to provide feedback to the user with a snap when keys are pressed. An elastomer membrane separating the keys from the underlying contacts helps to protect the electronic system from dust (8).

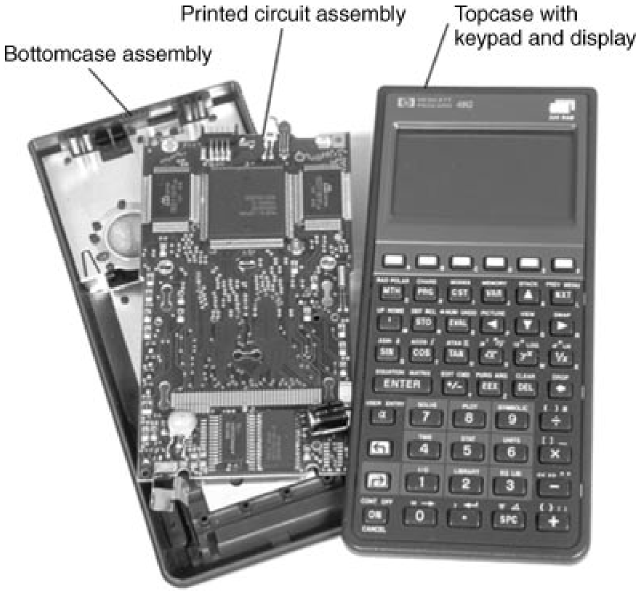


Figure 4.

The keypad is an input device, because it is a way for the user to provide information to the calculator. The display is an output device, because it allows the calculator to convey information to the user. Early calculators, and today's simple calculators, make do with only these input and output devices. But as more and more memory has been added to calculators, allowing for more data storage and for more extensive programs to be stored in calculator memory, the keypad and display have become bottlenecks. Different ways to store and input data and programs have been developed to alleviate these input bottlenecks. Infrared and serial cable ports allow some calculators to communicate with computers and other calculators to quickly and easily transfer data and programs.

Circuits

The electronic components of a calculator form a circuit that includes small connecting wires, which allow electric current to flow to all parts of the system. The system is made up of diodes; transistors; passive components such as resistors, capacitors, and inductors; as well as conventional circuits and integrated circuits that are designed to perform certain tasks. One of these specialized circuits is an oscillator that serves as a clock and is used to control the movement of bits of information through the system. Another type of specialized circuit is a logic circuit, or processor, which stores data in registers and performs manipulations on data such as addition.

Printed Circuit Assembly

A printed circuit board (PCB) forms the backbone of a calculator's electronic circuit system, allowing various

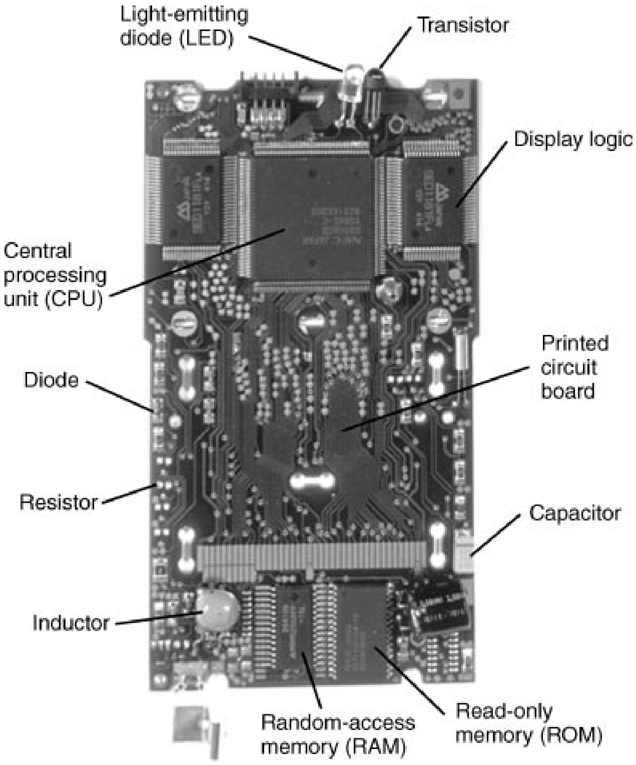


Figure 5.

components to be attached and connected to each other (9). Figure 5 shows a calculator printed circuit assembly with many electronic components labeled. Wires that transmit data and instructions among the logic circuits, the memory circuits, and the other components are called buses. Various integrated circuits may be used in a calculator, including a central processing unit (CPU), RAM, ROM, and Flash ROM memory circuits, memory controllers that allow the CPU to access the memory circuits, a controller for the display, quartz-crystal-controlled clocks, and controllers for optional additional input/output devices such as serial cable connectors and infrared transmitters and receivers. Depending on the design of the calculator and the choice of components, some of these pieces may be incorporated in a single integrated circuit called an application-specific integrated circuit (ASIC).

Central Processing Unit

The central processing unit (CPU) of a calculator or computer is a complicated integrated circuit consisting of three parts: the arithmetic-logic unit, the control unit, and the main storage unit. The arithmetic-logic unit (ALU) carries out the arithmetic operations of addition, subtraction, multiplication, and division and makes logical comparisons of numbers. The control unit receives program instructions, then sends control signals to different parts of the system, and can jump to a different part of a program under special circumstances such as an arithmetic overflow. The main storage unit stores data and instructions that are being used by the ALU or control unit.

Many calculators use custom microprocessors because commercially available microprocessors that have been designed for larger computers do not take into account the requirements of a small, handheld device. Calculator microprocessors must operate well under low power conditions, should not require too many support chips, and generally must work with a smaller system bus. This is because wider buses use more power and require additional integrated circuit pins, which increases part costs. Complementary metal-oxide semiconductor or CMOS technology is used for many calculator integrated circuits because it is well suited to very low power systems (10). CMOS has very low power dissipation and can retain data even with drastically reduced operating voltage. CMOS is also highly reliable and has good latch up and ESD protection.

Memory

RAM integrated circuits are made up of capacitors, which represent bits of information. Each bit may be in one of two possible states, depending on whether the capacitor is holding an electric charge. Any bit of information in RAM can easily be changed, but the information is only retained as long as power is supplied to the integrated circuit. In continuous memory calculators, the information in RAM is retained even when the calculator is turned OFF, because a small amount of power is still being supplied by the system. The RAM circuits used in calculators have very low standby current requirements and can retain their information for short periods of time without power, such as when batteries are being replaced.

ROM circuits contain information that cannot be changed once it is encoded. Calculator software is stored in ROM because it costs less and has lower power requirements than RAM. As software is encoded on ROM by the manufacturer and cannot be changed, the built-in software in calculators is often called firmware. When calculator firmware is operating, it must make use of some amount of RAM whenever values need to be recorded in memory. The RAM serves as a scratch pad for keeping track of inputs from the user, results of intermediate calculations, and final results. The firmware in most advanced scientific calculators consists of an operating system (which is a control center for coordinating the low-level input and output, memory access, and other system functions), user interface code, and the mathematical functions and other applications in which the user is directly interested.

Some calculators use Flash ROM instead of ROM to store the programs provided by the calculator manufacturer. This type of memory is more expensive than ROM, but it can be erased and reprogrammed, allowing the calculator software to be updated after the calculator has been shipped to the customer. Flash ROM does not serve as a replacement for RAM, because Flash ROM can only be reprogrammed a limited number of times (approximately tens of thousands of times), whereas RAM can accommodate an effectively unlimited number of changes in the value of a variable that may occur while a program is running.

OPERATING SYSTEM

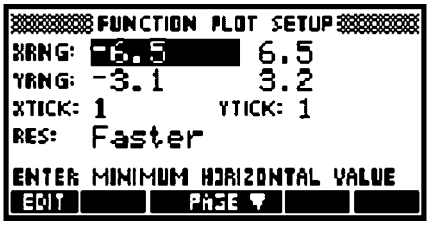
The operation of a calculator can be broken down into three basic steps of input, processing, and output. For example, to find the square root of a number, the user first uses the keypad to enter a number and choose the function to be computed. This input generates electronic signals that are processed by the calculator's electronic circuits to produce a result. The result is then communicated back to the user via the display. The processing step involves storing data using memory circuits and making changes to that data using logic circuits, as well as the general operation of the system, which is accomplished using control circuits.

A calculator performs many tasks at the system level, of which the user is not normally aware. These tasks include those that go along with turning the calculator ON or OFF, keeping track of how memory is being used, managing the power system, and all the overhead associated with getting input from the keypad, performing calculations, and displaying results. A calculator's operations are controlled by an operating system, which is a software program that provides access to the hardware computing resources and allows various application software programs to be run on the computer (or calculator).

The operating system deals with memory organization, data structures, and resource allocation. The resources that it is controlling include CPU time, memory space, and input/output devices such as the keypad and the display. When an application program is executed, the operating system is responsible for running the program by scheduling slices of CPU time that can be used for executing the program steps, and for overseeing handling of any interrupts that may occur while the program is executing. Interrupts are triggered by events that need to be dealt with in a timely fashion, such as key presses, requests from a program for a system-level service such as refreshing the display, or program errors. Some types of errors that may occur when a program is running are low power conditions, low memory conditions, arithmetic overflow, and illegal memory references. These conditions should be handled gracefully, with appropriate information given to the user. Operating systems provide convenience and efficiency: they make it convenient to execute application programs, and they manage system resources to get efficient performance from the computer or calculator (11).

USER INTERFACE

The user interface for one-line-display calculators is very simple, consisting of a single number shown in the display. The user may have some choice about the format of that number, such as how many digits to display to the right of the decimal point, or whether the number should be shown using scientific notation. Error messages can be shown by spelling out short words in the display. For more complicated calculators than the simple four-function calculators, the number of keys on the keypad may not be enough to use one per operation that the calculator can perform. Then it becomes necessary to provide a more extensive user interface than just a simple keypad. One way to increase the number of operations that the keypad can control is to add shifted keys. For example, one key may have the square-root symbol on the key and the symbol x^2 printed just above the key, usually in a second color. If the user presses the square-root key, the square-root function is performed. But if the user first presses the shift key and then presses the square-root key, the x-squared function will be performed instead.



FUNCTION PLOT SETUP

XRNG: -6.5 6.5

YRNG: -3.1 3.2

XTICK: 1 YTICK: 1

RES: Faster

ENTER MINIMUM HORIZONTAL VALUE

Figure 6.

Advanced scientific and graphing calculators provide systems of menus that let the user select operations. These menus may appear as lists of items in the display that the user can scroll through using arrow or cursor keys and then select by pressing the Enter key. Changeable labels in the bottom portion of the display, which correspond to the top row of keys, can also be used to display menu choices. These are called soft keys, and they are much like the function keys on a computer. Methods for the user to enter information into the calculator depend on the type of calculator. On simple, one-line-display calculators, the user presses number keys and can see the corresponding number in the display. Graphing calculators, with their larger displays, can prompt the user for input and then display the input using dialog boxes like the ones used on computers (12). Figure 6 shows a graphing calculator dialog box used to specify the plot scale.

NUMBERS AND ARITHMETIC

The most basic level of functionality that is apparent to the calculator user is the arithmetic functions: addition, subtraction, multiplication, and division. All calculators perform these functions, and some calculators are limited to these four functions. Calculators perform arithmetic using the same types of circuits that computers use. Special circuits based on Boolean logic are used to combine numbers, deal with carries and overdrafts, and find sums and differences. Various methods have been developed to perform efficient multiplication and division with electronic circuits (13).

Binary Numbers

Circuits can be used to represent zeros or ones because they can take on two different states (such as ON or OFF). Calculator (and computer memory) at any given time can be thought of as simply a large collection of zeros and ones. Zeros and ones also make up the binary or base-2 number system. For example, the (base-10) numbers 1, 2, 3, 4 are written in base-2 as 1, 10, 11, 100, respectively. Each memory circuit that can be used to represent a zero or one is called a binary digit or bit. A collection of eight bits is called a byte (or a word). Some calculator systems deal with four bits at a time, called nibbles. If simple binary numbers were used to represent all numbers that could possibly be entered into a calculator, many bits of memory would be needed to represent large numbers. For example, the decimal number 2^n is represented by the binary number consisting of a 1 followed by n zeros, and so requires $n + 1$ bits of memory storage. To be able to represent very large numbers with a fixed number of bits, and to optimize arithmetic operations for the design of the calculator, floating-point numbers are used in calculators and computers.

Floating-Point Numbers

Floating-point numbers are numbers in which the location of the decimal point may move so that only a limited number of digits are required to represent large or small numbers, which eliminates leading or trailing zeros, but its main advantage for calculators and computers is that it greatly increases the range of numbers that can be represented using a fixed number of bits. For example, a number x may be represented as $x = (-1)^s \times F \times b^E$, where s is the sign, F is the significand or fraction, b is the base used in the floating-point hardware, and E is a signed exponent. A fixed number of bits are then used to represent each number inside the calculator. The registers in a CPU, which is designed for efficient floating-point operations, have three fields that correspond to the sign, significand, and exponent, and can be manipulated separately.

Two types of errors can appear when a calculator returns an answer. One type is avoidable and is caused by inadequate algorithms. The other type is unavoidable and is the result of using finite approximations for infinite objects. For example, the infinitely repeating decimal representation for 2/3 is displayed as .6666666667 on a 10-decimal-place calculator. A system called binary-coded decimal (or BCD) is used on some calculators and computers as a way to deal with rounding. Each decimal digit, 0, 1, 2, 3, . . .,9, is represented by its four-bit binary equivalent: 0000, 0001, 0010, 0011,. . .,1001. So rather than convert each base-10 number into the equivalent base-2 number, the individual digits of the base-10 number are each represented with zeros and ones. When arithmetic is performed using BCD numbers, the methods for carrying and rounding follow base-10 conventions.

One way to improve results that are subject to rounding errors is to use extra digits for keeping track of intermediate results, and then do one rounding before the result is returned using the smaller number of digits that the user sees. For example, some advanced scientific calculators allow the user to input numbers using up to 12 decimal places, and return results in this same format, but 15-digit numbers are actually used during calculation.

Reverse Polish Notation and Algebraic Logic System

The Polish logician Jan Lukasiewicz demonstrated a way of writing mathematical expressions unambiguously without using parentheses in 1951. For example, given the expression $(2 + 3) \times (7 - 1)$, each operator can be written before the corresponding operands: $\times + 2\ 3 - 7\ 1$. Or each operator can be written after its operands: $2\ 3 + 7\ 1 - \times$. The latter method has come to be known as reverse polish notation (RPN) (14). Arithmetic expressions are converted to RPN before they are processed by computers because RPN simplifies the evaluation of expressions. In a non-RPN expression containing parentheses, some operators cannot be applied until after parenthesized subexpressions are first evaluated. Reading from left to right in an RPN expression, every time an operator is encountered it can be applied immediately, which means there is less memory and bookkeeping required to evaluate RPN expressions. Some calculators allow users to input expressions using RPN. This saves the calculator the step of converting the expression to RPN before processing it. It also means less keystrokes for the user bacause parentheses are never needed with RPN. Algebraic logic system (ALS) calculators require numbers and operators to be entered in the order they would appear in an algebraic expression. Parentheses are used to delimit subexpressions in ALS.

User Memory

On many calculators, the user can store numbers in special memory locations or storage register and then perform arithmetic operations on the stored values. This process is called register arithmetic. On RPN calculators, memory locations are arranged in a structure called a stack. For each operation that is performed, the operands are taken from the stack and then the result is returned to the stack. Each time a new number is placed on the stack, the previous items that were on the stack are each advanced one level to make room for the new item. Whenever an item is removed from the stack, the remaining items shift back. A stack is a data structure that is similar to a stack of cafeteria trays, where clean trays are added to the top, and as trays are needed, they are removed from the top of the stack. This scheme for placing and removing items is called last-in–first-out or LIFO.

ALGORITHMS

An algorithm is a precise, finite set of steps that describes a method for a computer (or calculator) to solve a particular problem. Many computer algorithms are designed with knowledge of the underlying hardware resources in mind, so that they can optimize the performance of the computer. Numerical algorithms for calculators take into account the way that numbers are represented in the calculator.

Square-Root Algorithm

A simple approximation method is used by calculators to find square roots. The basic steps to finding $y = x^{1/2}$ are to first guess the value of y, calculate y^2, and then find $r = x - y^2$. Then if the magnitude of r is small enough return y as the answer. Otherwise, increase or decrease y (depending on whether r is positive or negative, respectively) and repeat the process. The number of intermediate calculations required can be reduced by avoiding finding y^2 and $x - y^2$ for each value of y. This can be done by first finding the value of the largest place digit of y, then the next largest place digit, and so on. For example, if calculating $54756^{1/2}$, first find 200, then 30, and then 4 to construct the answer $y = 234$. This method is similar to a method once taught in schools for finding square roots by hand(15).

Trigonometric Function Algorithms

The algorithms for computing trigonometric functions depend on using trigonometric identities and relationships to reduce arbitrarily difficult problems to more manageable problems. First, the input angle θ is converted to an angle in radians, which is between 0 and 2π (or in some calculators, between 0 and π/4). Next θ is expressed as a sum of smaller angles. These smaller angles are chosen to be angles whose tangents are powers of 10: $\tan^{-1}(1) = 45^\circ$, $\tan^{-1}(0.1)$, $\tan^{-1}(0.01)$, etc. A process called pseudodivision is used to express θ in this way: First $\tan^{-1}(1)$ is repeatedly subtracted from θ until an overdraft (or carry) occurs, then the angle being subtracted from is restored to the value it had right before the overdraft occurred, then the process is repeated by subtracting $\tan^{-1}(0.1)$ until an overdraft occurs, and so forth, until we are left with a remaining angle r, which is small enough for the required level of accuracy of the calculator. Then θ can be expressed as

$$\theta = q_0 \tan^{-1}(1) + q_1 \tan^{-1}(0.1) + \cdots + r \tag{1}$$

Vector geometry is the basis for the formulas used to compute the tangent of θ once it has been broken up into the sum of smaller angles. Starting with a vector with angle θ_1 and then rotating it counter-clockwise by an additional angle of θ_2, Fig. 7 illustrates the following relationships:

$$X_2 = X_1 \cos\theta_2 - Y_1 \sin\theta_2$$
$$Y_2 = Y_1 \cos\theta_2 + X_1 \sin\theta_2$$

Dividing both sides of these equations by cos θ_2 we obtain:

$$X_2/\cos\theta_2 = X_1 - Y_1 \tan\theta_2 = X_2' \tag{2}$$

$$Y_2/\cos\theta_2 = Y_1 + X_1 \tan\theta_2 = Y_2' \tag{3}$$

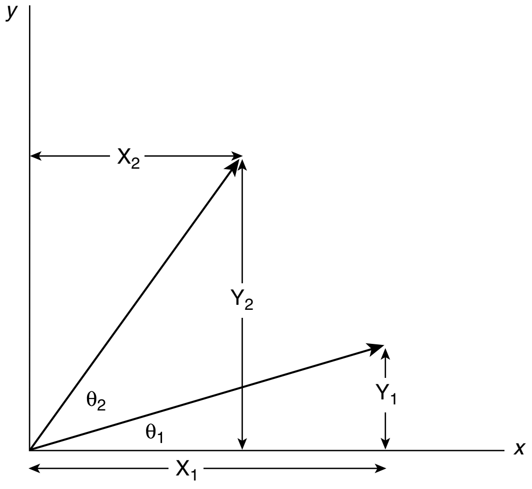


Figure 7.

As $Y_2/X_2 = \tan(\theta_1 + \theta_2)$, then by Equations 2 and 3, we can see that $Y_2'/X_2' = \tan(\theta_1 + \theta_2)$. Equations 2 and 3 can be used repeatedly to construct the tangent of θ, because θ has been broken down into a series of smaller angles, shown in Equation (1). The initial X_1 and Y_1 correspond to the small residual angle r. As r is a very small angle (in radians), $\sin(r)$ is close to r and $\cos(r)$ is close to 1, so if these values are close enough for our overall accuracy requirements, we can let Y_1 be r and X_1 be 1. Note Equations 2 and 3 involve finding tangents, but because we expressed θ as a sum of angles of the form $\tan^{-1}(10^{-k})$, $\tan(\tan^{-1}(10^{-k})) = 10^{-k}$, so each evaluation of Equations 2 or 3 will simply involve addition, subtraction, and multiplication by powers of 10. As the only multiplication involved is by powers of 10, the calculations can be accomplished more quickly and simply using a process called pseudomultiplication, which involves only addition and the shifting of contents of registers to simulate decimal point shifts that correspond to multiplication by powers of 10. The iterative process of using Equations 2 and 3 generates an X and Y, which are proportional to the sine and cosine of the original angle θ. Then elementary operations can be used to find the values of the various trigonometric functions for θ (16).

Logarithm Algorithms

Logarithms are found using a process similar to the approximation process used to compute trigonometric functions (17). It is a basic property of logarithms that $\ln(a_1 \times a_2 \times \ldots \times a_n) = \ln(a_1) + \ln(a_2) + \ldots + \ln(a_n)$. To find the logarithm of a number x, x is first expressed as product of factors whose logarithms are known. The number x will be stored in the calculator using scientific notation $x = M \times 10^k$, where M is called the mantissa and M is greater than or equal to 1 and less than 10. As $\ln = (M \times 10^k) = \ln(M) + k \times \ln(10)$, the problem of finding $\ln(x)$ is reduced to the problem of finding $\ln(M)$. Let a_j be numbers whose natural logarithms are known. Let $P = 1/M$. Then $-\ln(P) = \ln(M)$. Then express P as $P = P_n/r$, where $P_n = a_0^{k0} \times a_1^{k1} \times \ldots \times a_j^{kj}$ and r is a number close to 1. Note that $\ln(P) = \ln(P_n) - \ln(r)$, so now $\ln(M) = \ln(r) - \ln(P_n)$ and for r close to 1, $\ln(r)$ is close to 0. Also note that $M = 1/P = r/P_n$ implies that $M \times P_n = r$. So to find ln(M), we can first find P_n such that $M \times P_n$ is close to 1, where P_n is a product of specially chosen numbers a_j whose logarithms are known. To optimize this routine for a calculator's specialized microprocessor, values that give good results are $a_j = (1 + 10^{-j})$. Thus, for example, a_0, a_1, a_2, a_3, and a_4 would be 2, 1.1, 1.01, 1.001, and 1.0001. It turns out that M must first be divided by 10 to use these a_j choices. This choice of the a_j terms allows intermediate multiplications by each a_j to be accomplished by an efficient, simple shift of the digits in a register, similar to the pseudomultiplication used in the trigonometric algorithm.

CALCULATOR DESIGN CHOICES AND CHALLENGES

The requirements for a handheld calculator to be small, portable, inexpensive, and dedicated to performing computational tasks have driven many design choices. Custom integrted circuits and the CMOS process have been used because of low power requirements. Calculator software has been designed to use mostly ROM and very little RAM because of part cost and power constraints. Specialized algorithms have been developed and refined to be optimized for calculator CPUs. As calculators become more complicated, ease-of-use becomes an important design challenge. As memory becomes less expensive and calculators have more storage space, the keypad and display become bottlenecks when it comes to transferring large amounts of data. Improved input/output devices such as pen input, better displays, and character and voice recognition could all help to alleviate bottlenecks and make calculators easier to use.

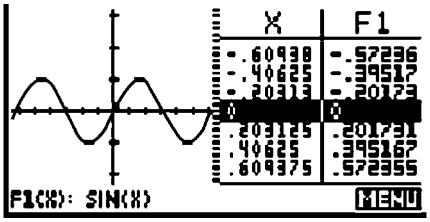
A desktop PC does not fit the needs of personal portability. Desktop or laptop PCs are not very convenient to use as a calculator for quick calculations. Also, a PC is a generic platform rather than a dedicated appliance. The user must take the time to start up an application to perform calculations on a PC so a PC does not have the back-of-the-envelope type of immediacy of a calculator. Handheld PCs and palmtop PCs also tend to be generic platforms, only in smaller packages. So they are as portable as calculators, but they still do not have dedicated calculating functionality. The user must go out of their way to select and run a calculator application on a handheld PC. The keypad of a handheld PC has a QWERTY keyboard layout, and so it does not have keys dedicated to calculator functions like sine, cosine, and logarithms. Handheld organizers and personal digital assistants (PDAs) are closer to the calculator model, because they are personal, portable, battery-operated electronic devices that are dedicated to particular functionality, but they currently emphasize organizer functionality rather than mathematics functionality.

COMMUNICATION CAPABILITY

Some calculators have already had communication capability for many years, using infrared as well as serial cable and other types of cable ports. These have allowed calculators to communicate with other calculators, computers, printers, and overhead display devices that allow an image of the calculator screen to be enlarged and projected for a roomful of people, data collection devices, bar code readers, external memory storage, and other peripheral devices. Protocols are standard formats for the exchange of electronic data that allow different types of devices to communicate with each other. For example, Kermit is a file transfer protocol developed at Columbia University. By coding this protocol into a calculator, the calculator can communicate with any of a number of different types of computers by running a Kermit program on the computer. By programming appropriate protocols into calculators, they could work with modems and gain access to the Internet. Calculations could then be performed remotely on more powerful computers, and answers could be sent back to the calculator. Or calculators could be used for the delivery of curriculum material and lessons over the Internet.

TECHNOLOGY IN EDUCATION

Curriculum materials have changed with the increased use of graphing calculators in mathematics and science classrooms. Many precalculus and calculus textbooks and science workbooks now contain exercises that incorporate the use of calculators, which allows the exercise to be more complicated than the types of problems that could be easily solved with pencil and paper in a few minutes. With the use of calculators, more realistic and thus more interesting and extensive problems can be used to teach mathematics and science concepts. Calculators are becoming a requirement in many mathematics classes and on some standardized tests, such as the Scholastic Aptitude Test taken by most U.S. high-school seniors who plan to attend college. Educational policy has, in turn, influenced the design of graphing calculators. In the United States, the National Council of Teachers of Mathematics promoted the use of the symbolic, graphic, and numeric views for teaching mathematics. These views are reflected in the design of graphing calculators, which have keys dedicated to entering a symbolic expression, graphing it, and showing a table of function values. Figure 8 shows a graphing calculator display of the symbolic, graphic, and numeric views of sin(x) (18).



X	F1
-.60938	-.57236
-.40625	-.39517
-.20313	-.20173
0	0
.203125	.201731
.40625	.395167
.609375	.572355

F1(X): SIN(X)

Figure 8.

FUTURE NEED FOR CALCULATORS

Technical students and professionals will always need to do some back-of-the-envelope-type calculations quickly and conveniently. The key-per-function model of a calculator fits in nicely with this need. So does a device that is dedicated, personal, portable, low cost, and has long battery life. Users' expectations will be influenced by improvements in computer speed and memory size. Also, video game users have higher expectations for interactivity, better controls, color, animation, quick responses, good graphic design, and visual quality. For the future, calculators can take advantage of advances in computer technology and the decreasing cost of electronic components to move to modern platforms that have the benefits of increased speed, more memory, better displays, color displays, more versatile input devices (such as pen and voice), and more extensive communication capability (such as wireless communication).

BIBLIOGRAPHY

1. A. Ralston and E. D. Reilly, Jr.,. (eds.), *Encyclopedia of Computer Science and Engineering*, 2nd ed. New York: Van Nostrand Reinhold, 1983.
2. *Jones Telecommunications and Multimedia Encyclopedia*, Jones Digital Century. Available: http://www.digitalcentury.-com.
3. G. C. Beakley and R. E. Lovell, *Computation, Calculators, and Computers*. New York: Macmillan, 1983.
4. T. W. Beers, D. K. Byrne, G. L. Eisenstein, R. W. Jones, and P. J. Megowan,HP 48SX interfaces and applications, *Hewlett-Packard J*, **42**(3): 13–21, 1991.
5. P. D. Brown, G. J. May, and M. Shyam, Electronic design of an advanced technical handheld calculator, *Hewlett-Packard J.*, **38**(8): 34–39, 1987.
6. M. A. Smith, L. S. Moore, P. D. Brown, J. P. Dickie, D. L. Smith, T. B. Lindberg, and M. J. Muranami, Hardware design of the HP 48SX scientific expandable calculator, *Hewlett-Packard J.*, **42**(3): 25–34, 1991.
7. C. Maze, The first HP liquid crystal display, *Hewlett-Packard J.*, **31**(3): 22–24, 1980.
8. T. Lindberg, Packaging the HP-71B handheld computer, *Hewlett-Packard J.*, **35**(7): 17–20, 1984.
9. B. R. Hauge, R. E. Dunlap, C. D. Hoekstra, C. N. Kwee, and P. R. Van Loan, A multichip hybrid printed circuit board for advanced handheld calculators, *Hewlett-Packard J.*, **38**(8): 25–30, 1987.
10. D. E. Hackleman, N. L. Johnson, C. S. Lage, J. J. Vietor, and R. L. Tillman,CMOSC: Low-power technology for personal computers, *Hewlett-Packard J.*, **34**(1): 23–28, 1983.
11. J. L. Peterson, and A. Silberschatz, *Operating System Concepts*, 2nd ed. Reading: Addison-Wesley, 1985.
12. D. K. Byrne, C. M. Patton, D. Arnett, T. W. Beers, and P. J. McClellan, An advanced scientific graphing calculator, *Hewlett-Packard J.*, **45**(4): 6–22, 1994.
13. N. R. Scott, *Computer Number Systems and Arithmetic*. Englewood Cliffs, NJ: Prentice-Hall, 1985.
14. T. M. Whitney, F. Rode, and C. C. Tung, The "powerful pocketful": An electronic calculator challenges the slide rule, *Hewlett-Packard J.*, **23**(10): 2–9, 1972.
15. W. E. Egbert, Personal calculator algorithms I: Square roots, *Hewlett-Packard J.*, **28**(9): 22–23, 1977.
16. W. E. Egbert, Personal calculator algorithms II: Trigonometric functions, *Hewlett-Packard J.*, **28**(10): 17–20, 1977.
17. W. E. Egbert, Personal calculator algorithms IV: logarithmic functions, *Hewlett-Packard J.*, **29**(8): 29–32, 1978.
18. T. W. Beers, D. K. Byrne, J. A. Donnelly, R. W. Jones, and F. Yuan, A graphing calculator for mathematics and science classes, *Hewlett-Packard J.*, **47**(3): 1996.

DIANA K. BYRNE
Corvallis, Oregon

FAULT-TOLERANT COMPUTING

INTRODUCTION

Fault-tolerant computing can be defined as the process by which a computing system continues to perform its specified tasks correctly in the presence of faults. These faults could be transient, permanent, or intermittent faults. A fault is said to be transient if it occurs for a very short duration of time. Permanent faults are the faults that continue to exist in the system, and intermittent faults repeatedly appear for a period of time. They could be either hardware or software faults caused by errors in specification, design, or implementation, or faults caused by manufacturing defects. They also could be caused by external disturbances or simply from the aging of the components.

The goal of fault-tolerant computing is to improve the dependability of a system where dependability can be defined as the ability of a system to deliver service at an acceptable level of confidence. Among all the attributes of dependability, reliability, availability, fault coverage, and safety commonly are used to measure the dependability of a system. Reliability of a system is defined as the probability that the system performs its tasks correctly throughout a given time interval. Availability of a system is the probability that the system is available to perform its tasks correctly at time *t*. Fault coverage, in general, implies the ability of the system to recover from faults and to continue to operate correctly given that it still contains a sufficient complex of functional hardware and software. Finally, the safety of a system is the probability that the system either will operate correctly or switch to a safe mode if it operates incorrectly in the event of a fault.

The concept of improving dependability of computing systems by incorporating strategies to tolerate faults is not new. Earlier computers, such as the Bell Relay Computer built in 1944, used ad hoc techniques to improve reliability. The first systematic approach for fault-tolerant design to improve system reliability was published by von Neumann in 1956. Earlier designers improved the computer system reliability by using fault-tolerance techniques to compensate for the unreliable components that included vacuum tubes and electromechanical relays that had high propensity to fail. With the advent of more reliable transistor technology, the focus shifted from improving reliability to improving performance. Although component reliability has drastically improved over the past 40 years, increases in device densities have led to the realization of complex computer systems that are more prone to failures. Furthermore, these systems are becoming more pervasive in all areas of daily life, from medical life support systems where the effect of a fault could be catastrophic to applications where the computer systems are exposed to harsh environments, thereby increasing their failure rates. Researchers in industry and academe have proposed techniques to reduce the number of faults. However, it has been recognized that such an approach is not sufficient and that a shift in design paradigm to incorporate strategies to tolerate faults is necessary to improve dependability. Moreover, advances in technologies, such as very large-scale integration (VLSI), have led to manufacturing processes that are not only complex but also sensitive, which results in lower yields. Thus, it is prudent to develop novel fault-tolerance techniques for yield enhancement.

Research in this area in the last 50 years has resulted in numerous highly dependable systems that span a wide range of applications from commercial transactions systems and process control to medical systems and space applications. However, the *ever increasing* complexity of computer systems and the *ever increasing* need for error-free computation results, together with advances in technologies, continue to create interesting opportunities and unique challenges in fault-tolerant computing.

PRINCIPLES OF FAULT-TOLERANT COMPUTER SYSTEMS

Although the goal of fault-tolerant computer design is to improve the dependability of the system, it should be noted that a fault-tolerant system does not necessarily imply a highly dependable system and that a highly dependable system does not necessarily imply a fault-tolerant system.

Any fault-tolerance technique requires the use of some form of redundancy, which increases both the cost and the development time. Furthermore, redundancy also can impact the performance, power consumption, weight, and size of the system. Thus, a good fault-tolerant design is a *trade-off* between the level of dependency provided and the amount of redundancy used. The redundancy could be in the form of hardware, software, information, or temporal redundancy.

One of the most challenging tasks in the design of a fault-tolerant system is to evaluate the design for the dependability requirements. All the evaluation methods can be divided into two main groups, namely the qualitative and the quantitative methods. Qualitative techniques, which typically are subjective, are used when certain parameters that are used in the design process cannot be quantified. For example, a typical user should not be expected to know about fault-tolerance techniques used in the system to use the system effectively. Thus, the level of transparency of the fault-tolerance characteristics of the system to the user could determine the effectiveness of the system.

As the name implies, in the quantitative methods, numbers are derived for a certain dependability attribute of the system and different systems can be compared with respect to this attribute by comparing the numerical values. This method requires the development of models. Several probabilistic models have been developed based on combinatorial techniques and Markov and semi-Markov stochastic processes. Some disadvantages of the combinatorial models are that it is very difficult to model complex systems and

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

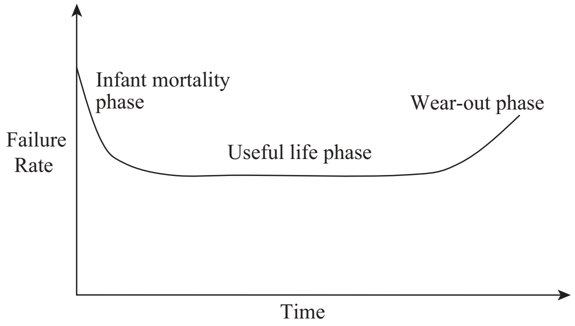


Figure 1. Bathtub curve.

difficult to model systems where repairing a faulty module is feasible.

Fundamental to all these models is the failure rate, defined as the expected number of failures per a time interval. Failure rates of most electronic devices follow a bathtub curve, shown in Fig. 1. Usually, the useful life phase is the most important phase in the life of a system, and during this time, the failure rate is assumed to be constant and denoted typically by λ. During this phase, reliability of a system and the failure rate are related by the following equation, which is known as the exponential failure law.

$$R(t) = e^{-\lambda t}$$

Figure 2 shows the reliability function with the constant failure rate. It should be noted that the failure rate λ is related to the mean time to failure (MTTF) as MTTF = 1/λ where MTTF is defined as the expected time until the occurrence of the first failure of the system.

Researchers have developed computer-aided design (CAD) tools based on quantitative methods, such as CARE III (the NASA Langley Research Center), SHARPE (Duke University), DEPEND (University of Illinois at Urbana-Champaign), and HIMAP (Iowa State University).

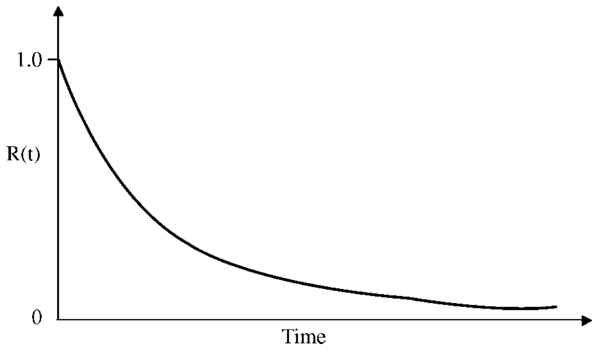


Figure 2. Reliability function with constant failure rate.

In the last several years, a considerable interest has developed in the experimental analysis of computer system dependability. In the design phase, CAD tools are used to evaluate the design by extensive simulations that include simulated fault injection; during the prototyping phase, the system is allowed to run under a controlled environment. During this time, physical fault injection is used for the evaluation purposes. Several CAD tools are available, including FTAPE developed at the University of Illinois at Urbana Champaign and Ballista developed at the Carnegie-Mellon University. In the following subsections, several common redundancy strategies for fault-tolerance will be described briefly.

Hardware Redundancy

Hardware redundancy is one of the most common forms of redundancy in which a computer system is partitioned into different subsystems or modules, and these are replicated so that a faulty module can be replaced by a fault-free module. Because each subsystem acts as a fault-containment region, the size of the fault-containment region dependes on the partitioning strategy.

Three basic types of hardware redundancy are used, namely, passive, active, and hybrid. In the passive techniques, also known as static techniques, the faults are masked or hidden by preventing them from producing errors. Fault masking is accomplished by voting mechanisms, typically majority voting, and the technique does not require any fault detection or system reconfiguration. The most common form of this type of redundancy is triple modular redundancy (TMR). As shown in Fig. 3, three identical modules are used, and majority voting is done to determine the output. Such a system can tolerate one fault. However, the voter is the single point of failure; that is, the failure of the voter leads to the compete failure of the system.

A generalization of the TMR technique is the N-modular redundancy (NMR), where N modules are used and N typically is an odd number. Such a system can mask up to $\lfloor \frac{N-1}{2} \rfloor$ faulty modules.

In general, the NMR system is a special case of an M-of-N system that consists of N modules, out of which at least M of them should work correctly in order for the system to operate correctly. Thus, system reliability of an M-of-N

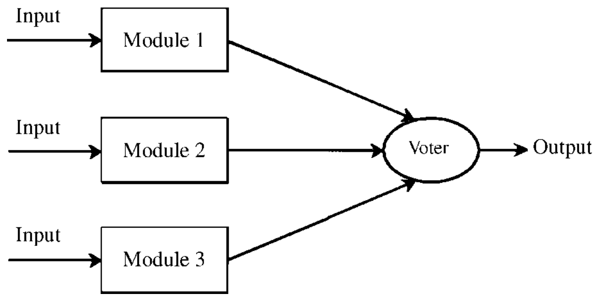


Figure 3. Basic triple modular redundancy.

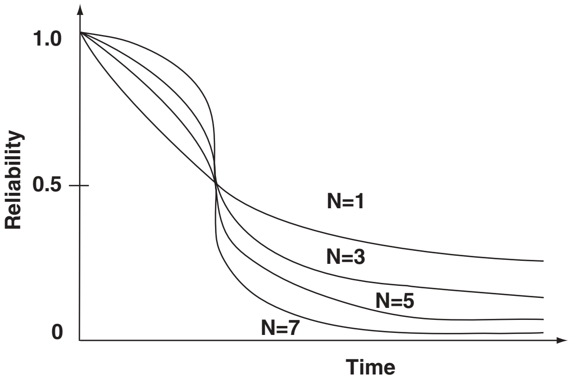


Figure 4. Reliability plots of NMR system with different values of N. Note that $N = 1$ implies a simplex system.

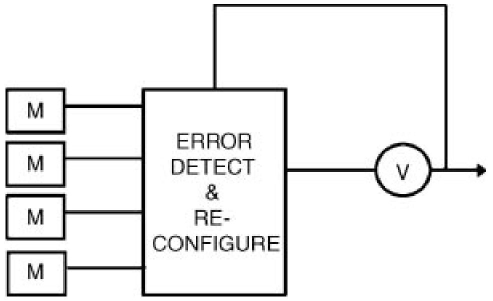


Figure 5. TMR system with hot spare.

system is given by

$$R(t) = \sum_{i=M}^{N} \binom{N}{i} R^{i}(t)\left[1 - R(t)\right]^{N-i}$$

For a TMR system, $N = 3$ and $M = 2$ and if an nonideal voter is assumed ($R_{\text{voter}}<1$), then the system reliability is given by

$$\begin{aligned} R(t) &= \sum_{i=2}^{3} \binom{3}{i} R^{i}(t)[1 - R(t)]^{3-i} \\ &= R_{voter}(t)\left(3R^{2}(t) - 2R^{3}(t)\right) \end{aligned}$$

Figure 4 shows reliability plots of simplex, TMR, and NMR when $N = 5$ and 7. It can be seen that higher redundancy leads to higher reliability in the beginning. However, the system reliability sharply falls at the end for higher redundancies.

In the active hardware redundancy, also known as dynamic hardware redundancy, some form of a reconfiguration process is performed to isolate the fault after it is detected and located. This process typically is followed by a recovery process to undo the effects of erroneous results. Unlike passive redundancy, faults can produce errors that are used to detect faults in dynamic redundancy. Thus, this technique is used for applications where consequences of erroneous results are not disastrous as long as the system begins to operate correctly within an acceptable time period. In general, active redundancy requires less redundancy than passive techniques. Thus, this approach is attractive for environments where the resources are limited, such as limited power and space. One major disadvantage of this approach is the introduction of considerable delays during the fault location, reconfiguration, and recovery processes. Some active hardware redundancy schemes include duplication with comparison, pair-and-a-spare technique, and standby sparing.

The hybrid hardware redundancy approach incorporates the desirable features of both the passive and active strategies. Fault masking is used to prevent the generation of erroneous results, and dynamic techniques are used when fault masking fails to prevent the production of erroneous results. Hybrid approaches usually are most expensive in terms of hardware. Examples of hybrid hardware redundancy schemes include self-purging redundancy, N-modular redundancy with spares, triple-duplex, sift-out modular redundancy, and the triple-triplex architecture used in the Boeing 777 primary flight computer.

Figure 5 shows a TMR system with a hot spare. The first fault is masked, and if this faulty module is replaced by the spare module, then the second fault also can be masked. Note that for a static system, masking two faults requires five modules. Reliability of the TMR system with a hot spare can be determined based on the Markov model shown in Fig. 6. Such models assume that (*1*) the system starts in the perfect state, (*2*) a single mode of failure exists, that is, only a single failure occurs at a time, and (*3*) each module follows the exponential failure law with a constant failure rate λ. A transition probability is associated with each state transition. It can be shown that if the module was operational at time t, then the probability that the module has failed at

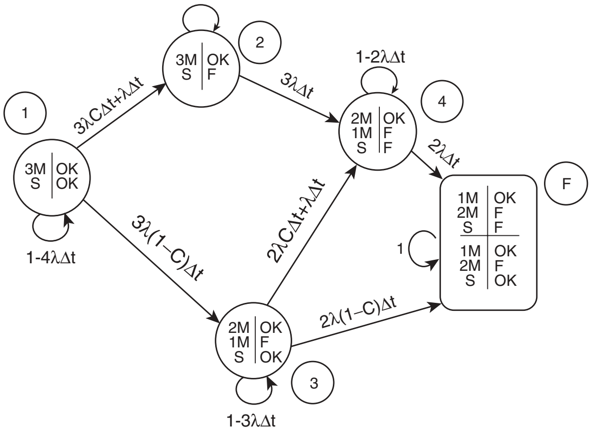


Figure 6. The Markov model of the TMR system with a hot spare.

time $t + \Delta t$ is

$$1 - e^{-\lambda \Delta t}$$

For $\lambda \Delta t < 1$,

$$1 - e^{-\lambda \Delta t} \approx \lambda \Delta t$$

The equations of the Markov model of the TMR system with a hot spare can be written in the matrix form as

$$\begin{bmatrix} P_1(t+\Delta t) \\ P_2(t+\Delta t) \\ P_3(t+\Delta t) \\ P_4(t+\Delta t) \\ P_F(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1 - 4\lambda\Delta t & 0 & 0 & 0 & 0 \\ 3\lambda C\Delta t + \lambda\Delta t & 1 - 3\lambda\Delta & 0 & 0 & 0 \\ 3\lambda(1-C)\Delta t & 0 & 1 - 3\lambda\Delta & 0 & 0 \\ 0 & 3\lambda\Delta t & 2\lambda C\Delta t + \lambda\Delta t & 1 - 2\lambda\Delta t & 0 \\ 0 & 0 & 2\lambda(1-C)\Delta t & 2\lambda\Delta t & 1 \end{bmatrix} \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \\ P_4(t) \\ P_5(t) \end{bmatrix}$$

The above discrete-time equations can be written in a compact form as

$$\mathbf{P}(t + \Delta t) = \mathbf{A}\mathbf{P}(t)$$

and the solution as

$$\mathbf{P}(n\Delta t) = \mathbf{A}^n \mathbf{P}(0)$$

where

$$P(0)| = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

It can be seen that the system reliability $R(t)$ is given by

$$R(t) = 1 - P_F(t) = \sum_{i=1}^{4} P_i(t)$$

where $P_F(t)$ is the probability that the system has failed. The continuous-time equations can be derived from the above equations by letting Δt approach zero; that is

$$P_1'(t) = \lim_{\Delta t \to 0} \frac{P_1(t+\Delta t) - P_1(t)}{\Delta t}$$

$$P_1'(t) = \lim_{\Delta t \to 0} \frac{P_1(t+\Delta t) - P_1(t)}{\Delta t} = -4\lambda P_1(t)$$

Similarly,

$$P_2'(t) = (3C + 1)\lambda P_1(t) - 3\lambda P_2(t)$$

$$P_3'(t) = (1 - C)3\lambda P_1(t) - 3\lambda P_3(t)$$

$$P_4'(t) = 3\lambda P_2(t) - 2\lambda P_4(t) + (2C\lambda + \lambda)P_3(t)$$

$$P_F'(t) = (2\lambda - 2\lambda C)P_3(t) + 2\lambda P_4(t)$$

Using Laplace transforms

$$sP_1(s) - P_1(0) = -4\lambda P_1(s)$$

$$P_1(s) = \frac{1}{s + 4\lambda} \Rightarrow P_1(t) = e^{-4\lambda t}$$

$$sP_2(s) - P_2(0) = P_1(s)(3C + 1)\lambda - 3\lambda P_2(s)$$

$$P_2(s) = \frac{(3C+1)\lambda}{(s+3\lambda)(s+4\lambda)} = (3C+1)\left(\frac{1}{s+3\lambda} - \frac{1}{s+4\lambda}\right)$$

$$P_2(t) = (3C + 1)(e^{-3\lambda t} - e^{-4\lambda t})$$

In the similar way,

$$P_3(t) = 3(1 - C)(e^{-3\lambda t} - e^{-4\lambda t})$$

$$P_4(t) = -3(C^2 - 2C - 1)(e^{-4\lambda t} + e^{-2\lambda t} - 2e^{-3\lambda t})$$

$$R(t) = 1 - P_F(t) = \sum_{i=1}^{4} P_i(t)$$

It can be seen that when the failure rate, λ, is 0.1 failures/hour and the fault coverage is perfect ($C = 1$) then the reliability of the TMR system with a hot spare is greater than the static TMR system after one hour. Moreover, it can be noted that the system reliability can be improved by modifying the system so that it switches to simplex mode after the second failure.

Software Redundancy

Unlike hardware faults, software faults are caused by mistakes in software design and implementation. Thus, a software fault in two identical modules cannot be detected by usual comparison. Many software fault-tolerance techniques have used approaches similar to that of the hardware redundancy. Most of these schemes assume fault independence and full fault coverage and fall under the software replication techniques, including *N*-version programming, recovery blocks, and *N* self-checking programming.

The basic idea of *N*-version programming is to design and code *N* different software modules; the voting is performed, typically majority voting, on the results produced by these modules. Each module is designed and implemented by a separate group of software engineers, based on the same specifications. Because the design and implementation of each module is done by an independent group, it is expected that a mistake made by a group will not be repeated by another group. This scheme complements *N*-modular hardware redundancy. A TMR system where all three processors are running the same copy of the software module will be rendered useless if a software fault will affect all the processors in the same way. *N*-version programming can avoid the occurrence of such a situation. This scheme is prone to faults from specification mistakes. Thus, a proven methodology should be employed to verify the correctness of the specifications.

In an ultra- reliable system, each version should be made as diverse as possible. Differences may include software developers, specifications, algorithms, programming languages, and verification and validation processes. Usually it is assumed that such a strategy would make different versions fail independently. However, studies have shown that although generally this is true, instances have been reported where the failures were correlated. For example, certain types of inputs, such as division by zero, that need special handling and potentially are overlooked easily by different developers could lead to correlated failures.

In the recovery block schemes, one version of the program is considered to be the primary version and the remaining *N*-1 versions are designated as the secondary versions or spares. The primary version normally is used if it passes the acceptance tests. Failure to pass the acceptance tests prompts the use of the first secondary version. This process is continued until one version is found that passes the acceptance tests. Failure of the acceptance tests by all the versions indicates failure of the system. This approach is analogous to the cold standby sparing approach of the hardware redundancy, and it can tolerate up to *N*-1 faults.

In *N* self-checking programming, *N* different versions of the software, together with their acceptance tests, are written. The output of each program, together with its acceptance tests results, are input to a selection logic that selects the output of the program that has passed the acceptance tests.

Information Redundancy

Hardware redundancy is very effective but expensive. In information redundancy, redundant, information is added to enable fault detection, and sometimes fault tolerance, by correcting the affected information. For certain systems, such as memory and bus, error-correcting codes are cost effective and efficient. Information redundancy includes all error-detecting codes, such as various parity codes, *m-of-n* codes, duplication codes, checksums, cyclic codes, arithmetic codes, Berger codes, and Hamming error-correcting codes.

Time Redundancy

All the previous redundancy methods require a substantial amount of extra hardware, which tends to increase size, weight, cost, and power consumption of the system. Time redundancy attempts to reduce the hardware overhead by using extra time. The basic concept is to repeat execution of a software module to detect faults. The computation is repeated more than once, and the results are compared. An existence of discrepancy suggests a fault, and the computations are repeated once more to determine if the discrepancy still exists. Earlier, this scheme was used for transient fault detection. However, with a bit of extra hardware, permanent faults also can be detected. In one of the schemes, the computation or transmission of data is done in the usual way. In the second round of computation or transmission, the data are encoded and the results are decoded before comparing them with the previous results. The encoding scheme should be such that it enables the detection of the faults. Such schemes may include complementation and arithmetic shifts.

Coverage

The coverage probability *C* used in the earlier reliability-modeling example was assumed to be a constant. In reality, of course, it may well vary, depending on the circumstances. In that example (TMR with a spare), the coverage probability is basically the probability that the voting and switching mechanism is functional and operates in a timely manner. This probability in actuality may be dependent on the nature of the fault, the time that it occurs, and whether it is the first or the second fault.

In systems in which the failure rate is dominated by the reliability of the hardware modules being protected through redundancy, the coverage probability may be of relatively minor importance and treating it as a constant may be adequate. If extremely high reliability is the goal, however, it is easy to provide sufficient redundancy to guarantee that a system failure from the exhaustion of all hardware modules is arbitrarily small. In this case, failures from imperfect coverage begin to dominate and more sophisticated models are needed to account for this fact.

It is useful to think of fault coverage as consisting of three components: the probability that the fault is detected, the probability that it is isolated to the defective module, and the probability that normal operation can be resumed successfully on the remaining healthy modules. These probabilities often are time dependent, particularly in cases in which uninterrupted operation is required or in which loss of data or program continuity cannot be tolerated. If too much time elapses before a fault is detected, for example, erroneous data may be released with potentially unacceptable consequences. Similarly, if a database is corrupted before the fault is isolated, it may be impossible to resume normal operation.

Markov models of the sort previously described can be extended to account for coverage effects by inserting additional states. After a fault, the system transitions to a state in which a fault has occurred but has not yet been detected, from there to a state in which it has been detected but not yet isolated, from there to a state in which it has been isolated but normal operation has not yet resumed successfully, and finally to a successful recovery state. Transitions out of each of those states can be either to the next state in the chain or to a failed state, with the probability of each of these transitions most likely dependent on the time spent in the state in question. State transition models in which the transition rates are dependent on the state dwelling-time are called semi-Markov processes. These models can be analyzed using techniques similar to those illustrated in the earlier Markov analysis, but the analysis is complicated by the fact that differences between the transition rates that pertain to hardware or software faults and those that are associated with coverage faults can be vastly different. The mean time between hardware module failures, for example, typically is on the order of thousands of hours, whereas the time between the occurrence of a fault and its detection may well be on the order of microseconds. The resulting state-transition matrix is called "stiff", and any attempt to solve it numerically may not converge. For this reason, some of the aforementioned reliability models use a technique referred to as behavioral decomposition. Specifically, the model is broken down into its component parts, one subset of models used to account for coverage failures and the second incorporating the outputs of those models into one accounting for hardware and software faults.

FAULT TOLERANT COMPUTER APPLICATIONS

Incorporation of fault tolerance in computers has tremendous impact on the design philosophy. A good design is a trade-off between the cost of incorporating fault tolerance and the cost of errors, including losses from downtime and the cost of erroneous results.

Over the past 50 years, numerous fault-tolerant computers have been developed. Historically, the applications of the fault-tolerant computers were confined to military, communications, industrial, and aerospace applications where the failure of a computer could result in substantial financial loses and possibly loss of life. Most of the fault-tolerant computers developed can be classified into five general categories.

General-Purpose Computing

General-purpose computers include workstations and personal computers. These computers require a minimum level of fault tolerance because errors that disrupt the processing for a short period of time are acceptable, provided the system recovers from these errors. The goal is to reduce the frequency of such outages and to increase reliability, availability, and maintainability of the system.

Because the attributes of each of the main components, namely the processor, memory, and input/output, are unique, the fault-tolerant strategies employed for these components are different, in general. For example, the main memory may use a double-error detecting code on data and on address and control information, whereas cache may use parity on both data and address information. Similarly, the processor may use parity on data paths and control store and duplication and comparison of control logic, whereas input/output could use parity on data and control information.

Moreover, it is well known that as feature sizes decrease, the systems are more prone to transient failures than other failures and it is desirable to facilitate rapid recovery. One of the most effective approaches for rapid error recovery for transient faults is instruction retry, in which the appropriate instruction is retried once the error is detected. The instruction retry concept has been extended to a wide variety of other systems, including TMR systems and Very Large Instruction Word (VLIW) architectures.

Long-Life Applications

Applications in which manual intervention is impractical or impossible, such as unmanned space missions, demand systems that have a high probability of surviving unattended for long periods of time. Typical specifications require a 95% or better probability that the system is still functional at the end of a five- or ten-year mission. Computers in these environments are expected to use the hardware in an efficient way because of limited power availability and constraint on weight and size. Long-life systems sometimes can allow extended outages, provided the system becomes operational again. The first initiative for the development of such computers was taken by NASA for the Orbiting Astronomical Observatory, in which basic fault masking at the transistor level was used. Classic examples of systems for long-life applications include the Self-Testing And Repairing (STAR) computer, the Fault-Tolerant Space-borne Computer (FTSC), the JPL Voyager computers, and the Fault-Tolerant Building Block Computer (FTBBC).

The STAR computer was the first computer that used dynamic recovery strategies that used extensive instrumentation for fault detection and signal errors. Galileo Jupiter system is a distributed system based on 19 microprocessors with 320 kilobytes of ROM. Block redundancy is used for all the subsystems, and they operate as standby pairs, except the command and data subsystem that operate as an active pair. Although special fault protection software is used to minimize the effects of faults, fault identification and isolation are by ground intervention.

Critical-Computation Applications

In these applications, errors in computations can be catastrophic, affecting such things as human safety. Applications include a certain type of industrial controllers, aircraft flight control systems, and military applications. A typical requirement for such a system is to have reliability of 0.9999999 at the end of 3-hour time period.

An obvious example of a critical-computation application is the space shuttle. Its computational core employs software voting in a five-computer system. The system is responsible for guidance, navigation, and preflight checkouts. Four computers are used as a redundant set during mission-critical phases, and the fifth computer performs noncritical tasks and also act as a backup for the primary four-computer system. The outputs of the four computers are voted at actuators while each computer compares its output with the outputs of the other three computers. A disagreement is voted by the redundancy-management circuitry of each computer, and it will isolate its computer if the vote is positive. Up to two computer failures can be tolerated in the voting mode operation. A second failure will force the system to function as a duplex system, that uses comparison and self-tests, and it can tolerate one more failure in this mode. The fifth computer contains the flight software package developed by another vendor to avoid common software error.

Another interesting system that was developed recently is the flight control computer system of Boeing 777. In avionics, system design diversity is an important design criterion to tolerate common-mode failures, such as design flaws. One of the specifications includes mean time between maintenance actions to be 25,000 hours. The system can tolerate certain Byzantine faults, such as disagreement between the replicated units, power and component failures, and electromagnetic interference. The system uses triple-triplex redundancy, in which design diversity is incorporated by using three dissimilar processors in each triplex node. Each triplex node is isolated physically and electrically from the other two nodes. However, recent studies suggest that a single-point of failure exists in the software system. Instead of using different teams to generate different versions of the source code in Ada Boeing decided to use the same source code but three different Ada compilers to generate the object code.

High-Availability Applications

The applications under this category include time-shared systems, such as banking and reservation systems, where providing prompt services to the users is critical. In these applications, although system-wide outages are unacceptable, occasional loss of service to individual users is acceptable provided the service is restored quickly so that the downtime is minimized. In fact sometimes the downtime needed to update software and upgrade hardware may not be acceptable, and so well-coordinated approaches are used to minimize the impact on the users. Some examples of early highly available commercial systems include the NonStop Cyclone, Himalaya K10000, and Integrity S2, all products of Tandem Computers now part of Hewlett Packard; the Series 300 and Series 400 systems manufactured by Sequoia Systems, now owned by Radisys Corporation; and the Stratus XA/R Series 300 systems of the Stratus family.

Figure 7 shows the basic Tandem NonStop Cyclone system. It is a loosely coupled, shared-bus multiprocessor

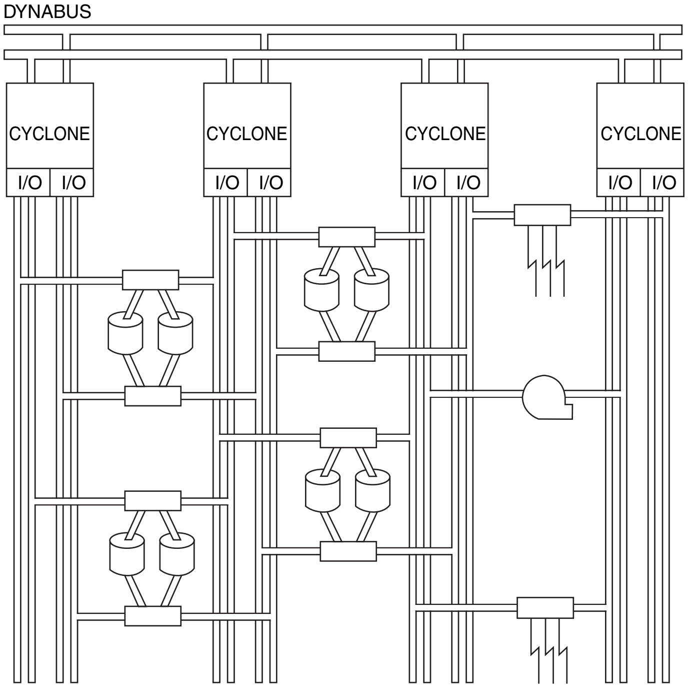


Figure 7. Tandem nonstop Cyclone system.

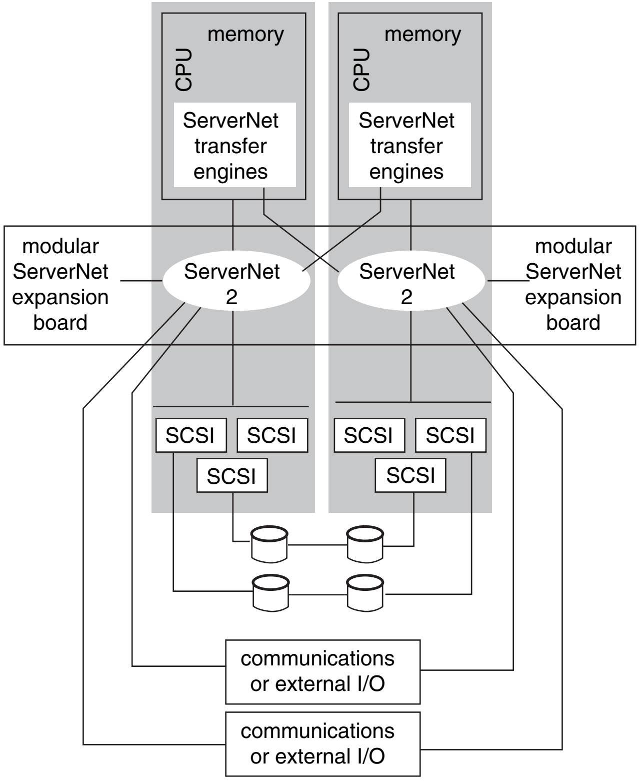

memory
CPU
ServerNet transfer engines
memory
CPU
ServerNet transfer engines
modular ServerNet expansion board
ServerNet 2
ServerNet 2
modular ServerNet expansion board
SCSI
SCSI
SCSI
SCSI
SCSI
SCSI
communications or external I/O
communications or external I/O

Figure 8. Tandem S-series NonStop server architecture.

system that was designed to prevent a single hardware or software failure from disabling the entire system. The processors are independent and use Tandem's GURADIAN 90 fault-tolerant operating system with load balancing capabilities. The NonStop Cyclone system employs both hardware and software fault-tolerant strategies. By extensively using hardware redundancy in all the subsystems—multiple processors, mirrored disks, multiple power supplies, redundant buses, and redundant input/output controllers—a single component failure will not disable the system. Error propagation is minimized by making sure that fault detection, location, and isolation is done quickly. Furthermore, Tandem has incorporated other features that also improve the dependability of the system. For example, the system supports on-site replaceable, hot-pluggable cards known as field replaceable units (FRUs) to minimize any downtime.

In the newer S-series, NonStop Himalaya servers (Figure 8), the processor and I/O buses are replaced by proprietary network architecture called the ServerNet that incorporates numerous data integrity schemes. Furthermore, redundant power and cooling fans are provided to ensure continuity in service. A recent survey suggests that more than half of the credit card and automated teller machine transactions are processed by NonStop Himalaya computers.

Stratus Technologies, Inc. has introduced fault-tolerant servers with the goal of providing high availability at lower complexity and cost. For example, its V series server systems that are designed for high-volume applications use TMR hardware scheme to provide at least 99.999% probability of surviving all hardware faults. These systems use Intel Xeon Processors and run on a Stratus VOS operating system that provides a highly secured environment. A single copy manages a module that consists of tightly coupled processors.

The Sequoia computers were designed to provide a high level of resilience to both hardware and software faults and to do so with a minimum of additional hardware. To this end, extensive use was made of efficient error detection and correction mechanisms to detect and isolate faults combined with the automatic generation of periodic system-level checkpoints to enable rapid recovery without loss of data or program continuity. Checkpoints are established typically every 40 milliseconds, and fault recovery is accomplished in well under a second, making the occurrence of a fault effectively invisible to most users.

Recently Marathon Technologies Corporation has released software solutions, *everRun FT* and *everRun HA,* that enable any two standard Windows servers to operate as a single, fully redundant Windows environment, protecting applications from costly downtime due to failures within the hardware, network, and data.

Maintenance Postponement Applications

Applications where maintenance of computers is either costly or difficult or perhaps impossible to perform, such as some space applications, fall under this category. Usually the breakdown maintenance costs are extremely high for the systems that are located in remote areas. A maintenance crew can visit these sites at a frequency that is cost-effective. fault-tolerance techniques are used between these visits to ensure that the system is working correctly. Applications where the systems may be remotely located include telephone switching systems, certain renewable energy generation sites, and remote sensing and communication systems. One of the most popular systems in the literature in this group is AT&T's Electronic Switching System (ESS). Each subsystem is duplicated. While one set of these subsystems perform all the functions, the other set acts as a backup.

PRINCIPLES OF FAULT-TOLERANT MULTIPROCESSORS AND DISTRIBUTED SYSTEMS

Although the common perception is that the parallel systems are inherently fault-tolerant, they suffer from high failure rates. Most of the massively parallel systems are built based on performance requirements. However, their dependability has become a serious issue; numerous techniques have been proposed in the literature, and several vendors have incorporated innovative fault-tolerance schemes to address the issue. Most of these schemes are hierarchical in nature incorporating both hardware and software techniques. Fault-tolerance techniques are designed for each of the following levels such that the faults that are not detected at lower levels should be handled at higher levels.

At the technology and circuit levels, decisions are made to select the appropriate technology and circuit design techniques that will lead to increased dependability of the modules. At the node level, architecture involves design of VLSI chips that implement the processor, whereas at the internode architecture level, considerations are given to how the nodes are connected and effective reconfiguration in the presence of faulty processors, switches, and links. At the operating system level, recovery of the system is addressed, which may involve checkpointing and rollback after a certain part of the system has been identified faulty and allocating tasks of the faulty processor to the remaining operating processor. Finally, at the application level, the user uses mechanisms to check the results.

A couple of most popular approaches to fault tolerance in multiprocessor systems are static or masking redundancy and dynamic or standby redundancy

Static Redundancy

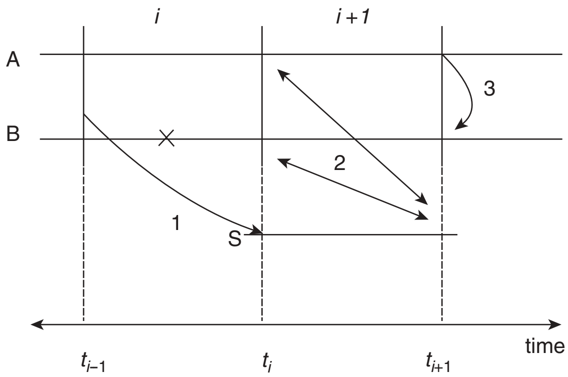
Static redundancy is used in multiprocessor system for reliability and availability, safety, and to tolerate nonclassic faults. Although redundancy for reliability and availability uses the usual NMR scheme, redundancy for safety usually requires a trade-off between reliability and safety. For example, one can use a k-out-of-n system where a voter produces the output y only if at least k modules agree on the output y. Otherwise the voter asserts the unsafe flag. The reliability refers to the probability that the system produces correct output, and safety refers to the probability that the system either produces the correct output or the error is detectable. Thus, high reliability implies high safety. However, the reverse is not true.

Voting of the NMR system becomes complicated when nonclassic faults are present. In systems requiring ultrahigh reliability, a voting mechanism should be able to handle arbitrary failures, including the malicious faults where more than one faulty processor may collaborate to disable the system. The Byzantine failure model was proposed to handle such faults.

Dynamic Redundancy

Unlike static redundancy, dynamic redundancy in multiprocessor systems includes built-in fault detection capability. Dynamic redundancy typically follows three basic steps: (*1*) fault detection and location, (*2*) error recovery, and (*3*) reconfiguration of the system. When a fault is detected and a diagnosis procedure locates the fault, the system is usually reconfigured by activating a spare module; if the spare module is not available, then the reconfigured system may have to operate under a degraded mode with respect to the resources available. Finally, error recovery is performed in which the spare unit takes over the functionality of the faulty unit from where it left off. Although various approaches have been proposed for these steps, some of them have been successfully implemented on the real systems. For example, among the recovery strategies, the most widely adopted strategy is the rollback recovery using checkpoints. The straightforward recovery strategy is to terminate execution of the program and to re-execute the complete program from the beginning on all the processors, which is known as the global restart. However, this leads to severe performance degradation. Thus, the goal of all the recovery strategies is to perform effective recovery from the faults with minimum overheads. The rollback recovery using checkpoints involves storing on adequate amount of processor state information at discrete points during the execution of the program so that the program can be rolled back to these points and restarted from there in the event of a failure. The challenge is when and how these checkpoints are created. For interacting processes, inadequate checkpointing may lead to a domino effect where the system is forced to go to a global restart state. Over the years, researchers have proposed strategies that effectively address this issue.

Although rollback recovery schemes are effective in most systems, they all suffer from substantial inherent latency that may not be acceptable for real-time systems. For such systems, forward recovery schemes are typically used. The basic concept common to all the forward recovery schemes is that when a failure is detected, the system discards the current erroneous state and determines the correct state without any loss of computation. These schemes are either based on hardware or software redundancy. The hardware based schemes can be classified further as static or dynamic redundancy. Several schemes based on dynamic redundancy and checkpointing that avoid rollback have been proposed in the literature. Like rollback schemes, when a failure is detected, the roll-forward checkpointing schemes (RFCS) attempt to determine the faulty module and the correct state of the system is restored once the fault diagnosis is done, although the diagnostic methods may differ. For example, in one RFCS, the faulty processing module is identified by retrying the computation on a spare processing module. During this retry, the duplex modules continue to operate normally. Figure 9 shows execution of two copies, A and B of a task. Assume that B fails in the checkpointing interval i. The checkpoints of A and B will not match at the end of the checkpointing interval i at time t_i. This mismatch will trigger concurrent retry of the checkpoint interval i on a spare module, while A and B continue to execute into the checkpointing interval i+1. The previous checkpoint taken



1: Copy state and executable code to spare
2: Compare states A and S and B and S
3: Copy state from A to B
X: A fault

Figure 9. A RFCS scheme.

at t_{i-1} together with the executable code is loaded into the spare, and the checkpoint interval i in which the fault occurred is retired on the spare module. When the spare module completes the execution at time t_{i+1} the state of the spare module is compared with the states of A and B at time t_i and B is identified faulty. State of A copied to B. A rollback is required when both A and B have failed. In this scheme, one fault is tolerated without paying the penalty of the rollback scheme. Several version of this scheme have been proposed in the literature.

FAULT-TOLERANCE IN VLSI CIRCUITS

The advent of VLSI circuits has made it possible to incorporate fault-tolerant techniques that were too expensive to implement earlier. For example, it has enabled schemes to provide redundant processors and fault detection and location capabilities within the chip. This is mainly from increasing packaging densities and decreasing cost and power consumption. However, VLSI also has several disadvantages. Some of these disadvantages include a higher percentage of internal faults compared with previous small-scale integration (SSI) and large-scale integration (LSI) during the manufacturing phase leading to low yield; increased complexity of the system has led to considerable increase in design mistakes, and imperfections can lead to multiple faults from smaller feature sizes. Furthermore, VLSI circuits are more prone to common-mode failures, and decreased feature sizes and lower operating voltages have made the circuits more susceptible to external disturbances, such as a particles and variations in the supply voltages, respectively, which lead to an increase in transient faults. Thus, the main motivations for incorporating fault-tolerance in VLSI circuits is yield enhancement and to enable real-time and compile-time reconfiguration with the goal of isolating a faulty functional unit, such as a processor or memory cells, during field operation.

Strategies for manufacture-time and compile-time reconfigurations are similar since they do not affect the normal operation of the system and no real-time constraints are imposed on reconfiguration time. Real-timer reconfiguration schemes are difficult to design since the reconfiguration has to be performed without affecting the operation of the system. If erroneous results are not acceptable at all, then static or hybrid techniques are typically used; otherwise, cheaper dynamic techniques would suffice. The effectiveness of any reconfiguration scheme is measured by the probability that a redundant unit can replace a faulty unit and the amount of reconfiguration overhead involved.

Numerous schemes have been proposed for all three types of reconfiguration in the literature. One of the first schemes that was proposed for an array of processing elements allowed each processing element to be converted into a connecting element for signal passing so that a failed processor resulted in converting other processors in the corresponding row and column to connecting elements and they ceased to perform any computation. This simple scheme is not practical for multiple faults since an entire row and column must be disabled for each fault. This problem has been addressed by several reconfiguration schemes that either use spare columns and rows or redundant processing elements are dispersed throughout the chip, such as the interstitial redundancy scheme.

Advances in nanometer-scale geometries have made devices prone to new types of defects that greatly limit the yields. Furthermore, the existing external infrastructure, such as automatic test equipment (ATE), are not capable of dealing with the new defect levels that nanometer-scale technologies create in terms of fault diagnosis, test time, and handling & the amount of test data generated. In addition, continual development of ATE facilities that can deal with the new manufacture issues is not realistic. This has considerably slowed down the design of various system-on-a-chip efforts. For example, although the embedded memory typically occupies half of the IC area, their defect densities tend to be twice that of logic. Thus, to achieve and maintain cost advantages requires improving memory yields. As described, one commonly used strategy to increase yield is to incorporate redundant elements that can replace the faulty elements during the repair phase. The exiting external infrastructure cannot perform these tasks in a cost-effective way. Thus, a critical need exists for an on-chip (internal) infrastructure to resolve these issues effectively. The semiconductor industry has attempted to address this problem by introducing embedded intellectual property blocks called the infrastructure intellectual property (IPP). For embedded memory, the IPP may consist of various types of test and repair capabilities.

TRENDS AND FUTURE

In the last few years, several developments have occurred that would define the future activities in the fault-tolerance area in general.

Conventional fault-tolerance techniques do not render themselves well in the area of mobile computing. For example, most schemes for checkpointing coordination and message logging schemes are usually complex and difficult to control because of the mobility of the hosts. Similarly, in the area of reconfigurable computing, including field programmable gate arrays (FPGAs), new fault-tolerance strategies are being developed based on the specific architectures.

Recent advances in nanoscience and nanometer-scale devices, such as carbon nanotubes and single electron transistors, will drastically increase device density and at the same time reduce power needs, weight, and size. However, these nanosystems are expected to suffer from high failure rates, both transient and permanent faults, because of the probabilistic behavior of the devices and lack of maturity in the manufacturing processes that will reduce the process yield. In fact, it is expected that the fault densities will be orders of magnitude greater than the current technology. This new circuit design paradigm is needed to address these issues. However, this will be an evolving area as the device fabrication processes are being scaled-up from the laboratory to the manufacturing level.

A good viable alternative approach for fault-tolerance would be to mimic biological systems. There is some activity in this area, but it is still in its infancy. For example, an approach to design complex computing systems with inherent fault-tolerance capabilities based on the embryonics was recently proposed. Similarly, a fault detection/location and reconfiguration strategy was proposed last year that was based on cell signaling, membrane trafficking, and cytokinesis.

Furthermore, there is an industry trend in space and aerospace applications to use commercial off-the-shelf (COTS) components for affordability. Many COTS have dependability issues, and they are known to fail when operated near their physical limits, especially in military applications. Thus, innovative fault-tolerance techniques are needed that can strike an acceptable balance between dependability and cost.

Moreover, although the hardware and software costs have been dropping, the cost of computer downtime has been increasing because of the increased complexity of the systems. Studies suggest that the state-of-the-art fault-tolerant techniques are not adequate to address this problem. Thus, a shift in the design paradigm may be needed. This paradigm shift is driven in large part by the recognition that hardware is becoming increasingly reliable for any given level of complexity, because, based on empirical evidence, the failure rate of an integrated circuit increases roughly as the 0.6th power of the number of its equivalent gates. Thus, for a given level of complexity, the system fault rate *decreases* as the level of integration increases. If, say, the average number of gates per device in a system is increased by a factor of 1000, the fault rate for that system decreases by a factor of roughly 16. In contrast, the rapid increase in the complexity of software over the years, because of the enormous increase in the number and variety of applications implemented on computers, has not been accompanied by a corresponding reduction in software bugs. Significant improvements have indeed been made in software development, but these are largely offset by the increase in software complexity and in the increasing opportunity for software modules to interact in unforeseen ways as the number of applications escalate. As a result, it is expected that the emphasis in future fault-tolerant computer design will shift from techniques for circumventing hardware faults to mechanisms for surviving software bugs.

FURTHER READING

G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, Detecting and locating faults in VLSI implementations of the advanced encryption standard, *Proc. 2003 IEEE Internat. Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 105–113.

N. Bowen and D. K. Pradhan, Issues in fault tolerant memory management, *IEEE Trans. Computers*, 860–880, 1996.

N. S. Bowen, and D. K. Pradhan, Processor and memory-based checkpoint and rollback recovery, *IEEE Computer*, **26** (2): 22–31, 1993.

M. Chatterjee, and D. K. Pradhan, A GLFSR-based pattern generator for near-perfect fault coverage in BIST, *IEEE Trans. Computers*, 1535–1542, 2003.

C. Chen,and A. K. Somani, Fault containment in cache memories for TMR redundant processor systems, *IEEE Trans. Computers*, **48** (4): 386–397, 1999.

R. Chillarege, and R. K. Iyer, Measurement-based analysis of error latency, *IEEE Trans. Computers*, 529–537, 1987.

K. Datta, R. Karanam, A. Mukherjee, B. Joshi, and A. Ravindran, A bio-inspired self-repairable distributed fault-tolerant design methodology with efficient redundancy insertion technique, *Proc. 16th IEEE NATW*, 2006.

R. J. Hayne, and B. W. Johnson, Behavioral fault modeling in a VHDL synthesis environment, *Proc. IEEE VLSI Test Symposium*, Dana Point, California, 1999, pp. 333–340.

B. W. Johnson, Design and analysis of fault tolerant digital systems. Reading, MA: Addison-Wesley, 1989.

B. Joshi, and S. Hosseini, Diagnosis algorithms for multiprocessor systems, *IEEE Workshop on Embedded Fault-Tolerant Systems*, Boston, MA, 1998.

L. M., Kaufman, S. Bhide, and B. W. Johnson, Modeling of common-mode failures in digital embedded systems", *Proc. 2000 IEEE Annual Reliability and Maintainability Symposium*, Los Angeles, California, 2000, pp. 350–357.

I. Koren, and C. Mani Krishna, Fault-Tolerant Systems. San Francisco, CA: Morgan Kauffman, 2007.

S. Krishnaswamy, I. L. Markov, and J. P. Hayes, Logic circuit testing for transient faults, *Proc. European Test Symposium (ETS'05)*, 2005.

P. Lala, Self-Checking and Fault Tolerant Digital Design. San Francisco, CA: Morgan Kaufmann, 2000.

L. Lamport, S. Shostak, and M. Pease, The Byzantine Generals Problem, *ACM Trans. Programming Languages and System*, 1982, pp. 382–401.

M. R. Lyu, ed., Software Reliability Engineering, New York: McGraw Hill, 1996.

M. Li, D. Goldberg, W. Tao, and Y. Tamir, Fault-tolerant cluster management for reliable high-performance computing, *13th International Conference on Parallel and Distributed Computing and Systems*, Anaheim, California, 2001, pp. 480–485.

C. Liu, and D. K. Pradhan, EBIST: A novel test generator with built-in-fault detection capability, *IEEE Trans. CAD*. In press.

A. Maheshwari, I. Koren, and W. Burleson, Techniques for transient fault sensitivity analysis and reduction in VLSI circuits, *Defect and Fault Tolerance in VLSI Systems*, 2003.

A. Messer, P. Bernadat, G. Fu, D. Chen, Z. Dimitrijevic, D. Lie, D. Mannaru, A. Riska, and D. Milojicic, Susceptibility of commodity systems and software to memory soft errors, *IEEE Trans. Computers*, Vol. **53**(12): 1557–1568, 2004.

N. Oh, P. P. Shirvani, and E. J. McCluskey, Error detection by duplicated instructions in superscalar processors, *IEEE Trans Reliability*, **51**: 63–75.

R. A. Omari, A. K. Somani, and G. Manimaran, An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems, *J. Parallel and Distributed Computing*, **65**(5): 595–608, 2005.

E. Papadopoulou, Critical Area Computation for Missing material Defects in VLSI Circuits, *IEEE Trans. CAD*, **20**: 503–528, 2001.

D. K. Pradhan,(ed.), Fault-Tolerant Computer System Design. Englowood Cliffs, NJ: Prentice Hall, 1996.

D. K. Pradhan, and N. K. Vaidya, Roll-forward checkpointing scheme: A novel fault-tolerant architecture, *IEEE Trans. Computers*, **43** (10), 1994.

A. V. Ramesh, D. W. Twigg, U. R. Sandadi, T. C. Sharma, K. S. Trivedi, and A. K. Somani, Integrated Reliability Modeling Environment, *Reliability Engineering and System Safety*, Elsevier Science Limited; UK, Volume 65, Issue 1, March 1999, pp. 65–75.

G. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee, Software-controlled fault tolerance, *ACM Trans. Architecture and Code Optimization*, 1–28, 2005.

A. Schmid, and Y. Leblebici, A highly fault tolerant PLA architecture for failure-prone nanometer CMOS and novel quantum device technologies, *Defect and Fault Tolerance in VLSI Systems*, 2004.

A. J. Schwab, B. W. Johnson, and J. Bechta-Dugan, Analysis techniques for real-time, fault-tolerant, VLSI processing arrays, *Proc. 1995 IEEE Annual Reliability and Maintainability Symposium,*Washington, D.C; 1995, pp. 137–143.

D. Sharma, and D. K. Pradhan, An efficient coordinated checkpointing scheme for multicomputers, *Fault-Tolerant Parallel and Distributed Systems*. IEEE Computer Society Press, 1995.

P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, Modeling the effect of technology trends on the soft error rate of combinational logic, *Proc. 2002 Internat. Conference on Dependable Systems and Networks*, 2002, pp. 389–399.

M. L. Shooman, Reliability of computer systems and networks, Hoboken, NJ: Wiley Inter-Science, 2002.

D. Siewiorek, and R. Swartz, The Theory and Practice of Reliable System Design. Wellesley, MA: A. K. Peters, 1999.

D. P. Siewiorek, (ed.), Fault Tolerant Computing Highlights from 25 Years, *Special volume of the 25th International Symposium on Fault-Tolerant Computing FTCS-25*, Pasadena, Califoria.

A. K. Somani, and N. H. Vaidya, Understanding fault tolerance and reliability, *IEEE Computer*, April 1997, pp. 45–50.

Tandem Computers Incorporated, *NonStop Cyclone/R*, Tandem Product report, 1993.

N. H. Vaidya, and D. K. Pradhan, Fault-tolerant design strategies for high reliability and safety, *IEEE Trans. Computers*, **42** (10), 1993.

J. von Neumann, *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*, Automata Studies, Annals of Mathematical Studies, Princeton University Press, No. **34**, 1956, pp. 43–98.

Y. Yeh, Design considerations in Boeing 777 fly-by-wire computers, *Proc. Third IEEE International High-Assurance Systems Engineering Symposium*, 1998, pp. 64–72.

S. R. Welke, B. W. Johnson, and J. H. Aylor, Reliability modeling of hardware-software systems, *IEEE Trans. Reliability*, **44** (3), 413–418, 1995.

Y. Zhang, and K. Chakrabarty, Fault recovery based on checkpointing for hard real-time embedded systems, *Proc. 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems* (DFT'03), 2003.

Y. Zorian, and M. Chandramouli, Manufacturability with Embedded Infrastructure IPs. Available: http://www.evaluationengineering.com/archive/articles/0504/0504embedded_IP.asp.

BHARAT JOSHI
University of
North Carolina-Charlotte
Charlotte, North Carolina

DHIRAJ PRADHAN
University of Bristol
Bristol, United Kingdom

JACK STIFFLER
Reliable Technologies, Inc.
Weston, Massachusetts

FIBER-OPTIC COMMUNICATION NETWORKS

INTRODUCTION

The continuing growth of traffic and demand for bandwidth has changed dramatically the telecommunication market and industry, and has created new opportunities and challenges. The proliferation of DSL and cable modems for residential homes and Ethernet links for businesses have spurred the need to upgrade network backbones with newer and faster technologies. It is estimated that Internet traffic doubles every 12 months driven by the popularity of Web services and the leveraged business model for data centers and centralized processing. The Telecom Deregulation Act of 1996 brought competition to the marketplace and altered the traditional traffic profile. Competition brought about technological advances that reduced the cost of bandwidth and allowed the deployment of data services and applications that in turn fueled higher demand for bandwidth. Coupled with the fact that most networks were optimized for carrying voice traffic, the telecommunication providers have had to change their business models and the way they build networks. As a result, voice and data traffic are now carried over separate networks, which doubles the cost and renders service delivery and network operation and management inefficient.

Optical networking seems to be the fix for all problems and limitations despite the downturn in the telecommunications industry in general and optical networking in particular, because all-optical or photonic switching is the only core networking technology that is capable of supporting the increasing amount of traffic and the demands for higher bandwidth, brought about by emerging applications such as grid computing and tele-immersion. Needless to say that optical fibers offer far more bandwidth than copper cables and are less susceptible to electromagnetic interference. Several fibers are bundled easily into a fiber cable and support transmissions with ultra-low bit error rates. As a result, optical fibers are currently used to alleviate the bandwidth bottleneck in the network core, whereas traditional copper cables are limited to the last mile. Advances in *wavelength-division multiplexing* (WDM) have provided abundant capacity in effect dividing the optical fiber into several channels (or wavelengths) where each wavelength can support upwards of 10 Gbps. In the first generation of optical networks, deployed primarily to support SONET/SDH, these wavelengths are provisioned manually and comprise static connections between source and destination nodes. However, as the wavelength data rate increases, the intermediate electronic processing at every node becomes problematic and limits the available bandwidth. The second generation of optical networks integrates intelligent switching and routing at the optical layer to achieve very high data throughput. All optical transparent connections, or *lightpaths*, are dynamically provisioned and routed where wavelength conversion and fiber delay lines can be used to provide spatial and temporal reuse of channels. These networks are referred to as *wavelength-routed networks* (WRNs) where several optical switching technologies exist to leverage the available wavelengths among various traffic flows while using a common signaling and control interface such as *generalized multi-protocol label switching* (GMPLS).

The rest of this article first discusses the SONET transport, which is used heavily in the core networks and presents some of the proposed enhancements for reducing SONET's inefficiencies when supporting Ethernet traffic. The WRNs are introduced, and the concepts of lightpath, traffic grooming, and waveband switching are discussed later in this article. Several switching technologies are then presented highlighting their advantages and disadvantages as well as their technical feasibility. Finally, the various control and signaling architectures in optical networks are discussed, emphasizing the interaction between the IP and optical layers as well as path protection and restoration schemes.

CURRENT TRANSPORTS

A *Synchronous Optical Network* (SONET) (1) has been the predominant transport technology used in today's wide area networks, and forms the underlying infrastructure for most voice and data services. SONET was designed to provide carrier-class service (99.999%) reliability, physical layer connectivity, and self-healing. Time-division multiplexing (TDM) standards are used to multiplex different signals that are transported optically and switched electronically using timeslots to identify destinations. SONET is deployed generally in ring networks to provide dedicated point-to-point connections between any two source and destination node pairs. Two common ring architectures with varying degrees of protection exist: *unidirectional path switched rings* (UPSRs) and *bi-directional line switched rings* (BLSRs). These rings are often built using *add/drop multiplexers* (ADMs) to terminate connections and multiplex/demultiplex client signals onto/from the SONET frames, which implies that the optical signal undergoes optical–electro–optical (O–E–O) conversions at every intermediate hop.

The fundamental frame in traditional and optical SONET is the *Synchronous Transport Signal level-1* (STS-1), which consists of 810 bytes often depicted as a 90-column by 9-row structure. With a frame length of 125 μs (or 8000 frames per second), STS-1 has a bit rate of 51.84 Mbps. The STS-1 frame can be divided into two main areas: the transport overhead and the synchronous payload envelope (SPE). A SONET connection is provisioned by the *contiguous concatenation* (CC) of STS-1 frames to achieve the desired rate. SONET framing has proven inefficient for transporting bursty data traffic, because additional protocols, such as asynchronous

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Table 1. Wasted Bandwidth

Application	Data rate	NG-SONET	SONET
Ethernet	10 Mbps	12%	80%
Fast Ethernet	100 Mbps	1%	33%
Gigabit Ethernet	1 Gbps	5%	58%

transfer mode (ATM), are required to adapt the various traffic flows, such as constant bit rate (voice) and unspecified bit rate (data) connections, to the SPE containers. The layering of ATM on SONET has worked adequately but proved to be inefficient when carrying data traffic; the ATM layering incurs about 15% overhead. In addition, because of the SPE fixed-size, SONET connections are provisioned to support the peak data rates, which often leads to over-allocation especially when the traffic load is low.

A more recent approach is the *next-generation SONET* (NG-SONET) specification (2), which provides carriers with the capabilities of optimizing the bandwidth allocation and using any unused and fragmented capacity across the ring network, to better match the client data rate as shown in Table 1.

The new protocols required in NG-SONET are the *generic framing procedure* (GFP), *virtual concatenation* (VC), and *link capacity adjustment scheme* (LCAS). GFP is an encapsulation mechanism used to adapt different services to the SONET frames. The GFP specification provides for single-and multi-bit header error correction capabilities and channel identifiers for multiplexing client signals and data packets onto the same SONET SPE. VC is the main component of NG-SONET. It is used to distribute traffic into several smaller containers, which are not necessarily contiguous, but can be concatenated into virtual groups (VCGs) to form the desired rate. The benefits of VC include:

1. Better network utilization: Low-order data connections can be provisioned in increments of 1.5-Mbps virtual tributary (VT-1.5) instead of STS-1 frames.
2. Noncontiguous frame allocation: This facilitates the provisioning of circuits and improves the network utilization as available, but fragmented capacity can be allocated rather than wasted.
3. Load balancing and service resiliency: Through the use of diverse paths, the source can balance the traffic load among different paths, which provides some degree of resiliency; a fiber cut will only affect some VCs, thereby degrading the service without complete interruption.

LCAS is a two-way handshake protocol that allows for dynamic provisioning and reconfiguration of VCGs to enlarge or shrink incrementally the size of a SONET connection. The basic approach is to have the Network Management System instruct the source node to resize an existing connection. The source and destination nodes coordinate the addition, deletion, and sequencing of the new or existing VC by using the H4 control bytes in the frame headers.

WAVELENGTH-ROUTED NETWORKS

Wavelength-division multiplexing offers an efficient way to exploit the enormous bandwidth of an optical fiber; WDM allows the transmission of multiple optical signals simultaneously on a single fiber by dividing the fiber transmission spectrum into several nonoverlapping wavelengths (or channels), with each wavelength capable of supporting a single communication channel operating at peak electronic processing speed. With the rapid advance and use of dense WDM (DWDM) technology, 100 or more wavelength channels can be multiplexed onto a single fiber, each operating at 10 to 100 Gbps, leading to Tbps aggregate bandwidth.

WDM optical network topologies can be divided into three basic classes based on their architecture: *(1)* point-to-point, *(2)* broadcast-and-select star, and *(3)* Wavelength-routed. In point-to-point and broadcast-and-select networks, no routing function is provided, and as such, they are simple but inflexible and only suitable for small networks. (WRNs), on the other hand, support true routing functionality and hence are highly efficient, scalable, and more appropriate for wide area networks (3). A wavelength-routed optical network is shown in Fig. 1. The network can be built using switching nodes, wavelength routers, optical cross-connects (OXCs), or optical add–drop multiplexers (OADMs) connected by fiber links to form an arbitrary physical topology. Some routing nodes are attached to access stations where data from several end users (e.g., IP, and ATM users) can be multiplexed or groomed on to a single optical channel. The access station also provides optical-to-electronic conversion and vice versa to interface the optical network with the conventional electronic network using transceivers.

Lightpath

In a WRN, end users communicate with one another via optical channels, which are referred to as lightpaths. A lightpath is an optical path (data channel) established between the source node and the destination node by using

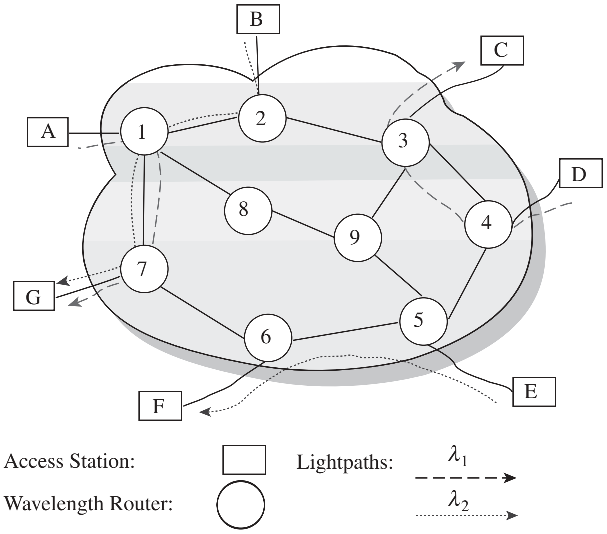


Figure 1. Structure of WRNs.

a dedicated wavelength along multiple intermediate links. Because of the limited number of wavelength channels, wavelength reuse is necessary when the network has to support a large number of users. Wavelength reuse allows the same wavelength to be reused in spatially disjoint parts of the network. In particular, any two lightpaths can use the same wavelength as long as they do not share any common links. For example, Fig. 1 shows the lightpaths A->1->7->G and D->4->3->C, which can use the same wavelength λ_1. However, two or more lightpaths traversing the same fiber link must be on different wavelengths so that they do not interfere with one another. Hence, the lightpaths A->1->7->G and B->2->1->7->G cannot use the same wavelength simultaneously because they have common links.

Without wavelength conversion, a lightpath is required to be on the same wavelength channel throughout its path in the network. This is known as the wavelength-continuity constraint. Another important constraint is the wavelength-capacity constraint, which is caused by the limited number of wavelength channels and transmitters/receivers in a network. Given these two constraints, a challenging and critical problem in WRNs is to determine the route that the lightpath should traverse and which wavelength should be assigned. This is commonly referred to as the routing and wavelength assignment (RWA) problem, which has been shown to be NP-complete for static traffic demands.

Optical Cross-Connects

A key node in WRNs is the OXC, which allows dynamic setup and tear down of lightpaths as needed (i.e., without having to be provisioned statically). The OXCs can connect (i.e., switch) any input wavelength channel from an input fiber port to any one of the output fiber ports in optical form. An OXC consists of optical switches preceded by wavelength demultiplexers and followed by wavelength multiplexers. Thus, in an OXC, incoming fibers are demultiplexed into individual wavelengths, which are switched to corresponding output ports and are then multiplexed onto outgoing fibers. By appropriately configuring the OXCs along the physical path, lightpaths can be established between any pair of nodes (4).

Wavelength Conversion

The function of a wavelength converter is to convert signals from an input wavelength to a different output wavelength. Wavelength conversion is proven to be able to improve the channel utilization in WRNs because the wavelength-continuity constraint can be relaxed if the OXCs are equipped with wavelength converters. For example, in Fig. 1, if node 4 has wavelength converters, lightpath E->5->4->3->C can be established using a different wavelength on link 5->4 and 4->3 (e.g., using wavelength λ_1 on links E->5->4, and a different wavelength, say λ_2, on links 4->3->C).

Wavelength conversion can be implemented by *(1)* (O–E–O) wavelength conversion and *(2)* all-optical wavelength conversion. When using O–E–O wavelength conversion, the optical signal is first converted into the electronic domain. The electronic signal is then used to drive the input of a tunable laser, which is tuned to the desired output wavelength. This method can only provide opaque data transmission (i.e., data bit rate and data format dependent), which is very complex while consuming a huge amount of power. On the other hand, no optical-to-electronic conversion is involved in the all-optical wavelength conversion techniques and the optical signal remains in the optical domain throughout the conversion process. Hence, all-optical wavelength conversion using techniques such as wave-mixing and cross-modulation are more attractive. However, all-optical technologies for wavelength conversion are still not mature, and all-optical wavelength converters are likely to remain very costly in the near future. Therefore, a lot of attention has been focused on compromising schemes such as limited number, limited range, and sparse wavelength conversion to achieve high network performance (5,6).

Traffic Grooming

Although WDM transmission equipment and OXCs enable the establishment of lightpaths operating at a high rate (currently at 10 Gb/s, or OC-192, expected to grow to 40 Gb/s, or OC-768), only a fraction of end users are expected to have a need for such high bandwidth that uses a full wavelength. Most users typically generate lower speed traffic, such as OC-12 (622 Mbps), OC-3 (155 Mbps), and OC-1 (51.84 Mbps), using SONET framing. Hence, the effective packing of these sub-wavelength tributaries onto high-bandwidth full-wavelength channels (i.e., by appropriate routing or wavelength and time-slot assignment) is a very important problem and is known as the traffic grooming problem. For example, 64 OC-3 circuits can be groomed onto a single OC-192 wavelength.

Traffic grooming has received considerable attention recently (7,8). In WDM SONET, an ADM is used to multiplex (combine) low-speed SONET streams into a higher speed traffic stream before transmission. Similarly, an ADM receives a wavelength from the ring and demultiplexes it into several low-speed streams. Usually, an ADM for a wavelength is required at a particular node only when the wavelength must be added or dropped at that particular node. However, the cost of ADMs often makes up a significant portion of the total cost, and as such, an objective of traffic grooming is to achieve efficient utilization of network resources while minimizing the network cost and the number of ADMs required. Similarly, in mesh network topologies, the number of electronic ADMs and the network cost can be reduced by carefully grooming the low-speed connections and using OXCs for bypass traffic.

Waveband Switching

The rapid advance in dense WDM technology and worldwide fiber deployment have brought about a tremendous increase in the size (i.e., number of ports) of OXCs, as well as the cost and difficulty associated with controlling such large OXCs. In fact, despite the remarkable technological advances in building photonic cross-connect systems and associated switch fabrics, the high cost (both capital and operating expenditures) and unproven reliability of large switches (e.g., with 1000 ports or more) have not justified their deployment. Recently, *waveband switching* (WBS) in conjunction with new *multigranular optical cross-connects* (MG-OXCs) have been proposed to reduce this cost and complexity (9–11). The main idea of WBS is to group several wavelengths together as a band, switch the band using a single port whenever possible (e.g., as long as it carries only bypass or express traffic), and demultiplex the band to switch the individual wavelengths only when some traffic needs to be added or dropped. A complementary hardware is an MG-OXC that not only can switch traffic at multiple levels such as fiber, waveband, and individual wavelength (or even sub wavelength), but also it can add and drop traffic at multiple levels, as well as multiplex and demultiplex traffic from one level to another. By using WBS in conjunction with MG-OXCs, the total number of ports required in such a network to support a given amount of traffic is much lower than that in a traditional WRN that uses ordinary OXCs (that switch traffic only at the wavelength level). The reason is that 60% to 80% of traffic simply bypasses the nodes in the backbone, and hence, the wavelengths carrying such transit traffic do not need to be individually switched in WBS networks (as opposed to WRNs wherein every such wavelength still has to be switched using a single port). In addition to reducing the port count, which is a major factor contributing to the overall cost of switching fabrics, the use of bands can reduce complexity, simplify network management, and provide better scalability.

WBS is different from wavelength routing and traditional traffic grooming in many ways. For example, techniques developed for traffic grooming in WRNs, which are useful mainly for reducing the electronic processing and/or the number of wavelengths required, cannot be applied directly to effectively grouping wavelengths into wavebands. This restriction is because in WRNs, one can multiplex just about any set of lower bit rate (i.e., subwavelength) traffic such as OC-1s into a wavelength, which is subject only to the wavelength-capacity constraint. However, in WBS networks, at least one more constraint exists: Only the traffic carried by a fixed set of wavelengths (typically consecutive) can be grouped into a band.

SWITCHING

Similar to the situation in electronic network, three underlying switching technologies for optical networks exist: *optical circuit switching* (usually referred to as *wavelength routing* in the literature), *optical packet switching, and optical burst switching* as shown in Fig. 2.

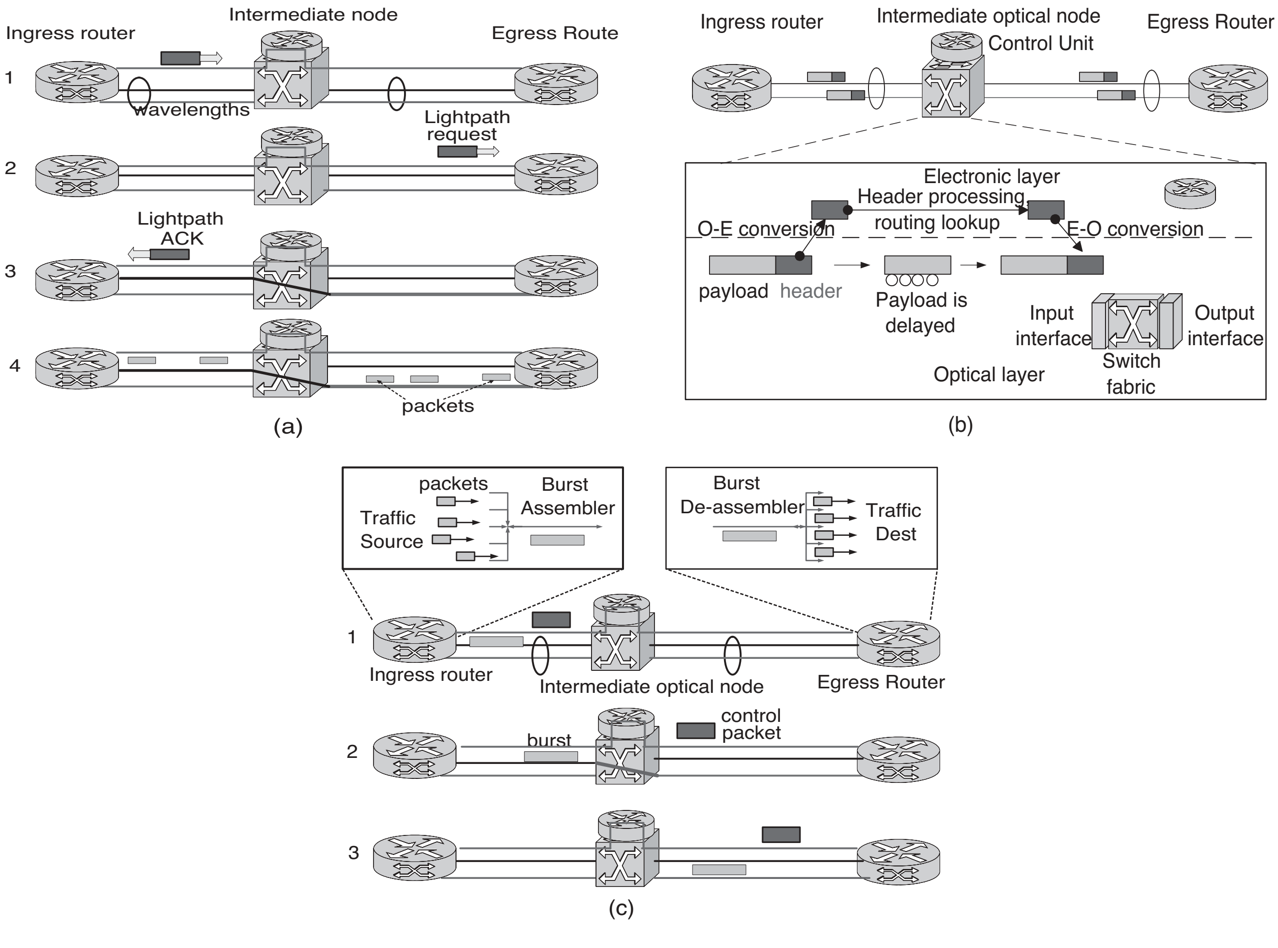


Figure 2. Optical switching: (a) OCS, (b) OPS, and (c) OBS.

Optical Circuit Switching

Optical circuit switching (OCS) takes a similar approach as in the circuit-switched telecommunication networks. To transport traffic, lightpaths are set up among client nodes (such as IP routers or ATM switches) across the optical network, with each lightpath occupying one dedicated wavelength on every traversed fiber link. The lightpaths are treated as logical links by client nodes, and the user data are then transported over these lightpaths. Figure 2(a) illustrates the lightpath setup procedure and the data packet's transport.

The major advantage of OCS is that data traffic is transported purely in the optical domain, and the intermediate hop-by-hop electronic processing is eliminated, in addition to other benefits such as protocol transparency, quality of service (QoS) support, and traffic engineering. In addition, OCS is only suitable for large, smooth, and long duration traffic flows but not for bursty data traffic. However, OCS does not scale well; to connect N client nodes, $O(N^2)$ lightpaths are needed. Moreover, the total number of lightpaths that can be set up over an optical network is limited, because of the limited number of wavelengths per fiber and dedicated use of wavelengths by lightpaths. These issues have driven the research community to study alternative switching technologies.

Optical Packet Switching

To address the poor scalability and inefficiency of transporting bursty data traffic using OCS, optical packet switching (OPS) has been introduced and well studied with a long history in the literature. OPS has been motivated by the (electronic) packet switching used in IP networks where data packets are statistically multiplexed onto the transmission medium without establishing a dedicated connection. The major difficulty of OPS is that the optical layer is a dumb layer, which is different from the intelligent electronic layer. Specifically, the data packet cannot be processed in the optical layer, but it has to be processed in the electronic layer to perform routing and forwarding functionality. To reduce processing overhead and delay, only the packet header is converted from the optical layer to the electronic layer for processing (12).

As illustrated in Fig. 2(b), the OPS node typically has a control unit, input and output interfaces, and a switch fabric. The input interface receives incoming packets from input ports (wavelengths) and performs wavelength division multiplexing in addition to many other functionalities, such as 3R (reamplification, reshaping, and retiming) regeneration to restore incoming signal quality, packet delineation to identify the beginning/end of the header and payload of a packet, and optical–to–electronic (O–E) conversion of the packet header. The control unit processes the packet header, looks up the routing/forwarding table to determine the next hop, and configures the switch fabric to switch the payload, which has been delayed in fiber delay lines (FDLs) while the header is being processed. The updated packet header is then converted back to the optical layer and combined with the payload at the output interface, which performs wavelength-division multiplexing, 3R regeneration, and power equalization before transmitting the packet on an output port wavelength to the next hop.

Packet loss and contention may occur in the control unit when multiple packets arrive simultaneously. In this case, the packet headers are buffered generally and processed one-by-one. Furthermore, the contention may also happen on the payloads when multiple packets from different input ports need to be switched to the same output wavelength. Note that a payload contention results in a packet header contention but not vice versa. The payload contentions can be resolved in the optical layer using FDLs, wavelength conversion, or deflection routing. FDLs delay a payload for deterministic periods of time, and thus, multiple payloads going to the same output port simultaneously can be delayed with different periods of time to resolve the contention. Wavelength conversion can switch multiple payloads to different output wavelengths. Deflection routing deflects a contending payload to an alternative route at a different output port. Note that deflection routing should be attempted last because it may cause contentions on the alternative paths and may lead to unordered packet reception (13).

OPS may take a *synchronous or asynchronous* approach. The former approach is more feasible technically (most research efforts have focused on this approach), where packets are of fixed length (like ATM cells) and the input/output interfaces perform packet synchronization in addition to other functions. In the asynchronous approach, the packets are of variable length and the packet processing in the input/output interface is asynchronous as well. This task is highly demanding with existing technologies and is expected to be viable in the very long term. A recent development of OPS is the *optical label switching*, which is a direct application of *multi-protocol label switching* (MPLS) technology. MPLS associates a label to a forwarding equivalence class (FEC), which as far as the optical network is concerned, can be considered informally as a pair of ingress/egress routers. The packet header contains a label instead of the source/destination address, and the data forwarding is performed based on the label, instead of the destination address. A major benefit of label switching is the speedup for the routing/forwarding table lookup, because the label is of fixed length and is easier to handle than the variable length destination address prefix.

Optical Burst Switching

Although OPS may be beneficial in the long run, it is difficult to build a cost-effective OPS nodes using the current technologies, primarily caused by the lack of "optical" random access memory and the strict synchronization requirement (the asynchronous OPS is even more difficult). Optical burst switching (OBS) has been proposed as a novel alternative to combine the benefit of OPS and OCS while eliminating their disadvantages (14–16). Through statistical multiplexing of bursts, OBS significantly improves bandwidth efficiency and scalability over OCS. In addition, when compared with OPS, the "optical" random access memory and/or fiber delay lines are not required in OBS (although having them would result in a better performance), and the synchronization is less strict. Instead of processing, routing, and forwarding each packet, OBS assembles multiple packets into a burst at the ingress router and the burst is switched in the network core as one unit. In other words, multiple packets are *bundled* with only one control packet (or header) to be processed, resulting in much less control overhead. The burst assembly at the ingress router may employ some simple scheme, for example, assembling packets (going to the same egress router) that have arrived during a fixed period into one burst or simply assembling packets into one burst until a certain burst length is reached, or uses more sophisticated schemes, for example, capturing an higher layer protocol data unit (PDU), for example, a TCP segment, into a burst.

Different OBS approaches primarily fall into three categories: *reserve-a-fixed-duration* (RFD), *tell-and-go* (TAG), or *in-band-terminator* (IBT). Figure. 2(c) illustrates a RFD-based protocol, called *just enough time* (JET), where each ingress router assembles incoming data packets that are destined to the same egress router into a burst, according to some burst assembly scheme (14). For each burst, a control packet is first sent out on a control wavelength toward the egress router and the burst follows the control packet (on a separate data wavelength) after an offset time. This offset time can be made no less than the total processing delay to be encountered by the control packet, to reduce the need for fiber delay lines, and at the same time much less than the round-trip propagation delay between the ingress and the egress routers. The control packet, which goes through optical-electronic-optical conversion at every intermediate node, just as the packet header in OPS does, attempts to reserve a data wavelength for just enough time (specifically, between the time that the burst arrives and departs), to accommodate the succeeding burst. If the reservation succeeds, the switch fabric is configured to switch the data burst. However, if the reservation fails, because no wavelength is available at the time of the burst arrival on the outgoing link, the burst will be dropped (retransmissions are handled by higher layers such as TCP). When a burst arrives at the egress router, it is disassembled into data packets that are then forwarded toward their respective destinations.

In the IBT-based OBS, the burst contains an IBT (e.g., silence of a voice circuit) and the control packet may be sent in-band preceding the burst or out-of-band over a control wavelength (before the burst). At an intermediate node, the bandwidth (wavelength) is reserved as soon as the control packet is received and released when the IBT of the burst is detected. Hence, a key issue in the IBT-based OBS is to *detect optically* the IBT. The TAG-based OBS is similar to circuit switching; a *setup* control packet is first sent over a control wavelength to reserve bandwidth for the burst. The burst is then transmitted without waiting for the acknowledgment that the bandwidth has been reserved successfully at intermediate nodes. A *release* control packet can be sent afterward to release the bandwidth, or alternatively the intermediate nodes automatically release the bandwidth after a timeout interval if they have not received a *refresh* control packet from the ingress router.

Similar to OPS, the contentions may happen in OBS when multiple bursts need to go to the same output port, or when multiple control packets from different fibers arrive simultaneously. The contention resolution techniques in OPS can be employed for OBS. Nevertheless, compared with OPS, the contention in OBS (particularly in terms of control packets) is expected to be much lighter because of the packet bundling. As far as the performance is concerned, the RFD-based OBS (e.g., using JET) can achieve higher bandwidth utilization and lower burst dropping than the IBT and TAG-based OBS, because it reserves the bandwidth for just enough time to switch the burst, and can reserve the bandwidth well in advance to reduce burst dropping.

Comparison of Optical Switching Technologies

A qualitative comparison of OCS, OPS, and OBS is presented in Table 2 where it is evident that OBS combines the benefits of OCS and OPS while eliminating their shortcomings. However, today's optical networks primarily employ OCS because of its implementation feasibility using existing technologies. OBS will be the next progression in the evolution of optical networks, whereas OPS is expected to be the long-term goal.

CONTROL AND SIGNALING

Optical networks using WDM technology provide an enormous network capacity to satisfy and sustain the exponential growth in Internet traffic. However, all end-user communication today uses the IP protocol, and hence, it has become clear that the IP protocol is the common convergence layer in telecommunication networks. It is therefore important to integrate the IP layer with WDM to transport end-user traffic over optical networks efficiently.

Control Architecture

General consensus exists that the optical network control plane should use IP-based protocols for dynamic provisioning and restoration of lightpaths within and across optical networks. As a result, it has been proposed to reuse or adapt the signaling and routing mechanisms developed for IP traffic engineering in optical networks, to create a common control plane capable of managing IP routers as well as optical switches.

Table 2. Comparison of OCS, OPS, and OBS

Technology	Bandwidth utilization	Setup latency	Implementation difficulty	Overhead	Adaptivity to bursty traffic and fault	Scalability
OCS	Low	High	Low	Low	Low	Poor
OPS	High	Low	High	High	High	Good
OBS	High	Low	Medium	Low	High	Good

Two general models have been proposed to operate an IP over an optical network. under the *domain services* model, the optical network primarily offers high bandwidth connectivity in the form of lightpaths. Standardized signaling across the *user network interface* (UNI) is used to invoke the services of the optical network for lightpath creation, deletion, modification, and status query, whereas the *network-to-network interface* (NNI) provides a method of communication and signaling among subnetworks within the optical network. Thus, the domain service model is essentially a client (i.e., IP layer)–server (i.e., optical layer) network architecture, wherein the different layers of the network remain isolated from each other (17). This is also known as the *overlay* model and is well suited for an environment that consists of multiple administrative domains, which is prevalent in most carrier networks today. On the other hand, in the *Unified Services* model, the IP and optical networks are treated as a single integrated network from a control plane view. The optical switches are treated just like any other router, and in principle, no distinction exists between UNIs and NNIs for routing and signaling purposes. This model is also known as the *peer-to-peer* model, wherein the services of the optical network are obtained in a more seamless manner as compared with the overlay model. It allows a network operator to create a single network domain composed of different network elements, thereby allowing them greater flexibility than in the overlay model. The peer model does, however, present a scalability problem because of the amount of information to be handled by any network element within an administrative domain. In addition, nonoptical devices must know the features of optical devices and vice versa, which can present significant difficulties in network operation. A third *augmented* model has also been proposed (18), wherein separate routing instances in the IP and optical domains exist but information from one routing instance is passed through the other routing instance. This model is also known as the *hybrid* model representing a middle ground between the overlay and the peer models; the hybrid model supports multiple administrative domains as in the overlay model, and supports heterogeneous technologies within a single domain as in the peer model.

Signaling

The Generalized Multi-Protocol Label Switching (GMPLS) framework (19) has been proposed as the control plane for the various architectures. Similar to traditional MPLS, GMPLS extends the IP layer routing and signaling information to the optical layer for dynamic path setup. In its simplest form, labels are assigned to wavelengths to provide mappings between the IP layer addresses and the optical wavelengths. Several extensions have been added for time slots and sets of contiguous wavelengths to support subwavelength and multiwavelength bandwidth granularities.

GMPLS signaling such as *resource reservation* (RSVP) and *constraint route label distribution* (CR-LDP) protocols map the IP routing information into preconfigured *labeled switched paths* (LSPs) in the optical layer. These LSPs are generally set up on a hop-by-hop basis or specified explicitly when traffic engineering is required. GMPLS labels can be stacked to provide a hierarchy of LSP bundling and explicit routing. Despite their similar functionalities, RSVP and CR-LDP operate differently. RSVP uses PATH and RESV messages to signal LSP setup and activation. PATH messages travel from source to destination nodes and communicate classification information. RESV messages travel from destination to source nodes to reserve the appropriate resources. RSVP uses UDP/IP packets to distribute labels, and as such, it can survive hardware or software failures caused by IP rerouting. CR-LDP on the other hand assumes that the network is reliable and uses TCP/IP packets instead. It has much lower overhead than RSVP, but it cannot survive network failures quickly. The advantages and disadvantages of RSVP and CR-LDP have long been discussed and compared in the literature without a clear winner. It seems, however, that RSVP is the industry's preferred protocol because it is coupled more tightly with IP-based signaling protocols.

Protection and restoration in GMPLS involve the computation of primary and backup LSPs and fault detection and localization. Primary and backup LSP computations consider traffic engineering requirements and network constraints. LSP protection includes dedicated and shared mechanisms with varying degrees of restoration speeds. In dedicated protection (1+1), data are transmitted on the primary and backup LSPs simultaneously, and as a result, 1+1 protection offers fast restoration and recovery from failures. In shared protection (*m:n*), *m* backup LSPs are preconfigured to provide protection for *n* primary LSPs. Data traffic is switched onto the backup LSPs at the source only after a failure has been detected. As a result, *m:n* schemes are slower than dedicated protection but use considerably less bandwidth. Fault detection and management are handled by the *link management protocol* (LMP), which is also used to manage the control channels and to verify the physical connectivity.

SUMMARY

Optical networks can sustain a much higher throughput than what can be achieved by pure electronic switching/routing techniques despite the impressive progress made in electronics and electronic switching. As such, optical networking holds the key to a future of unimagined communication services where true broadband connectivity and converged multimedia can be realized. Other anticipated services such as grid computing, video on demand, and high-speed Internet connections will require large amounts of bandwidth over wide-scale deployments that will permeate optical networking undoubtedly well into the last mile. However, exploiting the phenomenal data capacity of optical networks is anything but trivial. Both traditional architectures, such as SONET, and emerging paradigms, such as WRNs, require complex architectures and protocols while providing various degrees of statistical multiplexing. Although SONET is well entrenched and understood, it is very expensive and its static nature limits its scalability and, therefore, its reach into the last mile. WRNs, on the other hand, provide far more efficient and dynamic topologies that support a significantly larger number of all-optical connections.

Clearly, WRNs present several technological challenges, the most pressing of which is how to exploit this vast optical bandwidth efficiently while supporting bursty data traffic. Techniques such as waveband switching and traffic grooming help the realization of WRNs by reducing the overall cost. Switching technologies such as OBS and OPS provide statistical multiplexing and enhance the elasticity of WRNs in support of varying traffic volumes and requirements. Common and standardized signaling interfaces such as GMPLS allow for dynamic provisioning and ease of operation and maintenance. As a result, optical networks are more scalable and flexible than their electronic counterparts and can support general-purpose and special-purpose networks while providing a high degree of protection and restoration against failures.

Given today's available technologies, optical networks are realized using OCS techniques that are less scalable and efficient than OBS and OPS when supporting data traffic. However, as optical technologies mature, the next generation of optical networks will employ OBS and OPS techniques that are far more efficient at leveraging network resources. As such, much of the current research is focused on developing switching techniques that are fast and reliable, routing protocols that are scalable and have fast convergence, and signaling protocols that exploit network-wide resources efficiently while integrating the optical and data layers seamlessly.

BIBLIOGRAPHY

1. S. Gorshe, ANSI T1X1.5, 2001–062 SONET base standard, 2001. Available:http://www.t1.org.
2. T. Hills, Next-Gen SONET, Lightreading report, 2002. Available: http://www.lightreading.com/document.asp?doc id= 14781.
3. B. Mukherjee, *Optical Communication Networks*, New York: McGraw-Hill, 1997.
4. B. Mukherjee, WDM optical communication networks: Progress and challenges, *IEEE J. Selected Areas Commun.*, **18**: 1810–1824, 2000.
5. K. C. Lee and V. O. K. Li, A wavelength-convertible optical network, *IEEE/OSA J. of Lightwave Technology*, **11**: 962–970,1993.
6. M. Kovacevic and A. Acampora, Benefits of wavelength translation in all optical clear-channel networks, *IEEE J. on Selected Areas in Communications*, **14**(5): 868–880, 1996.
7. X. Zhang and C. Qiao, An effective and comprehensive approach for traffic grooming and wavelength assignment in SONET/WDM rings, *IEEE/ACM Trans. Networking*, **8**(5): 608–617¤, 2000.
8. E. Modiano, Traffic grooming in WDM networks, *IEEE Commun. Mag.*, **38**(7): 124–129, 2001.
9. X. Cao, V. Anand, and C. Qiao, Waveband switching in optical networks, *IEEE Commun. Mag.*, **41**(4): 105–112, 2003.
10. Pin-Han Ho and H. T. Mouftah, Routing and wavelength assignment with multigranularity traffic in optical networks, *IEEE/OSA J. of Lightwave Technology,*(8): 2002.
11. X. Cao, V. Anand, Y. Xiong, and C. Qiao, A study of waveband switching with multi-layer multi-granular optical cross-connects, *IEEE J. Selected Areas Commun.*, **21**(7): 1081–1095, 2003.
12. D. K. Hunter et al., WASPNET: a wavelength switched packet network, *IEEE Commun. Magazine*, 120–129, 1999.
13. M. Mahony, D. Simeonidou, D. Hunter, and A. Tzanakaki, The application of optical packet switching in future communication networks, *IEEE Commun. Mag.*, **39**(3): 128–135, 2001.
14. C. Qiao and M. Yoo, Optical Burst Switching (OBS) - A new paradigm for an optical Internet, *J. High Speed Networks*, **8**(1): 69–84, 1999.
15. M. Yoo and C. Qiao, Just-Enough-Time (JET): a high speed protocol for bursty traffic in optical networks, *IEEE Annual Meeting on Lasers and Electro-Optics Society LEOS 1997, Technologies for a Global Information Infrastructure*, 1997, pp. 26–27.
16. J. Turner, Terabit burst switching, *J. High Speed Networks*, **8**(1): 3–16, 1999.
17. A. Khalil, A. Hadjiantonis, G. Ellinas, and M. Ali, A novel IP-over-optical network interconnection model for the next generation optical Internet, *Global Telecommunications Conference, GLOBECOM'03*, **7**: 1–5, 2003.
18. C. Assi, A. Shami, M. Ali, R. Kurtz, and D. Guo, Optical networking and real-time provisioning: an integrated vision for the next-generation Internet, *IEEE Network*, **15**(4): 36–45, 2001.
19. A. Banerjee, L. Drake, L. Lang, B. Turner, D. Awduche, L. Berger, K. Kompella, and Y. Rekhter, Generalized multiprotocol label switching: an overview of signaling enhancements and recovery techniques, *IEEE Commun. Mag.*, **39**(7):144–151, 2001.

VISHAL ANAND
The College at Brockport—
State University of
New York
Brockport, New York

XIAOJUN CAO
Georgia State University
Atlanta, Georgia

SAMI SHEESHIA
American University of Science
and Technology
Beirut, Lebanon

CHUNSHENG XIN
Norfolk State University
Norfolk, North Carolina

CHUNMING QIAO
SUNY Buffalo
Buffalo, New York

HIGH-LEVEL SYNTHESIS

INTRODUCTION

Over the years, digital electronic systems have progressed from vacuum-tube to complex integrated circuits, some of which contain millions of transistors. Electronic circuits can be separated into two groups, digital and analog circuits. Analog circuits operate on analog quantities that are continuous in value and in time, whereas digital circuits operate on digital quantities that are discrete in value and time (1). Examples of analog and digital systems are shown in Fig. 1.

Digital electronic systems (technically referred to as digital logic systems) represent information in digits. The digits used in digital systems are the 0 and 1 that belong to the binary mathematical number system. In logic, the 0 and 1 values could be interpreted as True and False. In circuits, the True and False could be thought of as High voltage and Low voltage. These correspondences set the relations among logic (True and False), binary mathematics (0 and 1), and circuits (High and Low).

Logic, in its basic shape, deals with reasoning that checks the validity of a certain proposition—a proposition could be either True or False. The relation among logic, binary mathematics, and circuits enables a smooth transition of processes expressed in propositional logic to binary mathematical functions and equations (Boolean algebra), and to digital circuits. A great scientific wealth of information exists that strongly supports the relations among the three different branches of science that lead to the foundation of modern digital hardware and logic design.

Boolean algebra uses three basic logic operations: AND, OR, and NOT. Truth tables and symbols of the logic operators AND, OR, and NOT are shown in Fig. 2. Digital circuits implement the logic operations AND, OR, and NOT as hardware elements called “gates” that perform logic operations on binary inputs. The AND-gate performs an AND operation, an OR-gate performs an OR operation, and an Inverter performs the negation operation NOT. The actual internal circuitry of gates is built with transistors; two different circuit implementations of inverters are shown in Fig. 3. Examples of AND, OR, and NOT gates of integrated circuits (ICs—also known as chips) are shown in Fig. 4. Besides the three essential logic operations, four other important operations exist: the NOR (NOT-OR), NAND (NOT-AND), Exclusive-OR (XOR), and Exclusive-NOR (XNOR).

A logic circuit is usually created by combining gates together to implement a certain logic function. A logic function could be a combination of logic variables (such as A, B, and C) with logic operations; logic variables can take only the values 0 or 1. The created circuit could be implemented using a suitable gate structure. The design process usually starts from a specification of the intended circuit; for example, consider the design and implementation of a three-variable majority function. The function *F(A, B, C)* will return a 1 (High or True) whenever the number of 1s in the inputs is greater than or equal to the number of 0s. The truthtable of *F* is shown in Fig. 5(a). The terms that make the function *F* return a 1 are F(0, 1, 1), F(1, 0, 1), F(1, 1, 0), or F(1, 1, 1). This could be alternatively formulated as in the following equation:

$$F = A'BC + AB'C + ABC' + ABC$$

In Figure 5(b), the implementations using a standard *AND–OR–Inverter* gate structure are shown. Some other specifications might require functions with more number of inputs and accordingly a more complicated design process.

The complexity of the digital logic circuit that corresponds to a Boolean function is directly related to the complexity of the base algebraic function. Boolean functions may be simplified by several means. The simplification step is usually called optimization or minimization as it has direct effects on the cost of the implemented circuit and its performance. The optimization techniques range from simple (manual) to complex (automated using a computer).

The basic hardware design steps can be summarized in the following list:

1. Specification of the required circuit.
2. Formulation of the specification to derive algebraic equations.
3. Optimization of the obtained equations
4. Implementation of the optimized equations using suitable hardware (IC) technology.

The above steps are usually joined with an essential verification procedure that ensures the correctness and completeness of each design step.

Basically, three types of IC technologies can be used to implement logic functions on Ref. 2 these are full-custom, semi-custom, and programmable logic devices (PLDs). In full-custom implementations, the designer cares about the realization of the desired logic function to the deepest details, including the gate-level and the transistor-level optimizations to produce a high-performance implementation. In semi-custom implementations, the designer uses some ready logic-circuit blocks and completes the wiring to achieve an acceptable performance implementation in a shorter time than full-custom procedures. In PLDs, the logic blocks and the wiring are ready. In implementing a function on a PLD, the designer will only decide of which wires and blocks to use; this step is usually referred to as programming the device.

The task of manually designing hardware tends to be extremely tedious, and sometimes impossible, with the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

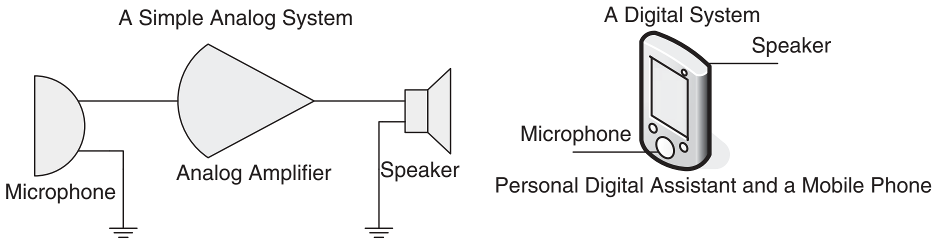


Figure 1. A simple analog system and a digital system; the analog signal amplifies the input signal using analog electronic components. The digital system can still include analog components like a speaker and a microphone; the internal processing is digital.

Input *X*	Input *Y*	Output: *X AND Y*
False	False	False
False	True	False
True	False	False
True	True	True

Input *X*	Input *Y*	Output: *X OR Y*
False	False	False
False	True	True
True	False	True
True	True	True

Input *X*	Output: *NOT X*
False	True
True	False

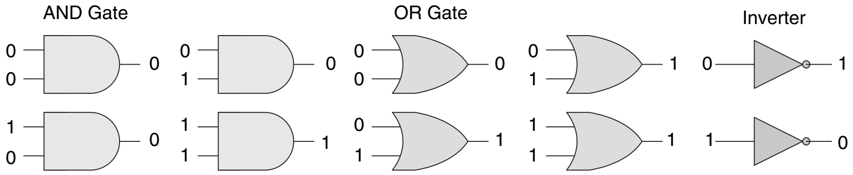
(a)

Input *X*	Input *Y*	Output: *X AND Y*
0	0	0
0	1	0
1	0	0
1	1	1

Input *X*	Input *Y*	Output: *X OR Y*
0	0	0
0	1	1
1	0	1
1	1	1

Input *X*	Output: *NOT X*
0	1
1	0

(b)



(c)

Figure 2. (a) Truth tables for AND, OR, and Inverter. (b) Truth tables for AND, OR, and Inverter in binary numbers. (c) Symbols for AND, OR, and Inverter with their operation.

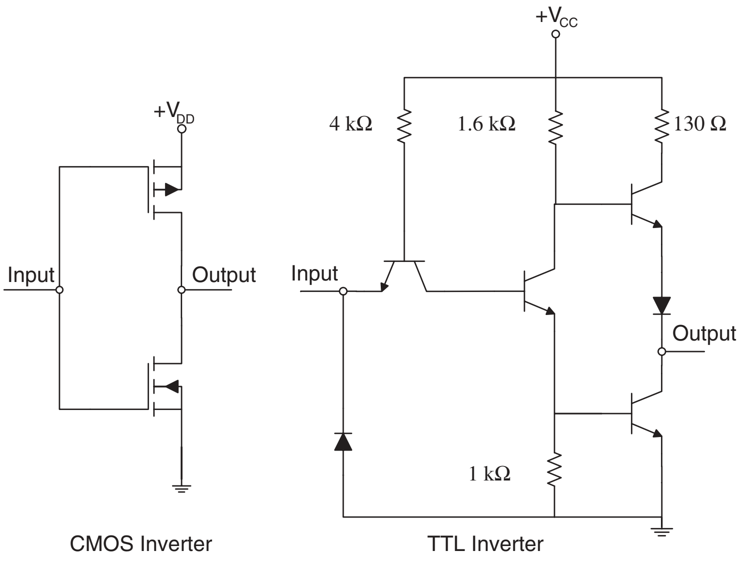


Figure 3. Complementary metal-oxide semiconductor (CMOS) and transistor-transistor logic (TTL) inverters.

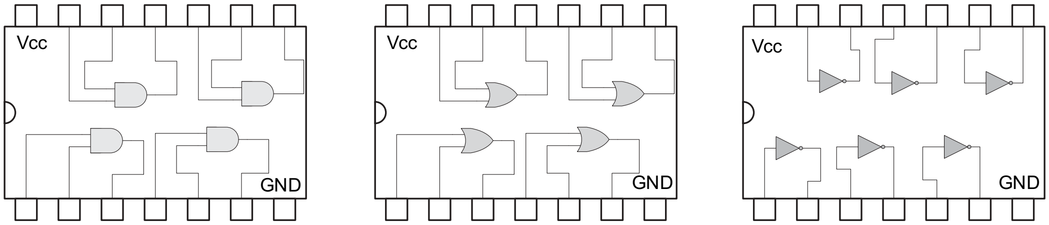
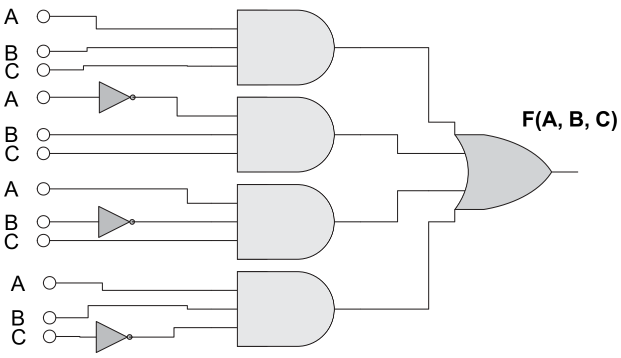


Figure 4. The 74LS21 (AND), 74LS32 (OR), and 74LS04 (Inverter) TTL ICs.

Input A	*Input B*	*Input C*	*Output F*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a)



(b)

Figure 5. (a) Truth table. (b) Standard implementation of the majority function.

increasing complexity of modern digital circuits. Fortunately, the demand on large digital systems has been accompanied with a fast advancement in IC technologies. Indeed, IC technology has been growing faster than the ability of designers to produce hardware designs. Hence, a growing interest has occurred in developing techniques and tools that facilitate the process of hardware design.

The task of making hardware design simpler has been inspired largely by the success story in facilitating the programming of traditional computers done by software designers. This success has motivated eager hardware designers to follow closely the footsteps of software designers, which leads to a synergy between these two disciplines that creates what is called hardware/software codesign.

SOFTWARE DESIGN

A computer is composed basically from a computational unit made out of logic components whose main task is to perform arithmetic and logic operations; this is usually called the arithmetic and logic unit (ALU). The computations performed by the ALU are usually controlled by a neighboring unit called the control unit (CU). The ALU and the CU construct the central processing unit (CPU) that is usually attached to a storage unit, memory unit, and input and output units to build a typical digital computer. A simplified digital computer is shown in Fig. 6.

To perform an operation using the ALU, the computer should provide a sequence of bits (machine instruction) that include signals to enable the appropriate operation, the inputs, and the destination of the output. To run a whole program (sequence of instruction), the computations are provided sequentially to the computer. As the program sizes grow, dealing with 0s and 1s becomes difficult. Efforts to facilitate dealing with computer programs concentrated on the creation of translators that hides the complexity of dealing with programming using 0s and 1s. An early proposed translator produced the binary sequence of bits (machine instructions) from easy-to-handle instructions written using letters and numbers called assembly language instructions. The translator performing the above job is called an assembler (see Fig. 7).

Before long, the limitations of assembly instructions became apparent for programs consisting of thousands of

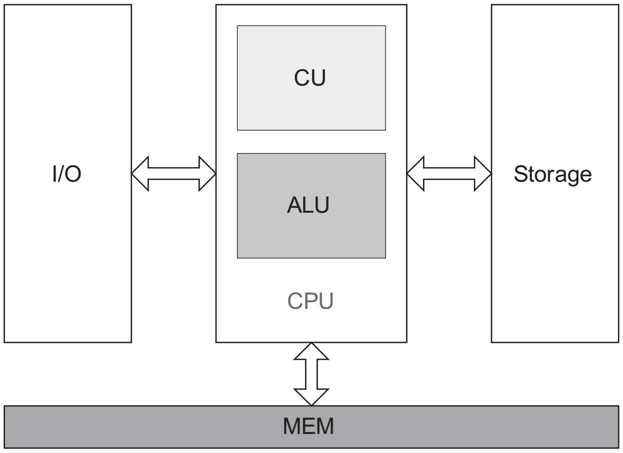
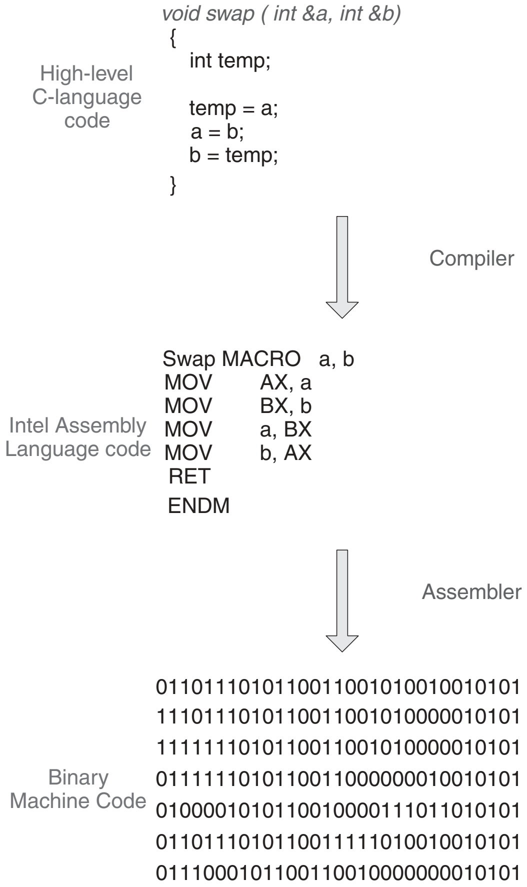


Figure 6. A typical organization of a digital computer.



```
void swap ( int &a, int &b)
{
   int temp;

   temp = a;
   a = b;
   b = temp;
}
```

```
Swap MACRO   a, b
MOV       AX, a
MOV       BX, b
MOV       a, BX
MOV       b, AX
RET
ENDM
```

```
0110111010110011001010010010101
1110111010110011001010000010101
1111111010110011001010000010101
0111111010110011000000010010101
0100001010110010000111011010101
0110111010110011111010010010101
0111000101100110010000000010101
```

Figure 7. The translation process of high-level programs.

instructions. The solution came in favor of translation again; this time the translator is called a compiler. Compilers automatically translate sequential programs, written in a high-level language like C and Pascal, into equivalent assembly instructions (see Fig. 7). Translators like assemblers and compilers helped software designers ascend to higher levels of abstraction. With compilers, a software designer can code with a fewer number of lines that are easy to understand. Then, the compiler will do the whole remaining job of translation to hide all the low-level complex details from a software designer.

TOWARD AUTOMATED HARDWARE DESIGN

Translation from higher levels of abstraction for software has motivated the creation of automated hardware design (synthesis) tools. The idea of hardware synthesis sounds very similar to that for software compilation. A designer can produce hardware circuits by automatically synthesizing an easy-to-understand description of the required circuit, to provide a list of performance-related requirements.

Several advantages exist to automating part or all of the hardware design process and to moving automation to higher levels of abstraction. First, automation assures a much shorter design cycle and time to market for the produced hardware. Second, automation allows for more exploration of different design styles because different designs can be synthesized and evaluated within a short time. Finally, with well-developed design automation tools, they may outperform human designers in generating high-quality designs.

HARDWARE DESIGN APPROACHES

Two different approaches emerged from the debate over ways to automate hardware design. On the one hand, the capture-and-simulate proponents believe that human designers have good design experience that cannot be automated. They also believe that a designer can build a design in a bottom-up style from elementary components such as transistors and gates. Because the designer is concerned with the deepest details of the design, optimized and cheap designs could be produced. On the other hand, the describe-and-synthesis advocates believe that synthesizing algorithms can outperform human designers. They also believe that a top-down fashion would be better suited for designing complex systems. In describe-and-synthesize methodology, the designers first describe the design. Then, computer aided design (CAD) tools can generate the physical and electrical structure. This approach describes the intended designs using special languages called hardware description languages (HDLs). Some HDLs are very similar to traditional programming languages like C and Pascal (3).

Both design approaches may be correct and useful at some point. For instance, circuits made from replicated small cells (like memory) are to perform efficiently if the cell is captured, simulated, and optimized to the deepest-level components (such as transistors). Another complicated heterogeneous design that will be developed and mapped onto a ready prefabricated device, like a PLD where no optimizations are possible on the electronics level, can be described and synthesized automatically. However, modern synthesis tools are well equipped with powerful automatic optimization tools.

HIGH-LEVEL HARDWARE SYNTHESIS

Hardware synthesis is a general term used to refer to the processes involved in automatically generating a hardware design from its specification. High-level synthesis (HLS) could be defined as the translation from a behavioral description of the intended hardware circuit into a structural description similar to the compilation of programming languages (such as C and Pascal) into assembly language. The behavioral description represents an algorithm, equation, and so on, Whereas a structural description represents the hardware components that implement the behavioral description. Despite the general similarity between hardware and software compilations, hardware synthesis is a multilevel and complicated task. In software compilation, you translate from a high-level language to a low-level language, Whereas in hardware synthesis, you step through a series of levels.

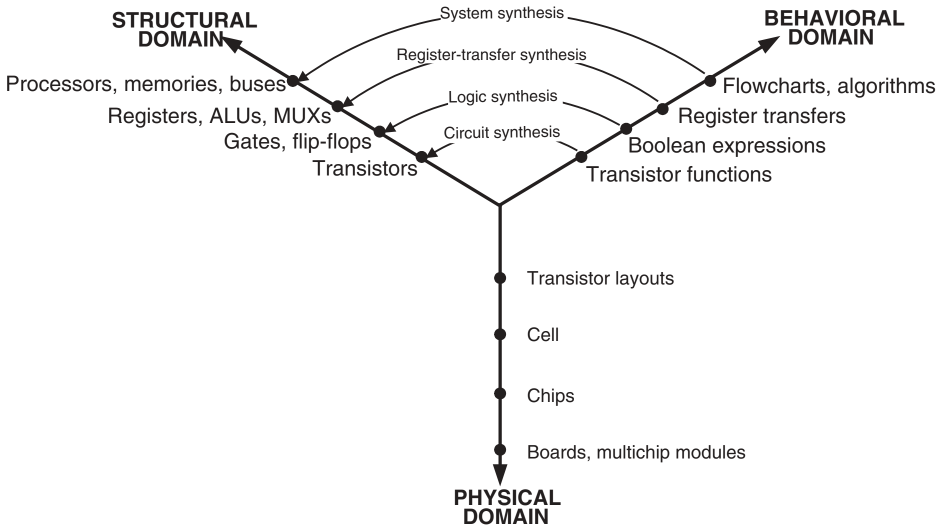


Figure 8. Gajski's Y-chart.

To explain more on behavior, structure, and their correspondences, Fig. 8 shows Gajski's Y-chart. In this chart, each axis represents a type of description (behavioral, structural, and physical). On the behavioral side, the main concern is for algorithms, equations, and functions but not for implementations. On the structural side, implementation constructs are shown; the behavior is implemented by connecting components with known behavior. On the physical side, circuit size, component placements, and wire routes on the developed chip (or board) are the main focus.

The chained synthesis tasks at each level of the design process include system synthesis, register-transfer synthesis, logic synthesis, and circuit synthesis. System synthesis starts with a set of processes communicating though either shared variables or message passing. It generates a structure of processors, memories, controllers, and interface adapters from a set of system components. Each component can be described using a register-transfer language (RTL). RTL descriptions model a hardware design as circuit blocks and interconnecting wires. Each of these circuit blocks could be described using Boolean expressions. Logic synthesis translates Boolean expressions into a list of logic gates and their interconnections (netlist). The used gates could be components from a given library such as NAND or NOR. In many cases, a structural description using one library must be converted into one using another library (usually referred to as technology mapping). Based on the produced netlist, circuit synthesis generates a transistor schematic from a set of input–output current, voltage and frequency characteristics, or equations. The synthesized transistor schematic contains transistor types, parameters, and sizes.

Early contributions to HLS were made in the 1960s. The ALERT (4) system was developed at IBM (Armonk NY) ALERT automatically translates behavioral specifications written in APL (5) into logic-level implementations. The MIMOLA system (1976) generated a CPU from a high-level input specification (6). HLS has witnessed & considerable growth since the early 1980s, and currently, it plays a key role in modern hardware design.

HIGH-LEVEL SYNTHESIS TOOLS

A typical modern hardware synthesis tool includes HLS logic synthesis, placement, and routing steps as shown in Fig. 9. In terms of Gajski's Y-chart vocabulary, these modern tools synthesize a behavioral description into a

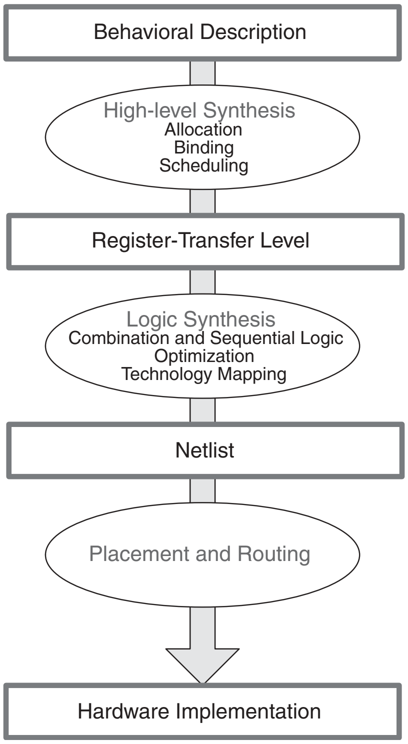


Figure 9. The process of describe-and-synthesize for hardware development.

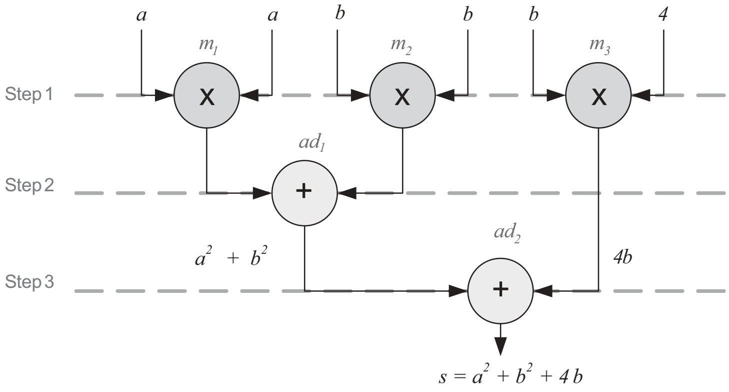


Figure 10. A possible allocation, binding, and scheduling of $s = a^2 + b^2 + 4b$.

structural network of components. The structural network is then synthesized even more, optimized, placed physically in a certain layout, and then routed through. The HLS step includes, first, allocating necessary resources for the computations needed in the provided behavioral description (the allocation stage). Second, the allocated resources are bound to the corresponding operations (the binding stage). Third, the operations order of execution is scheduled (the scheduling stage). The output of the high-level synthesizer is an RT-level description. The RT-level description is then synthesized logically to produce an optimized netlist. Gate netlists are then converted into circuit modules by placing cells of physical elements (transistors) into several rows and connecting input/output (I/O) pins through routing in the channels between the cells. The following example illustrates the HLS stages (allocation, binding, and scheduling).

Consider a behavioral specification that contains the statement, $s = a^2 + b^2 + 4b$. The variables a and b are predefined. Assume that the designer has allocated two multipliers (m_1 and m_2) and one adder (ad) for s. However, to compute s, a total of three multipliers and two adders could be used as shown in the dataflow graph in Fig. 10.

A possible binding and schedule for the computations of s are shown in Fig. 11. In the first step, the multiplier m_1 is bound with the computation of a^2, and the multiplier m_2 is bound with the computation of b^2. In the second step, m_1 is reused to compute ($4b$); also the adder (ad) is used to perform ($a^2 + b^2$). In the third and last step, the adder is reused to add ($4b$) to ($a^2 + b^2$). Different bindings and schedules are possible. Bindings and schedules could be carried out to satisfy a certain optimization, for example, to minimize the number of computational steps, routing, or maybe multiplexing.

HARDWARE DESCRIPTION LANGUAGES

HDLs, like traditional programming languages, are often categorized according to their level of abstraction. Behavioral HDLs focus on algorithmic specifications and support constructs commonly found in high-level imperative programming languages, such as assignment, and conditionals.

Verilog (7) and VHDL (Very High Speed Integrated Circuit Hardware Description Language) (8) are by far the most commonly used HDLs in the industry. Both of these HDLs support different styles for describing hardware, for example, behavioral style, and structural gate-level style, VHDL became IEEE Standard 1076 in 1987. Verilog became IEEE Standard 1364 in December 1995.

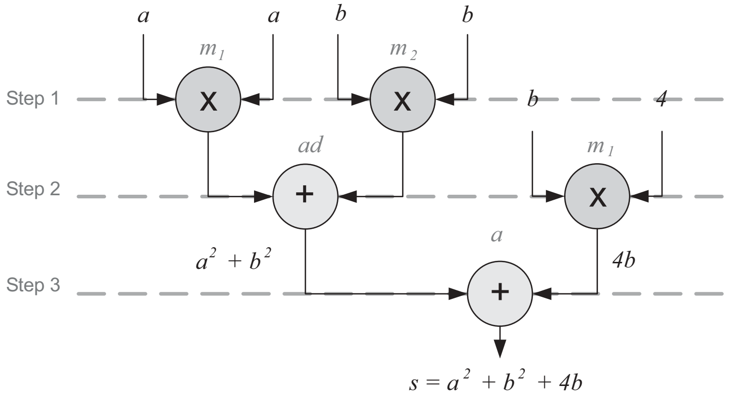


Figure 11. Another possible allocation, binding, and scheduling of $s = a^2 + b^2 + 4b$.

```
Module Half_Adder (a, b, c, s);
      input a, b;
      output c, s; //Output sum and carry.

      and Gate1 (c, a, b);     //an AND gate with two inputs a and b
                               //and one output c

      xor Gate2 (s, a, b)      //a XOR gate with two inputs a and b
                               //and one output s
endmodule
```

Figure 12. A Verilog description of a half-adder circuit.

The Verilog language uses the module construct to declare logic blocks (with several inputs and outputs). In Fig. 12, a Verilog description of a half-adder circuit is shown.

In VHDL, each structural block consists of an interface description and architecture. VHDL enables behavioral descriptions in dataflow and algorithmic styles. The half-adder circuit of Fig. 12 has a dataflow behavioral VHDL description as shown in Fig. 13; a structural description is shown in Fig. 14.

Efforts for creating tools with higher levels of abstraction lead to the production of many powerful modern hardware design tools. Ian Page and Wayne Luk (9) developed a compiler that transformed a subset of Occam into a netlist. Nearly 10 years later, we have witnessed the development of Handel-C (9), the first commercially available high-level language for targeting programmable logic devices (such as field-programmable gate arrays—FPGAs).

```
entity Half_Adder is
      port (
            a: in STD_LOGIC;
            b: in STD_LOGIC;
            c: out STD_LOGIC;
            s: out STD_LOGIC);
end Half_Adder

architecture behavioral of Half_Adder is
begin

s <= (a xor b) after 5 ns;
c <= (a and b) after 5 ns;

end behavioral;
```

Figure 13. A behavioral VHDL description of a `Half_Adder`.

Handel-C is a parallel programming language based on the theories of communicating sequential processes (CSPs) and Occam with a C-like syntax familiar to most programmers. This language is used for describing computations that are to be compiled into hardware. A Handel-C program is not compiled into machine code but into a description of gates and flip-flops, which is then used as an input to FPGA design software. Investments for research into rapid development of reconfigurable circuits using Handel-C have been largely made at Celoxica (oxfordshire, united kingdom)(10). The Handel-C compiler comes packaged with the Celoxica DK Design Suite.

Almost all ANSI-C types are supported in Handel-C. Also, Handel-C supports all ANSI-C storage class specifiers and type qualifiers except volatile and register, which have no meaning in hardware. Handel-C offers additional types for creating hardware components, such as memory, ports, buses, and wires. Handel-C variables can only be initialized if they are global or if they are declared as *static* or constant. Figure 15 shows C and Handel-C types and objects in addition to the design flow of Handel-C. Types are not limited to width in Handel-C because, when targeting hardware, no need exists to be tied to a certain width. Variables can be of different widths, thus minimizing the hardware usage. For instance, if we have a variable a that can hold a value between 1 and 5, then it is enough to use 3 bits only (declared as *int* 3 a).

The notion of time in Handel-C is fundamental. Each assignment happens in exactly one clock cycle; everything

```
entity Half_Adder is
      port (
            a, b: in bit;
            c, s: out bit;);
end Half_Adder

architecture structural of Half_Adder is
      component AND2    port (x, y: in bit; o: out bit);
      component EXOR2   port (x, y: in bit; o: out bit);

begin

Gate1 : AND2 port map (a, b, c);
Gate2 : EXOR2 port map (a, b, s);

end structural;
```

Figure 14. A structural VHDL description of a `Half_Adder`.

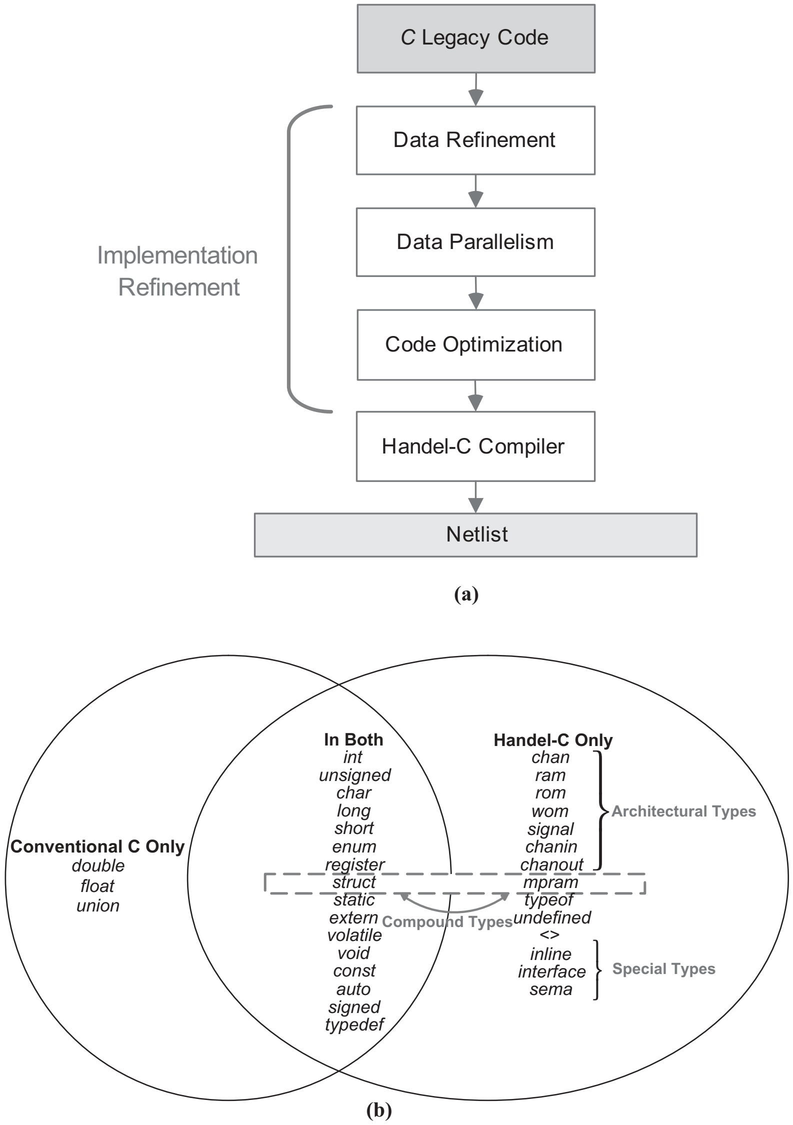


Figure 15. C and Handel-C types and objects. Handel-C types can be classified into common logic types, architectural types, compound types, and special types.

else is "free." An essential feature in Handel-C is the *par* construct, which executes instructions in parallel. Figure 16 provides an example showing the effect of using *par*.

Building on the work carried out in Oxford's Hardware Compilation Group by Page and Luk, Saul at Oxford's Programming Research Group (12) introduced a different codesign compiler, Dash FPGA-Based Systems. This compiler provides a cosynthesis and cosimulation environment for mixed FPGA and processor architectures. It compiles a C-like description to a solution containing both processors and custom hardware.

Luk and McKeever (13) introduced Pebble, a simple language designed to improve the productivity and effectiveness of hardware design. This language improves productivity by adopting reusable word-level and bit-level descriptions that can be customized by different parameter values, such as design size and the number of pipeline stages. Such descriptions can be compiled without flattening into various VHDL dialects. Pebble improves design effectiveness by supporting optional constraint descriptions, such as placement attributes, at various levels of abstraction; it also supports runtime reconfigurable designs.

Todman and Luk (14) proposed a method that combines declarative and imperative hardware descriptions. They investigated the use of Cobble language, which allows abstractions to be performed in an imperative setting. Designs created in Cobble benefit from efficient bit-level implementations developed in Pebble. Transformations are suggested to allow the declarative Pebble blocks to be used in cobble's imperative programs.

Weinhardt (15) proposes a high-level language programming approach for reconfigurable computers. This approach automatically partitions the design between hardware and software and synthesizes pipelined circuits from parallel *for* loops.

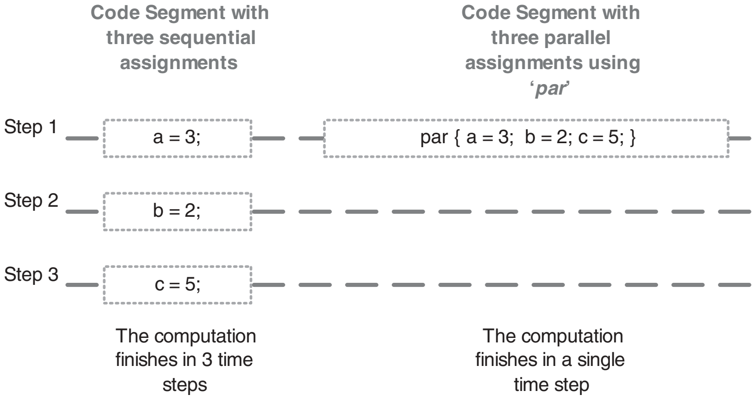

Code Segment with three sequential assignments
Code Segment with three parallel assignments using 'par'
Step 1
Step 2
Step 3
a = 3;
b = 2;
c = 5;
par { a = 3; b = 2; c = 5; }
The computation finishes in 3 time steps
The computation finishes in a single time step

Figure 16. Parallel execution using a *par* statement.

Najjar et al. (16) presented a high-level, algorithmic language and optimizing compiler for the development of image-processing applications on RC-systems. SA-C, a single assignment variant of the C programming language, was designed for this purpose.

A prototype HDL called Lava was developed by Satnam Singh at Xilinx, Inc. (San Jose, CA) and Mary Sheeran and Koen Claessen at Chalmers University in Sweden (17). Lava allows circuit tiles to be composed using powerful higher order combinators. This language is embedded in the Haskell lazy functional programming language. Xilinx's implementation of Lava is designed to support the rapid representation, implementation, and analysis of high-performance FPGA circuits.

Besides the above advances in the area of high-level hardware synthesis, the current market has other tools employed to aid programmable hardware implementations. These tools include the Forge compiler from Xilinx, the System*C* language, the Nimble compiler for Agileware architecture from Nimble Technology (now Actuate Corporation, San Mateo, CA) and Superlog.

Forge is a tool for developing reconfigurable hardware, mainly FPGAs. Forge uses Java with no changes to syntax. It also requires no hardware design skills. The Forge design suite compiles into Verilog, which is suitable for integration with standard HLS and simulation tools.

SystemC is based on a methodology that can be used effectively to create a cycle-accurate model of a system consisting of software, hardware, and their interfaces in C++. SystemC is easy to learn for people who already use C/C++. SystemC produces an executable specification, while inconsistencies and errors are avoided. The executable specification helps to validate the system functionality before it is implemented. The momentum in building the SystemC language and modeling platform is to find a proper solution for representing functionality, communication, and software and hardware implementations at various levels of abstraction.

The Nimble compiler is an ANSI-C-based compiler for a particular type of architecture called Agileware. The Agileware architecture consists of a general-purpose CPU and a dynamically configurable data path coprocessor with a memory hierarchy. It can parallelize and compile the code into hardware and software without user intervention. Nimble can extract computationally intensive loops, turn them into data flow graphs, and then compile them into a reconfigurable data path.

Superlog is an advanced version of Verilog. It adds more abstract features to the language to allow designers to handle large and complex chip designs without getting too much into the details. In addition, Superlog adds many object-oriented features as well as advanced programming construct to Verilog.

Other famous HLS and hardware design tools include Altera's Quartus (San Jose, CA), Xilinx's ISE,and Mentor Graphics's HDL Designer, Leonardo Spectrum, Precision Synthesis, and ModelSim (Wilsonville, OR).

HIGHER LEVEL HARDWARE DESIGN METHODOLOGIES

The area for deriving hardware implementations from high-level specifications has been witnessing a continuous growth. The aims always have been to reach higher levels of abstraction through correct, well-defined refinement steps. Many frameworks for developing correct hardware have been brought out in the literature (18–20).

Hoare and colleagues (20) in the Provably Correct Systems project (ProCoS) suggested a mathematical basis for the development of embedded and real-time computer systems. They used FPGAs as back-end hardware for realizing their developed designs. The framework included novel specification languages and verification techniques for four levels of development:

- Requirements definition and design.
- Program specifications and their transformation to parallel programs.
- Compilation of programs to hardware.
- Compilation of real-time programs to conventional processors.

Aiming for a short and precise specification of requirements, ProCoS has investigated a real-time logic to formalize dynamic systems properties. This logic provides a calculus to verify a specification of a control strategy based on finite state machines (FSMS). The specification language SL is used to specify program components and to support transformation to an Occam-like programming language PL. These programs are then transformed into hardware or machine code. A prototype compiler in SML has been produced, which converts a PL-like language to a netlist suitable for placement and routing for FPGAs from Xilinx.

Abdallah (19), at London South Bank University, created a step-wise refinement approach to the development of correct hardware circuits from formal specifications. A functional programming notation is used for specifying algorithms and for reasoning about them. The specifications are realized through the use of a combination of function decomposition strategies, data refinement techniques, and off-the-shelf refinements basedupon higher order functions. The off-the-shelf refinements are inspired by the operators of CSP and map easily to programs in Handel-C. The Handel-C descriptions are then compiled directly into hardware.

The development of hardware solutions for complex applications is no more a complicated task with the emergence of various HLS tools. Many areas of application have benefited from the modern advances in hardware design, such as automotive and aerospace industries, computer graphics, signal and image processing, security, complex simulations like molecular modeling, and DNA matching.

The field of HLS is continuing its rapid growth to facilitate the creation of hardware and to blur more and more the border separating the processes of designing hardware and software.

CROSS-REFERENCES

Programmable Logic Devices, see Programmable Logic Arrays.

BIBLIOGRAPHY

1. T. Floyd, *Digital Fundamentals with PLD Programming*, Englewood Cliffs, NJ: Prentice Hall, 2006.
2. F. Vahid et al., Embedded system design: *A Unified Hardware / Software Introduction*, New York: John Wiley & Sons, 2002.
3. S. Hachtel, *Logic Synthesis and Verification Algorithms*, Norwell: Kluwer, 1996.
4. T. Friedman and S. Yang, Methods used in an automatic logic design generator(ALERT), *IEEE Trans. in Comp.*, **C-18**: 593–614, 1969.
5. S. Pommier, *An Introduction to APL*, Cambridge: Cambridge University Press, 1983.
6. P. Marwedel, A new synthesis algorithm for the mimola software system, *Proc. Design Automation Conference*, 1986, pp. 271–277.
7. IEEE, *Verilog HDL language reference manual*, IEEE Standard 1364, 1995.
8. IEEE, *Standard VHDL reference manual*, IEEE Standard 1076, 1993.
9. I. Page and W. Luk, Compiling Occam into field-programmable gate arrays, *Proc. Workshop on Field Programmable Logic and Applications*, 1991, pp. 271–283.
10. I. Page, *Logarithmic Greatest Common Divisor Example in Handel-C*, Embedded Solutions, 1998.
11. Celoxica. Available: *www.celoxica.com*.
12. J. Saul. Hardware/software codesign for FPGA-based systems, *Proc. Hawaii Int'l Conf. on Sys. Sciences*, 3, 1999, p. 3040.
13. W. Luk and S. McKeever, Pebble: a language for parameterized and reconfigurable hardware design, *Proc. of Field Programmable Logic and Apps.*, 1482, 1998, p. 9–18.
14. T. Todman and W. Luk, Combining imperative and declarative hardware descriptions, *Proc. Hawaii Int'l Conf. on Sys. Sciences*, 2003, p. 280.
15. M. Weinhardt, Portable pipeline synthesis for FCCMs, *Field Programmable Logic: Smart Apps., New paradigms and compilers*, 1996, p. 1–13,
16. W. Najjar, B. Draper, W. Bohm, and R. Beveridge, The Cameron project: high-level programming of image processing applications on reconfigurable computing machines, *Workshop on Reconfigurable Computing*, 1998.
17. K. Claessen, *Embedded Languages for Describing and Verifying Hardware*. PhD Thesis, Göteborg, Sweden: Chalmers University of Technology and Göteborg University, 2001.
18. J. Bowen, M. Fränzle, E. Olderog, and A. Ravn, Developing correct systems, *Proc. Euromicro Workshop on RT Systems*, 1993, pp. 176–187.
19. A. E. Abdallah. Derivation of parallel algorithms: From functional specifications to CSP processes. In B. Moller (ed), *Proceedings of Mathematics of Program Construction, Vol. 947 of Lecture Notes in Computer Science*, Springer-Verlag, 1994, pp. 67–96.
20. J. Bowen, C. A. R. Hoare, H. Langmaack, E. Olderog, and A. Ravn, A ProCoS II project final report: ESPRIT basic research project 7071, *Bull. European Assoc. Theoret. Comp. Sc.*, **59**, 76–99, 1996.

FURTHER READING

T. Floyd, *Digital Fundamentals with PLD Programming*, Englewood Cliffs, NJ: Prentice Hall, 2006.

M. Mano et al., *Logic and Computer Design Fundamentals*, Englewood Cliffs, NJ: Prentice Hall, 2004.

F. Vahid et al., *Embedded System Design: A Unified Hardware / Software Introduction*, New York: John Wiley & Sons, 2002.

S. Hachtel, *Logic Synthesis and Verification Algorithms*, Norwell: Kluwer, 1996.

Issam W. Damaj
Dhofar University
Sultanate of Oman

I

INSTRUCTION SETS

A computer system's instruction set reflects the most primitive set of commands directly accessible to the programmer or compiler. Instructions in the instruction set manipulate components defined in the computer's instruction set architecture (ISA), which encompasses characteristics of the central processing unit (CPU), register set, memory access structure, and exception handling mechanisms.

In addition to defining the set of commands that a computer can execute, an instruction set specifies the format of each instruction. An instruction is divided into various fields that indicate the basic command (opcode) and the operands to the command. Instructions should be chosen and encoded so that frequently used instructions or instruction sequences execute quickly. Often there is more than one implementation of an instruction set architecture, which enables computer system designers to exploit faster technology and components while maintaining object code compatibility with previous versions of the computer system.

INSTRUCTION SET BASICS

Instructions contain an opcode—the basic command to execute, including the data type of the operands—and some number of operands, depending on hardware requirements. Historically, some or all of the following operands have been included: one or two data values to be used by the operation (source operands), the location where the result of the operation should be stored (destination operand), and the location of the next instruction to be executed. Depending on the number of operands, these are identified as one-, two-, three-, and four-address instructions. The early introduction of the special hardware register, the program counter, quickly eliminated the need for the fourth operand.

Types of Instructions

There is a minimum set of instructions that encompasses the capability of any computer:

- Add and subtract (arithmetic operations).
- Load and store (data movement operations).
- Read and write (input/output operations).
- An *unconditional* branch or jump instruction.
- A minimum of two *conditional* branch or jump instructions [e.g., BEQ (branch if equal zero) and BLT (branch if less than zero) are sufficient].
- A halt instruction.

Early computers could do little more than this basic instruction set. As machines evolved and changed, greater hardware capability was added, e.g., the addition of multiplication and division units, floating point units, multiple registers, and complex instruction decoders. Instruction sets expanded to reflect the additional hardware capability by combining frequently occurring instruction sequences into a single instruction. The expanding CISCs continued well into the 1980s until the introduction of RISC machines changed this pattern.

Classes of Instruction Set Architectures

Instruction sets are often classified according to the method used to access operands. ISAs that support memory-to-memory operations are sometimes called SS architectures (for Storage–Storage), whereas ISAs that support basic arithmetic operations only in registers are called RR (Register–Register) architectures.

Consider an addition, $A = B + C$, where the values of A, B, and C have been assigned memory locations 100, 200, and 300, respectively. If an instruction set supports three-address memory-to-memory instructions, a single instruction,

```
Add A, B, C
```

would perform the required operation. This instruction would cause the contents of memory locations 200 and 300 to be moved into registers in the arithmetic logic unit (ALU), the add performed in the ALU, and then the result stored into location 100.

However, it is unlikely that an instruction set would provide this three-address instruction. One reason is that the instruction requires many bytes of storage for all operand information and, therefore, is slow to load and interpret. Another reason is that later operations might need the result of the operation (for example, if $B + C$ were a subexpression of a later, more complex expression), so it is advantageous to retain the result for subsequent instructions to use.

A two-address register-to-memory alternative might be as follows:

```
Load   R1, B  ; R1 := B
Add    R1, C  ; R1 := R1 + C
Store  A, R1  ;  A := R1
```

whereas a one-address alternative would be similar, with the references to Rl (register 1) removed. In the latter scheme, there would be only one hardware register available for use and, therefore, no need to specify it in each instruction. (Example hardwares are the IBM 1620 and 7094.)

Most modern ISAs belong to the RR category and use general-purpose registers (organized either independently or as stacks) as operands. Arithmetic instructions require that at least one operand is in a register while "load" and "store" instructions (or "push" and "pop" for stack-based

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

machines) copy data between registers and memory. ISAs for RISC machines require both operands to be in registers for arithmetic instructions. If the ISA defines a register file of some number of registers, the instruction set will have commands that access, compute with, and modify all of those registers. If certain registers have special uses, such as a stack pointer, instructions associated with those registers will define the special uses.

The various alternatives that ISAs make available, such as

- both operands in memory,
- one operand in a register and one in memory,
- both operands in registers,
- implicit register operands such as an accumulator, and
- indexed *effective address* calculation, for $A[i]$ sorts of references,

are called the *addressing modes* of an instruction set. Addressing modes are illustrated below with examples of addressing modes supported by specific machines.

Issues in Instruction Set Design

There are many tradeoffs in designing an efficient instruction set. The code density, based on the number of bytes per instruction and number of instructions required to do a task, has a direct influence on the machine's performance. The architect must decide what and how many operations the ISA will provide. A small set is sufficient, but it leads to large programs. A large set requires a more complex instruction decoder. The number of operands affects the size of the instruction. A typical, modern instruction set supports 32-bit words, with 32-bit address widths, 32-bit operands, and dyadic operations, with an increasing number of ISAs using 64-bit operands. Byte, half-word, and double-word access are also desirable. If supported in an instruction set, additional fields must be allocated in the instruction word to distinguish the operand size.

The number of instructions that can be supported is directly affected by the size of the opcode field. In theory, $2^n - 1$ (a 0 opcode is never used), where n is the number of bits allocated for the opcode, is the total number of instructions that can be supported. In practice, however, a clever architect can extend that number by using the fact that some instructions, needing only one operand, have available space that can be used as an "extended" opcode. See the representative instruction sets in below for examples of this practice.

Instructions can either be fixed size or variable size. Fixed-size instructions are easier to decode and execute but either severely limit the instruction set or require a very large instruction size, i.e., wasted space. Variable-size instructions are more difficult to decode and execute but permit rich instruction sets. The actual machine word size influences the design of the instruction set. Small machine word size (see below for an example machine) requires the use of multiple words per instruction. Larger machine word sizes make single word instructions feasible. Very large machine word sizes permit multiple instructions per word (see below).

GENERAL PURPOSE ISAs

In this section, we discuss the most common categories of an instruction set. These categories include complex encoding, simplified encoding (often called load/store or reduced instruction sets), and wide instruction word formats. In the 1980s, CISC architectures were favored as best representing the functionality of high-level languages; however, later architecture designers favored RISC designs for the higher performance attained by using compiler analysis to detect instruction level parallelism. Another architectural style, VLIW, also attempts to exploit instruction level parallelism by providing multiple function units.

CISC

The CISC instruction set architecture is characterized by complicated instructions, many of which require multiple clock cycles to complete. CISC instructions typically have two-operand instructions in which one source also serves as a destination. CISC operations often involve a memory word as one operand, and they have multiple addressing modes to access memory, including indexed modes. Because the complexity of the instructions vary, instructions may have different lengths and vary in the number of clock cycles required to complete them. This characteristic makes it difficult to pipeline the instruction sequence or to execute multiple instructions in parallel.

RISC

RISC architectures were developed in response to the prevailing CISC architecture philosophy of introducing more and more complex instructions to supply more support for high-level languages and operating systems. The RISC philosophy is to use simple, fixed-size instructions that complete in a single cycle to yield the greatest possible performance (throughput and efficiency) for the RISC processor.

RISC designs may achieve instruction execution rates of more than one instruction per machine cycle. This result is accomplished by using techniques such as:

- Instruction pipelines
- Multiple function units
- Load/store architecture
- Delayed load instructions
- Delayed branch instructions

On any machine, a series of steps is required to execute an instruction. For example, these steps may be fetch instruction, decode instruction, fetch operand(s), perform operation, and store result. In a RISC architecture, these steps are *pipelined* to speed up overall execution time.

If all instructions require the same number of cycles for execution, a *full* pipeline will generate an instruction per cycle. If instructions require different numbers of cycles for execution, the pipeline will necessarily delay cycles while waiting for resources. To minimize these delays, RISC instruction sets include *prefetch* instructions to help ensure the availability of resources at the necessary point in time. The combination of pipelined execution with multiple internal function units allows a RISC processor sometimes to achieve multiple instruction execution per clock cycle.

Memory accesses require additional cycles to calculate operand address(es), fetch the operand(s), and store result(s) back to memory. RISC machines reduce the impact of these instructions by requiring that all operations be performed only on operands held in registers. Memory is then accessed only with load and store operations.

Load instructions fetch operands from memory to registers, to be used in subsequent instructions. As memory bandwidth is generally slower than processor cycle times, an operator is not immediately available to be used. The ideal solution is to perform one or more instructions, depending on the delay required for the load, which are *not* dependent on the data being loaded. This method effectively uses the pipeline, eliminating wasted cycles. The burden of generating effective instruction sequences is generally placed on a compiler, and of course, it is not always possible to eliminate all delays.

Lastly, branch instructions cause delays because the branch destination must be calculated and then that instruction must be fetched. As with load instructions, RISC designs typically use a delay on the branch instruction so they do not take effect until the one or two instructions (depending on the RISC design) immediately following the branch instruction have been executed. Again, the burden falls on the compiler to identify and move instructions to fill the one (or two) delay slots caused by this design. If no instruction(s) can be identified, a NOP (no op) has to be generated that reduces performance.

VLIW Instruction Sets

VLIW architectures are formed by using many parallel, pipelined functional units but with only a single execution thread controlling them all. The functional units are connected to a large memory and register banks using crossbars and/or busses. These elements are controlled by a single instruction stream. Each instruction has fields that control the operation of each functional unit, which enables the VLIW processor to exploit fine-grained instruction level parallelism (ILP).

Figure 1 shows a "generic" VLIW computer, and Fig. 2 shows an instruction word for such a machine.

To optimize code for a VLIW machine, a compiler may perform *trace* or *hyperblock scheduling* to identify the parallelism needed to fill the function units. Indirect memory references, generated by array indexing and pointer dereferencing can cause difficulties in the trace. These memory references must be disambiguated wherever possible to generate the most parallelism.

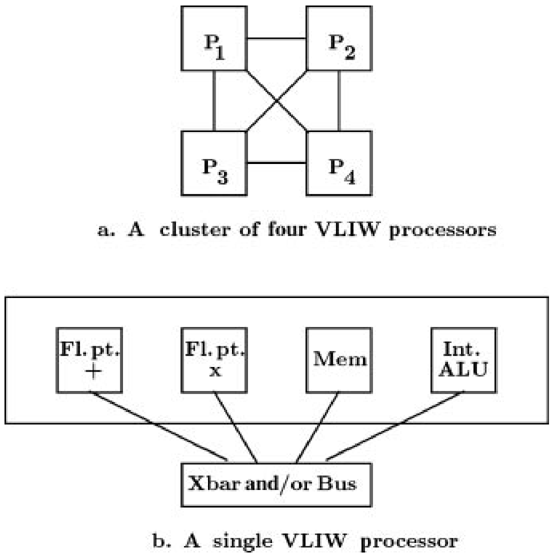

P1
P2
P3
P4
a. A cluster of four VLIW processors
Fl. pt. +
Fl. pt. x
Mem
Int. ALU
Xbar and/or Bus
b. A single VLIW processor

Figure 1. A generic VLIW machine.

SPECIALIZED INSTRUCTION SETS

The discussion so far has focused on instruction sets for most general-purpose machines. Often the basic instruction set is augmented for efficient execution of special functions.

Vector Instruction Sets

Vector architectures, such as the original Cray computers, supplement the conventional scalar instruction set with a vector instruction set. By using vector instructions, operations that would normally be executed in a loop are expressed in the ISA as single instructions. In addition to the normal fetch–decode–execute pipeline of a scalar processor, a vector instruction uses additional vector pipelines to execute the vector instructions. In a vector instruction, the vector register's set of data is pipelined through the appropriate function unit.

Categories of vector instructions include:

- Vector–vector instructions, where all operands of the instruction are vectors. An example is an add with vector registers as operands and a vector register as result.
- Vector–scalar instructions, where the content of a scalar register is combined with each element of the vector register. For example, a scalar value might be

P_1			P_2			P_3			P_4		
F+	Fx	ALU	F+	Fx	ALU	F+	Fx	ALU	F+	Fx	ALU

Figure 2. A VLIW instruction word.

multiplied by each element of a vector register and the result stored into another vector register.

- Vector–memory instructions, where a vector is loaded from memory or stored to memory.
- Vector reduction instructions, in which a function is computed on a vector register to yield a single result. Examples include finding the minimum, maximum, or sum of values in a vector register.
- Scatter–gather instructions, in which the values of one vector register are used to control vector load from memory or vector store to memory. Scatter uses an indirect addressing vector register and a base scalar register to form an effective address. Values in a data vector register corresponding to the indirect addressing vector register are stored to the calculated effective memory addresses. Similarly, a gather uses the indirect address register combined with a scalar base register to form a set of effective addresses. Data from those addresses are loaded into a vector data register.

SIMD Instruction Sets

Single instruction multiple data (SIMD) machines were popular in the late 1980s as a means of realizing massively parallel operations with relatively simple control.

Instruction sets for SIMD machines built in the 1980s such as the CM-2, DAP, and MasPar MP series are conceptually similar to vector instruction sets. SIMD instructions also operate on aggregate data. However, rather than processing multiple pairs of operands through a functional pipeline, the SIMD machine had a single instruction controller directing many identical processors, each operating in lockstep through the instruction stream. The instructions could be SS, as in the CM-2, or RR, as in the MasPar machines. As there was just one instruction stream, only the instruction controller could execute branch instructions. Conditional operation on the array of processors was accomplished through *contextualization,* meaning each processor had its own unique "context" that determined whether it executed the current instruction.

Instructions exist in a SIMD instruction set to evaluate an expression and set the context to the result of the expression evaluation. Thus, processors that evaluate the expression to true will execute subsequent instructions, whereas those that evaluate the expression to false will not. Naturally, some instructions execute regardless of the context value, so that "context" can be set and reset during computation. SIMD instruction sets usually include reduce instructions, as described above for vector machines. In addition, some SIMD machines had scan instructions, which set up variable-length vectors across the processor array on which reduced operations could be performed.

Digital Signal Processor (DSP) Instruction Sets

The architecture of a DSP is optimized for pipelined data flow. Many DSPs for embedded applications support only fixed point arithmetic; others have both fixed and floating point units; still others offer multiple fixed point units in conjunction with the floating point processor. All of these variations, of course, affect the instruction set of the DSP, determining whether bits in the instruction word are needed to specify the data type of the operands. Other distinguishing characteristics of DSP instruction sets include:

- A multiple-accumulate instruction (MAC), used for inner product calculations
- Fast basic math functions, combined with a memory access architecture optimized for matrix operations
- Low overhead loop instructions
- Addressing modes that facilitate (Fast Fourier Transform)-like memory access
- Addressing modes that facilitate table look-up.

Configurable Instruction Sets

Research into future generations of processors generalizes the notion of support for specialized operations. New designs call for *configurable hardware* to be available so new instructions can be synthesized, loaded into the configurable logic, and thus dynamically extend the processor's instruction set. The Xilinx Virtex2 Pro illustrates this concept. In conjunction with a PowerPC RISC processor, the Virtex2 Pro Integrated Circuit contains an embedded field programmable gate array (FPGA). By designing circuits for the FPGA portion of the device, a programmer can augment the instruction set of the RISC processor with arbitrary functionality. Control signals to activate the custom instructions are generated by memory-mapped writes to a communications bus that connects the RISC processor with the FPGA. Such architectures provide virtually unlimited, application-dependent extensibility to an ISA.

REPRESENTATIVE INSTRUCTION SETS

In this section, we describe six representative instruction sets in greater detail. These are the IBM System 360, the PDP-11 mini-computer, the Pentium (Intel Architecture—IA-32), the PowerPC, the IA-64 Itanium, and the Cray X1 supercomputer.

The IBM System 360

The IBM System 360, introduced in April 1964 with first delivery in April 1965, was the first of the third-generation (integrated circuit) computers. The general acceptance of a 32-bit word and 8-bit byte comes from this machine. The system 360 consisted of a series of models, with models 30, 40, 50, 65, and 75 being the best known. The model 20, introduced in November 1964, had a slightly different architecture from the others.

The 360 (any of the models) was a conventional mainframe computer, incorporating a rich, complex instruction set. The machine had 16 general-purpose registers (8 on the smaller models) and 4 floating-point registers. Instructions

Table 1. IBM System 360 Addressing Modes

addr mode	byte 1	byte 2		bytes 3–4	bytes 5–6	Notes
RR	opcode	Rl	R2		unused	Rl changes
		DS & DD	DS		unused	R2 is unchanged
					unused	
RX	opcode	Rl	X	storage ref	unused	memory is base + disp. + X
		DS or DD			unused	
					unused	
RS	opcode	Rl	R2	storage ref	unused	Rl & R2 specify
		DS or DD	DS or DD		unused	a register range
					unused	
SI	opcode	immed. data		storage ref	unused	
					unused	
SS	opcode	op1	op2	storage ref1	storage ref2	
		len	len	DD	DS	

mainly had two addresses, but 0, 1, and 3 were also permitted in some cases.

There are five addressing modes, using 2-, 4-, and 6-byte instructions. Table 1 shows these five modes. The notation in the table is (*1*) R1, R2, and X are general-purpose registers selected from the available set of 16; (*2*) R1 is either a data source (DS) *and* destination (DD) (in RR instructions) or, DS *or* DD, depending on the opcodes (other modes); (*3*) X is an index added to the specified storage reference; (*4*) a storage reference is a standard 360 memory reference consisting of a 4-bit base address and a 12-bit displacement value; (*5*) immed(iate) data are the second instruction data value and is the *actual* data value to be used, i.e., not a register or memory reference; and (*6*) op1 len and op2 len are the lengths of the of the instruction result destination and data source, respectively (op2 len is only needed for packed-decimal data).

Table 2 contains a list of 360 op codes along with the type (RR, RX, RS, SI, SS) of each operation.

DEC PDP-11

The DEC PDP-11 was a third-generation computer, introduced around 1970. It was a successor to the highly successful (also) third-generation PDP-8, introduced in 1968, which was a successor to second-generation PDP machines.

The PDP-11, and the entire PDP line, were minicomputers, where minicomputer is loosely defined as a machine with a smaller word size and memory address space, and a slower clock rate, than cogenerational mainframe computers. The PDP-11 was a 16-bit word machine, with 8 general-purpose registers (R0–R7), although R6 and R7 were “reserved” for use as the stack pointer (SP) and program counter (PC), respectively.

Instructions required one word (16 bits) immediately followed by one or two words used for some addressing modes. Instructions could be *single* operand instructions with a 10-bit opcode specifying the operation to be performed and a 6-bit destination of the result of the operation; or *double* operand instructions with a 4-bit opcode specifying the operation to be performed and two 6-bit data references for the data source and destination, respectively. Each data reference consists of a 3-bit register subfield and a 3-bit addressing mode subfield.

Instruction operands could be either a single byte or a word (or words using indirection and indexing). When the operand was a byte, the leading bit in the op code field was 1; otherwise, that bit was 0.

There are eight addressing modes, as in Table 3. The PDP-11 instruction set is given in Table 4.

Pentium Processor

The Intel Pentium series processor became the most prevalent microprocessor in the 1990s. The Pentium follows the ISA of the 80x86 (starting with 8086). It uses advanced techniques such as speculative and out-of-order execution, once used only in supercomputers, to accelerate the interpretation of the x86 instruction stream.

The original 8086 was a 16-bit CISC architecture, with 16-bit internal registers. Registers had fixed functions. Segment registers were used to create an address larger than 16 bits, so the address space was broken into 64K byte chunks. Later members of the x86 family (starting with the 386) were true 32-bit machines, with 32-bit registers and a 32-bit address space. Additional instructions in the later x86 instruction set made the register set more general purpose.

The general format of an “Intel Architecture” (IA-32) instruction is shown in Fig. 3. The instructions are a variable number of bytes with optional prefixes, an opcode, an addressing-form specifier consisting of the ModR/M and Scale/Index/Base fields (if required), address displacement of 0 – 4 bytes, and an immediate data field of 0 to 4 bytes. The instruction prefixes can be used to override default registers, operand size, and address size or to specify certain actions on string instructions. The opcode is either one or two bytes, although occasionally a third byte is encoded in the next field. The ModR/M and SIB fields have a complex encoding. In general, their purpose is to specify registers (general purpose, base, or index), addressing modes, scale factor, or additional opcode information.

Prefixes	opcode	ModR/M	SIB	Displacement	Immediate

Figure 3. Intel architecture instruction format.

Table 2. IBM System 360 Instruction Set

Command	Mnemonic	Type	Command	Mnemonic	Type
Add Register	AR	RR	Load Multiple	LM	RS
Add	A	RX	Load Negative Register	LNR	RR
Add Halfword	AH	RX	Load Negative Register (Long)	LNDR	RR
Add Logical Register	ALR	RR	Load Negative Register (Short)	LNER	RR
Add Logical	AL	RX	Load Positive Register	LPR	RR
Add Normalized (Long)	ADR	RR	Load Positive Register (Long)	LPDR	RR
Add Normalized (Long)	AD	RX	Load Positive Register (Short)	LPER	RR
Add Normalized (Short)	AER	RR	Load PSW	LPSW	SI
Add Normalized (Short)	AE	RX	Load Register (Short)	LER	RR
Add Packed	AP	SS	Load (Short)	LE	RX
Add Unnormalized Register (Long)	AWR	RR	Move Immediate	MVI	SI
Add Unnormalized (Long)	AW	RX	Move Character	MVC	SS
Add Unnormalized Register (Short)	AUR	RR	Move Numerics	MVN	SS
Add Unnormalized (Short)	AU	RX	Move with Offset	MVO	SS
AND Register	NR	RR	Move Zones	MVZ	SS
AND	N	RX	Multiply Register	MR	RR
AND Immediate	NI	SI	Multiply	M	RX
AND Character	NO	SS	Multiply Halfword	MH	RX
Branch and Link Register	BALR	RR	Multiply Register (Long)	MDR	RR
Branch and Link	BAL	RX	Multiply (Long)	MD	RX
Jzirancn on Condition rtegister	BCR	RR	Multiply Packed	MP	SS
Branch on Condition	BC	RX	Multiply Register (Short)	MER	RR
Branch on Count Register	BCTR	RR	Multiply (Short)	ME	RX
Branch on Count	BCT	RX	OR Register	OR	RR
Branch on Index High	BXH	RS	OR	O	RX
Branch on Index Low or Equal	BXLE	RS	OR Immediate	OI	SI
Compare Register	CR	RR	OR Character	OC	SS
Compare	C	RX	Pack	PACK	SS
Compare Halfword	CH	RX	Read Direct	RDD	SI
Compare Logical Register	CLR	RR	Set Program Mask	SPM	RR
Compare Logical	CL	RX	Set Storage Key	SSK	RR
Compare Logical Immediate	CLI	SI	Set System Mask	SSM	SI
Compare Logical Character	CLC	SS	Shift Left Double	SLDA	RS
Compare Register (Long)	CDR	RR	Shift Left Double Logical	SLDL	RS
Compare (Long)	CD	RX	Shift Left Single	SLA	RS
Compare Packed	CP	SS	Shift Left Single Logical	SLL	RS
Compare Register (Short)	CER	RR	Shift Right Double	SRDA	RS
Compare (Short)	CE	RX	Shift Right Double Logical	SRDL	RS
Convert to Binary	CVB	RX	Shift Right Single	SRA	RS
Convert to Decimal	CVD	RX	Shift Right Single Logical	SRL	RS
Divide Register	DR	RR	Start I/O	SIO	SI
Divide	D	RX	Store	ST	RX
Divide Register (Long)	DDR	RR	Store Character	STC	RX
Divide (Long)	DD	RX	Store Halfword	STH	RX
Divide Packed	DP	SS	Store (Long)	STD	RX
Divide Register (Short)	DER	RR	Store Multiple	STM	RS
Divide (Short)	DER	RX	Store (Short)	STE	RX
Edit	ED	SS	Subtract Register	SR	RR
Edit and Mark	EDMK	SS	Subtract	S	RX
Exclusive OR Register	XR	RR	Subtract Halfword	SH	RX
Exclusive OR	X	RX	Subtract Logical Register	SLR	RR
Exclusive OR Immediate	XI	SI	Subtract Logical	SL	RX
Exclusive OR Character	XC	SS	Subtract Normalized Register (Long)	SDR	RR
Execute	EX	RX	Subtract Normalized (Long)	SD	RX
Halt I/O	HIO	SI	Subtract Normalized Register (Short)	SER	RR
Halve Register (Long)	HDR	RR	Subtract Normalized (Short)	SE	RX
Halve Register (Short)	HER	RR	Subtract Packed	SP	SS
Insert Character	IC	RX	Subtract Unnormalized Register (Long)	SWR	RR
Insert Storage Key	ISK	RR	Souotract Unnormalized (Long)	SW	RX
Load Register	LR	RR	Subtract Unnormalized Register (Short)	SUR	RR
Load	L	RX	Subtract Unnormalized (Short)	SU	RX
Load Address	LA	RX	Supervisor Call	SVC	RR
Load and Test	LTR	RR	Test and Set	TS	SI
Load and Test (Long)	LTDR	RR	Test Channel	TCH	SI

Table 2. (*Continued*)

Command	Mnemonic	Type	Command	Mnemonic	Type
Load and Test (Short)	LTER	RR	Test I/O	TIO	SI
Load Complement Register	LCR	RR	Test Under Mask	TM	SI
Load Complement (Long)	LCDR	RR	Translate	TR	SS
Load Complement (Short)	LCER	RR	Translate and Test	TRT	SS
Load Halfword	LH	RX	Unpack	UNPK	SS
Load Register (Long)	LDR	RR	Write Direct	WRD	SI
Load (Long)	LD	RX	Zero and Add Packed	ZAP	SS

The register specifiers may select MMX registers. The displacement is an address displacement. If the instruction requires immediate data, it is found in the final byte(s) of the instruction.

A summary of the Intel Architecture instruction set is given in Table 5. The arithmetic instructions are two-operand, where the operands can be two registers, register and memory, immediate and register, or immediate and memory. The Jump instructions have several forms depending on whether the target is in the same segment or a different segment.

MMX Instructions. The Pentium augments the ×86 instruction set with several multimedia instructions to operate on aggregate small integers. The MMX multimedia extensions have many SIMD-like characteristics. An MMX instruction operates on data types ranging from 8 to 64 bits. With 8-bit operands, each instruction is similar to a SIMD instruction in that during a single clock cycle, multiple instances of the instruction are being executed on different instances of data. The arithmetic instructions PADD/PSUB and PMULLW/PMULHW operate in parallel on either 8 bytes, four 16-bit words, or two 32-bit double words.

The MMX instruction set includes a DSP-like MAC instruction, PMADDWD, which does a multiply-add of four signed 16-bit words and adds adjacent pairs of 32-bit results. The PUNPCKL and PUNKCKH instructions help with interleaving words, which is useful for interpolation. The arithmetic instructions in the MMX instruction set allow for saturation to avoid overflow or underflow during calculations.

The MMX instructions use the Pentium floating point register set, thus requiring the FP registers to be saved and restored when multimedia instruction sequences occur in conjunction with floating point operations.

PowerPC RISC Processor

The PowerPC (PPC) family of 32- and 64-bit processors, jointly developed by IBM, Motorola, and Apple, follows the RISC architecture and instruction set philosophy. In common with other RISC processors, the PPC uses register operands for all arithmetic and logical instructions, along with a suite of load/store instructions to explicitly access data from memory. A complex instruction pipeline with multiple internal function units is used to achieve execution of more than one instruction per clock cycle.

The PPC CPU contains 32, 32-bit or 64-bit general-purpose registers; 32, 64-bit floating point registers; a 32-bit condition register; a 32- or 64-bit link register; and a 32- or 64-bit count register. The condition register can be set by arithmetic/logical instructions and is used by branch instructions. The link register is used to form an effective

Table 3. Addressing Modes of the PDP-11

address mode	Name	Form	Meaning
0	register	Rn	operand is in register n
1	indirect register[a]	(Rn)	address of operand is in register n
2	autoincrement	(Rn)+	address of operand is in register n n (Rn):=(Rn)+2 after operand is fetched[b]
3	indirect autoincrement	@(Rn)+	register n contains the address of the *address* of the operand: (Rn):=(Rn)+2 after operand is fetched
4	autodecrement	−(Rn)	(Rn):=(Rn)−2 *before* operand is fetched[c] address of operand is in register n
5	indirect autodecrement	@−(Rn)	(Rn):=(Rn)−2 before operand is fetched register n contains the address of the *address* of the Operand
6	index	X(Rn)	address of operand is in X+(Rn); address of X is in the PC; (PC):=(PC)+2 after X is fetched
7	indirect index	@X(Rn)	X+(Rn) is the *address* of the address of the operand; address if X is in the PC; (PC):=(PC)+2 after X is fetched

[a] "Indirect" is also called "deferred."

[b] If the instruction is a byte instruction and the register is *not* the SP or PC, (Rn):=(Rn)+l.

[c] If the instruction is a byte instruction and the register is *not* the SP or PC, (Rn):=(Rn)−l.

Table 4. PDP-11 instruction set contains a list of PDP-11 op codes

Command	Mnemonic	No. Operands	Command	Mnemonic	No. Operands
Add	ADD	2	Clear Z (= 0 condition)	CLZ	0
Add Carry	ADC	1	Clear N (> or < 0 condition)	CLN	0
Add Carry Byte	ADCB	1	Clear C, V, Z, and N	CCC	0
Arithmetic Shift Right	ASR	1	Compare	CMP	2
Arithmetic Shift Right Byte	ASRB	1	Compare Byte	CMPB	2
Arthmetic Shift Left	ASL	1	Complement	COM	1
Arthmetic Shift Left Byte	ASLB	1	Complement Byte	COMB	1
Bi Test	BIT	2	Decrement	DEC	1
Bi Test Byte	BITB	2	Decrement Byte	DECB	1
Bi Clear	BIC	2	Halt	HALT	0
Bi Clear Byte	BICB	2	Increment	INC	1
Bi Set	BIS	2	Increment Byte	INCB	1
Bi Set Byte	BISB	2	Jump	JMP	1
Branch Not Equal Zero	BNE	1	Move	MOV	2
Branch Equal Zero	BEQ	1	Move Byte	MOVB	2
Branch if Plus	BPL	1	Negate	NEG	1
Branch if Minus	BMI	1	Negate Byte	NEGB	1
Branch on Overflow Clear	BVC	1	Rotate Right	ROR	1
Branch on Overflow Set	BVS	1	Rotate Right Byte	RORB	1
Branch on Carry Clear	BCC	1	Rotate Left	ROL	1
Branch on Carry Set	BCS	1	Rotate Left Byte	ROLB	1
Branch if Gtr than or Eq 0	BGE	1	Set C (carry condition)	SEC	0
Branch if Less than 0	BLT	1	Set V (overflow condition)	SEV	0
Branch if Greater than 0	BGT	1	Set Z (= 0 condition)	SEZ	0
Branch if Less than or Eq 0	BLE	1	Set N (> or < 0 condition)	SEN	0
Branch Higher	BHI	1	Set C, V, Z, and N	sec	0
Branch Lower or Same	BLOS	1	Subtract	SUB	2
Branch Higher or Same	BHIS	1	Subtract Carry	SBC	1
Branch Lower	BLO	1	Subtract Carry Byte	SBCB	1
Clear	CLR	1	Swap Bytes	SWAB	1
Clear Byte	CLRB	1	Test	TST	1
Clear C (carry condition)	CLC	0	Test Byte	TSTB	1
Clear V (overflow condition)	CLV	0	Unconditional Branch	BR	1

address for memory access. The count register is used for fixed iteration loops and can be automatically decremented by checking its value within a branch instruction.

Each instruction fits into a 32-bit word. The 32-bit instruction contains a 6-bit opcode (for register-to-register mode), three 5-bit register operand specifiers, and 11 remaining opcode-specific modifier bits. In register-immediate mode, only one source operand is in a register, and the 5 bits for the second source register are concatenated with the 11-bit modifier field to yield a 16-bit constant. Other instruction formats use a more complex encoding, as shown below:

Reg - Reg	$\texttt{opcode}_6$	$\texttt{rd}_5$	$\texttt{rs1}_5$	$\texttt{rs2}_5$	$\texttt{opex1}_{11}$

Reg - Imm	$\texttt{opcode}_6$	$\texttt{rd}_5$	$\texttt{rs1}_5$	$\texttt{const}_{16}$

Branch	$\texttt{opcode}_6$	$\texttt{opex1}_6$	$\texttt{rs1}_5$	$\texttt{const}_{14}$	$\texttt{opex2}_2$

Jump	$\texttt{opcode}_6$	$\texttt{const}_{24}$	$\texttt{opex}_2$

Reg-reg is the register–register format used for the arithmetic and logical instructions. Reg-Imm is the register–immediate format in which the second operand is a 16-bit constant. The branch format specifies a relative branch distance in a 14-bit constant, and the Jump format uses a 24-bit constant to hold the jump or call target. “rd” is the register number of the destination, “rs1” is the first source operand, “rs2” is the second source operand register, “const” is a constant, and “opex1” and “opex2” are extensions of the opcode. The subscript shows the number of bits for each field.

The core PPC instruction set contains three categories of instructions: arithmetic/logical for both fixed and floating point, load/store for both fixed and floating point, and branch instructions. In addition, there are specialized instructions to control caches and synchronize memory accesses. Arithmetic and logical operations must use either both source operands in registers or one operand in a register and one operand as a 16-bit constant value. Load/store instructions access memory and can occur in one of three addressing modes:

- Register indirect with index, where the effective address from which to load or store is calculated by adding rsl to rs2.

Table 5. Intel architecture instruction set summary

Command	Opcode	Command	Opcode
ASCII Adjust after Addition	AAA	Load Global Descriptor Table Register	LGDT
ASCII Adjust AX before Division	AAD	Load Pointer to GS	LGS
ASCII Adjust AX after Multiply	AAM	Load Interrupt Descriptor Table Register	LIDT
ASCII Adjust AL after Subtraction	AAS	Load Local Descriptor Table Register	LLDT
ADD with Carry	ADC	Load Machine Status	LMSW
Add	ADD	Assert LOCK Num. Signal Prefix	LOCK
Logical AND	AND	Load String Operand	LOD*
Adjust RPL Field of Selector	ARPL	Loop Count (with condition)	LOOP*
Check Array Against Bounds	BOUND	Load Segment Limit	LSL
Bit Scan Forward	BSF	Load Task Register	LTR
Bit Scan Reverse	BSR	Move Data, Registers	MOV*
Byte Swap	BSWAO	Unsigned Multiply	MUL
Bit Test	BT	Two's Complement Negation	NEG
Bit Test and Complement	BTC	No Operation	NOP
Bit Test and Reset	BTR	One's Complement Negation	NOT
Bit Test and Set	BTS	Logical Inclusive OR	OR
Call Procedure (m same segment)	CALL	Output to Port	OUT*
Call Procedure (in different segment)	CALL	Pop Word/Register(s) from Stack	POP
Convert Byte to Word	CWB	Push Word/Register(s) onto Stack	PUSH
Convert Doubleword to Qword	CDQ	Rotate thru Carry Left	RCL
Clear Carry Flag	CLC	Rotate thru Carry Right	RCR
Clear Direction Flag	CLD	Read from Model Specific Register	RDMSR
Clear Interrupt Flag	CLI	Read Performance Monitormg Counters	RDPMC
Clear Task-Switched Flag in CRO	CLTS	Read Time-Stamp Counter	RDTSC
Complement Carry Flag	CMC	Input String	REP INS
Conditional Move	CMOVcc	Load String	REP LODS
Compare to Operands	CMP	Move String	REP MOVS
Compare String Operands	CMP[S[W/D]]	Output String	REP OUTS
Compare/Exchange	CMPXCHG	Store String	[REP] STOS
Compare/Exchange 8 Bytes	CMPXCHG8B	Compare String	REP[N][E] CMPS
CPU Identification	CPUID	Scan String	[REP] [N][E] SCANS
Convert Word to Doubleword	CWD	Return from Procedure	RET
Convert Word to Doubleword	CWDE	Rotate Left	ROL
Decimal Adjust AL after Addition	DAA	Rotate Right	ROR
Decimal Adjust AL after Subtraction	DAS	Resume from System Management Mode	RSM
Decrement by 1	DEC	Store AH into Flags	SAHF
Unsigned Divide	DIV	Shift Arithmetic Left	SAL
Make Stack Frame for Proc.	ENTER	Shift Arithmetic Right	SAR
Halt	HLT	Subtract with Borrow	SBB
Signed Divide	IDIV	Jziyte oet on Condition	SETcc
Signed Multiply	IMUL	Store Global Descriptor Tabel Register	SGTD
Input From Port	IN	Shift Left [Double]	SHL[D]
Increment by 1	INC	Shift Right [Double]	SHR[D]
Input from DX Port	INS	Store Interrupt Descriptor Table Register	SIDT
Interrupt Type n	INT n	Store Local Descriptor Table	SLDT
Single-Step Interrupt 3	INT	Store Machine Status Word	SMSW
Interrupt 4 on Overflow	INTO	Set Carry Flag	STC
Invalidate Cache	INVD	Set Direction Flag	SDC
Invalidate TLB Entry	INVLPG	Set Interrupt Flag	STI
Interrupt Return	IRET/IRETD	Store Task Register	STR
Jump if Condition is Met	Jcc	Integer Subtract	SUB
Jump on CX/ECX Zero	JCXZ/JECXZ	Logical Compare	TEST
Unconditional Jump (same segment)	JMP	Undefined Instruction	UD2
Load Flags into AH Register	LAHF	Verify a Segment for Reading	VERR
Load Access Rights Byte	LAR	Wait	WAIT
Load Pointer to DS	LDS	Writeback and Invalidate Data Cache	WVINVD
Load Effective Address	LEA	Write to Model-Specific Register	WRMSR
High Level Procedure Exit	LEAVE	Exchange and Add	XCHG
Load Pointer to ES	LES	Table Look-up Translation	XLAT[B]
Load Pointer to FS	LFS	Logical Exclusive OR	XOR

- Register indirect with immediate index, in which the effective address is calculated by adding rs1 to the constant.
- Register indirect, in which the effective address is in rs1.

Branch instructions also have three categories of addressing modes:

- Immediate. The 16-bit constant is used to compute a relative or absolute effective address.
- Link register indirect. The branch address is in the link register.
- Count register indirect. The branch address is in the count register.

Like the Pentium, some PowerPC models offer a SIMD-like extended instruction set for aggregate operation on byte (or larger) data. The extensions are variously referred to as Altivec (Motorola) or VMX (IBM). There are 162 specialized SIMD instructions that operate on a set of 32, 128-bit registers. Each register can be used as either 16, 8-bit registers; 8, 16-bit registers; or 4 single precision floating-point registers. Unlike the Pentium, the SIMD instruction set operates on a completely different set of registers than the normal instruction set, and thus, the general-purpose registers do not need to be saved or restored when SIMD instructions are executed.

IA-64 Itanium Processor

As discussed, modern microprocessors achieve performance by executing multiple instructions in parallel. In most cases, the parallelism is hidden from the instruction set architecture view of the microprocessor. In contrast, the Intel and HP Itanium processor is a 64-bit architecture (Intel Architecture IA-64) that follows an explicitly parallel instruction computing (EPIC) model. The EPIC model exposes opportunities for ILP in the instruction set, allowing the compiler and the underlying microarchitecture to communicate about potentially parallel operations. This architecture incorporates ideas from CISC, RISC, and VLIW.

The IA-64 architecture provides a very large set of 64-bit registers, including 128 general registers: 128, 82-bit floating-point registers; and 128 application registers. In addition, there are 64, 1-bit predicate registers (called condition registers in other architectures) and 8, 64-bit branch registers. The 1-bit registers NaT (not-a-thing) and NatVal (not-a-thing-value) are used to signal potential exception conditions. There is a NaT register for each general register and a NatVal for each floating-point register. Other miscellaneous registers are used for memory mapping, system control, performance counters, and communicating with the operating system.

The 128-bit instruction word is called a "bundle" and contains three 41-bit instructions plus a 5-bit "template" that is used to help decode and route instructions within the instruction pipeline. The template bits also can signal the end of a group of instructions that can be executed in parallel. This instruction format is an outgrowth of the VLIW architectures described above.

The 128 general registers are divided into two groups. A set of 32 static registers is used similarly to RISC processors. The remaining 96 registers are called "stacked" registers and implement a register stack that is used to store parameters and results of procedure calls. Registers from the stacked set are allocated with an explicit "alloc" instruction.

IA-64 provides instructions to rename registers, which makes the registers appear to rotate. This mechanism is provided for the general registers, floating-point registers, and the predicate registers. The RRB (register rotation base) is used to specify a register number offset. Rotate instructions are used by the compiler to support "software pipelining," a technique whereby multiple loop iterations execute concurrently. By rotating a set of registers, a set of active loop iterations all refer to different registers and can execute in parallel. The general registers 32–127, floating point registers 32–127, and predicate registers 16–63 can be rotated. The RRB register is used to specify an offset to the subset of rotating registers. A reference to any register in the range of the rotating registers is offset by the value of the RRB. Thus, if the RRB has a current value of 15, a reference to GR[40] would actually refer to GR[55]. The effective register number is computed using modulo arithmetic, so that the register values appear to rotate.

The compiler creates "instruction groups" of instructions that can be executed in parallel. The size of the group is variable, with a stop bit in the template indicating the end of a group. Often, the amount of parallelism is limited by conditional execution ("if" statements or "if" expressions in most programming languages). The IA-64 architecture supports parallelism through conditional execution by using predicate bits in the instruction word: The instruction is executed only if the specified predicate bit is true. This feature is remniscent of SIMD-style processors, with a "context" bit determining whether a processor executes the instruction. Conditional branching is also provided in the instruction set by allowing each instruction to branch based on a different predicate register.

The IA-64 has unique instructions that allow operations such as loads and stores to memory to be executed *speculatively*. A speculative operation is executed before it would normally be executed in the sequential instruction stream. For example, consider the instruction sequence

1. Branch conditional to 3.
2. Load from memory.
3. Other instruction.

Speculative execution of the load instruction means that the load (instruction 2) is executed before the branch (instruction 1) completes. A set of speculation check instructions then determine whether the speculative load (or store) is kept or discarded.

Similarly, suppose the instruction sequence includes a store followed later in the instruction stream by a load. The load may be executed speculatively before the store even if the compiler cannot guarantee that the load and store refer to different addresses. A check instruction follows the store to determine whether the store and load refer to the same or different addresses. If they refer to the same address (called “aliasing”), the speculatively loaded value is discarded, and the most recently stored value is used. If they refer to distinct locations, the loaded value is immediately available in the register for use by other instructions.

Speculative operations cause the CPU to perform additional work. However, if they enable the CPU to not wait when values are needed, they improve execution rates. In some cases, however, speculative operations may cause exception conditions that, under normal sequential operation, would not have occurred. For example, if a load were performed speculatively before a branch, and the address to be loaded were illegal, then a true exception should not be raised because the load may never be executed. The NaT and NatVal registers record exceptions that occur during speculative execution. If the speculative operation is retained, an exception is raised; otherwise, the speculative operation is aborted.

Another unique aspect of the IA-64 architecture is the ability to emulate other instruction sets. There are special instructions in the instruction set to direct the IA-64 to operate in IA-32 mode, and an IA-32 instruction to return to IA-64 mode. The application register set is used to facilitate emulation of other instruction set architectures.

Although it is not feasible to include the entire IA-64 instruction set in a summary article, the core set of IA-64 instructions[1] are as follows:

- Load/store, memory operations.
- Logical, compare, shift, arithmetic operations.
- Aggregate operations on small integers, similar to the MMX (see above or Altivec).
- Floating-point operations, both simple and aggregate.
- Branch operations, including multiway branches and loop control branches.
- Cache management operations.

Cray X1 Computer

The Cray X1 was announced in November 2002, although five early production systems had already been shipped. The X1 combines vector processing (from the Cray C90, T90, and SV1) and massively parallel processing (MPP, from the Cray T3D and T3E) into a single unified architecture.

A single stream processor (SSP) is a RISC processor, consisting of a superscalar processing unit and a two-pipe vector processing unit, which is the basic component of the system. Four SSPs are combined to form a multistream processor (MSP). Four MSPs form a *node*. Cache memory is fully shared by the four SSPs in an MSP; memory is fully shared by the four MSPs of a node.

A maximum of 1024 nodes can be joined in a X1 system. A hypercube network combines nodes into groups of 128. A three-dimensional-torus network connects the hypercubes to form a *global shared nonuniform memory access* (NUMA) machine.

The X1 has two execution modes. In SSP mode, each SSP runs independently of the others, executing its instruction stream. In MSP mode, the MSP automatically distributes parallel parts of multistreaming applications to its SSPs. SSPs support vectorization; MMPs support multistreaming. The entire system support both the distributed (MPI, shmem) and the shared memory (UPC, coarray FORTRAN) parallel paradigms.

Table 6 shows the register types for the Cray X1 processors.

Although both the address and the scalar registers are general purpose and can be used for memory reference instructions, immediate loads, integer functions, and conditional branches, they each have specific uses as well. The address registers *must* be used for memory base addresses,

Table 6. Cray X1 Register Types

register type	designator	number	size in bits	comment
address	a	64	64-bits	general purpose
scalar	s	64	64-bits	general purpose
vector	V	31	32- or 64-bits	max. 64 elements in each
vector length	vl	1		max. elements a vector register can hold
mask	m	8	varies	control vector ops on per-element basis; only first four used in instuctions
vector carry	vc	1	varies	used w/64-bit vector add w/carry and subtract w/borrow inst.
bit matrix mult.	bmm	1	64 x 64-bit	loaded from a vector register
control	c	64		mostly kernel mode; only cO–c4, c28–c31 are user accessible
program counter	pc	1	64-bit	byte addr. of next instruction to fetch; invisible to user but content referenced in some instruction descriptions
performance ctrs	32	64-bits		accessible via c31

[1] Most of these instructions can be predicated.

field name	g	i	j	k	t	f
bit length	6	6	6	6	2	6
use	opcode	dest.	source	source	size+	opcode

Figure 4. Cray XI instruction format.

indirect jump addresses and returns, vector element indexing, vector length computations for the vector length register, reading and writing the vector length register and control registers, receiving results of mask analysis instructions [first(), last(), pop()], supplying the span for vector span() and cidx(), and 8- and 16-bit accesses. The scalar registers *must* be used for scalar bit matrix multiplications, floating-point operations, and scalar operands to vector operations.

The Cray X1 has fixed 32-bit instructions. All instructions (other than branch instructions) have six fields, although all fields may not be used by all instructions. The instruction format is shown in Fig. 4.

The g-field opcode is more of a general class such as "a-register integer instructions," "a-register halfword instructions," and "s-register logical instructions". The f-field opcode, when used, specifies the specific instruction in the general class such as "a-register integer add" and "a-register integer subtract". The source and destination fields, i, j, and k, can be any of the address, scalar, or vector registers or, when appropriate, a mask register. Additionally, the source may also be an "immediate" value. Immediates can be 6 bits, 8 bits (using the t-field), or 16 bits (using the t- and f-fields plus 2 bits from the j-field). The t-field is used for various flags; for example, "11" is used in register logical oprations to indicate that the second source operand is a register (rather than "immediate") and "01" and "00" are used to flag "64-bit" and "32-bit", respectively, in register move and conversion operations.

Branch instructions use only three fields: g, i, and k. The g-field contains the opcode, the i-field contains the location of the value to be tested for the condition, and the k-field is an immediate of 20 bits, which when combined with the program counter, yields the branch address.

The Cray X1 has a rich ISA for scalar and vector instructions. An overview of the vector instructions is given here.

The vector instruction set is too rich to be included here..

The vector instruction set is organized into five categories:

- Elemental vector operations
- Vector memory references
- Elemental vector functions
- Mask operations
- Other vector instructions

The *elemental vector operations* are vector versions of most scalar integer and floating-point functions and memory references. These operations process each vector element independently, under control of a mask register and the vector length register. The semantics of these operations is similar to a loop stepping through each element of a vector.

Vector registers are loaded and stored from a sequence of properly aligned byte addresses. The address sequence is computed from a base address register and either a scalar stride value or a vector of 64-bit offset values. The five *vector memory reference* instructions are **strided load, strided store, gather,** and two **scatters,** one with distinct offsets and one with arbitrary offsets.

The *elemental vector functions* include **arithmetic** operations, **bitwise** operations (and, or, etc.), **logical left shift, logical right shift,** and **arithmetic right shift,** several floating point to integer **convert** instructions, **compare** instructions ([not] equal, [not] less than, [not] greater than), **merge, square root, leading zero count, population count, bit matrix multiply,** and **arithment absolute value.** Most of these operations permit a scalar register, in place of a vector register, for one data source.

Mask operations operate directly on mask registers to set values and otherwise manipulate these registers. Instructions include **bitwise** operations (and, xor, etc.), **set leading n bits, clear remainder, find lowest/highest set bit index,** and **count number of bits set,** among others. Any mask register can be used in the mask operation instructions, but only the first four, m0–m3, can be used in vector instructions.

The *other vector operations* category contains those instructions that do not fit easily into the other four categories. These are **set vector length, retrieve vector length, read vector element, write vector element, load bit matrix,** and **declare vector state dead.** This last instruction undefines all vector registers, the vector carry register vc, and the mask registers. Mask register m0 remains defined if it has all of its bits set; otherwise, it too becomes undefined.

FURTHER READING

N. Chapin, *360 Programming in Assembly Language*, New York: McGraw-Hill, 1968.

Cray XI System Overview, S-2346-22.

Cray Assembly Language (CAL) for the Cray XI Systems Reference Manual, S-2314-50.

J.R. Ellis, *Bulldog: A Compiler for VLIW Architectures*, Cambridge, MA: The MIT Press, 1986.

A. Gill, *Machine and Assembly Language Programming of the PDP-11*, Englewood Cliffs, NJ: Prentice-Hall, 1978.

J. Huck, Introducing the IA-64 Architecture, *IEEE Micro*, **20**(5): 12–23, 2000.

K. Hwang, *Advanced Computer Architecture*, New York: McGraw Hill, 1993.

David Patterson and John Hennessy, *Computer Organization and Design (2nd ed.). The Hardware/Software Interface*. San Mateo, CA: Morgan Kaufmann, 1997.

MAYA B. GOKHALE
Lawrence Livermore National
Laboratory
Livermore, California
JUDITH D. SCHLESINGER
IDA Center for Computing
Science
Bowie, Maryland

INTERCONNECTION NETWORKS FOR PARALLEL COMPUTERS

The interconnection network is responsible for fast and reliable communication among the processing nodes in any parallel computer. The demands on the network depend on the parallel computer architecture in which the network is used. Two main parallel computer architectures exist (1). In the *physically shared-memory parallel computer, N* processors access *M* memory modules over an interconnection network as depicted in Fig. 1(a). In the *physically distributed-memory parallel computer,* a processor and a memory module form a processor–memory pair that is called *processing element* (PE). All *N* PEs are interconnected via an interconnection network as depicted in Fig. 1(b). In a *message-passing system,* PEs communicate by sending and receiving single messages (2), while in a *distributed-shared-memory system,* the distributed PE memory modules act as a single shared address space in which a processor can access any memory cell (3). This cell will either be in the memory module local to the processor, or be in a different PE that has to be accessed over the interconnection network.

Parallel computers can be further divided into SIMD and MIMD machines. In *single-instruction-stream multiple-data-stream* (SIMD) parallel computers (4), each processor executes the same instruction stream, which is distributed to all processors from a single control unit. All processors operate synchronously and will also generate messages to be transferred over the network synchronously. Thus, the network in SIMD machines has to support synchronous data transfers. In a *multiple-instruction-stream multiple-data-stream* (MIMD) parallel computer (5), all processors operate asynchronously on their own instruction streams. The network in MIMD machines therefore has to support asynchronous data transfers.

The interconnection network is an essential part of any parallel computer. Only if fast and reliable communication over the network is guaranteed will the parallel system exhibit high performance. Many different interconnection networks for parallel computers have been proposed (6).

One characteristic of a network is its topology. In this article we consider only point-to-point (non-bus-based) networks in which each network link is connected to only two devices. These networks can be divided into two classes: direct and indirect networks. In direct networks, each switch has a direct link to a processing node or is simply incorporated directly into the processing node. In indirect networks, this one-to-one correspondence between switches and nodes need not exist, and many switches in the network may be attached only to other switches. Direct and indirect network topologies are discussed in the following section.

The mechanism to transfer a message through a network is called *switching.* A section below is devoted to switching techniques. Switching does not take into consideration the actual route that a message will take through a network. This mechanism is termed *routing,* and will be discussed in turn. In indirect networks, active switch boxes are used to transfer messages. Switch box architectures are discussed in a final section.

NETWORK TOPOLOGIES

Direct Networks

Direct networks consist of physical interconnection links that connect the nodes (typically PEs) in a parallel computer. Each node is connected to one or more of those interconnection links. Because the network consists of links only, routing decisions have to be made in the nodes. In many systems, dedicated router (switch) hardware is used in each node to select one of the interconnection links to send a message to its destination. Because a node is normally not directly connected to all other nodes in the parallel computer, a message transfer from a source to a destination node may require several steps through intermediate nodes to reach its destination node. These steps are called *hops.*

Two topology parameters that characterize direct networks are the degree and the network diameter. The *degree* Γ of a node is defined as the number of interconnection links to which a node is connected. Herein, we generally assume that direct network links are bidirectional, although this need not always be the case. Networks in which all nodes have the same degree *n* are called *n-regular* networks. The network *diameter* Φ is the maximum distance between two nodes in a network. This is equal to the maximum number of hops that a message needs to be transferred from any source to any destination node. The degree relates the network topology to its hardware requirements (number of links per node), while the diameter is related to the transfer delay of a message (number of hops through the network). The two parameters depend on each other. In most direct network, a higher degree implies a smaller diameter because with increasing degree, a node is connected to more other nodes, so that the maximum distance between two nodes will decrease.

Many different direct network topologies have been proposed. In the following, only the basic topologies are studied. Further discussion of other topologies can be found in Refs. 7–9.

In a *ring network* connecting *N* nodes, each node is connected to only two neighbors ($\Gamma = 2$), with PE *i* connected to PEs $i - 1 \bmod N$ and $i + 1 \bmod N$. However, the network has a large diameter of $\Phi = \lfloor N/2 \rfloor$ (assuming bidirectional links). Thus, global communication performance in a ring network will decrease with increasing number of nodes.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

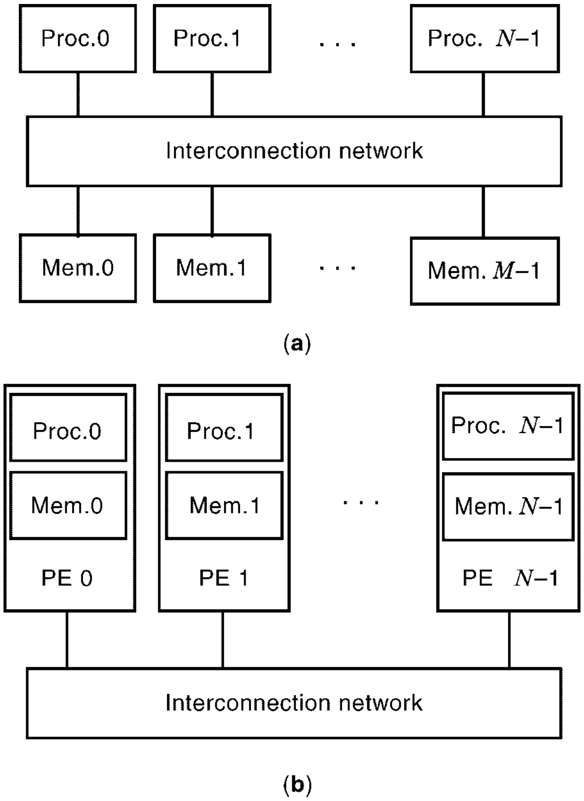


Figure 1. (a) Physically shared-memory and (b) distributed-memory parallel computer architecture.

A direct network quite commonly used in parallel computers is the mesh network. In a *two-dimensional mesh,* the nodes are configured in an $M_X \times M_Y$ grid (with M_X nodes in the X direction and M_Y nodes in the Y direction), and an internal node is connected to its nearest neighbors in the north, south, east, and west directions. Each border node is connected to its nearest neighbors only. A 4 × 4 two-dimensional mesh connecting 16 nodes is depicted in Fig. 2(a). Because of the mesh edges, nodes have different degrees. In Fig. 2(a), the internal nodes have degree $\Gamma = 4$, while edge nodes have degree $\Gamma = 3$ and $\Gamma = 2$ (for the corner nodes). Because the edge nodes have a lower degree than internal nodes, the (relatively large) diameter of a two-dimensional mesh is $\Phi = (M_X - 1) + (M_Y - 1)$.

To decrease the network diameter, the degree of the edge nodes can be increased to $\Gamma = 4$ by adding edge links. The topology of a two-dimensional *torus network* is created by connecting the edge nodes in columns and rows, as depicted in Fig. 2(b). All nodes of this two-dimensional torus network have degree $\Gamma = 4$, and the network diameter is reduced to $\Phi = \lfloor M_X/2 \rfloor + \lfloor M_Y/2 \rfloor$.

The disadvantage of two-dimensional mesh networks is their large diameter, which results in message transfers over many hops during global communication, especially in larger networks. To further reduce the diameter, higher-dimensional meshes can be used. Figure 3(a) depicts a three-dimensional mesh with open edge connections connecting 27 nodes. Internal nodes have degree $\Gamma = 6$, while edge nodes have degree of $\Gamma = 5$, $\Gamma = 4$, or $\Gamma = 3$, depending on their position. The network diameter is equal to $\Phi = (M_X - 1) + (M_Y - 1) + (M_Z - 1)$, with M_i equal to the number of nodes in the i direction. This diameter can be further reduced if edge connections are added.

In a *hypercube network* that connects N nodes, each node has degree $\Gamma = n = \log_2 N$, where n is called the *hypercube dimension* (8). Each link corresponds to a cube function (10). The *cube$_k$ function* on an address (node number) complements the kth bit of that address. To describe the hypercube topology, the Hamming distance H can be used. The *Hamming distance H* between two binary numbers is defined in Ref. 11 as the number of bits in which the two numbers differ. Thus, two nodes are directly connected in a hypercube if their Hamming distance is $H = 1$ (the node numbers differ in exactly one bit). The number of hops that a message will take through the network is therefore equal to the Hamming distance between its source and destination addresses. In Fig. 3(b), a four-dimensional hypercube that connects 16 nodes is depicted. The diameter of a hypercube network is $\Phi = n$, because in the worst case, a source and a destination address of a message can differ in all n bits, so that all n cube functions have to be executed in order to transfer that message.

One disadvantage of a hypercube network concerns scalability. To increase the number of nodes a hypercube can interconnect, the degree of each node has to be incremented by at least one. Thus, to obtain the next larger hypercube, the number of nodes has to be doubled. To alleviate this scalability problem, *incomplete hypercubes*

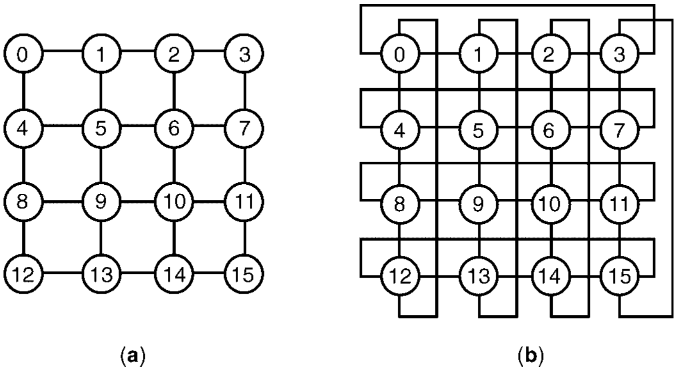


Figure 2. (a) Two-dimensional mesh network connecting 16 nodes, (b) torus network connecting 16 nodes. Because of the edge connections, the torus network has a uniform degree of four, while nodes in the mesh network have different degrees, depending on their location.

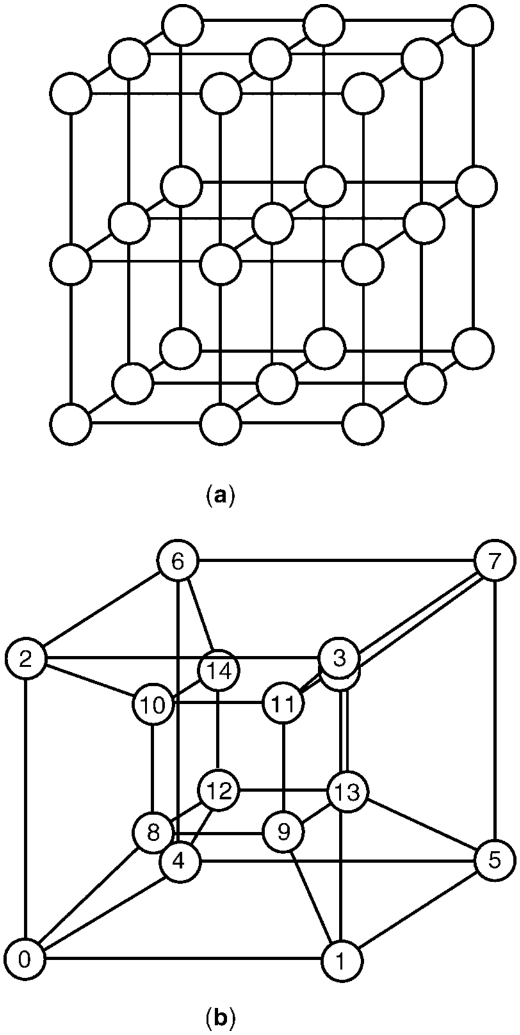


Figure 3. (a) Three-dimensional mesh connecting 27 nodes, (b) four-dimensional hypercube network connecting 16 nodes. In hypercube networks, the nodes that are directly connected have a Hamming distance of $H = 1$.

were introduced, in which any number of nodes can be interconnected (12).

To relate the different direct network topologies, the k-ary n-cube classification was introduced (13). A *k-aryn-cube network* connects $N = k^n$ nodes, where n is equal to the number of different dimensions the network consists of, while k is the network radix, which is equal to the number of nodes in each dimension. For example, a k-ary 1-cube is equivalent to a k-node ring network, a k-ary 2-cube is equivalent to a k^2-node torus network, and a 2-ary n-cube is equivalent to a 2^n-node n-dimensional hypercube. Figure 3(a) depicts a 3-ary 3-cube (assuming appropriate edge connections not shown in the figure), and Fig. 3(b) a 2-ary 4-cube. The diameter (Φ) is $n \times \lfloor k/2 \rfloor$.

Indirect Networks

In indirect networks, each processing node is connected to a network of switches over one or more (often bidirectional) links. Typically, this network consists of one or more stages of switch boxes; a network stage is connected to its successor and predecessor stage via a set of interconnection links. Depending on the number of stages, the number of switch boxes per stage, and the interstage interconnection topology, indirect networks provide exactly one path (*single-path networks*) or multiple paths (*multipath networks*) from each source to each destination.

Many different indirect network topologies have been proposed. This section is a brief introduction to multistage cube and fat-tree networks. Further discussion of these and other topologies can be found in Refs. 14–17.

One important indirect single-path network topology is the *generalized-cube network* topology (10), based on the cube interconnection function. A generalized-cube network that connects $N = 2^n$ sources with N destinations consists of $s = \log_B N$ stages of $B \times B$ switch boxes. The stages are numbered from $s - 1$ (stage next to the sources) to 0 (stage next to the destination). Each stage consists of N/B switch boxes; two consecutive stages are connected via N interconnection links. In Fig. 4(a), an 8 × 8 generalized-cube network comprising 2 × 2 switch boxes is shown, while Fig. 4(b) depicts a 16 × 16 generalized-cube network with 4 × 4 switches.

Consider the link labeling depicted in Fig. 4(a). The labels at the input (and output) side of each switch box differ in exactly one bit, which is bit k in stage k. Thus, if a message is routed straight through a switch box, its link number is not changed. If a message goes from the upper input to the lower output (or from the lower input to the upper output) at stage k, it moves to an output link that differs in bit k (the cube$_k$ operation transforms the link number). Each stage corresponds to a specific cube function, and all n cube-functions can be applied to a message on its way through the network.

A simple distributed routing algorithm can be used to transfer messages through the network. As routing information, each message header contains its destination address (*destination-tag routing*). If a message enters a switch box in stage k, this switch box will examine the kth bit of the message destination address. This bit determines the switch box output port to which the message is destined. If the bit is 0, the message is destined to the upper output port; if it is 1 to the lower output port. This scheme can be easily extended to $B \times B$ switch boxes, using the kth digit of the radix B representation of the destination address to select one of the B switch output links.

In shared memory parallel computers, many messages are requests for memory data, which results in reply messages that send data back to the original source. Thus, a read request sent through the network to the memory has to include the destination address (memory address) and also the source address (the node number to with the data is to be sent back). Thus, when destination-tag routing is used, the source address has to be added to the message header. This overhead can be avoided by using the XOR-routing algorithm. During *XOR routing,* an n-bit routing tag T that is formed by XOR-ing the source and the destination address ($T = S \oplus D$) is added to each message as a message header. If a message enters a switch box in stage k, this switch box will examine the kth bit of the message

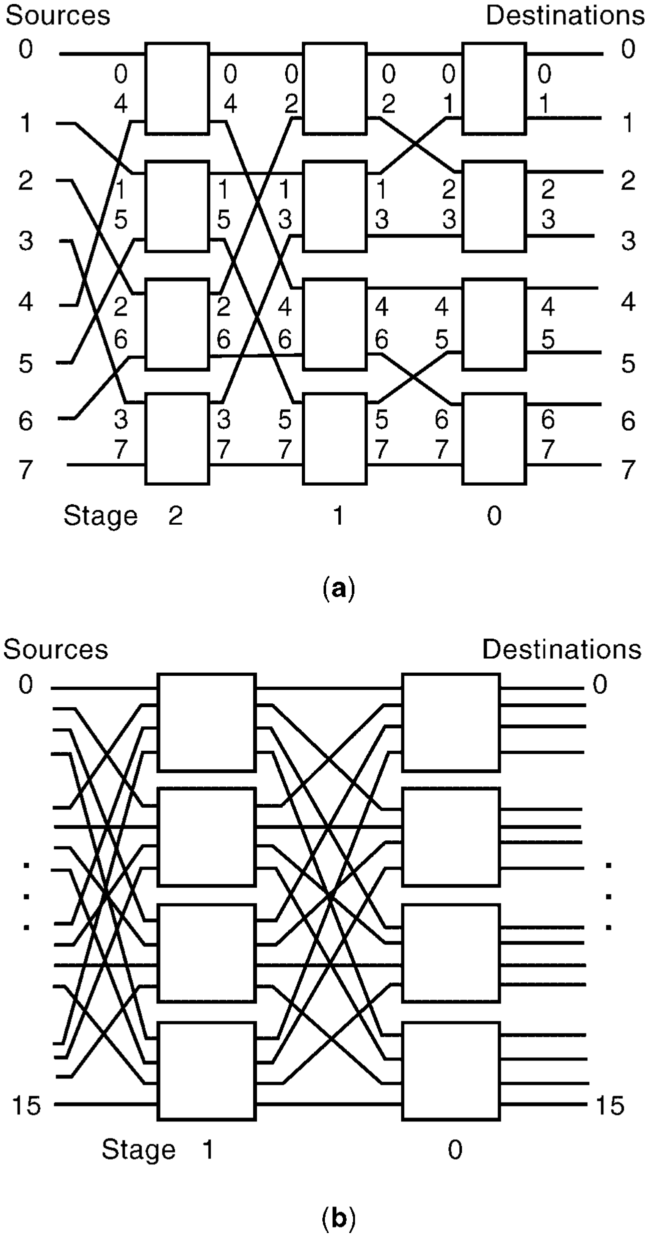


Figure 4. (a) 8 × 8 generalized-cube network comprising 2 × 2 switch boxes, (b) 16 × 16 generalized-cube network comprising 4 × 4 switch boxes. The link labels at the input (and output) side of each switch box in (a) differ in exactly one bit (bit k in stage k).

routing tag T. If this bit is 0 (the corresponding source address bit is equal to the destination address bit), the message will be routed straight through that switch box (e.g., if it arrived at the upper input, it will be routed to the upper output). If the routing bit is 1, the switch will be set to exchange (e.g., if the message arrived at the upper input, it will be routed to the lower output). Once a message has arrived at its destination, the destination can determine the message's source address by XORing its own address with the message's routing tag T. XOR routing works in networks comprising 2 × 2 switch boxes only. A similar scheme can be used in hypercube networks.

Many different single-path multistage networks have been proposed in the literature, among them the SW-banyan, omega, indirect binary n-cube, delta, baseline, butterfly, and multistage shuffle-exchange networks. In Ref. 10 it was shown (by reordering switches and/or renumbering links) that instances of these networks are typically equivalent to the generalized-cube network topology.

A generalized topology of a multipath indirect network is the three-stage network. This network consists of three stages of switches. Each switch box in the first and third network stages is connected to all switches in the network middle stage. A 16 × 16 multipath network comprising 4 × 4 switches is depicted in Fig. 5. The number of switches in the middle stage determines the number of distinct paths from each source to each destination (in Fig. 5, there are four distinct paths between any source and destination).

Another multipath indirect network topology that is used in parallel computers is the fat-tree network (18). The *binary fat-tree* network has the topology of a binary tree in which the leaves are connected to the processing elements and the root and intermediate tree nodes are switch boxes. All interconnection links are bidirectional. Unlike in an ordinary tree, the number of links between internal tree nodes is increasing when ascending the tree from the leaves to its root. Figure 6(a) depicts a binary fat-tree network that interconnects eight nodes. A cluster of processors is connected to the same switch box in the lowest switch level of the network (the switch level closest to the processors). This network provides only a single path that connects processors within a cluster. For all other connections, there exist multiple paths. To route a message between two nodes, the message first ascends in the network, rising to the lowest common ancestor of the source and destination, and then descends to the destination. This indirect topology thus rewards local communication by providing shorter paths between nearer nodes.

A different network topology that is similar to a fat tree is shown in Fig. 6(b). As in the binary fat-tree network, only a single path connects two processors within a cluster. However, each switch box on the lower switch level is connected to all switches on the next higher level. Thus, the number of switches in the higher switch level determines the number of different paths between two processors in different processor clusters. More switch levels can be added to the network, which will increase the number of distinct paths among processors in different clusters. However, with each switch level, the message transfer delay will increase, because more switches have to be traversed by a message if the message is routed through higher switch levels.

SWITCHING TECHNIQUES

The mechanism to transfer a message through a network is called *switching*. Switching does not take into consideration the actual route that a message will take through a network (this mechanism is termed routing and will be discussed in the next section). The four fundamental and

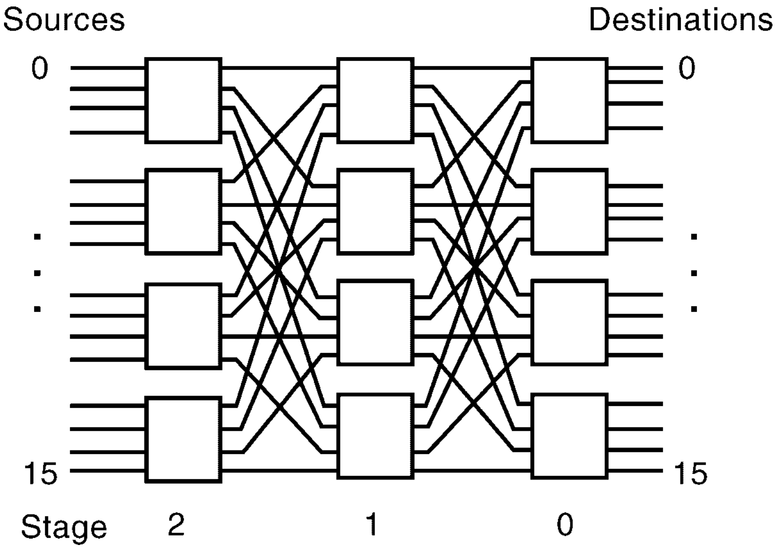


Figure 5. 16 × 16 three-stage multipath indirect network comprising 4 × 4 switch boxes. This network provides four link-disjoint paths from any source to any destination.

most-used switching techniques in interconnection networks are circuit switching, packet switching, wormhole routing, and virtual cut-through.

In a *circuit-switched* network, a complete connection through the network (from the source to the destination) is established before any data are sent. Network resources such as network links and switch ports are exclusively reserved for this connection. Once the connection is established, data are sent over the reserved network links and ports. After all data are sent, the established connection is disconnected to free the reserved resources for new connections. The connection establishment and disconnection can either be controlled centrally through a central network controller, or decentralized through messages that are sent through the network during connection establishment and disconnection. If a connection cannot be established because needed network resources are unavailable, the connection is refused (data cannot be transmitted) and the source has to try to establish the connection again.

In a *packet-switched* network, a message is divided into one or more data packets and routing information is added to each packet. These packets are sent through the network without the establishment of an overall connection between the source and destination. Network resources are reserved only when needed by a packet. Thus, network resources forming the path of a given packet that are not occupied by the given packet can be used to transfer other packets while the given packet is still in the network. This is impossible under circuit switching. The packet-switching technique is also called *store-and-forward packet-switching,* because a packet will be forwarded to the next node only if it was completely received by the current node. Therefore, nodes need enough space to buffer at least one complete packet. If a network resource such as a node's output port that a packet needs to use is unavailable (used by another message), the packet waits in its buffer within the node until the resource becomes available.

Wormhole routing is a switching technique similar to packet switching and is currently most often used in direct networks. In a wormhole-routed network, a message is divided into several *flow-control digits* (*flits*) (19). The first flit of a message (*header flit*) contains the message's routing information, and the last flit (*tail flit*) indicates its end. A message will be sent, flit by flit, in a pipelined fashion

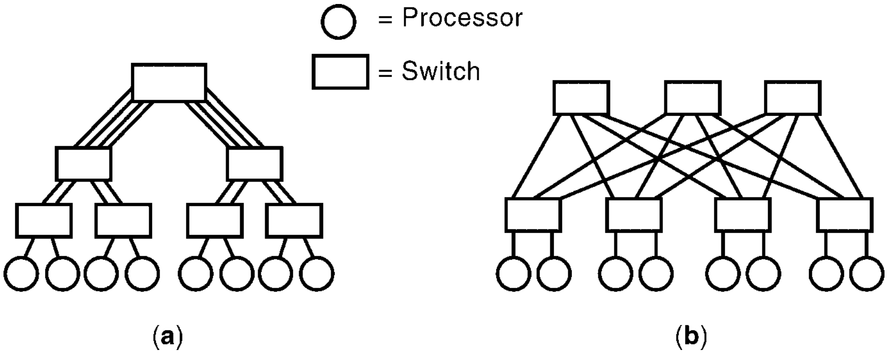


Figure 6. (a) Binary fat-tree network and (b) generalized fat-tree network connecting eight processors. This topology results in fast local communication, while the performance of global communication depends on the network size.

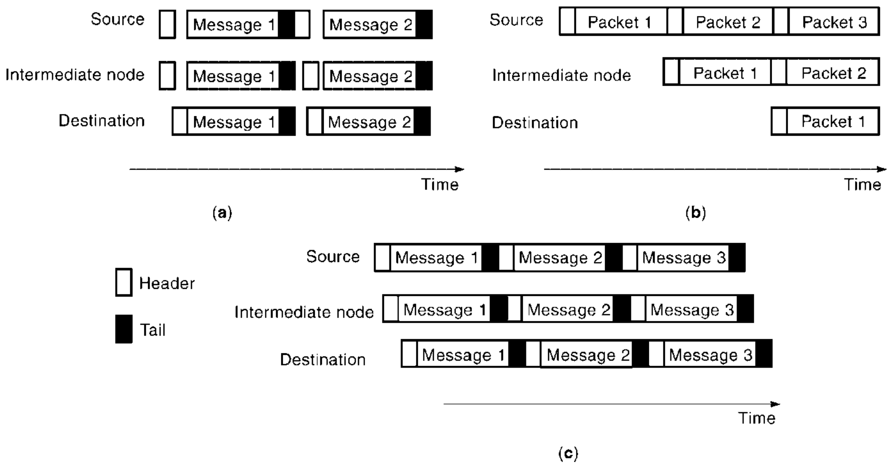


Figure 7. Data transport through an intermediate node in (a) a circuit-switching network, (b) a store-and-forward packet-switching network, and (c) a wormhole- routing network. Circuit switching and wormhole routing result in a shorter message transmission time, while packet-switching networks tend to have fewer message blockings.

through the network. The header flit will reserve network resources exclusively for its message, and the tail flit will release each resource after it has passed it. Thus, the message will traverse a network like a worm through a hole. Depending on the message length (number of flits) and the length of the path the message takes through the network (number of intermediate nodes), the tail flit will be submitted to the network either while the head is still in the network, or when part of the message is already received by the destination.

If a header flit cannot acquire a network resource (e.g., an output port of an intermediate node), it has to be temporarily buffered in that node (normally at the input port of that node). This will stop the worm from advancing through the network. To minimize the network hardware, normally each input port of a node has the capability of buffering one or two flits only. Therefore, once a worm has stopped advancing through the network, each flit of the worm will wait in the node it currently resides in, without releasing any network resources. Thus, while a worm is blocked in a network, it will block the corresponding network resources from being used by other messages. This can result in deadlocks within the network, and the routing algorithm used in the network has to handle those situations (see the next section).

The *virtual-cut-through* (VCT) switching technique combines characteristics of store-and-forward packet switching and wormhole routing. Each data packet is divided into flits again and sent through the network, as is done during wormhole routing. However, each node has the capability to buffer a whole packet. If a flit reaches an empty node buffer and is not blocked, it will either be directly routed through the node or be buffered in that buffer for one flit cycle and then routed through the node (depending on the implementation). If a message is blocked and cannot be forwarded to the next node, all the flits of that message will be received one by one and buffered in that blocked node. Thus, under a light network load, VCT behaves similarly to wormhole routing. Under heavier loads, when blocking occurs more frequently, the message worm will be completely buffered in the blocked node, similarly to store-and-forward packet switching. This way, the message does not block resources of several nodes and will therefore block fewer messages in the network.

In Fig. 7, the data transport from a source to a destination through an intermediate node over time is shown for a circuit-switching, a store-and-forward packet-switching, and a wormhole-routing network (in the circuit-switching example, line propagation delays are neglected). It can be seen that circuit-switching and wormhole-routing networks behave similarly over time, while the packet transmission in a store-and-forward packet-switching network takes longer. As long as the header and tail parts of a message are much shorter than the message itself, the transmission time for a message in a wormhole-routing and circuit-switching network is virtually independent of the length of the path the message has to take through the network. Pipelining of the message bits or flits on the network interconnection links can further reduce the transmission time. On the contrary, in a store-and-forward packet-switching network, the transmission time of a message is proportional to the length of the path through the network. This has to be weighted against the fact that blocked messages will normally block fewer other messages in a store-and-forward packet-switching network than in a wormhole-routing network, while in a circuit-switching network, connections might be refused due to internal blocking. As noted earlier, the behavior of virtual cut-through depends on the network load.

The main disadvantage of wormhole-routing networks is that a blocked message may spread over several nodes in the network and will then block several network links, which become unavailable for other messages. As an example, consider Fig. 8(a). Two interconnected wormhole switches are shown that have a flit buffer at each input port. Assume that a message is currently routed through switch 2 from port D to port E. This message blocks another message that enters switch 1 at port A, which is destined to port E as well. The head flit will wait in the flit buffer at input port C. However, this message blocks a third message

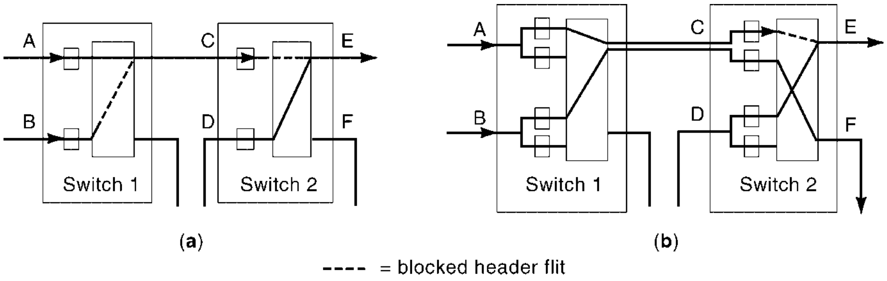


Figure 8. (a) Conventional wormhole-routing network, (b) wormhole-routing network with virtual channels. The virtual channels enhance the network performance substantially because fewer messages are blocked.

entering switch 1 at port B that is destined to port F. In this example, two messages are blocked because port E is currently unavailable.

To alleviate this problem, *virtual channels* were introduced (20). As depicted in Fig. 8(b) each switch now has two parallel flit buffers per input port, resulting in two virtual channels that are multiplexed over one physical interconnection link. In this case, the message entering switch 1 at input port A is still blocked at input port C because it is destined to the busy output port E. However, the third message is able to use the second virtual channel at input port C, so that it can proceed to the idle output port F.

The concept of virtual channels enhances the performance of wormhole-routing networks substantially, especially when the data traffic consists of a mixture of short and long messages. Without virtual channels, long messages can block short messages for quite some time. However, short messages often result from time-critical operations such as synchronization, so that a short latency is crucial for those messages. Because message latency also includes blocking time, virtual channels result in a decreased latency because there is less message blocking in the network.

ROUTING TECHNIQUES FOR DIRECT NETWORKS

The network mechanism that selects certain network resources (e.g., a specific output port of a switch) in order to transfer a message from a source to a destination is termed *routing*. Routing can either be done through a centralized network controller, or, as it is most often the case, decentralized in the individual network switches.

Routing algorithms can be either deterministic or adaptive. During *deterministic routing,* the path to be taken through the network is determined by the source and destination addresses only. The network load and the availability of network resources do not influence the routing of a message. *Adaptive routing protocols* take the availability of network links into account as well. To support adaptive routing, multiple paths between a source and a destination have to be present in the network.

Routing deadlock occurs when a set of messages has a cyclic dependency on resources (buffers or links). Because of the problem of deadlocks in direct networks, most routing algorithms have been proposed for direct networks to avoid deadlock situations. This section therefore focuses on routing algorithms for direct networks, and only a few basic algorithms are outlined here. Basic routing algorithms for indirect networks are covered in the subsection "Indirect Networks" of the section on "Network Topologies" above.

Deterministic Routing

The most common deterministic routing strategy used in direct networks is dimension-order routing in which a message traverses the network by successively traveling over an ordered set of dimensions of path. Two examples of dimension-ordered routine are *XY* routing and *e*-cube routing.

The *XY routing algorithm* used for mesh networks routes a message always in the *X* direction first. Once it has reached its destination column, the message will be routed in the *Y* direction (of course, this method also works if messages are routed in the *Y* direction first and then in the *X* direction). This routing strategy results in deadlock-free message delivery because cyclic dependences cannot occur (21). Consider the mesh network in Fig. 9(a), and assume *XY* routing (*X* dimension first, then *Y* dimension). A message from source 2 destined to node 7 will be routed through the intermediate nodes 1 and 4 as shown in the figure. If one of the network links on that path is blocked (e.g., the link between nodes 4 and 7), the message is blocked as well. An alternative path of the same length exists through nodes 5 and 8, but this path cannot be taken because of the *XY* routing algorithm. Thus, on the one hand, *XY* routing restricts the number of paths a message can take (and therefore increases the possibility of message

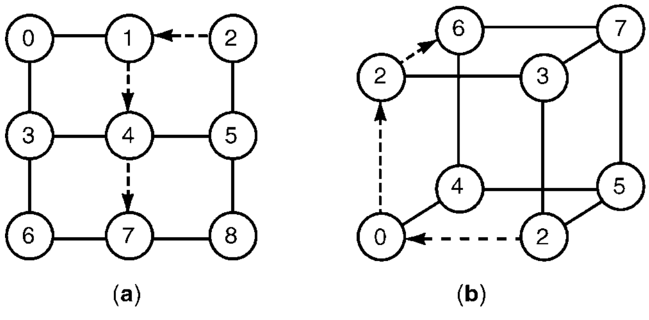


Figure 9. (a) *XY* routing in a mesh with $N = 9$, (b) *e*-cube routing in a hypercube with $N = 8$. Messages are routed in a dimension-ordered fashion.

blocking), but, on the other hand, guarantees deadlock freedom in the network (for a detailed explanation, see Ref. 19).

Similarly to the *XY* routing strategy in mesh networks, a message in a hypercube network under the *e-cube algorithm* will always traverse the dimensions of the network in the same order (e.g., cube_0, then cube_1, then cube_2, . . .). In Fig. 9(b), the transfer of a message from source node 1 to destination node 6 (over intermediate nodes 0 and 2) is shown in a hypercube with $N = 8$ using the e-cube algorithm. If a network resource on this path is blocked, the message has to wait, even though alternative paths exist (e.g., over intermediate nodes 5 and 7). However, cyclic dependences cannot occur when the e-cube algorithm is used, so that deadlocks are avoided [for a detailed explanation, see (21)].

The *e*-cube algorithm, initially proposed for hypercube networks, can be generalized for *k*-ary *n*-cubes (21). The original *e*-cube algorithm cannot guarantee deadlock freedom in these networks because of inherent cycles due to the wrap-around edge connections (see the subsection "Direct Networks" under "Network Topologies" above). Thus, in order to avoid deadlocks, the routing algorithm is not allowed to used certain edge connections. This results in some message paths that are longer than in the network with unrestricted routing, but deadlock freedom is guaranteed.

Adaptive Routing

Adaptive routing protocols can be characterized by three independent criteria: progressive versus backtracking, profitable versus misrouting, and complete versus partial adaptation (22).

Once a routing decision is made in a *progressive protocol,* it cannot be reversed. The path has to be taken even if the message might end up being blocked. In a *backtracking protocol,* routing decisions can be reversed if they lead to the blocking of a message. Thus, if a message reaches a blocked network resource (e.g., a temporarily unavailable network link), the message will track back its path taken so far to try to find an alternative route that is not blocked. This method is mainly used in circuit-switching or packet-switching direct networks with bidirectional links between nodes that enable the backtracking. Backtracking protocols are not well suited for wormhole-routing networks, because a message can be spread over several nodes, which makes it difficult to backtrack the worm.

A *profitable protocol* (also called *minimal routing protocol*) will always choose a network resource (e.g., a node output) that guides the message closer to its destination. If a message encounters a blocked link, it can only use other links that result in the same path length through the network. If those links are blocked as well, the message has to wait. This results in a minimal length of the path a message will take through a network. This routing restriction is omitted in *misrouting protocols* (also called *nonminimal routing protocols*) so that a misroute is preferred over message blocking. Thus, the length of the path a message will take can be longer than the minimum path from the source to its destination.

The two above-mentioned criteria define classes of paths that the routing algorithm can choose from. *Completely adaptive routing protocols* can use any path out of a class, while *partially adaptive* ones can only use a subset of those paths (to avoid deadlock situations). Examples of a progressive and a backtracking completely adaptive routing protocol are now given.

A very simple adaptive progressive routing protocol with a profitable path choice is the *idle algorithm*. It is based on a deterministic routing scheme (e.g., *XY* or *e*-cube routing). If the deterministic routing scheme encounters a blocked node output port, the adaptive protocol will choose a different output port that will bring the message closer to its destination. This way, a message either reaches its destination or is blocked when no other output port is available that would bring the message closer to its destination. The resulting path will always be of minimal length, and the network performance will be increased over the deterministic routing scheme because a message is allowed to take alternative paths. However, this routing protocol is not deadlock-free. Thus, if a deadlock occurs, it has to be detected by the routing algorithm (e.g., through timeouts) and dissolved. Each occurring deadlock will decrease the network performance, though, so that it is more efficient to use an adaptive routing protocol that is inherently deadlock-free.

A backtracking routing algorithm allows a message to reverse routing steps to avoid the blocking of the message. Deadlocks cannot occur, because messages will rather backtrack than wait. To avoid a livelock situation (i.e., when a message is routed indefinitely through the network without ever reaching its destination), information about path segments already taken has to be added to the message or stored in the network nodes in a distributed fashion.

A simple backtracking algorithms is the *exhaustive profitable backtracking protocol*. This protocol performs a depth-first search of the network, considering profitable network links only. If a shortest path that is not blocked exists between a source and a destination, this routing algorithm will find it. The *k-family routing protocol* speeds up the path search through a two-phase algorithm. As long as the distance of a message from its destination is larger then the parameter *k*, a profitable search heuristic is used that considers a subset of all available shortest paths only. If the distance is lower than *k*, then the exhaustive profitable search is used, which considers all available shortest paths (22).

Both routing protocols forbid misrouting, so that a nonblocked path through the network cannot always be found. *Exhaustive misrouting backtracking protocols* will always find an existing nonblocked path in a network, because messages can be misrouted. However, the search itself can degrade the network performance, especially when a nonblocked path does not exist. In this case, the routing algorithm will search the whole network before it recognizes that a path does not exist. Thus, a message may stay inside the network for quite a while and will use network resources during the search that are then unavailable for other messages.

To alleviate this search problem, the *two-phase misrouting backtracking protocol* can be used. This protocol divides

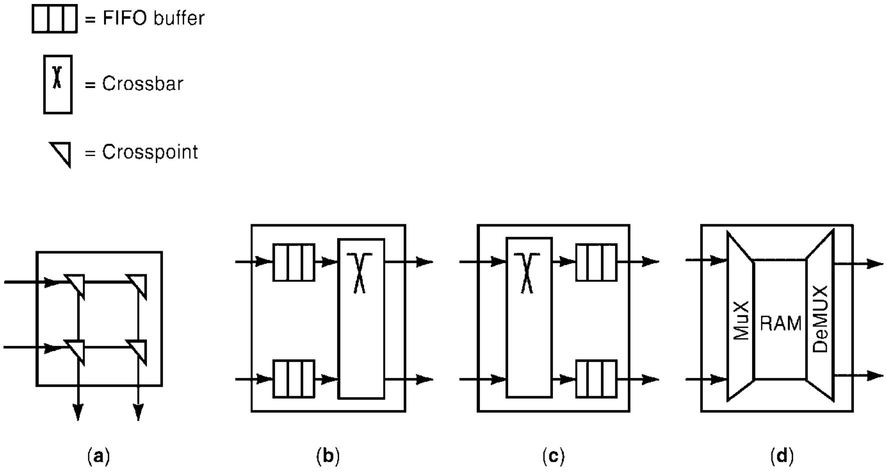


Figure 10. 2 × 2 (a) crossbar, (b) input-buffered, (c) output-buffered, and (d) central-memory-buffered switch box architectures. The placement of the buffers within a switch box has a major effect on the network performance and on the buffer requirements.

the search into two phases, similarly to the k-family routing protocol. Each phase is determined by the current distance between the message and its destination. If the distance is larger than a parameter d, then the protocol will use an exhaustive profitable search. If the message is closer to its destination than d, then the protocol switches to an exhaustive misrouting search. Because the second phase can route the message further away from its destination again, the search may switch between the two phases multiple times.

SWITCH BOX ARCHITECTURES

The architecture of the switch boxes depends on the underlying switching mechanism (see the section "Switching Techniques" above) and has a large effect on network performance. This section discusses architectural issues with respect to switch boxes and their effect on network performance.

When a connection is established in a circuit-switching network, each switch box is set in a specific switching state. For example, in the 2 × 2 switch boxes that are sometimes used to construct multistage indirect networks, there are four distinct settings for each switch: straight, exchange, upper broadcast, and lower broadcast. The *straight* setting connects the upper input port with the upper output port, and the lower input port with the lower output port. In the *exchange* setting, the upper input port is connected to the lower output port, while the lower input port is connected to the upper output port. Finally, in the *broadcast* setting, one of the input ports is connected to both switch output ports (in the *lower broadcast* the lower input port is chosen; in the *upper broadcast,* the upper input port). If during the connection establishment for a message transmission a switch box within the network already uses a setting that is different from the requested one, the connection cannot be established and will be refused. One way to implement 2 × 2 and larger switches is the *crossbar* [see Fig. 10(a)]. A $B \times B$ crossbar consists of B inputs, B outputs, and B^2 crosspoints that can connect the horizontal line with the corresponding vertical one.

In packet-switching (and wormhole-routing) networks, packets (or flits) can be blocked within the network and have to be temporarily buffered inside a switch box. The placement of these buffers within a switch box has a major effect on the network performance and on the buffer requirements. The method that results in the lowest hardware requirement is *input buffering,* where a first-in-first-out (FIFO) buffer for storing multiple packets is placed at each input port of a switch box [see Fig. 10(b)]. During each network cycle, each buffer must be able to store up to one packet and dequeue up to one packet. A packet reaching a switch box input port that cannot be transferred to an output port because that port is currently busy will be stored in that input buffer. Although these buffers are easy to implement, they have the major disadvantage of *head-of-line* (HOL) blocking because of their FIFO discipline. If the packet at the head of an input buffer is blocked, it will block all other packets in that buffer, although some of those packets might be destined to an idle switch box output port. This blocking reduces the switch box throughput significantly, especially in larger switches.

To eliminate the HOL-blocking effect, *output buffering* can be employed, where FIFO buffers reside at each switch box output port [see Fig. 10(c)]. Because, during each network cycle, up to B packets can be destined to one specific output port in a $B \times B$ switch box (one from each switch box input), an output buffer must be able to store up to B packets and dequeue up to one packet during each network cycle. Because in an output buffer only packets are stored that are destined to the same output port of that switch box, HOL blocking cannot occur. If buffers with an infinite length are assumed, a maximum switch throughput of 100% can be achieved.

To achieve high performance with output buffered switch boxes, considerable buffer space is needed. To reduce this buffer requirement, a central memory can be used. In *central-memory-buffered switch boxes,* there are no dedicated buffers at either the switch input or the output ports.

Packets arriving at a switch box input port are buffered in a central memory that is shared among all switch inputs [see Fig. 10(d)]. The central memory is divided into virtual FIFO queues of variable length (one for each output port) in which the packets are stored corresponding to their destination. The bandwidth requirement for the central memory is even higher than that for a buffer in an output buffered switch box, because during each network cycle, up to B packets have to be stored in the memory and up to B packets have to be read out of a $B \times B$ switch box. Because the length of each virtual queue is variable, virtual queues that are only lightly utilized require less memory and heavily utilized virtual queues can have more space (23). Thus, the buffer space can be very efficiently utilized, so that a smaller overall buffer space is needed as than for switch boxes with dedicated output buffers at each output port.

CONCLUSIONS

This article is a brief introduction to some of the concepts involved in the design of interconnection networks for parallel machines. See the references cited for more details. A reading list provides further sources of information.

BIBLIOGRAPHY

1. R. Duncan, A survey of parallel computer architectures, *IEEE Comput.*, **23** (2): 5–16, 1990.
2. W. C. Athas, C. L. Seitz, Multicomputers: Message-passing concurrent computers, *IEEE Comput.*, **21** (8): 9–24, 1988.
3. B. Nitzberg and V. Lo, Distributed shared memory: A survey of issues and algorithms, *IEEE Comput.*, **24** (8): 52–60, 1991.
4. M. Jurczyk and T. Schwederski, SIMD processing: Concepts and systems, in Y. Zomaya (ed.), *Handbook of Parallel and Distributed Computing*, New York: McGraw-Hill, 1996, pp. 649–679.
5. R. Duncan, MIMD architectures: Shared and distributed memory designs, in Y. Zomaya (ed.), *Handbook of Parallel and Distributed Computing*, New York: McGraw-Hill, 1996, pp. 680–698.
6. H. J. Siegel and C. B. Stunkel, Inside parallel computers: Trends in interconnection networks, *IEEE Comput. Sci. Eng.*, **3** (3): 69–71, 1996.
7. V. Cantoni, M. Ferretti, and L. Lombardi, A comparison of homogeneous hierarchical interconnection structures, *Proc. IEEE*, **79**: 416–428, 1991.
8. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, San Mateo, CA: Morgan Kaufmann, 1992.
9. I. Stojmenovic, Direct interconnection networks, in Y. Zomaya (ed.), *Handbook of Parallel and Distributed Computing*, New York: McGraw-Hill, 1996, pp. 537–567.
10. H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, 2nd ed., New York: McGraw-Hill, 1990.
11. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, Cambridge, MA: MIT Press, 1972.
12. H. P. Katseff, Incomplete hypercubes, *IEEE Trans. Comput.*, **C-37**: 604–608, 1988.
13. W. J. Dally, Performance analysis of k-ary n-cube interconnection networks, *IEEE Trans. Comput.*, **C-39**: 775–785, 1990.
14. D. P. Agrawal, Graph theoretical analysis and design of multistage interconnection networks, *IEEE Trans. Comput.*, **C-32**: 637–648, 1983.
15. H. Ahmadi and W. E. Denzel, A survey of modern high-performance switching techniques, *IEEE J. Sel. Areas Commun.*, **7**: 1091–1103, 1989.
16. K. Y. Lee and D. Lee, On the augmented data manipulator network in SIMD environments, *IEEE Trans. Comput.*, **37**: 574–584, 1988.
17. H. J. Siegelet al., Using the multistage cube network topology in parallel supercomputers, *Proc. IEEE*, **77**: 1932–1953, 1989.
18. C. E. Leiserson, Fat-trees: Universal networks for hardware-efficient supercomputing, *IEEE Trans. Comput.*, **C-34**: 892–901, 1985.
19. L. M. Ni and P. K. McKinley, A survey of wormhole routing techniques in direct networks, *IEEE Comput.*, **26** (2): 62–76, 1993.
20. W. J. Dally, Virtual-channel flow control, *IEEE Trans. Parallel Distrib. Syst.*, **3**: 194–205, 1992.
21. W. J. Dally and C.L. Seitz, Deadlock-free message routing in multiprocessor interconnection networks, *IEEE Trans. Comput.*, **C-36**: 547–553, 1987.
22. P. T. Gaughan and S. Yalamanchili, Adaptive routing protocols for hypercube interconnection networks, *IEEE Comput.*, **26** (5): 12–23, 1993.
23. M. Jurczyket al., Strategies for the implementation of interconnection network simulators on parallel computers, *Int. J. Comput. Syst. Sci. Eng.*, **13** (1): 5–16, 1998.

READING LIST

Books That Cover Interconnection Networks

J. Duato, S. Yalamanchili and L. Ni, *Interconnection Networks: An Engineering Approach*, Los Alamitos, CA: IEEE Computer Society Press, 1997.

F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, San Mateo, CA: Morgan Kaufmann, 1992.

I. D. Scherson and A. S. Youssef (eds.), *Interconnection Networks for High-Performance Parallel Computers*, Los Alamitos, CA: IEEE Computer Society Press, 1994.

T. Schwederski and M. Jurczyk, *Interconnection Networks: Structures and Properties*(in German), Stuttgart: Teubner, 1996.

H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, 2nd ed., New York: McGraw-Hill, 1990.

K. J. Thurber (ed.), *Tutorial: Distributed Processor Communication Architecture*, New York: IEEE Press, 1979.

A. Varma and C. S. Raghavendra (eds.), *Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice*, Los Alamitos, CA: IEEE Computer Society Press, 1994.

C.-L. Wu, T. Y. Feng (eds.), *Tutorial: Interconnection Networks for Parallel and Distributed Computing*, Los Alamitos, CA: IEEE Computer Society Press, 1984.

Books and Articles That Cover Interconnection Networks in Commercial Parallel Processing Systems

J. Beecroft, M. Homewood, and M. McLaren, Meiko CS-2 interconnect, Elan-Elite design, *Parallel Comput.*, **20**: 1626–1638, 1994.

T. Blank, The MasPar MP-1 architecture, *IEEE Int. Comput. Conf. CompCon*, 1990, pp. 20–24.

R. Esser and R. Knecht, Intel Paragon XP/S—architecture and software environment, in H. W. Meurer (ed.), *Supercomputer '93*, Berlin: Springer-Verlag, 1993.

K. Hwang, *Advanced Computer Architecture*, New York: McGraw-Hill, 1993.

K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, New York: McGraw-Hill, 1984.

R. E. Kessler and J. L. Schwarzmeier, Cray T3D: A new dimension for Cray Research, *IEEE Int. Comput. Conf. CompCon*, 1993 pp. 176–182.

N. Koike, NEC Cenju-3: A microprocessor-based parallel computer multistage network, *8th Int. Parallel Process. Symp.*, 1994 pp. 393–401.

C. B. Stunkel et al., The SP2 high-performance switch, *IBM Syst. J.*, **34** (2): 185–202, 1995.

L. W. Tucker and G. G. Robertson, Architecture and applications of the Connection Machine, *IEEE Comput.*, **21** (8): 26–38, 1988.

Papers That Cover Network Fault Tolerance

G. B. Adams II and H. J. Siegel, The extra stage cube: A fault-tolerant interconnection network for supersystems, *IEEE Trans. Comput.*, **C-31**: 443–454, 1982.

G. B. Adams III, D. P. Agrawal, and H. J. Siegel, A survey and comparison of fault-tolerant multistage interconnection networks, *IEEE Comput.*, **20** (6): 14–27, 1987.

G.-M. Chiu and S.-P. Wu, A fault-tolerant routing strategy in hypercube multicomputers, *IEEE Trans. Comput.*, **C-45**: 143–154, 1996.

M. Jeng and H. J. Siegel, Design and analysis of dynamic redundancy networks, *IEEE Trans. Comput.*, **C-37**: 1019–1029, 1988.

V. P. Kumar and S. M. Reddy, Augmented shuffle-exchange multistage interconnection, *IEEE Comput.*, **20** (6): 30–40, 1987.

R. J. McMillen and H. J. Siegel, Routing schemes for the augmented data manipulator network in an MIMD system, *IEEE Trans. Comput.*, **C-31**: 1202–1214, 1982.

K. Padmanabhan and D. H. Lawrie, A class of redundant path multistage interconnection networks, *IEEE Trans. Comput.*, **C-32**: 1099–1108, 1983.

Papers About Comparing Interconnection Networks

K. J. Liszka and J. K. Antonio, H. J. Siegel, Problems with comparing interconnection networks: Is an alligator better than an armadillo? *IEEE Concurrency*, **5** (4): 18–28, 1997.

Papers About Trends in Interconnection Networks

H. J. Siegel and C. B. Stunkel, Inside parallel computers: Trends in interconnection networks, *IEEE Comput. Sci. Eng.*, **3** (3): 69–71, 1996.

Michael Jurczyk
University of Missouri–Columbia
Columbia, Missouri
Howard Jay Siegel
Purdue University
West Lafayette, Indiana
Craig Stunkel
IBM T. J. Watson
Research Center
Yorktown Heights, New York

LCD DESIGN TECHNIQUES

Liquid crystal displays (LCDs) play a crucial role in almost all technology scenarios based on human interfaces as being the preferred device for visual information rendering in a wide variety of application domains. With respect to a few popular examples, LCDs are extensively used for video output in personal computers, portable phones, photo and video cameras, as well as diverse home entertainment multimedia appliances. The impact of LCD performance on an important share of the worldwide consumer electronics market justifies the effort devoted to clever design of LCD-based equipment. Broadly, LCD designers have to cope with two classes of realization issues, concerning both the choice of the LCD technology and the implementation of the LCD driving electronics. The technology selection is in many ways independent of the targeted display, being either a high-end PC monitor or a cellular phone display panel. Conversely, several display driving methodologies are available at a given display technology, each one posing tradeoffs among ease of implementation, realization cost, and overall optical performance. In general, the term *driving scheme* encompasses all features cooperating with the generation of the electrical signals applied to the display panel in order to build up the desired image. The generated picture commonly acts as an individual video frame out of an arbitrarily complex frame sequence: Hence, all mechanisms used for producing a “static” picture (i.e., refreshed continuously and invariably over the display) can be extended straightforwardly to video streaming (where the frame information varies over subsequent frames). Usually, a display driver can be programmed to sort the best matching display between driving scheme and display application. The definition of a driving scheme includes the panel scanning pattern as a major component. However, a driving scheme is also made of a combination of measures put into action in order to mitigate the effects of perceptible visual artifacts. An outstanding example of such techniques is the driving polarity inversion, which will be extensivley referred to in the following parts in conjunction with the presentation of the most dangerous optical artifacts.

Panel technology and driving mode directly affect the design of the driver electronics, which motivates the interest in LCD-specialized design flows. LCD engineers are provided with customized evaluation tools that can be used to assess the impact of vital design choices as early as possible throughout the product lifecycle. Joint LCD-driver simulation environments, for instance, are highly recommended to achieve the optimal driving-scheme/display match.

THE LCD ARENA

To date, the market of LCDs puts forward a huge amount of typologies, including monochromatic, color, passive-matrix, active-matrix, and organic-material-based panels. Nonetheless, all LCDs are built on liquid crystals, i.e., materials capable of modifying their microscopic spatial orientation under the effect of comparatively small electric fields (1). The observation that a light beam directed toward the liquid crystal cell is differently diverted depending on the particular orientation of the crystals themselves triggered off the pervasive development of LCD technologies over the past few decades. LC cells are not, by themselves, spontaneous light sources, but their operation depends on the reflection or absorption of light originating from some sort of external source. The way the display interacts with such source, as well as the techniques deployed for pixel addressing and image generation over the panel, allow LCDs to be classified and several well-known families to be distinguished. Figure 1 represents a broad classification of those families, as discussed in the next subsection.

The Twisted Nematic Technology

The working principle of displays based on the twisted nematic (TN) technologies (2,3) is depicted in Fig. 2, where the structure of a single LCD cell is shown. The indium tin oxide (ITO) layers are those used to realize the cell driving electrodes. Basically, the light beam reaching the cell is allowed to pass through the output polarizer provided that it is properly twisted by the spatial orientation of the liquid crystal in the nematic layer inside the cell. Applying a proper voltage at the boundaries of the cell can alter such orientation, so that the cell would in turn shield the nontwisted light beam. The cell organization is replicated to construct matrix-like panels where each cell represents a pixel. Pixel ON/OFF states, corresponding to dark/bright pixel in black-and-white displays, can be selectively driven by applying specific voltage patterns at any pixel location. The voltage control is realized via an array of connection electrodes accessible by the driving circuits external to the display. The TN technology has been used first in passive-matrix LCDs, although a significant evolution in the LCD design philosophy must be ascribed to the introduction of active-matrix driving modes. Passive-matrix and active-matrix LCDs essentially differ in the nature of their electrodes and in the way individual pixels are addressed.

Super Twisted Nematic Technology. In practice, pure TN technology has been replaced by super twisted nematic (STN) technology. By doping the nematic layer with an optically active material, the STN technology is characterized by a greater twist-angle impressed to the light beam by the LC layer, which achieves higher optical contrast, increases the chromatic yield, and produces faster-responding displays. The introduction of STN technology has played a particularly important role for passive-matrix-addressed displays, where under some fixed optical conditions, the multiplex ratio, i.e., the number of lines to be addressed in the pixel matrix for image generation, can be only increased if the response of the liquid crystal is faster.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

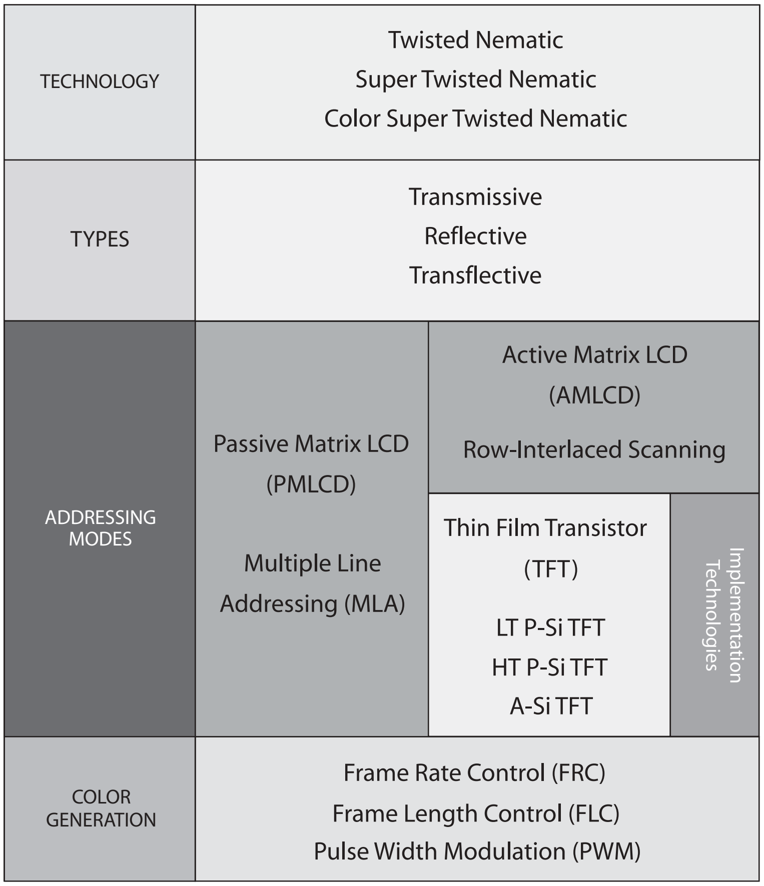

TECHNOLOGY
Twisted Nematic
Super Twisted Nematic
Color Super Twisted Nematic
TYPES
Transmissive
Reflective
Transflective
ADDRESSING MODES
Passive Matrix LCD (PMLCD)
Multiple Line Addressing (MLA)
Active Matrix LCD (AMLCD)
Row-Interlaced Scanning
Thin Film Transistor (TFT)
LT P-Si TFT
HT P-Si TFT
A-Si TFT
Implementation Technologies
COLOR GENERATION
Frame Rate Control (FRC)
Frame Length Control (FLC)
Pulse Width Modulation (PWM)

Figure 1. Overview of the most relevant liquid crystal display technologies and implementation techniques.

Color Super Twisted Nematic Displays. For color generation, each LC cell is physically replicated three times per pixel. Light passing through individual cells that belong to the same pixel (the so-called subpixels) is then selectively filtered through red, green, and blue channels, respectively. It is important to stress the absolute independence of subpixels from the voltage generation point of view, so the resulting color display can be simply thought of as an extended pixel-matrix with respect to its pure gray-scale counterpart. It is the spatial integration capability of the human eye over adjacent subpixel locations to ensure that individual channels out of the same pixels are concretely perceived as a single color by the observer.

Transmission-Voltage Curves

The light-transmitting capability of liquid crystals is usually characterized by plotting the LC cell transmission as a function of the applied voltage to form the electro-optical characteristic of the liquid crystal (Fig. 3). In the LC jargon, the expressions *threshold* and *saturation voltage* are used also to define the voltage below which the cell light transmission is none, and that above which the light transmission cannot be increased further (no matter the magnitude of the applied voltage), equaling 100%, respectively. Figure 3 also explains a major difference between TN and STN technologies: STN liquid transmission-voltage curves are generally steeper than the corresponding TN curves, so ON/OFF state transitions are appreciably faster.

Transmissive and Reflective LCDs

A first, coarse classification of LCDs distinguishes between transmissive panels, where the light source is positioned beyond the display with respect to the observer and the light beam is filtered depending on the orientation of the liquid crystal at a particular point in time, and reflective panels, where the *external* environmental light is reflected by a mirror located behind the display LC panel and afterward filtered by the crystals. Reflective displays obviously promote low-power operation since a built-in light source is not required; yet their optical performance tends to degrade when they are used in dim areas. For that reason, they are commonly avoided for indoor applications. Transflective LCDs offer a tradeoff between the performance of both types; they are increasingly penetrating into the market of displays for mobile appliances, offering guarantees of good performance under almost all illumination conditions.

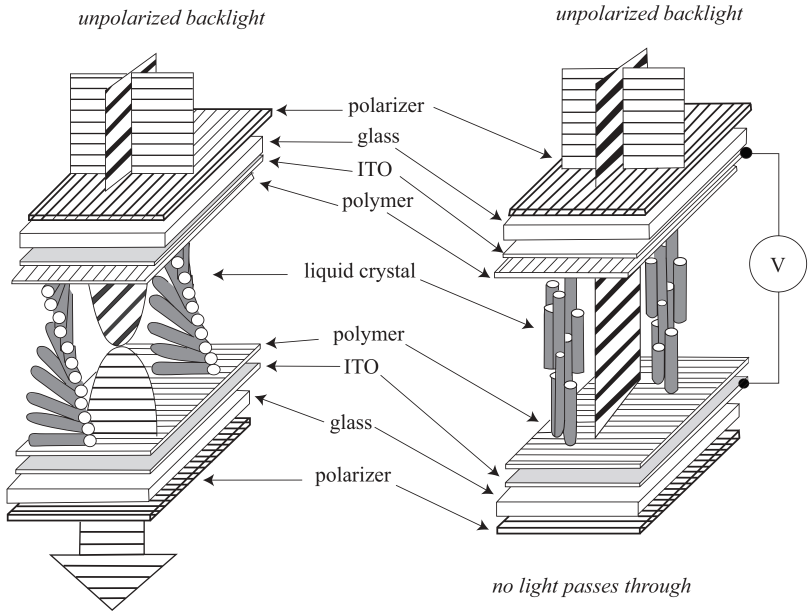


Figure 2. Building principle of a liquid crystal cell for display realization: light transmission and shielding depending on the applied voltage is also shown.

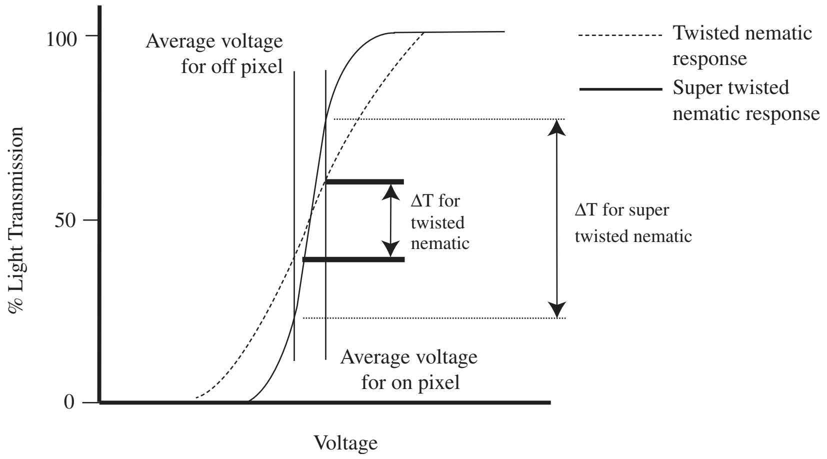


Figure 3. Exemplar transmission-voltage curves for TN and STN LCDs. The increased steepness of the STN characteristic is apparent.

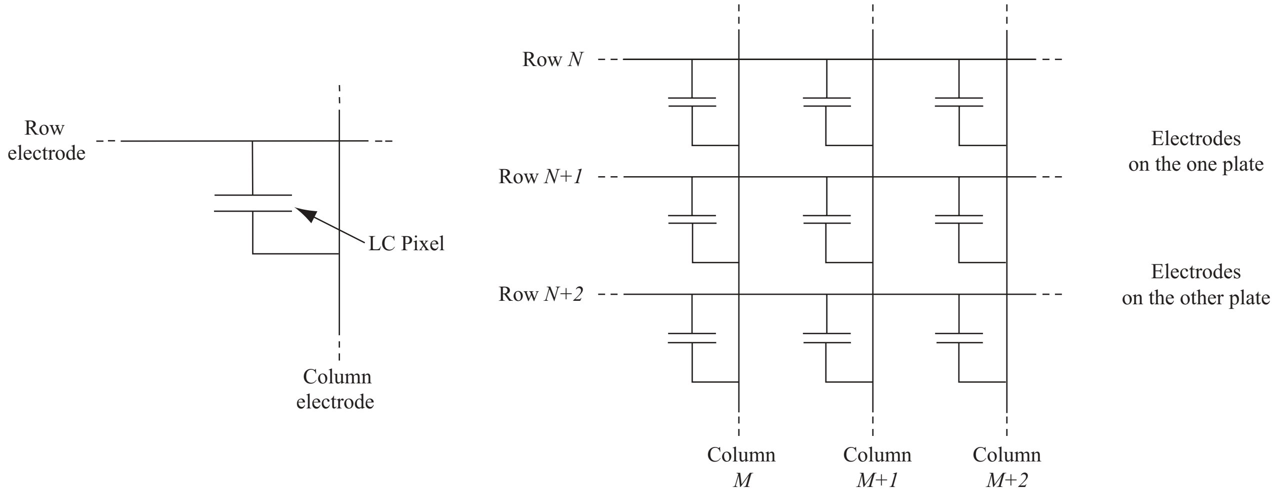


Figure 4. Pixel model and display matrix for PMLCDs.

LCD Addressing Modes

Passive-Matrix Displays. In passive-matrix LCDs (PMLCD), the electrodes are implemented by thin metal-oxide stripes layered over the surfaces of the LC panel. In particular, the mash of electrode stripes laid on one side of the panel is orthogonal to those placed on the other side. LC cells (i.e., individual pixels of the display) correspond to crossing points between electrodes belonging to different sides. On the whole, the display can be viewed as a pixel matrix, with each pixel being equivalent to a capacitor connected between electrode stripes from different plates (Fig. 4). Constructing a proper voltage between the electrodes that address that particular cell drives the pixel states ON or OFF. During each frame scan, i.e., over the time needed by the driving electronics to address all other rows and columns in the display, the ON/OFF states of the pixels must be maintained. The effectiveness of the approach rests on the inherent persistency of the human visual system, which ensures correctness of optical perception as long as the display scan period and the time each pixel's state is held are properly synchronized. Flickering would otherwise affect the perceived image: This is generally avoided by taking special measures at the driving scheme level. Inter-electrode crosstalk is another common but undesirable side effect of passive-matrix addressing, which manifests as halos around the displayed picture.

Active-Matrix Displays. The development of active-matrix addressing decreased most image quality flaws by associating a dedicated transistor switch at each pixel in the display. The switch allows the pixel to be driven ON/OFF without being affected by the electrical activities of the other pixels in the row. In other words, the pixel control is no longer committed to a combined row/column electrical driving performed on a frame basis: The pixel state can be held independently of the frame refresh period or the persistency interval, just as long as the switching transistor state is maintained. From the technological perspective, the increased complexity of active-matrix LCDs (AMLCDs) is abundantly compensated by the tremendous gain in the optical quality as much as by important savings in the display power consumption. In a fact, AMLCDs currently represent the mainstream solution in the market of displays, especially when wide-area panels are regarded.

Thin-film transistor (TFT) displays embody a popular implementation of the active-matrix addressing principle. In TFT displays, the switching transistors are integrated within thin plastic layers deposited over the internal surfaces of the glass panels, and they directly delimit the liquid crystal core. TFT displays can be diversely realized. The low temperature polysilicon TFT is a new technology allowing large-size panels with easy integration of the driver circuits. Integration of the driving circuits is also possible with the high temperature polysilicon TFT, which is a MOS-IC-like process applicable to small-size panels. Finally, large-area panels are based on the amorphous silicon TFT technology, which is the most mature and most popular technology.

Thanks to the constructive procedure of AMLCDs, parasitic capacitive effects are minimized with respect to PMLCDs. Furthermore, TFT displays are much brighter than PMLCDs, their viewing angle reaches up to 45 degrees with respect to the display axis, and their response time is one order of magnitude shorter than that of PMLCDs. On the other hand, constructive data for TFT displays demonstrate in what sense complexity is a major point: For instance, even a small 132 × 176-pixel color display requires up to 70,000 transistors for active-matrix addressing!

Advanced Technologies

Over the past few years, the poor performance exhibited by PMLCDs in appliances featuring video streaming facilities motivated display manufacturers to massively migrate toward active-matrix TFT panels. Currently, not only do AMLCDs represent the favorite solution for large display realization, but they are pervasively exploited in portable devices as well, such as last-generation high-end video cellular phones. Alternative technologies with respect to liquid crystals are also starting to creep into the market of displays. A particular class of devices based on organic light-emitting diodes (OLEDs) currently yields competitive performance, especially when flexibility (OLED displays can be significantly small and thin) and low power operation turn out to be important factors. In OLEDs, a layer of specific organic polymers placed between the driving electrodes is responsible for the light emission without external sources. For the time being, the use of OLEDs in portable battery-operated displays is only hindered by the comparatively short lifecycle of the embedded organic materials.

PASSIVE-MATRIX LCDS

Scanning Modes

The *driving voltage* applied to a PMLCD's pixel to modify the crystals orientation is something more than a mere constant voltage through the LC cell. In fact, PMLCDs are operatively sensitive to the *root-mean-square value* (rms) of some properly arranged steering voltage waveforms (4). The rms is the physical quantity actually responsible for the light transmission as readable from a transmission-voltage curve. Therefore, the driving-scheme designer chiefly focuses on alternative ways of constructing the desired rms value over the available driving time slots. The maximum allowable time length of such a driving window is an important constraint, as it is directly related to the avoidance of undesired visual artifacts that affect correct image displaying. Most driving schemes imply tradeoffs between performance and design factors, such as output image contrast, color gamut, visual artifact occurrence, driver hardware complexity, and power dissipation. In particular, hardware parameters, such as the number and type of the driving voltages to be built up or the complexity of the digital switching logic used for waveform generation, are directly involved.

The basic scanning mode is the Alt&Pleshko approach, which is essentially a simple one-row-at-a-time addressing scheme: Proper voltage pulses are sequentially sent over the row electrodes to select individual rows until the whole display has been scanned over a frame. During each row pulse, the column electrodes are driven according to the pixel data in order to construct the desired rms value at every location. Currently, the importance of the Alt&Pleshko technique has nothing to do with implementation, but is only limited to providing a better understanding of more sophisticated solutions derived from it. However, it is worth citing the so-called improved Alt&Pleshko technique, where non-negative column voltages and lower supply voltages are exploited to reduce power consumption and driver circuit area.

Significant progress in LCD driving rests on multiple-line-addressing (or multiple-row-addressing) methods (equally referred to as MLA or MRA techniques). With MLA, p rows are concurrently driven through sets of p orthogonal digital row functions (scan signals) $F_i(t)$. As a result, the total frame scanning period is automatically reduced. In typical settings, the scan signals are piecewise constant analog voltage waveforms; i.e., their repetition period is slotted into predefined equal-length intervals during which they take constants values F_i. The number of the time slots over a scan signal repetition period is usually fixed to be equal to p, although different choices are viable. Orthogonality requires that:

$$\begin{cases} \overline{F_i \cdot F_j} = \dfrac{1}{T} \cdot \displaystyle\int_o^T F_i \cdot F_j \cdot dt = F^2, & i = j \\ \overline{F_i \cdot F_j} = 0, & i \neq j \end{cases} \qquad (1)$$

The column functions (data signals) are constructed by combining the scan signals properly. Their value at a given time depends on the ON/OFF states of the pixels they are meant to activate, as follows:

$$G_j(t) = \frac{1}{\sqrt{N}} \sum_1^p \mathrm{a_{ij} F_i(t)} \quad i, j = 1, \ldots, p \qquad (2)$$

Orthogonality of the scan signals ensures that individual pixels remain unaffected by the state of the others along the same column. The advantages of MLA include the power savings achievable because of the moderate supply voltages required by the technique and the possibility of reducing the frame frequency without fearing "frame response." In the LC jargon, frame response is referred to as the relaxation of the liquid crystal directors over a frame time, which leads to contrast lowering and image flickering. By means of MLA, the LC is pushed several times within a frame so that virtually no relaxation occurs: The image contrast can be preserved, the flicker restrained, and the artifacts like smearing on moving objects (e.g., while scrolling) are suppressed by eventually adopting faster responding LC material. On the downside, all of this technology comes at the expense of an increase in the number of driving voltages to be generated (three row voltages and p + 1 column voltages are needed) and of more complex driver realizations.

Alternative MLA Schemes

The sets of the orthogonal scan signals used in MLA are assigned through matrices. Each signal is described along the rows, whereas each column shows the constant normalized value (+1, −1, or 0) assumed by the resulting waveform at the corresponding time slot. Theoretically, the matrices can be rectangular, where the number of rows indicates the number of concurrently addressed display rows (p), and the number of columns indicates the number of MLA scans needed to cover the whole panel (being in turn equal to the number of time slots composing each orthogonal signal). Different types of matrices that meet the orthogonality constraints are available and used diversely in practical implementations of MLA schemes (5). A first class is the set of Walsh functions, coming up as 2^s orthogonal functions derived from Hadamard matrices. The class of the so-called "Walking –1" functions is also used extensively: They are

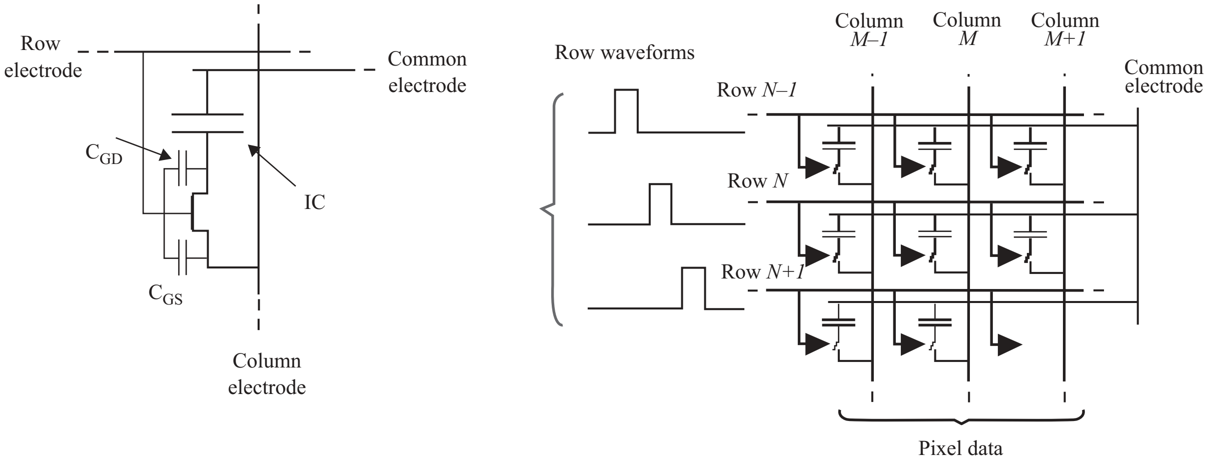


Figure 5. Pixel model and display matrix addressing for AMLCDs.

built up of p−1 positive pulses (+1) and 1 negative pulse (−1) shifting right or left from one function to the other. Hadamard and Walking –1 matrices, like many other standardized patterns, only contain 1 and −1 entries. However, since the number of nonzero entries in the matrix columns plus one corresponds to the number of column voltage levels used for waveform generation, it is desirable to introduce zeroes in the matrices. In this respect, conference matrices with one zero per row represent a smart solution.

Once the matrix structure has been chosen, the value of p must be also set, i.e., the number of rows addressed in parallel. The choice is dictated by optical performance considerations: It is widely agreed that for p up to 8, frame response is effectively suppressed, so that p is usually set to 2, 3, or 4 in commercial mobile display drivers (for ease of reference, 2-MLA, 3-MLA, or 4-MLA, respectively). For 2-MLA and 4-MLA, either Walsh or Walking –1 functions can be taken, whereas 3-MLA works with 4 × 4 conference matrices (the 0 entry corresponding to one nonselected row out of 4). Also, a particularly interesting solution exists, sometimes referred to as virtual-3-MLA, which is brought in as a modification of the 4-MLA scheme. In fact, virtual-3-MLA is identical to 4-MLA when the driver operation is regarded, but the fourth *rowdriver output* out of every set of four rows is *not* connected to the display but is just left open. On the display side, row number 4 is connected to the fifth driver row output, and so on for the remaining ones. It can be calculated that with virtual-3-MLA, only two voltage levels are required to construct the data signals, which represents significant improvement, making the driver less complex and reducing the power dissipation.

The rationale of MLA does not entail any built-in measure to guarantee optical uniformity in the' brightness of the pixels. Extended features are commonly then added up on top of the basic MLA scheme. The drive pattern switching is a popular example consisting of the periodical interchange of the set of used orthogonal functions; the change usually is scheduled to occur before every p-row addressing sequence.

ACTIVE-MATRIX LCDS

Addressing Concept

In AMLCDs, each pixel within the display matrix can be accessed only when the corresponding switch is closed. Pixel driving is realized by means of three separate electrodes, as shown in Fig. 5, where a circuit model of the pixel has been worked out. The concept of AMLCD driving is theoretically simple with respect to PMLCD: Adjacent rows are addressed sequentially by applying a positive pulse to the row electrodes. The row electrodes are connected directly to the switches gates and then are used to switch ON/OFF at the same time for all pixels along a given row. In turn, the column electrodes are used to build up the voltage throughout the LC cell according to the pixel data. In practical scenarios, row-interlaced patterns are used in place of the rudimentary row-sequential addressing, with beneficial effects with respect to the driver overall power consumption. Even if the addressing principle is straightforward, various dedicated countermeasures usually are exploited to target AMLCD-specific drawbacks, which complicate the realization of the driving schemes.

Effects of NonIdealities in Active-Matrix Addressing

Parasitic gate-source and gate-drain capacitances of the switching transistors have been explicitly drawn in Fig. 5. They are responsible for several undesired consequences that affect the display dynamic operation. The gate-drain capacitance, for instance, brings in an appreciable overshoot that distorts the driving voltage throughout the pixel cell at the switching point of the row driving pulses. The mechanism is shown in Fig. 6 for a typical voltage waveform: The effect is known as “kickback effect” and is practically caused by some extra charge loaded into the pixel capacitance. The overshoot, unless mitigated purposely, results in some sort of visible display flickering. A common technique used to fight the kickback effect is described as “common electrode modulation”: In essence, the common electrode is driven so as to compensate for the additional

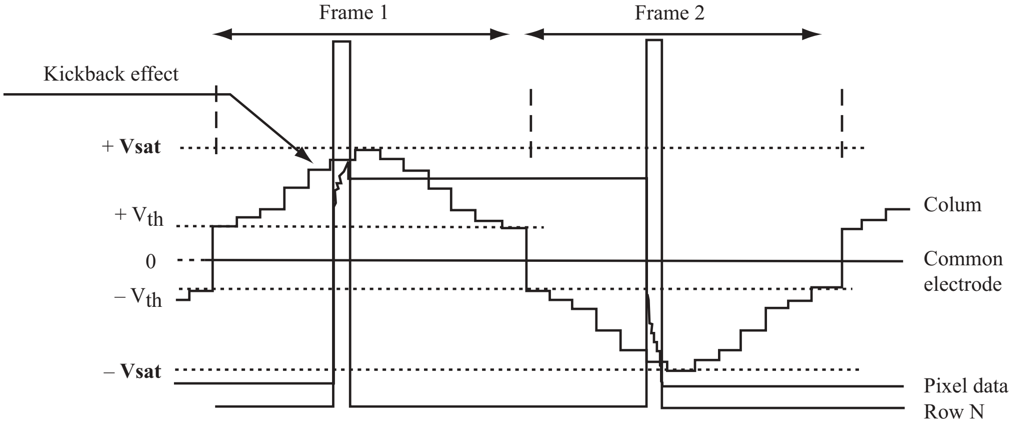


Figure 6. Driving waveforms affected by the "kickback effect" in AMLCDs panels.

spurious charge injection affecting the pixel capacitance at critical time points within the frame.

The kickback phenomenon is not the sole effect of the active-matrix nonidealities. The quality of the display is also influenced by the sensitivity of the LC capacitance to the applied voltage and by different forms of leakage currents, generally depending on either the liquid crystal itself or the technology of the active devices. The LC capacitance variability often nullify the advantages of common electrode modulation. On the other hand, leakage determines a loss in contrast and compromises the visual uniformity of the displayed pictures, as well as inducing vertical crosstalk among pixels. Both LC capacitance variations and leakage can be controlled by using a storage capacitor. The storage capacitor is connected between the switch drain and the row electrode of the previous or next display row (or to a separate electrode), so as to work in parallel with the pixel's real capacitance. Kickback correction is then made more effective, as the increase in the overall capacitance makes the cell less sensitive to any parasitic current. Another specific measure against leakage currents in the active device is the driving voltage polarity inversion, which is a technique that belongs to a broad class of polarity inversion methods used extensively to mitigate different forms of optical artifacts.

GRAY TONES AND COLOR GENERATION TECHNIQUES

The concept of ON/OFF pixel driving is suitable for mere black- and- white displays. Extending this principle to gray-scale panels requires that different gray-levels be constructed throughout a sequence of black/white states driven over consecutive frames. *Frame rate control* (FRC) serves this purpose: Sequences of N frames (N-FRC) are grouped together to compose a superframe. Over each superframe, the sequence of black- and- white states created consecutively at each pixel are perceived as homogeneous gray tones thanks to the persistency of the human visual system (3). Proper operation only requires that the superframe frequency, forced to equal the frame frequency divided by N, be above a minimum admissible value: Over a 50-Hz superframe frequency usually is agreed on for flicker-free image visualization. A refinement of the FRC method is *frame length control*, where different gray tones are generated over a superframe by varying the time–length ratio between adjacent frames in the superframe (and thereof the duration of individual black/white phases).

FRC is a very basic gray shading solution, so that cooperating schemes usually are combined in industrial drivers to enhance the color gamut: A common design choice is to modulate the data signal pulses to enrich the color resolution (*pulse width modulation*, PWM). As for the hardware, joint PWM-FRC is costly in terms of extra chip complexity, but it successfully cuts off the power consumption. The programmable gray shades are defined by means of a gray-scale table (GST), which specifies the sequence of ON/OFF state scheduled to produce a given tone. An important concern in designing the color generation mechanism is the smart configuration of such table: For instance, when applying a mixed FRC-PWM approach, pure-FRC color tones (i.e., gray tones obtained without in-frame shade modulation) should be avoided strictly in the GST to prevent awkward perceptible artifacts.

Generalization of the gray shading techniques to color displays is straightforward, as color channels are created by diverse color filtering at each subpixel, without any extra complexity on the driver side.

OPTICAL PERFROMANCE IN LCD DRIVING

Frequency Dependence of the Electro-Optical Characteristic

When thinking of optical performance, the main aspect to be considered is the non-negligible dependence of the LCD electro-optical transmission curve on the frequency of the applied signals, which can be modeled in different ways based on the particular view one may want to stress. At the physical level, the frequency dependence of the LC characteristic can be ascribed to frequency drifts of the threshold voltage. More deeply, threshold frequency variations can be traced back to frequency shifts of the liquid crystal dielectric constant, which will most likely show discrepancies between the programmed rms voltage across LC cells and the actual driving level. Extensive data are categorized by LCD manufacturers, which helps in selecting the most suitable LC technology. In a fact, the dynamic operation of an LCD is jointly determined by the cross-correlation among several constructive and material-dependent parameters (6,7). Not only must LC-specific factors be regarded, but also information about the display module arrangement is equally important. As an example, desired versus actual voltage mismatching may also arise from the capacitive coupling between row and column electrodes, for both individual pixels and between adjacent pixels.

For design purposes, all frequency dependencies can be translated into an input–output frequency-selective relationship between the LCD applied driving waveforms and the effective voltage signals controlling the LC cells. This approach is the most common, which also suits the electrical modeling of the LC cell as a simplified passive RC network. By studying the display response, several design rules can be worked out to serve as a reference when designing the driver architecture. First and foremost, it is desirable that the frequency band of the drive signals be narrow to prevent uneven frequency filtering and optical distortion. MLA guarantees band collimation since most of the spectrum becomes concentrated around p times the frame frequency. However, experimental results show that when other schemes (such as PWM) are mounted on top of MLA, frequency multiplication may turn out to reinforce more than to suppress artifacts. Polarity inversion is another established methodology for bandwidth reduction, which entails that the signs of both the row and the column signals be periodically inverted, with an inversion period set on a superframe, frame, or block-of-N-lines basis. Dot inversion also is possible in AMLCDs, where the inversion takes place from one pixel location to the adjacent one. Whatever inversion is used, the lowest frequency of the spectrum is upshifted, and the DC offset in the driving signals is suppressed. The latter is another important result, since the DC component is a primary cause of LC degeneration and of panel lifetime reduction.

Keeping the waveform frequency spectrum under control also is vital with respect to energy awareness. Curbing the chip power consumption is possible when frame/superframe frequencies are low enough: In fact, the superframe frequency can be made even lower than 50 Hz if phase mixing is supported.

Phase Mixing. Phase mixing exploits the spatial low-pass filtering capability of the human eye to construct the same gray level in adjacent pixels (blocks of pixels) by driving the same number of ON/OFF states (phases) but throughout different sequences over the superframe. If individual ON/OFF states out of the GST are referred to as *phases* for each gray tone, *phase mixing* implies scheduling different phases during the same frame for each pixel out of a region of adjacent ones. Phase mixing better distributes the voltage switching activities over the columns and produces a lowering of global frequency. To yield the best optical performance, phase mixing is typically applied on an RGB-subpixel basis (subpixel blocks instead of pixel blocks) and the phase pattern (which phases are driven at which position) is switched from one pixel (respectively, subpixel) block to another. The designer wanting to implement phase mixing arranges a phase mixing table holding the basic phase sequencing for pixel blocks within the display matrix together with the related phase switching rule (which phase follows a given one at any location). The setting of the phase switching table has a strong impact on the chip functionalities, so that it must be granted particular care.

Flicker

The meaning of flicker in the scope of LCD artifacts is familiar; however, the reasons why flicker affects display visualization and the countermeasures needed to remove it might be less clear. Flicker generally stems from an incorrect distribution of the driving waveforms frequency spectrum: As it is, time-uneven or amplitude-unbalanced contributions to the rms value over a frame are likely to bring about flicker. Hence, common solutions to suppress flicker are part of those general measures used to regulate the frequency spectrum of the voltage waveforms: Lowering the frame frequency, using MLA-based driving (for flicker caused by uneven rms contribution over a frame), and embedding some smart phase-mixing scheme (for flicker caused by uneven rms contributions over a superframe) currently are deployed in industrial PMLCD modules. As for AMLCDs, flicker always represents a crucial concern because of the kickback effect unless dedicated measures are taken, like the common electrode modulation technique or the use of an additional storage capacitor as described above.

Crosstalk

By crosstalk we define all pattern-dependent effects of mutual interference among the gray-scale values of pixels (3,8). Those effects tend to grow worse with increasing display size, higher resolution, and faster responding LC, all of which unfortunately are features that the display market's evolution is more and more heading toward. Diverse mechanisms are responsible for crosstalk, although they can be connected generally to the frequency-selectiveness of the LC response.

Static Crosstalk Artefacts. It is commonly agreed that *static* crosstalk is defined as all sorts of crosstalk-related visual artifacts affecting the displaying of single (in this sense, *static*) pictures. Therefore, frame switching over time such as in video streaming is not considered. Static crosstalk appears as differences in the brightness of theoretically equal gray-scale pixels and manifests in multiple manners. Simply stated, at least three types can be distinguished: vertical crosstalk, horizontal crosstalk, and block shadowing. Vertical crosstalk usually hinges on different frequency contents of different column waveforms. It is growing more and more important with the increasing steepness of the LC transmission-voltage curve as required for acceptable contrast in visualization. Horizontal crosstalk occurs when differences in the LC dielectric constant for black- and-white pixels induce spatially asymmetrical capacitive coupling between rows and columns. The amount of perceptible artifacts depends on the width of dark/bright horizontal blocks along a row. Finally, when current spikes result from symmetrically and simultaneously changing column waveforms in areas where sizable blocks of darker pixels determine different coupling between rows and columns, vertical block shadowing is likely.

Dynamic Crosstalk Artefacts. In conjunction with static artifacts, LCD modules supporting video streaming may be affected by dynamic crosstalk. The exact characterization of dynamic crosstalk often turns out to be difficult, since many cooperative causes contribute to it. Loosely speaking, dynamic crosstalk—also called "splicing"—can be associated with uneven voltage contributions to the perceived rms value *on switching from one frame to anothe*r. This view of the problem generally allows for quantifying the impact of improper driving schemes and for putting into action concrete measures to oppose splicing.

Crosstalk Minimization. The problem of reducing crosstalk has been attacked diversely. Apart from technological advances in the display manufacturing, either dedicated and more sophisticated hardware in the driver integrated circuits (such as built-in voltage-correction facilities) or specialization of the addressing schemes have been devised. Beyond the particular features of the huge amount of available alternatives, it is, however, possible to outline some basic design guidelines that help to identify the very essential concepts underneath crosstalk suppression.

Uniformity of the rms contributions over time, for instance, can be pursued through smart selection of the MLA driving mode (e.g., virtual-3-MLA) or the adoption of specific schemes such as the so-called self-calibrating driving method (SCDM), described in the literature. Both such approaches actually eliminate static crosstalk and significantly reduce, although do not entirely suppress, dynamic crosstalk. In particular, virtual-3-MLA also makes the overall optical performance insensitive to asymmetries or inaccuracies in the column voltage levels and contemporarily allows for reducing the number of such levels, which is unquestionably beneficial with respect to static artifacts. Similar results can be attained by using rectangular instead of square phase mixing tables and by enabling drive pattern switching. However, joint application of those methods should be supported by extensive back-end performance evaluation activities to diagnose potential side effects that may stem from their reciprocal interaction at particular image patterns. High-accuracy simulations usually serve this goal.

The above-mentioned polarity inversion modes commonly are also employed as good measures to alleviate static crosstalk artifacts along with all other shortcomings of the nonlinear response of LC cells. However, an important point must be made in selecting the most appropriate inversion strategies. For instance, although separate row or column inversion effectively reduces the impact of horizontal and vertical crosstalk, respectively, vertical or horizontal crosstalk is likely to manifest if either is used independently of each other, with limited advantages in terms of power consumption. Simultaneous suppression of both vertical and horizontal crosstalk is possible with dot inversion in AMLCDs at the cost of extra driving energy. Finally, frame inversion promotes low power operation, but its efficiency in crosstalk reduction is minimal.

The selection of unconventional FRC and PWM schemes, at particular choices of the number of frames in the superframe and of the signal modulation pulses in the scan signals, frequently leads to some controllable reshaping of the frequency spectrum with favorable effects on static crosstalk. It must be noted, however, that all spectral manipulations are only effective when they match concretely the frequency characteristic of the particular liquid: Early assessment of possible drawbacks is mandatory, and customized top-level simulators are valuable in this respect.

Useful suggestions can be also drawn when looking into the technology side of the matter. Interactions between display module parameters and static crosstalk can be tracked down easily: It is well established that static crosstalk is hampered when the resistivity of the ITO tracks is lowered, when a less frequency-dependent liquid is used, or when the deviations in the LC cell capacitances are restrained (the cell capacitance basically depends on the inter-cell gap).

As a final remark, from a theoretical perspective, the phenomenology behind static crosstalk is more easily kept under control when compared with dynamic effects. Experimental verification or system-level simulations are often the sole viable approaches to work into the issues of dynamic artifacts.

Gray-Tone Visualization Artifacts. When GSTs are too simplistic, artifacts altering the visualization of particular color patterns may develop. For instance, with patterns characterized by (although not limited to) the incremental distribution over the pixels of the full gray-tone gamut itself from one side of the display to the other, spurious vertical dim lines may occur. Popular solutions rest essentially on some clever redefinition of the GST, e.g., by the elimination of FRC-only gray tones, the usage of redundancies in the PWM scheme for generating identical gray levels, or the shift of every gray tone one level up with respect to the default GST. A key factor is that the color alteration only affects some particular and well-known gray tones, so that the problem usually can be confined. Because problematic gray tones commonly cause static crosstalk artifacts, their removal yields an added value with respect to crosstalk suppression.

LCD DRIVER DESIGN OVERVIEW

Architectural Concept

In a display module, the driver electronics is responsible for the generation of the proper panel driving waveforms depending on the pixel RGB levels within the image to be displayed. Therefore, optimal driver design is imperative for the realization of high-quality display-based appliances. Display drivers generally are developed as application-specific integrated circuits (ASICs). When considering the hardware architecture, drivers targeting PMLCDs and AMLCDs can be treated jointly in that they share the same building units.

At the architecture level, analog subsystems play a central role in generating the high level voltages required for driving the rows and the columns, and they usually occupy most of the electronics on-chip area. On the other hand, digital blocks do not commonly require any massive area effort: Yet, a deal of vital functions takes place in digital logic. The digital part accommodates all units involved in instruction decoding (the driver is usually fed with commands from an on-board microcontroller) and interface-data handling as well as with display specific functionalities responsible for orthogonal function and scan signal generation, timing, and switching scheduling. Finally, many display drivers also are equipped and shipped with some sort of built-in video memory for local pixel data storage; this facilitates the independence of operation from the host system.

LCD Driver Design Flow

A typical industrial LCD driver design flow includes all those steps needed for mixed analog–digital, very-large-scale-of-integration ASIC design, usually structured throughout the following milestones:

1. Project proposal: analysis of system level requirements based on the application needs.
2. Feasibility study: architecture definition based on system level requirements, preliminary evaluation based on a dedicated LCD module simulator, and project planning.
3. Project start approval.
4. Architecture implementation: block level design of both analog and digital parts (design front-end). Analog circuit design and digital register-transfer-level description and coding are included. Top-level simulation can be also performed at this stage, usually by means of behavioral modeling for the analog parts.
5. Analog and digital place & route and timing verification (design back-end). Cosimulation of functional testbenches with hardware description, including back-annotated timing information, is frequently employed at this stage. Top-level verification and checks follow.
6. Prototype-based on-chip evaluation.

All the steps outlined above describe some very general design phases that are common to the implementations of both active-matrix and passive-matrix addressing display drivers.

Front-End Driving Scheme Selection and Performance Evaluation: Simulation Tools

When regarding the structure of a typical LCD design flow, entailing the availability of testable prototypes at the very end of the design chain, the interest in tools devoted to early stage evaluation of the driver-display module performance should become plain. As a matter of fact, although the hardware development environments comprise mostly standard analog and digital ASIC design tools (those for circuit design, register-transfer-level description and synthesis, technology mapping, cell place & route, and chip verification), specialized tools should be featured to simulate all functional interactions between the front-end driver and the back-end panel. Indeed, merely relying on the chip prototype evaluation phase for performance assessment is often impractical because of the severe costs associated with hardware redesigns on late detection of operational faults. Proprietary simulation tools are likely to be embedded into advanced industrial LCD design flows, and they usually allow for:

1. Functional simulation of the hardware-embedded or software-programmable driving schemes.
2. Functional simulation of all targeted color generation modes.
3. Functional simulations of all embedded measures to prevent visual artifact generation (for testing and evaluation purposes).
4. Highly reconfigurable input modes (such as single-picture display mode, or multiframe driving mode for video streaming evaluation).
5. Sophisticated LCD-modeling engine (including the frequency model of the LC cell response).
6. Reconfigurability with respect to all significant physical display parameters (such as material conductivity, material resistivity, LC cell geometry, and electrode track lengths).
7. Multiple and flexible output formats and performance figures.

The availability of an LCD circuit model is a particularly important aspect, as it opens the possibility of performing reliable evaluation of the module dynamic operation within different frequency ranges.

Display Modeling for LCD Design. Any simulation engine used for driver-display joint modeling cannot function without some form of electrical characterization of an LC cell to work as the electrical load of the driving circuits. A satisfactorily accurate model of the frequency behavior of the LC cell broadly used in practice (9) treats the cell as a simplified RC network whose resistances and capacitances can be calculated as functions of some very basic process and material parameters for the liquid crystal, the polyimide layers, and the connection tracks (e.g., LC conductivity and ITO sheet resistivity). The resulting typical frequency response of the LC cell turns out to be that of a 2-pole, 1-zero network. Consequently, apart from the claim that the driving signals bandwidth be narrow for crosstalk minimization, their lowest frequency bound must be also high enough to prevent distortion induced by the first pole's attenuation.

BIBLIOGRAPHY

1. P. Yeh, and C. Gu, *Optics of Liquid Crystal Displays*, 1st ed., John Wiley & Sons, 1999.
2. E. Lueder, *Liquid Crystal Displays: Addressing Schemes and Elecrto-Optical Effects*, John Wiley & Sons, 2001.
3. T. J. Scheffer, and J. Nehring, Supertwisted nematic LCDs, *SID Int. Sym. Dig. Tech. Papers*, M-12, 2000.
4. T. N. Ruckmongathan, Addressing techniques for RMS responding LCDs - A review, *Proc. 12th Int. Display Res. Conf. Japan Display '92*, 77–80, 1992.
5. M. Kitamura, A. Nakazawa, K. Kawaguchi, H. Motegi, Y. Hirai, T. Kuwata, H. Koh, M. Itoh, and H. Araki, Recent developments in multi-line addressing of STN-LCDs, *SID Int. Sym. Dig. Tech. Papers*, 355–358, 1996.
6. H. Seiberle, and M. Schadt, LC-conductivity and cell parameters; their influence on twisted nematic and supertwisted nematic liquid crystal displays, *Mol. Cryst, Liq. Cryst.*, **239**, 229–244, 1994.
7. K. Tarumi, H. Numata, H. Prücher, and B. Schuler, On the relationship between the material parameters and the switching dynamics on twisted nematic liquid crystals, *Proc. 12th Int. Display Res. Conf. Japan Display '92*, 587–590, 1992.
8. L. MacDonald, and A. C. Lowe, *Display Systems: Design and Applications*, John Wiley & Sons, 1997.
9. H. Seiberle, and M. Schadt, Influence of charge carriers and display parameters on the performance of passively and actively addressed, *SID Int. Sym. Dig. Tech. Papers*, 25–28, 1992.

FURTHER READING

J. A. Castellano, *Liquid Gold: The Story Of Liquid Crystal Displays and the Creation of an Industry*, World Scientific Publishing Company, 2005.

P. A. Keller, *Electronic Display Measurement: Concepts, Techniques, and Instrumentation*, 1st ed., Wiley-Interscience, 1997.

M. A. Karim, *Electro-Optical Displays*, CRC, 1992.

P. M. Alt, and P. Pleshko, Scanning limitations of liquid crystal displays, *IEEE Trans. El. Dev.*, **ED-21**(2), 146–155, 1974.

K. E. Kuijk, Minimum-voltage driving of STN LCDs by optimized multiple-row addressing, *J. Soc. Inf. Display*, **8**(2), 147–153, 2000.

M. Watanabe, High resolution, large diagonal color STN for desktop monitor application, *SID Int. Sym. Dig. Tech. Papers*, **34**, M-81–87, 1997.

S. Nishitani, H. Mano, and Y. Kudo, New drive method to eliminate crosstalk in STN-LCDs, *SID Int. Sym. Dig. Tech. Papers*, 97–100, 1993.

Simone Smorfa
Mauro Olivieri
"La Sapienza," University of Rome
Rome, Italy
Roberto Mancuso
Philips Semiconductors
Zurich, Switzerland

LOGIC DESIGN

INTRODUCTION

Over the years, digital electronic systems have progressed from vacuum tube to complex integrated circuits, some of which contain millions of transistors. Electronic circuits can be separated into two groups, digital and analog circuits. Analog circuits operate on analog quantities that are continuous in value and in time, whereas digital circuits operate on digital quantities that are discrete in value and time (1).

Analog signals are continuous in time besides being continuous in value. Most measurable quantities in nature are in analog form, for example, temperature. Measuring around the hour temperature changes is continuous in value and time, where the temperature can take any value at any instance of time with no limit on precision but on the capability of the measurement tool. Fixing the measurement of temperature to one reading per an interval of time and rounding the value recorded to the nearest integer will graph discrete values at discrete intervals of time that easily could be coded into digital quantities. From the given example, it is clear that an analog-by-nature quantity could be converted to digital by taking discrete-valued samples at discrete intervals of time and then coding each sample. The process of conversion is usually known as analog-to-digital conversion (A/D). The opposite scenario of conversion is also valid and known as digital-to-analog conversion (D/A). The representation of information in a digital form has many advantages over analog representation in electronic systems. Digital data that are discrete in value, discrete in time, and limited in precision could be efficiently stored, processed, and transmitted. Digital systems are said practically to be more noise immune as compared with analog electronic systems because of the physical nature of analog signals. Accordingly, digital systems are more reliable than their analog counterpart. Examples of analog and digital systems are shown in Fig. 1.

A BRIDGE BETWEEN LOGIC AND CIRCUITS

Digital electronic systems represent information in digits. The digits used in digital systems are the *0* and *1* that belong to the *binary* mathematical number system. In logic, the *1* and *0* values correspond to *True* and *False*. In circuits, the *True* and *False* could be thought of as *High* voltage and *Low* voltage. These correspondences set the relationships among logic (*True* and *False*), *binary* mathematics (*0* and 1), and circuits (*High* and *Low*).

Logic, in its basic shape, deals with reasoning that checks the validity of a certain proposition—a proposition could be either *True* or *False*. The relationship among logic, *binary* mathematics, and circuits enables a smooth transition of processes expressed in propositional logic to *binary* mathematical functions and equations (*Boolean* algebra) and to digital circuits. A great scientific wealth exists that strongly supports the relationships among the three different branches of science that lead to the foundation of modern digital hardware and logic design.

Boolean algebra uses three basic logic operations *AND, OR*, and *NOT*. The *NOT* operation if joined with a proposition *P* works by negating it; for instance, if *P* is *True*, then *NOT P* is *False* and vice versa. The operations *AND* and *OR* should be used with two propositions, for example, *P* and *Q*. The logic operation *AND*, if applied on *P* and *Q*, would mean that *P AND Q* is *True* only when both *P* and *Q* are *True*. Similarly, the logic operation *OR*, if applied on *P* and *Q*, would mean that *P OR Q* is *False* only when *P* and *Q* are *False*. Truth tables of the logic operators *AND, OR*, and *NOT are* shown in Fig. 2(a). Fig. 2(b) shows an alternative representation of the truth tables of *AND, OR*, and *NOT in* terms of *0s* and *1s*.

COMBINATIONAL LOGIC CIRCUITS

Digital circuits implement the logic operations *AND, OR*, and *NOT* as hardware elements called "gates" that perform logic operations on binary inputs. The *AND-gate* performs an *AND* operation, an *OR-gate* performs an *OR* operation, and an *Inverter* performs the negation operation *NOT*. Figure 2(c) shows the standard logic symbols for the three basic operations. With analogy from electric circuits, the functionality of the *AND* and *OR* gates are captured as shown in Fig. 3. The actual internal circuitry of gates is built using transistors; two different circuit implementations of inverters are shown in Fig. 4. Examples of *AND, OR, NOT* gates integrated circuits (*ICs*) are shown in Fig. 5. Besides the three essential logic operations, four other important operations exist—the *NOR* (*NOT-OR*), *NAND* (*NOT-AND*), Exclusive-*OR* (*XOR*), and Exclusive-*NOR* (*XNOR*).

A combinational logic circuit is usually created by combining gates together to implement a certain logic function. A combinational circuit produces its result upon application of its input(s). A logic function could be a combination of logic variables, such as *A, B, C*, and so on. Logic variables can take only the values *0* or *1*. The created circuit could be implemented using *AND-OR-Inverter* gate-structure or using other types of gates. Figure 6(a) shows an example combinational implementation of the following logic function *F*(*A, B, C*):

$$F(A,B,C) = ABC + A'BC + AB'C'$$

F(*A, B, C*) in this case could be described as a standard sum-of-products (*SOP*) function according to the analogy that exists between *OR* and addition (+), and between *AND* and product (.); the *NOT* operation is indicated by an apostrophe "'" following the variable name. Usually, standard representations are also referred to as canonical representations.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

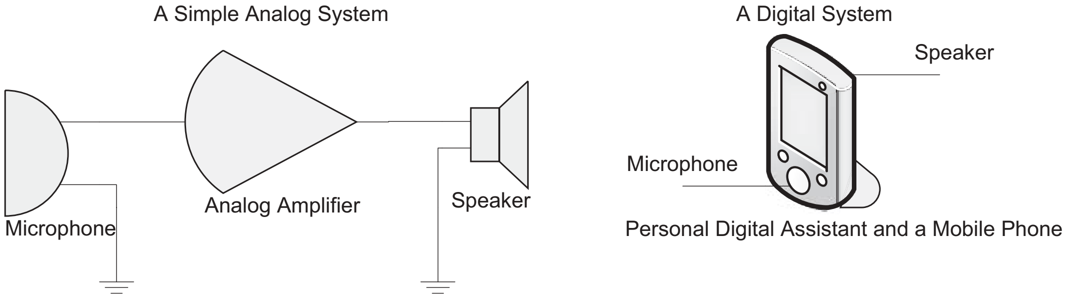


Figure 1. A simple analog system and a digital system; the analog signal amplifies the input signal using analog electronic components. The digital system can still include analog components like a speaker and a microphone; the internal processing is digital.

Input P	Input *Q*	Output: *P AND Q*
False	False	False
False	True	False
True	False	False
True	True	True

Input P	Input *Q*	Output: *P OR Q*
False	False	False
False	True	True
True	False	True
True	True	True

Input *X*	Output: *NOT P*
False	True
True	False

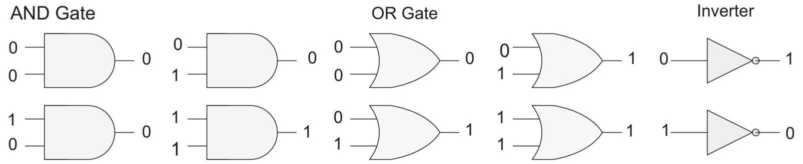
(a)

Input P	Input *Q*	Output: *P AND Q*
0	0	0
0	1	0
1	0	0
1	1	1

Input P	Input *Q*	Output: *P OR Q*
0	0	0
0	1	1
1	0	1
1	1	1

Input *X*	Output: *NOT P*
0	1
1	0

(b)



(c)

Figure 2. (a) Truth tables for AND, OR, and Inverter. (b) Truth tables for AND, OR, and Inverter in binary numbers. (c) Symbols for AND, OR, and Inverter with their operation.

In an alternative formulation, consider the following function *E*(*A,B,C*) in a product-of-sums (*POS*) form:

$$E(A,B,C) = (A + B' + C).(A' + B + C)(A + B + C')$$

The canonical *POS* implementation is shown in Fig. 6(b). Some other specifications might require functions with more inputs and accordingly with a more complicated design process.

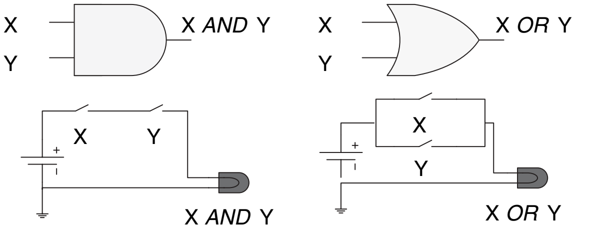


Figure 3. A suggested analogy between *AND*and *OR* gates and electric circuits.

The complexity of a digital logic circuit that corresponds to *a Boolean* function is directly related to the complexity of the base algebraic function. *Boolean* functions may be simplified by several means. The simplification process that produces an expression with the least number of terms with the least number of variables is usually called minimization. The minimization has direct effects on reducing the cost of the implemented circuit and sometimes on enhancing its performance. The minimization (optimization) techniques range from simple (manual) to complex (automated). An example of manual optimization methods is the *Karnough* map (*K*-map).

K-MAPS

A *K*-map is similar to a truth table as it presents all the possible values of input variables and their corresponding output. The main difference between *K*-maps and truth tables is in the cells arrangement. In a *K*-map, cells are arranged in a way so that simplification of a given algebraic expression is simply a matter of properly grouping the cells.

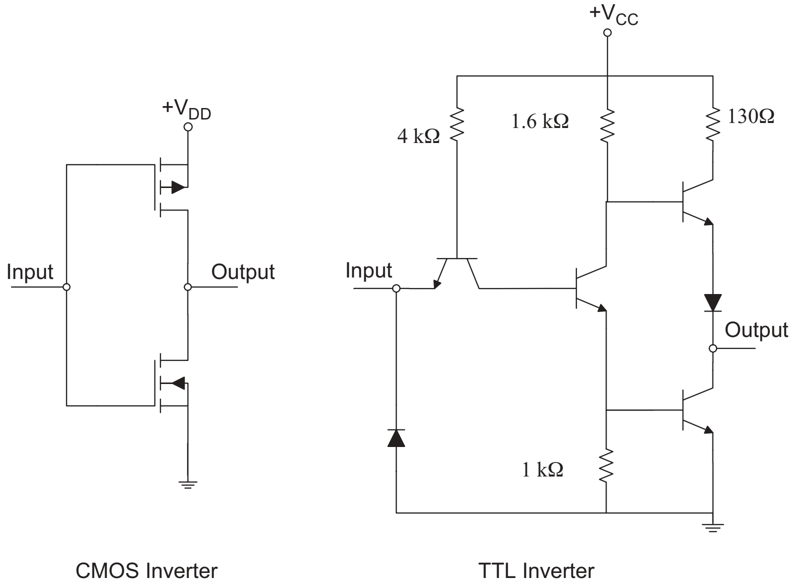


Figure 4. Complementary metal-oxide semiconductor (*CMOS*) and transistor-transistor logic (*TTL*) inverters.

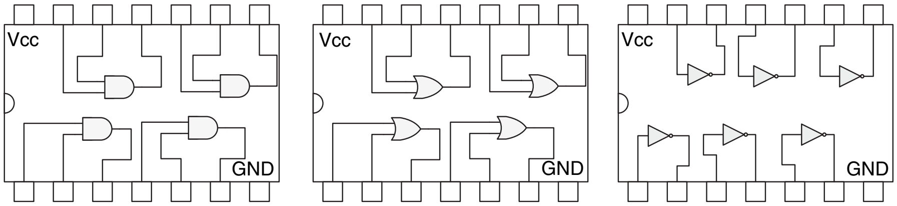


Figure 5. The 74LS21 (*AND*), 74LS32 (*OR*), and 74LS04 (*Inverter) TTL* ICs.

K-maps can be used for expressions with different numbers of input variables: three, four, or five. In the following examples, maps with only three and four variables are shown to stress the principle. Methods for optimizing expressions with more than five variables can be found in the literature. The *Quine–McClusky* method is an example that can accommodate several variables larger than five (2).

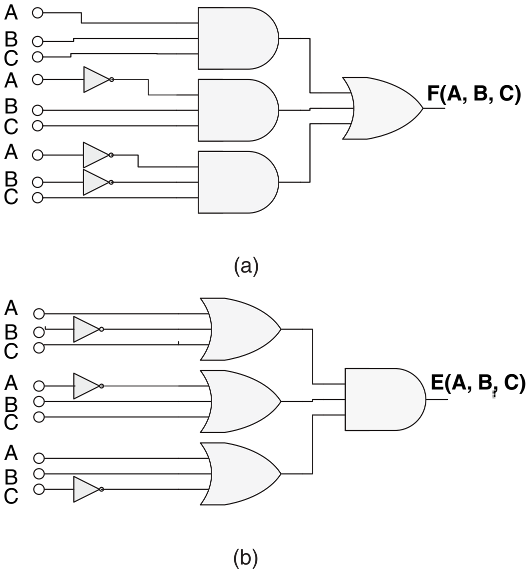


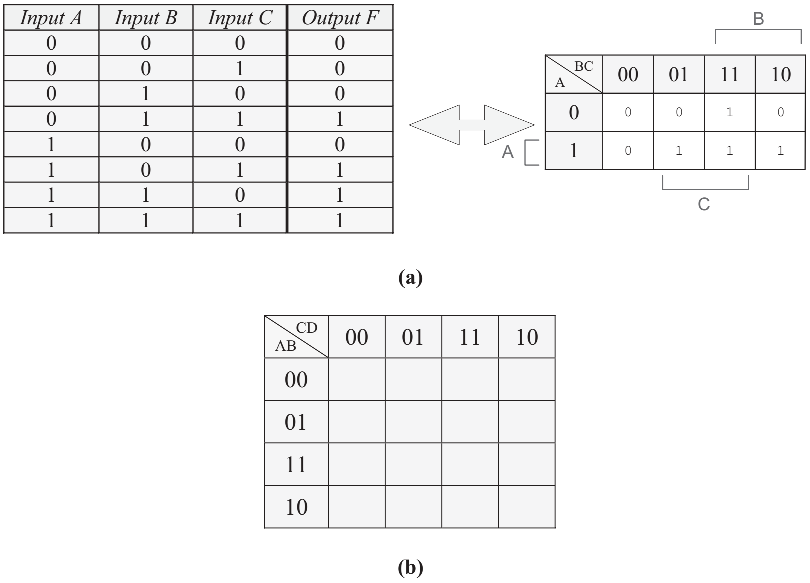
Figure 6. AND–OR–Inverter implementation of the function (a) SOP: F(A, B, C) = ABC + A′BC + AB′C′. (b) POS: E(A, B, C) = (A + B′ + C).(A′ + B + C)(A + B + C′).

A three-variable *K*-map is an array of *8* (or *2*^{3}) cells. Figure 7(a) depicts the correspondence between a three-input (*A*, *B*, and *C*) truth table and a *K*-map. The value of a given cell represents the output at certain binary values of *A, B*, and *C*. In a similar way, a four-variable *K*-map is arranged as shown in Fig. 7(b). *K*-maps could be used for expressions in either *POS* or *SOP* forms. Cells in a *K*-map are arranged so that they satisfy the *Adjacency* property, where only a single variable changes its value between adjacent cells. For instance, the cell *000*, that is the binary value of the term *A′B′C′*, is adjacent to cell *001* that corresponds to the term *A′B′C*. The cell *0011* (*A′B′CD*) is adjacent to the cell *0010* (*A′B′C′D*).

MINIMIZING *SOP* EXPRESSIONS

The minimization of an algebraic *Boolean* function *f* has the following four key steps:

1. Evaluation
2. Placement
3. Grouping
4. Derivation



Input A	*Input B*	*Input C*	*Output F*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

(a)

AB \ CD	00	01	11	10
00				
01				
11				
10				

(b)

Figure 7. (a) The correspondence between a three-input (*A, B*, and *C*) truth table and a *K-map*. (b) An empty four-variable *K*-map.

The minimization starts by evaluating each term in the function *f* and then by placing a *1* in the corresponding cell on the *K*-map. A term *ABC* in a function *f* (A, *B, C*) is evaluated to *111*, and another term *AB 'CD* in a function *g(A, B, C, D)* is evaluated to *1011*. An example of evaluating and placing the following function *f* is shown in Fig. 8(a):

$$f(A,B,C) = A'B'C' + A'B'C + ABC' + AB'C'$$

After placing the *1s* on a *K*-map, grouping filled-with-*1s* cells is done according to the following rules [see Fig. 8(b)]:

- A group of adjacent filled-with-*1s* cells must contain several cells that belong to the set of powers of two (1, 2, 4, 8, or 16).

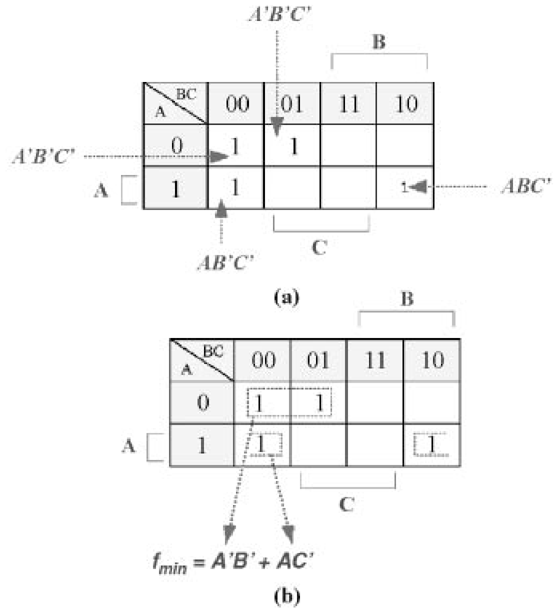


Figure 8. (a) Terms evaluation of the function *f*(*A, B, C*) = *A'B'C'* + *A'B'C* + *ABC'* + *AB'C'*.(b) Grouping and derivation.

- A group should include the largest possible number of filled-with-*1s* cells.
- Each *1* on the *K*-map must be included in at least one group.
- *Cells* contained in a group could be shared within another group as long as overlapping groups included noncommon *1s*.

After the grouping step, the derivation of minimized terms is done according to the following rules:

- Each group containing *1s* creates one product term.
- The created product term includes all variables that appear in only one form (completed or uncomplemented) across all cells in a group.

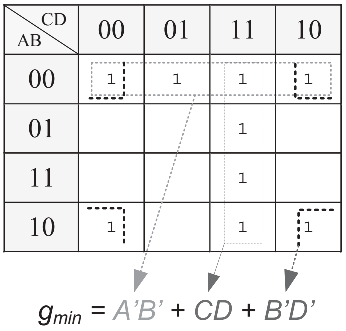
After deriving terms, the minimized function is composed of their sum. An example derivation is shown in Fig. 8(b). Figure 9 presents the minimization of the following function:

$$\begin{aligned} g(A,B,C,D) = {} & AB'C'D' + A'B'C'D' + A'B'C'D' + A'B'CD \\ & + AB'CD + A'B'CD' + A'BCD + ABCD \\ & + AB'CD' \end{aligned}$$

COMBINATIONAL LOGIC DESIGN

The basic combinational logic design steps could be summarized as follows:

1. Specification of the required circuit.
2. Formulation of the specification to derive algebraic equations.
3. Optimization (minimization) of the obtained equations.
4. Implementation of the optimized equations using a suitable hardware (*IC*) technology.



AB \ CD	00	01	11	10
00	1	1	1	1
01			1	
11			1	
10	1		1	1

$g_{min} = A'B' + CD + B'D'$

Figure 9. Minimization steps of the following function: $g(A, B, C, D) = AB'C'D' + A'B'C'D' + A'B'C'D' + A'B'CD + AB'CD + A'B'B'CD' + A'BCD + ABCD + AB'CD'$.

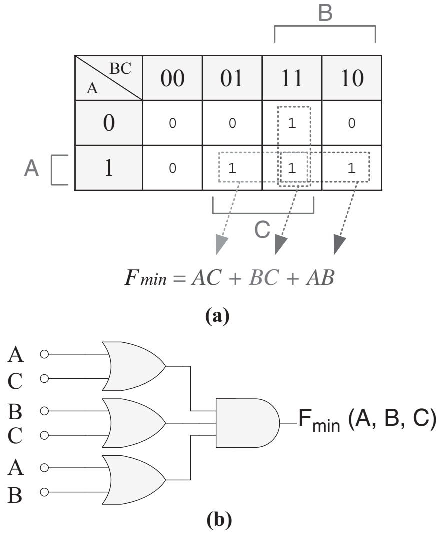
The above steps are usually joined with an essential verification procedure that ensures the correctness and completeness of each design step.

As an example, consider the design and implementation of a three-variable majority function. The function $F(A, B, C)$ will return a *1* (*High* or *True*) whenever the number of *1s* in the inputs is greater than or equal to the number of *0s*.

The above specification could be reduced into a truth table as shown in Fig. 7(a). The terms that make the function F return a *1* are the terms $F(0, 1, 1)$, $F(1, 0, 1)$, $F(1, 1, 0)$, or $F(1, 1, 1)$. This truth table could be alternatively formulated as in the following equation:

$$F = A'BC + AB'C + ABC' + ABC$$

Following the specification and the formulation, a *K*-map is used to obtain the minimized version of F (called $F_{\min}$). Figure 10(a) depicts the minimization process. Figure 10(b) shows the implementation of $F_{\min}$ using standard *AND-OR-NOT* gates.



A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$F_{min} = AC + BC + AB$

(a)

(b)

Figure 10. (a) Minimization of a three-variable majority function. (b) Implementation of a minimized three-variable majority function.

COMBINATIONAL LOGIC CIRCUITS

Famous combinational circuits that are widely adopted in digital systems include encoders, decoders, multiplexers, adders, some programmable logic devices (*PLDs*), and so on. The basic operation of multiplexers, half-adders, and simple *PLDs* (*SPLDs*) is described in the following lines.

A multiplexer (*MUX*) selects one of n input lines and provides it on a single output. The select lines, denoted S, identify or address one of the several inputs. Figure 11(a) shows the block diagram of a 2-to-1 multiplexer. The two inputs can be selected by one select line, S. If the selector $S = 0$, input line d_0 will be the output O, otherwise, d_1 will be produced at the output. An *MUX* implementation of the majority function $F(A, B, C)$ is shown in Fig. 11(b).

A half-adder inputs two binary digits to be added and produces two binary digits representing the sum and carry. The equations, implementation, and symbol of a half-adder are shown in Fig. 12.

Simple *PLDs* (*SPLDs*) are usually built from combinational logic blocks with prerouted wiring. In implementing a function on a *PLD*, the designer will only decide of which wires and blocks to use; this step is usually referred to as programming the device. Programmable logic array (*PLA*) and the programmable array logic (*PAL*) are commonly used *SPLDs*. A *PLA* has a set of programmable *AND* gates, which link to a set of programmable *OR* gates to produce an output [see Fig. 13(a)]. A *PAL* has a set of programmable *AND* gates, which link to a set of fixed *OR* gates to produce an output [see Fig. 13(b)]. The *AND-OR* layout of a *PLA*/*PAL* allows for implementing logic functions that are in an *SOP* form. A *PLA* implementation of the majority function $f(A, B, C)$ is shown in Fig. 13(c).

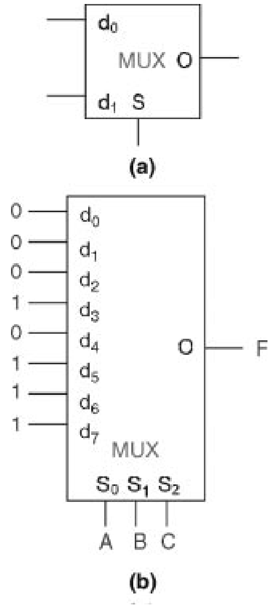


Figure 11. (a) Minimization of a three-variable majority function. (b) Implementation of a minimized three-variable majority function.

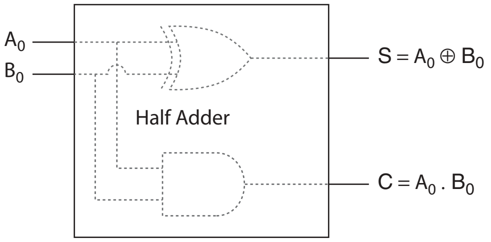


Figure 12. The equations, implementation, and symbol of a half-adder. The used symbol for a *XOR* operation is "⊕".

SEQUENTIAL LOGIC

In practice, most digital systems contain combinational circuits along with memory; these systems are known as sequential circuits. In sequential circuits, the present outputs depend on the present inputs and the previous states stored in the memory elements. Sequential circuits are of two types: synchronous and asynchronous. In a synchronous sequential circuit, a clock signal is used at discrete instants of time to synchronize desired operations. A clock is a device that generates a train of pulses as shown in Fig. 14. Asynchronous sequential circuits do not require synchronizing clock pulses; however, the completion of an operation signals the start of the next operation in sequence.

In synchronous sequential circuits, the memory elements are called flip-flops and are capable of storing only one bit. Arrays of flip-flops are usually used to accommodate for bit-width requirements of binary data. A typical synchronous sequential circuit contains a combinational part, sequential elements, and feedback signals coming from the output of the sequential elements.

FLIP-FLOPS

Flip-flops are volatile elements, where the stored bit is stored as long as power is available. Flip-flops are designed using basic storage circuits called latches. The most common latch is the *SR* (Set to 1 - Reset to 0) latch. An *SR* latch

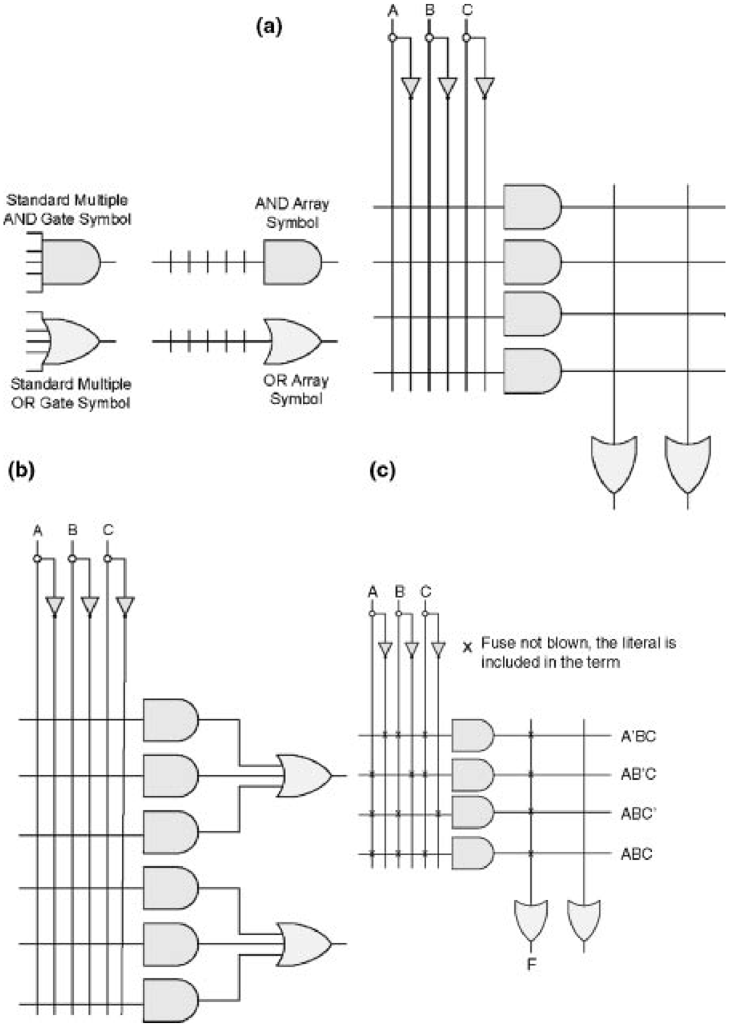

(a)
A
B
C
Standard Multiple AND Gate Symbol
AND Array Symbol
Standard Multiple OR Gate Symbol
OR Array Symbol
(b)
(c)
A
B
C
A
B
C
x Fuse not blown, the literal is included in the term
A'BC
AB'C
ABC'
ABC
F

Figure 13. (a) A three-input, two-output *PLA* with its *AND* arrays and *OR* arrays. An *AND* array is equivalent to a standard multiple-input *AND* gate, and an *OR* array is equivalent to a standard multiple-input *OR* gate. (b) A three-input, two-output *PAL*. (c) A *PLA* implementation of the majority function *F*(*A, B, C*).

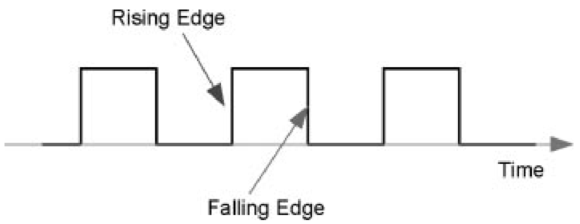


Figure 14. Clock pulses.

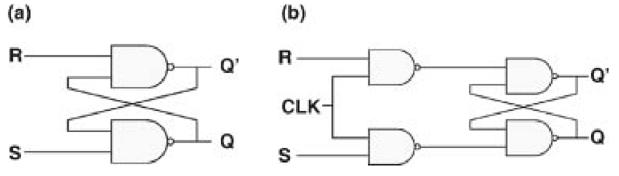
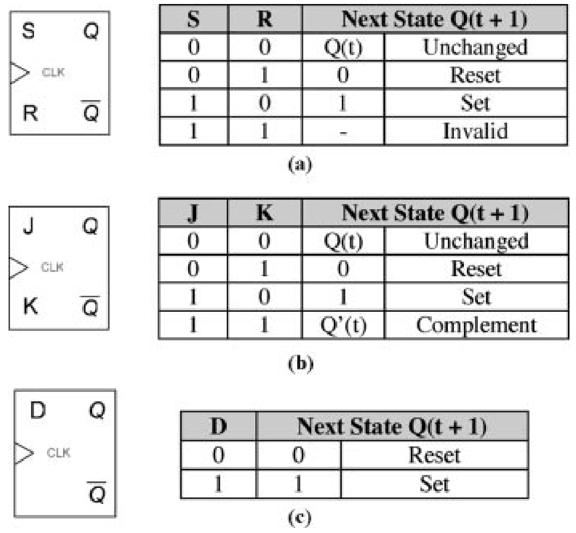


Figure 15. (a) An *SR* latch. (b) An *RS* flip-flop.

could be formed with two cross-coupled *NAND* gates as shown in Fig. 15. The responses to various inputs to the *SR* latch are setting *Q* to *1* for an *SR* input of *01* (*S* is active low; i.e., *S* is active when it is equal to *0*), resetting *Q* to *0* for an *SR* input of *10* (*R* here is also active low), and memorizing the current state for an *SR* input of *11*. The *SR* input of *00* is considered invalid.

A flip-flop is a latch with a clock input. A flip-flop that changes state either at the positive (rising) edge or at the negative (falling) edge of the clock is called an edge-triggered flip-flop (see Fig. 14). The three famous edge-triggered flip-flops are the *RS, JK*, and *D* flip-flops.

An *RS* flip-flop is a clocked *SR* latch with two more *NAND* gates [see Fig. 15(b)]. The symbol and the basic operation of an *RS* flip-flop are illustrated in Fig. 16(a). The operation of an *RS* flip-flop is different from that of an *SR* latch and responds differently to different values of *S* and *R*. The *JK* and *D* flip-flops are derived from the *SR* flip-flop.


S
Q
CLK
R
Q̄
J
Q
CLK
K
Q̄
D
Q
CLK
Q̄

S	R	Next State Q(t + 1)	
0	0	Q(t)	Unchanged
0	1	0	Reset
1	0	1	Set
1	1	-	Invalid

(a)

J	K	Next State Q(t + 1)	
0	0	Q(t)	Unchanged
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

(b)

D	Next State Q(t + 1)	
0	0	Reset
1	1	Set

(c)

Figure 16. (a) The symbol and the basic operation of (a) *RS* flip-flop, (b) JK flip-flop, and (c) *D* flip-flop.

However, the *JK* and *D* flip-flops are more widely used (2). The *JK* flip-flop is identical to the *SR* flip-flop with a single difference, where it has no invalid state [see Fig. 16(b)]. The *D* flip-flop has only one input formed with an *SR* flip-flop and an inverter [see Fig. 16(c)]; thus, it only could set or reset. The *D* flip-flop is also known as a transparent flip-flop, where output will have the same value of the input after one clock cycle.

SEQUENTIAL LOGIC DESIGN

The basic sequential logic design steps are generally identical to those for combinational circuits; these are Specification, Formulation, Optimization, and the Implementation of the optimized equations using a suitable hardware (*IC*) technology. The differences between sequential and combinational design steps appear in the details of each step.

The specification step in sequential logic design usually describes the different states through which the sequential circuit goes. A typical example for a sequential circuit is a counter that undergoes eight different states, for instance, zero, one, two, three up to seven. A classic way to describe the state transitions of sequential circuits is a state diagram. In a state diagram, a circle represents a state and an arrow represents a transition. The proposed example assumes no inputs to control the transitions among states. Figure 17(a) shows the state diagram of the specified counter. The number of states determines the minimum number of flip-flops to be used in the circuit. In the case of the *8*-states counter, the number of flip-flops should be *3*; in accordance with the formula, *8* equals 2^3. At this stage, the states could be coded in binary. For instance, the stage representing count *0* is coded to binary *000*; the stage of count *1* is coded to binary *001*, and so on.

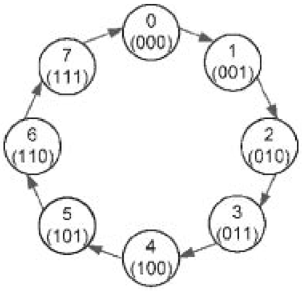
The state diagram is next to be described in a truth table style, usually known as a state table, from which the formulation step could be carried forward. For each flip-flop, an input equation is derived [see Fig. 17(b)]. The equations are then minimized using *K*-maps [see Fig. 17(c)]. The minimized input equations are then implemented using a suitable hardware (*IC*) technology. The minimized equations are then to be implemented [see Fig. 17(d)].

MODERN LOGIC DESIGN

The task of manually designing hardware tends to be extremely tedious, and sometimes impossible, with the increasing complexity of modern digital circuits. Fortunately, the demand on large digital systems has been accompanied with a fast advancement in *IC* technologies. Indeed, *IC* technology has been growing faster than the ability of designers to produce hardware designs. Hence, there has been a growing interest in developing techniques and tools that facilitate the process of logic design.

Two different approaches emerged from the debate over ways to automate hardware logic design. On one hand, the capture-and-simulate proponents believe that human designers have good design experience that cannot be automated. They also believe that a designer can build

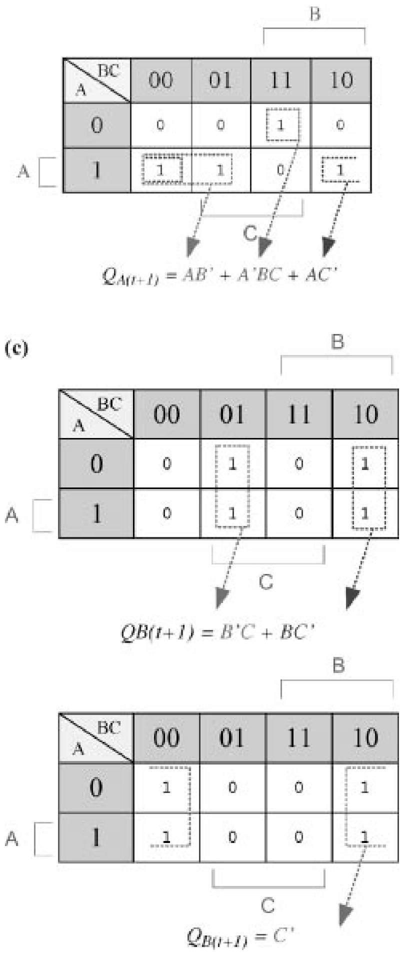
(a)



(a)

Present State			Next State		
$Q_{A(t)}$	$Q_{B(t)}$	$Q_{C(t)}$	$Q_{A(t+1)}$	$Q_{B(t+1)}$	$Q_{C(t+1)}$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

(b)



A \ BC	00	01	11	10
0	0	0	1	0
1	1	1	0	1

$Q_{A(t+1)} = AB' + A'BC + AC'$

(c)

A \ BC	00	01	11	10
0	0	1	0	1
1	0	1	0	1

$QB(t+1) = B'C + BC'$

A \ BC	00	01	11	10
0	1	0	0	1
1	1	0	0	1

$Q_{B(t+1)} = C'$

Figure 17. (a) The state diagram of the specified counter. (b) The state table. (c) Minimization of input equations. (d) Impliementation of the counter.

(d)

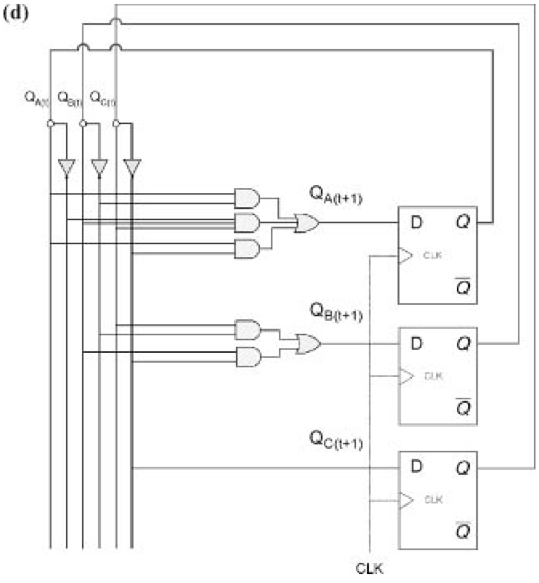


Figure 17. (*Continued*)

a design in a bottom-up style from elementary components such as transistors and gates. As the designer is concerned with the deepest details of the design, optimized and cheap designs could be produced. On the other hand, the describe-and-synthesize advocates believe that synthesizing algorithms can out-perform human designers. They also believe that a top-down fashion would be better suited for designing complex systems. In describe-and-synthesize methodology, the designers first describe the design. Then, computer-aided design (*CAD*) tools can generate the physical and electrical structure. This approach describes the intended designs using special languages called hardware description languages (*HDLs*). Some *HDLs* are very similar to traditional programming languages like *C*, *Pascal*, and so on. (3). *Verilog*(4) and *VHDL* (Very High Speed Integrated Circuit Hardware Description Language) (5) are by far the most commonly used *HDLs* in industry.

Hardware synthesis is a general term used to refer to the processes involved in automatically generating a hardware design from its specification. High-level synthesis (*HLS*) could be defined as the translation from a behavioral description of the intended hardware circuit into a structural description. The behavioral description represents an algorithm, equation, and so on, whereas a structural description represents the hardware components that implement the behavioral description.

The chained synthesis tasks at each level of the design process include system synthesis, register-transfer synthesis, logic synthesis, and circuit synthesis. System synthesis starts with a set of processes communicating though either shared variables or message passing. Each component can be described using a register-transfer language (*RTL*). *RTL* descriptions model a hardware design as circuit blocks and interconnecting wires. Each of these circuit blocks could be described using *Boolean* expressions. Logic synthesis translates *Boolean* expressions into a list of logic gates and their interconnections (netlist). Based on the produced netlist, circuit synthesis generates a transistor schematic from a set of input–output current, voltage and frequency characteristics, or equations.

The logic synthesis step automatically converts a logic-level behavior, consisting of logic equations and/or finite state machines (*FSMs*), into a structural implementation (3). Finding an optimal solution for complex logic minimization problems is very hard. As a consequence, most logic synthesis tools use heuristics. A heuristic is a technique whose result can hopefully come close to the optimal solution. The impact of complexity and of the use of heuristics on logic synthesis is significant. Logic synthesis tools differ tremendously according to the heuristics they use. Some computationally intensive heuristics require long run times and thus powerful workstations producing high-quality solutions. However, other logic synthesis tools use fast heuristics that are typically found on personal computers producing solutions with less quality. Tools with expensive heuristics usually allow a user to control the level of optimization to be applied.

Continuous efforts have been made, paving the way for modern logic design. These efforts included the development of many new techniques and tools. An approach to logic minimization using a new sum operation called multiple valued *EXOR* is proposed in Ref. 6 based on neural computing.

In Ref. 7, Tomita et al. discuss the problem of locating logic design errors and propose an algorithm to solve it. Based on the results of logic verification, the authors introduce an input pattern for locating design errors. An algorithm for locating single design errors with the input patterns has been developed.

Efforts for creating tools with higher levels of abstraction in design lead to the production of many powerful modern hardware design tools. Ian Page and Wayne Luk(8) developed a compiler that transformed a subset of *Occam* into a netlist. Nearly ten years later we have seen the development of *Handel-C*, the first commercially available high-level language for targeting programmable logic devices (such as field programmable gate arrays (9).

A prototype *HDL* called Lava was developed by Satnam Singh at *Xilinx* and by Mary Sheeran and Koen Claessen at Chalmers University in Sweden (10). Lava allows circuit tiles to be composed using powerful high-order combinators. This language is embedded in the *Haskell* lazy functional programming language. *Xilinx* implementation of *Lava* is designed to support the rapid representation, implementation, and analysis of high-performance *FPGA* circuits.

Logic design has an essential impact on the development of modern digital systems. In addition, logic design techniques are a primary key in various modern areas, such as embedded systems design, reconfigurable systems (11), hardware/software co-design, and so on.

CROSS-REFERENCES

Programmable Logic Devices, see Programmable Logic Arrays
Synthesis, see High-Level Synthesis
Synthesis, see Logic Synthesis

BIBLIOGRAPHY

1. F. Vahid et al., *Embedded System Design: A Unified Hardware / Software Introduction*. New York: Wiley, 2002.
2. T. Floyd, *Digital Fundamentals with PLD Programming*. Englewood Cliffs, NJ: Prentice Hall, 2006.
3. S. Hachtel, *Logic Synthesis and Verification Algorithms*. Norwell: Kluwer, 1996.
4. IEEE Standard 1364, *Verilog HDL language reference manual*, 1995.
5. IEEE Standard 1076, *Standard VHDL reference manual*, 1993.
6. A. Hozumi, N. Kamiura, Y. Hata, and K. Yamato, Multiple-valued logic design using multiple-valued EXOR, *Proc. Multiple-Valued Logic*, 1995, pp. 290–294.
7. M. Tomita, H. Jiang, T. Yamamoto, and Y. Hayashi, An algorithm for locating logic design errors, *Proc. Computer-Aided Design*, 1990, pp. 468–471.
8. I. Page and W. Luk, Compiling Occam into field-programmable gate arrays, *Proc. Workshop on Field Programmable Logic and Applications*, 1991, pp. 271–283.
9. S. Brown and J. Rose, Architecture of FPGAs and CPLDs: A tutorial, *IEEE Design Test Comput.*, **2**: 42–57, 1996.
10. K. Claessen, Embedded languages for describing and verifying hardware, PhD Thesis, Chalmers University of Technology and Göteborg University, 2001.
11. E. Mirsky and A. DeHon, MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources, *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, 1996, pp. 157–166.

FURTHER READING

T. Floyd, *Digital Fundamentals with PLD Programming*. Englewood Cliffs, NJ: Prentice Hall, 2006.

M. Mano et al., *Logic and Computer Design Fundamentals*. Englewood Cliffs, NJ: Prentice Hall, 2004.

ISSAM W. DAMAJ
Dhofar University
Salalah, Oman

LOGIC SYNTHESIS

The design process for an electronics system begins when an idea is transformed into a set of specifications to be verified by the future system. These specifications become the basis for a series of steps or design tasks that eventually will produce a circuit that represents the physical expression of the original idea. The process of generating a final circuit from the initial specifications is known as circuit synthesis.

The design flow for a digital system is composed of a series of stages in which system models are established in accordance with different criteria. Each stage corresponds to a level of abstraction.

To illustrate how these levels of abstraction may be classified, we might, for example, consider three levels: the system level, the RT (register transfer) level, and the logic level. In the system level, the architecture and algorithms necessary to verify the required performance are specified. The RT level represents the system specification as an RT model, in this case establishing an architecture for data flow between registers subject to functional transformations. Finally, the logic level determines the system's functionality using logic equations and descriptions of finite state machines (FSMs). The data handled is logic data with values such as 0, 1, X, Z, etc.

Design tasks in each of the levels usually are supported by different computer aided design (CAD) tools. In each level, the design process basically involves two stages: (*1*) description of the system at the corresponding level and (*2*) verification of the description's behavior via simulation. The synthesis process consists of obtaining the system structure from a description of the behavior. Depending on the level of abstraction in which the work is being carried out, the synthesis will be high level synthesis, logic synthesis, etc. This article addresses logic synthesis, which involves the generation of a circuit at the logic level based on an RT level design specification. The automation of the synthesis process has allowed the development of several tools that facilitate the tasks involved. Automatic synthesis tools offer several advantages when implementing an electronic circuit. First, automation allows the design flow to be completed in less time, which is relevant particularly today because of the high competitiveness and the requirements to solve demands in a short period of time. Second, automatic synthesis also makes the exploration of design space more viable because it enables different requirements, such as cost, speed, and power, to be analyzed. Third, a fundamental aspect of the whole design process is its robustness, that is, its certainty that the product is free from any errors attributable to the designer. In this regard, the use of automatic synthesis tools guarantees the "correct construction" of the system being designed.

The following section of this article deals with aspects associated with logic design such as data types, system components, and modes of operation. Next, the hardware description languages will be presented as tools to specify digital systems. Two standard languages (VHDL and Verilog) will be examined in detail, and the use of VHDL for synthesis will be explained to illustrate specific aspects of logic synthesis descriptions. The article ends with an illustrative example of the principal concepts discussed.

LOGIC DESIGN ORGANIZATION: DATAPATH AND CONTROL UNIT

The RT level is the level of abstraction immediately above the logic level (1,2). In contrast with the logic level, generally concerned with bitstreams, the RT level handles "data." Data is a binary word of *n* bits. Data are processed through arithmetic or logic operations that normally affect one or two data: A + B, NOT(A), and so on.

Data are stored in "registers," which constitute the electronic component for storing *n* bits. Source data must be "read" from its registers and the result of the operation is then "written" in another register to be stored there. The data operation is performed in a "functional unit" (for example, an arithmetic-logic unit). The writing operation is sequential and, in a synchronous system, for example, is therefore executed while the clock is active. The operations of the functional unit and the reading of the register are combinational functions.

Data is transmitted from the source registers toward the functional unit and from there to the target register via "buses," which are basically "*n*" cables with an associated protocol to allow their use.

The core operation is data transfer between registers—hence the name given to this level. It includes both reading and writing operations. Description techniques suitable at the logic level (FSMs and switching theory) are not sufficient at the RT level. One of the simplest ways to describe these operations is as follows:

$$\text{writing}: \quad R_{\text{target}} < \; = DataA * DataB$$

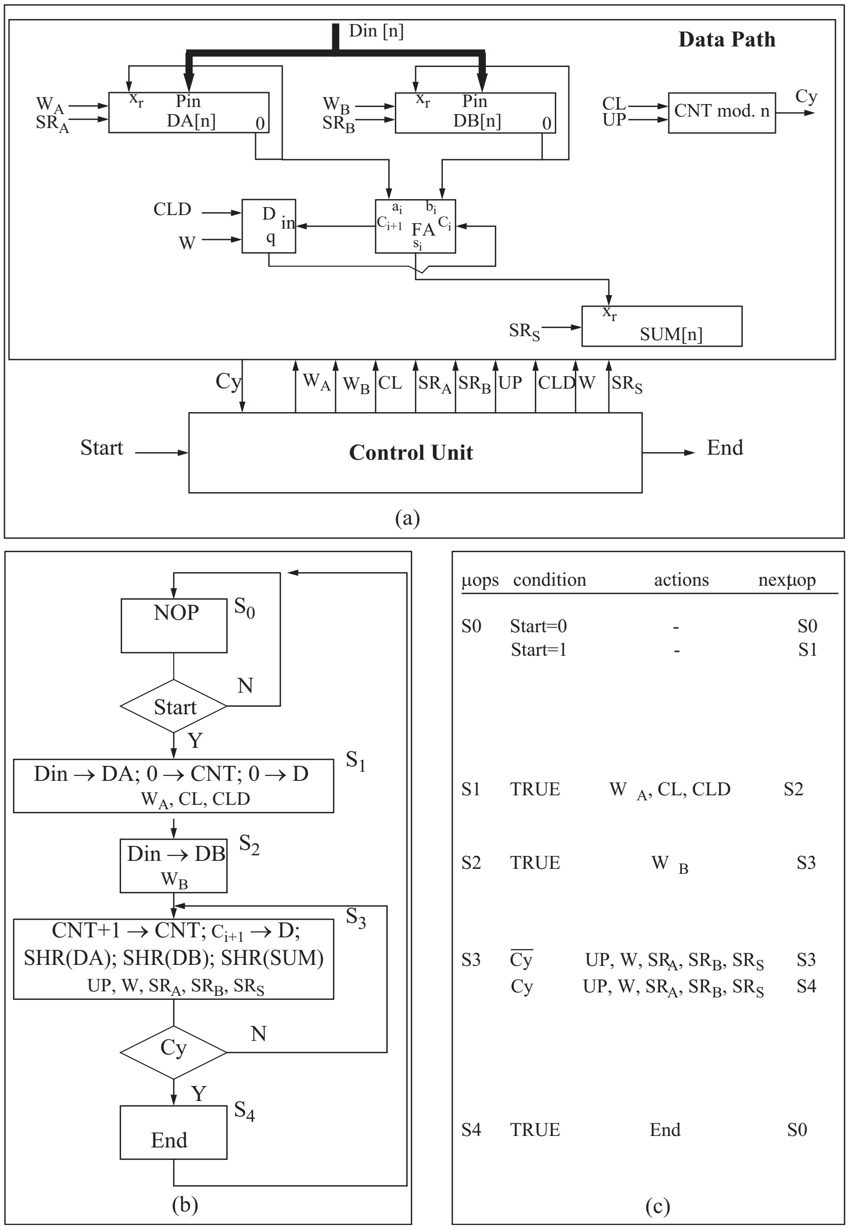
$$\text{reading}: \quad DataOut = [R_{source}]$$

where "*" is the operation between *DataA* and *DataB*.

Because a digital system is very complex at the RT level, it is advisable to split up its tasks into actions that affect the data (storage, transport, calculation, etc.) and control actions (sequencing, decision taking, etc.). Digital system structures at the RT level, therefore, have two units: the data path unit and the control unit.

The data path encompasses all of the functional units that store and process data and the buses that interconnect them. For example, Fig. 1 shows an *n*-bit serial adder with a start signal (*Start*) and an end-of-operation signal (*End*), in which both input data enter consecutively via the parallel bus *Din*. The data path unit contains the three *n*-bit registers where *A* and *B* data and the result of the addition (*SUM*) are stored. It also has an *n*-module counter (*CNT*) to count the number of bits, a full-adder and a 1-bit register

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.



μops	condition	actions	nexμop
S0	Start=0	-	S0
	Start=1	-	S1
S1	TRUE	W_A, CL, CLD	S2
S2	TRUE	W_B	S3
S3	$\overline{Cy}$	UP, W, SR_A, SR_B, SR_S	S3
	Cy	UP, W, SR_A, SR_B, SR_S	S4
S4	TRUE	End	S0

Figure 1. Binary Serial Adder: (a) data path, and control unit; (b) ASM chart; and (c) FSMD microprogram.

(i.e., bistable) to carry out the series addition. These components have specific operation selection inputs, which, in this case, are as follows: clear (*CL*, *CLD*), shift right (*SRA*, *SRB*, *SRS*), count up (*UP*), and parallel write (*WA*, *WB*, *W*).

The set of register transfers that can be executed in one single clock cycle is called the micro-operation (μop), and is the basic operation performed by a digital system. However, from a broader perspective, a digital system executes essentially an instruction or a specific macro-operation belonging to a set of instructions of the digital system. For an instruction to be executed, several clock cycles—or several μops—usually are required. Therefore, a sequence of μops must be obtained for each instruction so that, when the sequence is executed, the instruction can be delivered. The set of μops sequences for all instructions constitutes the digital system‘s control microprogram.

Evidently, the data path design should allow execution of all the μop of the control microprogram. The purpose of the control unit is to execute the control microprogram, and therefore it has the following functions: (*1*) to control which μop must be performed in each clock cycle, (*2*) to generate the system control outputs (i.e., *End* in Fig. 1) and the data path operation selection signals that execute the μop (i.e., in the example in Fig. 1, it will activate *SRA* to shift the DA register, it will activate *UP* to make CNT count, etc.), and (*3*) to evaluate the compliance of the control conditions: in Fig. 1, *Start* and *Cy*.

The control microprogram might be seen as a sequence of actions from two perspectives: first, as a data flow expressed by the RT operations of each μop and, second, as a control signal activation sequence, whether those signals are digital system outputs (for example, End) or signals from the registers (for example, SRA). The two perspectives should be coherent. To describe the control microprogram, graphic techniques are employed, such as ASM (algorithmic state machine) charts or techniques associated with the computer languages generically known as hardware description languages (HDL), of which a great number have been developed (ABEL, AHPL, DDL, Verilog, VHDL, etc.). The most relevant of these languages will be addressed in more detail in the article.

The ASM chart is a very intuitive, useful, and simple technique to describe control microprograms, especially for synchronous digital systems. Each rectangular box identifies a μop—a state of the program or, in other words, the actions executed within the corresponding clock cycle, whether they affect data (RT operations) or control (activating control signals). Figure 1(b) shows the ASM chart that describes both the data flow and the control instructions for our example. In this case, the actions to be executed in the S2 state are DB < = Din (data view) and WB = 1 (control view). An alternative description of the control microprogram, now using a very simplified HDL, which corresponds to what is known as the finite state machine with data path(FSMD) (3) is shown in Fig. 1(c).

The data path in Fig. 1 is very simple and its design is specific. Designing more complex systems is far more difficult. In general, for the best chance of success, the data path should specify the following aspects in as much detail as possible: the functional blocks available (for both registers and processing), the data flow sequencing structures (parallel/pipeline level), and the interconnection architecture (one, two, three, or more buses). The better these elements are specified, the easier and more efficient it will be to synthesize the data path from the description at the most abstract level.

The control unit design can be obtained with the techniques employed habitually in FSM design, and the control unit therefore can be implemented with random logic, using bistables and gates. However, because of implementation restrictions, it is preferable to synthesize the control unit from the RT level. Automatic synthesis from the ASM control microprogram description is easier if it is used one bistable per state in the design of the control unit:

- every Yes state leads to a D-type bistable,
- every decision (i.e., if Cy then S4 else S3, in S3 on Fig. 1(b)) requires a 1:2 demultiplexer with Cy as the selection signal,
- and OR gates join signals that are activated (with 1) in different parts, for example, bistable input D3 is ORing between the q2 output of bistable 2 and the 0 (NOT Cy) output of the demultiplexer controlled by Cy.

Because the ROM-register or PLA-register structures are general FSM design structures, they facilitate control unit synthesis from the RT description. In this case, the microprogram is “written” in ROM (or PLA). In this technique, known as microprogrammed control, the register is a pointer to the μop that is executed and transmitted (together with other control inputs) to ROM/PLA, whereas the ROM/PLA content produces the code for the following μop via a subset of outputs and the values for the control unit output signals via the remaining ROM/PLA outputs. The control microprogram, written in ROM/PLA, constitutes the firmware. Firmware engineering studies ways to optimize the size and the performance of the ROM/PLA-register solution by limiting the data path operation.

HARDWARE DESCRIPTION LANGUAGES

Most automatic synthesis tools divide the synthesis process up into hierarchically organized stages that transfer a specific system description to another description with a greater level of detail. The initial system description usually is expressed in a high level programming language (Pascal, C, etc.) or a an HDL (VHDL, Verilog, etc.).

Hardware specifications can be represented in different ways. Tables and graphs produce representations of greater visual clarity, but do not handle large sizes efficiently. In these cases, language-based descriptions are more versatile than tables and are more machine-readable than graphs.

Specific hardware description languages should be used because high level programming languages (such as C, C++, etc.), although feasible, are not efficient. Their inefficiency stems from the fact that, because they do not possess elements suitable to describe hardware (for example, integrated procedural and nonprocedural paradigms), they require more code lines than specific HDL to describe the same function. As a result, the descriptions they generate are more difficult to understand. Furthermore, in languages designed to describe software, the compiler or translator adapts the descriptions to the machine that will execute the program (resources, architecture, etc.), whereas in HDLs the specification represents the machine that is executing the algorithm, its resources, and so an.

VHDL was created in the early 1980s as part of a U.S. Department of Defence project called VHSIC (Very High Speed Integrated Circuits). The project required a language that could be used to describe the systems being designed and would perform two specific functions: first, allow the designs to be self-documenting and, second, serve as a means of simulating the circuits being studied. In 1985, the DATC (Design Automation Technical Committee) of the IEEE (Institute of Electrical and Electronics Engineers) expressed an interest in VHDL as a result of its need to describe circuits via a language that was independent of the design tools and that could cover the different levels of abstraction in the design process. VHDL provided a solution to the problem of compatibility between designs and the different CAD platforms. Considering that, at that time, VHDL was a language that met all the DATC‘s requirements, the VASG (VHDL Analysis and Standardization Group) was created to begin the process of standardization. Subsequently, in December 1987, the standard designated IEEE 1076-1987 officially appeared for the first time (4). The language has been revised to ensure its development over time. VHDL was created specifically to describe digital systems (5–7), but today a new language called VHDL-AMS (VHDL-Analog Mixed Signal) is available to describe analog and mixed signal circuits.

The VHDL standardization process coincided with that of Verilog, a logic simulation language for the Verilog-XL simulator owned by Cadence Design Systems. Verilog was freed in 1990, allowing the creation of the OVI (Open Verilog International) organism and marking the beginning of the language‘s standardization process. The first standard version appeared in 1995 and was designated IEEE 1364-1995 (8). Later, the Accellera organism (9) was created when OVI and VI (VHDL International) merged to promote new standards and to develop those already in existence. VHDL and Verilog, like most high level programming languages, are imperative languages. These languages are based on a declarative syntax in which the desired problem is expressed through a set of instructions that does not detail the method of solution: That is to say, the sequence of instructions is not relevant. But VHDL and Verilog also allow a procedural syntax (where the desired action is described via a sequence of steps in which the order of execution is important) to be applied for certain specific instructions such as function, procedure, and process.

A VHDL description is composed of a series of design units that allow the different elements that define a circuit to be specified. The basic design unit is called an entity. The entity allows the circuit‘s interfaces (for example, input and output ports) to be defined. Through this unit, the circuit communicates with its surroundings. The entity represents the system as a black box interface accessible only via the ports. Inside that black box, another design unit—called architecture—is described. Architecture enables the behavior of a circuit or its structure to be specified. Because any system can be described in several ways, a circuit can be modeled by several architectures, but for any specific design, only one entity exists.

Architecture specification has two areas: a declarative area and the architecture body. In the former, those elements to be used in the description are declared, including the components that describe the circuit diagram, internal signals, functions and procedures, the data types to be used, and so on. But it is in the architecture body that the system is described. The instructions included in the architecture body are concurrent: That is, the instructions are executed simultaneously. These instructions serve to instance and interconnect components, execute procedures, assign values to signals via conditional or unconditional assignation instructions, and so on.

This type of description can be used to specify the circuit both structurally (schematically) and functionally (describing the system‘s equations). Procedural syntax is required to specify an algorithm, which, in VHDL, is possible through processes, functions, and procedures. A process is a concurrent instruction (because it is used within the body of an architecture) that contains sequential instructions that are executed one after another according to the established programming flow. These instructions are typical of any procedural programming language, such as loop instructions, “if . . . then . . . else” instructions, variable assignation instructions, and jumps and subroutine returns. Functions and procedures also have a procedural syntax; the difference between the two is that functions can return one value but procedures can return more than one value.

In VHDL, the user is responsible to define data types, operators, attributes, and functions. Specific instructions exist to create new data types or even to use previously defined types to create new ones. In this respect, the overload capacity of the operators and the functions is very useful: Different operators or functions can be created with the same name, distinguishable only by their parameters.

In the standard Verilog language, the basic design unit is the module. A module contains the definition of all the elements that constitute the system. It is headed by a list of input/output gates equivalent to the entity in VHDL. Inside the module internal signals (*wire*), inputs (*input*) and outputs (*output*) are defined. The module also describes the functions/structure of the system.

Certain similarities exist between the VHDL and Verilog languages: They both have a set of concurrent instructions as the basis for their descriptions. The instructions within the Verilog module are concurrent instructions and are executed when events occur at the inputs. For algorithmic descriptions, a sequentially executed program flow must be represented, and therefore a set of sequential instructions exists, composed of *always*, *initial* instructions (equivalent to processes in VHDL), procedures (*task*), and functions. *Always* and *initial* instructions are concurrent instructions (specified within a module) that contain sequential instructions. The difference between *always* and *initial* is that the latter only is executed once during the simulation whereas the former is executed each time an event occurs at input signals.

Perhaps the most important difference to be found between the VHDL and Verilog languages is their respective philosophies. Verilog is a language that originated in logic level descriptions (it was created as a logic simulator language), which makes it very suitable to generate descriptions at this level, because it contains elements that facilitate specifications (data types, primitives, specification of timing parameters, etc.). In this language, the user employs the facilities available but is not able to define new elements. In contrast, in VHDL, the user defines the elements to be used (data types, operators, etc.).

All these characteristics of the VHDL and Verilog languages make them easily adaptable to system modeling involving different description techniques (structural, functional, algorithmic). Therefore, languages are very powerful as logic synthesis languages because they cover different levels of abstraction for digital systems. These are alive languages and have update mechanisms to adapt to new requirements. Indeed, although originally designed for documentation and for simulation, today the use of these two languages is extended to other areas of application, such as high level synthesis, electrical level circuit modeling, and performance analysis.

For other applications, more suitable languages are emerging. For example, in the late 1990s the Vera language for system verification was developed. This language is oriented toward verification tasks (hardware verification language, HVL). Its features include constructions that facilitate functional verification, such as testbench creation, simulation, and formal verification. Vera has had great influence in the development of new languages such as SystemVerilog (standard IEEE 1800-2005). SystemVerilog is an extension of Verilog that includes C constructions, interfaces, and other descriptive elements. The aim of the language is to cover description levels with a greater degree of abstraction to include synthesis and, above all, verification applications; therefore, it is known as a system level design/verification language (hardware design and verification language).

To describe system specifications at a higher level, languages are required that allow those specifications to be defined without undertaking a priori decision with regard to their implementation. To meet this challenge, the SystemC language was created. It was approved as a standard language in December 2005 under the designation IEEE 1666-2005. Basically, it is composed of a C++ library aimed to facilitate hardware description from C++. The level of abstraction addressed by this language is required to specify systems that contain a global description: That is, both hardware-related aspects and those aspects associated with the software to be executed on the hardware are described. Therefore, it is a very suitable language to generate specifications in the field of hardware–software codesign environments. In such environments, the baseline is to produce system descriptions that do not pre-establish the nature of the system‘s eventual implementation (either in hardware or software). It is the codesign tools that will decide which functions will be implemented in hardware and which in software. Languages employed in this type of system are usually object-oriented programming languages.

VHDL FOR SYNTHESIS

One of the new applications for HDLs is circuit synthesis (1,2,10–12). When VHDL or Verilog are used for synthesis, certain restrictions must be imposed on the language. Basically, two factors are involved. First, the way the language handles time is decisive. Because both languages were designed for simulation, time is well defined. Simulation is controlled by events; the simulator clock runs in accordance with the status of the queued events. But synthesis tools are not controlled by events. The tool determines the timing of the tasks. In other words, it is not possible to predict when the operations will be executed because the synthesis tool schedules the tasks. The differences that exist between simulation modeling and synthesis modeling should also be taken into account. In simulation modeling, the designer can specify delays in signal assignments and in the execution of processes. In synthesis modeling, the designer can establish no absolute conditions on time whatsoever, because it depends on how the circuit has been implemented, on the technology employed, and on the objectives and restrictions that have been established. These factors will determine delays. Restrictions must be imposed on the language to limit signal assignments, beginnings, and ends of processes. Descriptions tend to be generated with syntax that is more declarative than procedural. The second decisive factor to restrict HDLs when they are used for synthesis is that certain instructions only make sense when they form part of a simulation. For example, with VHDL, file type and file object are only significant from a computing point of view, but these terms are meaningless in terms of circuits.

Moreover, the way hardware codes should be written is different from the way they should be written for programming or simulation. It would be possible to have codes that, although complying with synthesis restrictions, produce inefficient or even functionally incorrect designs. The specific rules depend on each individual synthesis tool. In most of these tools, the restrictions imposed are very similar.

This section examines, from a practical point of view, a series of guidelines that should be followed to obtain a VHDL code that is suitable not only for synthesis but also is efficient in terms of results.

Prohibited or Nonrecommendable Sentences

Some VHDL data types are not useful, or not supported for synthesis. These data types include physical types (such as time, voltage, etc.), real number types, and floating point types. Arithmetical operations that are supported by synthesis include add, subtract, and product operations. As a rule, synthesis tools do not support division or more complicated operations. For supported operations, synthesis tools implement predesigned structures that vary with the restrictions imposed on the design. The use of other structures generates a detailed description.

Another restrictive condition is the use of time. Usually, synthesis tools prohibit expressly the use of delayed signal assignations. Others simply ignore them. But the synthesis tool evidently will attempt to implement a circuit‘s functionality, and the explicit declaration of these delays makes no sense, which explains why multiple assignations to a signal are not allowed within a single sentence. Neither is it allowed, within a process, to have several assignations for the same signal all of which must be executed at the same moment. Similarly, “WAIT for XX ns” sentences are not allowed because it would be very difficult to implement such sentences (with reasonable accuracy) in hardware, which would also be a nonrecommendable design practice. The use of WAIT is very much restricted and, as will be shown, it can only be used to synchronize a process with a clock edge.

The initialization of signals or variables in the declaration sentence is not taken into account by the synthesis tool, but no error is produced. Therefore, these sentences should be used with great care, because they may cause different behavior in simulations before and after synthesis.

Design Issues

As mentioned, the way the code is written can be used to obtain designs with the same functionality but that differ greatly in terms of complexity and performance. The following is a list of recommendations to build the code to obtain efficient results:

- Hierarchical design. Hierarchy facilitates the reuse, debugging, and readability of the design, but certain guidelines should still be followed. To facilitate reusability, blocks should be built as standardized as possible (registers, FIFO's, etc.). To facilitate readability, the interblock data flow must be appropriate, with minimal routing between blocks. To facilitate documentation, both the different components used and the different elements inside preferably should be tagged and comments should be added.

- Use of parametrizable blocks. The main synthesis tools support construction of a generic unit that can be assigned the values of certain parameters at the moment of instantiation. The value assignment is done by including "generic" sentences in the entity, which makes it possible to have a library of intellectual property (IP) components that can be used and adapted for different designs.
- Avoid embedded IF instructions. Usually, tools do not synthesize efficiently several embedded conditions. It is advisable to use more ELSIF clauses or to separate the IF-THEN sentences. In some cases, it may be better to use CASE sentences because synthesis tools have a model based on multiplexers that is generally better than the description of the same using IFs.
- Use the style most appropriate for the state machines. Many digital problems can be solved simply by means of a state machine. VHDL has many different styles to describe state machines, but the synthesis tool may not identify them and may not produce the optimum end circuit. Usually, synthesis tool manuals contain examples of how to produce such descriptions in such a manner that they will be understood as state machines.
- Types of binary data. For binary signals, it is advisable to use "std_logic" types for 1-bit signals and "std_logic_vector" for buses. These types contain not only the '0' and '1' values but also contain additional values such as 'X', and 'Z', which allow the functional simulation to imitate reality more effectively by incorporating unknown values into the data.
- Buses with binary numbers. For those buses that contain binary data to be used for synthesizable arithmetical operations (add, subtract, and product), it is advisable to use the "signed" and "unsigned" types for signed and unsigned numbers, respectively. The latest versions of functions packages do not have definitions of arithmetical operators for the "std_logic_vector" type.
- Use of integers. Integers may be used in synthesis, but should be limited in range to ensure the minimum number of bits is employed when implemented by the synthesis tool. Integer values are capped at the moment of declaration:

```
signal number1: integer range 0 to 15;
```

Inference of Combinational Circuits

Synthesis tools do not involve elements of memory unless the elements are necessary. In VHDL, combinational logic can be described via concurrent signal assignation or via processes. A set of concurrent signal assignations describes combinational logic whenever the assigned signal does not form part of the assignation and the set of concurrent assignations has no combinational links that loop assignations.

Combinational logic can also be described through processes. A process describes combinational logic whenever its sensitivity list includes all of the signals involved in the assignations and all of the output signals are specified completely, which usually applies to conditional instructions. The presence of a condition for which the signal is not assigned—that is, it remains unchanged—implies a latch.

Inference of Sequential Circuits

The synthesis of sequential circuits via VHDL descriptions is more effective for synchronous processes than for asynchronous implementations. Synchronous circuits work better because events are propagated on the clock edges, that is, at well-defined intervals. Logic stage outputs also have the whole clock cycle to pass on to the next stage, and skew between data arrival times is tolerated within the same clock period. The description of asynchronous circuits in VHDL employed for synthesis is more difficult.

A clock signal exists in synchronous circuits, for which both event and clock edge must be identified. In VHDL, the most usual form of specification is:

```
clk'event and clk = '1'
```

In this case, a rising edge has been specified. Clock signals should be used in accordance with a series of basic principles:

- Only one edge detection should be allowed per process: That is, processes may only have one clock signal.
- When a clock edge is identified in an IF, it should not be followed by an ELSE.
- The clock, when specified with an edge, should not be used as an operand. The instruction

```
IF NOT (clk'event and clk = '1')
THEN ... is incorrect.
```

These language restrictions are imposed with hardware implementation results in mind. Other alternatives either do not make sense or are not synthesizable. One consequence of these restrictions is that signals can only change with one single edge of one single clock.

In accordance with these restrictions, two basic structures exist to describe synchronous circuits, one with asynchronous reset and the other without asynchronous reset. These two structures are shown in Fig. 2. For processes with asynchronous reset, the process sensitivity list should include only the clock signal and the asynchronous reset.

FSM Descriptions

Generally an FSM is used to describe a system's control unit. In this case such a machine generates the values of the control signals that act on the data path or that act as

```
process
begin
  wait on clk'event until clk = '1';
  Q <= Din;
end process;
```

a)

```
process(clk, reset)
begin
 if (reset = `1') then
      Q <= '0';
 elsif (clk'event and clk = `1') then
      Q <= Din;
 end if;
end process;
```

b)

Figure 2. Synchronous processes, (a) edge triggered flip-flop, (b) edge triggered flip-flop with asynchronous reset.

system output control signals. State machines can be defined in VHDL using different description styles, but the results obtained may vary with the style used. Some are recognized directly by the synthesis tools and can therefore take full advantage of the optimizations offered by such tools. Although a distinction generally is drawn between descriptions for Moore machines and descriptions for Mealy machines, the codes are very similar; the only difference is that the codes for Mealy machine outputs are dependent on both the current state and the input taking place at that moment.

A typical style describes a state machine using two processes. One of them is totally combinational and describes the next state function and the output assignations. The second is sequential, triggered by a clock signal, and controls assignations on the state register. The corresponding code scheme is as follows:

```
entity FSM is
port (clock, reset: in std_logic; -- clock and
  reset signals
      x1, ..., xn: in std_logic; -- input signals
      z1, ..., zm: out std_logic); -- output
        signals
end FSM;
architecture BEHAVIOR1 of FSM is
 type STATE_TYPE is (S0, S1, ..., Sp);
 signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
begin
-- Process to hold combinational logic
COMBIN: process(CURRENT_STATE, x1, ..., xn)
begin
 NEXT_STATE <= CURRENT_STATE;
 case CURRENT_STATE is
      when S0 => -- state s0 assignations
                 -- next state assignation
                 -- output assignations
      when S1 =>
                 -- state s1 assignations
                 -- next state assignation
                 -- output assignations
      ...
      when Sp => -- state Sp assignations
                 -- next state assignation
                 -- output assignations
 end case;
end process;
-- Process to hold synchronous elements (flip-flops)
SYNCH: process
begin
 wait until CLOCK event and CLOCK = 1;
 CURRENT_STATE <= NEXT_STATE;
end process;
end BEHAVIOR;
```

The sequential process may also include a synchronous reset signal:

```
SYNCH: process
begin
 wait until CLOCK event and CLOCK = 1;
 if (reset = '1') then
       CURRENT_STATE <= S_reset;
 else
       CURRENT_STATE <= NEXT_STATE;
 end if;
end process;
```

This form of description is the one recommended, for example, for the commercial synthesis tool Design Compiler by Synopsys, Inc. (13).

Application Example

In the state machine example below, the control unit (see Fig. 3) for the sequential adder described in Fig. 1 is implemented. The corresponding VHDL code architecture is:

```
architecture fsm of control_unit is
       type STATE_TYPE is (S0, S1, S2, S3, S4);
       signal CURRENT_STATE, NEXT_STATE:
         STATE_TYPE;
begin

COMBIN: process(CURRENT_STATE)
begin
-- NEXT_STATE <= CURRENT_STATE;
case CURRENT_STATE is
     when S0 =>
           wa <= '0'; wb <= '0'; cl <= '0';
             sr_a <= '0'; srb <= '0';
           up <= '0'; cd <= '0'; w <= '0';
             srs <= '0'; s_end <= '0';
           if start = '0' then
                 NEXT_STATE <= S0;
           else
```

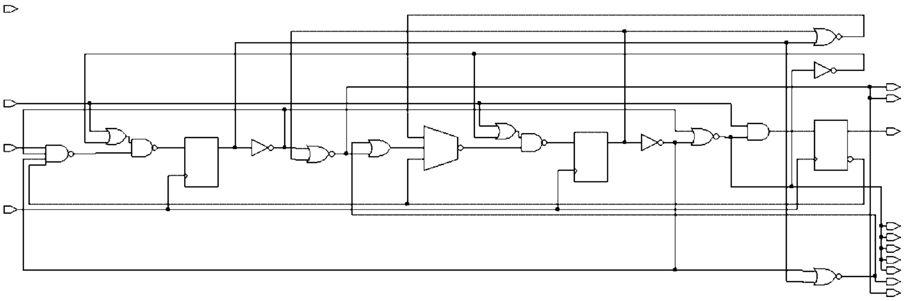


Figure 3. Control unit schematic.

```
                NEXT_STATE <= S1;
           end if;
     when S1 =>
           wa <= '1'; wb <= '0'; cl <= '1';
             sr_a <= '0'; srb <= '0';
           up <= '0'; cd <= '1'; w <= '0';
             srs <= '0'; s_end <= '0';
           NEXT_STATE <= S2;
     when S2 =>
           wa <= '0'; wb <= '1'; cl <= '0';
             sr_a <= '0'; srb <= '0';
           up <= '0'; cd <= '0'; w <= '0';
             srs <= '0'; s_end <= '0';
           NEXT_STATE <= S2;
     when S3 =>
           wa <= '0'; wb <= '0'; cl <= '0';
             sr_a <= '1'; srb <= '1';
           up <= '1'; cd <= '0'; w <= '1';
             srs <= '1'; s_end <= '0';
           if cy = '0' then
                NEXT_STATE <= S3;
           else
                NEXT_STATE <= S4;
           end if;
     when S4 =>
           wa <= '0'; wb <= '0'; cl <= '0';
             sr_a <= '0'; srb <= '0';
           up <= '0'; cd <= '0'; w <= '0';
             srs <= '0'; s_end <= '1';
           NEXT_STATE <= S2;
  end case;
end process;

-- Process to hold synchronous elements (ip-ops)
SYNCH: process
begin
     wait on clk until clk = '1';
     CURRENT_STATE <= NEXT_STATE;
end process;
end fsm;
```

The corresponding FSMD includes the operation of the datapath instead of the control signals assignations (14). Each state defines the operations that execute the processing unit in a single clock cycle. The code of the above example is shown as follows:

```
architecture fsmd of example is
       type STATE_TYPE is (S0, S1, S2, S3, S4);
       signal CURRENT_STATE, NEXT_STATE:
         STATE_TYPE;
begin

SUM(N-1) <= DA(0) + DB(0);

COMBIN: process(CURRENT_STATE)
begin
-- NEXT_STATE <= CURRENT_STATE;
case CURRENT_STATE is
       when S0 =>
            if start = '0' then
                   NEXT_STATE <= S0;
            else
                   NEXT_STATE <= S1;
            end if;
       when S1 =>
            DA <= Din; CNT <= (others=>'0');
              D <= (others=>'0');
            NEXT_STATE <= S2;
       when S2 =>
            DB <= Din;
            NEXT_STATE <= S2;
       when S3 =>
            CNT <= CNT+1; D <= C;
            DA <= shr(DA); DB <= shr(DB);
              SUM <= shr(DB);
            if cy = '0' then
                  NEXT_STATE <= S3;
            else
                  NEXT_STATE <= S4;
            end if;
       when S4 =>
       s_end <= '1';
       NEXT_STATE <= S2;
  end case;
end process;

-- Process to hold synchronous elements (ip-ops)
SYNCH: process
begin
     wait on clk until clk = '1';
     CURRENT_STATE <= NEXT_STATE;
end process;
end fsmd;
```

BIBLIOGRAPHY

1. H. Bhatnagar, *Advanced ASIC Chip Synthesis*, Norwell, MA: Kluwer Academic, 2002.
2. M. Zwolinski, *DigitalSystem Design with VHDL*, Englewood Cliffs, NJ: Pearson Prentice Hall, 2004.
3. D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis. Introduction to Chip and System Design*, Norwell, MA: Kluwer Academic, 1992.
4. *IEEE Standard VHDL Language Reference Manual*, 1987, 1993
5. R. Lipsett, C. Schaefer, and C. Ussery, *VHDL: Hardware Description and Design*, Norwell, MA: Kluwer Academic, 1990.
6. J. M. Berge, A. Fonkoua, S. Maginot, and J. Rouillard, *VHDL Designer‘s Reference*, Norwell, MA: Kluwer Academic, 1992.
7. P. J. Ashenden, *The Designer‘s Guide to VHDL*, San Francisco, CA: Morgan Kaufmann, 2002.
8. *IEEE Standard Verilog Language Reference Manual*, 1995.
9. Accellera Organization, Inc., Available: http://www.accellera.org/
10. D . Naylor and S. Jones, *VHDL: A Logic Synthesis Approach*, London: Chapman & Hall, 1997.
11. A. Rushton, *VHDL for Logic Synthesis*, New York: John Wiley & Sons, 1998.
12. K. C. Chang, *Digital Systems Design with VHDL and Synthesis: An Integrated Approach*, New York: Wiley-IEEE Computer Society Pr, 1999.
13. HDL Compiler (Presto VHDL) Reference Manual, Synopsys, Inc.
14. P. P. Chu, *RTL Hardware Design Using VHDL*, New York: Wiley-Interscience, 2006.

FURTHER READING

C. R. Clare, *Designing Logisc Systems Using State Machines*, New York: McGraw-Hill, 1973.

F. P. Prosser and D. E. Winkel, *The Art of Digital Design. An Introduction to Top-Down Design*, Englewood Cliffs, NJ: Prentice-Hall, 1987.

J. Ganssle, *The Firmware Handbook(Embedded Technolgies)*, New York: Elsevier, 2004.

S. Palnitkar, *Verilog HDL. A Guide to Digital Design and Synthesis*, Englewood Cliffs, NJ: SunSoft Press, A Prentice-Hall Title, 1996.

P. Michel, U. Lauther, and P. Duzy, *The Synthesis Approach to Digital System Design*, Norwell, MA: Kluwer Academic, 1992.

G. DeMicheli, *Synthesis and Optimization of Digital Circuits*, New York: McGraw Hill, 1994.

M. A. Bayoumi, (ed.), *VLSI Design Methodologies for Digital Signal Processing Architectures*, Norwell, MA: Kluwer Academic, 1994.

J. Rozenblit and K. Buchenrieder, (eds.), *Codesign. Computer-Aided Software/Hardware Engineering*, Piscataway, NJ: IEEE Press, 1994.

R. Merker and W. Schwarz, *System Design Automation: Fundamentals, Principles, Methods, Examples*, Norwell, MA: Kluwer Academic, 2001.

J. P. Mermet, *Electronic Chips & Systems Design Languages*, Norwell, MA: Kluwer Academic, 2001.

ANGEL BARRIGA
CARLOS J. JIMENEZ
MANUEL VALENCIA
University of Seville-Institute of
Microelectronics of Seville
(CNM-CSIC)
Seville, Spain

M

MICROPROGRAMMING

INTRODUCTION

At the heart of any synchronous digital system is one or more state machines that, on each clock edge, examines inputs received from the system and generates control outputs. The traditional method of implementing such state machines has been combinational logic followed by flip-flops that hold the machine state. Design of the combinational logic for such state machines now can be automated by describing the state transitions in a hardware descrip tion language like VHDL or Verilog.

Microprogramming was introduced by Wilkes (1) in the early 1950s as an alternative to the traditional approach for implementing complex state machines like those found in computer control units (2–5) In this application, many inputs need to be examined, such as status register bits (carry, overflow, sign, etc.), and bus data ready. Many outputs also are needed, such as arithmetic-logic-unit function select, register load enables, memory access control bits, and so on. In the 1980s, standalone microsequencers became popular for other complex state machine tasks like network and direct memory access controllers (6,7). One reason for the early popularity of microprogramming was that, before the advent of hardware description languages, it translated the hardware design problem into a programming problem, which made it tractable to a wider range of designers. Control information is stored in the microprogram memory, and a new microinstruction is fetched from memory every clock cycle. In this way, microprogramming is similar to assembly language programming and typically supports subroutines, loops, and other structured programming concepts. Because program changes only require a change in memory contents, the rate at which the controller can be clocked does not change, no matter how significant the program change. This approach is in contrast to the traditional approach where design changes dramatically can impact the logic equations, amount of combinational logic, and clock frequency. This impact is true even if a hardware description language is used to automate the traditional design process. Complex state machines that fit and meet timing requirements before a design change may either not fit, not meet timing constraints, or both, following even simple modifications. On the other hand, once a microprogrammed design meets clock rate and logic resource requirements, state machines of any size within the microprogram memory capacity can be implemented and modified without changing this maximum clock frequency or the logic resources required.

A BASIC MICROPROGRAM SEQUENCER

The block diagram of a typical microprogram control unit is shown in Fig. 1(5). To simplify labeling the diagram, it is assumed that the microprogram memory has N address lines (2^N microinstruction locations) and that each location is $N + \mathrm{L} + 4$ bits wide, where L is the number of control output lines. In this example it also is assumed that seven (or fewer) condition inputs are to be examined. The two registers are standard clocked D registers, and the incrementer outputs a value equal to the input plus one. Multiplexer MUX1 is eight-input multiplexer used to select either logic 0 or one of the seven condition inputs, and multiplexer MUX2 represents N two-input multiplexers used to select the source of the microprogram memory address.

The microprogram memory address can be forced to location zero to fetch the first microinstruction by taking the $\overline{\text{Reset}}$ line low to force the MUX2 outputs to zero. In its simplest form, all that is required of the sequencer is to fetch and output the contents of successive microprogram memory locations. Such instructions are referred to as continue instructions and are implemented by setting the polarity bit to logic 0 with the select bits picking the Logic-0 input of MUXl. In this way, the MUX2 select input is always 0 and the incrementer and microprogram counter register form a counter. For the microprogram control unit to be able to select other than the next sequential address, it is necessary that it be able to load a branch address. Such unconditional branch instructions are implemented by setting the polarity bit to 1 with the select bits picking the logic-0 input of MUXl, which forces the next address to come from the branch address lines. Notice that in this case, the incrementer output becomes the branch address plus one so that the microprogram counter register is loaded with the correct value following the branch. The controller gains decision-making capability whenever the select inputs choose one of the condition inputs. In this case, assuming the polarity input is 0, the next sequential address is taken if the condition is 0 and the branch address is taken if the condition is 1. Such an instruction is referred to as a conditional branch if 1. Polarity of the branch can be reversed by setting the polarity bit that inverts the exclusive–or gate output.

Example 1 shows the state diagram of a very simple sequence detector, along with an equivalent microprogram for the control unit of Fig. 1 The detector input is shown on the transition arrows, and the detector output in a given state is shown within the state circle. In the state diagram and in the microprogram, X represents a "don't care." It is assumed that the sequence detector input is applied to MUX1 input 1 so that it is selected by condition select 001. Unconditional continue and branch are obtained by selecting MUX1 input 0, which is always 0.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

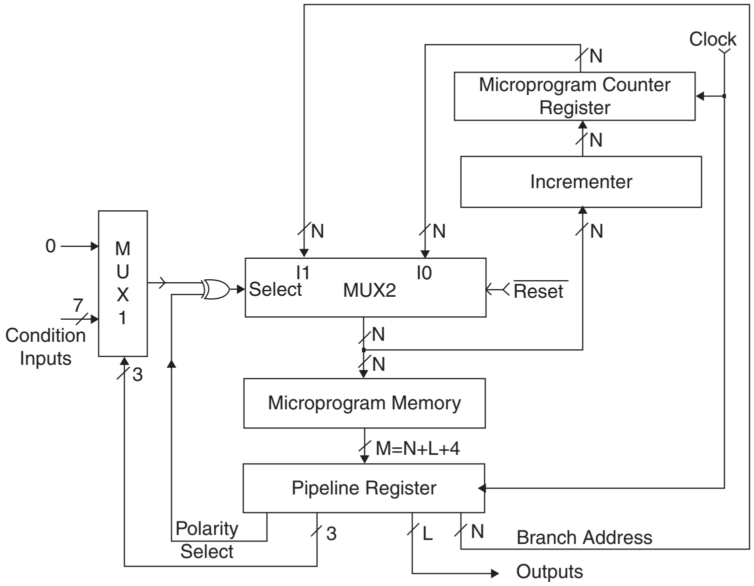
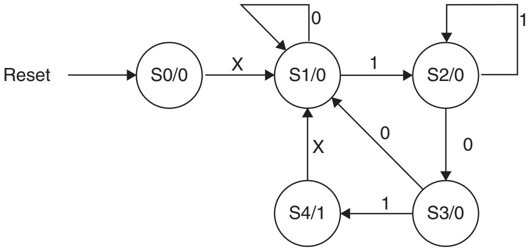


Figure 1. Basic microprogram control unit.

SUBROUTINES AND LOOPS

Figure 2 is a block diagram of an enhanced microprogram control unit that permits microprogram subroutines and microprogram loops. This microsequencer has a novel architecture that takes advantage of the enhancements that exist in coarse-grained FPGAs to implement efficiently four basic functions: registers, multiplexers, adders, and counters. Subroutine capability has been added to the basic design of Fig. 1 by including a subroutine last-in first-out (LIFO) stack for storing return addresses. Subroutine calls simply are branches for which what would have been the next sequential address is stored into the stack from the microprogram counter register. Microcode subroutines return to the calling program by selecting input C on the next address select logic and popping the stack.

Looping capability has been included with an *N*-bit down counter that can be loaded, via MUX2, either from another LIFO stack or from the branch address field. Counter load instructions are special continue instructions for which the branch address bit field is selected in MUX2 and loaded into the counter. Then it is possible to define a loop instruction that decrements the counter and selects either the branch address (top of loop) or, the next sequential address, based on the value of the counter. When the counter is zero at the time of the decrement, the counter holds at zero and the loop



Memory Location	State	Instruction	Branch Address	Condition Select	Polarity	Output
0	S0	Continue	X	0 0 0	0	0
1	S1	Branch if 0	1	0 0 1	1	0
2	S2	Branch if 1	2	0 0 1	0	0
3	S3	Branch if 0	1	0 0 1	1	0
4	S4	Branch	1	0 0 0	1	1

Example 1. Example Microprogram.

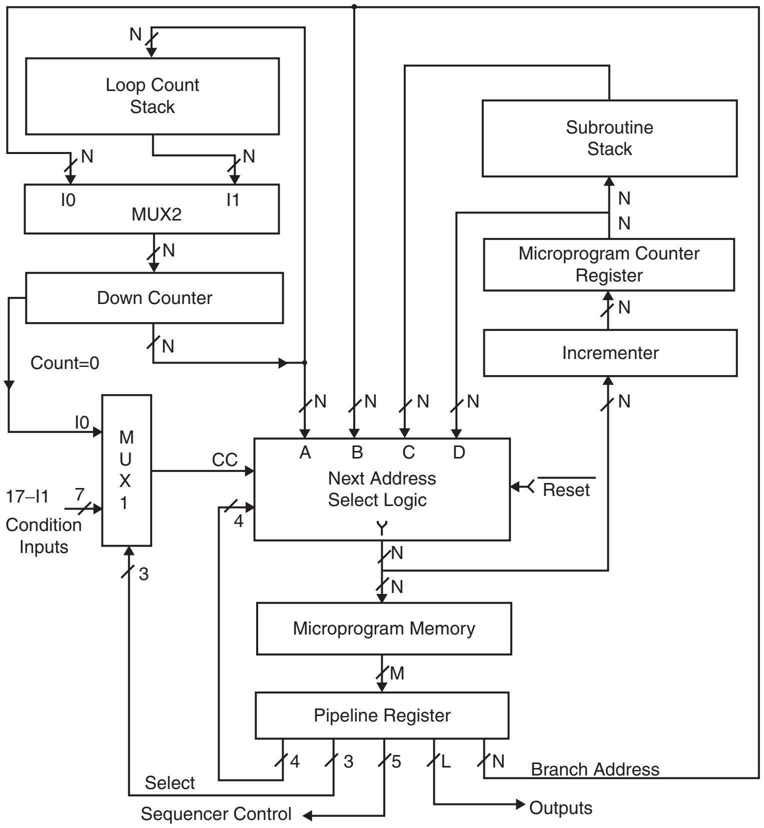


Figure 2. Enhanced microprogram control unit.

is exited. Thus, loading the counter with zero causes a loop to execute once, and loading it with K causes the loop to execute K + 1 times. The LIFO stack is used to store the counter value, if desired, whenever a new counter value is loaded, so loops can be nested. A load counter and pop counter stack instruction would be used to load the counter from the stack and pop the stack.

An alternate use of the down counter is in two-way branching. By loading the counter with an instruction address rather than a count, it is possible to use a condition input to select either the counter address (next address select logic input A) or the branch address (input B). In this way, one of two non sequential addresses can be selected.

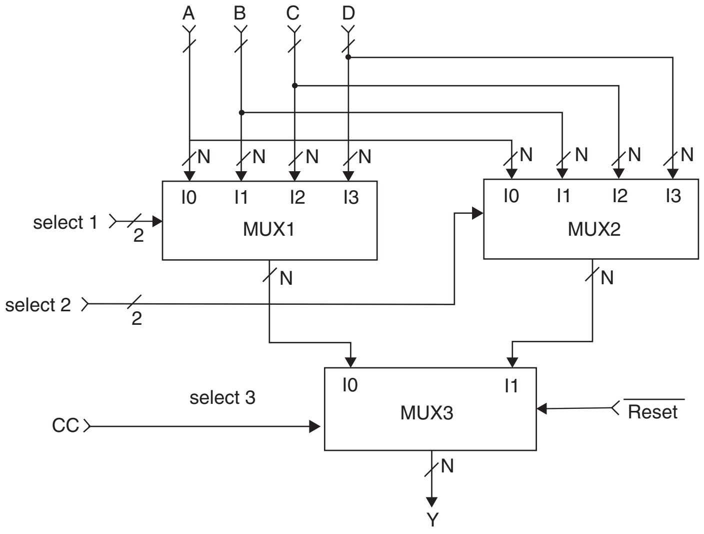


Figure 3. Multiplexer-based next address select logic.

Table 1. Instruction set

Instruction Mnemonic	Function
CONT	Continue to the next sequential instruction.
LDCT	Load counter and continue.
PSHLDCT	Push counter value onto the stack; load counter and continue.
POPCT	Load counter from the stack and pop the stack;
LOOP	Decrement the counter. If the counter was not zero prior to the decrement, take the branch address. Otherwise, take the next sequential address.
BR	Unconditional branch.
CBF(C)	Conditional branch if condition input C is false (next sequential address if C is true).
CBT(C)	Conditional branch if C is true.
TWB(C)	Two-way branch. Take the counter value if C is false and the branch address if C is true.
CALL	Unconditional subroutine call.
RET	Unconditional return from subroutine.

Figure 3 shows an implementation of the next address select logic based on multiplexers. This approach has an advantage in field programmable gate array (FPGA) implementations because FPGAs are designed to be capable of efficiently implementing multiplexers (8). Notice that the four bits from the pipeline register form two separate multiplexer-select fields, selectl and select2, for multiplexers MUX1 and MUX2. The condition input, CC, serves as the select for MUX3. If both selectl and select2 are set to the same input, then that becomes the output regardless of the condition input. However, different values for selectl and select2 allow conditional selection of one of two inputs. Reversing selectl and select2 swaps the polarity of this conditional selection. Because MUX3 is composed of simple 2-input multiplexers, the $\overline{\text{Reset}}$ input is brought into this multiplexer. In this way, each bit of each multiplexer can be implemented in a single look-up table block on most FPGAs. Something to observe about this implementation is that the delay through MUX1 and MUX2 in Fig. 3 occurs in parallel with the delay through the condition code selection multiplexer (MUX1 of Fig. 2). This balancing of delays is important for achieving high-speed operation because the maximum clock rate of the sequencer usually is limited by the path from the pipeline register through the select inputs of MUX1 to the next address select logic.

MICROINSTRUCTION DEFINITION

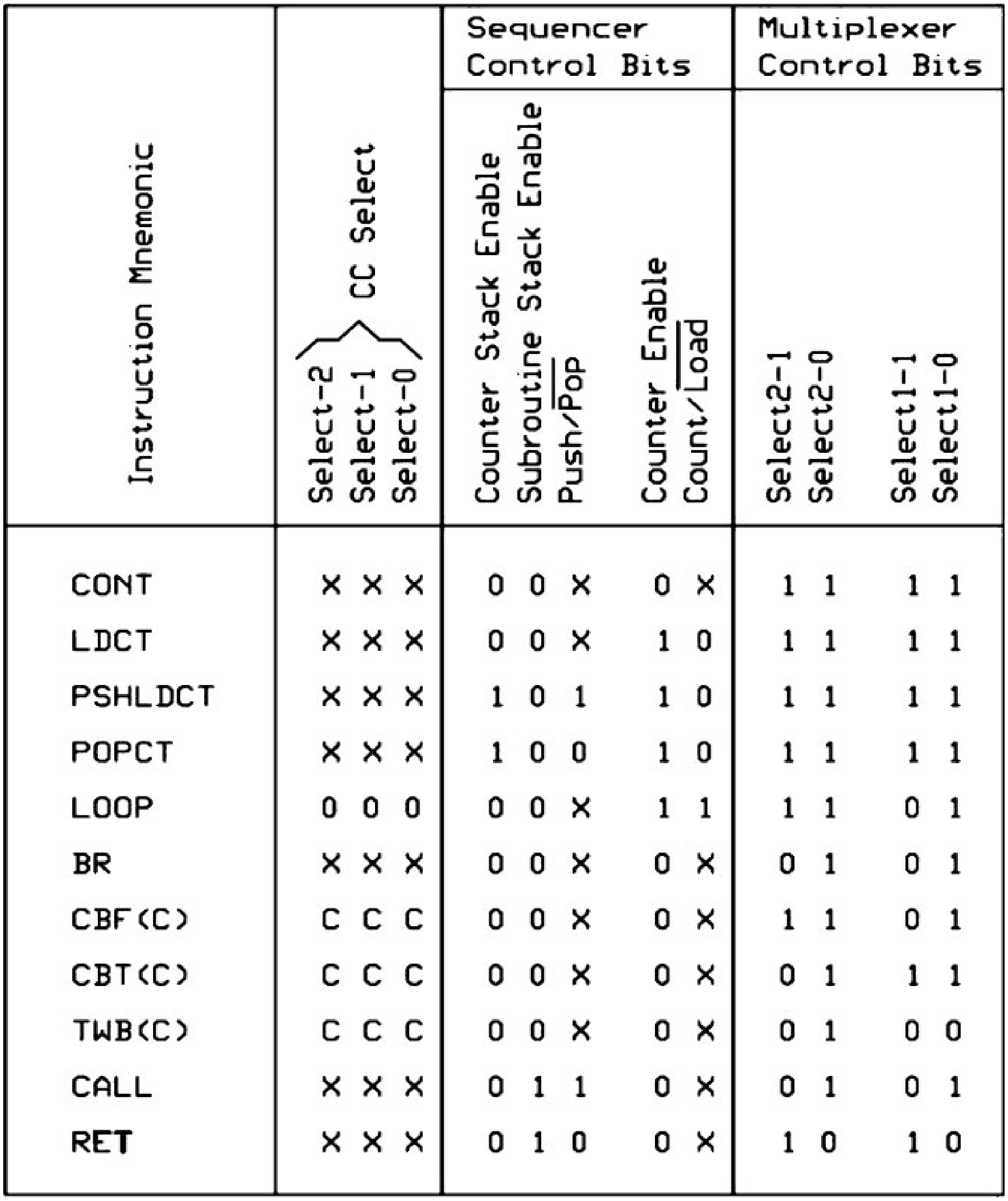
The microsequencer of Fig. 2 and Fig. 3 can be used to define and implement many different microinstructions. Table 1 lists one possible set of instructions that would be sufficient for most applications, and Table 2 shows how they are encoded. Referring to Fig. 2, these instructions are defined by 12 bits from the pipeline register. Three of these bits select the condition input via MUX1 and will be referred to as the *CC select* bits. This field would increase if a larger MUX1 were used to look at more than seven condition inputs. Four more of the instruction bits are the four bits from the pipeline register to the next address select logic that form the select 1 and select2 fields on Fig. 3. These bits will be referred to as *multiplexer control* bits. Finally, the 5 bits are shown as *sequencer control* bits on Fig. 2. These consist of an enable line for each stack, a push/$\overline{\text{pop}}$ line common to both stacks, and an enable and count/ $\overline{\text{load}}$ lines for the down counter. Referring to Table 2, the Xs represent bits that are "don't cares." For example, push/$\overline{\text{pop}}$ is a "don't care" if neither stack is enabled. Because the instruction bits will be stored in the microprogram memory, no advantage exists to considering them as "don't cares" and they simply can be set to 0. The value of C ranges from 1 to 7 depending on the condition input selected, and its binary value appears as C C C in the table. The encoding of select2 and selectl reflects the binary value of the input number shown on Fig. 3.

Figure 4 demonstrates how the microcode can be represented in a microassembly language format. The first field contains address labels for reference when performing branches and subroutine calls. The values in square brackets in the second field are the desired control outputs. The third field contains the instruction mnemonic, and the last field contains branch and subroutine addresses and loop counts. This example shows how to set up a nested loop and perform a conditional branch, a two-way branch, and a subroutine call. The nested loop will cause the innermost statements to execute 2500 times. The reader familiar with representing state machines in VHDL or Verilog will note that this microassembly language representation is more compact. However, it does have the disadvantage of requiring familiarity with a custom language. Conversion of microassembly language to microprogram memory contents is a simple matter of converting the instruction mnemonic to a bit pattern using Table 2 and entering the proper value for the branch address lines. A standard meta assembler with proper definition files can be used to accomplish this conversion with automatic determination of the correct address information from the label field.

PIPELINING

The maximum clock frequency of a microprogrammed control unit usually can be improved by modifying the sequencer, as shown in Fig. 5 to pipeline the microprogram memory address (8). Observe that the microprogram counter register has been moved to the output of the next address generation logic. In this way, the delay through the memory takes place in parallel with the delay through the next address logic. However, now a one clock cycle delay exists between when an address is generated

Table 2. Instruction encoding



	CC Select			Sequencer Control Bits					Multiplexer Control Bits			
Instruction Mnemonic	Select-2	Select-1	Select-0	Counter Stack Enable	Subroutine Stack Enable	Push/$\overline{\text{Pop}}$	Counter Enable	Count/$\overline{\text{Load}}$	Select2-1	Select2-0	Select1-1	Select1-0
CONT	X	X	X	0	0	X	0	X	1	1	1	1
LDCT	X	X	X	0	0	X	1	0	1	1	1	1
PSHLDCT	X	X	X	1	0	1	1	0	1	1	1	1
POPCT	X	X	X	1	0	0	1	0	1	1	1	1
LOOP	0	0	0	0	0	X	1	1	1	1	0	1
BR	X	X	X	0	0	X	0	X	0	1	0	1
CBF(C)	C	C	C	0	0	X	0	X	1	1	0	1
CBT(C)	C	C	C	0	0	X	0	X	0	1	1	1
TWB(C)	C	C	C	0	0	X	0	X	0	1	0	0
CALL	X	X	X	0	1	1	0	X	0	1	0	1
RET	X	X	X	0	1	0	0	X	1	0	1	0

and when the corresponding microinstruction appears at the output of the pipeline register. Therefore, when a branch instruction of any type takes place, the change in program flow is delayed a clock cycle. This delay means that whenever a branch appears in microcode, one additional instruction will always exist following the branch that is executed before the branch actually takes place. This delay is true of both conditional and unconditional branches, as well as loop instructions, subroutine calls, and subroutine returns. Such delayed branching is common in pipelined digital signal processors like the Texas Instruments TMS320C3X family where as many as three instructions may follow a branch (9). The use of pipelining is very beneficial if high clock rates are needed and most instructions execute sequentially, or if a productive operation can be placed in many microinstruction locations that follow branches. However, if many branches exist and nothing productive can be done in the microinstruction that follows most of them, then pipelining may not be beneficial. It is worth noting that in some FPGA architectures, the address lines to embedded memory are registered within the embedded memory block. In such cases, pipelining must be used and it may be necessary to take the memory address directly from the next address select logic to the embedded memory, with duplicate microprogram counter registers in the sequencer logic and in the embedded memory block.

```
                            ORG      O          :Start at address 0
begin      [100101]         CONT
           [111100]         LDCT     99         :Set up 100-pass loop
loop1      [100010]         PSHLDCT  24         :Set up 25-pass loop
loop2      [111111]         CBF(3)   cpass      :Skip next if 3 is false
           [001010]         CONT
cpass      [101110]         LOOP     loop2      :Inner loop
           [010010]         POPCT               :Restore outer loop count
           [101011]         LOOP     loop1      :Outer loop
           [101111]         CALL     subrtn1    :Expample subroutine call
           [111100]         LDCT     address1   :Set up two-may branch
           [110011]         TWB(5)   address2   :Two-way branch based on 5
                            .
                            .
                            .
subrtn1                     .                   :Subroutine entry point
                            .

           {111101}         RET                 :Subroutine exit
                            .
                            .
address1                    .
                            .
                            .
address2                    .

                            END
```

Figure 4. Microassembly code listing.

SUMMARY AND CONCLUSIONS

Microprogramming should be considered as a control technique for complex finite state machines where flexibility with fixed timing characteristics is important. Examples include computer control units, digital filters, hardware to compute fast Fourier transforms, disk controllers, and so forth. The key advantage to the microprogrammed

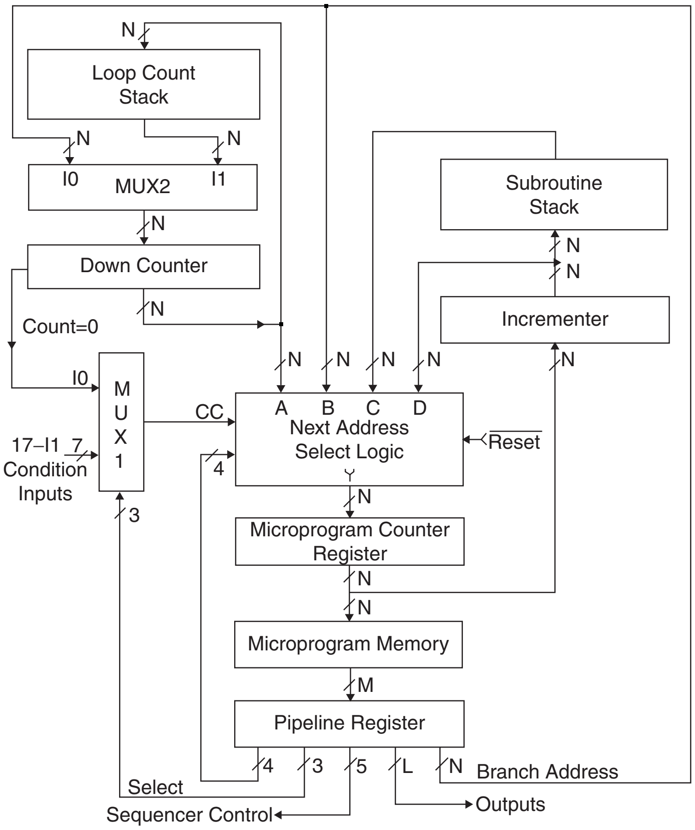


Figure 5. Pipelined microprogram control unit.

approach is that the state machine can adapt to changing algorithms by changing a bit pattern in memory that has no impact on logic resources or timing. An added benefit of the microprogrammed approach can be a more structured organization of the controller.

Although the microprogrammed approach has advantages over a traditional state machine described in a hardware description language, it has the disadvantage of requiring a custom microassembly language. However, this disadvantage may be offset for large state machines by the ability to do nested loops and subroutines, to take advantage of embedded memory blocks in FPGAs, and by the faster and more efficient design that can result. As FPGAs get larger and implement entire computing circuits for tasks like digital signal processing, the control requirement becomes correspondingly complex. In such applications, the microprogrammed approach provides an attractive solution.

BIBLIOGRAPHY

1. M. Wilkes, and C. Stringer, Micro-programming and the design of control circuits in an electronic digital computer, *Proc. Camb. Phil Soc*, Vol. **49**, 1953, pp. 230–238.
2. J. W. Carter, *Microprocessor Architecture and Microprogramming: A State-Machine Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.
3. E. L. Johnson, and M. A. Karim, *Digital Design*, Boston, MA: PWS-Kent, 1987, pp. 445–449.
4. B. Wilkinson, *Digital System Design*. Englewood Cliffs, NJ: Prentice Hall, 1987, pp. 413–423.
5. Advanced Micro Devices, Inc., in Chapter II - Microprogrammed Design Build a Microcomputer. Sunnyvale, CA: AMD Pub. AM-PUB073, 1978.
6. Advanced Micro Devices, Inc., *Am29PL141 Fuse Programmable Controller Handbook*. Sunnyvale, CA: AMD Pub. 06591A, 1986.
7. Altera, Inc., *Stand Alone Microsequencers: EPS444/EPS448*. Santa Clara, CA: Altera Pub. 118711 DFGG, 1987.
8. Bruce W. Bomar, Implementation of microprogrammed control in FPGAs, *IEEE Transactions on Industrial Electronics*, **42**, (2): pp. 415–421, 2002.
9. Texas Instruments, Inc., *TMS320C3X User's Guide*. Dallas, TX: Texas Instruments Pub. SPRU031E, 1997.

BRUCE W. BOMAR
The University of Tennessee
Space Institute
Tullahoma, Tennessee

PEN-BASED COMPUTING

INTRODUCTION

An Overview of Pen-Computing

Pen-based computing first came under the mainstream spotlight in the late 1980s when *GO Computers* developed the first computer operating system (OS) customized for pen/stylus input called *PenPoint,* which was used in the early tablet PCs by companies such as Apple Computer (cuperline CA) and IBM (Among NY). A pen-based computer replaces the keyboard and the mouse with a pen, with which the user writes, draws, and gestures on the screen that effectively becomes digital paper. The value proposition of pen-based computing is that it allows a user to leverage familiarity and skills already developed for using the pen and paper metaphor. Thus, pen-based computing is open to a wider range of people (essentially everybody that can read and write) than conventional keyboard and mouse-based systems, and is inline with the theme of Ubiquitous Computing, as such a computer is perceived as an electronic workbook, and thus provides a work environment resembling that which exists without computers. pen-based computers exist primarily in two forms, as mentioned above; tablet PCs, which often have a clip board-like profile and personal digital assistants (PDAs) that have a portable/handheld profile. Both forms (particularly the PDA) lend themselves very well to applications such as on site data entry/manipulation where the conventional approach is pen and paper-form based (1).

The Digitizing Tablet

The function of the digitizing tablet within a pen-based computer is to detect and to measure the position of the pen on the writing surface at its nominal sampling rate. Typically, this sampling rate varies between 50–200 Hz depending on the application, in which a higher sampling rate causes a finer resolution of cursor movement, and the computer can measure fast strokes accurately. The digitizing tablet is combined with the display to give the user the same high level of interactivity and short cognitive feedback time between their pen stroke/gesture and the corresponding digital ink mark. The user perceives the digitizing tablet and screen as one, which makes it a direct-manipulation input device and gives the user a similar experience to that of writing/drawing using the conventional pen and paper method. To enable this high level of interactivity, the digitizer must also operate with speed and precision. The display must be a flat panel display for the integrated unit to provide the user with the optimum writing surface. The display and digitizer can be combined in two ways. If the digitizer is mounted on top of the display (as illustrated below in Fig.1), the digitizing tablet must be transparent, although it can never be so infinitely, and thus the display's contrast is reduced and the user sees a blurred image.

The other way to combine the two is to mount the display on top of the digitizer; although it does not have to be transparent, the accuracy of the digitizer is reduced because of the greater distance between its surface and the tip of the pen when it is in contact with the writing surface.

The two types of digitizers are active and passive. Active digitizers are the most common type used in pen-based computers. These digitizers measure the position of the pen using an electromagnetic/RF signal. This signal is either transmitted by a two-dimensional grid of conducting wires or coils within the digitizer or transmitted by the pen.

The digitizer transmits the signal in two ways: it is either induced through a coil in the pen and conducted through a tether to the computer, or as is more commonly seen the pen, it reflects the signal back to the digitizer or disturbs the magnetic-field generated by the set of coils in the exact location of the pen tip. This reflection or disturbance is detected subsequently by the digitizer. Figure 2 depicts the latter configuration where a magnetic-field is transmitted and received by a set of coils, and this is disturbed by the inductance in the tip of a pen/stylus.

In this type of configuration, the horizontal and vertical position is represented in the original signal/magnetic-field transmitted by either signal strength, frequency, or timing, where each wire or coil in the grid carries a higher value than its neighboring counterpart. The position of the pen is evaluated using the values reflected back or disturbed.

The configuration where the pen reflects or disturbs the signal is found more commonly in modern day pen-based computers, as it allows for a lighter pen that is not attached to the computer, which provides the user with an experience closer to that of using conventional pen and paper. This method also allows more advanced information about the user's pen strokes to be measured, which can be used to provide more sophisticated and accurate handwriting recognition. The pressure being exerted on the screen/tablet can be measured using a capacitative probe within the tip of the pen, where its capacitance changes as it closes (as a result of being pressed against the screen/tablet), which changes the strength of the signal being reflected back. The angle of the pen can be measured using electronic switches that change the frequency of the signal reflected back, and these switches operate in a similar manner to tilt-switches. However, these advanced features require the pen to be powered by a battery or by the computer via a cord.

Typically, passive digitizers consist of two resistive sheets separated by a grid of dielectric spacers, with a voltage applied across one of the sheets in both the horizontal and vertical directions. When the pen makes contact with the screen/tablet, it connects the two sheets together at the point of impact, which acts to change the resistance (which is proportional to length of the resistive material) across the sheet in the two directions. In turn, it changes the two voltages across the sheet. essentially, these voltages represent a coordinate-pair of the pen's position.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

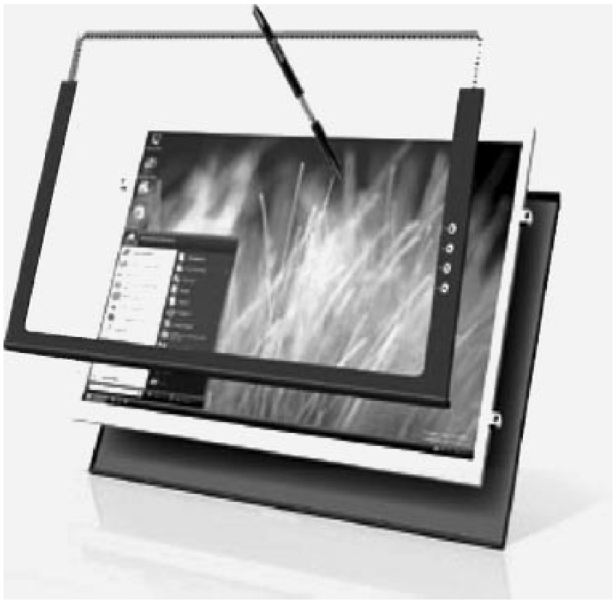


Figure 1. A digitizing tablet shown in the configuration where the digitizer is mounted on top of the screen (2) .

Being sensitive to pressure, passive digitizers also can receive input from the users fingers, and thus the computer can provide the user with a more natural/familiar mechanism for pressing on-screen (*virtual*) buttons. A disadvantage of this pressure-based operation is an increase in errors when using the pen, which are caused by the user resting against or holding the screen/tablet. However, passive digitizers can also be configured to be sensitive only to pen input by placing the dielectric spacers between the two resistive sheets closer together, where the higher amount of pressure needed to force the two sheets together can only be exerted through the small area of the pen tip.

Handwriting Recognition

Handwriting recognition (HWX) gives the user the ability to interface to a computer through the already familiar activity of handwriting, where the user writes on a digitizing tablet and the computer then converts this to text. Most users, especially those without prior knowledge of computing and the mainstream/conventional interface that is the keyboard and mouse, initially see this as a very intuitive and attractive human-computer interface as it allows them to leverage a skill that has already been acquired and developed. Although a major disadvantage of pen-based computers is that current HWX methods are not completely accurate. Characters can be recognized incorrectly and these recognition errors must be corrected subsequently by the user. Another aspect of current HWX technology that impacts the user's productivity and experience in a negative way is the fact that characters have to be entered one at a time, as the technology required to support the recognition of cursive handwriting is still a long way off and possibly requires a new programming paradigm (4). Handwriting of individuals varies immensely, and an individual's handwriting tends to change over time in both the short-term and the long-term. This change is apparent in how people tend to shape and draw the same letter differently depending on where it is written within a word what the preceding and subsequent letters are, all of which complicates the HWX process.

The two types of handwriting recognition are *on line* (real-time) and *Off line* (performed on stored handwritten data). When using conventional pen and paper, the user sees the ink appear on the paper instantaneously; thus, on line recognition is employed in pen-based computers for the user to be presented with the text form of their handwriting immediately. The most common method for implementing a HWX engine (and recognition engines in general) is a neural network, which excel in classifying input data into one of saveral categories. Neural networks are also ideal for HWX as they cope very well with noisy input data. This quality is required as each user writes each letter slightly different each time, and of course altogether differently from other users as explained previously. Another asset of neural networks useful in the HWX setting is their ability to learn over time through back-propagation, where each time the user rejects a result (letter) by correcting it,

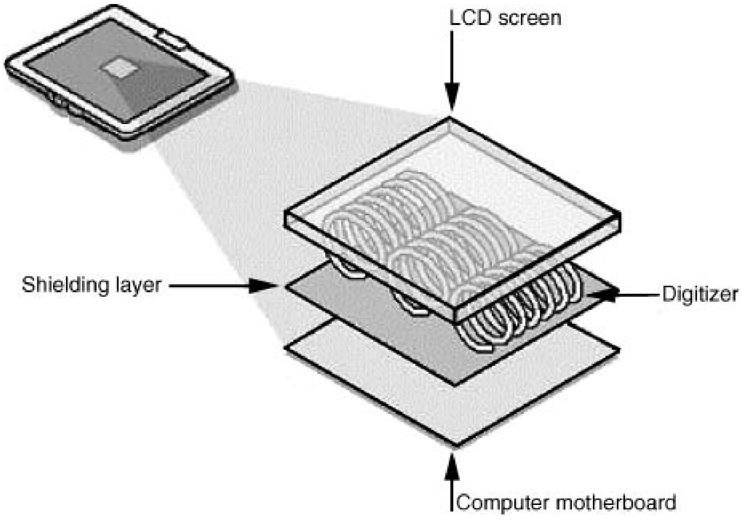


Figure 2. A digitizing tablet shown in the configuration where a set of colls generated a magenetic-field which is disturbed by the presence of the pen tip in that exact location (3).

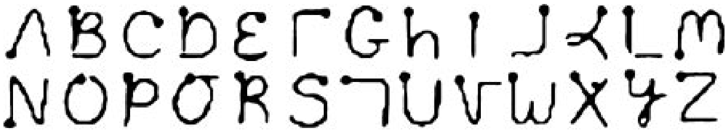


Figure 3. The Graffiti character set developed by palm computing. The dot on each character shows its starting point.

the neural network adjusts its weights such that the probability of the HWX engine making the same error again is reduced. Although this methodology can be used to reduce the probability of recognition errors for a particular use. When met with another user, the probability of a recognition error for each letter may have increased compared to what it would have been before the HWX engine was trained for recognizing the handwriting of a specific user. These problems are overcome to a good extent by using a character set where all the letters consist of a single stroke and their form is designed for ease of recognition (i.e., a specific HWX engine has been pre trained for use with that particular character set). One such character set is shown in Fig 3.

As can be seen in Fig. 3, the characters are chosen to resemble their alphabetic equivalents as much as possible. The use of such a character set forces different users to write largely the same way and to not form their own writing style, which is the tendency when using more than one stroke per letter.

The User Interface

The overall objective of a user interface that supports handwriting recognition is to provide the user with virtually the same experience as using conventional pen as paper. Thus, most of its requirements can be derived from the pen and paper interface. This means that ideally, no constraints should as to where on the digitizing tablet/screen (paper) the user writes, the size of their writing, or even when they write and when online recognition can take place. Obviously, conventional pen and paper does not impose any restrictions on special characters with accents, so ideally the user could write such characters and they should be recognized and converted to text form. Although even in pen-based computing, the standard character set used is ASCII as opposed to Unicode, which is the standardized character set that contains all characters of every language throughout the world.

As the pen is the sole input device, it is used for all the user's interfacing action. As well as text entry, it is also used for operations that a conventional mouse would be used for, such as selecting menu options, clicking on icons, and so. The action of moving the mouse cursor over a specific icon/menu-option and clicking on it is replaced entirely with simply tapping the pen on the item you would have clicked on. A dragging operation is performed by tapping the pen on the item to be dragged twice in quick succession (corresponding to the double clicking associated with the mouse), keeping the pen in contact with the tablet after the second tap and then dragging the selected item, and finally lifting the pen off the tablet to drop the item. A pen-based computer has no function keys or control keys (such as return, space-bar, etc.) like a keyboard does, and as such a pen-based computer's user interface must support something called *gesture recognition*, where the functions and the operations associated with these keyboard keys are activated by the user drawing a gesture corresponding to that function/operation. Typical examples common among most pen-based computing user interfaces are crossing a word out to delete it, circling a word to select it for editing, and drawing a horizontal line to insert a space within text.

The user will enter textual, as well as gesture and graphic (drawing/sketching) input, and thus the user interface is required to distinguish between these forms of input. The user interface of some pen-based computing OS, especially those that are extensions of conventional OS, separate the different types of input by forcing the user to set the appropriate mode, for example pressing an on-screen button to enter the drawing/sketching mode. This technique is uncomfortable for the user as it detracts from the conventional pen and paper experience, and so the preferred method is for the OS to use semantic context information to separate the input. For example, if the user is in the middle of writing a word and they write the character ‘O’, the OS would consider the fact that they were writing a word and so would not confuse the character ‘O’ with a circle or the number zero. This latter method is typical of OS written specifically for pen-based computing (Pen-centric OS).

The causes of recognition errors fall into two main categories. One is where the errors are caused by indistinguishable pairs of characters, (for example the inability of the recognizer to distinguish between ‘2’ and ‘Z’). The best solution in this case is for the OS to make use of semantic context information. The other main source of errors is when some of the user's character forms are unrecognizable. As explained previously, this situation can be improved in two ways. One is for the user to adapt their handwriting style, so that their characters are a closer match to the recognizer's pre-stored models; the other way is to adapt the recognizer's pre-stored models; to be a closer match with the user's character forms (trained recognition) (5).

An Experiment Investigating the Dependencies of User Satisfaction with HWX Performance

A joint experiment carried out by Hewlett Packard (palo Alto, CA) and The University of Bristol Research. Laboratories (Bnstd, UK) in 1995 investigated the influence of HWX performance on user satisfaction (5). Twenty-four subjects with no prior knowledge of pen-based computing carried out predetermined tasks using three test applications as well as text copying tasks, after being given a brief tutorial on pen-based computing. The applications were run on an IBM-compatible PC with Microsoft Windows for Pen Software and using a Wacom digitizing tablet (which

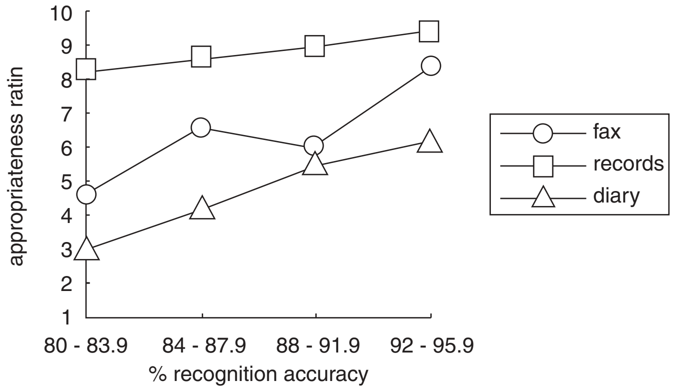


Figure 4. Plots of appropriateness rating against recognition accuracy .

did not present a direct-manipulation input device as it was not integrated with the screen). The three applications were Fax/Memo, Records, and Diary, and they were devised to contrast with each other in the amount of HWX required for task completion, tolerance to erroneous HWX text, and the balance between use of the pen for text entry and other functions performed normally using a mouse. The mean recognition rate for lowercase letters was found to be 90.9%, and 76.1% for upper case letters, with the lower recognition rate for uppercase letters caused mainly by identical upper and lower case forms with letters such as 'C', 'O', 'S', 'V'. Pen-based computing OS' attempt to deal with this problem by comparing the size of drawn letters relative to each other or relative to comb-guides when the user input is confined to certain fields/boxes.

As can be observed Fig. 4, the application that required the least amount of text recognition (Records) was rated as most appropriate for use with pen input, and the application with the most amount of text recognition (Diary) was rated as the least appropriate in this respect. Figure 4 also shows that higher recognition accuracy is met with a higher appropriateness rating by the user, and the more dependent an application is on text recognition the stronger this relationship is.

An indication of this last point can be observed from the plots shown in Fig. 4, as the average gradient of an application's plot increases as it becomes more dependent on text recognition. The results shown in fig. 4 also suggest that the pen interface is most effective in performing the non textual functions associated normally with the mouse. Thus, improving recognition accuracy would increase the diversity of applications in pen-based computing, as those more dependent on text entry would be made more effective and viable.

COGNITIVE MODELLING

Introduction to Cognitive Models

Cognitive models as applied to human–computer interaction represent the mental processes that occur in the mind of the user as they perform tasks by way of interacting with a user interface. A range of cognitive models exits, and each models a specific level within the user's goal-hierarchy from high-level goal and task analysis to low-level analysis of motor-level (physical) activity. cognitive models fall into two broad categories: those that address how a user acquires or formulates a plan of activity and those that address how a user executes that plan. Considering applications that support pen input, whether they are extensions of conventional applications that support only keyboard and mouse input or special pen-centric applications, the actual tasks and associated subtasks that need to be performed are often the same as those in conventional applications. Only the task execution differs because of the different user interface. Thus, only cognitive models that address the user's execution of a plan once they have acquired/formulated it shall be considered, as a means of evaluating the pen interface.

Adaptation of the Keystroke-Level Model for Pen-based Computers

The keystroke-level model (KLM) (6) is detailed in Table 1. This model was developed and validated by Card, Newell, and Moran, and is used to make detailed predictions about user performance with a keyboard and mouse interface in terms of execution times. It is aimed at simple command sequences and low-level unit tasks within the interaction hierarchy and is regarded widely as a standard in the field of human-computer interaction.

KLM assumes a user first builds up a mental representation of a task in working out deciding exactly how they will accomplish the task using the facilities and functionality offered by the system. This assumption means that no high-level mental activity is considered during the second phase of completing a task, which is the actual execution of the plan acquired and is this execution phase that KLM focuses on. KLM consists of seven operators, five *physical motor* operators, a *mental* operator, and a system response operator. The KLM model of the task execution by a user consists of interleaved instances of the operators. Table 2 details the penstroke-level model (PLM), which represents a corresponding set of operators for pen-based systems.

As stated previously, the actual tasks that need to be performed to achieve specific goals are largely the same with pen-based and keyboard and mouse-based systems,

Table 1. A description of the KLM model's seven operators

Operator	Description
K	Key stroking, actually striking keys, including shifts and other modifier keys
B	Pressing a mouse button
P	Pointing, moving the mouse (or similar device) as a target
H	Homing, switching the hand between mouse and keyboard
D	Drawing lines using the mouse
M	Mentally preparing for physical action
R	System response which may be ignored if the user does not have to wait for it, as in copy typing

Table 2. The PLM model, a set of operators for a pen-based user interface corresponding to those of the KLM model

Operator	Description
K'	Striking a key or any combination of keys is replaced in pen-based systems with writing a single character or drawing a single gesture. It is widely accepted in the field of HCI that a reasonably good typist can type faster than they can write. Although holding down a combination of keys is effectively one individual key press for each key, as the position of each key is stored as a one *chunk* in human memory. Some keys are not as familiar to a user as the character keys, as they are used less frequently. Thus if the user has to look for a key (e.g. the shift key) then it is reasonable to assume that the action of drawing a single gesture would be of a comparable speed.
B'	This operation is replaced in pen-based systems with tapping the pen once on the screen, which is essentially the same action as marking a full-stop/period or doting an 'i' or 'j', and it is a better developed motor skill. No reason exists to believe or evidence either way to suggest that B or B' is faster than the other for a specific user.
P'	This operation is replaced in pen-based systems with the action of moving the pen over the screen, but the pen does not have to be in continuous contact with the digitizing tablet. Thus, the cursor can be moved from one side of the screen to the other simply by lifting it from one side and placing it on the other side. This action makes the pen interface more ergonomic than the mouse in this type of situation as the user's hand need no longer be in a state of tension while performing this action. This action is a prime example of the benefits of a direct-manipulation interface.
H'	The homing operator is one for which no corresponding operator exists for pen-based systems, as the pen is the sole input device. This decreases the overall execution time when using a pen interface compared with a keyboard and mouse.
D'	This operation is replaced in pen-based systems with the action of drawing with the pen, where again the benefits of direct manipulation are observed, as it is just like drawing using pen and paper, which is a much better developed set of motor-skills, especially when drawing curved lines/strokes that compose a sketch/drawing.
M'	Because the KLM model assumes the user has formulated a plan and worked out how to execute it, the mental preparation modelled by the M and M'-operator is simply the time taken by the user to recall what to do next. Thus it is reasonable to assume that the time taken up by an occurrence of an M or M'-operator is the same for pen-based systems as it is for keyboard and mouse-based systems for each user. Although it is reasonable to assume that if one was the slower of the two it would be M, because with keyboard and mouse-based systems, the user has to recall which input device (keyboard or mouse) they need to use.
R'	Assuming the two systems were of a similar overall capability, it is reasonable to assume that the system response time for a pen-based system and a keyboard and mouse-based system would be the same. However, the process of HWX is more intensive computationally than reading keyboard input, so a pen-based system's processor would need to be faster than that of a keyboard and mouse-based system to be perceived by the user as being of comparable speed.

and I have thus adapted the KLM model into the PLM model for application to pen-based systems.

From Table 2, it can be reserved that at worst, pen-based systems have one less physical motor operator than keyboard and mouse-based systems, and at best two of its four physical motor operators have quicker execution times compared with the corresponding operators of keyboard and mouse-based systems for each user. This abservation suggests that pen-based systems are faster to use than keyboard and mouse-based systems, and the previous discussion would suggest that this is especially true for applications that contain many pointing and dragging tasks.

Experiment to Compare the Performance of the Mouse, Stylus and Tablet and Trackball in Pointing and Dragging Tasks

An experiment by Buxton, et al. in 1991 (7) compared the performance of the mouse, stylus (and digitizing tablet), and the trackball (which is essentially an upside-down mouse, with a button by the ball) in elemental pointing and dragging tasks. Performance was measured in terms of mean execution times of identical elemental tasks. However, the digitizing tablet was used in a form where it was not integrated with the display and sat on the users desktop, and thus it could not be considered a direct-manipulation interface as is the case when it is integrated with the screen. The participants of the experiment were 12 paid volunteers who were computer-literate college students. During both the pointing and the dragging task, two targets were on either side of the screen as shown in Fig. 5.

In the experiment, an elemental pointing action was considered to be moving the cursor over a target and then clicking the mouse/trackball button or pressing the pen down onto the tablet to close its tip-switch, (as opposed to the tapping action with the modern stylus and digitizer combinations which terminated an action and initiated the next action. An elemental dragging action was considered to be selecting an object within one target and holding down the mouse/trackball button or maintaining the pressure of the stylus on the tablet, dragging it to within the other target and then releasing the mouse/trackball button or pressure on the tablet to drop the object in that target, which terminated an action and initiated the next action where each time the new object would appear halfway between the two targets. Fitts' law provides a formula [shown below in its most common form in Equation (1) to

Figure 5. The on-screen user-interface used for dragging tasks.

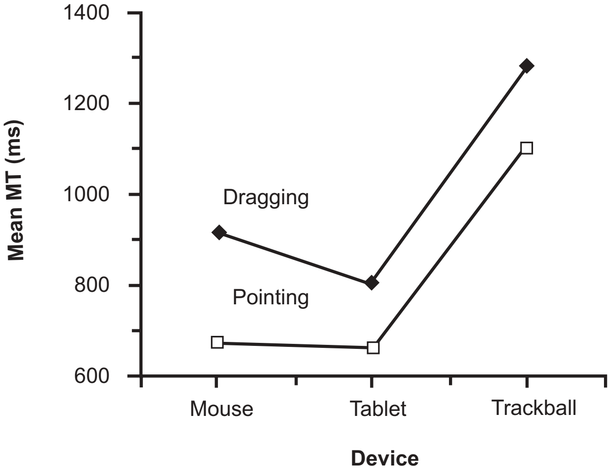


Figure 6. Graph showing execution times of three devices for elemental pointing and dragging tasks.

calculate the time taken for a specific user to move the cursor to an on-screen target.

$$\text{Movement\_Time} = a + b\log2(\text{Distance}/(\text{Size} + 1)) \tag{1}$$

As can be observed from Equation (1), according to Fitts' law the time taken to move to the target depends on the distance the cursor needs to be moved and the size of the target. Constants a and b are determinable empirically for each user. As Equation (1) shows, a greater distance is moved in a greater time, and a smaller target is more difficult to acquire and thus also increases movement time. The distance between the targets shown in Fig. 5 was varied over the range of discrete values (A = 8, 16, 32, 64 units) where a unit refers to eight pixels. The size parameter of Equation (1) was represented by the width of the targets, as the movement of the cursor would largely be side-to-side, and this too was varied over the discrete range (W = 1, 2, 4, 8 units). All values of the distance between targets A were fully crossed with all values of the width of the targets W for both the pointing and the dragging tasks, and each A-W combination was used for a block of 10 elemental tasks (pointing or dragging), where the user's objective was to carry out the ten tasks in succession as quickly and a accurately as possible. Sixteen blocks were ordered randomly into a session, and five sessions were completed for each device for each of the two types of tasks. The results showed that subjects occasionally would drop the object a long way from the target. This was not because of normal motor variability but occurred because of the difficulty in sustaining the tension in the hand required to perform dragging. It was particularly evident with the trackball, where the ball has to be rolled with the fingers while holding the button down with the thumb. The results were adjusted to remove these errors by eliminating elemental task executions within each block that were terminated (by a click or release) a horizontal distance from the mean termination distance greater than three standard deviations. This adjustment was made separately for each subject, A-W combination, device, and task type. Elemental task executions immediately after those that were terminated erroneously (which the user was notified of via a beep) were also eliminated, as many people who had investigated repetitive, self paced, and serial tasks concluded that erroneous executions were disruptive to the user and could cause an abnormally long response time for the following trial, which would have skewed the average execution time. Analysis also showed a significant reduction in execution times after the first session, and thus the entire first session for each subject, for each device task type combination was also eliminated. Figure 6 and Table 3 show the mean movement times (execution times) over all blocks for each device and for the two task types, after the adjustments mentioned above were made.

Table 3. Tabulated form of the plots shown in Figure 6

Input device	Average MT: elemental pointing	Average MT: elemental dragging
Stylus & Tablet	665 ms	802 ms
Mouse	674 ms	916 ms
Trackball	1101 ms	1284 ms

As can be seen in Fig. 6 and in Table 3, the pen and digitizing tablet was fastest in both task types, although the performance of the mouse was comparable in the pointing task. The results show that the performance of each device was better in the pointing task than in the dragging task. This seems reasonable as when dragging the hand (and the forearm in the case of the pen and digitizing tablet) is in a state of greater tension compared with when pointing. The big difference in performance between the mouse and the pen and digitizing tablet was in the dragging task, where the performance gap for the pointing task was the greatest with the mouse. Error rates for each device-task type were also evaluated and are shown in Fig. 7.

As with mean movement time, error rates were worse for the dragging task than they were for pointing. The unadjusted results were evaluated before making the modifications described above. The adjustment of eliminating errors greater than three standard deviations from the mean

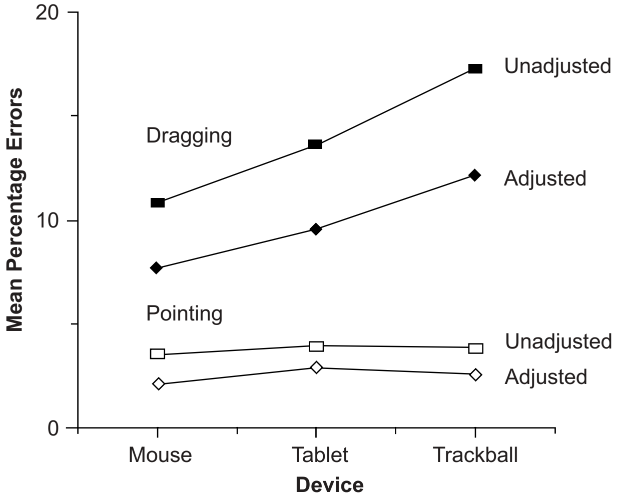


Figure 7. Graphs to show the mean error rates for three devices during pointing and dragging tasks.

termination distance from the target (as described above) was also applied to the pointing task, although dropping errors could not have occurred during the pointing task. The results shown in Fig. 7 show that the mouse had a lower (but comparable) error rate compared with the pen and digitizing tablet in the pointing task, and had a much lower error rate than in the dragging task. The 12 participants were computer literate, and because the mouse is the standard interface device for pointing and dragging, this finding suggests that on average they would have been more familiar with the mouse than they were with the pen and digitizing tablet (in the non–direct-manipulation form that was employed in this experiment). Thus, they would have had better developed motor skills for pointing and dragging using a mouse. It is reasonable to assume that the pen and digitizing tablet interface in its direct-manipulation form would have yielded fewer errors all round, but almost certainly in pointing as the user would no longer have to track the position of the cursor between the targets. The user could simply perform a dotting action on the targets in an alternate fashion, which as well as eliminating errors would have boosted the speed of elemental pointing tasks.

CONCLUSIONS

The results of the experiment conducted by Hewlett Packard and The University of Bristol research labs (5) discussed previously suggest that the pen interface is very effective for pointing and dragging tasks. Although subject feedback gave applications a lower appropriateness rating (for use with pen input), its handwriting recognition functionality was more significant and vital in task execution. Overall, the results of the experiment conveyed the impression that the pen was comparable with the mouse for pointing and dragging tasks, but not as good as the keyboard for text entry because of relatively high recognition error rates.

In my attempt at a direct comparison of pen-based and keyboard and mouse-based systems it was shown that pen-based systems are the simpler of the two in a cognitive sense, because the overhead associated with switching the hand between the mouse and the keyboard is removed with pen-based systems as the pen is the sole input device. The other operators in the model most likely have quicker execution times than their KLM equivalents for a specific user as a result of the direct-manipulation input device that is the integrated digitizer and screen and because of the more advanced motor skills that most users will have for using a pen than for using a mouse. The only exception to this is the B or B'-operator (pressing a mouse button/ tapping the pen on the digitizing tablet), with which no evidence suggests which is faster than the other, but usually this operator is combined with the P or P'-operator (pointing), and reasons exist to believe this method is faster using pen-based systems than keyboard and mouse-based systems.

The results of the experiment conducted by Buxton et al. (7) discussed previously support these facts by showing the pen to be faster than the mouse (and the trackball) for both pointing and dragging tasks. Although the results showed higher error rates during both pointing and dragging for the pen than for the mouse. It is reasonable to assume that this was because the experiment used the indirect-manipulation form of the digitizing tablet (where it is not integrated with the screen). With this configuration of the digitizing tablet the user is confined to using a low interactivity-level input mechanism just as they are when using a mouse, but with a less familiar input device as this configuration of the digitizing tablet does not resemble the conventional pen and paper metaphor like the direct-manipulation configuration does.

BIBLIOGRAPHY

1. PDA vs. Laptop: a comparison of two versions of a nursing documentation application. Center for Computer Research Development, Puerto Rico University, Mayaguez, Puerto Rico.
2. N-trig http://www.n-trig.com.
3. http://msdn2.microsoft.com/en-us/library/ms811395.aspx.
4. Multimodal Integration for Advanced Multimedia Interfaces ESPRIT III Basic Research Project 8579. Avalable: http://hwr.nici.kun.nl/~miami/taxonomy/node1.html.
5. Recognition accuracy and user acceptance of pen interfaces. University of Bristol Research Laboratories; Hewlett Packard CHI '95 Proceedings and Papers. Avalable: http://www.acm.org/sigs/sigchi/chi95/Electronic/documnts/.
6. A. Dix, J. Finlay, G. Abowd, R. Beale*"Humancomputer Interaction 2nd ed."* Englewood.Cliffs, NJ: 1998.
7. A Comparison of Input Devices in Elemental Pointing and Dragging Tasks. Avalable: http://www.billbuxton.com/fitts91.html.

DR. SANDIP JASSAR
Cambridge, United Kingdom

PROGRAMMABLE LOGIC ARRAYS

INTRODUCTION

Programmable logic arrays (PLAs) are widely used traditional digital electronic devices. The term "digital" is derived from the way digital systems process information; that is by representing information in digits and operating on them. Over the years, digital electronic systems have progressed from vacuum-tube circuits to complex integrated circuits, some of which contain millions of transistors. Currently, digital systems are included in a wide range of areas, such as communication systems, military systems, medical systems, industrial control systems, and consumer electronics.

Electronic circuits can be separated into two groups, digital and analog. Analog circuits operate on analog quantities that are continuous in value, where as digital circuits operate on digital quantities that are discrete in value and are limited in precision. Analog signals are continuous in time and are continuous in value. Most measurable quantities in nature are in analog form, for example, temperature. The measurements of temperature changes are continuous in value and in time; the temperature can take any value at any instance of time without a limit on precision but with the capability of the measurement tool. Fixing the measurement of temperature to one reading per an interval of time and rounding the value recorded to the nearest integer will graph discrete values at discrete intervals of time that could be coded into digital quantities easily. From the given example, it is clear that an analog-by-nature quantity could be converted to digital by taking discrete-valued samples at discrete intervals of time and then coding each sample. The process of conversion usually is known as analog-to-digital conversion (A/D). The opposite scenario of conversion is also valid and known as digital-to-analog conversion (D/A). The representation of information in a digital form has many advantages over analog representation in electronic systems. Digital data that is discrete in value, discrete in time, and limited in precision could be efficiently stored, processed, and transmitted. Digital systems are more noise-immune than analog electronic systems because of the physical nature of analog signals. Accordingly, digital systems are more reliable than their analog counterpart. Examples of analog and digital systems are shown in Fig. 1.

A BRIDGE BETWEEN LOGIC AND CIRCUITS

Digital electronic systems represent information in digits. The digits used in digital systems are the *0* and *1* that belong to the *binary* mathematical number system. In logic, the *0* and *1* values correspond to *true* and *false*. In circuits, the *true* and *false* correspond with *high* voltage and *low* voltage. These correspondences set the relations among logic (*true* and *false*), *binary* mathematics (*0* and *1*), and circuits (*high* and *low*).

Logic, in its basic shape, checks the validity of a certain proposition — a proposition could be either *true* or *false*. The relation among logic, *binary* mathematics, and circuits enables a smooth transition of processes expressed in propositional logic to *binary* mathematical functions and equations (*Boolean* algebra) and to digital circuits. A great scientific wealth exists to support strongly the relations among the three different branches of science that led to the foundation of modern digital hardware and logic design.

Boolean algebra uses three basic logic operations *AND*, *OR*, and *NOT*. If joined with a proposition *P*, the *NOT* operation works by negating it; for instance, if *P* is *True* then *NOT P* is *False* and vice versa. The operations *AND* and *OR* should be used with two propositions, for example, *P* and *Q*. The logic operation *AND*, if applied on *P* and *Q* would mean that *P AND Q* is *True* only when both *P* and *Q* are *True*. Similarly, the logic operation *OR*, if applied on *P* and *Q* would mean that *P OR Q* is *True* when either *P* or *Q* is *True*. Truth tables of the logic operators *AND*, *OR*, and *NOT* are shown in Fig. 2a. Figure 2b shows an alternative representation of the truth tables of *AND*, *OR*, and *NOT* in terms of *0s* and *1s*.

Digital circuits implement the logic operations *AND*, *OR*, and *NOT* as hardware elements called "gates" that perform logic operations on binary inputs. The *AND*-gate performs an *AND* operation, an *OR*-gate performs an *OR* operation, and an *Inverter* performs the negation operation *NOT*. Figure 2c shows the standard logic symbols for the three basic operations. With analogy from electric circuits, the functionality of the *AND* and *OR* gates are captured as shown in Fig. 3. The actual internal circuitry of gates is built using transistors; two different circuit implementations of inverters are shown in Fig. 4. Examples of *AND*, *OR*, *NOT* gates integrated circuits (ICs) are shown in Fig. 5. Besides the three essential logic operations, four other important operations exist—the *NOR*, *NAND*, exclusive-*OR* (*XOR*), and Exclusive-*NOR*.

A logic circuit usually is created by combining gates together to implement a certain logic function. A logic function could be a combination of logic variables (such as A, B, C, etc.) with logic operations; logic variables can take only the values *0* or *1*. The created circuit could be implemented using *AND-OR-Inverter* gate-structure or using other types of gates. Figure 6 shows an example combinational implementation of the following logic function *F(A, B, C)*:

$$F(A,B,C) = ABC + A'BC + AB'C'$$

In this case, *F(A, B, C)* could be described as a sum-of-products (SOP) function according to the analogy that exists between *OR* and addition (+), and between *AND*

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

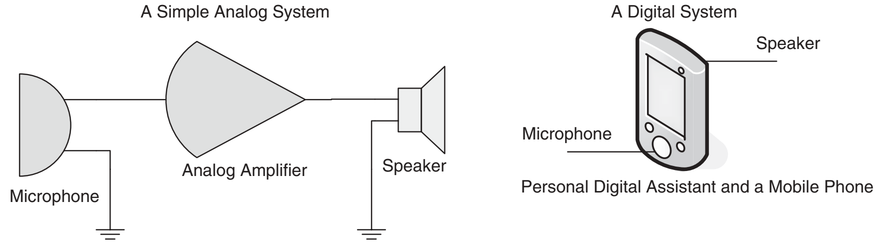


Figure 1. A simple analog system and a digital system; the analog signal amplifies the input signal using analog electronic components. The digital system can still include analog components like a speaker and a microphone, the internal processing is digital.

Input *X*	Input *Y*	Output: *X AND Y*
False	False	False
False	True	False
True	False	False
True	True	True

Input *X*	Input *Y*	Output: *X OR Y*
False	False	False
False	True	True
True	False	True
True	True	True

Input *X*	Output: *NOT X*
False	True
True	False

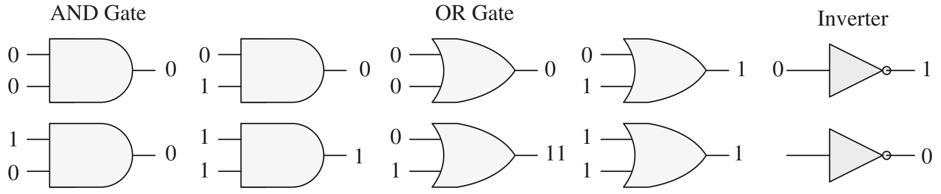
(a)

Input *X*	Input *Y*	Output: *X AND Y*
0	0	0
0	1	0
1	0	0
1	1	1

Input *X*	Input *Y*	Output: *X OR Y*
0	0	0
0	1	1
1	0	1
1	1	1

Input *X*	Output: *NOT X*
0	1
1	0

(b)



(c)

Figure 2. (a) Truthtables for AND, OR, and Inverter. (b) Truthtables for AND, OR, and Inverter in binary numbers, (c) Symbols for AND, OR, and Inverter with their operation.

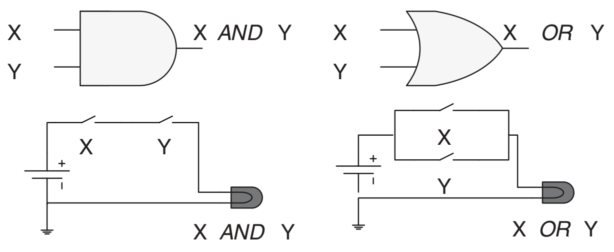


Figure 3. A suggested analogy between AND and OR gates and electric circuits.

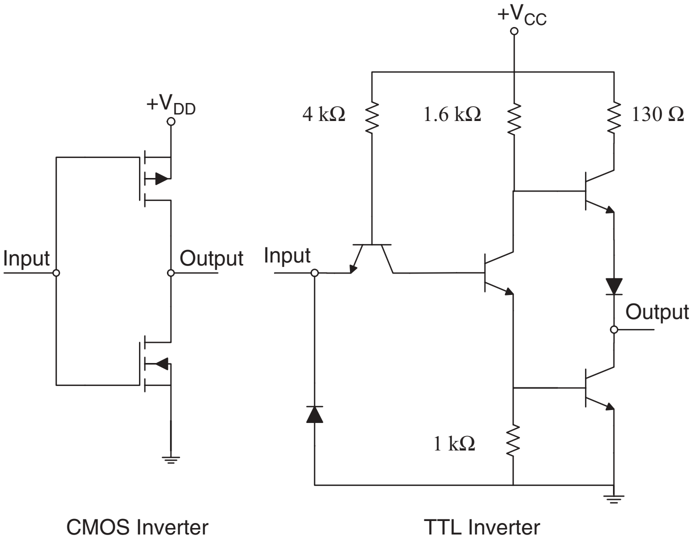


Figure 4. Complementary Metal-oxide Semiconductor (CMOS) and Transistor-Transistor Logic (TTL) Inverters.

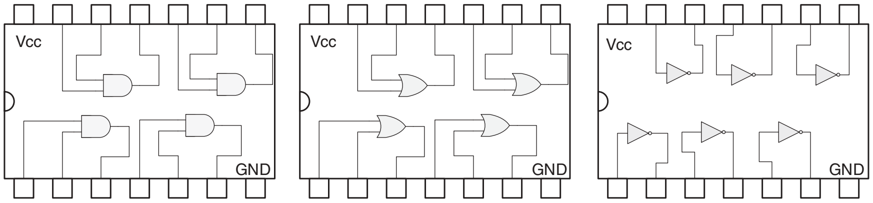


Figure 5. The 74LS21 (AND), 74LS32 (OR), 74LS04 (Inverter) TTL ICs.

and product (.); the *NOT* operation is indicated by an apostrophe " ' " that follows the variable name.

PROGRAMMABLE LOGIC

Basically, three types of IC technologies exist that can be used to implement logic functions on which Ref. 1, include, full-custom, semi-custom, and programmable logic devices (PLDs). In full-custom implementations, the designer is concerned about the realization of the desired logic function to the deepest details, which include the transistor-level optimizations, to produce a high-performance implementation. In semi-custom implementations, the designer uses ready logic-circuit blocks and completes the wiring to achieve an acceptable performance implementation in a shorter time than full-custom procedures. In PLDs, the logic blocks and the wiring are ready. To implement a function on a PLD, the designer will decide which wires and blocks to use; this step usually is referred to as programming the device.

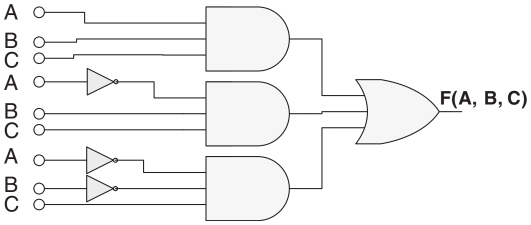


Figure 6. AND-OR-Inverter implementation of the function $F(A, B, C) = ABC + A'BC + AB'C'$.

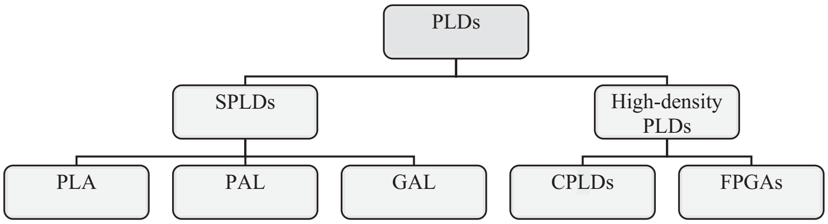


Figure 7. Typical PLD device classification.

Obviously, the development time using a PLD is shorter than the other full-custom and semi-custom implementation options. The performance of a PLD varies according to its type and complexity; however a full-custom circuit is optimized to achieve higher performance. The key advantage of modern programmable devices is their reconfiguration without rewiring or replacing components (reprogrammability). Programming a modern PLD is as easy as writing a software program in a high-level programming language.

The first programmable device that achieved a widespread use was the programmable read-only memory (PROM) and its derivatives Mask-PROM, and Field-PROM (the erasable or electrically erasable versions). Another step forward led to the development of PLDs. Programmable array logic, programmable logic array (PLA), and generic array logic are commonly used *PLDs* designed for small logic circuits and are referred to as simple-PLDs (SPLDs). Other types, Such as the mask-programmable gate arrays, were developed to handle larger logic circuits. Complex-PLDs (CPLDs) and field programmable gate arrays (FPGAs) are more complicated devices that are fully programmable and instantaneously customizable. Moreover, FPGAs and CPLDs have the ability to implement very complex computations with millions of gates devices currently in production. A classification of PLDs is shown in Fig. 7.

PLAs

A (PLA) is an (SPLD) that is used to implement combinational logic circuits. A PLA has a set of programmable *AND* gates, which link to a set of programmable *OR* gates to produce an output (see Fig. 8). Implementing a certain function using a PLA requires the determination of which connections among wires to keep. The unwanted routes could be eliminated by burning the switching device (possibly a fuse or an antifuse) that connects different routs. The *AND-OR* layout of a *PLA* allows logic functions to be implemented that are in an SOP form.

Technologies used to implement programmability in PLAs include fuses or antifuses. A fuse is a low resistive element that could be blown (programmed) to result in an open circuit or high impedance. An antifuse is a high

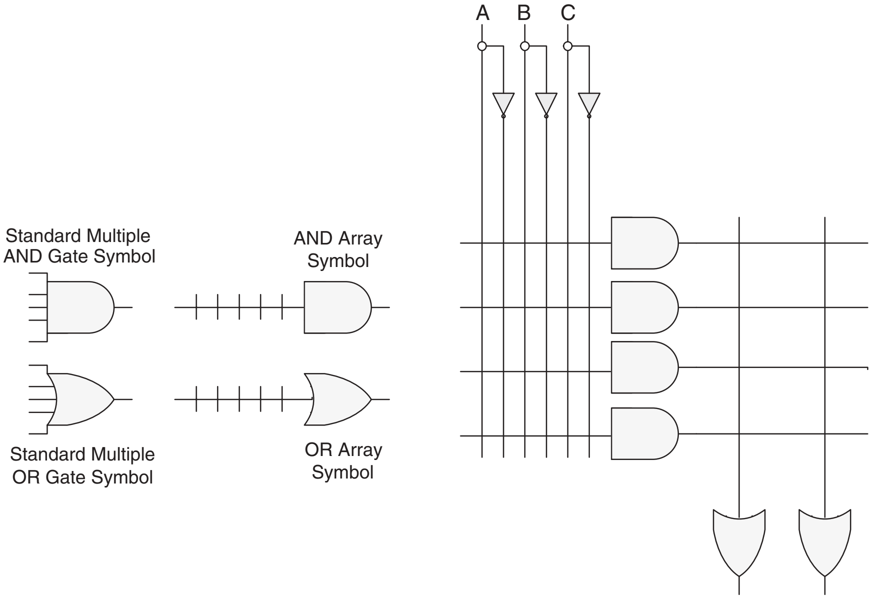

A
B
C
Standard Multiple AND Gate Symbol
AND Array Symbol
Standard Multiple OR Gate Symbol
OR Array Symbol

Figure 8. A 3-input 2-output PLA with its AND Arrays and OR Arrays. An AND array is equivalent to a standard multiple-input AND gate, and an OR array is equivalent to a standard multiple-input OR gate.

resistive element (initially high impedance) and is programmed to be low impedance.

Boolean expressions can be represented in either of two standard forms, *SOPs* and the product-of-sums (POSs). For example, the equations for *F* (an SOP) and *G* (a POS) are as follows:

$$F(A,B,C) = ABC + A'BC + AB'C'$$
$$G(A,B,C) = (A+B+C).(A'+B+C).(A+B'+C')$$

A product term consists of the *AND* (Boolean multiplication) of literals (*A*, *B*, *C*, etc.). When two or more product terms are summed using an *OR* (*Boolean* addition), the resulting expression is an SOP. A standard SOP expression $F(A,B,C,\ldots)$ includes all variables in each product term. Standardizing expressions makes evaluation, simplification, and implementation much easier and systematic.

The implementation of any *SOP* expression using *AND*-gates, *OR*-gates, and inverters, could be replaced easily using the structure offered by a PLA. The algebraic rules of hardware development using standard SOP forms are the theoretical basis for designs targeting PLAs. The design procedure simply starts by writing the desired function in an SOP form, and then the implementation works by choosing which fuses to burn in a fused-PLA.

In the following two examples, we demonstrate the design and the implementation of logic functions using PLA structures. In the first example, we consider the design and the implementation of a three-variable majority function. The function *F(A, B, C)* will return a *1* (*high* or *true*) when the number of *1s* in the inputs is greater than or equal to the number of *0s*. The truth-table of *F* is shown in Fig. 9. The terms that make the function *F* return a *1* are the terms *F(0, 1, 1)*, *F(1, 0, 1)*, *F(1, 1, 0)*, or *F(1, 1, 1)*. which could be formulated alternatively as in the following equation:

$$F = A'BC + AB'C + ABC' + ABC$$

In Fig. 10, the implementations using a standard *AND-OR-Inverter* gate-structure and a PLA are shown.

Another function G(A, B, C, D) could have the following equation:

$$G = A'B + AB'CD + AB' + ABD + B'C'D'$$

Input A	*Input B*	*Input C*	*Output F*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 9. Truthtable of the majority function.

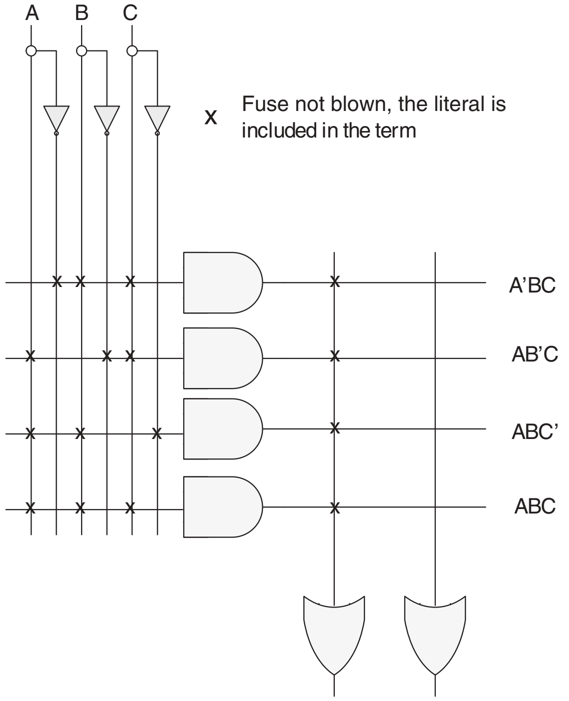


Figure 10. PLA implementation of F(A, B, C).

The implementation of *G* using a PLA is shown in Fig. 11.

EARLY PLAs

Near the beginning of 1970s, companies such as *Philips*, *Texas Instruments*, *National Semiconductor*,*Intersil*, *IBM*(2),and *Signetics* introduced early PLA and PLA-based devices. Early PLAs had limited numbers of input/output ports (around 20), array cells count (from hundreds to few thousands), and speeds (with around 1 to 35 nanoseconds delay). Later PLAs performed with greater speeds (with around 2 to 5 nanoseconds delay), with array sizes of thousands of cells, and input/output ports number of around 100 (3). Currently, some PLA-structures are parts of high-density, high-performance, and CPLDs.

Currently,*PLAs* are available in the market in different types. PLAs could be stand-alone chips, or parts of bigger processing systems. Stand-alone PLAs are available as mask programmable (MPLAs) and field programmable (FPLAs) devices. MPLAs are programmed at the time of manufacture, whereas FPLAs can be programmed by the user with a computer-aided design tool.

PLAs IN MODERN COMPLEX SYSTEMS AND AREAS OF APPLICATION

PLAs have largely motivated the development of many modern programmable systems. usually, PLAs are used as a part of a more complicated processing system. PLAs have also inspired the creation of complex PLA-based

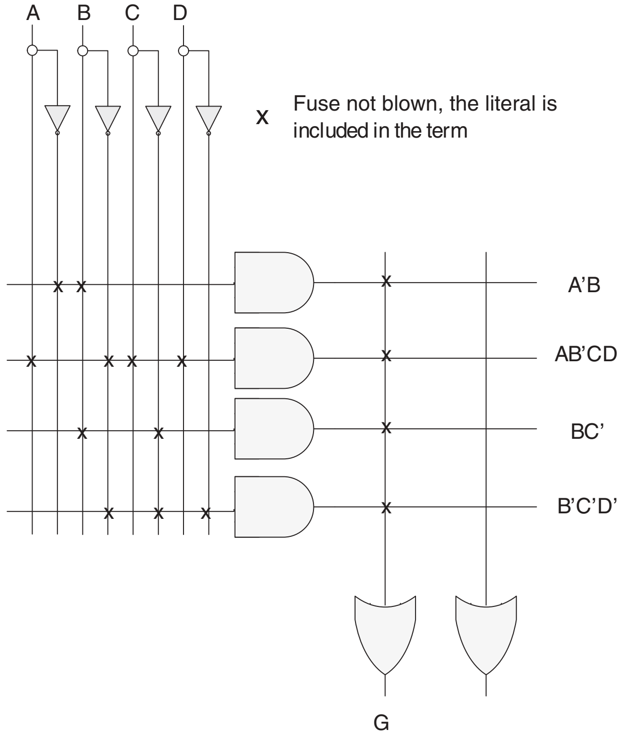


Figure 11. PLA implementation of G(A, B, C, D).

systems with PLA-like structure. The available variety of PLAs and PLA-based systems paved the way for their employment in many areas of application.

The *CoolRunner II* CPLD from *Xilinx* uses a PLA-type structure. This device has multiple function blocks (FBs). Each FB contains 16 macrocells and the FBs are interconnected by an advanced interconnect matrix. A basic architectural block diagram for the CoolRunner II with a greatly simplified diagram of an FB is shown in Fig. 12.

The CoolRunner II series of CPLDs contains from 32 macrocells to 512 macrocells. The number of FBs range from 2 to 32. The PLA structure contains a programmable *AND* array with *56 AND*-gates, and a programmable *OR* array with *16 OR*-gates. With the PLA structure, any product term can be connected to any *OR*-gate to create an SOP output. Each FB can produce up to *16* SOP outputs each with *56* product terms.

The main additions to the traditional PLA structure in a device such as *CoolRunner II* are the complex macrocells. A macrocell can be configured for combinational logic or sequential logic (with availability of flip-flops). The macrocell in a CoolRunner II also contains an XOR-gate to enable complementing the SOP output (coming from the PLA OR-gate) to produce a POS form. A *1* on the input of the XOR-gate complements the *OR* output (a POS form is produced) and a *0* keeps the output uncomplemented (in an SOP form). Choices between SOP forms and POS forms, various clock inputs, flip-flop include or bypass, are completed using different multiplexers.

Another famous device with a PLA-like structure is the ICT programmable electrically erasable logic (PEEL) array (4). PEEL arrays are large PLAs that include macrocells with flip-flops. The PEEL array structure is shown in Fig. 13 with its PLA-like planes; the outputs of the *OR*-plane are divided into groups of four, and each group can be input to any of the logic cells. The logic cells, depicted in Fig. 14, provide registers for the sum terms and can feed back the sum terms to the *AND*-plane. The logic cells can also connect the sum terms to the input–output pins. The multiplexers each produce an output of the logic cell and can provide either a registered or a combinational output. Because of their PLA-like planes, the PEEL arrays are well-suited to applications that require SOP terms.

The multiple *ALU* architecture with reconfigurable interconnect experiment system (MATRIX) is another modern architecture that benefits from the PLA architecture (5). The MATRIX architecture is unique because it aims to unify resources for instruction storage and computation. The basic unit (BFU) can serve either as a memory or a computation unit. The *8* BFUs are organized in an array, and each BFU has a *256*-word memory, an ALU-multiply unit, and a reduction control logic. The interconnection network has a hierarchy of three levels; it can deliver up

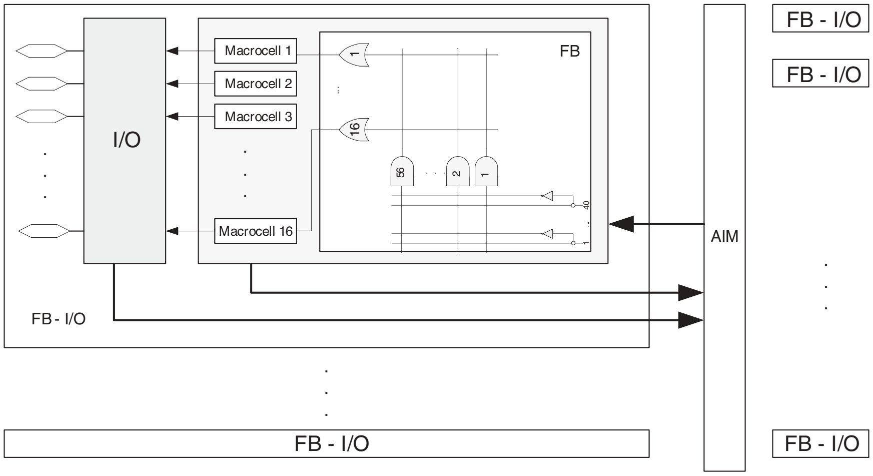


Figure 12. Architectural block diagram for the CoolRunner II.

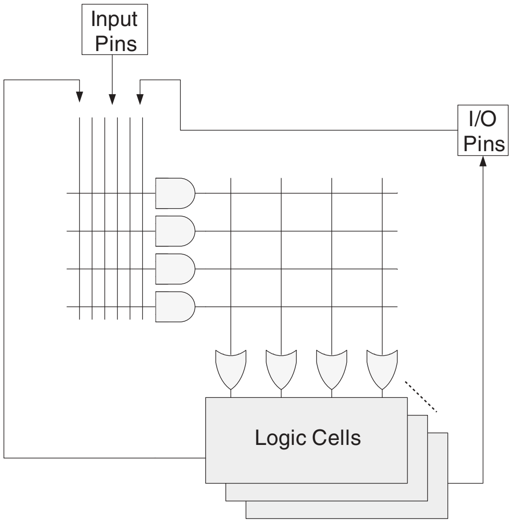


Figure 13. Main components in the architecture of ICT PEEL Arrays.

to 10 GOPS with 100 BFUs when operating at 100 MHz. The MATRIX controller is composed of a pattern matcher to generate local control from the ALU output, a reduction network to generate local control, and a 20-input, *8*-output NOR block that serves as half of a PLA.

One famous application of PLAs is to implement the control over a datapath in a processor. A datapath controller usually follows predefined sequences of states. In each control state, the PLA part of the controller will determine what datapath control signals to produce and the next state of the controller. The design of the controller usually starts by formulating different states and transitions using a state diagram. The state diagram is then formulated in a truth-table form (state transition table), where SOP equations could be produced. Then, the derived SOP equations are mapped onto the PLA. A design example of a datapath controller is shown in Figs. 16, 17, and 18. Figure 16 shows a typical controller state diagram. Figure 17 depicts the block diagram of the datapath controller. Figure 18 suggests a *PLA* implementation of the controller.

Many areas of application have benefited from PLAs and PLA-based devices, such as cryptography (6), signal processing (7), computer graphics (8), image processing (9), data mining (9), and networking (10).

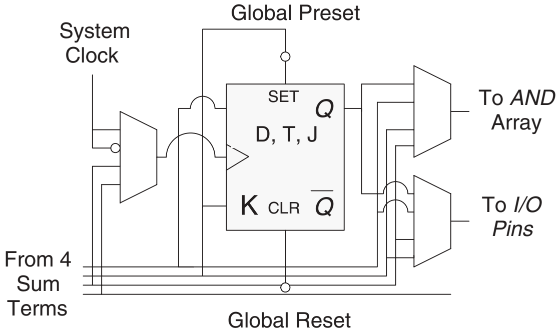


Figure 14. Structure of PEEL Array Logic Cell.

PROGRAMMING PLAs

Traditional PLAs usually are programmed using a PLA device programmer (such as traditional PROMs and EPROM-based logic devices). Some more complex PLA-based devices, such as CPLDs, can be programmed using device programmers; modern CPLDs are in-circuit programmable. In other words, the circuit required to perform device programming is provided within the CPLD chip. In-circuit programmability makes it possible to erase and reprogram the device without an external device programmer.

Modern CPLDs, which include the internal *PLA*-like structures, benefit from the latest advances in the area of hardware/software codesign. Descriptions of the desired hardware structure and behavior are written in a high-level context using hardware description languages such as Verilog. Then, the description code is compiled and downloaded in the programmable device before execution. Schematic captures are an option for design entry. Schematic captures have become less popular especially with complex designs. The process of hardware describe-and-synthesize development for programmable logic devices is shown in Fig. 15.

Hardware compilation consists of several steps. Hardware synthesis is the first major step of compilation, where an intermediate representation of the hardware design (called a netlist) is produced. A netlist usually is stored in a standard format called the electronic design interchange format and it is independent of the targeted device details. The second step of compilation is called place and route, where the logical structures described in the netlist are mapped onto the actual macrocells, interconnections, and input and output pins of the targeted device. The result of the place and route process is a called a bitstream. The bitstream is the binary data that must be loaded into the

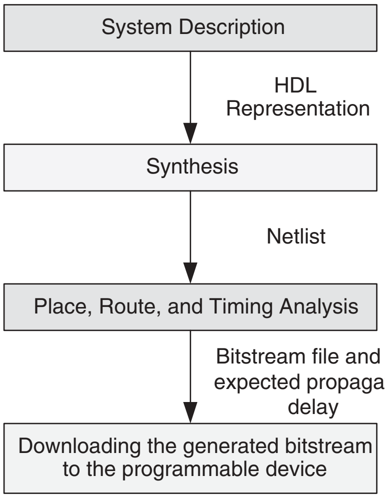

System Description
HDL Representation
Synthesis
Netlist
Place, Route, and Timing Analysis
Bitstream file and the expected propagation delay
Downloading the generated bitstream to the programmable device

Figure 15. The process of hardware describe-and-synthesize development for programmable logic devices.

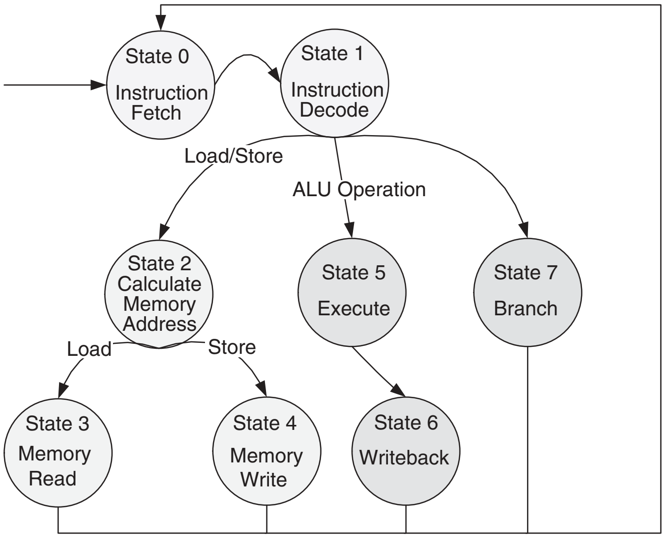


Figure 16. A design example of a datapath controller; the State Diagram.

PLD to program it to implement a particular hardware design.

THE RENEWABLE USEFULNESS OF PLAs

PLAs and their design basis have witnessed a renewable importance and have been the choice of designers for many systems as well as the target of different design methodologies. The renewable usefulness of PLAs is clear from the number of investigations carried out relying on the basis of PLAs.

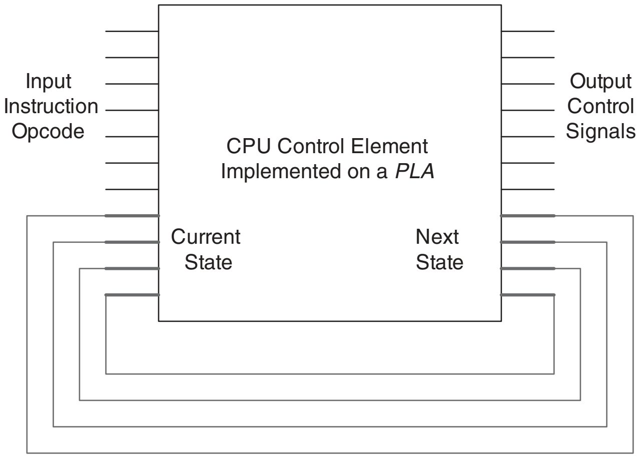


Figure 17. A design example of a datapath controller; Control Element block diagram,

A subthreshold circuit design approach based on asynchronous micropipelining of a levelized network of PLAs is investigated in Ref.11. The main purpose of the presented approach is to reduce the speed gap between subthreshold and traditional designs. Energy saving is noted when using the proposed approach in a factor of four as compared with a traditional designed networks of PLAs.

In Ref. 12, the authors propose a maximum crosstalk minimization algorithm that takes logic synthesis into consideration for PLA structures. To minimize the crosstalk, technique of permuting wires is used. The PLA product terms lines are partitioned into long set and short set, then product lines in the long set and the short set are

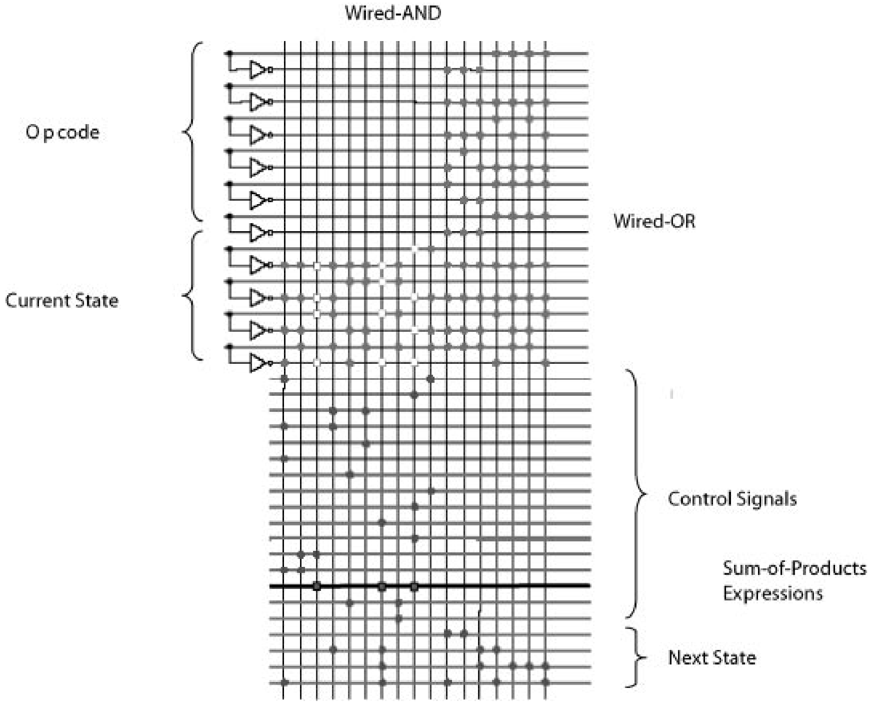


Figure 18. A design example of a datapath controller; PLA internal implementation.

interleaved. The interleaved wires are checked for the maximum coupling capacitance to reduce the maximum crosstalk effect of the PLA.

A logic synthesis method for an *AND-XOR-OR* type sense-amplifying PLA is proposed in Ref.13. Latch sense-amplifiers and a charge sharing scheme are used to achieve lowpower dissipation in the suggested PLA.

Testable design to detect stuck-at and bridging faults in *PLAs* is suggested in Ref. 14. The testable design is based on double fixed-polarity reed-muller expressions. An *XOR* part is proposed in the design implemented in a tree structure to reduce circuit delay.

A VLSI approach that addresses the cross-talk problem in deep sub-micron IC design is investigated in Ref.15. Logic netlists are implemented in the form of a network of medium-sized PLAs. Two regular layout "fabrics" are used in this methodology, one for areas where PLA logic is implemented, and another to route regions between logic blocks.

In Ref. 16, a PLA-based performance optimization design procedure for standardcells is proposed. The optimization is completed by implementing circuits' critical paths using PLAs. PLAs are proven to be good for such a replacement approach because they exhibit a gradual increase in delay as additional items are added. The final optimized hybrid design contains standard cells and a PLA.

A performance-driven mapping algorithm for CPLDs with a large number of PLA-style logic cells is proposed in Ref .17 .The primary goal of the mapping algorithm is to minimize the depth of the mapped circuit. The algorithm included applying several heuristic techniques for area reduction, threshold control of PLA fan-outs and product terms, slack-time relaxation, and PLA-packing.

The attractions of PLAs for mainstream engineers include their simplicity, relatively small circuit area, predictable propagation delay, and ease of development. The powerful but simple nature of PLAs brought them to rapid prototyping, synthesis, design optimization techniques, embedded systems, traditional computer systems, hybrid high-performance computing systems, and so on. Indeed, there has been renewable interests in working with the simple *AND*-to-*OR* PLAs.

BIBLIOGRAPHY

1. F. Vahid, T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*, New York: John Wiley & Sons, 2002.
2. R. A. Wood, High-speed dynamic programmable logic array chip. *IBM J. Res. Develop*. 379–383, 1975.
3. Z. E. Skokan, Symmetrical Programmable Logic Array, U.S. Patent 4,431,928.
4. S. Brown and J. Rose, Architecture of FPGAs and CPLDs: A Tutorial *IEEE Design Test Comp. 2*: 42–57, 1996.
5. E. Mirsky and A. DeHon, MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources, *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, 1996, pp. 157–166.
6. R. W. Ward and T. C. A. Molteno, A CPLD coprocessor for embedded cryptography, *Proc. Electronics New Zealand Conf*, 2003.
7. S. Pirog, M. Baszynski, J. Czekonski, S. Gasiorek, A. Mondzik, A. Penczek, and R. Stala, Multicell DC/DC converter with DSP/CPLD control, *Power Electronics and Motion Control Conf*, 2006, pp. 677–682.
8. J. Hamblen, Using large CPLDs and FPGAs for prototyping and VGA video display generation in computer Architecture design laboratories, *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, 1999, pp. 12–15.
9. A. Esteves, and A. Proença, A hardware/software partition methodology targeted to an FPGA/CPLD architecture, *Proc. Jornadas sobre Sistemas Reconfiguráveis*, 2005.
10. Z. Diao, D. Shen, and V. O. K. Li, CPLD-PGPS scheduling algorithm in wireless OFDM systems ,*Proc. IEEE Global Telecom. Conf 6: /bookTitle>, 2004, pp. 3732–3736.*
11. N. Jayakumar, R. Garg, B. Gamache, and S. P. Khatri, A PLA based asynchronous micropipelining approach for subthreshold circuit design, *Proc. Annu. Conf. on Design Automation*, 2006, pp. 419–424.
12. Y. Liu, K. Wang, and T. Hwang, Crosstalk minimization in logic synthesis for PLA, *Proc. Conf. on Design, Automation and Test in Europe, 2:*, 2004, pp. 16–20.
13. H. Yoshida, H. Yamaoka, M. Ikeda, and K. Asada, Logic synthesis for AND-XOR-OR type sense-amplifying PLA, *Proc. Conf. on Asia South Pacific Design automation/VLSI Design*, 2002, pp. 166.
14. H. Rahaman, and D. K. Das, Bridging fault detection in Double Fixed-Polarity Reed-Muller (DFPRM) PLA, *Proc. Conf. on Asia South Pacific Design Automation*, 2005, pp. 172–177.
15. S. P. Khatri, R. K. Brayton, and A. Sangiovanni-Vincentelli, Cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric, *Proc. IEEE/ACM Conf. on Computer-Aided Design*, 2000, pp. 412–419.
16. R. Garg, M. Sanchez, K. Gulati, N. Jayakumar, A. Gupta, and S. P. Khatri, A design flow to optimize circuit delay by using standard cells and PLAs, *Proc. ACM Great Lakes Symposium on VLSI*, 2006, pp. 217–222.
17. D. Chen, J. Cong, M. D. Ercegovac, and Z. Huang, Performance-driven mapping for CPLD architectures, *Proc. ACM/SIGDA Symposium on Field Programmable Gate Arrays*, 2001, pp. 39–47.

FURTHER READING

T. Floyd, *Digital Fundamentals with PLD Programming*, Englewood Cliffs, NJ: Prentice Hall, 2006.

M. Mano et al., *Logic and Computer Design Fundamentals*, Englewood Cliffs, NJ: Prentice Hall, 2004.

ISSAM W. DAMAJ
Dhofar University
Sultanate of Oman

REDUCED INSTRUCTION SET COMPUTING

ARCHITECTURE

The term *computer architecture* was first defined in the article by Amdahl, Blaauw, and Brooks of International Business Machines (IBM) Corporation announcing the IBM System/360 computer family on April 7, 1964 (1,2). On that day, IBM Corporation introduced, in the words of an IBM spokesperson, "the most important product announcement that this corporation has made in its history."

Computer architecture was defined as the attributes of a computer seen by the machine language programmer as described in the *Principles of Operation*. IBM referred to the Principles of Operation as a definition of the machine that enables the machine language programmer to write functionally correct, time-independent programs that would run across a number of implementations of that particular architecture.

The architecture specification covers all functions of the machine that are observable by the program (3). On the other hand, Principles of Operation are used to define the functions that the implementation should provide. In order to be functionally correct, it is necessary that the implementation conforms to the Principles of Operation.

The Principles of Operation document defines computer architecture, which includes:

- Instruction set
- Instruction format
- Operation codes
- Addressing modes
- All registers and memory locations that may be directly manipulated or tested by a machine language program
- Formats for data representation

Machine Implementation was defined as the actual system organization and hardware structure encompassing the major functional units, data paths, and control.

Machine Realization includes issues such as logic technology, packaging, and interconnections.

Separation of the machine architecture from implementation enabled several embodiments of the same architecture to be built. Operational evidence proved that architecture and implementation could be separated and that one need not imply the other. This separation made it possible to transfer programs routinely from one model to another and expect them to produce the same result which defined the notion of *architectural compatibility*. Implementation of the whole line of computers according to a common architecture requires unusual attention to details and some new procedures which are described in the Architecture Control Procedure. The design and control of system architecture is an ongoing process whose objective is to remove ambiguities in the definition of the architecture and, in some cases, adjust the functions provided (1,3,4).

RISC Architecture

A special place in computer architecture is given to RISC. RISC architecture has been developed as a result of the 801 project which started in 1975 at the IBM Thomas J. Watson Research Center and was completed by the early 1980s (5). This project was not widely known to the world outside of IBM, and two other projects with similar objectives started in the early 1980s at the University of California Berkeley and Stanford University (6,7). The term RISC (reduced instruction set computing), used for the Berkeley research project, is the term under which this architecture became widely known and recognized today.

Development of RISC architecture started as a rather "fresh look at existing ideas" (5,8,9) after revealing evidence that surfaced as a result of examination of how the instructions are actually used in the real programs. This evidence came from the analysis of the *trace tapes*, a collection of millions of the instructions that were executed in the machine running a collection of representative programs (10). It showed that for 90% of the time only about 10 instructions from the instruction repertoire were actually used. Then the obvious question was asked: "why not favor implementation of those selected instructions so that they execute in a short cycle and emulate the rest of the instructions?" The following reasoning was used: "If the presence of a more complex set adds just one logic level to a 10 level basic machine cycle, the CPU has been slowed down by 10%. The frequency and performance improvement of the complex functions must first overcome this 10% degradation and then justify the additional cost" (5). Therefore, RISC architecture starts with a small set of the most frequently used instructions which determines the pipeline structure of the machine enabling fast execution of those instructions in one cycle. If addition of a new complex instruction increases the "critical path" (typically 12 to 18 gate levels) for one gate level, then the new instruction should contribute at least 6% to 8% to the overall performance of the machine.

One cycle per instruction is achieved by exploitation of parallelism through the use of pipelining. It is *parallelism through pipelining* that is the single most important characteristic of RISC architecture from which all the remaining features of the RISC architecture are derived. Basically we can characterize RISC as a *performance-oriented architecture based on exploitation of parallelism through pipelining*.

RISC architecture has proven itself, and several mainstream architectures today are of the RISC type. Those include SPARC (used by Sun Microsystems workstations, an outgrowth of Berkeley RISC), MIPS (an outgrowth of

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

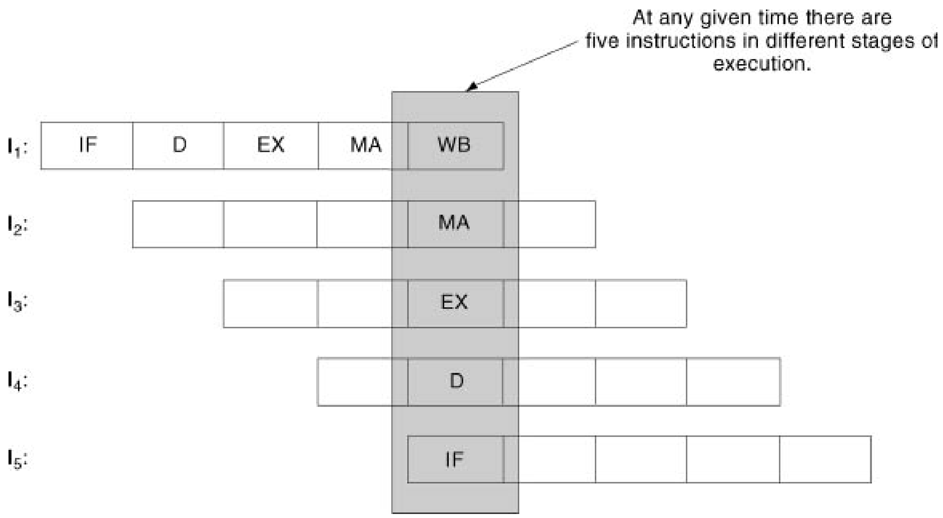

At any given time there are five instructions in different stages of execution.
I1:
IF
D
EX
MA
WB
I2:
MA
I3:
EX
I4:
D
I5:
IF

Figure 1. Typical five-stage RISC pipeline.

Stanford MIPS project, used by Silicon Graphics), and a superscalar implementation of RISC architecture, IBM RS/6000 (also known as PowerPC architecture).

RISC Performance

Since the beginning, the quest for higher performance has been present in the development of every computer model and architecture. This has been the driving force behind the introduction of every new architecture or system organization. There are several ways to achieve performance: technology advances, better machine organization, better architecture, and also the optimization and improvements in compiler technology. By technology, machine performance can be enhanced only in proportion to the amount of technology improvements; this is, more or less, available to everyone. It is in the machine organization and the machine architecture where the skills and experience of computer design are shown. RISC deals with these two levels—more precisely their interaction and trade-offs.

The work that each instruction of the RISC machine performs is simple and straightforward. Thus, the time required to execute each instruction can be shortened and the number of cycles reduced. Typically the instruction execution time is divided into five stages, namely, machine cycles; and as soon as processing of one stage is finished, the machine proceeds with executing the second stage. However, when the stage becomes free it is used to execute the same operation that belongs to the next instruction. The operation of the instructions is performed in a pipeline fashion, similar to the assembly line in the factory process. Typically, those five pipeline stages are as follows:

IF: Instruction Fetch

ID: Instruction Decode

EX: Execute

MA: Memory Access

WB: Write Back

By overlapping the execution of several instructions in a pipeline fashion (as shown in Fig. 1), RISC achieves its inherent execution parallelism which is responsible for the performance advantage over the complex instruction set architectures (CISC).

The goal of RISC is to achieve an execution rate of one cycle per instruction (CPI = 1.0), which would be the case when no interruptions in the pipeline occurs. However, this is not the case.

The instructions and the addressing modes in RISC architecture are carefully selected and tailored upon the most frequently used instructions, in a way that will result in a most efficient execution of the RISC pipeline.

The simplicity of the RISC instruction set is traded for more parallelism in execution. On average, a code written for RISC will consist of more instructions than the one written for CISC. The typical trade-off that exists between RISC and CISC can be expressed in the total time required to execute a certain task:

$$\text{Time (task)} = I \times C \times P \times T_0$$

where

$I =$ number of instructions/task

$C =$ number of cycles/instruction

$P =$ number of clock periods/cycle (usually $P=1$)

$T_0 =$ clock period (ns)

While CISC instruction will typically have less instructions for the same task, the execution of its complex operations will require more cycles and more clock ticks within the cycle as compared to RISC (11). On the other hand, RISC requires more instructions for the same task. However, RISC executes its instructions at the rate of one instruction per cycle, and its machine cycle requires only one clock tick (typically). In addition, given the simplicity of the instruction set, as reflected in simpler machine implementation, the clock period T_0 in RISC can be shorter, allowing the RISC machine to run at the higher speed as compared to CISC. Typically, as of today, RISC machines have been running at the frequency reaching 1 GHz, while CISC is hardly at the 500 MHz clock rate.

The trade-off between RISC and CISC can be summarized as follows:

1. CISC achieves its performance advantage by denser program consisting of a fewer number of powerful instructions.
2. RISC achieves its performance advantage by having simpler instructions resulting in simpler and therefore faster implementation allowing more parallelism and running at higher speed.

RISC MACHINE IMPLEMENTATION

The main feature of RISC is the architectural support for the exploitation of parallelism on the instruction level. Therefore all distinguished features of RISC architecture should be considered in light of their support for the RISC pipeline. In addition to that, RISC takes advantage of the principle of locality: *spatial* and *temporal*. What that means is that the data that was used recently is more likely to be used again. This justifies the implementation of a relatively large general-purpose register file found in RISC machines as opposed to CISC. Spatial locality means that the data most likely to be referenced is in the neighborhood of a location that has been referenced. It is not explicitly stated, but that implies the use of caches in RISC.

Load/Store Architecture

Often, RISC is referred to as Load/Store architecture. Alternatively the operations in its instruction set are defined as Register-to-Register operations. The reason is that all the RISC machine operations are between the operands that reside in the General Purpose Register File (GPR). The result of the operation is also written back to GPR. When restricting the locations of the operands to the GPR only, we allow for determinism in the RISC operation. In the other words, a potentially multicycle and unpredictable access to memory has been separated from the operation. Once the operands are available in the GPR, the operation can proceed in a deterministic fashion. It is almost certain that once commenced, the operation will be completed in the number of cycled determined by the pipeline depth and the result will be written back into the GPR. Of course, there are possible conflicts for the operands which can, nevertheless, be easily handled in hardware. The execution flow in the pipeline for a Register-to-Register operation is shown in Fig. 2.

Memory Access is accomplished through Load and Store instructions only; thus the term *Load / Store Architecture* is often used when referring to RISC. The RISC pipeline is specified in a way in which it must accommodate both operation and memory access with equal efficiency. The various pipeline stages of the Load and Store operations in RISC are shown in Fig. 3.

Carefully Selected Set of Instructions

The principle of locality is applied throughout RISC. The fact that only a small set of instructions is most frequently

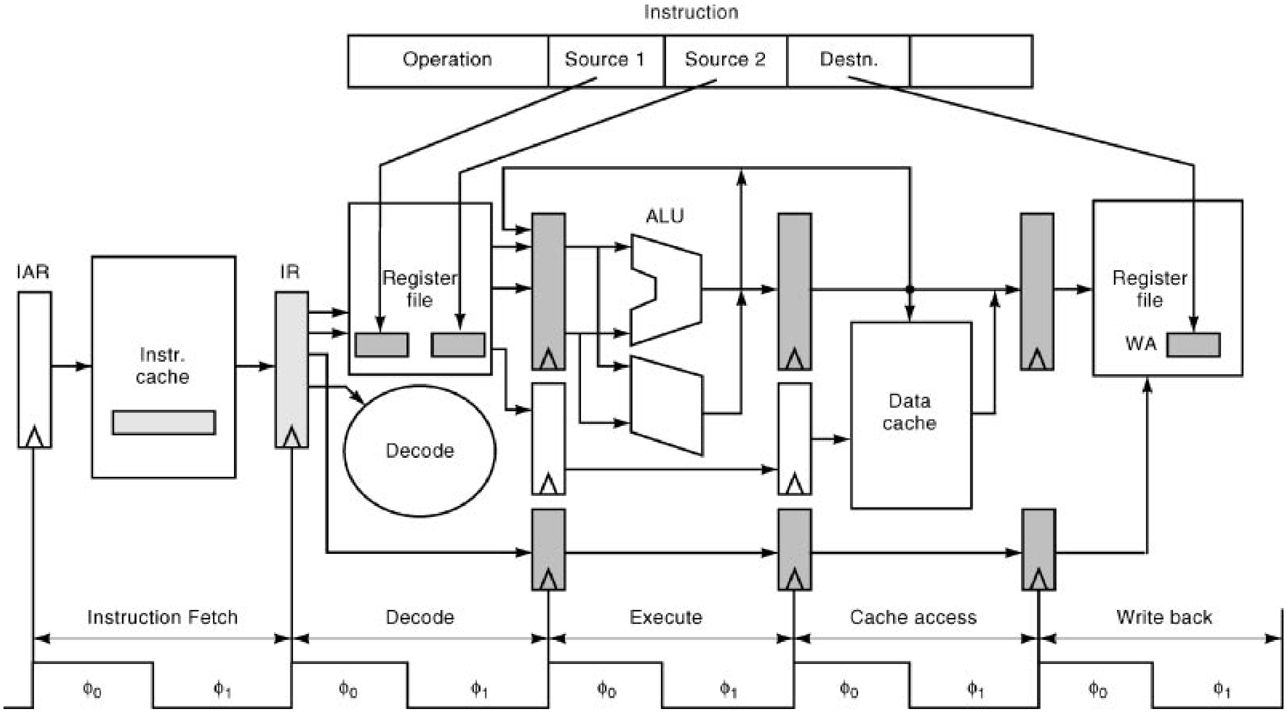


Figure 2. Pipeline flow of a Register-to-Register operation.

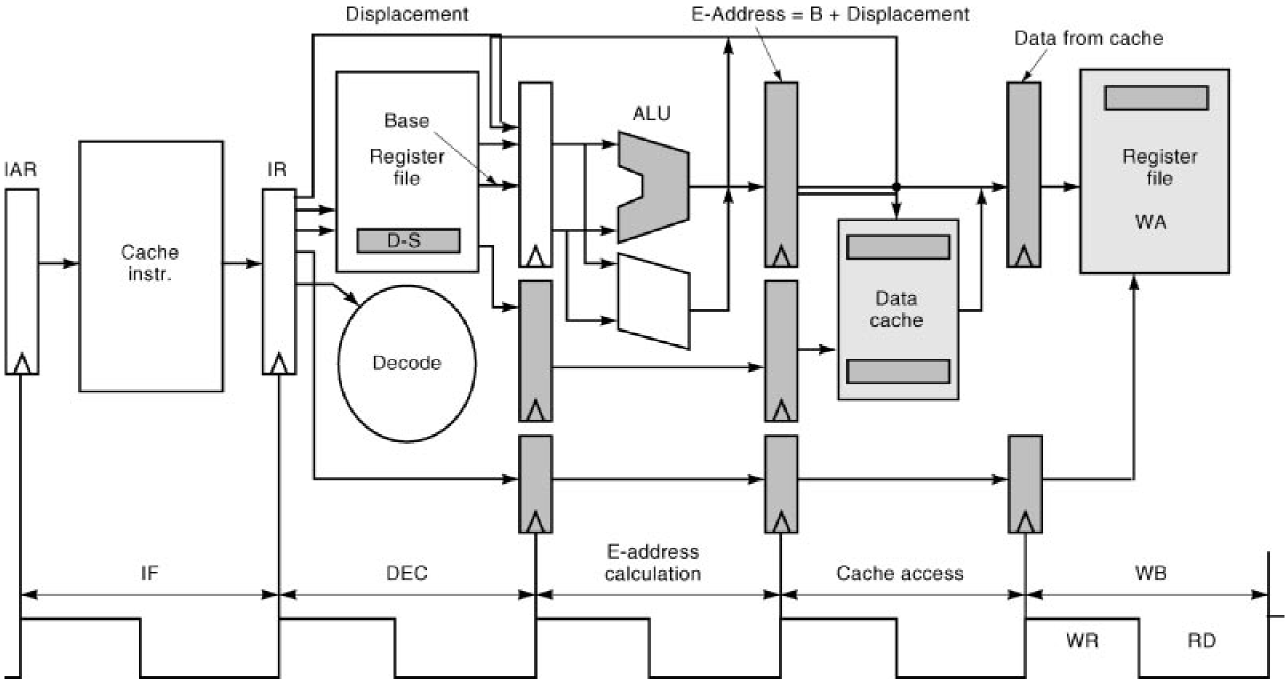


Figure 3. The operation of Load/Store pipeline.

used, was used in determining the most efficient pipeline organization with a goal of exploiting instruction level parallelism in the most efficient way. The pipeline is “tailored” for the most frequently used instructions. Such derived pipelines must serve efficiently the three main instruction classes:

- Access to Cache: Load/Store
- Operation: Arithmetic/Logical
- Branch

Given the simplicity of the pipeline, the control part of RISC is implemented in hardware—unlike its CISC counterpart, which relies heavily on the use of microcoding.

However, this is the most misunderstood part of RISC architecture which has even resulted in the inappropriate name: RISC. Reduced instruction set computing implies that the number of instructions in RISC is small. This has created a widespread misunderstanding that the main feature characterizing RISC is a small instruction set. This is not true. The number of instructions in the instruction set of RISC can be substantial. This number of RISC instructions can grow until the complexity of the control logic begins to impose an increase in the clock period. In practice, this point is far beyond the number of instructions commonly used. Therefore we have reached a possibly paradoxical situation, namely, that several of representative RISC machines known today have an instruction set larger than that of CISC.

For example: IBM PC-RT Instruction architecture contains 118 instructions, while IBM RS/6000 (PowerPC) contains 184 instructions. This should be contrasted to the IBM System/360 containing 143 instructions and to the IBM System/370 containing 208. The first two are representatives of RISC architecture, while the latter two are not.

Fixed Format Instructions

What really matters for RISC is that the instructions have a fixed and predetermined format which facilitates decoding in one cycle and simplifies the control hardware. Usually the size of RISC instructions is also fixed to the size of the word (32 bits); however, there are cases where RISC can contain two sizes of instructions, namely, 32 bits and 16 bits. Next is the case of the IBM ROMP processor used in the first commercial RISC IBM PC/RT. The fixed format feature is very important because RISC must decode its instruction in one cycle. It is also very valuable for superscalar implementations (12). Fixed size instructions allow the Instruction Fetch Unit to be efficiently pipelined (by being able to determine the next instruction address without decoding the current one). This guarantees only single I-TLB access per instruction.

One-cycle decode is especially important so that the outcome of the Branch instruction can be determined in one cycle in which the new target instruction address will be issued as well. The operation associated with detecting and processing a Branch instruction during the Decode cycle is illustrated in Fig. 4. In order to minimize the number of lost cycles, Branch instructions need to be resolved, as well, during the Decode stage. This requires a separate address adder as well as comparator, both of which are used in the Instruction Decode Unit. In the best case, one cycle must be lost when Branch instruction is encountered.

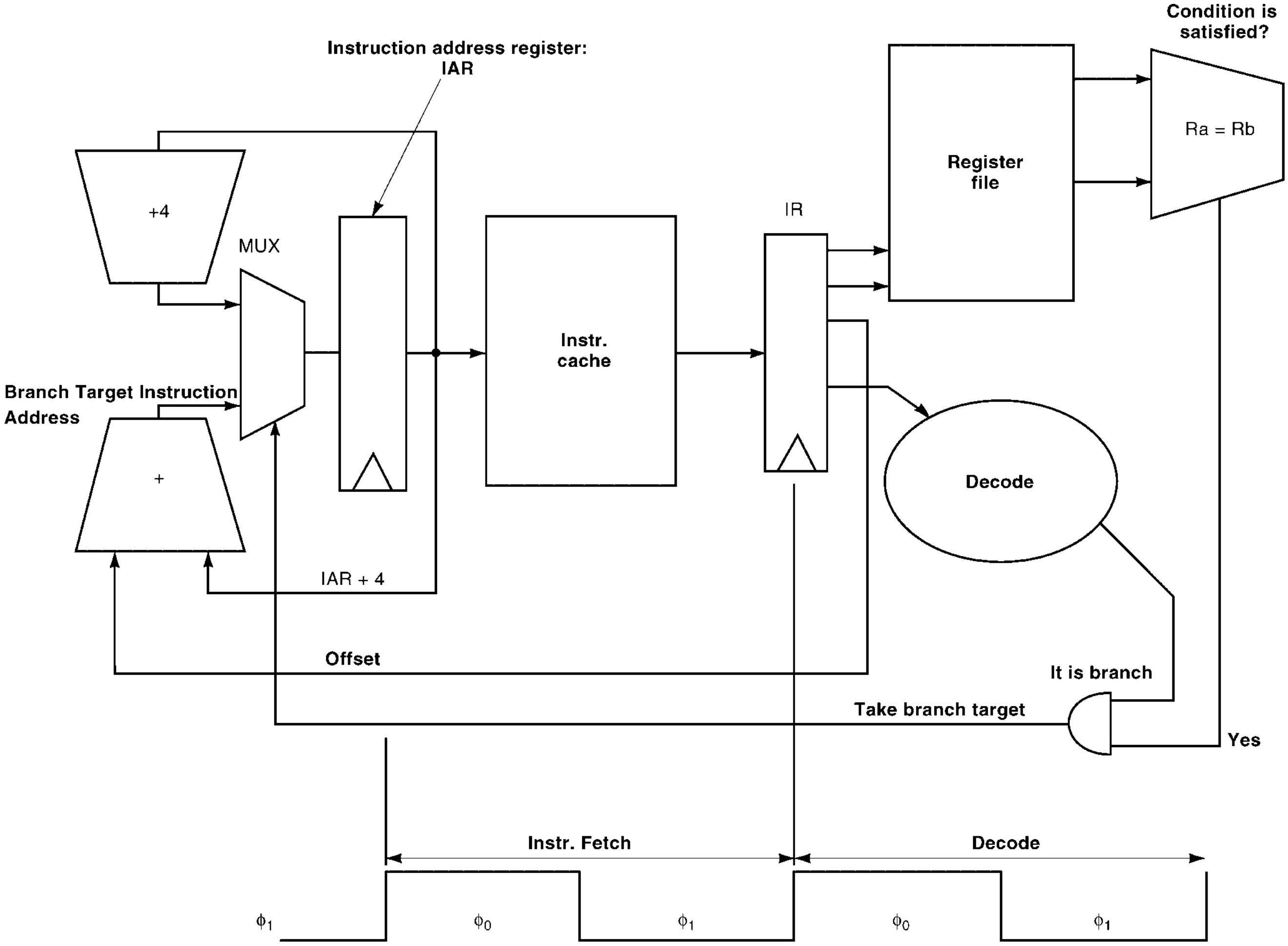


Figure 4. Branch instruction.

Simple Addressing Modes

Simple Addressing Modes are the requirements of the pipeline. That is, in order to be able to perform the address calculation in the same predetermined number of pipeline cycles in the pipeline, the address computation needs to conform to the other modes of computation. It is a fortunate fact that in real programs the requirements for the address computations favors three relatively simple addressing modes:

1. Immediate
2. Base+Displacement
3. Base+Index

Those three addressing modes take approximately over 80% of all the addressing modes according to Ref. (3): (1) 30% to 40%, (2) 40% to 50%, and (3) 10% to 20%. The process of calculating the operand address associated with Load and Store instructions is shown in Fig. 3.

Separate Instruction and Data Caches

One of the often overlooked but essential characteristics of RISC machines is the existence of cache memory. The second most important characteristic of RISC (after pipelining) is its use of the locality principle. The locality principle is established on the observation that, on average, the program spends 90% of the time in the 10% of the code. The instruction selection criteria in RISC is also based on that very same observation that 10% of the instructions are responsible for 90% of the code. Often the principle of the locality is referred to as a 90–10 rule (13).

In case of the cache, this locality can be spatial and temporal. *Spatial locality* means that the most likely location in the memory to be referenced next will be the location in the neighborhood of the location that was just referenced previously. On the other hand, *temporal locality* means that the most likely location to be referenced next will be from the set of memory locations that were referenced just recently. The cache operates on this principle.

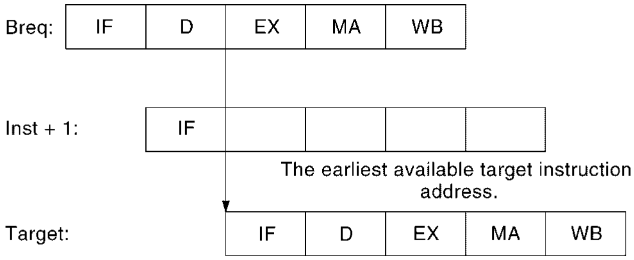


Figure 5. Pipeline flow of the Branch instruction.

The RISC machines are based on the exploitation of that principle as well. The first level in the memory hierarchy is the general-purpose register file GPR, where we expect to find the operands most of the time. Otherwise the Register-to-Register operation feature would not be very effective. However, if the operands are not to be found in the GPR, the time to fetch the operands should not be excessive. This requires the existence of a fast memory next to the CPU—the Cache. The cache access should also be fast so that the time allocated for Memory Access in the pipeline is not exceeded. One-cycle cache is a requirement for RISC machine, and the performance is seriously degraded if the cache access requires two or more CPU cycles. In order to maintain the required one-cycle cache bandwidth the data and instruction access should not collide. It is from there that the separation of instruction and data caches, the so-called Harvard architecture, is a must feature for RISC.

Branch and Execute Instruction

Branch and Execute or Delayed Branch instruction is a new feature of the instruction architecture that was introduced and fully exploited in RISC. When a Branch instruction is encountered in the pipeline, one cycle will be inevitably lost. This is illustrated in Fig. 5.

RISC architecture solves the lost cycle problem by introducing Branch and Execute instruction (5,9) (also known as Delayed Branch instruction), which consists of an instruction pair: *Branch* and the *Branch Subject* instruction which is always executed. It is the task of the compiler to find an instruction which can be placed in that otherwise wasted pipeline cycle.

The subject instruction can be found in the instruction stream preceding the Branch instruction, in the target instruction stream, or in the fall-through instruction stream. It is the task of the compiler to find such an instruction and to fill-in this execution cycle (14).

Given the frequency of the Branch instructions, which varies from 1 out of 5 to 1 out of 15 (depending on the nature of the code), the number of those otherwise lost cycles can be substantial. Fortunately a good compiler can fill-in 70% of those cycles which amounts to an up to 15% performance improvement (13). This is the single most performance contributing instruction from the RISC instruction architecture.

However, in the later generations of superscalar RISC machines (which execute more than one instruction in the pipeline cycle), the *Branch and Execute* instructions have been abandoned in favor of *Brand Prediction*(12,15).

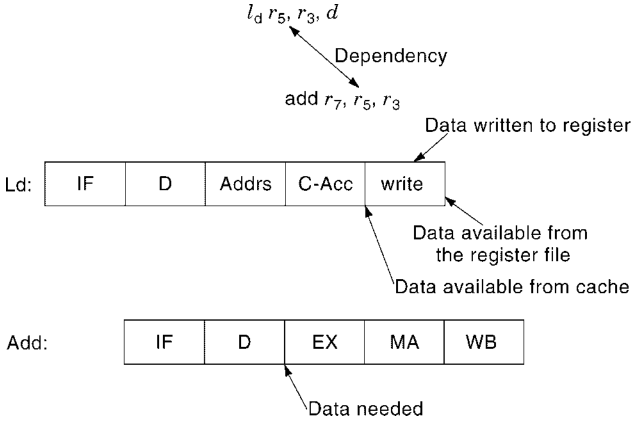


Figure 6. Lost cycle during the execution of the load instruction.

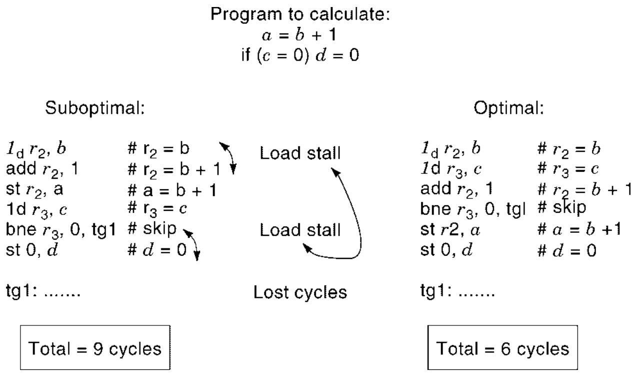
The Load instruction can also exhibit this lost pipeline cycle as shown in Fig. 6.

The same principle of scheduling an independent instruction in the otherwise lost cycle, which was applied for in Branch and Execute, can be applied to the Load instruction. This is also known as delayed load.

An example of what the compiler can do to schedule instructions and utilize those otherwise lost cycles is shown in Fig. 7 (13,14).

Optimizing Compiler

A close coupling of the compiler and the architecture is one of the key and essential features in RISC that was used in order to maximally exploit the parallelism introduced by pipelining. The original intent of the RISC architecture was to create a machine that is only *visible through the compiler*(5,9). All the programming was to be done in High-Level Language and only a minimal portion in Assembler. The notion of the "Optimizing Compiler" was introduced in RISC (5,9,14). This compiler was capable of producing a code that was as good as the code written in assembler (the hand-code). Though there was strict attention given to the architecture principle (1,3), adhering to the absence of the implementation details from the principle of the operation, this is perhaps the only place where this principle was violated. Namely, the optimizing compiler needs to "know"



```
1d r2, b            # r2 = b
add r2, 1           # r2 = b + 1
st r2, a            # a = b + 1
1d r3, c            # r3 = c
bne r3, 0, tg1      # skip
st 0, d             # d = 0

tg1: .......
```

```
1d r2, b            # r2 = b
1d r3, c            # r3 = c
add r2, 1           # r2 = b + 1
bne r3, 0, tgl      # skip
st r2, a            # a = b +1
st 0, d             # d = 0

tg1: .......
```

Figure 7. An example of instruction scheduling by compiler.

the details of the implementation, the pipeline in particular, in order to be able to efficiently schedule the instructions. The work of the optimizing compiler is illustrated in Fig. 7.

One Instruction per Cycle

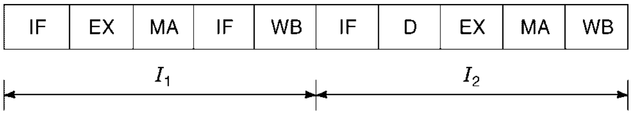
The objective of *one instruction per cycle* (CPI = 1) execution was the ultimate goal of RISC machines. This goal can be theoretically achieved in the presence of infinite size caches and thus no pipeline conflicts, which is not attainable in practice. Given the frequent branches in the program and their interruption to the pipeline, Loads and Stores that cannot be scheduled, and finally the effect of finite size caches, the number of "lost" cycles adds up, bringing the CPI further away from 1. In the real implementations the CPI varies and a CPI=1.3 is considered quite good, while CPI between 1.4 to 1.5 is more common in single-instruction issue implementations of the RISC architecture.

However, once the CPI was brought close to 1, the next goal in implementing RISC machines was to bring CPI below 1 in order for the architecture to deliver more performance. This goal requires an implementation that can execute more than one instruction in the pipeline cycle, a so called *superscalar* implementation (12,16). A substantial effort has been made on the part of the leading RISC machine designers to build such machines. However, machines that execute up to four instructions in one cycle are common today, and a machine that executes up to six instructions in one cycle was introduced in 1997.

Pipelining

Finally, the single most important feature of RISC is pipelining. The degree of parallelism in the RISC machine is determined by the depth of the pipeline. It could be stated that all the features of RISC (that were listed in this article) could easily be derived from the requirements for pipelining and maintaining an efficient execution model. The sole purpose of many of those features is to support an efficient execution of RISC pipeline. It is clear that without pipelining, the goal of CPI = 1 is not possible. An example of the instruction execution in the absence of pipelining is shown in Fig. 8.

We may be led to think that by increasing the number of pipeline stages (the pipeline depth), thus introducing more parallelism, we may increase the RISC machine performance further. However, this idea does not lead to a simple and straightforward realization. The increase in the number of pipeline stages introduces not only an overhead in hardware (needed to implement the additional pipeline registers), but also the overhead in time due to the delay of the latches used to implement the pipeline stages as well as the cycle time lost due to the clock skews and clock jitter. This could very soon bring us to the point of diminishing returns where further increase in the pipeline depth would result in less performance. An additional side effect of deeply pipelined systems is hardware complexity necessary to resolve all the possible conflicts that can occur between the increased number of instructions residing in the pipeline at one time. The number of the pipeline stages is mainly determined by the type of the *instruction core* (the most frequent instructions) and the operations required by those instructions. The pipeline depth depends, as well, on the technology used. If the machine is implemented in a very high speed technology characterized by the very small number of gate levels (such as GaAs or ECL), and a very good control of the clock skews, it makes sense to pipeline the machine deeper. The RISC machines that achieve performance through the use of many pipeline stages are known as *superpipelined* machines.



Total of cycles for two instructions

Figure 8. Instruction execution in the absence of pipelining.

Today the most common number of pipeline stages encountered is five (as in the examples given in this text). However, 12 or more pipeline stages are encountered in some machine implementations.

The features of RISC architecture that support pipelining are listed in Table 1.

Table 1. Features of RISC Architecture

Feature	Characteristic
Load/store architecture	All operations are Register to Register, so *Operation* is decoupled from *access to memory*
Carefully selected subset of instructions	Control implemented in hardware (no microcoding in RISC); set of instructions not necessarily small[a]
Simple addressing modes	Only most frequently used addressing modes used; important to fit into existing pipeline
Fixed size and fixed field instructions	Necessary to decode instruction and access operands in one cycle (there are, however, architectures using two sizes for instruction format (IBM PC-RT))
Delayed branch instruction (known also as *Branch* and *Execute*)	Most important performance improvement through instruction architecture (no longer true in new designs)
One instruction per cycle execution rate (CPI = 1.0)	Possible only through use of pipelining
Optimizing compiler	Close coupling between architecture and compiler (compiler *knows* about pipeline)
Harvard architecture	Separation of *Instruction* and *Data Cache* resulting in increased memory bandwidth

[a] IBM PC-RT Instruction architecture contains 118 instructions, while IBM RS/6000 (PowerPC) contains 184 instructions. This should be contrasted to the IBM System/360 containing 143 instructions and IBM System/370 containing 208. The first two are representatives of RISC architecture; the latter two are not.

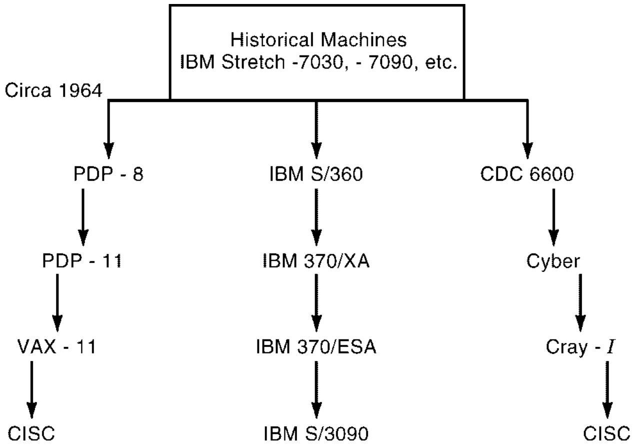


Figure 9. Main branches in development of computer architecture.

HISTORICAL PERSPECTIVE

The architecture of RISC did not come about as a planed or a sudden development. It was rather a long and evolutionary process in the history of computer development in which we learned how to build better and more efficient computer systems. From the first definition of the architecture in 1964 (1), there are the three main branches of the computer architecture that evolved during the years. They are shown in Fig. 9.

The CISC development was characterized by (1) the PDP-11 and VAX-11 machine architecture that was developed by Digital Equipment Corporation (DEC) and (2) all the other architectures that were derived from that development. The middle branch is the IBM 360/370 line of computers, which is characterized by a balanced mix of CISC and RISC features. The RISC line evolved from the development line characterized by Control Data Corporation CDC 6600, Cyber, and ultimately the CRAY-I supercomputer. All of the computers belonging to this branch were originally designated as *supercomputers* at the time of their introduction. The ultimate quest for performance and excellent engineering was a characteristic of that branch. Almost all of the computers in the line preceding RISC carry the signature of one man: Seymour Cray, who is by many given the credit for the invention of RISC.

History of RISC

The RISC project started in 1975 at the IBM Thomas J. Watson Research Center under the name of the 801. 801 is the number used to designate the building in which the project started (similar to the 360 building). The original intent of the 801 project was to develop an emulator for System/360 code (5). The IBM 801 was built in ECL technology and was completed by the early 1980s (5,8). This project was not known to the world outside of IBM until the early 1980s, and the results of that work are mainly unpublished. The idea of a simpler computer, especially the one that can be implemented on the single chip in the university environment, was appealing; two other projects with similar objectives started in the early 1980s at the University of California Berkeley and Stanford University (6,7). These two academic projects had much more influence on the industry than the IBM 801 project. Sun Microsystems developed its own architecture currently known as SPARC as a result of the University of California Berkeley work. Similarly, the Stanford University work was directly transferred to MIPS (17).

The chronology illustrating RISC development is illustrated in Fig. 10.

The features of some contemporary RISC processors are shown in Table 2.

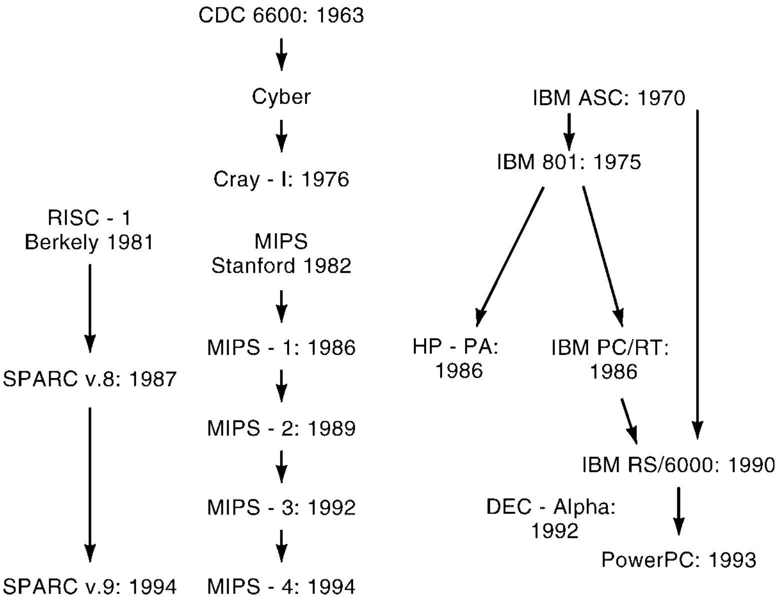


Figure 10. History of RISC development.

Table 2. Some features of RISC Processors

Feature	Digital 21164	MIPS 10000	PowerPC 620	HP 8000	Sun UltraSparc
Frequency (MHz)	500	200	200	180	250
Pipeline stages	7	5–7	5	7–9	6–9
Issue rate	4	4	4	4	4
Out-of-order execution	6 Loads	32	16	56	None
Register renaming (int/FP)	None/8	32/32	8/8	56	None
Transistors/logic transistors	9.3 M/1.8 M	5.9 M/2.3 M	6.9 M/2.2 M	3.9 M[a]/3.9 M	3.8 M/2.0 M
SPEC95 (Intg/FlPt)	12.6/18.3	8.9/17.2	9/9	10.8/18.3	8.5/15
Performance/log-trn (Intg/FP)	7.0/10.2	3.9/7.5	4.1/4.1	2.77[a]/4.69	4.25/7.5

[a] No cache.

BIBLIOGRAPHY

1. G. M. Amdahl, G. A. Blaauw, and F. P. Brooks, Architecture of the IBM System/360, *IBM J. Res. Develop.*, **8**: 87–101, 1964.
2. D. P. Siewiorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples*, Advanced Computer Science Series, New York: McGraw-Hill, 1982.
3. G. A. Blaauw and F. P. Brooks, The structure of System/360, *IBM Syst. J.*, **3**: 119–135, 1964.
4. R. P. Case and A. Padegs, Architecture of the IBM System/370, *Commun. ACM*, **21**: 73–96, 1978.
5. G. Radin, The 801 Minicomputer, IBM Thomas J. Watson Research Center, Rep. RC 9125, 1981; also in *SIGARCH Comput. Archit. News*, **10** (2): 39–47, 1982.
6. D. A. Patterson and C. H. Sequin, A VLSI RISC, *IEEE Comput. Mag.*, **15** (9): 8–21, 1982.
7. J. L. Hennessy, VLSI processor architecture, *IEEE Trans. Comput.*, **C-33**: 1221–1246, 1984.
8. J. Cocke and V. Markstein, The evolution of RISC technology at IBM, *IBM J. Res. Develop.*, **34**: 4–11, 1990.
9. M. E. Hopkins, A perspective on the 801/reduced instruction set computer, *IBM Syst. J.*, **26**: 107–121, 1987.
10. L. J. Shustek, Analysis and performance of computer instruction sets, PhD thesis, Stanford Univ., 1978.
11. D. Bhandarkar and D. W. Clark, Performance from architecture: Comparing a RISC and a CISC with similar hardware organization, *Proc. 4th Int. Conf. ASPLOS*, Santa Clara, CA, 1991.
12. G. F. Grohosky, Machine organization of the IBM RISC System/6000 processor, *IBM J. Res. Develop.*, **34**: 37, 1990.
13. J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, San Mateo, CA: Morgan Kaufman.
14. H. S. Warren, Jr., Instruction scheduling for the IBM RISC System/6000 processor, *IBM J. Res. Develop.*, **34**: 37, 1990.
15. J. K. F. Lee and A. J. Smith, Branch prediction strategies and branch target buffer design, *Comput.*, **17** (1): 1984, 6–22.
16. J. Cocke, G. Grohosky, and V. Oklobdzija, *Instruction control mechanism for a computing system with register renaming, MAP table and queues indicating available registers*, U.S. Patent No. 4,992,938, 1991.
17. G. Kane, *MIPS RISC Architecture*, Englewood Cliffs, NJ: Prentice-Hall, 1988.

READING LIST

D. W. Anderson, F. J. Sparacio, R. M. Tomasulo, The IBM 360 Model 91: Machine philosophy and instruction handling, *IBM J. Res. Develop.*, **11**: 8–24, 1967.

Digital RISC Architecture Technical Handbook, Digital Equipment Corporation, 1991.

V. G. Oklobdzija, Issues in CPU—coprocessor communication and synchronization, *EUROMICRO '88, 14th Symp. Microprocessing Microprogramming*, Zurich, Switzerland, 1988, p. 695.

R. M. Tomasulo, An efficient algorithm for exploring multiple arithmetic units, *IBM J. Res. Develop.*, **11**: 25–33, 1967.

VOJIN G. OKLOBDZIJA
Integration Corporation
Berkeley, California

SPECULATION

Modern compilers and processors employ many different kinds of speculation to improve parallel execution. Broadly speaking, the speculation can be categorized into three different groups.

1. Control speculation: Guess which instructions will be executed.
2. Value speculation: Guess the output value computed by an instruction.
3. Data access speculations: Guess what data will be accessed by a program.

Several things must be taken into consideration when deciding whether to speculate, such as the number of situations that may benefit from the speculation, the potential gain of the speculation, the accuracy of the speculation, and the penalty of misspeculation. If a particular form of speculation provides only modest performance improvement each time it is applied, it may be worthwhile if the speculation is applicable often, the speculation accuracy is sufficiently high, and the misspeculation penalty is small. On the other hand, speculation that offers huge performance gains may be inappropriate if the speculation accuracy is low or the misspeculation penalty is too high.

The remainder of this article will discuss various forms of speculation employed in modern processors and compilers. For each situation, the need for speculation will be described, the speculation and recovery mechanisms will be outlined, and the benefits of the speculation will be discussed.

CONTROL SPECULATION

In the von Neumann architecture, programs are expressed as a sequence of instructions executed by the processor. The processor retrieves each instruction from memory, performs the computation indicated by the instruction, and stores its result. The process then repeats for the instruction that lies in the adjacent memory location. Branch instructions interrupt this sequential processing and instruct the processor to begin executing from a new memory address (rather than the next sequential address). Most modern processors, however, do not implement the von Neumann architecture directly. Instead, several instructions execute in parallel (via pipelining, superscalar execution, very long instruction word execution). With multiple instructions in flight simultaneously, processors normally fetch later instructions from memory *before* prior instructions have completed. This action proceeds unhindered during sequential execution, but branch instructions create *control hazards*. To fetch from the correct location, a nonspeculative processor must wait for the result of a branch instruction to know from where to fetch subsequent instructions. Stalling for these control hazards reduces the processor's instruction throughput. These stalls lead to many unused cycles and resources because control instructions occur frequently (every 5 to 10 instructions) in program streams (1).

Branch Prediction

To alleviate the performance degradation imposed by branch instructions, processor microarchitectures employ *branch prediction*, which is a form of control speculation. When using branch prediction, rather than waiting for the result of a branch instruction, the processor predicts what the next instruction address will be and begins fetching and executing instructions from this location speculatively. When the branch instruction completes execution, the actual target address of the branch is compared with the predicted address. If the addresses match, then no recovery action is taken and the processor can continue executing normally. Otherwise, the processor must discard the results of the speculative instructions (because they should not have been executed) and begin executing instructions from target of the branch.

Although it may seem that predicting an arbitrary 64-bit (or even 32-bit) target address would be extremely difficult, several factors in instruction set architecture (ISA) design and programming paradigms make this task far less challenging (although the task is still nontrivial). Branch instructions in most modern ISAs can be classified along two orthogonal axes. First, branch instructions can be unconditional or conditional. Unconditional branches always transfer control to a new target. Conditional branches, as the name suggests, may redirect control to a new address or may allow execution to continue sequentially depending on a condition that guards branch. Second, branch instructions can be direct or indirect. Direct branches encode the target of the control transfer directly in the instruction. Indirect branches, on the other hand, refer to a register or memory location that contains the target of the branch. Consequently, the target for an indirect branch can be computed by other instructions whereas the target for a direct branch is known at program compile time.

The majority of branch instructions in a program are unconditional or conditional direct branches. Consequently, a branch predictor must only decide whether the branch will be taken. Predicting this binary value is more simple than speculating a target address. Many schemes for predicting branches have been presented in the literature and infact, this was a principle research focus in computer architecture in the past decade. The simplest scheme is to always predict a single direction for the branch (2). For example, a processor could predict statically that all branches will be taken. Such an approach works surprisingly well because loop back-edge branches are taken more often than not. More advanced techniques make predictions based on the history of past branches. Empirical observations show that if previous instances of a branch

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

in the program have gone in a particular direction, then future instances are likely to go the same way. Consequently, using branch history can improve prediction accuracy tremendously. Other techniques that leverage correlations between different branches or hints provided by compilers are also used frequently (3). Branch predictors implemented in commercial processors, using the techniques described here as well as more advanced techniques, have been able to predict in excess of 95% of conditional branches in many programs successfully.

Although the mechanisms just described allow predictions to be made for direct jumps, they are inadequate for indirect jumps. Many branch predictors include a structure called a *branch target buffer* (BTB) to help predict indirect jumps. The BTB stores the target for the last execution of a particular branch in the program (2).When the branch is encountered again, the target stored in the BTB is predicted. This target can be combined with traditional branch prediction for conditional indirect jumps.

In the event of a branch misprediction, it is necessary for the processor to discard the results of speculative instructions and restart execution at the correct point in the program. Depending on the microarchitecture, the hardware support necessary to roll back speculative instructions varies. In an in-order pipelined processor, the predicted branch instruction is guaranteed to execute *before* later instructions reach the pipeline stages where results are stored to the register files or memory. Consequently, when encountering a branch misprediction, the microprocessor need only throw out all instructions that were initiated after the branch and then begin re-executing at the correct target. On machines that execute instructions out of order, instructions that follow a speculated branch can execute before the branch. To ensure that the effects of these instructions can be undone, out-of-order processors allow instructions to complete in an arbitrary order but ensure that they *commit* in program order. Architectural state is only updated when an instruction commits, not when it completes. Consequently, if misspeculation occurs, all instructions after the branch can be discarded safely because none of them have yet committed. Various microarchitectural structures are used to guarantee in-order commit: a reorder buffer tracks instructions that have not yet committed, a store buffer manages values that need to be written to memory upon commit, physical register files and register rename tables store the results of uncommitted instructions. Despite the complexity of branch prediction and misspeculation recovery, branch prediction plays a vital role to provide performance on modern processors.

Compiler Control Speculation

In addition to branch prediction performed at runtime by the microprocessor, compilers also employ control speculation. During the scheduling phase of compilation, the compiler may choose to move an instruction after a branch to a position before the branch. The compiler must ensure that the hoisted instruction will not interfere with the program's execution if, in the original program, the instruction would not have executed. The code motion is speculative because it may not improve performance. If, in the original code, the instruction would not have been executed frequently, then this extra computation may hurt (or in the best case may not improve) performance. However, if the instruction has long latency, this code motion may separate a data definition and data use by many cycles to prevent or reduce stalls later in the program. Note however, that no recovery is needed here in the case of misspeculation; the transformation is always legal, it is just the performance benefit that is speculative.

Memory loads that miss in the processor's cache have particularly long latency. Unfortunately, hoisting a load above a branch is not always safe because the address used by the load may be invalid when the branch is not taken. In certain architectures (like Intel's Itanium architecture), the ISA has a speculative load instruction. Rather than throw an exception when given a bad address, the speculative load will fail by setting a bit on the destination register. Code is placed after the branch in the load's original location to verify that the load occurred successfully (4). Compiler control speculation can be important on architectures like Intel's Itanium that execute instructions in order. Because any stalled instruction prevents the machine from making forward progress, it is particularly important to prevent the stall conditions by hoisting long latency instructions (such as loads) as early as possible.

VALUE SPECULATION

Branch prediction involved predicting the outcome of branch instructions to prevent processor stalls while fetching instructions. The concept of predicting an instructions outcome can be extended beyond branch instructions and applied to other instructions. This type of speculation is classified broadly as value speculation. In value speculation, the goal is to break dependences between data producing instructions and definite or potential data consuming instructions. By breaking the dependence, the producing instruction (or chain of instructions) can be run in parallel with the consumer or potential consumers to enhance the amount of parallelism exploitable by the processor. Value speculation is often employed in aggressive out-of-order processors to allow load instructions to execute before prior store instructions. Other types of value speculation are less common but have been studied in the literature (5).

Data Dependence Speculation

Out-of-order processors attempt to execute instructions as soon as the instructions' operands have been computed. Unfortunately, it is difficult for the instruction dependence tracking hardware in out-of-order processors to know when memory operands (for load instructions) have been computed because any earlier (not-yet-executed) store instruction could potentially write to the location read by the load instruction.

Because the address written to by store instructions is computed dynamically by other instructions, the processor must wait until the store instruction executes to know what location in memory it will alter. Rather than wait for the store instructions to complete, many processors will speculate that no data dependence exists between the unexecuted stores and the pending load instruction. The load instruction then speculatively executes ahead of the store instruction (6). When the address for the store instruction is known, the speculation can be verified by comparing the address of the load with the address of the store. If the two addresses match, then the data dependence speculation has failed and appropriate recovery steps must be taken. Otherwise, the load instruction has received the correct value from the memory subsystem and the processor can continue executing instructions. More recently, the research community has investigated similar techniques to speculate data dependences across program threads. Transactional memories (7) and thread-level speculation (8,9) represent two significant thrusts in this effort. Much like control misspeculations, when the a data dependence misspeculation occurs, the processor needs to undo the effects of the misspeculated instructions. The mechanism used for branch mispredictions can be used for data dependence misspeculations as well. When misspeculation is detected, the processor discards all instructions after (and including) the misspeculated load instruction and restarts execution at the load instruction. By this time, some of the stores that precede the load have completed, so the load may reissue this time speculating fewer dependences. If all preceding stores have completed, then the load is nonspeculative and will not misspeculate again. To reduce the penalty associated with misspeculation, the processor may decide *not* to execute the load instruction speculatively a second time. That is to say, once a misspeculation has been detected, the processor will wait until all previous store instructions have completed before re-executing the load.

Various techniques have been proposed (10–13) to improve the accuracy of speculation. Rather than always assuming that a load does not alias with a not-yet-executed store, these techniques use a history of previous aliases to speculate more accurately. This improvement in accuracy retains the benefit of correct speculation while mitigating the penalty of misspeculation.

Compiler Data Speculation

The compiler can perform data dependence speculation instead of, or in addition to, the hardware. This technique was implemented in Intel's Itanium architecture. While performing scheduling, the compiler may not know whether it is legal to move a load instruction above an earlier store instruction. The compiler has two ways it can hoist the load while ensuring the resulting code behaves correctly. First, it can use static memory alias analysis techniques to determine whether the load and store could *possibly* access the same location, and if not hoist the load. The alternative is to move the load above the store speculatively and then insert code to check whether the load and store aliased. In the event that an alias *did* occur, the program would have to branch to fixup code where the load, and any already executed dependent instructions, are re-executed with the correct data values. The fixup code would then branch back to the normal code and resume normal execution.

To facilitate an efficient check and fixup mechanism, the Intel Itanium architecture provides two specialized instructions and a dedicated hardware structure for data speculation. The architecture provides an advanced load instruction and a load check instruction. The processor provides a table, known as the, advanced load address table (ALAT), which records the addresses accessed by all advanced loads. When the processor executes an advanced load, a bit is set in the ALAT. Any subsequent store to the same address clears the bit in the ALAT. The load check instruction inspects the bit in the ALAT, and if it is set, the instruction does nothing. On the other hand, if the bit is unset, the load check instruction branches to recovery code where any necessary fix up can occur. Note, that even in the absence of an ALAT, it is possible for the compiler to insert an instruction sequence to detect whether or not a store aliased with a given load. The problem, however, is that each store that is bypassed requires a small section of code to check for a potential aliasing. Consequently, the overhead of detecting misspeculation will probably outweigh the benefits of correct speculation.

DATA ACCESS SPECULATION

The growing speed differential between microprocessors and main memory has created a problem feeding data to a microprocessor. In an attempt to make memory access faster, most modern processors have several layers of cache memories. The goal of these small, fast memories is to store data that is likely to be accessed in the future, which reduces the latency of accesses. Unlike the previous two speculation techniques, cache memories do not try to predict information to run instructions earlier. Rather, the cache memories speculate on what *data* will be accessed soon, which allows fast memory accesses. Caches operate by storing data that has been accessed recently in the hope that it will be accessed again (a phenomenon known as *temporal locality*). Similarly, caches store data near recently accessed data in the hope that it will be accessed (a phenomenon known as *spatial locality*). A related speculation technique, prefetching, goes one step further by bringing values into cache memories by using the program's past memory access patterns to predict future accesses. This section will discuss both of these techniques and their relation to speculation.

Caches

Caches are small, fast memories that store a subset of the data that a program has stored into memory. Each time a processor performs a load or store instruction, the cache memory is consulted to see if the address being accessed is stored in the cache. If so, a *cache hit* has occurred, and the cache returns the data (or stores the new value in the case of stores). If the data is absent, a *cache miss* has occurred, and main memory (or a lower level of the cache hierarchy) is accessed.

Caches are considered speculative because of how data are added to the cache memory. Each time a cache miss occurs, the cache must choose whether or not to take the data returned by the remainder of the memory hierarchy and store it into the cache. Typically, all read misses will result in a cache block (the basic unit of storage in a cache) to be allocated to hold the resulting data. On cache writes, however, two different policies can be employed: allocate on write and no allocate on write. As the name implies, in one policy a write miss will cause a cache block to be allocated, whereas the other does not allocate a block. When an allocate on write policy is employed, the data to fill the cache block is fetched from the memory hierarchy, and then the piece that is being written is stored into the cache.

Unfortunately, when allocating a cache block, other data from the cache must be evicted. This eviction occurs because the cache memory is full (a capacity miss has occurred) or restrictions on where data can be placed forces another piece of data out of the cache (a conflict miss has occurred). Because it is possible that the data being evicted will be the next data accessed, the decision to store data in the cache may be detrimental. On the other hand, if the data just read is accessed again, storing the data in the cache will improve performance. Consequently, the decision is speculative. Because programs typically exhibit *temporal locality*, that is the same address is often accessed more than once in a local window of time, storing recently accessed data is typically a good guess when speculating.

Programs also exhibit *spatial locality*. That is to say, if a particular address is accessed, it is likely that adjacent locations will also be accessed. Spatial locality is a result of several programming paradigms. First, data structures are typically larger than a word and are often accessed together. Iteration across an array also creates significant amounts of spatial locality. Even local variables stored on a procedure's stack create spatial locality since variables within a single function are accessed together. Caches exploit spatial locality by operating at the cache block granularity rather than operating at the byte or word granularity. Block sizes vary from tens of bytes to hundreds of bytes. By placing an entire block into the cache memory, the processor is speculating that adjacent memory locations will be accessed.

Prefetching

Although most programs exhibit some amount spatial and temporal locality, not all accesses result in cache hits. However, patterns in the access streams often hint at which addresses will soon be accessed. For example, the addresses accessed by a processor pipeline's instruction fetch stage often form a linear sequence. Consequently, it is fairly simple to predict which memory locations will be accessed next. A technique known as *prefetching* attempts to exploit this regularity in accesses to pull data preemptively into caches. Once again, it is possible that prefetching will displace useful data from the caches and consequently is a speculative optimization.

The most basic prefetching algorithm is stride-based prefetching. If addresses that form an arithmetic sequence are observed, then the prefetch hardware predicts that the sequence will continue and accesses lower levels of memory to pull this data into the cache. The intent is that when the program accesses the data, it will already be present in the cache. Furthermore, prefetehers strive to avoid evicting other useful data. Instruction prefetchers often use stride prefetching because th access pattern forms a arithmetic sequence. Stride prefetchers are also used for data caches to improve the access speeds for array traversals. More complex prefetchers have been suggested in the literature for prefetching recursive data structures and other irregular access patterns (14,15).

In addition to hardware-based prefetching, certain architectures provide prefetch instructions. These instructions resemble load instructions in which the result of the instruction is never used. The compiler can insert these instructions if it has a prediction of what addresses will soon be accessed. The compiler may be able to prefetch more effectively than hardware because it is able to statically analyze an entire program region and can predict accesses that are not related directly to any previous accesses.

BIBLIOGRAPHY

1. S. Bird, A. Phansalkar, L. K. John, A. Mericas, and R. Indukuru, Characterization of performance of SPEC CPU benchmarks on Intel's Core microarchitecture based processor, *2007 SPEC Benchmark Workshop*, 2007.
2. J. Lee and A. J. Smith, Branch prediction strategies and branch target buffer design, *IEEE Computer*, 6–22, 1984.
3. T. Y. Yeh and Y. N. Patt, A comparison of dynamic branch predictors that use two levels of branch history, *Proceedings of the 20th Annual International Symposium on Computer Architecture*, 1993, pp. 257–266.
4. W. Y. Chen, S. A. Mahlke, and W.-M. W. Hwu, Tolerating first level memory access latency in high-performance systems, *Proceedings of the 1992 International Conference on Parallel Processing*, (Boca Raton, FL): CRC Press, 1992, pp. 36–43.
5. M. H. Lipasti and J. P. Shen, Exceeding the dataflow limit via value prediction, *Proceedings of the 29th International Symposium on Microarchitecture*, 1996, pp. 226–237.
6. R. Kessler, The Alpha 21264 microprocessor, *IEEE Micro*, **19**: 24–36, 1991.
7. M. Herlihy and J. E. B. Moss, Transactional memory: architectural support for lock-free data structures, *Proceedings of the 20th Annual International Symposium on Computer Architecture*, 1993, pp. 289–309.
8. G. S. Sohi, S. Breach, and T. N. Vijaykumar, Multiscalar processor, *Proceedings of the 22th International Symposium on Computer Architecture*, 1995.
9. J. G. Steffan, C. Colohan, A. Zhai, and T. C. Mowry, The STAMPede approach to thread-level speculation, *ACM Transactions on Computer Systems*, **23** (3). 253–300, 2005.
10. J. Gonzalez and A. Gonzalez, Speculative execution via address prediction and data prefetching, *Proceedings of the 1997 International Conference on Supercomputing*, 1997, pp. 196–203.
11. A. Moshovos, S. E. Breach, T. N. Vijaykumar, and G. S. Sohi, Dynamic speculation and synchronization of data dependences, *Proceedings of the 1997 International Symposium on Computer Architecture*, 1997.
12. A. Moshovos and G. S. Sohi, Streamlining inter-operation memory communication via data dependence prediction, *Proceedings of the 30th Annual International Symposium on Microarchitecture*, 1997, pp. 235–245.
13. G. Z. Chrysos and J. S. Emer, Memory dependence prediction using store sets, *Proceedings of the 25th Annual International*

Symposium on Computer Architecture, IEEE Computer Society, 1998, pp. 142–153.

14. A. Roth and G. S. Sohi, Effective jump-pointer prefetching for linked data structures, *Proceedings of the 26th International Symposium on Computer Architecture*, 1999.
15. O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, Runahead execution: an alternative to very large instruction windows for out-of-order processors, *Proceedings of the 9th International Symposium on High Performance Computer Architecture*, 2003.

David I. August
Neil Vachharajani
Princeton University
Princeton, New Jersey

STORAGE AREA NETWORKS

With both the quantity and the quality of information growing at unprecedented rates in recent years, data has become an increasingly vital asset for organizations of all sizes. This fact has fostered efforts to ensure that a suitable infrastructure exists to support, manage, protect, and reconfigure this data. Many organizations also require nearly continuous availability of their mission-critical data, which requires planning for business continuity and disaster recovery, including data replication over extended distances. All of these requirements are different from the traditional mainframe or enterprise computing environment, in which processing and storage resources are centralized. Traditional computing models based on the mainframe approach featured storage devices that were connected directly to a host server, and managed by dedicated information technology (IT) staff. Historically, the earliest approaches to storage management included server-attached storage, which tightly coupled storage devices to a server to reduce overhead. With storage devices dedicated to a single server, it was not necessary for any intelligence to reside on the disk and tape storage devices. Later devices incorporated storage control units, or storage off-load servers, which can perform more advanced functions such as cacheing of input/output (I/O) requests or dual redundant data copying. The advent of client/server-based computing created a new set of issues, as data was distributed among many servers and storage devices. With the complexity that developed from multiple operating systems, access methods, load balancing requirements, and disseminated management, this environment required a different approach to manage stored information. The various efforts to improve connectivity between these isolated storage resources and to manage distributed storage resources has led to the concept of a storage area network (SAN).

The terminology for different types of networks can be somewhat ambiguous and may not be used consistently in the technical literature. The term SAN is usually identified with block I/O services, rather than file access services. It can also refer to a storage system that consists of storage elements and devices, computer systems, and/or appliances, plus all of the associated control software, communicating over a network. Today, most SANs are based on either Fibre Channel, FICON, SCSI, or Ethernet protocols, although some other protocols such as ESCON or InfiniBand can also be involved; the parallel SCSI interface can also be run over Fibre Channel, where it is known as the Fibre Channel Protocol (FCP). This definition is not standardized, however; for example, the Storage Network Industry Association definition does not identify specifically the term *SAN* with Fibre Channel technology; instead, this group encourages the use of a qualified phrase such as "Fibre Channel SAN." Furthermore, according to the broadest definition, an Ethernet-based network whose primary purpose is to provide access to storage elements would be considered a SAN, although the term *network attached storage* (*NAS*) can also be applied. SANs are sometimes also used for system interconnection in computing clusters. For our purposes, the SAN constitutes any type of high speed network that provides interconnections between storage devices and servers, usually employing a dedicated switch fabric, regardless of the underlying protocol. It is based on the classic three-tiered computing model, which distinguishes among the presentation layer (end users with desktop personal computers), the middle tier (application servers), and the bottom tier (storage devices that contain the actual data). A SAN can be shared between multiple serves and storage devices. Although the most common implementations reside in a single data center, the SAN may be extended over large geographic distances for disaster recovery applications using protocol-specific channel extenders or wavelength division multiplexing (WDM). Although a WDM network is protocol independent, care must be taken to accommodate the performance and topology requirements of a SAN environment. For example, WDM equipment can be used to construct a ring topology that is not compliant with the Fibre Channel Arbitrated Loop (FC-AL) specifications.

In other publications, Ethernet and other IP or file-based storage networks are also known as NAS, which basically refers to a LAN-attached file server that uses a network access protocol, such as network file system (NFS) or CIFS. Thus, NAS is a generic term to refer to storage elements that connect to a network and provide file access services to computer systems. A NAS storage element consists of a processing engine that implements the file services (using access protocols such as NFS or CIFS), and one or more devices on which data is stored. Although Ethernet is the most common approach, NAS elements may be attached to any type of network. NAS devices can also coexist in a SAN environment, and various gateways between NAS and SAN environments are available. From a SAN perspective, a SAN-attached NAS engine is treated just like any other server. NAS solutions have evolved over time. Early NAS implementations used a standard UNIX or NT server with NFS or CIFS software to operate as a remote file server. Clients and other application servers access the files stored on the remote file server as though the files are located on their local disks. The location of the file is transparent to the user. Several hundred users could work on information stored on the file server, each one unaware that the data is located on another system. The file server has to manage I/O requests accurately, queuing as necessary, fulfilling the request, and returning the information to the correct client. As in many SANs, the NAS server handles most aspects of security and lock management.

Many different types of devices exist in a SAN besides storage and switching equipment. Host bus adapters (HBAs) are devices that connect to a server or storage

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

device and control the protocol for communications. These adapter cards will contain optical transceivers that will interface to channel extension or WDM equipment when the SAN is extended over longer distances (native SAN links based on Ethernet or Fibre Channel can accommodate up to about 10 km without using repeaters or channel extenders). A gateway (also referred to as a bridge or a router) is a fabric device used to interconnect one or more storage devices with different protocol support, such as SCSI to FC or FC to SCSI devices. Typically, hubs are used in a SAN to attach devices or servers that do not support switched fabrics but only FC-AL. Switches are useful to interconnect large numbers of devices, increase bandwidth, reduce congestion, and provide high aggregate throughput. The Fibre Channel protocol was designed specifically by the computer industry to remove the barriers of performance with legacy channels and networks. When a Fibre Channel switch is implemented in a SAN, the network is referred to as a fabric or switched fabric. Each device connected to a port on the switch can access potentially any other device connected to any other port on the switch, which enables an on-demand connection to every connected device. Various FC switch offerings support both switched fabric and/or loop connections. As the number of devices increases, multiple switches can be cascaded for expanded access (known as fanout). Inter-switch links (ISLs) can also be used to cascade multiple switches together, which may be more cost effective than constructing a single large switch. No industry standards for ISL connections exist, however, and many will operate only with the same vendor's equipment attached to either end of the link. Some ISLs also support trunking (the concatenation of multiple data channels using time division multiplexing techniques), which reduces the number of links in the SAN. Trunking between remote locations for data backup and disaster recovery can be done over WDM to increase the supported distances; in this case, trunking also reduces the number of inter-site links required. Directors are SAN devices, similar in functionality to switches, but because of the redundancy of many hardware components, they can supply a higher level of reliability, availability, and serviceability with a smaller footprint. Today, many directors can be used to connect FICON or FC devices at the same time. In the SAN environment, an extended distance gateway component [for example, an optical extender or a dense wave length division multiplexing (DWDM) device can connect two different remote SANs with each other over a wide area network (WAN).

SAN topologies may include point-to-point, arbitrated loops, fabrics, and other configurations. In particular, one topology specific to the SAN is the FC-AL, which is designed such that for a node to transfer data, it must first arbitrate to win control of the loop. Once the node has control, it is now free to establish a point-to-point (virtual) connection with another node on the loop. After this connection is established, the two nodes consume all of the loop's bandwidth until the data transfer operation is complete. Once the transfer is complete, any node on the loop can now arbitrate to win control of the loop. Support of up to 126 devices is possible on a single loop, but the more devices that are on a single loop, the more competition to win arbitration. A loop is self-discovering, and logic in the port allows a failed node to be isolated from the loop without interfering with other data transfers. A loop can be interconnected to other loops essentially to form its own fabric. An arbitrated loop supports communication between devices that do not recognize fabrics (private devices), and the resulting arbitrated loops are sometimes called private loops. Some unique implementations exist; for example, a Fibre Channel topology known as QuickLoop that combines arbitrated loop and fabric topologies is implemented only on the IBM 2109 family of switches (except the IBM TotalStorage SAN Switch M12), which allows a device with a private-loop HBA to communicate with FC-AL storage devices through IBM TotalStorage SAN switches. It is also possible to mix a switched fabric with an arbitrated loop, provided that the switches can detect and act on the correct protocols.

A SAN can be used to bypass traditional network bottlenecks. It facilitates direct, high speed data transfers between servers and storage devices and allows the same storage device to be accessed serially or concurrently by multiple servers. SANs also enable new network architectures where multiple hosts access multiple storage devices connected to the same network. A SAN may be used for high speed, high volume communications between servers or between storage devices. This outboard data movement capability enables data to be moved with minimal or no server intervention, thereby freeing up server processor cycles for other activities like application processing. Examples include a disk device backing up its data to a tape device without server intervention, or remote device mirroring across the SAN. When implemented properly, the ability for storage to be accessible through multiple data paths offers better reliability, application availability, and serviceability. If storage processing is off-loaded from servers and moved onto a separate network, higher application performance can result. SANs can centralize and consolidate storage systems, which reduces management overhead and improves scalability and flexibility of the storage design. Finally, SANs extended over remote distances enable disaster recovery, business continuity, and data vaulting applications.

The term *data sharing* describes the access of common data for processing by multiple computer platforms or servers. Data sharing can be between platforms that are similar or different; this term is also referred to as homogeneous and heterogeneous data sharing. With storage sharing, two or more homogeneous or heterogeneous servers share a single storage subsystem whose capacity has been partitioned physically so that each attached server can access only the units allocated to it. Multiple servers can own the same partition but only with homogeneous servers. Data-copy sharing allows different platforms to access the same data by sending a copy of the data from one platform to the other. In the ideal case, only one copy of the data is accessed by multiple platforms, whether homogeneous or heterogeneous. Every platform attached has read and write access to the single copy of data. This approach, sometimes called "true" data sharing, exists in practice only on homogeneous platforms. SANs enable multiple copy, data vaulting, and data backup operations on servers to be faster and independent of the primary network (LAN), which has led to the delivery of data movement applications such as LAN-free backup and server-less backup (to be discussed in more detail later).

Much attention is also being given to storage virtualization or the pooling of physical storage from multiple network storage devices into what seems to be a single storage device that is managed from a central console. Storage virtualization forms one of several layers of virtualization in a storage network; generally, it refers to the abstraction from physical volumes of data storage to a logical view of data storage. Storage virtualization separates the representation of storage to the operating system (and its users) from the actual physical components. Storage virtualization has been represented and taken for granted in the mainframe environment for many years.

SAN MANAGEMENT

SAN fabric monitoring and management is an area where a great deal of standards work is being focused. Two management techniques are in use: in-band and out-of-band management.

Device communications to the network management facility are most commonly done directly across the Fibre Channel or other network transport. This process is known as in-band management. It is simple to implement, requires no LAN connections, and has inherent advantages, such as the ability for a switch to initiate a SAN topology map by means of queries to other fabric components. However, in the event of a failure of the network transport itself, the management information cannot be transmitted. Therefore, access to devices is lost, as is the ability to detect, isolate, and recover from network problems. This problem can be minimized by a provision of redundant paths between devices in the fabric. In-band management is evolving rapidly. Proposals exist for low level interfaces such as return node identification and return topology identification to gather individual device and connection information, and for a management server that derives topology information. In-band management also allows attribute inquiries on storage devices and configuration changes for all elements of the SAN. Because in-band management is performed over the SAN itself, administrators are not required to make additional TCP/IP connections.

Out-of-band management means that device management data are gathered over a TCP/IP connection such as Ethernet, separate from the paths for the data traffic. Commands and queries can be sent using Simple Network Management Protocol (SNMP), Telnet (a text-only command line interface), or a Web browser Hyper Text Transfer Protocol (HTTP). Telnet and HTTP implementations are more suited to small networks. Out-of-band management does not rely on the transport network. Its main advantage is that management commands and messages can be sent even if a loop or fabric link fails. Integrated SAN management facilities are implemented more easily, especially by using SNMP. However, unlike in-band management, it cannot automatically provide SAN topology mapping.

A management information base (MIB) organizes the statistics provided by the out-of-band management interface. The MIB runs on an SNMP device and on the managed device. The SNMP protocol is supported widely by LAN/WAN routers, gateways, hubs, and switches, and it is the predominant protocol used for multivendor networks. Device status information (vendor, machine serial number, port type and status, traffic, errors, and so on) can be provided to an enterprise SNMP manager. Usually this runs on a workstation attached to the network. A device can generate an alert by SNMP, in the event of an error condition. The device symbol, or icon, displayed on the SNMP manager console, can be made to turn red or yellow, and messages can be sent to the network operator. Element management is concerned with providing a framework to centralize and to automate the management of heterogeneous elements, and to align this management with application or business policy. Several industry standard MIBs have been defined for the LAN/WAN environment. Special MIBs for SANs are being built by SNIA, which will enable multivendor SANs to be managed by common commands and queries. Two primary SNMP MIBs are being implemented for SAN fabric elements that allow out-of-band monitoring. The ANSI Fibre Channel Fabric Element MIB provides significant operational and configuration information on individual devices. The emerging Fibre Channel Management MIB provides additional link table and switch zoning information that can be used to derive information about the physical and logical connections between individual devices. Even with these two MIBs, out-of-band monitoring is incomplete. Most storage devices and some fabric devices do not support out-of-band monitoring. In addition, many administrators simply do not attach their SAN elements to the TCP/IP network.

A key aspect of SAN management is the ability to virtualize storage resources, or to isolate selected resources for security or performance reasons. This task can be done through logical partitioning and zoning. Zoning allows for finer segmentation of the switched fabric. Zoning can be used to instigate a barrier between different environments. Only members of the same zone can communicate within that zone, and all other attempts from outside are rejected. Zoning could also be used for test and maintenance purposes. For example, not many enterprises will mix their test and maintenance environments with their production environment. Within a fabric, it is possible to separate the test environment from the production bandwidth allocation on the same fabric using zoning.

One approach to securing storage devices from hosts wishing to take over already assigned resources is logical unit number (LUN) masking. Every storage device offers its resources to the hosts by means of LUNs. For example, each partition in the storage server has its own LUN. If the host (server) needs to access the storage, it needs to request access to the LUN in the storage device. The purpose of LUN masking is to control access to the LUNs. The storage device itself accepts or rejects access requests from different hosts. The user defines which hosts can access the LUN by means of the storage device control program. Whenever the host accesses a particular LUN, the storage device will check its access list for that LUN, and it will allow or disallow access to the LUN.

MULTIPROTOCOL ROUTING

Multiprotocol routers provide the means to connect SAN islands over multiple networks and across longer distances, which includes a combination of services such as FCP channel-to-channel routing, FCIP tunneling, iSCSI gateways, or encapsulation of other data protocols in a digital wrapper. These services can be supported on a switch or a router with a centralized management function, and they can be deployed on a per port basis. The primary advantage of this approach is the ability to connect devices between two or more fabrics without merging those fabrics, which provides a more flexible storage networking environment. Potential applications of this approach include connecting SAN environments across multiple geographies, functions, or departments (with centralized security and control), enabling low-cost SCSI-based servers and IP networks to support fabric-based business continuity and disaster recovery solutions over longer distances, and improving asset use through more efficient resource sharing. This function makes it possible to interconnect devices without having to redesign and reconfigure their entire environment, thereby eliminating the potential risks and costs of downtime. Moreover, the need to ensure secure connectivity for selected resources—especially in heterogeneous environments—additionally compounds the troubleshooting, fault isolation, and management challenges posed by large SANs. FCP routing addresses these issues by enabling organizations to connect devices in different SANs without merging the fabrics. Using this capability, organizations can share resources across multiple SANs and scale beyond current SAN port count support constraints. The benefits of enhanced SAN connectivity must be weighted against the potential increase in administrative workload, risk, and expense.

When devices on different fabrics are allowed to communicate through a multiprotocol router, the resulting connectivity group may be known as a logical SAN (LSAN). LSANs enable selective and secure resource sharing across multiple SANs by leveraging current zoning tools and methodologies. In addition to optimizing resource use, this approach helps to improve scalability by minimizing the risk and the complexity of large fabrics, simplifying management and fault isolation, and future-proofing current technology investments (LSANs are a logical or virtual construct, so they do not require changes to the existing SAN switches or attached edge devices).

Tunneling FCP over an IP network, known as FCIP, enables organizations to extend their Fibre Channel SANs over longer distances that would be technically impractical or prohibitively expensive with native Fibre Channel links, or in situations where dark fiber links would be impractical but in which IP WAN connectivity already exists. Furthermore, FCP routing enables two fabrics connected to an FCIP link to remain separate rather than merging them into a single fabric, which would permit any-to-any connectivity between all devices. This level of SAN connectivity facilitates applications such as migration to new storage (multiprotocol routers can be used to migrate data from lower data rate to higher data rate storage systems), data center consolidation, or data migration between test/development SANs and production SANs (enabling data movement between physically or logically separated environments).

As another example, iSCSI-to-Fibre Channel protocol conversion provides a standards-based iSCSI integration solution. The primary benefit is that low-cost servers can access centrally managed Fibre Channel resources. This approach leverages existing Ethernet infrastructure with IT staff knowledge to simplify implementation and management. It also reduces costs by eliminating the need to purchase HBAs in order for servers to access SAN resources.

LONG-DISTANCE SAN EXTENSION

With the advent of industry compliance regulations that govern the retention and the privacy of data, requirements have emerged to store and recover data over longer distances as part of an overall business continuance solution. One approach is the use of optical WANs equipped with wavelength multiplexing technology; this technology is capable of consolidating multiple fiber-optic connections over a single network, while providing distance extension using optical amplifiers. If a Fibre Channel protocol is being used with guaranteed data packet delivery, then performance at extended distances will be limited by credit-based flow control at layer 2 and 4. This difficulty can be overcome by using special adapters that provide large amounts of buffer credits, or by pooling buffer credits from multiple switch ports to serve a single extended distance port. Some types of channel extension have implemented "spoofing," which defeats true credit flow control in order to achieve higher performance; however, recovery from lost packets or link failures is problematic in such cases.

Sharing SAN resources and moving data over geographical boundaries introduces the complexity of merging resources and overcoming the distance and performance limitations caused by credit-based flow control on native Fibre Channel networks. It is possible to use an FCIP tunneling service to enable remote devices to connect to SANs by leveraging the existing IP WAN infrastructure for longer distance connectivity. As a result, organizations can share SAN resources and can move data between geographies much more efficiently. Integrating an FCIP tunneling solution within the fabric simplifies overall manageability by minimizing protocol conversion events. This standards-based FCIP approach, in combination with FC-to-FC routing, does not require connected fabrics to merge or to reconfigure. As a result, organizations can extend remote replication and backup functions over longer distances through a single platform hosting fabric-based applications. This approach increases resource use as well as reduces management, training, and troubleshooting requirements.

The advent of SANs enables data to be transferred directly from disk storage to the backup server and then directly over the SAN to tape. This process is called LAN-free backup. This process not only reduces LAN traffic but also reduces traffic through the backup server. Generally, this traffic is processor-intensive because of TCP/IP translations. With LAN-free backup, the backup server orchestrates the data movement, manages the tape library and drives, and tells the clients what data to move. The client is connected to the SAN; its data can be on the SAN, or the data can be on storage attached directly to the server. The LAN still is used to pass metadata back and forth between the backup server and the client. However, the actual backup data is passed over the SAN. The metadata is the data needed by the backup server to manage the entire backup process and includes the file name, the file location, the date and time of the data movement, and where the new copy resides. The metadata is small compared with the actual client data being moved. Server-less backup refers to the ability to take a snapshot of the data to be backed up with minimal or no disruption to productive work, then move it intelligently between tape and disk without the data going through a server.

REMOTE COPY SOLUTIONS

To improve availability, avoid outages, and minimize the adverse effects on business critical applications when they do occur, many businesses are using some form of remote copy service to mirror their critical data to a backup location. This service can be done over a WDM network to reduce the required number of inter-site optical fibers and to extend the native attached distance of many protocols. It protects the data against both planned outages (for maintenance, application software updates, or other reasons) and unplanned outages of various types.

A good disaster recovery plan must establish a recovery point objective (how much data can afford to be lost), a recovery time objective (how long critical systems can be unavailable), and a network recovery objective (how much of the network must be restored for operations to continue). Some research in this field has proposed a seven-tier recoverability model, based on the recovery method and recovery time. The nature of these recovery objectives will determine whether the remote backup solution needs to be synchronous or asynchronous. Generally, synchronous replication (also called synchronous mirroring) ensures that an I/O write is committed at the remote site before committing it to the primary site, and maintains data integrity with the remote site. This approach is required if no data loss can be tolerated, and immediate data restoration is required (such as applications in stock trading or airline reservation systems). Note that when using a synchronous solution, the distance and latency can impact performance because the writing application has to incur a round-trip delay; as a rule of thumb, practical systems are limited to about 50–100 km distances. Beyond this, asynchronous data replication is used, in which the data is copied to a remote site as expediently as possible, but not in real time. This approach is suited for users who can absorb some minimal data loss and cannot afford to have application performance impacted by the network round-trip delays.

One example of a remote copy solution is peer to peer remote copying (PPRC), also known as Metro Mirror, developed by IBM. This solution is an example of hardware-based synchronous replication that has been used for many years in mainframe environments. Similar technologies are available for other types of storage devices. It is used primarily to protect an organization's data against disk subsystem loss or, in the worst case, complete site failure. Metro Mirror is a synchronous protocol that allows real-time mirroring of data from one LUN to another LUN. Because the copying function occurs at the disk subsystem level, Metro Mirror is application independent. The protocol guarantees that the secondary copy is up to date by ensuring that the primary copy will be written only if the primary receives acknowledgment that the secondary copy has been written.

A related data copy solution is PPRC extended distance (PPRC-XD), which is intended to replicate log copies or static point-in-time copies of data. This solution maintains a "fuzzy" copy of the data on a secondary, remote volume, which can be synchronized with the source on demand. Thus, it is not a disaster recovery solution on its own, because the remote data is not in a state of integrity with its local copy. Technically, it is not asynchronous remote copy either, because that requires time stamp consistency between multiple control units, which is not part of PPRC-XD. Instead, this approach requires that the user periodically quesces the application writes, builds consistency between the local and remote copies, and then establishes application consistency over multiple storage volumes. Although the remote data integrity cannot be guaranteed at all times, PPRC-XD allows much greater distances (theoretically unlimited) and has a much lower impact on application performance because it does not require processor resources.

Although PPRC/Metro Mirror was developed by IBM, this technology has been licensed to other companies and is also supported on their platforms. Other remote copy solutions are available, for example, EMC Corporation offers a hardware-based solution that runs on the firmware of their Symmetrix disk storage array, called the Symmetrix remote data facility (SRDF). It provides several modes of operation, including synchronous copy, semisynchronous (which eases the latency impact a bit by allowing for one write behind), multihop (which uses a daisy chain of multiple storage arrays to reduce latency), and adaptive copy (which asynchronously updates the remote volumes without regard for the local volume write sequence). SRDF typically requires ESCON connectivity over channel extenders or WDM, but it can also operate over IP or Fibre Channel connections. Concurrent SRDF allows for simultaneous mirroring of data from one location to two different target locations; however, having two synchronous mirrors active at the same time may cause sensitivity to response time. The multihop solution can also be used to generate three or more copies of the data. As another example of remote copy solutions, Hitachi Data Systems offers several features including Hitachi remote copy (which is similar to PPRC), Hitachi extended remote copy (similar to XRC), and

Hitachi asynchronous remote copy, which timestamps all data and offers host-independent XRC functions. There is also a semisynchronous option similar to the EMC product. Typically, all of these solutions reach extended distances using a DWDM infrastructure.

Although the above solutions protect against hardware failures or environmental disasters in a SAN, they do not necessarily protect against user or application logical errors. In such cases, a hot standby database would be in the same inconsistent state as the live database. Split mirror backup/recovery functions as a high availability backup/recovery scenario, where the backup is taken on a remote disk subsystem that is connected to an application disk subsystem. Normally, the connection is suspended (mirror split) and will only be resumed for the resynchronization of the primary and the secondary volumes. In the case of a user or application error, the primary database is available for analysis, while a secondary database is recovered to a consistent point in time.

FlashCopy, another mirroring technique originally developed by IBM, provides an instant or point-in-time copy of a logical volume. The point-in-time copy functions provides an instantaneous copy of the original data at a specific point-in-time, known as the T0 (time-zero) copy. When a FlashCopy is invoked, the command returns to the operating system as soon as the FlashCopy pair has been established and the necessary control bitmaps have been created. This process takes only a few seconds to complete. Thereafter, the systems have access to a copy of the source volume. As soon as the pair has been established, it is possible to read and write to both the source and the target volumes. The point-in-time copy created by FlashCopy typically is used where a copy of production data needs to be produced with minimal application downtime.

An alternative approach is extended distance remote copy (XRC) also known as Global Mirror. It is an asynchronous, software-centric remote copy implementation. A software component called system data mover (SDM) will copy writes issued to primary volumes by primary systems, to the secondary devices. XRC uses the concept of a consistency group that contains records that have their order of update preserved across multiple logical partitions within a storage control unit, across multiple control units, and across other storage subsystems that participate in the same XRC session. Maintaining the update sequence for applications whose data is being copied in real time is a critical requirement for applications that execute dependent write I/Os. If data is copied out of sequence, serious integrity exposures could render the recovery procedures useless. XRC uses special algorithms to provide update sequence consistency for all data. XRC connectivity is provided by FICON or Fibre Channel links, and it is also part of the FICON extended distance solution. Various hybrid solutions are available that use components of both synchronous and asynchronous technology, such as a three-site solution in which two sites are close together enough to enable synchronous copy, whereas the third is far enough away that asynchronous copy is preferred for performance reasons.

REMOTE TAPE VAULTING AND CONSOLIDATION

Many companies have deployed SANs for tape backup and archiving using applications such as LAN-free backup and server-less backup. These near-zero backup window environments have driven the need for backup and archiving at remote and/or off-site locations; this process is called remote tape vaulting. An extension of remote tape vaulting is remote tape disaster tolerance, which involves a tape library at each of the sites. Remote tape vaulting consists of transmitting electronically and creating backup tapes at a secure off-site facility, moving mission-critical data off-site faster and with greater frequency than traditional data backup processes. The traditional backup process involves creating a tape in a locally attached tape library, ejecting it from the library, and removing it to an off-site location.

Tape backup solutions can be done with a combination of ESCON, FICON, or Fibre Channel links; the number of links depends on the size of the database concerned (terabytes or petabytes) and may be large. Many variations and proprietary implementations are not discussed here, such as IBM's point-to-point virtual tape server. All of these devices are suported by optical networking solutions, which usually are qualified by the storage device provider; an example is IBM's Total Storage Proven program. The major performance consideration with a long-distance solution is calculating the correct number of links between sites. This number can only be determined by performing a detailed performance profile of the servers and storage that will be remote. Overestimating the number of links will increase costs dramatically, whereas under sizing the number of links will affect the performance of the SAN dramatically. It is vital that detailed performance data is available prior to sizing the number of links required. Typically, latency will increase over long distances; a good rule of thumb is 4.8 microseconds per kilometer. Performance is highly application-dependent and requires careful configuration planning; for example, a remote disk consolidation solution might work well with a server-to-storage ratio of 6:1, whereas a tape vaulting solution may be acceptable with half this value.

Related concepts (which have no industry standard definition) include tape library sharing and tape drive pooling. A tape library consists of the physical robotics that move cartridges, one or more tape drives, and slots for tape storage. It must also have a mechanism to control the robotics (a library controller), and may also have a library manager, which maintains inventory and mediates sharing. Tape library sharing has been practiced for some time by partitioning a physical library into multiple logical libraries. Alternatively, the library can seem to be shared by multiple hosts when, in reality, one of the hosts (the library manager) is issuing all the library commands both for itself and for the other hosts (clients), but all of them have direct access to the tape drives (tape pooling). Tape pooling is the ability to allow two or more servers to logically share tape drives within a tape library. In this case, the servers need to be attached to the same SAN as the tape drives. Partitioning is the ability to partition tape drives and slots to create logical libraries within the same physical library. The server attached to each logical library has no knowledge of any drives or slots outside the partition, and the partitions are fixed.

Organizations can also use FC-to-FC routing functions to consolidate their backup activities in a SAN environment, sometimes known as tape backup consolidation. By centralizing the backup of multiple SANs in a single location, it is possible to streamline the overall backup process and help to ensure that both data and networks remain highly available. Additional benefits include reduced tape drive requirements, fewer idle resources, better overall use of tape libraries, and the ability to leverage off-peak network connectivity. A centralized backup architecture also optimizes the value of backup devices and resources, reduces management overhead, and increases asset utilization. The ability to share a tape library fabric without merging disk fabrics improves the overall flexibility and quality of SAN management. In addition, this approach saves money by eliminating the need to deploy separate HBAs on every backup server on every SAN fabric.

An example of the highest level of availability in remote copy systems is the Geographically Dispersed Parallel Sysplex (GDPS) developed by IBM and recently licensed to other vendors. It provides a combination of software and hardware to switch all resources automatically from one location to another. In 1994, IBM developed the Parallel Sysplex architecture for the System/390 (S/390) mainframe computing platform. This proprietary architecture uses high speed, fiber optic data links to couple processors together in parallel, thereby increasing reliability, performance, server capacity, and scalability. Several possible configurations exist for a Parallel Sysplex. First, the entire sysplex may reside in a single physical location, within one data center. Second, the sysplex can be extended over multiple locations with remote fiber optic data links. Finally, a multisite sysplex in which all data is remote copied from one location to another is known as a GDPS. This architecture provides the ability to manage remote copy configurations, automates both planned and unplanned system reconfigurations, and provides rapid failure recovery from a single point of control. Different configuration options exist for a GDPS. The single site workload configuration is intended for those enterprises that have production workload in one location (site 1) and discretionary workload (system test platforms, application development, etc.) in another location (site 2). In the event of a system failure, unplanned site failure, or planned workload shift, the discretionary workload in site 2 will be terminated to provide processing resources for the production work from site 1 (the resources are acquired from site 2 to prepare this environment, and the critical workload is restarted). The multiple site workload configuration is intended for those enterprises that have production and discretionary workload in both site 1 and site 2. In this case, discretionary workload from either site may be terminated to provide processing resources for the production workload from the other site in the event of a planned or unplanned system disruption or site failure.

Four building blocks exist for a Parallel Sysplex, with optical fiber links between them; the host processor (or Parallel Enterprise Server), the coupling facility (CF), the Sysplex Timer or Server Time Protocol source, and a storage network based on ESCON, FICON, or Fibre Channel. Remote copy functions are implemented between the storage at each physical location. Many different processors may be interconnected through the CF, which allows them to communicate with each other and with data stored locally. The CF provides data caching, locking, and queuing (message passing) services. By adding more processors to the configuration, the overall processing capacity of the sysplex will increase. Software allows the sysplex to break down large database queries into smaller ones, which can then be passed to the separate processors; the results are combined to arrive at the final query response. The CF may be implemented either as a separate piece of hardware or as a logical partition of a larger system. Fiber optic coupling links (also known as InterSystem Channel (or HiPerLinks) are used to connect a processor with a CF. Because the operation of a Parallel Sysplex depends on these links, it is highly recommended that redundant links and CFs be used for continuous availability. Coupling links are based on (but not fully compliant with) the ANSI Fibre Channel Standard. In synchronous applications, all servers must be connected to a common time-of-day clock, provided by either a Sysplex Timer or Server Time Protocol (STP) connection. Connectivity between remote locations is provided by a protocol independent WDM solution.

Many other data replication solutions over WDM exist; these include the High Availability Geographic Clustering software (HAGEO) from IBM, which operates over IP networks at essentially unlimited distances. This software provides options for either manual or automatic failover between two locations. Data I/O rates can be a problem, and HAGEO does not work well on high latency or low bandwidth networks. A properly designed application can run over thousands of kilometers. Another offering similar to this that provides real-time data mirroring over IP networks of RS/6000 or pSeries class workstations is GeoRM. Because the network is IP based, distances once again are unlimited essentially and many-to-one mirroring of critical data at remote location is possible; both synchronous and asynchronous versions are available. Many other SAN solutions exist that will not be discussed in detail here; these include Tivoli Storage Manager, logical volume mirroring with either sequential or parallel copy ordering and optional mirror write consistency checks, Microsoft MSCS for Fibre Channel SANs, Sun StorEdge Network Data Replicator, DataCore SAN Symphony, Veritas Volume Manager, and many others.

FUTURE DEVELOPMENTS

Looking beyond today's SAN requirements, some researchers are working on a model for on-demand storage networks that will adjust rapidly to spikes in usage, natural disasters, electronic viruses, and other dynamic changes to provide continuous access to data; this has been called e-business on demand. This concept includes utility computing (or pay-as-you-go models), autonomic or self-healing networks, server and storage capacity on demand, pervasive computing (computing elements embedded in distributed devices from laptops to cell phones), and other technologies beyond the scope of the optical network. All of these factors will influence the type and amount of traffic placed on future SANs.

FURTHER READING

Automated Remote Site Recovery Task Force report, SHARE 78, session M028, Anaheim, California, 1992.

A. Benner, *Fibre Channel for SANs*. New York: The McGraw-Hill Companies, 2001.

Brocade white paper, Extending SAN value with multiprotocol routing services. Available: www.brocade.com (Jan. 2007).

T. Clark, *Designing Storage Area Networks: A Practical Reference for Implementing Fibre Channel SANs*, Reading, MA: Addison-Wesley Longman, Inc., 1999.

C. DeCusatis, Data processing systems for optoelectronics. In *Optoelectronics for Data Communicaztion*, R. Lasky, U. Osterberg, and D. Stigliani, (eds.) New York: Academic Press, 1995.

C. DeCusatis, Optical data communication: fundamentals and future directions, *Opt. Eng.*, **37**(12): 3082–3099, 1998.

C. DeCusatis, Dense wavelength division multiplexing for storage area networks, *Proc. 2nd Online Symposium for Electrical Engineers (OSEE)*, Available: http://www.techonline.com/community/ed_resource/feature_article/14414.

C. DeCusatis, Storage area network applications, Invited talk, *Proc. OFC 2002*, Anaheim, California, 2002, pp. 443–444.

C. DeCusatis, Security feature comparison for fibre channel storage area network switches, *Proc. 5th annual IEEE Information Assurance Workshop*, U.S. Military Academy, West Point, New York, 2004, pp. 203–209.

C. DeCusatis, Fiber optic cable infrastructure and dispersion compensation for storage area networks, *IEEE Comm Mag.,* **43**(3): 86–92, 2005.

C. DeCusatis, Developing a threat model for enterprise storage area networks, *Proc. 7th annual IEEE Workshop on Information Assurance*, U.S. Military Academy, West Point, New York, 2006.

C. DeCusatis, ed., *Handbook of Fiber Optic Data Communication*, 3rd ed., New York: Elsevier/Academic Press, 2008.

C. DeCusatis and P. Das, Subrate multiplexing using time and code division multiple access in dense wavelength division multiplexing networks, *Proc. SPIE Workshop on Optical Networks*, Dallas, Texas, 2000, pp. 3–11.

C. DeCusatis, D. Petersen, E. Hall, F. Janniello, Geographically distributed parallel sysplex architecture using optical wavelength division multiplexing, *Optical Engineer.*, **37**(12): 3229–3236, 1998.

C. DeCusatis, D. Stigliani, W. Mostowy, M. Lewis, D. Petersen, and N. Dhondy, Fiber optic interconnects for the IBM S/390 parallel enterprise server, *IBM J. Res. Devel.*, **43**(5,6): 807–828, 1999.

M. Farley, *Building Storage Networks*, New York: The McGraw-Hill Companies, 2000.

The Fibre Channel Association. 1994. *Fibre Channel: Connection to the Future*, San Diego, CA: Elsevier Science and Technology Books.

Optical interface for multichannel systems with optical amplifiers, Draft standard G.MCS, annex A4 of standard COM15-R-67-E. Available from the International Telecommuncation Union, 1999.

C. Partridge, *Gigabit Networking*: Reading, MA: Addison- Wesley, 1994.

M. Primmer, An introduction to Fibre Channel. *Hewlett Packard J.***47**: 94–98, 1996.

A. Tanenbaum, *Computer Networks*, Englewood Cliffs, NJ: Prentice-Hall, 1989.

Websites

Hardcopies of industry standards documents may be obtained from Global Engineering Documents, An IHS Group Company, at http://global.ihs.com/. Also, electronic versions of most of the approved standards are also available from http://www.ansi.org. Additional information on ANSI standards and on both approved and draft international, regional and foreign standards (ISO, IEC, BSI, JIS, etc.) can be obtained from the ANSI Customer Service Department. References under development can be obtained from NCITS (National Committee for Information Technology Standards) at http://www.x3.org.

The following websites provide information on technology related to Fibre Channel. SANs and storage networking:

- http://webstore.ansi.org—Web store of the American National Standards Institute Softcopies of the Fibre Channel Standards documents.
- http://www.fibrechannel.org—Fibre Channel Industry Association. http://www.snia.org—Storage Networking Industry Association. http://www.storagep erformance.org—Storage Performance Council.

Industry organizations related to various aspects of storage networking:

- http://www.infinibandta.org—InfiniBand Trade Association—provides information on the objectives, history, and specification of InfiniBand network I/O technology.
- http://www.iol.unh.edu—University of New Hampshire InterOperability Laboratory Tutorials on many different high-performance networking standards.

Casimer DeCusatis
IBM Corporation
Poughkeepsie, New York

VIRTUAL MEMORY AND BUFFER STORAGE

CONCEPT

As applications become larger and more complex, and numerous applications are loaded onto computer systems, some processes may need memory larger than the physical main memory whereas a combined number of applications/processes may also overload the allowable physical main memory. Also, to make the operating system (OS) user-friendly, the OS has become complex using huge amounts of memory to maintain the OS. As it is not cost effective to have a huge physical memory, the *virtual memory* concept is used in almost all of the computer systems today to augment that lack of memory. The virtual memory is a technique that combines the main memory and the slower storage (usually hard disk) to provide the appearance of having a larger memory than actually installed to the computer system. This technique allows the execution of processes that are larger than the physical main memory and allows more processes to be in the physical memory simultaneously. Using the virtual memory concept, it can be assumed that the computer has near unlimited amount of memory at its disposal for use in loading various applications and processes. Here “unlimited” means that the programmer usually does not need to be concerned with the memory size requirements of his/her application that needs to be executed.

Consider the types of memory accesses that would be required to execute a single machine instruction: The instruction is fetched from memory, the data on which the instruction will operate is read from memory, and after the instruction completes, the results of the instruction are written back to memory. The actual number of bytes necessary for each memory access varies according to the CPU’s architecture, the actual instruction, and the data type. However, the memory requirement for several instructions will be still a lot less than the memory necessary to load the whole application. Using a method to keep track of an application’s memory requirements as it executes, it would be possible to keep that application running while using less than its total memory space. The virtual memory technique allows this, and only part of the application is in main memory at any given time, whereas the rest of the application remains on disks. This method might at first seem to have a very large performance problem as disk drives are so much slower than the main memory. Although the problem is there, it is possible to take advantage of the sequential and localized access behavior of applications and eliminate most of the performance implications of using disk space for virtual memory. This is done by structuring the virtual memory system so that it attempts to ensure that parts of the application that are currently needed, or likely to be needed in the near future (using locality among other information), are kept in the main memory only for as long as they are needed. In many aspects this is similar to the relationship between the cache and the main memory: making a little fast storage and a lot of slow storage look like a lot of fast storage. The main differences, in current computer systems, are that replacement on cache misses are primarily controlled by hardware, whereas virtual memory replacement is primarily controlled by the operating system and the size of the processor address determines the size of the virtual memory but not for caches.

For physical memory, there is *physical address space* where the address is used to access the physical memory (i.e., main memory). Whereas *virtual address space* is the program’s address space, that is, the memory requirement of an application or a program if it needed all the memory at once. This is the total number of uniquely addressable memory locations required by the application, and *not* the amount of physical memory that must be dedicated to the application at any given time. All of the memory references within a process are *virtual addresses* that are dynamically translated into*physical addresses* at run time, and a process may be swapped in and out of the main memory such that it occupies different regions of main memory at different times during the course of its execution. The address translation is performed by the OS and/or CPU. Therefore, programs that require memory larger than the physical main memory can be executed without the programmer’s manually overlaying and swapping blocks of the large program.

PAGING

A process to be executed may be broken up into a several pieces (pages or segments), and these pieces need not be contiguously located in the main memory during execution. That is, it is not necessary for all parts of a process to be in the main memory during execution. Naturally, if a page or a segment is not available in the main memory during execution, a mechanism is needed to bring the piece from the disk, into the main memory. The mechanism to transfer pages between main memory and an auxiliary store, such as hard disk, is called *paging*. The bringing of a required part of a process incurs some performance penalty. This penalty may degrade the performance severely, since we need to bring the required page from a disk to a main memory, even to a cache. Instead of a programmer concentrating on how much memory an application needs to run and trying to manage the physical memory usage of the application manually, a virtual memory operating system continually attempts to find the answer to the minimum amount of memory space an application needs to execute. Because processes are broken into parts and only some parts are sent to the main memory, the amount of main memory required to execute an application at any given time is less than the size of the application/process, usually a lot less.

To apply paging, the physical memory is broken into fixed-sized blocks called *frames*. Although the virtual

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

memory is broken into blocks of the same size called *pages,* the virtual address generated by the CPU is divided into two parts: a page number and a page offset. The page number is the index in the page table. The page table contains the frame number of the block in the main memory, and the offset is used to then determine the position of the memory that was needed. Virtual memory systems initially load pages only as they are needed, and other pages may be demanded as the loaded pages are used and no longer needed. Pages that are never accessed are never loaded into the physical main memory. There are two common page fetch policies, and they are *demand paging* and *prepaging*. For the demand paging scheme, a page is brought to the main memory only when a reference is made to a location on that page. The prepaging scheme loads other pages including the page that is referenced into the main memory. Assuming a spatial locality (pages near the requested page in terms of space location may be accessed very soon), a prepaging scheme may effectively reduce the wait time when there is a page fault and a page must be loaded into the main memory as the expected page is already loaded. A page fault occurs when the requested page is not available in the main memory.

PAGE TABLE

The primary job of the page table is to translate the virtual memory addresses to physical memory addresses. There are other uses for the page table. Because only a subset of the total pages for a process is loaded to the main memory, the page table needs to maintain the information of which pages are present in the main memory. Another usage is to indicate whether the page was altered since it was last loaded into main memory. If a page was altered, then that page needs to be written to the hard disk when it comes time to replace the page in the main memory. If the page was not altered, there is no action to be done for the page.

In most computer systems, there is one page table per process. This may become problematic as some process can occupy huge amounts of virtual memory. If each process can have up to $2^{31} = 2$ Gbytes of virtual memory, using $2^{12} = 4096$ byte pages, this means that there can be as many as 2^{19} page table entries per process. This is clearly unacceptable. To overcome this problem, only some portion of the page table is stored in the physical memory. When a process is executing, a part of the page table resides inside the main memory that includes the page table entry of the pages currently executing. Therefore, the page table is also subject to paging. The size of the page tables for different processes is governed by the computer architecture and amount of main memory available. Also, since the size of the page table is large, we use multiple levels (a hierarchy) of page tables for size reduction. In a typical computer, the root page table (page directory) is an array of addresses and each entry points to a page table. The index into the root page table is determined by the highest order bits of the virtual memory address. The next bits in the virtual address are used to index the page table that is denoted by the first index. Typically, the maximum length of a page table is restricted to the size of one page and the Pentium processor uses this approach.

MEMORY MANAGEMENT UNIT

To implement virtual memory and map the virtual address to a physical address, a memory management module is necessary for the translation of addresses. The address translation hardware is often known as an *MMU* (Memory Management Unit). In some cases, the size of a physical and virtual memory space is the same, as in uClinux. In this case, when the processor accesses the main memory, the locations on the main memory never change. That is, an example memory address 00001 is always the same physical location inside the main memory.

With an MMU, memory addresses go through a translation step prior to each memory access. An intuitive virtual address to physical address is shown in Fig. 1, which means that a virtual address 00001 might be directed to physical address 24568 at one time, and to physical address 87006 at another time. The overhead of individually tracking the virtual address to physical address translations for billions of bytes of memory would be too great and costly. Instead, the MMU divides the main memory into *pages* or *segments*. These are contiguous sections of memory of a fixed size or variable size, respectively, which are handled by the MMU as single entities. Keeping track of these sections and their address translations is necessary in implementing virtual memory. For example, if the processor had stored data to the main memory in the past and wants to use that data again, it would have to know where the data were stored on the physical memory by performing the address translations from virtual to physical.

Similar to the cache, which stores recently accessed data from the main memory for the CPU, the translation lookaside buffer (TLB) is a cache/buffer that assists the virtual memory page table scheme (Fig. 2). The TLB contains the page table entries that were recently used. Given a virtual address, the processor will first examine the TLB rather than access the page table, which is typically larger than the TLB and may lead to performance degradation. If the desired page table entry is present in the TLB (called TLB hit), then the frame number is retrieved and the physical address is acquired. If the desired entry is not in the TLB (called TLB miss), the processor or the OS then accesses the page table.

PAGE SIZE

Several factors need to be considered when deciding on the size of the pages. One is internal fragmentation. This problem occurs when the page has unused portions. Thus, if the page size is small, there will be less internal fragmentation. Reducing the page size will, therefore, use the main memory more efficiently. However, smaller page sizes mean larger number of pages required for processes, and larger number of pages means larger page tables. Large page tables may mean that some part of the page table that maintains pages that are in the main memory can be in the virtual memory, and therefore, there can be a lot of

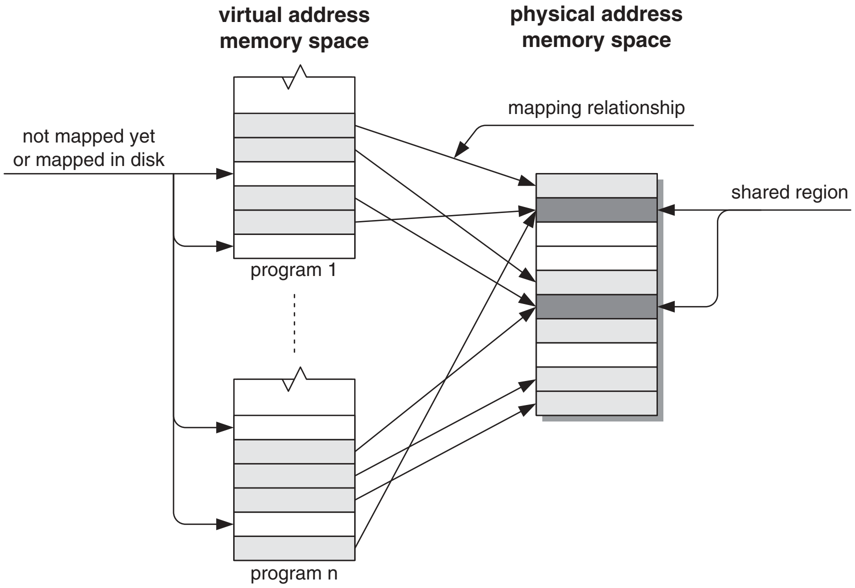

virtual address memory space
physical address memory space
mapping relationship
not mapped yet or mapped in disk
shared region
program 1
program n

Figure 1. An intuitive relationship between the physical memory space and the virtual memory space where the memory management unit (MMU) determines the mapping relationship to translate the virtual memory addresses to physical memory addresses.

page faults occurring even when accessing the page table. The effect of page size on the page fault rate complicates the problem even more. If the page size is small, then a relatively large number of pages will be available for a process in the main memory. Thus, the page fault rate should be low. As the page size is increased, the individual pages will contain addresses that are further apart, weakening the principle of locality and the page fault rate begins to rise. And as the page size becomes as large as the process itself, the page fault rate decreases until the rate is 0 when the page size is the size of the entire process. The number of frames in the main memory allocated to a process also affects the page fault rate. Finally, the size of the main memory and the application or process size need to be considered when determining the page size.

SEGMENTATION

Segmentation allows a programmer to view memory as consisting of multiple address spaces or segments (1). This scheme is different from paging in that although pages are fixed in size, segments may vary in size, can be unequal, and the size can be dynamically changed. The virtual address for segmentation will consist of a segment number and an offset similar to pages. The segment table, however, is different in that there is the segment base address and a length associated with the segment. There are several advantages to the programmer over a nonsegmented address space. Segmentation simplifies the handling of dynamically changing data structures, sharing among processes can be easily done, and the programmer can

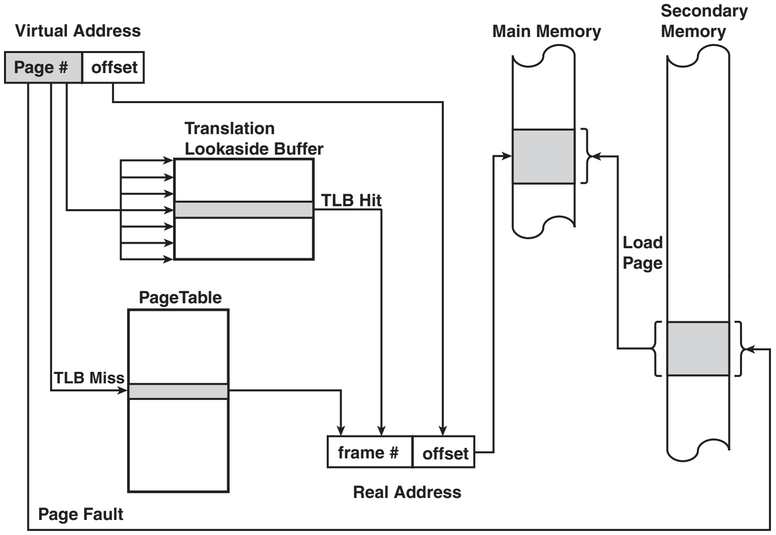


Figure 2. An intuitive representation of how the translation lookaside buffer (TLB) is used when accessing a page.

assign access privileges that can effectively protect a segment.

PLACEMENT

A placement policy determines where in the physical memory a page or a segment is stored. In a segmentation system, the placement policy is an important design issue and policies such as best-fit and first-fit methods are used. The best-fit scheme is a data placement scheme that places fetched data in an area where it would be "best" fit with smallest gaps between placements as possible. The first-fit scheme is a simpler method where the fetched data are placed in a large enough space that is discovered first. For a paging system, placement is usually irrelevant because the pages are usually of equal size; therefore, data can be placed in any page. However, there can be issues on which page to put data according to how the main memory is scanned for data.

REPLACEMENT

When all of the frames in the main memory are occupied and a page needs to be loaded into the memory to satisfy a page fault, this new page must replace an existing page in the main memory. The replacement policy determines which page in the memory can be replaced with the new page requested. There is one restriction on the replacement policy, and it is the frames that must not be replaced. These frames are used by the kernel of the operating system, I/O buffers, and other critical operations. Frames can be locked such that the replacement policy does not try to replace those frames. This can be achieved by introducing a lock bit or lock flag with each frame or include the frame into the page tables to indicate that the frame is occupied (or being executed) and must not be touched by the policy.

There are several basic algorithms for the replacement policy. These are the optimal, least recently used (LRU), first-in-first-out (FIFO), clock, and so on. The optimal policy selects the page that has the longest time to the next reference for replacement. Obviously, implementation of this policy is impossible because this policy assumes that the replacement module or the operating system has perfect knowledge of the future. The LRU strategy replaces the page that has not been referenced in the longest time. This scheme can be very costly to realize as each page needs a tag that indicates the last time it was referenced. For the FIFO policy, pages are replaced in a round-robin style. The clock policy combines the LRU and the FIFO methods. When a page is first loaded into a frame, in main memory, a use bit is set to 1 for that frame. Whenever the frame is referenced again, the bit is set to 1. When there is need to replace a frame, it starts as a round-robin-type scheme where a pointer goes around the frames to find the first frame with the use bit being 0. As the pointer encounters frames with use bit 1, it passes to the next frame but the use bit is set to 0. Therefore, if all of the pages have the use bit 1, then the very first frame that was checked for this use bit will be the page that will be replaced.

IMPLEMENTATION DETAILS: CASE STUDY OF AE32000C/LINUX

Here we introduce a case study of a virtual memory system implementation describing the workings of the processor and the operating system. Detailed implementation depends on the architecture of an operating system and MMU of a processor, because the design space is determined by architecture parameters. To provide intuitive insights on the implementation, the implementation section is based on how a virtual memory system is implemented with the Linux operating system on an embedded microprocessor, AE32000C (2), where the MMU provides simple straightforward features. These features are as follows:

- Separated On-Chip Instruction/Data TLB, 4-Way Set Associative, 32-Entries per TLB
- 4-KB page size
- Physically addressed Instruction/Data cache
- 8-bit ASN (Address-space number)
- Software page table walking

Linux supports up to a three-leveled page table in order to reduce the amount of memory occupied by page table entries anticipating virtual memory space lager than 4 GB (3). However, AE32000C/Linux uses a two-leveled page table as shown in Fig. 3. The first level is a page directory table that contains 1K number of entries to reference the second-level tables. The second level is the actual page table that contains 1K entries to reference physical pages. Therefore, the page table covers 4 GB of virtual address space; 1K (first level table) × 1K (second level table) × 4 KB (page size) = 4 GB. Since the second-level table is created only when the corresponding page is to be accessed, we can efficiently save memory occupied by the tables.

Figure 4 shows the structure of the page table entry. The primary field is a physical page number that is used in address translation. The others are used for access control and access tracking of the page. For instance, Linux provides writing permission on a page by setting the writable field. When the content of a page is modified or referenced, each field is set or unset to be used when the time comes to replace pages efficiently.

Usually the virtual address space is divided into user spaces and a kernel space. Although the user spaces are private address spaces for each process, the kernel space is a shared space for all processes. Moreover, Linux maintains the kernel space by dividing into two segments: an identity-mapped kernel segment and a page-table-mapped kernel segment (4). The identity-mapped segment is the space for allowing a kernel to be easily addressed virtually or physically where a direct translation relationship exists between virtual address and physical address. Most kernel objects, including the page tables, reside in this segment to avoid the overhead of a page table or to reduce implementation complexity such as handling nested TLB miss from kernel space and page-table swapping out. The page-table-mapped kernel segment is for dynamically allocated kernel objects especially requiring continuous address space such as kernel modules.

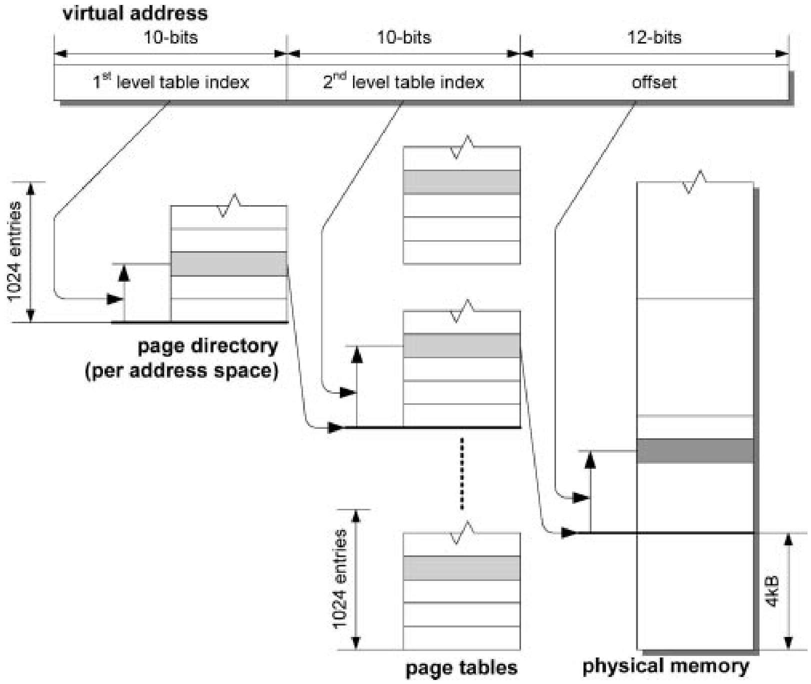


Figure 3. AE32000C/Linux page table.

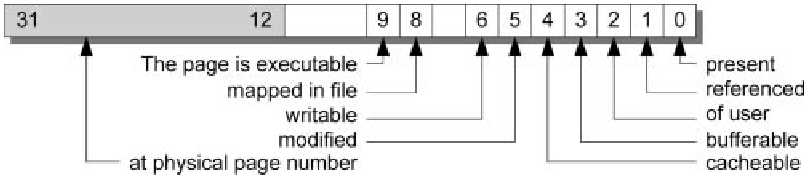


Figure 4. Structure of the page table entry.

In the case of AE32000C, the physical address of the SDRAM starts from 0xC0000000, which is the most popularly used start address of virtual kernel address space as shown in Fig. 5. Therefore, AE32000C/Linux sets the virtual address of the identity-mapped kernel segment the same as the physical address. This approach simplifies the kernel implementation and at the same time gets rid of any address translation mechanism for the segment.

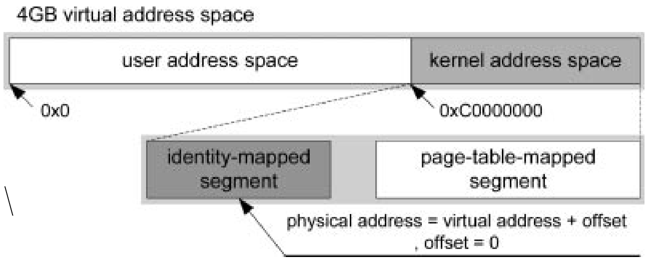

4GB virtual address space
user address space
kernel address space
0x0
0xC0000000
identity-mapped segment
page-table-mapped segment
physical address = virtual address + offset
, offset = 0

Figure 5. Structure of AE32000C/Linux address space.

Figure 6 depicts how the MMU translates a virtual address to a physical address. The MMU and the way of TLB management by the OS are highly dependent on the cache addressing scheme. Cache may be addressed by virtual address as well. The MMU for the cache does address translation only on a cache miss. The advantage of this scheme is fast cache access time due to no address translation on the cache hit. But, the drawback is the need for complicated cache coherence maintenance. For example, it should anticipate an alias/synonym problem since cache lines of several different virtual addresses, but of the same physical-address, can exist at the same instance. To alleviate this problem, the cache could contain tags as part of the physical address. But the cache size is restricted so that the cache index bits shall be part of page offset bits. The caches of AE32000C are physically addressed, and the MMU does virtual-to-physical address translation prior to every cache access. Although the drawback of this method is slower access time because it requires a TLB lookup before every cache access, the coherence maintenance is quite

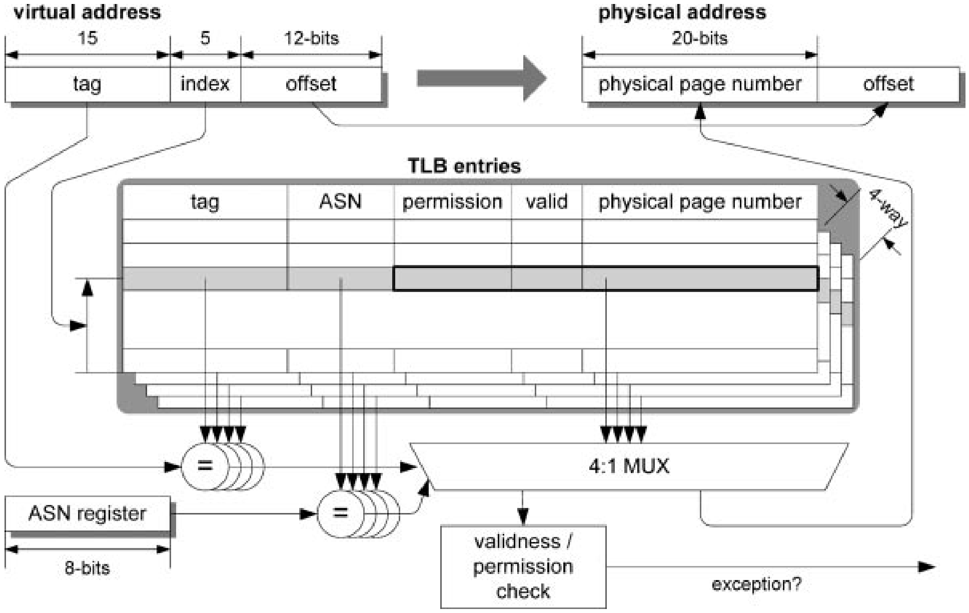


Figure 6. AE32000C MMU.

simple. To find the corresponding TLB entry for a virtual address, TLB entries are indexed by five bits of the address. From the indexed four entries, the MMU associatively finds out the right entry through matching tag field with the address and address-space number (ASN) with the current active address space. If the right entry is available, the MMU checks the validness of the entry, access permission for the corresponding page, and the presence of the page in the physical memory. Finally the physical address is obtained and then is used for accessing the cache memory, when all conditions are satisfied. Otherwise, an exception is generated in order to request the OS to handle it. As the MMU does not provide page table walking, the OS is responsible for the management of the TLB. Now, let's look at the OS implementation for TLB management.

Whenever no empty entries are available for a new translation, the TLB miss handler of the OS evicts an existing entry and inserts the new one with a carefully chosen TLB replacement policy. Popularly used policies are LRU (Least Recently Used) and NRU (Not Recently Used) for expecting recently used entry to be accessed in the near future in order to avoid frequent TLB misses. But, since it is difficult to implement such policies in software without any hardware help, AE32000C/Linux uses a round-robin/cyclic replacement policy for implementation simplicity.

The OS must maintain coherence between the page table and the TLB. In addition to the alias problem we discussed, the OS should be facilitated for supporting multiple address space. As an example, with the MMU that does not support multiple address space at a time, we should flush all the TLB entries whenever contexts are switched. To reduce the overhead from TLB flushing at the context switching, many processors, including AE32000C, provide address-space number (ASN) to keep TLB entries of multiple address space at a time (4). Each entry of TLB contains an 8-bit ASN field, and a the MMU has an ASN register indicating the address space of the current running process supporting up to 256 address spaces at a time. The ASN is extracted from the context information, which is platform-dependent. In the case of AE32000C/Linux, the context information is composed of an 8-bit ASN and a 24-bit ASN generation number. On every context switching, the validity of the context information of the process to be activated is checked by matching the ASN generation number with the current generation. The ASN is given in round-robin fashion for the context of old generation or a newly created context. When all ASNs are distributed for the current generation, the generation number increases and all TLB entries are flushed for coherence.

As mentioned, an exception is generated if there is no corresponding TLB entry for an access (TLB miss) or the access is not permitted on the page (access violation). On the exception from the MMU, the OS invokes an exception handler. The rest of this section describes how the handler works.

Figure 7 briefly illustrates an operation flow of the handler. There are four kinds of actions that the handler takes on an exception from the MMU. The first action is a TLB entry insertion/update where the exception is caused by a TLB miss, or when the page table entry is created or changed by other actions. If (*1*) MMU did not generate the exception due to access violation and the corresponding page table entry is already present, the exception was simply caused by absence of the corresponding TLB entry (i.e., TLB miss). Therefore, the handler inserts the TLB entry. Otherwise, (*2*) other actions may create a new page table entry or change the properties of an existing entry. In these cases, the handler inserts or updates the TLB entry. The second action is a copy-on-write for write access violation where the access is not illegal. All processes are created (i.e., forked) by their parent while inheriting memory space. The inheritance could be done by copying.

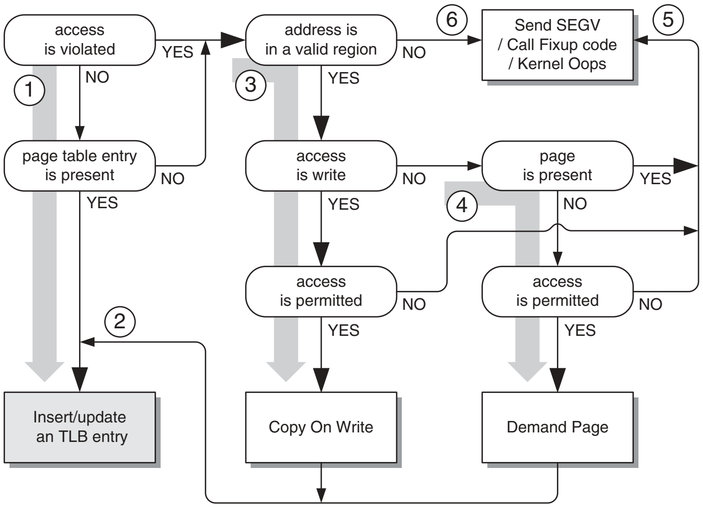


Figure 7. Operation flow of the exception handler.

But because child processes usually do not use or update all of the inherited pages, Linux avoids page copying by just sharing the pages unless there is a write access on them. The copying is done only on a write from either a parent or children. Hence, the access violation was caused by the purpose of this copy-on-write mechanism. To distinguish this violation from others, the handler checks (*3*) whether the write access violation was performed in a valid region and is actually permitted. The action, demanding page, is for a read access violation to allocate a physical page where there is no physical page corresponding to the virtual address because Linux allocates physical pages dynamically. A condition for triggering the demanding page is similar to that of the copy-on-write. But (*4*) the corresponding page table entry should not be present yet, because the presence of a page table entry implies that a physical page was already allocated. The last action is for accesses that (*5*) violate the permission or (*6*) reference an invalid memory region. If the access is from a user process (i.e., not kernel). Linux sends a segment violation signal (SEGV) to the process. If the kernel itself makes the violation, there is a handler (fixup code) to correct the access. The kernel sometimes accesses memory without checking its validness to reduce the overhead of checking code, because it is expected that the handler would be called if the access is illegal. The other exceptions are due to kernel errors. On the exception, Linux kills the current process and displays an error message. This is called Kernel Oops.

HISTORY

At the early stages of our computer era, the limited amount of physical memory caused a programmer to concentrate on a way to fit parts of a larger program or divide a large program into blocks and schedule the moves between the two levels of the memory hierarchy (i.e., between nonvolatile hard disks and main memory). The blocks are called pages or segments, and the movement operations are called swaps. The hard disks were slow; therefore, it was important to try and keep most of the executing applications in the fast main memory. A programmer's dream would be to have an unlimited amount of main memory, and thus, the concept of virtual memory was born (5).

It was obvious to the operating system designers in the early 1960s that automatic storage allocation could significantly unload the burden from the programmers. The first operating system design group to accomplish this was the Atlas team at the University of Manchester who, by 1959, had produced the first working prototype of a virtual memory system. They called it a *one-level storage system*. Their idea was to distinguish between "address" and "memory location." This distinction led them to three inventions: (*1*) a hardware that automatically translated each address generated by the processor to its current memory location; (*2*) demand paging, which is an interrupt mechanism triggered by the address translator that moved a missing page of data that is requested into the main memory; and (*3*) a replacement algorithm, a procedure to detect and move the least useful pages back to the secondary memory (5). In 1961, Burroughs released the B5000, the first commercial computer with virtual memory (6). It was based on segmentation, not paging.

Before virtual memory could be implemented in the mainstream operating systems, many models, experiments, and theories had to be developed to overcome numerous problems. Dynamic address translation required a specialized, expensive, and difficult-to-build hardware, which in early development made the access to the main memory slightly slow. More than anything, the consistent and robust address translation was a concern. By 1969, the debate over virtual memory for commercial computers was over (5). An IBM research team led by David Sayre showed that the virtual memory overlay system consistently worked better than the best manually controlled systems. Possibly the first minicomputer to introduce virtual memory was the

Norwegian NORD-1 (7). In 1972, virtual memory was announced for the IBM 370 series (8).

The difference between virtual memory implementations using pages or segments is not only about the fixed versus variable size memory division, respectively. In some systems, (e.g., Multics (9,10), the segmentation was actually visible to the user processes, as part of the semantics of a memory model. This is different from using pages, where the model is invisible to the process. In Multics, the segmentation method was used to provide a single-level virtual memory model, in which there was no differentiation between process memory and the file system. The active address space of a process consisted of only a list of variable segments (files) that were mapped into its potential address space, both code and data (9). This also worked when different processes mapped the same file into different places in their private address spaces (10). Today, most computer systems use virtual memory except for a few supercomputers and embedded CPUs and older personal computers (8).

REFERENCES

1. W. Stallings, *Operating Systems: Internals and Design Principles*, 5th ed., Englewood cliffs, NJ: Prentice Hall.
2. Advance Digital Chips Inc. Available: http://www.adc.co.kr
3. D. P. Bovet and M. Cesati, *Understanding the LINUX KERNEL*, 2nd ed. O'Reilly.
4. S. Eranian, D. Mosberger, and B. Perens, *The IA-64 Linux Kernel: Design and Implementation*. Englewood Cliffs, NJ: Prentice Hall PTR.
5. P. Denning, Before memory was virtual, *In the Beginning: Recollections of Software Pioneers*, 1997.
6. H. Cragon, *Memory Systems and Pipelined Processors*. Jones and Bartlett Publishers, 1996, pp. 113.
7. Norsk Data Annual Report 1982, ND Publications, April 6, 1983.
8. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann.
9. A. Bensoussan and C. T. Clingen, The multics virtual memory: Concepts and design, *Commun. ACM* **15**(5): 308–318, May 1972.
10. E. I. Organick, *The Multics System: An Examination of Its Structure*. MIT Press, 1972.

FURTHER READING

A. Silberschatz, P. B. Galvin, and G. Gangne, *Operating System Principles*, 7th ed. New York: Wiley.

M. Hailperin, *Operating Systems and Middleware*. Hailperin.

SEOK JOONG HWANG
SEON WOOK KIM
JONG-KOOK KIM
Korea University
Seoul, Korea

VLSI CIRCUIT LAYOUT

INTRODUCTION

The significant achievement of integrated circuits (ICs) in the past half a century has driven the current revolution of computing and communication technologies. The applications of ICs have penetrated everywhere at work and at home, such as computer networking, switching systems, communication systems, vehicles, household appliances, and so on. As the IC technology itself has been experiencing rapid evolution with the number of transistors on a single chip growing from a few to over hundreds of million, the design and fabrication of very-large scale integration (VLSI) chips have developed beyond the capabilities of any number of humans without computer support. Thus, the trend toward automation will accelerate as improved circuit fabrication technologies permit higher levels of integration and as more powerful computers allow more sophisticated tools.

Integrated circuits consist of several electronic components, built by layering several different materials in a well-defined fashion on a silicon base called a *wafer*. The VLSI design cycle starts with a formal specification of a VLSI chip, follows a series of steps, and eventually produces a packaged chip. A typical design flow is depicted in Fig. 1. The first step is to lay down the *design specification*, i.e., a high-level presentation of the system. The considered factors in this stage include functionality, performance, physical dimensions, design techniques, and fabrication technology. As the second step, the *architectural design* is to determine the basic architecture of the system. Its output is a micro-architectural specification, by which performance, power, and die size of the design can be predicted. Then in the *behavioral design* step, the main functional units of the system and the requirements of interconnects between the units are identified. The system behavior is specified in terms of input, output, and timing of each unit without considering the internal structure. The output of the behavioral design is usually a timing diagram or other relationships between units. Then the design process comes to *logic design* and *circuit design*. In logic design, the control flow, word widths, register allocation, arithmetic operations, and logic operations are derived and verified by means of a Hardware Description Language (HDL). The purpose of the circuit design is to develop a circuit representation based on the output of the logic design by taking into account the speed and power requirements of the system.

Physical design is a process of converting the behavioral or structural representations from the previous phases into a geometric description, called *layout*, which is used in the fabrication of the system. A layout consists of a set of planar geometric shapes in several layers. It is created by converting each logic component (blocks, cells, gates, and transistors) into a geometric representation (specific shapes in multiple layers), which performs the intended logic function. Interconnects between different components are also expressed as geometric patterns in multiple layers. The exact details of the layout also depend on design rules, which are guidelines based on the limitations of the fabrication process and the electrical properties of the fabrication materials. Physical design is a very complex process and therefore usually broken down into various subprocesses. After layout design and verification, the design is ready for fabrication. Layout data are converted into photolithographic masks, one for each layer. Masks identify spaces on the wafer, where certain materials need to be deposited, diffused, or even removed. Finally, the wafer is fabricated and diced into individual chips in a fabrication facility. Each chip is then packaged and tested to ensure that it properly meets all the design specifications and functions.

To meet the quick time-to-market and high-yield requirements, restricted models and design styles are applied to reduce the complexity of physical design. The design styles can be classified to three categories: full-custom, semi-custom, and universal. In a full-custom layout, all circuitry and interconnect paths are designed. A circuit is partitioned into a collection of blocks according to certain criteria. The process is performed hierarchically, and thus, the full-custom design has several levels of hierarchy. Different blocks of any size can be placed at any location on a chip as long as all the blocks are non-overlapping. In a semi-custom layout such as standard cell and gate array, a library of predesigned cells is available and some parts of a circuit are predesigned/placed on a specific location of a chip. Standard cell layout consists of rectangular cells of the same height, whereas all the cells in the gate array layout are identical gates or cells. In the universal layout design style such as field programmable gate array (FPGA), the design is somewhat fixed and the designer programs the interconnections.

The choice of a layout style depends on many factors, including type of chip, cost, and time-to-market. Full-custom layout is a preferred style for complex and mass-produced chips, since the time required to produce a highly optimized layout can be justified. On the other hand, to design an application-specific IC (ASIC), a semi-custom layout style is usually preferred. Further more, FPGA can dramatically reduce manufacturing turn-around time and cost for low-volume production. Table 1 summarizes the differences of distinct layout design styles in multiple categories.

Physical design is an extremely tedious and error-prone process. VLSI physical design automation is essentially the research, development, and productization of algorithms and data structures related to the physical design process. Its objective is to investigate optimal arrangements of components on a plane (or in three dimensions) and efficient interconnection schemes between components to lower cost and improve yield without compromising

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

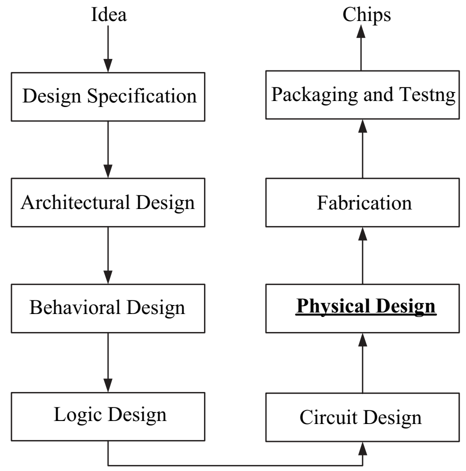


Figure 1. A typical design flow of VLSI systems.

functionality and performance. In the following discussion, general physical design methodology mainly for digital VLSI systems is first discussed. Three major intermediate stages (i.e., partitioning, floorplanning and placement, and routing) in a typical physical design process are described one after another. Then the physical design techniques/features for the advanced technology and analog ICs are outlined. Finally, some emerging topics regarding the modern VLSI layout are presented.

PHYSICAL DESIGN METHODOLOGY

Layout is a process of translating schematic symbols into their physical representations (1). To design a large circuit, it is necessary to understand the layout of simple gates, which are the basic building blocks of any circuit. As an example, the layout of a CMOS inverter is depicted in Fig. 2. The inverter consists of two polysilicon (*poly*) regions overhanging a path in the diffusion layer that runs between VDD and GND. This forms pMos (within the region covered by the Nwell layer) and nMos, which act as pull-up and pull-down transistors, respectively. The transistors are shown by hatched regions in Fig. 2. The inverter input A is connected to the poly that forms the gates of both transistors, whereas the output B on the metal layer is connected to the source/drain terminals of pMos and nMos transistors. VDD and GND are laid out in a metal layer, and contacts are used to connect metal and diffusion layers.

For the physical design process of a VLSI circuit, the input is circuit netlist and the output is circuit layout as depicted in Fig. 3. The physical design automation is typically solved in a hierarchical framework. Each stage should be optimized, while making the problem manageable for subsequent stages. Typically, the following subproblems are considered:

1. Partitioning: The objective is to divide a circuit into smaller parts so that the size of each part is within prescribed ranges and the number of interconnections between parts is minimized.
2. Floorplanning and Placement: Floorplanning is the determination of the approximate location of each block in a rectangular chip area. The shape of each block and the location of the pins on the boundary of each block may also be determined in this phase. When each block is fixed, i.e., fixed shape and fixed pins, placement determines the best exact position for each block.
3. Routing: The objective is to complete the interconnections between blocks according to the specified netlist.
4. Compaction: As a postprocessing phase, compaction is simply the task of compressing the layout in all directions such that the total area is reduced.
5. Verification: It is the testing of a layout to determine whether it satisfies design and layout rules. Design rule checking (DRC) is a process that verifies that all geometric patterns meet the design rules imposed by the fabrication process.

Figure 3 shows the physical design flow with emphasis on timing constraints for VLSI digital systems. The timing performance is estimated after floorplanning and placement and routing, and these steps are iterated if some interconnections fail to meet the timing requirements. After the layout is complete, parasitics can be extracted and accurate timing can be calculated. If some interconnections or components fail to meet their timing requirements, some or all phases of physical design need to be repeated.

Table 1. Comparison of Distinct Layout Design Styles

	Style			
	Full-custom	Standard cell	Gate array	FPGA
Cell size	variable	fixed height	fixed	fixed
Cell type	variable	variable	fixed	programmable
Cell placement	variable	in row	fixed	fixed
Interconnection	variable	variable	variable	programmable
Cost	high	medium	medium	low
Area	compact	compact to moderate	moderate	large
Performance	high	high to moderate	moderate	low
Fabrication layers	all	all	routing layers only	none

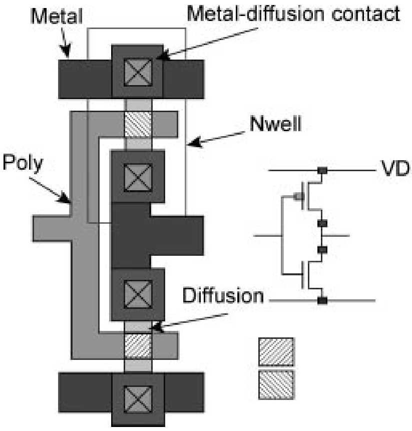


Figure 2. Layout of a CMOS inverter.

PARTITIONING

In VLSI physical design, efficient handling of any complex system necessitates decomposition of the system into a set of smaller subsystems of manageable size. This process is called *partitioning*. If a system is divided into k partitions, the problem is called *k-way partitioning*. Two-way partitioning is called *bipartitioning*. In particular, if the two partitions have the same size, this bipartitioning problem is called *bisectioning*. The partitioning can be conducted in a hierarchical manner until each transformed subsystem is tractable. At any level of partitioning, the input to the partitioning algorithm is a set of components and a netlist. The output is a set of subcircuits and the affiliated terminals for interconnections between subcircuits.

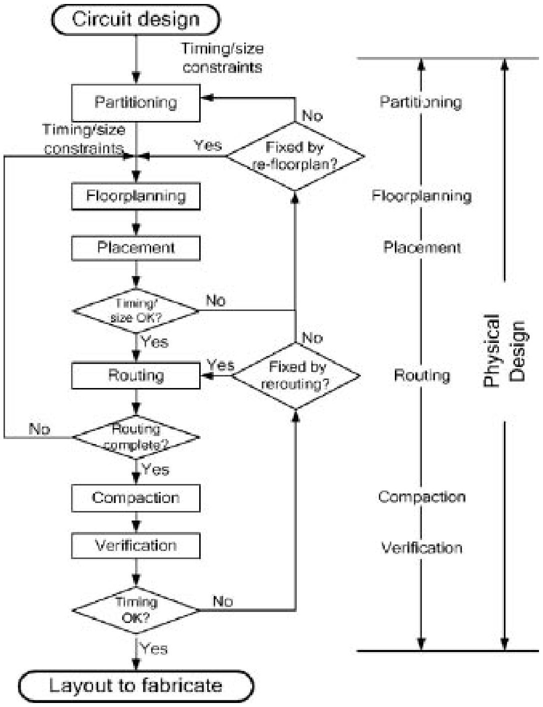


Figure 3. Complete physical design flow.

Problem Formulation

The partitioning problem can be modeled with the aid of hypergraph. Let $G = (V, E)$ be a hypergraph, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ is a set of hyperedges. Each vertex represents a component. A hyperedge joins the vertices whenever the components corresponding to these vertices are connected. Thus, each hyperedge is a subset of the vertex set; i.e., $e_i \subseteq V, i = 1, 2, \ldots, m$. The area of each component is denoted as $a\ (v_i)$, $1 \leq i \leq n$. Fig. 4 depicts a hypergraph, where vertices from a to f represent circuit components, and three hyperedges (interconnecting a-b-c, c-d-e, and e-f, respectively) represent three distinct nets. The partitioning problem is to decompose V to $V_1, V_2, \ldots V_k$, where

$$V_i \cap V_j = \varnothing \quad (1 \leq i \leq n, 1 \leq j \leq n, i \neq j)$$
$$\cup_{i=1}^{k} V_i = V$$

Partition is also referred to as *cut*. The cost of partition is called *cut-size*, which is the number of hyperedges crossing the cut. Let c_{ij} be the cut-size between partitions V_i and V_j. Minimization of the number of interconnections between partitions is called the *mincut problem*, which is a very important objective function for any level partitioning algorithms. The number of interconnections between partitions represents the dependence of partitions. The least dependence of partitions is preferred so that the subsystem of one partition can be implemented without much concern of the other partitions. This strategy can also ease the following physical design stages, especially for complex systems. The objective function above can be stated as follows:

$$\sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}, (i \neq j) \quad \text{is minimized}$$

Another objective function might be the minimization of the maximum number of times a path crosses the partition boundaries. The constraints for the partitioning problem may include the following: area, terminal number, and partition number. The objective functions and constraints used in the partitioning problem vary depending on the

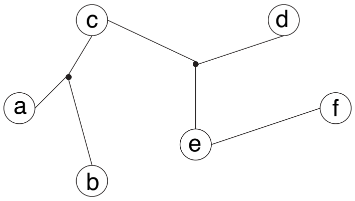


Figure 4. A hypergraph for modeling the partitioning problem.

partitioning level, the applied design style, and the specific problem.

Partitioning Algorithms

The partitioning problem is NP-complete. As a result, a variety of heuristic algorithms have been developed. The partitioning algorithms can be classified based on the process used for partitioning, including group migration algorithms, simulated annealing/evolution-based algorithms, and other partitioning algorithms.

The group migration algorithms start with some partitions, usually generated randomly, and then move components between partitions to improve the partitioning. They are deterministic algorithms, which can produce repeatable or deterministic solutions. One such algorithm is an iterative bipartition algorithm proposed by Kernighan and Lin (2). Based on graph, this algorithm starts with a random initial partition and then uses pairwise swapping of vertices between partitions, until no improvement is possible. Among a series of variant algorithms, Fiduccia and Mattheyses modified the Kernighan–Lin algorithm and reduced the time complexity to *O(t)*, where t is the number of terminals (3). The group migration algorithms are generally simple and efficient. However, the number of partitions, usually unknown when the partitioning process starts, has to be specified. And an algorithm used to find a minimum cut at one level may sacrifice the quality of cuts at the following levels.

Simulated annealing/evolution algorithms are the techniques used to solve general optimization problems, which include the partitioning problems. These techniques are especially useful when the solution space of the problem is not well understood. Simulated annealing is the simulation of crystal formation. The process starts with a random initial partitioning. An altered partitioning is generated by exchanging some elements between partitions. The resulting change in the cost, δ_c, is calculated. If $\delta_c < 0$ (representing lower energy), the move is accepted. If $\delta_c \geq 0$, the move is accepted with the probability of $e^{\frac{\delta_c}{kt}}$, where k is Boltzmann's constant and t is temperature. Thus, the probability of accepting an increased cost decreases with the decrease of temperature t, which allows the simulated annealing algorithm to climb out of local optimums in search of a global minimum. On the other hand, the simulated evolution algorithms simulate the biological process of evolution. The algorithm starts with an initial set of configurations, which is called *a population*. The simulated evolution algorithm is iterative, and each iteration is called a *generation*. The quality of generations is improved by using operators that imitate the biological events in the evolution process. For instance, the crossover operator is to generate offspring by combining schemata of two individuals at a time, whereas the mutation operator is to produce new offspring by causing incremental random changes. Simulated annealing/evolution algorithms allow a designer to come close to the optimal solution statistically, but the user needs to trade off quality and computation time.

Besides the group migration and simulated annealing/evolution methods, there are other partitioning algorithms such as the maximum-flow min-cut algorithm and eigenvector decomposition algorithm (4). The maximum-flow min-cut algorithm transforms the min-cut problem into the maximum flow problem. To separate a pair of nodes into two subsets, the minimum number of crossing edges is equal to the maximum amount of flow from one node to the other. Although this algorithm can find the optimal solution between any pair of vertices in a network, there is no constraint on the sizes of resultant subsets. The eigenvector decomposition algorithm finds an allocation by some metrics other than the graph structure. It requires the transformation of every multi-terminal net into several two-terminal nets first. The connections are represented in a matrix, and the eigenvectors of the matrix define the locations of all components and thus derive partitioning results.

FLOORPLANNING AND PLACEMENT

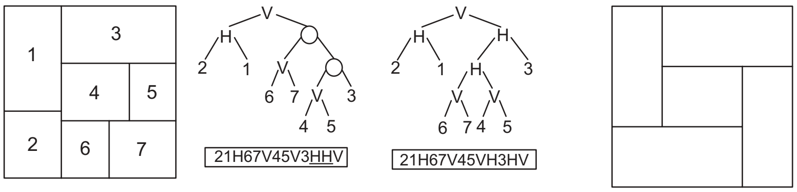
Once the partitioning phase is done, the next task is to assign a specific shape to each block and arrange the blocks on the wafer. This is done by floorplanning and placement phases. The blocks, whose dimensions are known, are called fixed blocks, whereas the blocks, whose dimension are yet to be determined, are called flexible blocks. The problem of assigning locations of any flexible blocks on a layout surface is called *floorplanning*, where the details, such as shapes of blocks, I/O pin positions, and so on., are not fixed yet. If all of the blocks are fixed, the positioning problem is called *placement*. Thus, the placement problem is a restricted version of the floorplanning problem.

Floorplanning

The floorplanning problem is to plan the positions and shapes of the blocks for optimizing the circuit performance. Some objectives, such as chip area, total wirelength, delay of critical path, routability, noise, heat dissipation, and so on., are normally considered in the floorplanning phase. Compared with the placement problem, floorplanning is more difficult. In floorplanning, several layout alternatives for each block are considered. Usually the blocks are assumed to be rectangular and the lengths and the widths of these blocks are determined besides their locations.

The input to the floorplanning problem is a set of blocks $(B_1, B_2, \ldots, B_n)$, the area of each block $(a_1, a_2, \ldots, a_n)$, possible shapes of each block, the number of terminals for each block, and the netlist. The aspect ratio of a block is the ratio of its width to its height. A_i^l and A_i^u (l ≤ z ≤ n) define the lower and upper bounds on the aspect ratio of each block. The floorplanning algorithm has to determine the width w_i and height h_i of each block B_i such that $A_i^l \leq \frac{h_i}{w_i} \leq A_i^u$. Finally the floorplanning algorithm has to generate a valid placement such that the area of the layout is minimized.

Floorplanning methods can be classified into two categories: slicing floorplanning and non-slicing floorplanning. A slicing floorplan is a floorplan that can be obtained by recursively partitioning a rectangle into two parts either by a vertical or horizontal line. The cut tree obtained by min-cut algorithm is known as *slicing tree*. A slicing tree is a binary tree in which each leaf represents a partition and



(a) slicing floorplan. (b) slicing tree. (c) slicing tree (skewed). (d) non-slicing floorplan.

Figure 5. A floorplan with slicing trees and a non-slicing floorplan.

each internal node represents a cut direction (or cut). A slicing floorplan is depicted in Fig. 5(a), and its corresponding slicing trees are shown in Fig. 5(b) and Fig. 5(c). Figure 5(d) shows a non-slicing floorplan.

Wong and Liu proposed a simulated annealing algorithm using a normalized Polish expression to tackle the slicing floorplanning problem (5). A Polish expression [as given in the blocks within Fig. 5(b) and Fig. 5(c)] is the postorder traversal of a slicing tree. By restricting a slicing tree [as depicted in Fig. 5(b) where one H node can have another H node as its son node on the right side] to a *skewed slicing tree* [i.e., no node and its son node on the right side in the slicing tree are the same as shown in Fig. 5(c)], a normalized Polish expression [i.e., no consecutive H's or V's as shown in Fig. 5(c)] can be obtained. And it can be proved that there is a 1–1 correspondence among slicing floorplan, skewed slicing tree, and normalized Polish expression. Thus, normalized Polish expressions can be used to represent slicing floorplans. And the slicing floorplan problem can be formulated as a state space search problem by defining a number of neighborhood structure move patterns within simulated-annealing optimization.

Compared with the slicing floorplan, the non-slicing floorplan is a more general form and thus much harder to handle. A conventional method for solving non-slicing floorplanning is mixed integer linear programming (6). This problem is formulated by a mathematical program, which includes a linear objective function and linear constraints. Each block is associated with four variables, including X/Y coordinates of the lower left corner and width and height. To keep blocks at non-overlap, linear inequalities can be set up to represent the location relationship of blocks. Finally the problem is solved with a linear programming (LP) package. However, because of the time-consuming feature of LP, practically this method can only solve small size problems. To explore more efficient solutions to the non-slicing floorplan, recently extensive research work has been conducted on non-slicing representations. These include sequence pair, bounded slicing grid, O-tree, B*-tree, corner block list, transitive closure graph, and so on. (7). Experimental results show that these non-slicing representations can generate efficient floorplan results.

Placement

The placement phase is very crucial in the overall physical design cycle. A problematic placed layout cannot be improved by high-quality routing in terms of area and performance. The placement problem can be formulated as follows. Let $B_1, B_2, \ldots, B_n$ be the blocks to be placed on a chip. Each $B_i 1 \leq i \leq n$, is associated with a height h_i and a width w_i. Let $N = \{N_1, N_2, \ldots, N_m\}$ be the set of nets representing the interconnection between different blocks. Let $Q = \{Q_1, Q_2, \ldots, Q_k\}$ represent rectangular empty areas allocated for routing between blocks. Let L_i denote the estimated length of net $N_u 1 \leq i \leq m$.

The placement problem is to find rectangles for each of these blocks on the plane denoted by $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ such that (*1*) each block can be placed in its corresponding rectangle, that is, R_i has width w_i and height h_i; (*2*) no two rectangles overlap, that is, $R_i \cap R_j = \varnothing, \quad 1 \leq i \leq j \leq n$; (*3*) placement is routable, that is, Q_j, $1 \leq j \leq, k$ is sufficient to route all the nets; (*4*) the total area of the rectangle bounding $\Re$ and Q is minimized; and (*5*) the total wirelength is minimized, that is, $\sum_{i=1}^{m} L_i$ is minimized. In the high-performance-circuits, the length of the longest net (i.e., $\max\{L_i | i = 1, \ldots, m\}$) has to be minimized. A few popular methods for wirelength estimation exist, such as half bounding box of the interconnection (also known as half-perimeter method), complete graph, minimum spanning tree, minimum Steiner tree, squared Euclidean distance (squares of all pairwise terminal distances in a net using a quadratic cost function), and so on (8).

The general placement problem is NP-complete. Thus, the placement algorithms are generally heuristic in nature. Three types of the most popular techniques are used in the state-of-the-art placers (9): the simulated-annealing-based approach, the partitioning-based approach, and the analytical approach. Independent of the placement techniques used, most modern placers consist of three major steps: *(1)* global placement: determines the most desired position for each cell/block, which might results in cell/block overlaps; *(2)* cell legalization: removes the cell/block overlaps; and *(3)* detailed placement: refines the placement solution.

The simulated-annealing-based placement algorithm normally consists of two stages. At Stage 1, blocks are moved between different rows as well as within the same row. Block overlaps are allowed at this stage, and they will be removed at the second stage. When the temperature falls below a certain value, Stage 2 begins. At Stage 2, any overlaps are removed and only adjacent blocks are interchanged within the same row. The solution perturbations are based on certain predefined move patterns. The drawback of the simulated-annealing-based placement algorithm is the long running time.

Partition-based algorithms are designed to group closely connected blocks together. The idea is to repetitively divide a circuit into subcircuits such that the cut value is minimized. The placement region is partitioned by cutlines accordingly. Each subcircuit is assigned to one partition of the placement region. The partitioning methods normally apply the Fiduccia–Mattheyses algorithm or other variants with the multilevel capability. Some typical academic placement tools include Capo, Feng-shui, NTUplacer, and so on (9). The partition-based algorithms are normally fast in terms of execution time, but they are not stable.

Analytical placement algorithms represent the placement problem as an analytical mathematical problem. One example is quadratic placement algorithms (10). As sum of squared wire length is quadratic in the cell coordinates, the wirelength minimization problem can be formulated as a quadratic program. It can be proved that the quadratic program is convex; hence, a polynomial time solution can be derived. The force-directed method is another typical form of analytical placement algorithm (11). This method transforms the placement problem into the classic mechanics system of objects attached to springs. Circuit blocks are analogous to objects, whereas circuit nets can be seen as springs. If the net weights are treated as spring constants, an optimal placement is actually an equilibrium configuration of objects. To avoid overlapping of blocks, repulsive forces inversely proportional to distance and connections to the fixed I/O pins on the boundary of the placement region can be added. Compared with blind search of simulated annealing, the force directed methods use directions of forces to guide the search. Thus, it is usually much faster than simulated annealing. However, this method focuses on connections, not on shapes of blocks. As it is only a heuristic, an equilibrium configuration does not necessarily stand for a good placement.

ROUTING

Introduction to Routing

After placement, components are arranged on a plane and the task remains to insert the electrical interconnections among the components to make them function. The process of finding the geometric layout of all the nets is called *routing*. The input to the general routing problem is as follows: *(1)* netlist; *(2)* timing budget for nets, typically for critical nets only; *(3)* placement information including location of blocks, locations of pins on the block boundary, and location of I/O pins on the chip boundary; and *(4)* RC delay per unit length on each metal layer, as well as RC delay for each type of via.

The objective of the routing problem is dependent on the nature of the chip. For general-purpose chips, it is sufficient to minimize the total wire length, while completing all the interconnections. For high-performance chips, it is important to route each net such that it meets its timing budget. Usually routing involves special treatment of such nets as clock, power, and ground nets.

One approach to the general routing problem is called *Area Routing*, which is a single-phase routing technique. It routes one net at a time considering all the routing regions. However, because of the extremely large size of the modern VLSI chips, this technique is computationally infeasible for an entire chip and is typically used for specialized problems or smaller routing regions. The conventional approach is to divide the routing into two phases as represented in Fig. 6. The first phase is called *global routing*, which generates a loose route for each net. It assigns a list of routing regions to each net without specifying the actual geometric layout of wires. The second phase, which is called *detailed routing*, finds the actual geometric layout of each net within the assigned routing regions. Unlike global routing, which considers the entire layout, the detailed routing considers just one region at a time. The exact layout is produced for each wire segment assigned to a region, and vias are inserted to complete the layout. Both global routing and detailed routing are NP-complete.

Global Routing

The global routing consists of three distinct phases: region definition, region assignment, and pin assignment. The region definition is to partition the entire routing space into routing regions, which includes spaces between blocks and above blocks (due to over-the-cell routing). Each routing region has a capacity, which is the maximum number of nets that can pass through that region. The capacity of a region is a function of the design rules and dimensions of the routing regions and wires. As the second phase of global routing, region assignment is to identify the sequence of regions through which a net will be routed. This phase must take into account the timing budget of each net and the congestion of each routing region. After the region assignment, each net is assigned a pin on region boundaries. This phase is called pin assignment, which allows the regions to be somewhat independent. After global routing is complete, the output includes pin locations for each net on all the region boundaries it crosses. Using this information, the length of the net can be extracted and the corresponding delay can be estimated. If any net fails to meet its timing budget, it needs to be ripped-up or the global routing phase needs to be repeated.

The global routing problem is typically studied as a graph problem (12). As shown in Fig. 6, there are two kinds of approaches to solving the global routing problem: sequential and concurrent. In the sequential approach, nets are routed one by one. However, once a net has been routed, it may block other nets that are yet to be routed. As a result, this approach is very sensitive to the order in which the nets are considered for routing. Usually, the nets are sequenced according to their criticality, half perimeter of the bounding rectangle, and number of terminals.

As an important two-terminal routing algorithm, maze algorithms are used to find a path between a pair of points, called the source and the target, respectively, in a planar rectangular grid graph (13). The geometric regularity in the standard cell and gate array design style allow one to model the whole plane as a grid. The areas available for routing are represented as unblocked vertices, whereas the obstacles are represented as blocked vertices. The objective of a

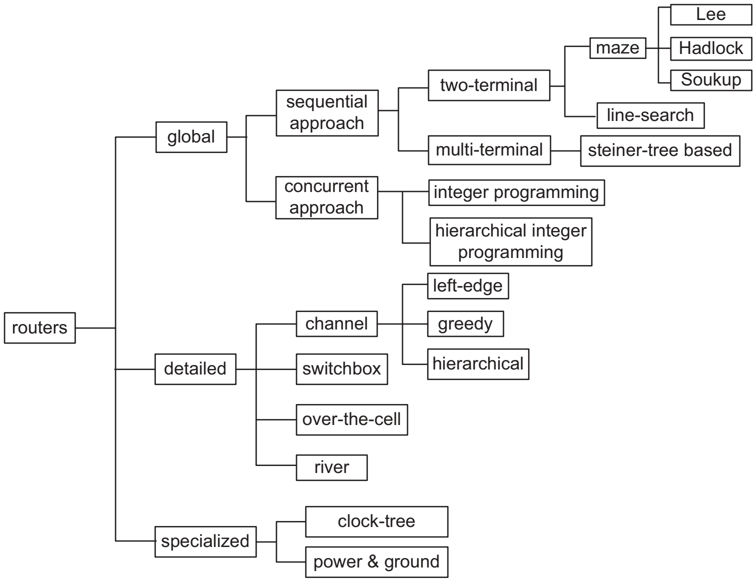


Figure 6. Classification of routing approaches.

maze routing algorithm is to find a path between the source and the target vertex without using any blocked vertex. The process of finding a path begins with the exploration phase, in which several paths start at the source and are expanded until one of them reaches the target. Once the target is reached, the vertices need to be retraced to the source to identify the path. The retrace phase can be easily implemented as long as the information about the parentage of each vertex is kept during the exploration phase.

The second global routing approach is concurrent approach, which avoids the ordering problem by considering the routing of all the nets simultaneously. The concurrent approach is computationally hard and no efficient polynomial algorithms are known even for two-terminal nets. As a result, integer programming methods as shown in Fig. 6 were proposed. The corresponding integer program is usually too large to be employed efficiently. Hence, hierarchical methods are employed to partition the problem into smaller subproblems, which can be handled by integer programming.

Detailed Routing

In a two-phase routing flow, detailed routing follows the global routing phase. The detailed router places the actual wire segments within the region indicated by the global router, thus completing the required connections between the terminals. The detailed routing has multiple forms, including channel routing, switch-box routing (2-D-switchbox and 3-D-switchbox), over-the-cell routing, and river routing. A channel is a rectangular area bounded by two opposite sides of blocks. A 2-D-switchbox is a rectangular area bounded all by block sides, whereas a 3-D-switchbox is a rectangular area with pins on all six sides. The detailed routing problem is usually solved incrementally. The ordering of the regions is determined by several factors, including the criticality of routing certain nets and the total number of nets passing through a region. Typically channel and 2-D-switchboxes should be routed first, since channels may expand. After channels and 2-D-switchboxes have been routed, the pin locations for 3-D-switchboxes are fixed and then their routing can be completed. After detailed routing is completed, exact wire geometry can be extracted and used to compute RC delays. The delay model not only considers the geometry (length, width, layer assignment, and vias) of a net, but also the relationship of this net with other nets. If some nets fail to meet their timing constraints, they need to be ripped-up or the detailed routing of the specific routing region needs to be repeated.

Different detailed routing strategies have been developed with a variety of objectives. A primary objective function of a detailed router is to meet the timing constraints for each net and to complete the routing of all the nets. Various secondary objective functions can also be considered, such as reducing total routing area, improving manufacturability by minimizing the number of vias and jogs, and improving performance by minimizing crosstalk between nets and delay for critical nets. Other objective functions may include minimization of the average or total length of a net or minimization of the number of vias per net.

The detailed routing algorithms can be classified in many different ways. Besides the classification method shown in Fig. 6, the algorithms can be categorized on the basis of the routing models used. Some routing algorithms use grid-based models, whereas others use the gridless models (14). The gridless models are more flexible as all the wires in a design need not have the same width. Another possible scheme is to classify the algorithms based on the strategy they use. Thus, we could have greedy routers, hierarchical routers, and so on. The number of layers used for routing can also be taken as a criterion to classify the algorithms. Thus, single-layer, two-layer, three-layer, and multilayer routing algorithms can be formed.

OTHER ISSUES IN VLSI PHYSICAL DESIGN

Design Techniques for the Advanced Technology

As the CMOS technology enters the deep submicron and nano design era, the interconnect density combined with the increase in wiring levels and the growth in chip size make the interconnect design become the most challenging area in the advanced technology. Wire delay and area tend to be the primary contributors to IC performance, power, and area. When using the conventional VLSI design, flow depicted in Fig. 1, the inaccuracy of wire modeling in the front-end design (including architectural design, behavioral design, logic design and circuit design) and the initial phases of the physical design often leads to long physical and timing closure periods to achieve a placement that is routable and meets timing requirements.

To overcome the inherent unpredictability of physical design in the preceding design process, floorplan-based design methodology, which takes into account the effect of floorplanning in the front-end design or initial phases of physical design, exhibits its promising applications especially in the advanced technology. For instance, chip floorplanning is performed continuously throughout the front-end process to discover physical design issues earlier and to impact register-transfer level (RTL) design decisions. Instead of relying on placement and routing results to correlate with prelayout models, floorplanning is deployed because of its relatively fast execution. Thus, an appropriate level of accuracy derived by floorplanning can guide the search in the front-end design process. This flow ensures rapid closure by characterizing netlist quality continuously during RTL implementation using iterative refinement of prelayout wire models.

In deep submicron and nano technology, the metal width tends to decrease with the length increase because of the complex system integration into single silicon. Therefore, the resistance along the power metal line increases. In addition, because of the nonlinear scaling of threshold voltage compared with power supply voltage in the advanced technology, IR-drop and clock skew issues become more crucial to the functionality of chip. As the design complexity exceeds millions of transistors on chip, it is reasonable to consider IR-drop and clock skew problems earlier in the design cycle. A floorplan-based planning methodology can be used for power network and clock network planning (15). From a floorplan, the power network size is determined at an early design stage based on the estimated power consumption. And even without a detailed gate-level netlist, clock interconnect sizing, the number and strength of clock buffers can be planned for balanced clock distribution. This early planning methodology at the full-chip level enables designers to fix the global interconnect issues by progressively refining the layout as the design proceeds for both the power and the clock distribution.

VLSI Physical Design of Analog ICs

The physical design methodologies discussed in the previous sections are mainly targeted to digital VLSI systems. Analog ICs have significantly different characteristics from digital circuits. These impose noticeable challenges on the physical design of analog circuits (16). In terms of the optimized parameters, digital ICs mainly consider the factors such as delay, area, power dissipation, and signal-to-noise ratio. On the other hand, for the analog circuits, there are numerous additional trade-offs among specifications, such as DC gain, bandwidth, offset voltage, 1/f noise, power supply rejection ratio, common mode rejection ratio, input swing, output swing, slew rate, and so on. Because of the considerable conflicts among these parameters, the optimization of analog ICs is a hard job requiring both knowledge and experience. In particular, the performance of analog circuits is strongly dependent on the detailed layout geometry and style.

Since the signal-to-noise ratio of digital circuits is large, over-the-cell routing is commonly used. However, this is not desirable in the analog layout because of the sensitive coupling effect of nets. Therefore, the structured custom layout approaches, which have been successful in speeding up the design of digital ICs, have difficulty in providing the same productivity gains to analog and mixed-signal ICs. A fixed library is unable to cover adequately the range of functions needed to construct arbitrary analog systems. Normally at least some fraction of an analog circuit on a mixed-signal chip must be full-custom designed at the considerable expense of time and skill.

In the view of layout, at least three distinctions exist between digital and analog ICs. First of all, analog blocks may have several alternative implementations, whereas digital blocks have only limited implementations. The transistors in digital circuits are always with the minimum size, whereas the transistors in analog circuits differ greatly in their width and length. Second, analog ICs are relatively small, which always include a limited number of blocks in all. In contrast, digital systems are very large (up to millions of transistors). Finally, lots of constraints are involved in the layout of analog circuits, such as parasitics, device matching, symmetric requirements, piezoelectric effects, and so on. For the layout of digital circuits, the constraints are limited.

The well-developed digital design tools/algorithms cannot be directly used for analog layout generation. In the following disussion, some typical methods used in the analog layout automation tools are described. Most of the existing analog layout automation tools apply a top-down design approach that takes the already optimized circuit netlist as the input and subsequently generates the layout (16). Meye zu Bexten et al. developed a rule-based analog layout system called ALSYN. The quality of the resulting layout greatly depends on the quality of the rule set, which is formulated in a context-dependent manner. The constructive technique proposed in Ref. 17 generates analog layouts by mapping features and special design constraints into an effective construction. A variety of interacting quality measures in the analog layout make this approach more suitable for layout generation at the device level. Algorithm-based tools, such as KOAN-ANAGRAM II (18), LAYLA (19), and ALADIN (20), can incorporate the analog layout knowledge into the design flow. These tools normally include complete placement and routing phases, which can handle special analog constraints. Recently, template-based layout automation, which retargets an existing analog layout to a different process or updated specification, was proposed (21). This technique, which is significant for supporting effective retargeting of analog intellectual properties, can produce good quality layouts in a reasonable amount of CPU time.

EMERGING TOPICS

The semiconductor industry has been driven by Moore's law for almost a half century. Device size miniaturization has allowed dense packing of transistors, whereas the improved transistor performance has led to significant increase in frequency. The VLSI physical design, which is closely related to performance requirements and high yield of ICs, has always faced new challenges posed by emerging technologies.

Among the grand challenges posted in the latest International Technology Roadmap for Semiconductors (ITRS, http://www.itrs.net/), layout design problems originating in state-of-the-art mask lithography and manufacturability take the lead. Of these, optical proximity correction of the mask and phase shift masking for enhancing sharpness are demanded. Dummy fill synthesis to achieve uniform density preferred for chemical-mechanical polishing needs to be realized.

Some research topics in deep submicron and nano regimes such as interconnect planning and synthesis, which involve interconnect architecture design, delay estimation, delay reduction by buffer insertion, and wire sizing, are attracting an increasing amount of attention. The resultant techniques are also pertinent to clock skew management in the clock-tree synthesis. Postplacement logic rewiring techniques for additional optimization of delay, power, and reliability are mandated.

Modern VLSI designers can integrate a whole system with large-scale logic/functional blocks on a single chip. This is called system-on-a-chip (SoC) design. The physical design for such a system needs to consider the integration of large-scale digital and analog (mixed-signal) circuit blocks, the design of system interconnections/buses, and the optimization of circuit performance, area, power consumption, and signal and power integrity. In addition, the highly competitive IC market requires faster design convergence, faster incremental design turnaround, and better silicon area utilization.

For modern VLSI circuits, more than half of the silicon area, power dissipation and delays are consumed by interconnects rather than devices. To tackle this issue, the 3D-Stacked IC (3D-SIC) technology aims to enable direct stacking of thinned ICs while realizing through-die area array interconnects (i.e., 3D vias) between them with an extremely high density. Thus, placing and wiring devices in the third dimension can promise higher clock rates, less power dissipation, and higher integration density. The physical design challenges of 3D-SIC include area, power and performance optimization, interconnect complexity, thermal distribution, process variations, signal integrity, and power/clock distribution.

Recently manufacturers of high-performance consumer electronics are turning to System-in-a-Package (SiP) design because it can provide a number of advantages over SoC. The SiP technology encompasses the packages with wire-bond die-stacks, or package-on-package 3D stacks. In addition to reduced cost, lower power, and higher performance, the SiP design offers the flexibility to mix RF and high-speed digital circuitry in the same package. It requires expert engineering talent in widely divergent fields (e.g., chip-package codesign).

As 3D packaging technology continues to advance, an emerging 3D system integration concept is to incorporate ultrathin films at microscale to embed both active and passive components (22). Besides the optimization of area, power and performance, the physical design of such a system needs to consider thermal distribution, mixed-signal substrate coupling and electromagnetic interference, power supply noise, and optical routing.

Furthermore, with the continued increase of on-chip packing density and the continued shrinking of device feature sizes due to the nanometer IC technologies, some other issues such as reliability (antenna effect, electrostatic discharge, electromigration, etc.), thermal, and yield (redundant via, process variation, etc.) will soon become first-order effects for VLSI physical design, as comparably important as the traditional design metrics—timing, power, signal/power integrity, and area. These effects have imposed tremendous challenges and opened many research opportunities to modern physical design.

BIBLIOGRAPHY

1. R. Baker, *CMOS: Circuit design, layout, and simulation*, revised 2nd ed., Hoboken, NJ: Wiley, 2007.
2. W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, *Bell Syst. Tech. J.*, **49**: 291–307, 1970.
3. C. Alpert and A. Kahng, Recent directions in netlist partitioning: A survey, *Integration: VLSI J.*, **19** (1,2): 1–81, 1995.
4. W. Mak and D. Wong, A fast hypergraph min-cut algorithm for circuit partitioning, *Integration: VLSI J.*, **30** (1): 1–11, 2000.
5. D. Wong and C. Liu, A new algorithm for floorplan design, *Proc. Design Automation Conference*, 1986, pp. 101–107.
6. P. Chen and E. Kuh, Floorplan sizing by linear programming approximation, *Proc. Design Automation Conference*, 2000, pp. 468–471.
7. T. Chen and Y. Chang, Packing floorplanning representations, *Physical Design Handbook*. in C. Alpert, S. Sapatnekar, D. Mehta, (eds.), Boca Raton, FL: CRC Press, 2007.
8. T. Yan and H. Murata, Fast wire length estimation by net bundling for block placement, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2006, pp. 172–178.
9. S. Adya, M. Yildiz, I. Markov, P. Villarrubia, P. Parakh, and P. Madden, Benchmarking for large-scale VLSI placement and beyond, *IEEE Trans. Comput.-Aided Design*, **23** (4): 472–488, 2004.

10. B. Yao, H. Chen, C. Cheng, N. Chou, L. Liu, and P. Suaris, Unified quadratic programming approach for mixed mode placement, *Proc. International Symposium on Physical Design*, 2005, 193–199.

11. T. Chan, J. Cong, and K. Sze, Multilevel generalized force-directed method for circuit placement, *Proc. International Symposium on Physical Design*, 2005, pp. and 185–192.

12. J. Hu, S. Sapatnekar, A survey on multi-net global routing for integrated circuits, *Integration: VLSI J.*, **31** (1): 1–49, 2001.

13. F. Mo, A. Tabbara, and R. Brayton, A force-directed maze router, *Proc. IEEE/ACM International Conference on Computer Aided Design*, 2001, pp. 404–407.

14. J. Cong, J. Fang, and Y. Zhang, Multilevel approach to full-chip gridless routing, *Proc. IEEE/ACM International Conference on Computer Aided Design*, 2001, pp. 396–403.

15. C. Liu and Y. Chang, Power/ground network and floorplan cosynthesis for fast design convergence, *IEEE Trans. Comput.-Aided Design*, **26** (4): 693–704, 2007.

16. G. Gielen and R. Rutenbar, Computer-aided design of analog and mixed-signal integrated circuits, *Proc. IEEE*, **88**: 1825–1852, 2000.

17. H. Mathias, J. Berger-Toussan, G. Jacquemod, F. Gaffiot, and M. Helley, FLAG: A flexible layout generator for analog MOS transistors, *IEEE J. Solid-State Circuits*, **33** (6): 896–903, 1998.

18. J. Cohn, D. Garrod, R. Rutenbar, and L. Carley, *Analog Device-Level Layout Automation*. Norwell, MA: Kluwer Academic Publishers, 1994.

19. K. Lampaert, G. Gielen, and W. Sansen, *Analog Layout Generation for Performance and Manufacturability*. Norwll, MA: Kluwer Academic Publishers, 1999.

20. L. Zhang, U. Kleine, and Y. Jiang, An automated design tool for analog layouts, *IEEE Trans. VLSI Syst.*, **14**, 881–894, 2006.

21. L. Zhang, N. Jangkrajarng, S. Bhattacharya, and C. Shi, Parasitic-aware optimization and retargeting of analog layouts: A symbolic template approach, *IEEE Trans. Computer-Aided Design*, **27** (5): 791–802, 2008.

22. S. Lim. Physical design for 3D system-on-package, *IEEE Des. Test Comput.*, **22** (6): 532–539, 2005.

FURTHER READING

N. Sherwani, *Algorithms for VLSI Physical Design Automation*, 3rd ed., Norwell, MA: Kluwer Academic Publishers, 1999.

A. Hastings, *The Art of Analog Layout*, 2nd ed., Englewood Cliffs, NJ: Pearson Prentice Hall, 2006.

S. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*, Singapore: World Scientific, 1999.

M. Sarrafzadeh and C. Wong, *An Introduction to VLSI Physical Design*, New York: McGraw-Hill, 1996.

LIHONG ZHANG
Memorial University of
Newfoundland
St. John's, Newfoundland
Canada

C

COLLABORATIVE VIRTUAL ENVIRONMENT: APPLICATIONS

INTRODUCTION

As the Internet evolves, people realize that a more natural interaction and communication via local area network (LAN), wide area network (WAN), or Internet can improve the efficiency of their social activities and the productivity of their daily work. Rapid advances in networking technology in the past decade have offered the potential of achieving real-time collaborative applications via network connection, in which people are able to interact with each other and the virtual surrounding environment in a way that they experience in real life. Collaboration is part of basic communication form in human society. If a virtual environment or a cyberspace is connected by computer network to allow real-time sharing and exchanging of information by multiple users, efficient remote collaboration can be realized. Collaborative Virtual Environment (CVE) has been developed for such purposes. With the capability to support multiple geographically dispersed human-to-human and human-to-machine communication and interaction in a shared virtual environment, CVE opens a door to a broader spectrum of potential applications in various areas ranging from academic study to collaborative engineering to military war-game simulation applications. There are many scenarios that can significantly benefit from the solution presented by CVE over simply noncollaborative virtual reality (VR) or 3-D workstation computer graphics. An important pioneer application in this area, the SIMNET networked tank simulators (1) demonstrate the feasibility of a CVE system used for military training in a distributed interactive simulation environment. Besides military applications, CVE has also been applied in medical or industrial team training, collaborative design and engineering, collaborative scientific visualization, and social activity and entertainment-like multiuser VR games. Recently, the game industry has also adopted the findings in this field of research and developed new types of games based on CVE technology. Examples of these developments are the multiplayer games (2,3). These games aim to support very large user bases with massive multiplayer online role-playing interactions. This type of game has proven to be quite popular and a way of generating revenue for application and content providers.

In the following sections of this article, we discuss the CVE applications in the following four main categories: military applications, social interaction and entertainment, collaborative learning, and collaborative design and group work.

MILITARY APPLICATIONS

A military force must rehearse its options if it is to be prepared for a complex battlefield environment. However, battle training with troops and equipment in the real world is both expensive and dangerous. Thus, virtual military training using a computer-generated synthetic environment has an important role to play. Training individual soldiers with stand-alone equipment simulators using VR technology has achieved great success for many years. As modern warfare gets more complicated, it often requires joint force of Army, Navy, Marine Corps, and Air Force to accomplish a military task. Thus, individual stand-alone virtual training is no longer sufficient. It requires collaborative efforts of different military services. It becomes the natural need for CVE technology to construct a multi-user shared battlefield that supports simulation of various military weapons, realistic battlefield effects, and intelligent agent-controlled virtual soldiers called a computer-generated force (CGF) (4). The SIMNET networked tank simulator was the first such military application. It allows armor companies and mechanized infantry companies to practice fighting in cooperation against some foe in a LAN or WAN. A trainee in a tank simulator can see and interact with the avatar of a Bradley Fighting Vehicle located hundreds of kilometers away in the SIMNET war simulation.

Karr et al. (4) categorized military training simulation by the degree of human interaction as “live, virtual, and constructive.” In “live” simulation, data feeds from computers replace sensor information from real equipment. The simulated sensor readings are matched as closely as possible with physical cues such as 3-D imagery and realistic audio. Each of the services has programs incorporating live simulation. The Air Force has the Contingency Theater Automated Planning System, sailors aboard ships train in a simulated world using the Battle Force Tactical Training System, and Army reservists can train in M-60 tanks using the Guardfist Training System. In “virtual” simulation, the second category, the work environment is also simulated. The real trainee tank gunner, for example, is in a physical mockup of the inside of a tank, and when he looks into the gun sight, he sees a television screen displaying a computer-generated image. In the Army, the crews for hundreds of tanks, Bradley Fighting Vehicles, and helicopters are trained each year in their vehicles. In live and virtual simulation, the trainee is known as the man in-the-loop. In the third and last category of simulation, “constructive,” the man departs from the loop, and battles can be fought with no or minimal human intervention. CGF systems belong in this category. Entities in the virtual world—soldiers, ships, tanks, aircraft, and the like—also can be “manually” controlled by commanders undergoing training, just as scale models were pushed around large map tables in the old days. (In practice, the officer in training directs his commands to a specialist operator, who implements them on workstations.)

Besides battlefield training, CVE has also been used for military clinical therapy treatment for postwar personnel as reported by Rizzo et al. (5). The virtual environment of various war scenarios when combined with real-time clinician input via the “Wizard of Oz” clinical interface is

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.
Copyright © 2008 John Wiley & Sons, Inc.

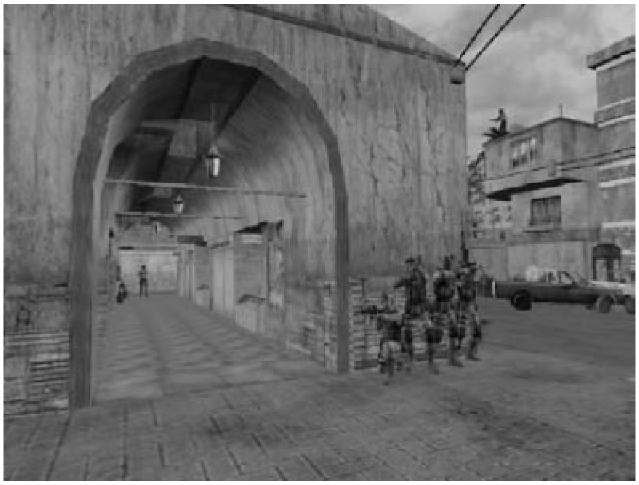


Figure 1. Flocking patrol (5).

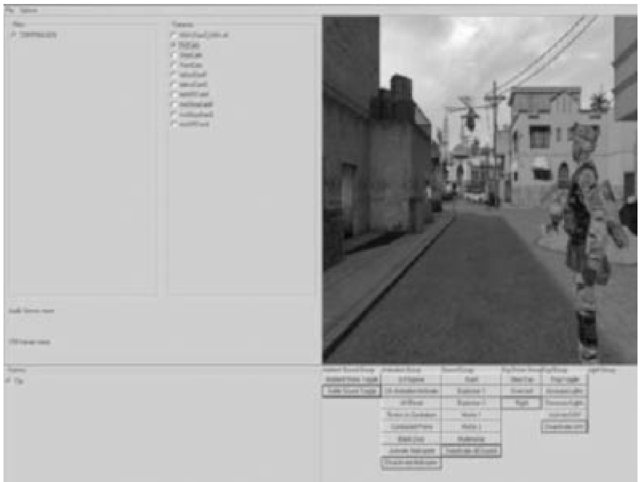


Figure 3. Clinical CVE postwar therapy interface (5).

envisioned to allow for the creation of a user experience that is specifically customized to the needs of the patient participating in treatment. It is designed such that clinical users can be teleported to specific scenario settings based on a determination as to which environment most closely matches the patient's needs, relevant to their individual combat-related experiences. These settings include city scenes, checkpoints, city building interiors, small rural villages, desert bases, and desert roads as illustrated in Figs. 1–3:

SOCIAL INTERACTION AND ENTERTAINMENT

CVE provides a virtual space to allow an electronic means of information exchange through network connections. In a CVE, a group of people can get together and communicate with each other, and they can gather information while freely moving about and engaging in a variety of activities. It provides a new information communications environment for people to exchange information, develop mutual understanding, interact freely with each other, and participate in entertainment activities (e.g., online multiplayer games, online digital community activities, etc.) in an enjoyable and efficient manner. It is a multiparticipant communication support platform that enables each participant's avatar to roam around freely in a 3-D virtual space. It also allows each avatar to engage in conversation with other avatars, with each avatar that may be mapped with facial image or live facial video image of the corresponding participant from a video camera mounted on his or her personal computer. This Feature enhances the feeling of presence in the shared social interaction and entertainment environment. Live audio and text chat in a shared 3-D environment engage online participants with easy communication and information exchange. For example, "There" see (http://www.there.com) (6) provides a persistent online environment for social interaction, as shown in Fig. 4. "There" shares many of the features of other online virtual environments and persistent role-playing games, such as Ultima Online (7), Everquest (2), Asheron's Call (8), World of Warcraft (9), and so on. Yet There is marketed as a "virtual getaway" for social interaction and exploration. Unlike online games, no overall goal exists to "There." Its

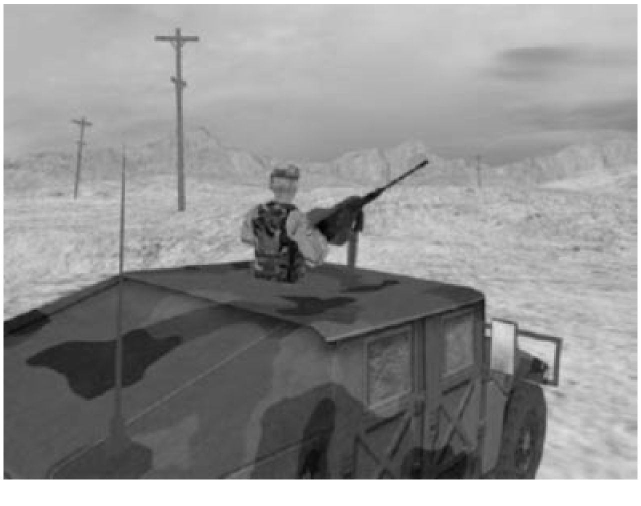
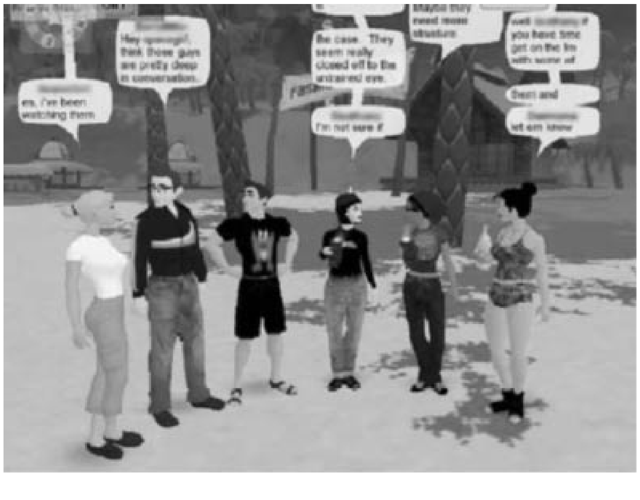


Figure 2. Desert road (5).



es, I've been watching them
Hey ... think those guys are pretty deep in conversation.
the case. They seem really closed off to the untrained eye.
I'm not sure if
maybe they need more structure
well ... if you have time get on the ... with some of
them and
let em know

Figure 4. Social interaction in "There" online environment (6).

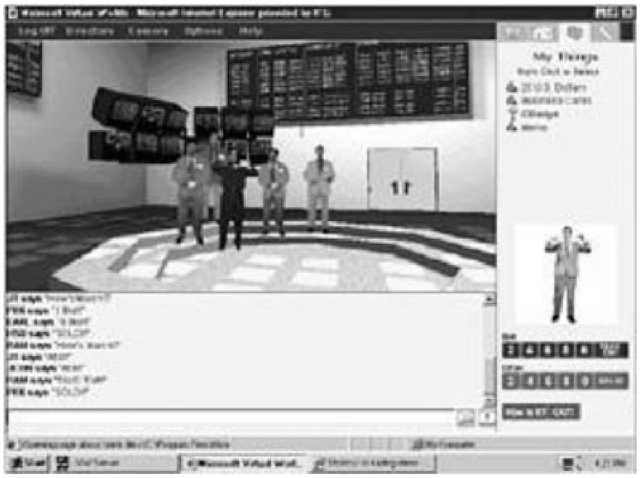


Figure 5. Virtual Worlds Platform-based-electronic mercantile exchange (10).

environment supports activities such as buggy races, paintball, flying jetpacks, treasure hunts, and even playing with virtual pets. Moreover, as with the real world, much attention is given to personal appearance, and one of the main activities in "There" is designing virtual clothes and selling them through in-game auctions. "There's" virtual world is augmented with support for instant messaging (both text and audio), forums, tools for organizing virtual "events," and forming groups. Specific attention has also been given to supporting social interactions in "There". Avatars can make emotional gestures, and chat is displayed in speech bubbles, within the game world, word by word, rather than in the complete lines of text displayed in instant messaging. Considerable attention has also been given to how avatars interact around objects. For example, unlike most games, the most commonly used camera angle is above the head and some distance back, which increases the field of view allowing easier interaction with objects that are close to the avatar. This view can also be turned with the mouse, an operation that is visible to others by the avatar turning its head.

Microsoft Research has also developed a Virtual Worlds Platform (10), which can be used for various social activity applications. The Electronic Mercantile Exchange, as shown in Fig. 5, is a prototype environment built by Acknowledge Systems using the Virtual Worlds Platform. It is an online environment for the financial services industry and a major futures exchange was consulted in the creation of this prototype. Among the features of the environment are a collaborative research center and a simulated trading floor that allows users to take part in a multi-user mock trading session, complete with 3-D graphics and audio. Used for marketing and education, the environment allows visitors to see, hear, and participate in simulations of these complex markets.

MusicWorld, as shown in Fig. 6, is another social interaction application built based on the Virtual World Platform. It is a graphical collaborative musical performance environment where avatars join together to explore a live, online "album" of multiple songs. Within each soundscape, avatars can mix sounds and compose their own musical patterns that are shared in real-time. Although all of the avatars can affect the music individually, all the changes they made are instantly updated across the network, guaranteeing that the music is truly collaborative and heard the same way for all participants. The space exemplifies the capabilities of V-Worlds to host a dynamically changing environment in which avatars could be expressive and change their world in real-time.

The Virtual World Platform has also been used to create a collaborative educational environment for exploring ancient Egyptian civilization and the crafts of archaeology and museum studies. It simulates the excavation of a new kingdom royal tomb in Egypt's Valley of the Kings. The tomb features a wide range of objects and artifacts representative of the late-eighteenth to early-nineteenth dynasties in ancient Egypt. It is intended as an educational simulation to be used by grade school students (age 9–12) in the classroom environment, and it is being developed in collaboration with teachers and museum educators. It features multi-user interaction in a 3-D virtual environment with rich textures, as shown in Fig. 7.

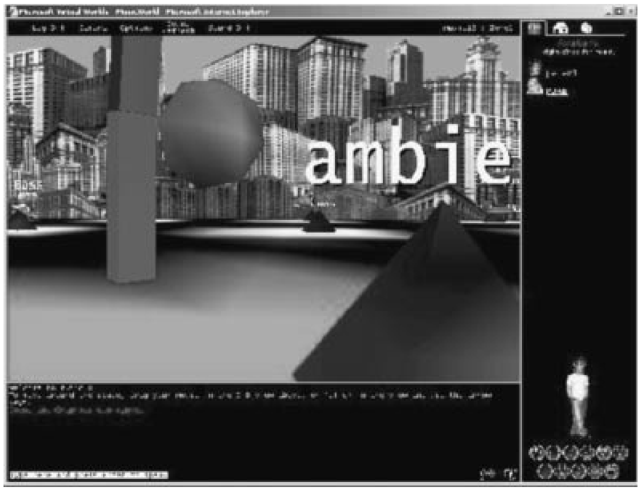


Figure 6. Virtual Worlds Platform-based MusicWorld (10).

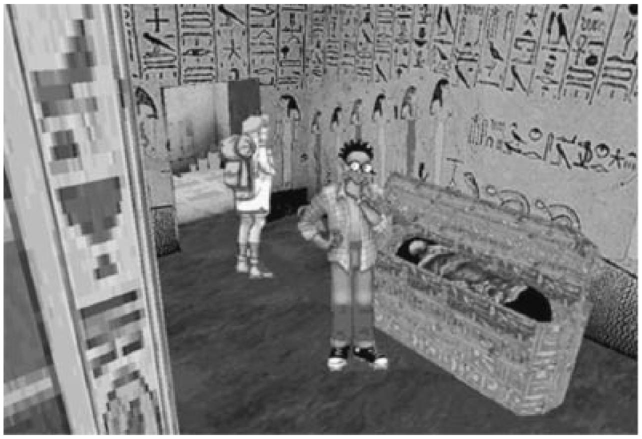


Figure 7. "Explorers of the Ancient World: Egypt" based on Virtual Worlds Platform (10).

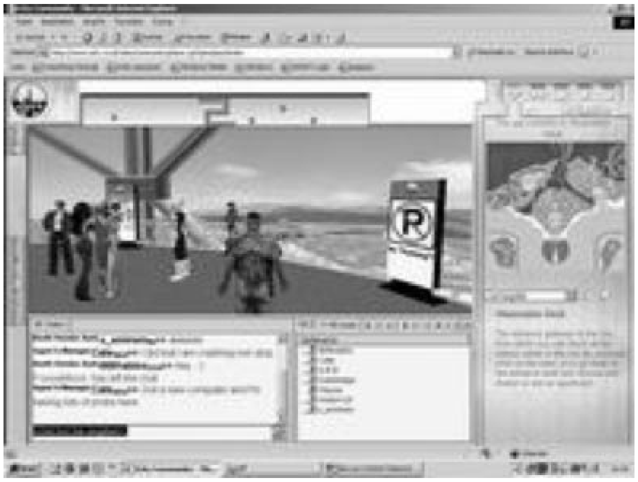


Figure 8. Blaxxun Shared Virtual Worlds (14).

Figure 9. MMORPG in Butterfly Grid (18).

There are also many other CVE-based social interaction applications, including Community Places (11) and Living Worlds (12) by Sony Research Laboratories, Diamond Park by Mitsubishi Electric Research Laboratories (13), Blaxxun Community Platform by Blaxxun Technologies (14), Virtual Playground by HITLab in Washington University (15), and VELVET in University of Ottawa (16). It provides facilities to allow users to create communities for various social interactions. For example, Blaxxun Community Platform allows users to set up homes, clubs, student dormitories, company offices, or other distinct locales on the Web, as shown in Fig. 8. These entities are divided into public and private spaces. For example, a community may have places like plazas, cafes, and offices where its members congregate. The community might also have neighborhoods devoted to members sharing special interests, like films, into which they can easily "homestead." Each member's home (complete with its own URL) can act as their personal communication center on the Web within which they can invite their friends, family, and other groups for scheduled or impromptu meetings and chats. Many present community members also attach this URL to their e-mail messages, which has the effect of promoting both the users' private home as well as the community URL. Homeowners receive all the tools required to maintain their places and can set up special access privileges to protect their privacy. The homes themselves can easily be built from one-click templates or designed more elaborately with objects like furniture, pets, and agent servants. Homes have all of the communication functionality delivered by the community manager.

MMORPG

Massively Multiplayer Online Role-Playing Games (MMORPGs) is another CVE application area. Recent developments on persistent MMORPGs, such as Everquest (2), allow game players to play in a persistent shared game world. It is a CVE type of game, which maintains a consistent game world for the game players so that they are given the feeling of playing in the same 3-D game scene while playing different game roles in a collaborative manner. In 2003, IBM and Butterfly launched a new network gaming environment for Sony Computer Entertainment, Inc.'s PlayStation2 that enables online video game providers to reliably deliver state-of-the-art games to millions of concurrent users (17). Traditionally, online video games have segmented players onto separate servers, limiting the number that could interact and creating reliability and support obstacles. In the first generation of online games, when one server is down, overloaded, or patches are being installed, game play comes to a halt. With Butterfly's grid technology, the server interaction is completely transparent and seamless to the user, delivering a resilient gaming infrastructure in which servers can be added or replaced without interrupting game play. In the Butterfly grid platform, game players can log on to the Grid through either a video game console, PC, set-top box, or mobile device running Butterfly Grid client software. During the course of a game, the Butterfly Grid divides the world into a series of mutually exclusive sectors known as "locales," each of which is assigned to a specific server. In the course of a game, a player may encounter areas of excessive activity within a locale—caused by factors internal to the game (such as a battle triggered by artificial intelligence systems) or simply large numbers of users—which increase the use of that locale's server. In such an instance, the Grid's "heartbeat monitoring" feature will flag the server as overutilized and move the high-activity locale to a less-utilized server on the Grid. Similarly, in the event a server goes down, game play is automatically and seamlessly routed to the nearest optimal server in the Grid using the same resource monitoring feature. Figure 9 shows a sample game screen from the Butterfly Grid platform.

COLLABORATIVE LEARNING AND TRAINING

Visualization has been used to enhance human understanding of conceptual work, from describing earth magnetic field (e.g., intensity and direction), to evaluating fighter jet design before prototyping, to aerodynamic study, to understanding of abstract fluid dynamic computation, to visualizing complex molecular structure. 3-D images bring better visualization of knowledge/information than any 2-D image does, whereas interactivity provided by a 3-D virtual environment makes the learning experience closer to real life than any animation technique offers. There is no dispute over the role of visualization in enhancing human understanding of abstract and complex concepts.

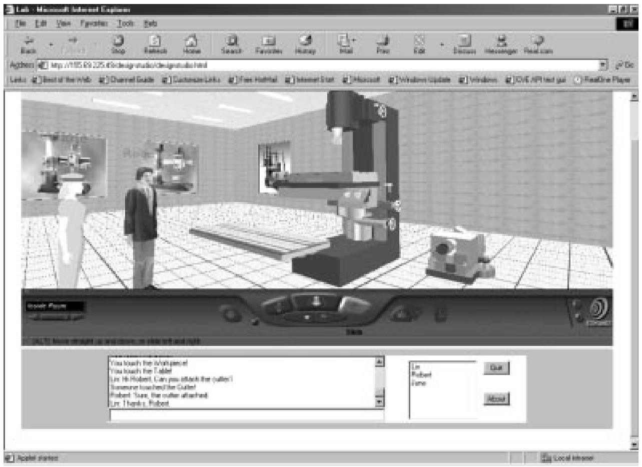
However, learning is contextualized in a social setting that may involve verbal interaction, collective decision making, conflict resolutions, peer teaching, and other group learning situations that are characteristic of a classroom setting (19). An engaging learning environment would stimulate learning interests and enhance students' knowledge acquiring performance. With CVE, the learning performance can be improved by providing intelligent and interactive learning contents in an engaging environment with multi-user participation and collaboration. A CVE-based learning system supports active exploratory and discovery learning (i.e., "learning by doing"), which is one of the most effective ways of learning. This method encourages the students' active thinking in every step of the learning process. One of the most important advantages is the computer supported collaborative learning that can model learning activities and interactions close to the traditional classroom, which is not possible in stand-alone visualization-based learning methods.

The NICE garden, as shown in Fig. 10, is one such CVE-based learning application. It was originally designed as an environment for young children to learn about the effects of sunlight and rainfall on plants, the growth of weeds, the ability to recycle dead vegetation, and similar simple biological concepts that are a part of the lifecycle of a garden. As these concepts can be experienced by most children in a real garden, the NICE garden provides its users with tools that allow its exploration from multiple different perspectives. In addition to planting, growing, and picking vegetables and flowers, the children have the ability to shrink down and walk beneath the surface of the soil to observe the roots of their plants or to meet other underground dwellers. They can also leap high up in the air, climb over objects, factor time, and experience first hand the effects of sunlight and rainfall by controlling the environmental variables that cause them. NICE supports real-time distributed collaboration. Multiple children can interact with the garden and each other from remote sites. Each remote user's presence in the virtual space is established using an avatar, a graphical representation of the person's body in the virtual world. The avatars have a separate head, body, and hand that correspond to the user's actual tracked head and hand motions, allowing the environment to record and transmit sufficiently detailed gestures between the participants, such as the nodding of their heads, the waving of their hand, and the exchange of objects. Additionally, voice communication is enabled by a real-time audio connection.

We have also developed a CVE-based learning system for collaborative learning. When the students work in multi-user collaborative mode, their interactions with 3-D objects are synchronized among all users in the same group. The student interface in multi-user mode also provides chat interface for students/lecturers in the group to communicate with each other in real-time. The students can learn/work collaboratively. For example, three students, Robert, Jane, and Lin, study collaboratively to learn assembling a milling machine in a shared 3-D virtual laboratory over the Internet, as shown in Fig. 11. The students can discuss and communicate with each other regarding the actions/tasks to be taken by each student to assemble the milling machine. Only after the students successfully assemble all components of the machine can its power be switched on. Then, its cutter can be tested, and so on. If it is needed, the course lecturer can also join the students in the virtual laboratory through the Internet and demonstrate to the students the correct procedures of assembling the milling machine before they practice themselves. This technology is an example we have created to illustrate the use of CVE in collaborative learning and training.

Our system experiment shows that the prototype learning system can successfully support multi-user collaborative learning in the CVE. It has the advantage of providing the students with active engagement in the learning

Figure 10. NICE project (19).



You touch the Workpiece!
You touch the Table!
Lin: Hi Robert, Can you attach the cutter?
Someone touched the Cutter!
Robert: Sure, the cutter attached.
Lin: Thanks, Robert

Lab - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Stop Refresh Home Search Favorites History Mail Print Edit Discuss Messenger Real.com
Address
Links
Lin
Robert
Jane
Quit
About
Applet started
Local intranet

Figure 11. CVE-based learning system for machine assembly and maintenance.

process. In particular, it promotes role-playing among students, which is important in creating a stimulating learning environment. It also allows students and lecturers to interact with each other in CVE where lecturers can demonstrate the learning contents to the students or the students can collaboratively study/work together to carry out one learning task, which would be particularly useful for those subjects that require multi-user collaboration.

Industrial Training

For similar motivation of using CVE in military training, industrial training looks to CVE as a cost-effective solution. Instead of working with physical equipment, they are modeled as virtual objects with relevant interactive behaviors in a virtual environment accessible to many users. The users, represented by avatars, can then manipulate and interact with the objects as in the real world, gaining valuable experience and training before using the real equipment, which is particularly useful for expensive and complex equipment. For example, Oliveira et al. (20) developed a multi-user teletraining application, which allows users, represented by avatars, to learn how to operate on a faulty asynchronous transfer mode (ATM) switch. The avatars repair the switch in steps that precisely reflect those necessary to perform the same actions in the real world. The prototype consists of two general modules: the user interface module and the network communication module. The user interface itself consists of a graphical interface (GUI), a 3-D interface (VR), and media interfaces (speech recognition, voice streaming, head tracking), as shown in Fig. 12. The upper-right area of the interface, which takes the largest part, is the *3-D environment*. On the left and below the 3-D environment are the *controls* used by the trainees to interact with objects and navigate in the environment. At the top left is the *head-tracking facility*. Below the head-tracking window is a *utility panel* that is used for different purposes as discussed later. There is also a *chat space* where users can exchange textual messages. To avoid navigation problems with inexperienced users, it is possible to view the world from a set of pre-defined camera views.

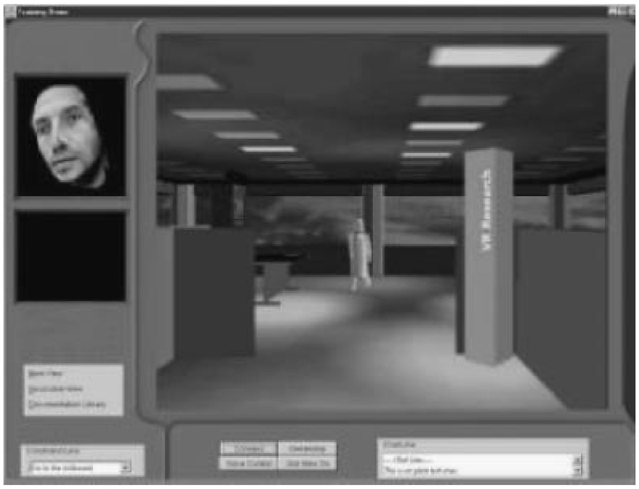


Figure 12. Training application's interface (19).

A user is able to approach and verify the operation of the switch and its cards, remove a faulty card and put it on the repair table, and replace it by installing a new card into the switch. Other parties will be able to watch that user's avatar taking such actions. All of the above actions can be performed by directly navigating in the scene and manipulating objects with the mouse or by selecting the action in a menu. A user can also view video segments showing the correct procedure for performing certain actions. The utility panel is used to display the video clips. If chosen by the trainer, the video will be displayed in every participant's screen. Another use for the utility panel is the *secondary view* feature, with which a user can simultaneously watch the current actions from an alternative point of view. In addition to the interfaces explained above, the prototype offers voice recognition technology whereby the user simply enters commands by talking into the computer's microphone. In this case, the user may simply speak pre-defined commands such as "Go to the table" for the avatar to perform, or may change views by saying "Table Top View," and so on, which enhances the effectiveness of collaborative training.

COLLABORATIVE DESIGN AND GROUP WORK

Randy Pausch at the University of Virginia suggested that the most promising use of CVE would be for applications in which people at different locations need to jointly discuss a 3-D object, such as radiologists using a VR representation of a CAT scan (21), the virtual medicine project developed by SRI International, a teleoperated laproscopic device uses force reflection to provide haptic feedback to a distant surgeon (22). And in the Microelectronics Center for North Carolina (MCNC), a virtual library, Cyberlib, is designed to allow patrons to venture into "the information space independently" or go to a "virtual reference desk" from anywhere across the United States via the Internet (23).

From a product design point of view, the traditional manner of collaboration is geographically limited. Colleagues are not easily able to collaborate and exchange their ideas if they are situated in different locations. Virtual collaboration is intended to solve this problem by incorporating CVE technology to facilitate the collaborative design for small-to medium-sized teams as demonstrated in VRCE (24). The collaborative design functions in VRCE multiview include collaborative design support, multiple opinions via collaborative, multilayer information exchange and experimenting with multiple designs. Daily et al. (25) also reported a system for Distributed Design Review In Virtual Environments (DDRIVE) by HRL Laboratories and General Motors Research & Development Center (GM R&D), as shown in Fig. 13. One important component of the DDRIVE system is the Human Integrating Virtual Environment (HIVE) collaboration infrastructure and toolset developed at HRL Laboratories to support research in multi-user, geographically distributed, 2-D and 3-D shared applications. The HIVE provides virtual meeting tools and high-fidelity multiway audio communication for heterogeneous interface environments. The HIVE is designed to work cooperatively with other applications to add collaboration

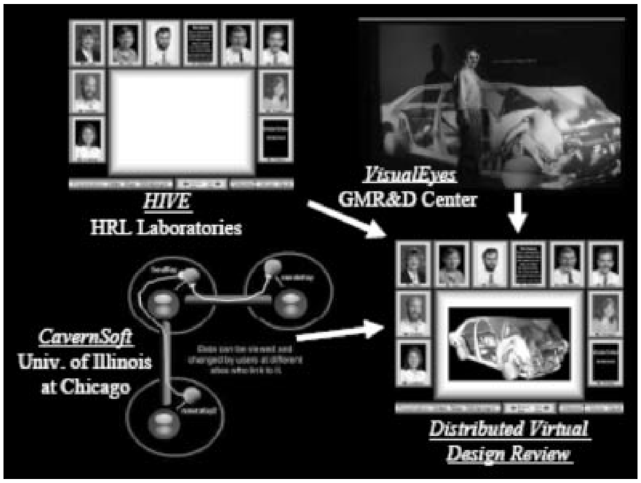


Figure 13. Components of the DDRIVE system (25).

capabilities. Another component is the VisualEyes, which provides a full-size immersive view of a shared model for high-quality visualization. VisualEyes is an interaction tool for visualizing data using mathematics and light. GM uses this tool for testing car designs and collaborating on projects in virtual environments. Imagine being able to sit in the driver's seat of a car before anything is built and change the placing of a dashboard control by a simple gesture because it is too difficult to reach. VisualEyes enables this kind of interactive design via an easy-to-use scripting language that allows control of the environment. Virtual environments can be built much more quickly than with other toolkits by merely bringing in models and applying simple rules.

Frederick et al. (26) also developed a collaborative virtual sculpting system to support a team of geographically separated designers/engineers connected by networks to participate in 3-D virtual engineering sculptures through a collaborative virtual sculpting framework, called VSculpt, as shown in Fig. 14. It provides a real-time intuitive environment for collaborative design. In particular, it addresses issues on efficient rendering and transmission of deformable objects, intuitive object deformation using the *CyberGlove*, and concurrent object deformation by multiple clients.

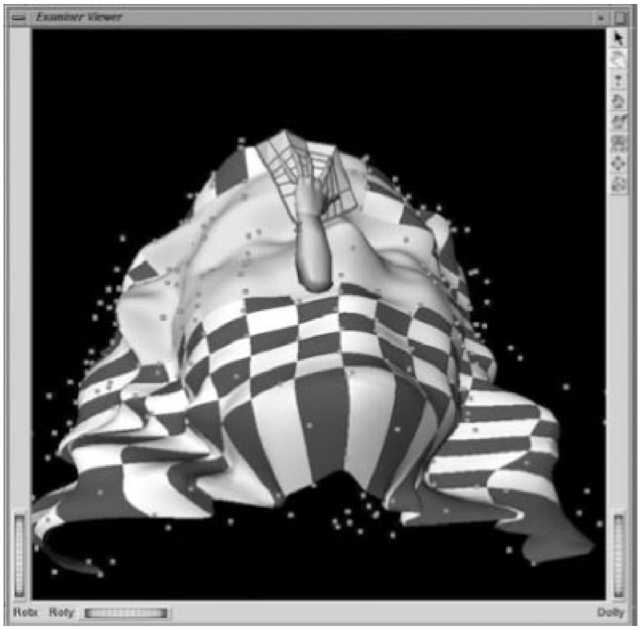


Figure 14. Virtual sculpting of a human head model (26).

SUMMARY

In summary, CVE has vast potential for a wide spectrum of applications. It not only provides an engaging environment for social interaction and entertainment, but also offers cost-effective solutions for military/industrial training and collaborative group work in education, science, and engineering. The performance of CVE applications relies on the natural way of communication and interaction provided by a CVE system. As CVE applications may run in different devices such as desktop PCs, mobile devices like PDAs and wearable computers, and large display systems like CAVEs or HMDs, the advancement in computer processing power, network bandwidth, input devices, tracking and output of text, live voice/video, and so on will play important role in determining the performance of a CVE. The popularity of CVE applications will not only depend on the ease of use of such applications but also will depend on the cost of using them. As computer processors get faster, graphics techniques get more advanced, and the network connection gets faster and cheaper, we believe that CVE applications will be used by ordinary users in everyday life. For ordinary users, CVE applications will become a tool mostly for social interaction/communication, entertainment, distance learning, and working from home. At the same time, military and industrial CVE applications will focus more on visualization large-scale and complex data, decision-making training, task solving, and manipulating objects for training and collaborative work purposes.

In the future, the richness of CVE will be enhanced by new technologies under development. For example, intelligent recognition of faces, expressions, gestures, and speech and automatic detecting, accurate tracking, and localizing of human participants will allow CVE to be intelligent and responsive in nature. Responsive interfaces allow human actions and reactions to be sensed and interpreted by sensors embedded in an environment whereas a collaborative environment allows users to interact with each other through a network connection in a shared 3-D virtual space. Thus, a collaborative and responsive environment will be able to respond to the actions by mixing responses and interaction from other actors in a shared environment. The human participants can manipulate and interact with the environment for control and command as well as be presented with auditory and visual responses and displays in a mixed-reality environment with digital citizens, virtual entertainment, and so on, which will make the communication in CVE more intuitive and natural, thus improving its performance.

To make responsive interface possible, we need the technologies for sensing and interpreting human behavior and natural interaction through gesture, body language, and voice communication. To make collaborative environments with live auditory and visual representation of human participants possible, we need technologies for large-scale collaborative mixed reality, which allows a large number of users to interact with each naturally through the responsive sensing interface. There is no HMD, no body suit, not data glove, but pure human-to-machine and human-to-human interaction naturally in a comfortable and intelligent responsive room where users not only can have fun, like playing multi-user games, visiting places of interest, and meeting friends without any traveling, but also can conduct serious collaborative work like military warfare training, collaborative design and learning, and so on. Although such intuitive and intelligent interfaces will probably not be available and affordable in the near future, they will be necessary to take full advantage of the potential of CVEs.

BIBLIOGRAPHY

1. J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen, The SIMNET virtual world architecture, *Virtual Reality Annual International Symposium*, Seattle, WA, 1993, pp. 450–455.
2. Sony Online Entertainment, EverQuest. Available: http://everquest.station.sony.com/.
3. BigWorld Technology, BigWorld. Available: http://www.bigworldgames.com/.
4. C. R. Karr, D. Reece, and R. Franceschini, Synthetic soldiers-[military training simulators], *IEEE, Spectrum*, **34**: 39–45, 1997.
5. A. Rizzo, J. F. Morie, J. Williams, J. Pair, and J. G. Buckwalter, Human emotional state and its relevance for military VR training, *11th International Conference on Human Computer Interaction*, Los Angeles, CA: New York, Erlbaum, 2005.
6. B. Brown, and M. Bell, Social interaction in 'There', *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, Vienna, Austria: ACM Press, 2004, pp. 1465–1468.
7. Electronic Arts, Inc., Ultima Online. Available: http://www.uo.com/.
8. Turbine, Inc., Asheron's Call. Available: http://ac.turbine.com/.
9. Blizzard Entertainment, World of Warcraft. Available: http://www.worldofwarcraft.com/.
10. Microsoft Research, Virtual World Platform. Available: http://research.microsoft.com/scg/vworlds/vworlds.htm.
11. R. Lea, Y. Honda, K. Matsuda, and S. Matsuda, Community Place: Architecture and performance, *Proc. Second Symposium on Virtual Reality Modeling Language*. Monterey, CA, ACM Press, 1997, pp. 41–50.
12. Sony Corporation, Living Worlds. Available: http://www.sony.net/SonyInfo/News/Press_Archive/199802/980213/index.html.
13. R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W. Yerazunis, Diamond Park and Spline: A social virtual reality system with 3-D animation, spoken interaction, and runtime modifiability, *Presence: Teleoperators and Virtual Environments*, **6**: 461–481, 1997.
14. Blaxxun Technologies, Blaxxun Community Server. Available: http://www.blaxxuntechnologies.com/en/products-blaxxun-communication-server-applications-community.html.
15. P. Schwartz, L. Bricker, B. Campbell, T. Furness, K. Inkpen, L. Matheson, N. Nakamura, L.-S. Shen, S. Tanney, and S. Yen, Virtual playground: Architectures for a shared virtual world, *ACM Symposium on Virtual Reality Software and Technology*, New York, 1998, pp. 43–50.
16. J. C. de Oliveira and N. D. Georganas, VELVET: An adaptive hybrid architecture for very large virtual environments, *Presence: Teleoperators and Virtual Environments*, **12**: 555–580, 2003.
17. IBM Corporation, *IBM and Butterfly to run Playstation Game in Grid*. Available: http://www-1.ibm.com/grid/announce_227.shtml, 2003.
18. IBM Corporation, *Butter Butterfly.net: Powering Next-Generation Gaming with On-Demand Computing*, 2003. Available: http://www.ibm.com/grid/pdf/bufferfly.pdf.
19. A. Johnson, M. Roussos, J. Leigh, C. Vasilakis, C. Barnes, and T. Moher, The NICE project: Learning together in a virtual world, *Virtual Reality Annual International Symposium, Proc. IEEE*, Atlanta, GA, 1998, pp. 176–183.
20. J. C. Oliveira, X. Shen, and N. D. Georganas, Collaborative virtual environment for industrial training and e-commerce, *Globecom'2000 Conference's Workshop on Application of Virtual Reality Technologies for Future Telecommunication Systems*, San Francisco, CA, 2000.
21. R. Pausch, Three views of virtual reality: An overview, *Computer*, **26**: 79–80, 1993.
22. G. Taubes, Surgery in cyberspace, *Discover*, **15**: 85–94, 1994.
23. J. T. Johnson, NREN: Turning the clock ahead on tomorrow's networks, *Data Communications Int.*, **21**: 43–62, 1992.
24. H. Y. Kan, V. G. Duffy, and C.-J. Su, An Internet virtual reality collaborative environment for effective product design, *Computers in Industry*, **45**: 197–213, 2001.
25. M. Daily, M. Howard, J. Jerald, C. Lee, K. Martin, D. McInnes, and P. Tinker, Distributed design review in virtual environments, *Proc. Third International Conference on Collaborative Virtual Environments*, San Francisco, CA, ACM Press, 2000, pp. 57–63.
26. F. W. B. Li, R. W. H. Lau, and F. F. C. Ng, VSculpt : A distributed virtual sculpting environment for collaborative design, *IEEE Trans. Multimedia*, **5**: 570–580, 2003.

Qingping Lin
Liang Zhang
Nanyang Technological University
Singapore

COLLABORATIVE VIRTUAL ENVIRONMENT: SYSTEM ARCHITECTURES

INTRODUCTION

A collaborative virtual environment (CVE) is a shared synthetic space that provides users a shared sense of presence (or the feeling of "being there without physically going there") in a common context with natural interaction and communication. A CVE allows geographically dispersed users to interact with each other and virtual entities in a common synthetic environment via network connections. CVEs are normally associated with three-dimensional (3-D) graphical environments although it may be in the form of two-dimensional (2-D) graphics or may even be text-based. In a CVE, users, which are represented in the form of graphical embodiments called avatars, can navigate through the virtual space, meeting each other, interacting with virtual entities, and communicating with each other using audio, video, text, gesture, and graphics. For example, in a collaborative 3-D digital living community, every digital citizen with a unique digital identification number can create his/her own virtual house. People residing in the digital city can interact and communicate with each other naturally. Furthermore, each digital citizen may create and maintain communities such as a pet community for pet owners, a sports community for sports fans, an online gaming community, a music community, and so on. All real-life activities or pure imaginary activities like space travel or fiction games may be constructed in a collaborative digital living space where users can meet their friends and fans through a network connection from the comfort of their homes/offices. The human participants in a CVE can manipulate and interact with the environment for control and command as well as be presented with auditory and visual responses. Imagine you are flying in space with all the galaxies and clouds as you descend nearer over virtual London. You utter, "I like to see London," and there you are on the busy streets of London. You stroll along the streets and roam about the places with all the familiar sight and sound of London. You make a turn round a corner of a street, and there you are walking along Oxford Street with seas of virtual shoppers (or avatars representing the activities of other users in the collaborative virtual community) dashing around you. Or you can join your friends for a soccer game in the virtual sport community without traveling and physically meeting them. Yes, you can experience all these while you are immersed in your room at home with a network connection to the collaborative virtual community.

In a CVE, the behavior of an avatar acts as a visual description of a human player's movement or action or response to an event in the virtual world. The term "virtual entity" is used in CVE to describe any static or interactive virtual object in the form of text, 2-D/3-D graphics, audio/video streams, an agent, computer-controlled avatar, and those objects that may be invisible (graphically transparent) but will affect user interactions in CVE, for example, wind or transparent barriers. Static objects refer to those virtual entities whose states will not change as the CVE contents evolve. Background objects are often static. Whereas interactive objects refer to the virtual entities whose states will change based on a user's interaction with them; for instance, virtual doors are interactive objects as they can be opened or closed by users. Each interactive object may have several behaviors or animations associated with it. The behaviors may be triggered by the users when they interact with the object. Any changes to the states/behaviors of an interactive object triggered by one user are propagated via network connection to all other users in the same CVE to achieve the sense of sharing the same virtual environment by all users. The virtual entities, description files, sound files, texture images files, user state/interaction information data, and so on are usually stored in a database. The 3-D virtual entities may be stored in different formats, for example wrl, 3ds, obj, and nff. Users may interact with a CVE application through a user interface that could be in various forms ranging from a desktop with keyboard and mouse, to a game console with joystick or other game-playing devices, to mobile and wearable devices, to an immersive virtual reality head-mounted display with data glove and body suite depending on CVE application areas and their requirements.

SYSTEM ARCHITECTURES

One of the challenging tasks for the CVE creator is to determine where the data of the virtual world should be stored and how the data changes (e.g., interaction with an object by a user) should be propagated to all the users who share the same virtual world. When a user navigates in a virtual world, the movement of the avatar representing the user as well as his/her interaction with virtual entities should be perceived by other users in the same CVE. These require data communication among CVE users over the network. The design of system architecture centers around the choice of data communication models, database distribution models, and world consistency maintenance methods, which will have a direct impact on the CVE system performance. It is the core of a CVE design.

Client-Server System Architecture

The client-server architecture (also known as the centralized architecture), as illustrated in Fig. 1, has a server at the center of all communications. In this architecture, the server keeps a persistent copy of the virtual world database. To allow a user to interact with a CVE remotely through network connection, the user's computer ("Client") normally runs a client application program that can communicate with the CVE host computer ("Server") that runs a

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

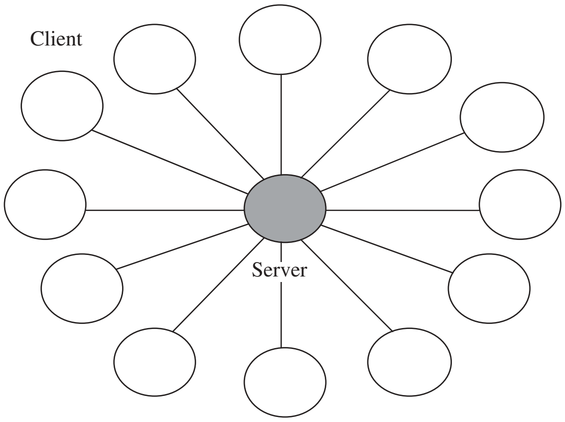


Figure 1. Client–server system architecture.

server program. The initial virtual environment data are first transmitted to the client computer and rendered into 3-D graphics using a client program. The user can then interact with the virtual environment through an input device, e.g., keyboard, joystick, 3-D space ball, or data glove. The user's interaction/navigation commands are captured by the client program. The user's commands may be processed locally by the client software or sent to the server program over the network depending on the computing power requirement. The virtual environment will then change according to the user's commands. The client program will also be responsible for updating CVE states changed by other users sharing the same virtual world. All clients pass their update messages to the server, which in turn distributes these messages to every other client. A database storing the current states of the VE is also kept in the server. When a new client joins a CVE, the server sends its world data to the new client program, which will update all virtual entities to their latest states. For example, RING (1) and Community Place (2) are typical client-server-based CVE systems.

Most small-scale CVEs are realized as client-server architectures because this is a conceptually and practically simple approach and are provided possibilities for accounting and security management. The scene database is centralized in one location: the server. Clients request CVE scene data from the server and update any relevant changes originated from each client end to the server. Because there is only one copy of the CVE scene data held at the server, then the issue of updating the data in a consistent manner is simple. Inconsistencies in the virtual world description can occur if the clients hold local caches of the world. If any changes to the cache should be necessary, the server has to invalidate the cached data and replace it by an up-to-date master copy.

The client–server architecture has the advantage of easy consistency control among all clients in the CVE. In addition, the server can tailor its communication to match the network and machine capabilities of each client. Empirical evidence suggests the scalability of sophisticated client/server systems to several hundred users. Many commercial CVEs use the client–server model, not only for technical but also for administrative and financial reasons: Clients are free, whereas servers and services are sold (4).

However, the problem with this model is the server will soon become the bottleneck when the number of user connections increases. The latency of the system also prolongs because the server basically does data storing and forwarding tasks. The two tasks will make the delay at least twice the time of sending data directly peer to peer.

Peer-to-Peer System Architecture

Peer-to-peer system architecture, as illustrated in Fig. 2, is a common CVE system architecture No server exists in peer-to-peer architecture. Thus, it requires the initialization of every peer host computer (also called the peer node) participating in the CVE with a homogeneous virtual world database containing information about the terrain, model geometry, textures, and behavior of all objects in the virtual environment. Since this architecture requires each peer node to establish a link to every other node in the system, there will be $n(\ n\ -\ 1)/2$ full duplex connections in the system for n participants. For each update, each node would have to send one packet for $n-1$ times, while receiving and processing another $n-1$ packets. This may be alleviated by the use of multicast. However, it is difficult to maintain the consistency of the data and conflicts between user interactions. Each peer host has to take part in this activity. Processes must run on each peer side to communicate with other peer processes and frequently exchange their latest updates to the VE database. Thus, a lot of computation has to be performed on the peer host computer. Furthermore, some peer hosts have to play the referee role to resolve the conflicts and inconsistency of the data replicated among all the participants. The peer nodes, however, may not be powerful enough to cope with the heavy traffic and computation load. This is especially true for Internet-based CVE. The slower peer nodes may soon become bottlenecks because the other peers have to reduce

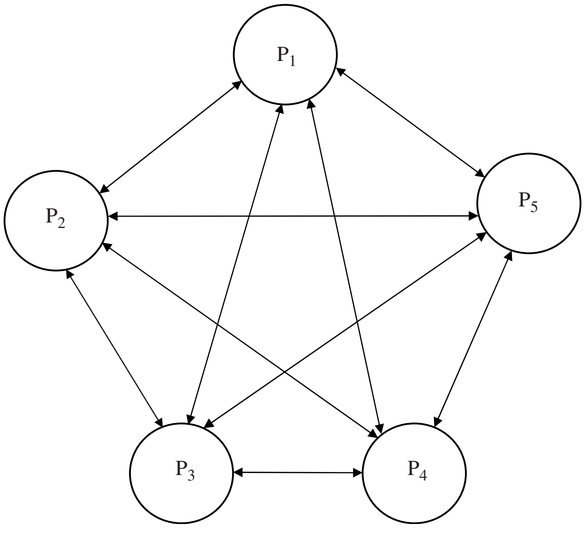


Figure 2. Peer-to-peer unicast architecture (P1 to P5 are indicative peers).

their speed to accommodate their slow peers. If any of the referee peer hosts crashes, it takes a long period before the CVE world state consistency is fully synchronized. Several existing CVE systems use peer-to-peer system architecture, e.g., NPSNET (5), MASSIVE (6), and DIVE (7). The peer-to-peer system architecture can be further classified into peer-to-peer unicast, peer-to-peer multicast, and distributed database with peer-to-peer communication based on data communication and database distribution models.

Peer-to-Peer Unicast. In peer-to-peer unicast architecture, each peer program sends information directly to other peer programs. Every peer has to establish a point-to-point connection of all other peers in a CVE This leads to the burst of the network traffic and to the burden on a single peer's processor (8). Typically, this is the most bandwidth-intensive of the three peer-to-peer approaches, but it avoids placing additional load on particular server computers and introduces lower network delays. This was used for communication in MASSIVE-1 (6). It is also commonly used to provide initial world state information to new participants.

Peer-to-Peer Multicast. Peer-to-peer multicast architecture, as illustrated in Fig. 3, is similar to peer-to-peer unicast except that the same information is not sent simultaneously and directly to many other peer hosts. It normally uses a bandwidth-efficient IP multicast network mechanism. This approach is used exclusively in NPSNET, and it is used for all updates in DIVE and MASSIVE-2. When an application subscribes to a multicast group, it can exchange messages with all applications subscribing to the same multicast group. It is also used for audio in many systems, e.g., SPLINE (9), even when a client/server approach is used for graphical data. However, multicast is not currently available on all networks, and wide-area availability is particularly limited. Consequently, some systems now include application-specific multicast bridging and proxy servers, which simplify use over wide-area and non-multicast networks.

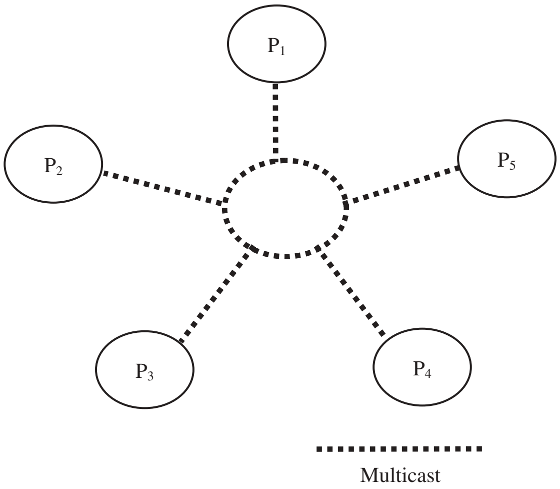


Figure 3. Peer-to-peer multicast architecture (P1 to P5 are indicative peers).

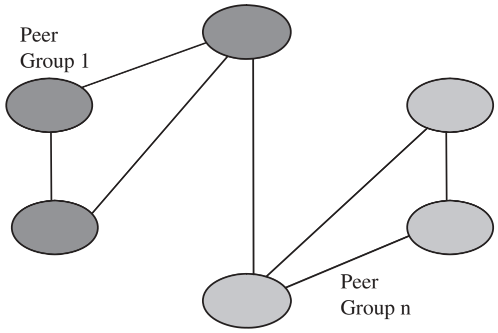


Figure 4. Distributed peer-to-peer architecture.

Distributed Database with Peer-to-Peer Communication. Distributed database with peer-to-peer communication architecture, as illustrated in Fig. 4. It is similar to peer-to-peer unicast/multicast architecture except that not all data of a virtual world are replicated on every CVE participating node. It attempts to segregate a large virtual world into smaller connected scenes so that a group of peer hosts who are close to each other will only update a database that is related to their area of interest. That database is replicated among this group of users.

The disadvantage with this approach is its high communication costs associated with maintaining reliability and consistent data across wide area networks. For example, if a client happens to belong to more than one interest group, it has to maintain multiple databases. Or if he goes across a boundary of groups, the computations and communications involved in reconstructing two areas will impose a significant burden on other peers and the network.

Hybrid Architecture

Hybrid system architecture, as illustrated in Fig. 5, attempts to take advantage of both client–server and peer-to-peer architectures. It merges client–server and peer-to-peer architectures. One approach is to replicate and update the world database on several servers while providing client–server service. The server is replicated or distributed. And server-to-server adopts the peer-to-peer communication model. By replicating the servers, this system can avoid the critical drawback of the client–server model in performance bottleneck and single point of failure due to the single server through which all communication goes. Many CVE systems adopt the hybrid architecture, such as BrickNet (10), RING (1), NetEffect (11), SpaceFusion (12), and CyberWalk (13). However, since every message still has to pass through the servers, certain servers are still possible to become a bottleneck and the breakdown of any server will cause services to the related clients to be shut down. In addition, since the packet pass through the servers, it will cause more delays for the packet.

Another approach is to use a variant form of the client–server model in which the database is partitioned among clients and communication is mediated by a central server.

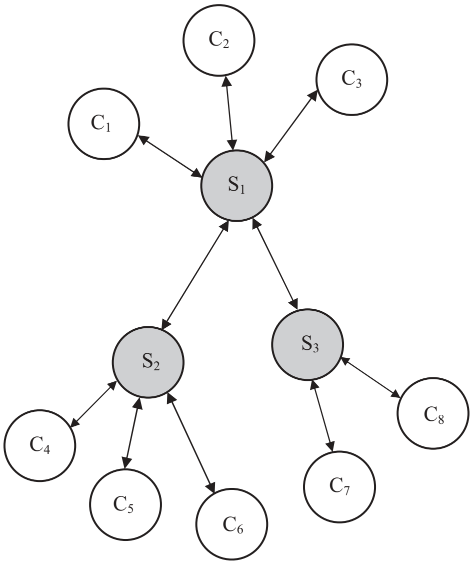


Figure 5. Hybrid system architecture (C1 to C8 are clients and S1 to S3 are servers, the number of clients and servers only serve an indicative purpose).

As an entity moves through the virtual environment, its database is updated by one of the servers, which maintain that part of the virtual world.

In hybrid architecture, the CVE system generally maintains client–server connections to provide the clients with a database of the initial virtual world. The clients will directly communicate with its peers when point-to-point communication appears to be more efficient. The advantage of this model is that it reduces the workload of a server computer. However, the effectiveness of this approach depends on a detailed design of the inter-server and client–server communication model, e.g., the role of servers, the types of data transmitted between client and server, and world consistency maintenance methods.

Mobile Agent-Based System Architecture

The mobile agent-based system architecture (14) models a CVE as a group of collaborative agents. Each agent is a software component to assume an independent task to provide a certain service for the system. Agents collaborate with each other to maintain the entire CVE system. To improve the system scalability, it allows all agents to be mobile without bonding with any fixed host. As the system scales up, agents will be able to clone or migrate to any suitable participating host (including trusted user nodes) to provide more CVE services, e.g., consistency maintenance, persistency maintenance, or scene data delivery. The mutual independence of services and hosts provides large flexibility to utilize the computing and network resources of the system efficiently.

The system architecture is divided into three layers: the resource layer for system resource management, content layer for VE content management, and gateway layer for VE directory management as illustrated in Figure 6. Each layer is composed of multiple collaborative mobile agents to achieve the management tasks.

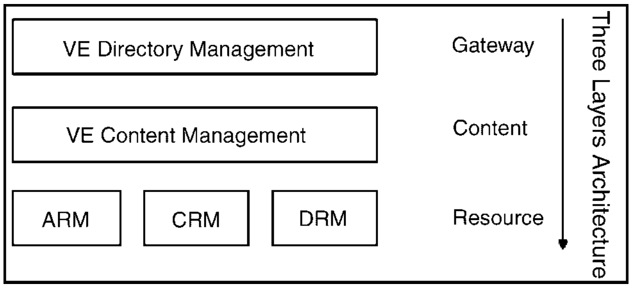


Figure 6. Mobile agent-based CVE architecture.

The resource layer manages the distribution of system resources. In this architecture, mobile agents, system computing nodes, and the data storage space are defined as system agent resource, system computing resource, and system database resource, respectively. Accordingly, the resource layer is further subdivided into three parts: agent resource management (ARM), computing resource management (CRM), and database resource management (DRM). This layer is independent with a different CVE application and scenario. It provides resource management services for the high layers and hides the complexity of the resource distribution. System scalability is further improved by adaptive data communication. The data communication in different parts of a CVE (e.g., region or cell) may adopt client–server, distributed multicast, peer-to-peer, or hybrid architecture depending on run-time activities and consistency requirements in each part of a CVE; for example, when a strict consistency is required for one activity (e.g., group work), then the cell consistent agent will enforce consistent data communication for the activity using client–server architecture (in which the cell consistent agent will act as the server); whereas in another activity that has less consistency requirements (e.g., individual animal hunting power gathering in a game), the cell consistent agent will activate multicast for such activity; or the system may adopt hybrid architecture for different data streams in one activity (e.g., peer-to-peer for audio/video data stream while using client–server for 3-D object interaction).

STANDARDS

Networked/distributed simulations are the ancestors of networked virtual reality/CVE. Thus the standards for networked/distributed simulations can be used as basic architecture for CVE. These standards include distributed interactive simulation and high-level architecture.

Distributed Interactive Simulation

Distributed interactive simulation (DIS) is an IEEE standard protocol for interoperability of distributed simulation. The heart of the DIS paradigm lies in establishing connectivity between independent computational nodes to create a consistent, time and space-coherent synthetic world environment with respect to human perception and behavior (15). This connectivity is achieved through a combination of network communication services, data exchange protocols, and algorithms and databases common to each. Local dead-reckoning is used to improve the DIS function as a standard to extend the SIMNET (16) underlying principle to heterogeneous simulation across local and wide area networks.

An advantage of the DIS-standard is that all DIS-compliant simulations, including CVE, can operate within one virtual environment. However, DIS's underlying data transport mechanism causes problems (17). First, messages may get lost or arrive in the wrong order due to the use of the UDP/IP protocol. Second, the messages sent are part of standardized, fixed-sized protocol data units (PDUs), although generic PDUs exist to communicate any type of data. Finally, because of the broadcast mechanism, the scalability is limited. In the case of CVE, reliable data transfer is crucial. Thus DIS using the UDP/IP protocol is suitable for CVE in a reliable and stable network environment like the local area network (LAN), but it may not be suitable for CVE in heterogeneous network like WANs and Internet.

High-Level Architecture

High-level architecture (HLA) (18) is a general architecture for simulation reuse and interoperability developed by the U.S. Department of Defense. The HLA architecture is an IEEE standard for distributed simulations. It provides a common architecture for reducing the cost and time required to create a synthetic environment for a new purpose. Two basic concepts have been proposed in the HLA: federate and federation. Federate is a software application participating in a federation, which may be a simulation model, data collector, simulator, autonomous agents, or passive viewer. Federation is a named set of federate applications that are used as a whole to achieve some specific objective. All federates incorporate specified capabilities to communicate through the runtime infrastructure (RTI), which is an implementation of a distributed operating system for the federation. The HLA runtime interface specification provides a standard way for federates to communicate with each other by interacting with the RTI. Routing spaces, which are a formal definition of a multidimensional coordinate space, is another important concept offered by HLA to standardize multicast schemes through the RTI. It uses federates' expressions of interest to establish the network connectivity needed to distribute all relevant data and minimal irrelevant data from producers to consumers. The HLA has the desirable features suitable for a basic CVE architecture (17).

CONSISTENCY MAINTENANCE IN CVE WORLD

To support massive interactions among the virtual entities in a CVE and maintain the consistent status of the interactions among the users, appropriate event detection and propagation mechanisms have to be provided to a CVE system. The task of detecting and propagating interactions is of order n-squared, where n is the number of entities in the VE. When the system is scaled up, this task may become too heavy to be handled. To improve the efficiency of world state consistency maintenance in CVE, various approaches have been developed by researchers in the field, including the broadcast method, distance-based method, acuity-based method, region/cell-based method, receiver interest method, peer/group-based method, sight view and spatial aura method, as well as behavior-based method.

Broadcast Method

The conventional approach maintaining the status consistency of a shared virtual world for distributed users is to update the status changes through central server broadcast, e.g., earlier version of aggregate level simulation protocol (ALSP)-based systems (19). In a client/server-based CVE system, when a user joins a virtual world maintained by a group of servers, the user's interaction with the virtual world will be captured and sent to the central servers. The servers then broadcast the interaction messages to all other users who share the same virtual world. The advantage of the broadcast method is that it has no computational cost for resolving the destinations for propagating the interaction messages. However, as the concurrent user number and VE size grows, the servers soon become a bottleneck. At the same time, the users are overwhelmed with the interaction messages that are not of interest, or even not relevant, to them. It results in unnecessary consumption of system network resource and processing power, and thus poor system scalability and performance.

Distance-Based Method

With the distance-based method, the distance between a user and the entities around him is used to decide the user's ability to know (see, hear, etc.) the entities. Only the states from the entities that are close enough to the user are actually sent to the particular user's host.

There are two ways of applying this method. The first one uses spatial distance to enable the interaction. Different mediums can have their corresponding spatial distance. For example, someone's voice could be heard before he is observed by others. Another form of the distance-based method is to enable interaction through the *horizon count* method. The horizon count indicates the maximum number of entities from which the user is prepared to receive updates. The system typically sorts the entities by distance from the virtual embodiment—the avatar—of the user, and only the closest ones can send their updates to the user. This is the approach of the interaction management used in Blaxxun's CyberHub (21). The distance-based method also uses spatial distance to calculate the level of detail (LOD) of the interaction information (22, 23). When the interaction is enabled, the distance between the two interaction entities will be used as the parameter to calculate the LOD of the interaction information sent between them.

The advantage of the distance-based method is that it considers the spatial distance as the dependent condition for enabling the interaction. That is very natural for some interactions occurring in a CVE. Another strength of this method is its simple logic for implementation. But it ignores the dependent conditions of the interaction entities to attract others' attentions other than the spatial distance. And it also has a high CPU requirement for the calculation of the distance among the interactive entities.

Acuity-Based Method

The distance-based approach works, but it can fail in some cases. A large object that is far away may be more relevant than a small object nearby. For example, a user in the VE may be able to see a tall building far away but not an insect nearby even though the insect is closer. To better reflect this scenario, the acuity-based method (24) is introduced. Acuity is a measure of the ratio between size and distance. In this model, every user has an acuity setting, which indicates the minimum size-over-distance ratio required in order for an entity to be known (seen, heard, etc.) to him. If the entity's acuity value is less than the acuity setting for the user, then the entity is too small or too far away for the user to see. In such a case, no updates from the entity are sent to the user. The acuity setting of the user is different for each type of medium. In this method, the ability of the entity to attract others is also considered.

The acuity-based method considers the size and the spatial distance of the interaction entities as the dependent condition to attract others' attentions. It also has a simple logic to implement. But it requires high CPU time for the calculation of the acuity among the interactive entities. The user's interest/intention is not taken into consideration, either.

Region/Cell-Based Method

In the region/cell-based method, the virtual world is partitioned into smaller pieces known as "regions" or "cells." The partition may be static (1) or adaptive (25). The system will decide which pieces are applicable to each particular user. Only the updating information from these pieces is transmitted to the corresponding user. The partition of the regions/cells is transparent to the users. Users can move among the regions/cells. When this kind of migration occurs, the LOD of the information between the user and other entities is dynamically changed accordingly. Bounding boxes and binary space partitioning (BSP) trees are used to define the region/cell partition in the VE.

The region/cell partition is widely used in CVE systems, such as the *locale* in Spline (3), and the *hexagonal cell* in NPSNET (5). It also appears as a third-party object in MASSIVE II (26). The advantage of this method is that it has a light computational load on CPU as multicast groups can be organized according to region or cell. It also reduces the interaction message propagation scope to a particular region or cell. However, this approach only provides a rough interaction message filtering for consistency maintenance of a CVE; more information may be received and processed than is needed. Another difficulty is that a VE can suffer from what is known as crowding or clumping. If many entities crowd, or clump, into one region, some entities that subscribe to the region may be overwhelmed by other entities not of interest to them.

Receiver Interest-Based Method

In the receiver interest-based method, the interaction messages are propagated to the interested users only. The interest management is accomplished by expressing the user's interest in a particular object to a server. The server in turn sends the updated state information of the objects whenever it is needed. Or the client application is required to subscribe to certain interest groups, i.e., to express the user's interests, before it can receive the status changes among the entities in those groups. Systems based on such a type of consistency maintenance method include the Minimal Reality (MR) Toolkit (27), Joint Precision Strike Demonstration (JPSD) (28), BrickNet (10), Close Combat Tactical Trainer (CCTT) (29), and Proximity Detection (30).

The advantage of the receiver interest-based method is that remote clients receive only messages of interest, and they never need to throw away information. But the sending entity, or the Interest Manager, needs to have knowledge of, as well as evaluate, the interests of all other entities in the virtual world. Thus, it has high CPU computation cost and increased latency and bandwidth requirement for a CVE system that supports a large number of concurrent users.

Peer/Group-Based Method

If a user does not wish to receive information from some entities, he could choose to "ignore" those entities. A variant on this "ignore" selection is the use of peers. By designating an entity as a peer, certain data streams are sent only to that peer. This provides a form of private communication, like whispering.

Grouping is another variation of this idea. By designating certain entities as part of a group, private conversation could be set up among the group members. The users out of the group will not receive the updating information of the entities in the group. Different from the region partition model, which divides the VE spatially, this approach partitions entities that could be involved in the interactions. The CyberSockets used in CyberHub (21), for example, supports the concepts of peers and groups.

Another example that uses the group-based method is the NPSNET (31). It partitions VE into associating spatial (hexagonal cell region division), temporal, and functionally related entity class to form network multicast groups. The Area of Interest Manager (AOIM) is used to identify the multicast groups that are potentially of interest to the users, and to restrict network traffic to these groups.

The main advantage of the peer/group-based consistency maintenance method is that it can reduce the network bandwidth consumption through the use of grouping methods. But virtual entity grouping computation in a large CVE requires high CPU processing power.

Spatial Aura Method

The distance-based method and the region-based method only propagate interaction information among the interaction entities according to the spatial distance among the entities, or the spatial position of the entities. However, more precise information filtering is needed when the system requires a more realistic spatial-based effect on the

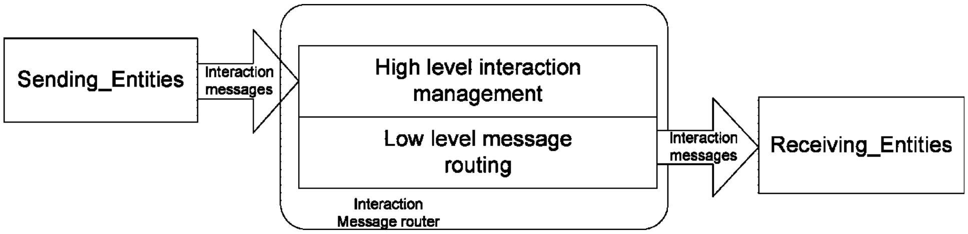

Sending_Entities
Interaction messages
High level interaction management
Low level message routing
Interaction Message router
Interaction messages
Receiving_Entities

Figure 7. Message routing in behavior based method for CVE consistency maintenance.

consistency maintenance. The spatial aura method is developed for this purpose in MASSIVE (6). The spatial interaction model is used to maintain users' awareness in the VE. Other systems, including DIVE (7) and the Community Place (2), also use the spatial aura interaction model.

The key concepts of the spatial model used in MASSIVE include *medium, aura, awareness, focus, nimbus*, and *adapters*. When a user's aura overlaps a virtual entity's aura, then interaction becomes possible. After the interaction is enabled by the aura collision, awareness is negotiated through combining the observer's focus and the observed entity's nimbus. A third-party object (26) is extended into the spatial model to allow for richer context-sensitive patterns of interaction.

The advantage of the spatial aura method is that it has a natural awareness model. It can support peripheral awareness. But it uses a passive view to the interaction. The intentions of the users are ignored. It has a high CPU requirement for the calculation of the Aura collision. To implement it, a server must perform an $O(N^2)$ collision detection between each entity within the virtual world.

Behavior-Based Method

The behavior-based method (32) incorporates a two-tiered architecture, which includes the high-level role of behavior-based interaction management and low-level message routing. In the high level, the interaction management is achieved by enabling the natural interactions based on the collaborative behavior definitions provided by CVE application developers. Thus, it extends the developer's management ability of the collaborations in the CVE application. In the low level, the message routing controls the propagation scope of the interaction messages according to the runtime status of the interactive entities, and hence reduces the network bandwidth consumption. The behavior-based interaction management supports routing of the interaction message via controlling the destination of the message and the LOD of the message. As illustrated in Fig. 7, the high-level interaction management serves as the first layer of interaction message filtering through identifying the virtual entities (human users, robot type intelligent agents, interactive objects, etc) involved in an interaction event, which is represented by a message received by the message router, based on role behavior definitions. The low-level message routing serves as the second layer of interaction message filtering based on the output from the first layer. This significantly reduces the number of virtual entities need to be evaluated for computing the low-level message routing. Accordingly, it greatly reduces the server computation time required to resolve the message routing destinations and LOD in the low-level message routing models, which are commonly used by existing interaction management approaches.

DESIGN ISSUES IN LARGE-SCALE CVE

CVE has drawn significant research interests in recent years. As CVE scales up in terms of the number of concurrent users and the complexity of a CVE virtual world, the greatest challenge for a large-scale CVE (LCVE) system design is not how to simulate the objects on individual client machine, but how to transfer the state changes of virtual entities (e.g., users or virtual objects) over the heterogeneous network like the Internet effectively, despite potentially vast variation in client machine computing powers. In constructing an LCVE, many issues need to be considered such as extensibility, scalability, consistency, and persistency issues. Extensibility refers to the ability to extend and evolve, in the sense that it can accommodate a dynamic number of users, dynamic number of 3-D virtual entities, as well as dynamic scope of the virtual world itself. Scalability is the capability to maintain the LCVE performance in the extended environment with a large number of virtual entities and concurrent users. To achieve scalability, the system has to make sure that the resources are not limited to certain scope and that the states of the virtual world are maintained persistent at all times. All LCVE participants also need to perceive the same state of virtual world at any point of time. In addition, fault tolerance, easy deployment, and configuration are also necessary.

Each of the issues mentioned above has its own challenge. The scalability of a system is limited by the available network bandwidth and processing power of participating computer hosts. To achieve scalability and maintain performance quality, the network load has to be divided carefully between machines. However, this will cause a challenge to the consistency and persistency issue because the need to synchronize every activity by the machines becomes higher.

Two general models are based on which LCVE can be built, a centralized system and distributed system. To build a scalable and extensible LCVE while maintaining the performance quality, a centralized system should not be used. A centralized system has a single controller that controls every data transmission and maintains the system database. Although this system will allow simple world consistency maintenance because of the single database that is only accessible by the central controller, the controller itself will become a bottleneck to the system. Thus, another option is the distributed or hybrid system. A distributed system does not have a central controller, and the network entities synchronize themselves by communicating with each other. Although good world consistency maintenance is crucial to a distributed system, this will provide a better load balancing to the system compared with the centralized system. However, because of the limitations of the distributed or hybrid approach discussed in previous section, to further enable the construction of a scalable and extensible LCVE, a mobile agents-based approach may be a good alternative.

A mobile agent is an autonomous network entity that can migrate between heterogeneous machines. Mobile agents can work independently to perform some tasks assigned to them. They can also work together with another entity, which requires them to have the capability to communicate using a definite communication protocol. Mobile agents can make runtime intelligent decision such that when overloaded, they can clone new agents to share their workload. They can also migrate to other machines or to terminate autonomously when their services are not needed. In short, mobile agent can have dynamic existence and provide a good load balancing, fault tolerance, and service deployment. Therefore, it can be observed how mobile agent mechanism can further contribute to the scalability and extensibility of LCVE.

BIBLIOGRAPHY

1. T. A. Funkhouser, RING: A client-server system for multi-user virtual environments, *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, Monterey, California, 1995.
2. R. Lea, Y. Honda, K. Matsuda, and S. Matsuda, Community place: Architecture and performance, *Proc. of the Second Symposium on Virtual Reality Modeling Language*, Monterey, California, ACM Press, 1997, pp. 41–50.
3. S. Benford, C. Greenhalgh, T. Rodden, and J. Pycock, Collaborative virtual environments, *Comm. ACM*, **44**: 79–85, 2001.
4. J. W. Barrus, R. C. Waters, and D. B. Anderson, Locales: Supporting large multiuser virtual environments, *Computer Graphics and Applications, IEEE*, **16**: 50–57, 1996.
5. M. R. Macedomia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, Exploiting reality with multicast groups: a network architecture for large-scale virtual environments, *Proc. of Virtual Reality Annual International Symposium*, IEEE Computer Society, Washington, DC, 1995.
6. C. Greenhalgh and S. Benford, MASSIVE: a distributed virtual reality system incorporating spatial trading, *Proceedings of 15th International Conference on Distributed Computing Systems*, Vancouver, BC, Canada, 1995.
7. E. Frecon and M. Stenius, DIVE: A scaleable network architecture for distributed virtual elements, *Distrib. Sys. Engineer.*, **5**: 91–100, 1998.
8. M. R. Macedonia and M. J. Zyda, A taxonomy for networked virtual environments, *IEEE MultiMedia*, **4**: 48–56, 1997
9. R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W. Yerazunis, Diamond park and spline: A social virtual reality system with 3D animation, spoken interaction, and runtime modifiability, *Presence: Teleoperat. Virt. Environ.*, **6**: 461–481, 1997.
10. G. Singh, L. Serra, W. Pang, and H. Ng, BrickNet: A software toolkit for networks-based virtual worlds, *Presence: Teleoperat. Virt. Environ.*, **3**: 19–34, 1994.
11. T. K. Das, G. Singh, A. Mitchell, P. S. Kumar, and K. McGee, NetEffect: A network architecture for large-scale multi-user virtual worlds, *Proc. of the ACM Symposium on Virtual Reality Software and Technology*, Lausanne, Switzerland, 1997.
12. H. Sugano, K. Otami, H. Ueda, S. Hiraiwa, S. Endo, and Y. Kohda, SpaceFusion: A multi-server architecture for shared virtual environments, *Proc. of 2nd Annual Symposium on the Virtual Reality Modelling Language*, Monterey, CA, 1997.
13. J. Chim, R. W. H. Lau, V. Leong, and A. Si, CyberWalk: A web-based distributed virtual walkthrough environment, *IEEE Trans. Multimedia*, **5**: 503–515, 2003.
14. L. Zhang and Q. Lin, MACVE: a mobile agent based framework for large-scale collabortive virtual environments, *Presence: Teleoperat. Virt. Environ.*, **16**(3): 279–292, 2007.
15. R. C. Hofer and M. L. Loper, DIS today [Distributed interactive simulation], *Proc. of the IEEE*, **83**: 1124–1137, 1995.
16. D. C. Miller and J. A. Thorpe, SIMNET: The advent of simulator networking, *Proc. of the IEEE*, **83**: 1114–1123, 1995.
17. EPFL, Geneva, IIS, Nottingham, Thomson, TNO, Review of DIS and HLA techniques for COVEN, ACTS Project N. AC040, 1997.
18. J. S. Dahmann, High Level Architecture for Simulation, *Proc. of the 1st International Workshop on Distributed Interactive Simulation and Real-Time Applications*, 1997, pp. 9–15 .
19. K. L. Morse, L. Bic, and M. Dillencourt, Interest management in large-scale virtual environments, *Presence: Teleoper. Virt. Environ.*, **9**: 52–68, 2000.
20. C. Greenhalgh, An experimental implementation of the spatial model, *Proc. of 6th ERCIM Workshops*, Stockhom, 1994.
21. B. Roehl, J. Couch, C. Reed-Ballreich, T. Rohaly, and G. Brown, *Late Night VRML 2.0 with Java*, 1997.
22. R. Kazman, Load balancing, latency management and separation concerns in a distributed virtual world, *Parallel Comput. - Parad. Applicat.*, 1995, pp. 480–497.
23. S. Pettifer, J. Cook, J. Marsh, and A. West, DEVA3: Architecture for a large scale virtual reality system, *Proc. of ACM Symposium in Virtual Reality Software and Technology*, Seoul, Korea, ACM Press, 2000, pp. 33–39.
24. M. Reddy, B. Watson, N. Walker, and L. F. Hodges, Managing level of detail in virtual environments - A perceptual framework, *Presence: Teleoperat. Virt. Environ.*, **6**: 658–666, 1997.
25. R. W. H. Lau, B. Ng, A. Si, and F. Li, Adaptive partitioning for multi-server distributed virtual environments, *Proc. of the Tenth ACM International Conference on Multimedia* Juan-les-Pins, France, ACM Press, 2002, pp. 271–274.
26. C. Greenhalgh, Large scale collaborative virtual environments, Ph.D Dissertation, Nottingham, UK Department of Computer Science, University of Nottingham, 1997.
27. C. Shaw and M. Green, MR toolkit peers package and experiment, *IEEE Symposium on Research Frontiers in Virtual Reality*, San Jose, CA, 1993, pp. 463-469.
28. E. T. Powell, L. Mellon, J. F. Watson and G. H. Tarbox, Joint precision strike demonstration (JPSD) simulation architecture, *14th Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando, FL, 1996, pp. 807–810.
29. T. W. Mastaglio and R. Callahan, A large-scale complex virtual environment for team training, *IEEE Computer*, **28**(7): 49–56, 1995.

30. J. S. Steinman and F. Wieland, Parallel proximity detection and The Distribution List algorithm, *Proc. of 8th Workshop on Parallel and Distributed Simulation*, Edinburgh, UK, 1994.

31. M. Macedonia, D. Pratt, and M. Zyda, NPSNET: A network software architecture for large-scale virtual environments, *Presence: Teleoperat. and Virt. Environ.*, **3**: 265–287, 1994.

32. Q. Lin, W. Wang, L. Zhang, J. M. Ng, and C. P. Low, Behavior-based multiplayer collaborative interaction management, *J. Comp. Animat. Virt. Worlds*, **16**: 1–19, 2005.

QINGPING LIN
LIANG ZHANG
Nanyang Technological
University
Singapore

COLLABORATIVE VIRTUAL ENVIRONMENT: WEB-BASED ISSUES

INTRODUCTION

With the Internet's exponential growth in the past decade, it has evolved from a repository of information to an interactive digital social world. Interactive multimedia contents can be embedded into Web pages to enrich the information presented in the cyberspace for enhanced social and business applications. Collaborative virtual environments (CVEs) function as a new type of interface that attempts to model the Internet into a real social world using the power of Internet communication and virtual reality. VRML/X3D has been developed as an International Standard Organization (ISO) standard to deliver three-dimensional (3D) interactive multimedia contents over the Internet as a 3-D extension to the World Wide Web (WWW). Java3D has also been developed to construct 3-D graphical applications for visualization and interaction over the Web. With VRML/X3D/Java3D, a Web-based CVE can be created using standard http protocol. However, due to heterogeneous nature of the Internet, special cares must be taken to address the Internet-related issues for CVE. In this article, we discuss the existing standards, methods for constructing Web-based CVE, and popular solutions to the Web-based CVE issues.

STANDARDS

Traditional CVEs are often application-based; i.e., the client or peer program communicates with each other or with the server directly for content delivery and virtual world states maintenance. Whereas a Web-based CVE requires the 3-D virtual world contents to be rendered and embedded in a Web page using a Web browser. The existing standards that can be used for such purposes include VRML, X3D, and Java3D.

VRML

Virtual Reality Modeling Language (VRML) (1) is an industrial standard file format for representing 3-D interactive vector graphics. The 3-D scene described by a VRML file is known as virtual world and can be distributed over the WWW and presented in special VRML browsers, most of which are plug-ins for the Web browsers. A reference to a VRML file can be embedded in a Hyper Text Markup Language (HTML) page, and hyperlinks to other media such as text, sounds, movies, and images can also embedded in VRML files. Thus, VRML can be seen as a 3-D visual extension of the WWW (2).

VRML originated in 1994 and the first official VRML 1.0 specification was released in 1995. The VRML 1.0 was a good start as an Internet-based 3-D graphics format, but it was a static scene description language, which cannot support user interactions. In 1996, VRML 2.0 was released that supports dynamic, interactive 3-D scenes. In 1997, VRML 2.0 was accepted by the ISO as ISO/IEC 14772 standard, also known as VRML97. It has the following main features.

Hierarchical Scene Graph. In VRML97, the basic element is the node that describes 3-D shape and geometry, appearance properties to be applied to the shape's geometry, light sources, viewpoint, and so on. Each node is typed and has a set of fields that parameterize the node. The 3-D scene consists of a set of nodes arranged in a hierarchical fashion. The hierarchy is built up by using a parent–child relationship in which a parent may have any number of children, some of whom may, in turn, be parents themselves. Scene graph refers to the entire ordered collection of these scene hierarchy.

Event Routing Mechanism. VRML97 provide an event routing mechanism to support a dynamic and interactive 3-D scene. Some nodes can generate events in response to environmental changes or user interaction, and other nodes can receive events to change the state of the node, generate additional events, or change the structure of the scene graph. These nodes may be connected together by ROUTE to achieve real-time animations, including entity behaviors, user–entity interaction, and inter-entity coordination.

Prototyping Mechanism. Prototyping provides a way to extend the build-in node types. It defined a new type of nodes, known as a prototype node, to parameterize the scene graph. Once defined, prototyped node types may be instantiated in the scene graph exactly like the built-in node types.

External Authoring Interface. External Authoring Interface (EAI) is an interface specification that allows an external program to manipulate the VRML scene graph while not directly being part of the scene graph. The implementation of EAI in Java is a set of classes with methods that can be called to control the VRML world to support dynamic scene changes.

However, VRML does not address any of the issues having to do with networking these worlds to enable multiple participants to interact with each other or distribute the workload associated with the simulation. Thus, developing Web-based CVEs using VRML requires network data communication support in order to allow multiple users to have collaborative interaction in a shared virtual world.

X3D

Extensible 3-D (X3D) (3) is an open standards Extensible Markup Language (XML)-enabled 3-D file format to enable real-time communication of 3-D data across all applications and network applications. It has a rich set of features for use in engineering and scientific visualization, CAD and

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

architecture, medical visualization, training and simulation, multimedia, entertainment, education, and more.

The X3D specification expressing the geometry and behavior capabilities of VRML using the Web-compatible tag sets of the XML. Scene graph, nodes, and fields, respectively, correspond to document, elements, and attributes in XML parlance. X3D is a more mature and refined standard than its VRML97 predecessor, so authors can achieve the behaviors they expect. X3D also provides compatibility with VRML97. In X3D, there is still a "Classic VRML" encoding that can play most nonscripted VRML97 worlds with only minor changes. None of the technology has been lost. It instead has evolved into X3D. Many features requested for VRML have been provided in X3D in a manner that is completely integrated into the standard. Thus, you can think of X3D as "VRML3"(4).

However, in contrast to the monolithic nature of VRML97, which requires the adoption of the entire feature set for compliance, X3D allows developers to support subsets of the specification ("Profiles"), composed of modular blocks of functionality ("Components") (3). A component-based architecture supports creation of different "profiles" that can be individually supported. Components can be individually extended or modified through adding new "levels," or new components can be added to introduce new features, such as streaming. Through this mechanism, advancements of the specification can move quickly because development in one area doesn't slow the specification as a whole. Importantly, the conformance requirements for a particular piece of content are unambiguously defined by indicating the profiles, components, and levels required by that content.

Java3D

Java3D (5) is an API released by Sun Microsystem, now a community source project developed on java.net. It provides methods for 3-D graphics application development. It is a standard extension to the Java 2 SDK. This would mean that Java3D removes the unreliability of communicating through an external program. Similar to all Java programs, Java3D is portable, robust, and platform independent.

Contrary to popular belief that it will replace VRML, Java3D and VRML are actually complimentary technologies. In terms of 3-D user interface support, VRML is Java3D's closest predecessor. VRML provides a standard 3-D interchange format that allows applications to save, transmit, and restore 3-D images and behaviors that are independent of application, hardware, platform, or programming language. Java3D provides a Java class library that lets an application developer to create, manipulate, and render 3-D geometry and portable 3-D applications and applets. In fact, Java3D provides support for runtime loaders that can load 3-D files of different formats such as VRML (.wrl), 3D Studio (.3ds), Lightwave (.lwo), and Wavefront (.obj).

Java3D provides a high-level, object-oriented view of 3-D graphics using a *scene graph*-based 3-D graphics model. It is designed to help programmers without much graphics or multimedia programming experience use 3-D in their applications. However, Jav3D is a high-level API and hides the rendering pipeline details from developers. This makes it unsuitable for several problems where such details are important. In addition, most Java3D components are heavyweight. They use native non-Java code to implement the actual rendering. This can complicate the Graphical User Interface development if a program uses Java Swing and its all-Java, or lightweight, components. In general, lightweight and heavyweight components do not mix well in the same container objects and windows.

CONSTRUCTING WEB-BASED CVEs

To embed CVEs in a Web page, which can be delivered via http protocol, and support collaborative interaction in a Web-based CVE, a 3-D modeling language is required to provide an interface that can capture local user's interactions with virtual entities and propagate the interactions to others through the network. At the same time, the interface should also be able to update the states of shared objects, which are changed by other users in the CVE, in the local virtual environment. To achieve this, EVENTS, ROUTES, and Sensors nodes in VRML (or X3D) can be used in combination to access a 3-D scene in the runtime, collect data from the scene, and change VRML nodes, properties of the scene.

Events. When one node needs to pass some information to another node, it creates an event. An event contains two pieces of information: the data and the timestamp. A timestamp is when the event was originally generated, not when it arrives at the destination. In VRML syntax, the behavior of events are defined by fields; three kinds of fields can be accessed: (1) eventIn, which is write-only and used to set value of a node; (2) eventOut, which is read-only and used to output data when there's an event; and (3) exposedField, which is both readable and writable.

Route. Route is used to pass information between nodes in the scene. It can be used to create an explicit connection between events. It can link an eventOut of a node to an eventIn of another node; for example, a statement "ROUTE node1.eventOutField TO node2.eventInField" results in the node1's evenOutField value to be sent to node2's eventInField.

Sensors. In normal cases, an eventOut field cannot generate events. VRML use sensors to get users' input and detect users' behavior/interaction or get time information. When the user interacts with a virtual entity in a CVE, a corresponding event will be generated. Table 1 defines the sensors in VRML and their functions.

CVE Scene Access

Besides generating interactive behaviors and capturing user interaction with VRML world, communication between the virtual environment and the network is further required to allow multiuser collaborative interaction. VRML's *External Authoring Interface* (EAI) (6) has been designed for such a purpose to allow communication between the VRML-based 3-D virtual world and its external environment. (Note: X3D uses Scene Access Interface (SAI) to achieve similar functionalities as VRML EAI.). It defines a set of functions on

Table 1. Sensors defined in VRML97

Name	Description
CylinderSensor	Translates user input into a cylindrical rotation. The CylinderSensor node maps pointer motion (e.g., a mouse or wand) into a rotation on an invisible cylinder that is aligned with the Y-axis of the local coordinate system.
PlaneSensor	Translate user input into a motion along the X–Y plane of the local coordinate system.
SphereSensor	Translates user input into a spherical rotation. It maps pointing device motion into spherical rotation about the origin of the local coordinate system.
ProximitySensor	Generates events when the viewer enters, exits, and moves within a region in space (defined by a box).
TimeSensor	Generates events as time passes. It can be used for driving continuous simulations and animations, controlling periodic activities (e.g., one per minute), and initiating single occurrence events such as an alarm clock.
TouchSensor	Detects when the user touches a given object. A TouchSensor node tracks the location and state of the pointing device and detects when the user points at geometry contained by the TouchSensor node's parent group.
VisibilitySensor	Detects if an object is currently visible to the user. The VisibilitySensor node detects visibility changes of a rectangular box as the user navigates the world.

the VRML browser that the external environment can perform to affect the VRML world. The external environment may take the form of a container application that holds a VRML browser, or a client/server style environment where the VRML browser forms the client and the application is a server program located on a remote computer. VRML EAI allows the developers to easily extend the functionality of the VRML 2.0 browser and thereby build the dynamic content of a 3-D application. In essence, EAI provides a set of methods for developing customized applications to interact with and dynamically update a 3-D scene so that the applications can "talk" to the VRML scene in real time. The EAI specifies the binding of its functions to Java, and hence, it can be considered a Java application programming interface. It defines a set of Java package and class for bridging Java applications to the VRML world. As a result, Java's strong ability in networking and multithreading can be used to establish the network connection and then achieve real-time collaboration among the users.

As illustrated in Fig. 1, by using VRML's EAI, together with Java Applet and Java networking capability, a VRML-based multi-user collaborative virtual environment can be built over the Internet. The VRML browser window generates and renders the 3-D geometry, whereas Java Applet delivers the control of the behavior and the logic of VRML scene graphical objects. EAI provides a two-way interface that lets VRML models and Java applets interact. It offers a set of functions of the VRML browser that the external program can call to control the content in the VRML scene. VRML EAI allows an external environment to access nodes in a VRML scene by using the existing VRML event model. In this model, an eventOut of a given node can be routed to an eventIn of another node. When the eventOut of a node generates an event, the eventIn (i.e., the receiver of the event generated by the eventOut node) is notified and its node processes that event.

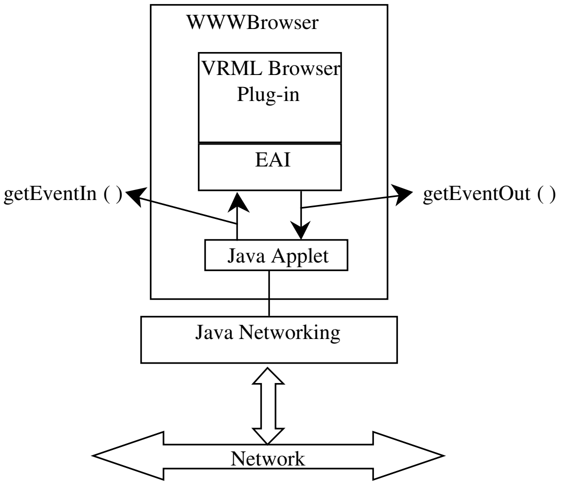


Figure 1. Communication between VRML world and the network through EAI.

Additionally, if a *script* in a Script node has a reference to a given node, it can send events directly to any eventIn of that node and it can read the last value sent from any of its eventOuts.

The Java implementation of the External Authoring Interface is specified in three Java *packages*: *vrml.external*, *vrml.external.field*, and *vrml.external.exception*. All members of package *vrml.external.exception* are classes derived from *java.lang.RuntimeException*; the rest of the members of the packages are specified as interfaces (with the exception of *vrml.external.field.FieldTypes*, which merely defines an integer constant for each EventIn/EventOut type). This allows the compiled Java applet to be used with any VRML browser's EAI implementation.

The EAI provides four types of accesses to a VRML scene:

1) Accessing the Browser
2) EventIn processing
3) EventOut processing
4) Getting notification from VRML scene

Accessing the Browser. The application communicates with a VRML world by first obtaining a reference to a browser object. This allows the application to uniquely identify a particular VRML scene in a computer where multiple scenes are active. To get a browser in Java applet, the following lines should be included in the applet program:

```
import vrml.external.Browser
Browser browser = Browser.getBrowser(this);
```

Once the Browser object is created, the Java applet can access scene nodes in the VRML Browser.

To gain access to a node (e.g., “ROOT” node), the Java applet needs to include the following codes:

import vrml.external.Node
Node rootnode = browser.getNode(“ROOT”);

Then, any or all events or leaf nodes of the Node can be observed and/or modified.

EventIn Processing. Once a node reference is obtained, all its eventIns are accessible using the *getEventIn()* method. This method is passed in with the name of the eventIn and returns a reference to an *EventIn* instance if an matching eventIn name is found. ExposedFields can also be accessed, either by giving a string for the exposedField itself (such as “*translation*”) or by giving the name of the corresponding eventIn (such as “*set_translation*”).

After an instance of the desired *EventIn* is obtained, an event can be sent to it. But *EventIn* has no methods for sending events. It must first be cast into the appropriate eventIn subclass, which contains methods for sending events of a given type.

Here is an example of sending an eventIn to a VRML scene containing a “House” node as follows:

DEF House Transform {. . .}

The Java codes for sending an event to change the *translation* field of the “House” node are as shown in Code Listing 1. (assume *browser* is the instance of a *Browser* class created from a previous call):

```
Node house = browser.getNode(“House”);

EventIn SFVec3f translation =

(EventIn SFVec3f)house.getEventIn(“set_translation”);

float value[3] = new float[3];
value[0] = 7;

value[1] = 8;

value[2] = 9;

translation.setValue(value);
```

Code Listing 1. EventIn processing.

In the above example, the translation value (7, 8, 9) is sent to the *translation* field of the Transform node of the “House.”

EventOut Processing. EventOut processing is similar to the EventIn processing. Once a node reference is obtained, all eventOuts of the node are accessible using the *getEventOut()* method. This method is passed in with the name of the eventOut and returns a reference to an EventOut instance if a matching eventOut name is found. ExposedFields can also be accessed, either by giving a string for the exposedField itself (such as “*translation*”) or by giving the name of the corresponding eventOut (such as “*translation_changed*”).

After an instance of a desired *EventOut* is obtained, two operations can be performed. The current value of the eventOut can be retrieved, and a callback can be setup to be invoked whenever the eventOut is generated. *EventOut* does not have any methods for getting the current value, so it must be cast into the appropriate eventOut subclass type, which contains appropriate access methods.

Similar to the eventIn example above, the current output value of the translation field can be read as follows:

```
float current[] = ((EventOut SFVec3f)
      (house.getEventOut(“translation_changed”))).getValue();
```

The array *current* now contains three floats with the x, y, and z components of the current translation value.

Getting Notification from VRML Scene. To receive notification when an eventOut is generated from the scene, the applet must first subclass the *EventOutObserver* class and implement the *callback()* method. Next the *advise()* method of *EventOut* is passed into the *EventOutObserver*. Then whenever an event is generated for that eventOut, the *callback()* method is executed and is passed the value and timestamp of the event. The *advise()* method is also passed a user-defined object. This value is passed to the *callback()* method when an event is generated. It can be used by the application author to pass user-defined data to the callback. It allows a single *EventOutObserver* subclass to handle events from multiple sources. It is a subclass of the standard java *Object* class; thus, it can be used to hold any data.

Using the above example again, the applet will be notified when the translation field of the Transform is changed, as shown in Code Listing 2.

```
public class MyObserver implements EventOutObserver {

  public void callback(EventOut value,

              double timeStamp,

              Object data)

   {

     //cast value into an EventOut SFVec3f and use it
       for intended operations

  }

}

  . . .

MyObserver observer = new MyObserver;

house.getEventOut(“translation_changed”).advise-
   (observer, null);
```

Code Listing 2. Getting notification from the VRML scene.

When the eventOut from the *translation* exposedField occurs, *observer.callback()* is invoked.

The EventOut class also has an *unadvise()* method that is used to terminate notification for that eventOut. The original *EventOutObserver* instance is supplied to distinguish which observer should be terminated if a single eventOut has multiple observers.

CVE Scene Construction

With the VRML EAI mechanism, we can construct a VRML-based CVE with avatars' representing users and shared objects with which users can interact.

Constructing Avatar. Avatar is a special shared object in a CVE virtual environment to represents the user's spatial location and interaction. It walks through the virtual space by following the user's navigation step, and it contains features to control the interaction between the user and the CVE, for example, mapping the user's viewpoint to the virtual world.

VRML's node mechanism can be used to support the concept of avatar. A VRML node is used to define an avatar, which includes the definition for the avatar's height, radius, and a 3-D model. A node is added into the CVE when the new user joins in the virtual environment. The two ways of node-creation methods provided by EAI: createVrmlFromString and createVrmlFromURL can be combined to embed the user's avatar into the virtual world. An avatar handler is created through a String, whereas the avatar's model is created through a URL and embedded into the avatar handler.

After creating avatar, its location and orientation in the virtual environment should be updated at run time to reflect the user's movement and interaction. EAI's VRML scene access mechanism can be used to update a user's avatar, i.e., by sending events to the eventIns of nodes inside a scene. From the process of creating an avatar node, the instance of the desired eventIn for setting avatar's location and rotation can be obtained. Once the instance of eventIn is created, an event can be sent to it.

To get an avatar's location and orientation, it requires a sensor to track the movement of a user's navigation in a CVE. As VRML ProximitySensor can output the changes of a viewer's position and orientation when the viewer is within the range covered by the ProximitySensor, a ProximitySensor node, with its size parameter to cover the whole CVE scene, can be used as the observer for the user's navigation. An EventOut Observer for the ProximitySensor and its callback function are used to get the notification when an eventOut is generated from the scene. Once a user moves, rotates, or enters/exits a defined region in the multi-user VRML world, two fields of the ProximitySensor node, one for location and the other for rotation, will reflect such movement by sending out the relative events.

But using only the ProximitySensor is not enough to define an avatar. NavigationInfo node should be used to define an avatar size, navigate speed, and type. We further use a Collision node to detect and to prevent two avatars from colliding. In the VRML file for the CVE, Code Listing 3 is added for the multi-user application.

```
NavigationInfo{

    avatarSize [0.25, 1.75, 0.75]

    headlight TRUE

    speed 2.0

}

Collision {

    collide TRUE

    children[

      DEF AvatarRoot Group{

        children[

        ]

      }

    ]

}

DEF LocalAvatarSensor ProximitySensor{

    center 0 0 0

    size 1000 1000 1000

}
```

Code Listing 3. Creating avatar nodes in VRML-based CVEs.

When a new user joins in, its avatar node will be added as a child to the AvatarRoot, which is a child of the collision node.

Creating Shared and Interactive Objects. In VRML, the fields of many nodes can be changed dynamically. These changes are made by routing events to the nodes. An event is a message sent to the node informing it that one of its fields should be changed. The process of the interactions on the 3-D interactive objects is illustrated in Fig. 2.

The VRML sensors provide a new level of user involvement with the virtual world. To interact with the virtual world, the sensor nodes are defined to monitor the inputs from the users. After the desired inputs were given, the sensor's eventOut will be routed to another sensor/script/interpolator's eventIn. Finally, the event will be routed to the animated object and the computed value will be set to the animated field. The VRML scene will be redrawn and show the animation of the objects to the user. In this

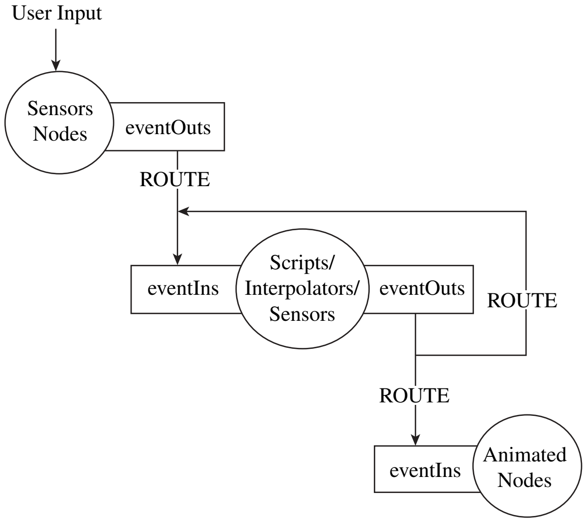


Figure 2. The process of interactions in VRML objects.

process, the routing requires that the eventOut and the eventIn have the same type.

For a multi-user collaborative virtual world, it is not enough to just allow an individual user to interact with the virtual objects or to allow the users to see each other's embodiment avatar. The CVEs also contain some shared interactive objects to make it more realistic and improve users' ability to interact with others as well as with the shared virtual world. For example, one user turns on a light and the other users should see the light in the virtual scene turned on. The states of these interactive objects are shared and can be changed by any user. It will be displayed locally, and the changes of the states will be propagated to other users through the network. Other more complex interactive objects are also required.

As discussed, the mechanism in EAI can be used to observe certain fields of a VRML node. As in the avatar node, it is also used to set parameters' values to the interactive objects. Once the value of the specified/affected field of the interactive object's VRML node is changed (an EventOut is sent from that field), the callback function will be called with the new value of that field. Hence, the system is notified when any changes happen and the operation to be carried out is implemented by the callback() function.

Although the basic idea is similar with creating avatar, implementing the interactive object is a little different. Because the interactive object and its behaviors have to be defined, the desired node can be obtained directly from the VRML scene. The node is designed in such a way that only one eventOut field is needed to be observed and only one eventIn field can trigger the status change of an interactive behavior. This will help to minimize the complexity of an EAI-aware Java applet because it only needs to observe one eventOut and set one eventIn.

To initialize the Event Observer in the client-side Java applet, information of the interactive objects has to be sent to the applet. The information includes the name of the Sensor node that will trigger the interaction, the name of the Script node that will carry out the operations for the interaction after it has been triggered, and the names of the fields in the Sensor and the Script that should be observed. A special Anchor node in the VRML file for the CVE can be defined for such a purpose as shown in Code Listing 4.

```
DEF InteractiveObjectsInfo Anchor{

    description

      "2/light/LightTS/isActive/lightScript/lightIsActive/

          door/DoorTS/isActive/DoorScript/doorIsActive/"

}
```

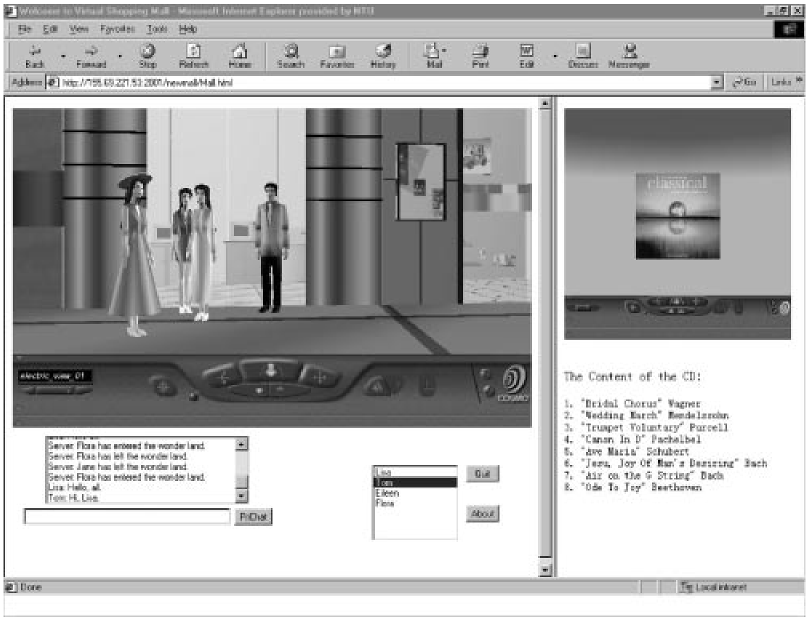
Code Listing 4. Creating sample shared interactive object information nodes in VRML-based CVEs.

The required information about the interactive objects in the CVE is provided by the description field of the node. In the above example, this Anchor node defines two shared interactive objects. One is the light, and another one is the door. With the techniques for creating avatar and synchronizing and maintaining a consistent virtual world among multiple users, 3-D VRML-based CVEs can be constructed. We have developed various Web-based CVEs using VRML to allow multi-user collaborative interaction, as shown in Figs. 3 and 4.

WEB-BASED CVE ISSUES

As discussed in the previous sections, a Web-based CVE can be constructed using VRML/X3D/Java3D. However, due to the heterogeneous nature of the Internet, system scalability, virtual world database delivery, and consistent world states maintenance across all participants in a Web-based CVE become more challenging issues than with its non-Web-based counterpart. This is particularly true if a Web-based CVE may have potentially a very large number of users at a time, which can easily overload a fast network. Real-time interactions require powerful networking capabilities to support large numbers of concurrent users. The underlying network support provided by Hyper Text Transfer Protocol (http) is insufficient for Web-based large-scale CVE. It is inadequate for supporting lightweight interactions and real-time streaming. To maintain collaboration, dynamic scene changes will need to be simulated by a variable combination of message passing, user commands, or entity behavior protocols. So Web-based CVEs will demand significant real-time data communication in addition to http-based client–server interactions.

As computing resources are limited, obvious problems will arise once the number of users in a Web-based CVE rises beyond a certain limit. One may expect a Web-based CVE to produce undesirable effects such as choppy rendering, loss of interactivity, and alike, due to lack of processing power to handle the ever-increasing load. If such a case


Welcome to Virtual Shopping Mall - Microsoft Internet Explorer provided by NTU
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss Messenger
Address http://155.69.221.53:2001/newmall/Mall.html
Go Links
electric_view_01
COSMO
PriChat
Quit
About
Done
Local intranet

Server: Flora has entered the wonder land.
Server: Flora has left the wonder land.
Server: Jane has left the wonder land.
Server: Flora has entered the wonder land.
Lisa: Hello, all.
Tom: Hi, Lisa.

Lisa
Tom
Eileen
Flora

The Content of the CD:

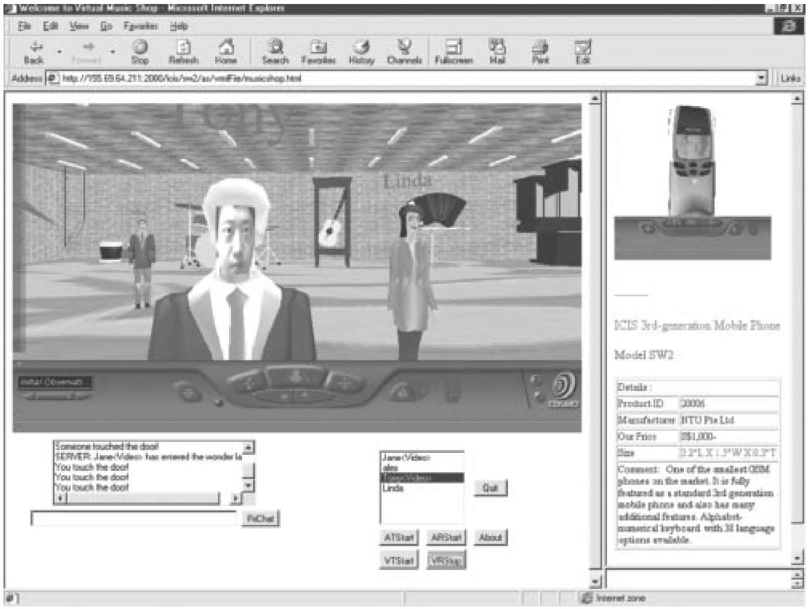
1. "Bridal Chorus" Wagner
2. "Wedding March" Mendelssohn
3. "Trumpet Voluntary" Purcell
4. "Canon In D" Pachelbel
5. "Ave Maria" Schubert
6. "Jesu, Joy Of Man's Desiring" Bach
7. "Air on the G String" Bach
8. "Ode To Joy" Beethoven

Figure 3. Web-based VRML virtual shop CVE.

occurs, the collaboration will be lost. Thus, it requires a protocol designed for a Web-based CVE in addition to http. Virtual Reality Transfer Protocol (VRTP) (7) is designed for such a purpose. It is an application-level protocol that provides network capabilities for a large-scale interlinked VRML world. It provides additional capabilities, for many-to-many peer-to-peer communications plus network monitoring need to be combined with the client–server capabilities of http. To accomplish this task, VRTP is designed to support VRML worlds in the same manner as http was designed to support interlinked HTML pages.

VRTP define four basic components: client components, server components, peer-to-peer components, and monitoring components. By calling the services of these components, the CVE application can have the capability to act as a client to request 3-D world data from other application; act as a server to share its 3-D world data with others; act as peers to attend group communication; and act as a monitor to diagnose and correct the network problems. However, it should be noted that VRTP needs multicast-enabled network backbone services.

Besides providing an efficient protocol for Web-based CVEs, it is important that appropriate virtual world database delivery methods are used in combination to improve the system scalability. It is not practical to download and store an entire virtual world in the user's local machine each time and render the whole scene, because it will often lead to a long start-up time to wait for the whole database to be completely downloaded, and it could be unacceptably long due to Internet latency. This scenario is particularly


Welcome to Virtual Music Shop - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address http://155.69.64.211:2000/lcis/sw2/as/vrmlFile/musicshop.html
Links
Linda
Virtual Observatn
COSMO
PriChat
Quit
ATStart ARStart About
VTStart VRStop
Internet zone

Someone touched the door!
SERVER: Jane<Video> has entered the wonder la
You touch the door!
You touch the door!
You touch the door!

Jane<Video>
alex
[illegible]<Video>
Linda

ICIS 3rd-generation Mobile Phone

Model SW2

Details :	
Product ID	20006
Manufacturer	NTU Pte Ltd
Our Price	S$1,000-
Size	3.2"L X 1.5"W X 0.5"T
Comment: One of the smallest GSM phones on the market. It is fully featured as a standard 3rd generation mobile phone and also has many additional features. Alphabet-numerical keyboard with 38 language options available.	

Figure 4. Web-based 3-D VRML virtual community CVE with live audio and video avatar face. (Note: Live audio and video data are transmitted in peer-to-peer.)

true for a large-scale CVE. Furthermore, all participants must maintain up-to-date object status, but updates for objects that they are not interested in waste processing power and network bandwidth (8). A popular solution to this issue is to partition the CVE into regions or cells. The virtual world partition may be static (9) or adaptive at run time (10). Once CVE is partitioned, each server may be assigned to provide service to a particular region. For example, in Spline (11), a region server maintains a record of all object models and states, as well as of interaction among all users in a given region. When a user enters a new region, the corresponding region server delivers object model data and initial information about the state of objects in the region to the user process. After this initial download, the user process obtains further object states' updates via group multicast communication. Algorithms may be introduced to achieve balanced workload for each server.

The virtual world database delivery may also be improved by an on-demand transmission approach as proposed in CyberWalk (12). It achieves the necessary performance with a multiresolution caching mechanism. First, it reduces the model transmission and rendering times by employing a progressive multiresolution modeling technique. Second, it reduces the Internet response time by providing a caching and prefetching mechanism. Third, it allows a client to continue to operate, at least partially, when the Internet is disconnected. The caching mechanism of CyberWalk tries to maintain at least a minimum resolution of the object models to provide at least a coarse view of the objects to the viewer.

Demand-driven geometry transmission (13) is another possible solution for improving content delivery performance. The proposed solution for content delivery includes area of interest (AOI) for delivering only part area of content; level of details (LOD) for delivering only given detailed data for an object; pre-fetching for hiding content delivery delay; and client memory cache management for improving client performance.

The virtual world content delivery and caching performance may further be improved by Pervasive Virtual Environment Scene Data Caching as proposed in MACVE (14). In traditional CVE systems, when a user navigates through a cell or region, it downloads the VE scene data of the cell/region and caches them for the possible future reloading. Whereas in MACVE, the cached virtual world data are used as an additional content delivery point if the caching user machine satisfies the Trusted User Node condition. Once a Trusted User Node caches virtual world data, it will be able to provide content delivery service to other users who are geographically located closer to the Trusted User. This process will be faster than downloading the same data from the server. Furthermore, it can reduce the workload and network bandwidth consumption on the server. MACVE achieves the Pervasive Virtual Environment Scene Data Caching by cloning Cell Agent and migrating it to the corresponding Trusted User Nodes. A Cell Agent may have multiple cloned ones running at different Trusted User Nodes. When a new user node needs to fetch the CVE scene data, it will be directed to the "nearest" Cell Agent.

To provide low latency interaction, world states synchronization methods should be carefully designed to incorporate with the corresponding world data delivery methods. For example, in Spline (11), world states changes are updated only to a small group of users in the region that are actually interested in it, rather than to all other concurrent users in the CVE via group multicast. Furthermore, Spline uses the "approximate database replication" idea on which DIS is founded (15). Spline world states are only approximately equal in the sense that different users observe events occurring at slightly different times. The time it takes to update a world model running locally in one user process with respect to changes made by a remote process depends on the network traveling time between the two user processes. The time difference is usually less than a couple of hundred milliseconds, which is still within the tolerance level for collaborative interactions. Further improvement may be achieved by fine-tuning the data communication approach based on the data type in the virtual world model. For example, state updates for small objects can be sent using multicast or unicast UDP or TCP packets, whereas graphic models, recorded sounds, and behaviors are identified by URLs and communicated using standard http protocols. And real-time streaming data, such as live audio/video, should be communicated using peer-to-peer direct transmission approach.

Another possible method that can be used to reduce user interaction (or world states changes) traffic is to use motion prediction (8) or dead reckoning (16). This method reduces network latency effects and improves the smoothness of collaborative interactions by extrapolating a virtual entity's next movement based on its history motion vector, thus reducing the number of states update packets required for synchronizing CVE states. However, it should be noted that this method does not work well with predicting user's movement in a CVE as users often have abrupt change in their motion.

To achieve good interactivity, data generated for scene consistency should be transferred as soon as possible to maintain the synchronization among remote users. This kind of data should have higher priority when communicating. Whereas persistent data are transparent to clients, it can tolerate a certain level of delay so long as the correctness of final virtual world states can be ensured. So, they can be queued and scheduled for processing at a lower priority. Thus, more bandwidth can be allocated for high-priority data.

SUMMARY

Similar to all other Internet applications, bandwidth is an important factor to be taken into consideration in Web-based CVEs. Because of limited bandwidth and appreciable network latency, Web applications have to be carefully constructed to avoid high bandwidth interaction between the client application on the user's machine and the server process on the other side of the Internet. For Web-based 3-D interactive CVEs, due to the heterogeneous nature of the Internet, the most challenging issues of system scalability, virtual world database delivery, and consistent world states maintenance across all participants should be addressed with efficient protocol and data communication architecture incorporated with appropriate virtual world partition, content delivery, and world states synchronization methods if it is intended for use by a large number of concurrent users over the Internet.

The WWW today is no longer just text and image displayed on the two-dimensional computer screen. The evolution of technology—higher computer processing speed and higher bandwidth—has enabled more interactivity, more content, and more realism on the Internet. Through the personal computers with network connections, Internet users can now "live" in three-dimensional worlds, where they can meet and interact with other users. They can now show their emotions and behavior through the representation of an avatar and communicate with each other using live audio and video streaming. With web3D, the future of the Internet will be an unlimited collaborative cyberspace where people can gain access to a shared 3-D interactive multimedia environment.

REFERENCES

1. Web3D Consortium, *The Virtual Reality Modeling Language*. Available: http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/. 21 May 2006.
2. R. Lea, K. Matsuda, and K. Miyashita, *Java for 3D and VRML worlds*, Indianapolis IN: New Riders Publishing, 1996.
3. Web3D Consortium, *X3D Overview*. Available: http://www.web3d.org/x3d/overview.html. 21 May 2006.
4. Web3D Consortium, *Why use X3D over VRML 2.0? Here are 10 compelling reasons*. Available: http://www.web3d.org/x3d/x3d_vs_vrml.html. 21 May 2006.
5. Sun Microsystem, *Java3D API*. Available: http://java.sun.com/products/java-media/3D/reference/api/index.html. 21 May 2006.
6. B. Roehl, J. Couch, C. Reed-Ballreich, T. Rohaly, and G. Brown, *Late Night VRML 2.0 with Java*, Emeryville, CA: Ziff Davis Press, 1997.
7. D. P. Brutzman, M. Zyda, K. Watsen, and M. R. Macedonia, Virtual reality transfer protocol (VRTP) design rationale, *Proc. of the 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises,* Massachusetts Institute of Technology, Cambridge, MA, IEEE Computer Society, 1997, pp. 179–186.
8. R. W. H. Lau, F. Li, T. L. Kunii, B. Guo, B. Zhang, N. Magnenat-Thalmann, S. Kshirsagar, D. Thalmann, and M. Gutierrez, Emerging Web graphics standards and technologies, *Computer Graphics and Applications, IEEE*, **23**: 66–75, 2003.
9. T. A. Funkhouser, RING: A client-server system for multi-user virtual environments, *Proc. of the 1995 symposium on Interactive 3D graphics*. Monterey, CA, ACM Press, 1995, pp. 85–92.
10. R. W. H. Lau, B. Ng, A. Si, and F. Li, Adaptive partitioning for multi-server distributed virtual environments, *Proc. of the tenth ACM international conference on Multimedia*, Juan-les-Pins, France, ACM Press, 2002, 271–274.
11. R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W. Yerazunis, Diamond park and spline: A social virtual reality system with 3d animation, spoken interaction, and runtime modifiability, *Presence: Teleoperators and Virtual Environments*, **6**: 461–481, 1997.
12. J. Chim, R. W. H. Lau, V. Leong, and A. Si, CyberWalk: A Web-Based Distributed Virtual Walkthrough Environment, *IEEE Transactions on Multimedia*, **5**: 503–515, 2003.
13. D. Schmalstieg and M. Gervautz, Demand-driven geometry transmission for distributed virtual environments, *European Association for Computer Graphics 17th Annual Conference and Exhibition*, Poitier, France, 1996, pp. 421–432.
14. L. Zhang and Q. Lin, MACVE: A mobile agent based framework for large-scale collabortive virtual environments, *Presence: Teleoperators and Virtual Environments*, 2006, In Press.
15. R. C. Waters and J. W. Barrus, The rise of shared virtual environments, *Spectrum, IEEE*, **34**: 20–25, 1997.
16. R. C. Hofer and M. L. Loper, DIS today [Distributed interactive simulation], *Proc. of the IEEE*, **83**: 1124–1137, 1995.

QINGPING LIN
LIANG ZHANG
Nanyang Technological
University
Singapore

C

COMPUTER GAMES

INTRODUCTION

Computer (and video) games have gained significant interest in recent years, particularly the United States, where in total game sales for 2005 were over 10.5 billion US dollars, which repersented a 6% improvement over the game sales in 2004. Demographically, 69% of American heads of households play computer and video games. In the United Kingdom, which owns the world's third largest computer game market, 82% of 9- to 19-year-old youngsters own a game console. In this article, we discuss the existing technologies and the design issues of computer games. As computer and video games are closely related and technologically comparable, without loss of generality, our discussion will use the term *computer game* to also include video games.

GAME HISTORY

Computer game has a long history, in which we can trace back its root from 1947, when Thomas T. Goldsmith, Jr. and Estle Ray Mann designed the first game for playing on a cathode ray tube in the United States. Here, we are not intended to elaborate the full computer game history. Instead, we focus on the technological evolution of computer games and technical issues occuring throught the change. During the 1960s and 1970s, games developed were simple, primitive, and mainly in two dimensions. Many games of different types were developed. Two of the most unforgettable examples are Space Invaders and Pac-Man. In addition, handheld game devices were also developed, most of which were hard coded to run only a single game. For game playing, game players used simple button controllers, keyboards, or joysticks to control their game characters. The main forms of feedback were offered through screen displays with limited color and resolution as well as simple sound outputs. In the 1980s, there was a major growth in computer game technologies. For hardware, a variety of personal computers and game consoles were developed. For software, 3-D games and network games were first developed. In addition, different forms of input and output devices were developed, which included color monitors, sound cards, and various types of gamepads. They offer game players better game feedback and greater flexibility in controlling game characters. In the 1990s, games developed planted the seeds of today's game development. Many classic game types, including first-person shooters (FPS), real-time strategy (RTS), daily life simulators, and graphical multiplayer games, were developed during this period. Also, there was a trend for developing 3-D games. Nowadays, many new games are developed based on these classic games. The major difference of the new games from the classic ones is that the new games are mainly in three dimensions. Hence, hardware graphics accelerators were urged into development to support real-time rendering of 3-D game content. To take the advantage that human visual sense has a dominant influence of a game player to determine whether a game is good, game companies put their very first priority in enhancing the graphics output for their games. They put advanced graphics content, such as high detailed 3-D models, texture images, and various special effects, into the games. Multimedia elements, such as videos and songs, are also used to enrich the game content. However, such arrangements increase the complexity of the hardware requirement for running computer games. It also demands the development of efficient algorithms to manage such a variety of game content. To optimize both the manpower and time spent in game development, game developers begin to make use of ready-made game engines (please refer to the Game Engine section), which comprise many useful tools to support general game functions, to build their games.

When we go through the history of computer games, we note that, during the early stage, games were mainly simple and small. They could be generally handled by simple computation and graphics processing power. Hence, there were not really any stringent requirements put on the development of these games. Later, when game developers turned their focus to 3-D games, working out both hardware solutions and software solutions in supporting real-time rendering of 3-D graphics has subsequently become a critical part of game development. Besides, game physics and game artificial intelligence (AI) also played an important part of the games. Whereas game physics provides support to collision detection and motion control of game characters, game AI offers autonomy to the nonperson-controlled game characters to govern how these characters behave in the game environment. Recently, as multiplayer online games begin to dominate the game market, issues such as network latency, system scalability, and security are turned out consequently. Eventually, these issues make the game development face various technological design issues from different disciplines in computer sciences.

TYPES OF GAMES

2-D and 3-D Games

Technologically, computer games can be broadly categorized into 2-D and 3-D games. 2-D games generally manage the game environment into a logical 2-D space, where the game objects can be moving around and interacting. Practically. a majority of 2-D games, such as Pac-Man, Load-Runner, and Mario, can be implemented using a simple tile-based concept (1), which partitions the game environment into cells that are hosted in a 2-D array. Then different states can be assigned to an individual array element to logically represent different game objects and game scene elements in the game environment. When some objects are moving around in the game environment, the correspond-

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

ing elements of the 2-D array are then updated to reflect the change. To render game objects or game scene elements, it can be done as easy as using simple graphics primitives, such as line, point, and polygon, or alternatively using picture images to present the game objects or game scene elements in a more impressive and realistic way.

As the successor of 2-D games, 3-D games offer greater attractions to game players in terms of game interaction and visual effect. The game environment of such games is hosted in a 3-D space. As one more dimension is provided, game players have been offered a greater degree-of-freedom (DOF) in controlling their game characters and interacting with other game objects. They can also navigate and interact in the game environments with a variety of camera views. In addition, the 3-D setting of these games also gives a variety of advanced computer graphics effects a sufficient environment for implementation, which can then be visualized by the game players. Examples of the graphics effects range from some low level graphics techniques, such as lighting and shadowing, to some high level graphics techniques, which cover a wide variety of natural phenomenon simulations (2). In addition, computer animation techniques (3) can also be employed to make the game objects move in a realistic way. Unlike 2-D games, which can natively be displayed by most of the 2-D display controllers, 3-D games need a rendering process to convey 3-D game content to present on 2-D display devices. In light of supporting interactive game playing, the rendering process must be carried out with a frame rate of 25–60 frames per second, which means 25–60 pictures should be produced every second for the relevant portion of the game content for display purposes. To support such a performance, the rendering process is often needed to carry out by hardware graphics accelerators.

Multiplayer Online Games

Multiplayer online games have been developed as early as the late 1980s. The uniqueness of this game type is that it connects people from geographically dispersed locations to a common game environment for game playing. One of the early developed online games was the *Modem Wars,* which was a simple 2-D game designed for the personal computer—Commodore 64. The game connected two game players using modems into a shared game environment. During game playing, game players can interactively move the game items around the game environment to attack any enemy items located within a certain predefined range. Despite the game design being simple, it set a good foundation for the multiplayer online game development.

Nowadays, multiplayer online games have become one of the most popular game types. Unlike the old days, instead of running such games on peer computers and connecting the game machines through modems, the new online games are generally hosted in some server machines running on the Internet, which are referred as *game servers.* Game players from different geographical locations can connect to the games through some broadband network connections using their preferred game platforms, which can be a computer or a game console. Such game platforms are referred as *game clients.* The first commercial multiplayer online game of this type was, *Meridian 59,* published in 1996 by 3DO (4). Thereafter, some major game types, including FPS, RTS, and RPG, have dominated the market of multiplayer online games. Technically, the main responsibility of a game server is to keep track of the actions issued by the game clients and to propagate these actions and the state updates of the game environment to the relevant game clients. The game server also needs to build up a profile for each game player to store the game state and the possessions of the player, such that the players can keep on playing the game in future based on the saved profile. Today, most of the game developers focus on two major issues when developing multiplayer online games. First, they try to seek good game server architectures for hosting games to allow a larger number of game players to join and play their games. Second, to attract more people to play their games, the game developers take advantage of the advancement in computer graphics technologies to make their game have very impressive 3-D graphics presentations and visual effects.

Handheld Games

In contrast to computer-based or game console-based games, handheld games are run on machines with a small machine size. It allows people to carry it along anywhere and play around with it at any time when they are free. Generally, such machines can be referred to as dedicated handheld game consoles, personal digital assistants (PDAs), or mobile phones. Due to the hardware limitation, such game devices often suffer from small screen size and limited processing power and storage space, as well as the problem of short battery life. These problems do not only impose difficulties to handheld game development, they also make some people reluctant to play handheld games. Fortunately, these shortcomings have been addressed or have been worked around during recent years.

The first handheld game console is *Tic Tac Toe,* which was made in 1972. Similar to most of the early handheld game consoles, it came with one hard-coded game only. This limitation lasted until 1979, when the first handheld game console with changeable cartridges, *Microvision,* was developed. In general, as handheld game consoles at that period suffered from the small screen size and limited battery life problems, handheld games had not received great success. With the release of Game Boy (5) in 1989, which came with a monochrome display with improved resolution, used rechargeable batteries, and had a long list of game cartridges for game players to pick and play, handheld games began to attract a significant amount of game players. More importantly, Game Boy virtually set the "design standard" for today's game consoles. In addition, in 1998, the color display version of Game Boy was released to further improve the attractiveness of handheld game consoles. Nevertheless, up until the release of Game Boy and its color display version, as the processing power of the handheld game consoles was still quite limited, most of the games developed for the handheld game consoles were still essentially 2-D games.

Starting in 2000, there was a dramatic development in handheld game consoles, particularly in terms of computation and graphics processing power. In light of this improvement, 3-D games had been engaged to these devices, an example of which was in Game Boy Advance (5). On the other hand, useful accessories such as network connections, external memory storage and new types of input devices were added to the game consoles. Regarding the network capability, examples could be found in Nokia N-Gage (6) and Nintendo DS (5), which made multiplayer online games possible to be supported. Sony made use of UMD disks and Memory Stick Duo as a media to extend the amount of storage of its newest handheld game console, Sony PSP (7). For input device, Nintendo DS adopted a touch screen approach, where game players could use a stylus or even the player's finger as an input method to control the games objects.

On the other hand, similar to dedicated handheld game consoles, PDAs and mobile phones are also featured with high mobility, which makes such devices become alternate platforms for handheld games. More importantly, from the business point of view, putting games on essential devices, such as PDAs or mobile phones, is favorable as this frees people from investing or carrying addition game devices for entertainment. In addition, mobile phones and modern PDAs also natively come with network capability to provide a critical support for running online games. However, before PDAs and mobile phones become substantial handheld game devices, the technical problems of these devices, such as small screen size and limited storage space, which could also be found in dedicated handheld game consoles, must be solved.

GAME DEVELOPMENT PROCESS

Nowadays, making a game no longer focuses only on working out the game logic or game features and the graphical display for the game. Depending on the resource availability and the business strategy of a game company, a variety of associated tasks may also involve in the modern game development process. These tasks include game hardware and software development, media construction, localization, and even the handling of the cultural and social issues:

- **Game Hardware:** Gave hardware refers to the development of game consoles and game input/output devices. Such development may usually introduce new attractions to game players. It also offers game companies niches in developing proprietary hardware-specific games or obtaining license fees from developers who develop games on such game hardware. However, as developing game hardware usually requires quite a significant amount of investment in terms of manpower, time, and money, only large game companies can afford such development. More technical design issues on game hardware will be discussed in the Modern Game Design Issues Section.
- **Game Software:** Game software refers to the technical part of the game development process. It involves the development of various game software components, which may include some hard core game functionality, such as game content management and rendering, game animation, game AI, and game physics. In addition, it may also involve the development of game networking and game security, which depends on the type of the game for development. More details on this topic will be discussed in the next two sections.
- **Media Construction:** Media construction refers to the artistic part of the game development process. It involves the development of game content by using different types of media, which may include image, 2-D/3-D graphics model, audio, video, and motion capture information (8). As media offers the presentation of game content, which determines how game players perceive a game, media construction becomes an inevitable part of the game development process. Nowadays, many game companies have been investing a significant amount of resources in the media construction process.
- **Localization:** Localization is the process of turning a computer game into a country-specfic or a target-market-specific version, which helps a computer game broaden its market share and helps introduce country- or market-specific attractions to the game players. The localization process can be done as simple as by conveying the language of the user interface and the textual content of a game. In a more complicated way, we may further change the game characters or other game media content to country- or market-specific ones. Furthermore, we may even consider altering the storyline of a computer game to suit the culture or custom of the country-specific or target-market-specific game players.
- **Cultural and Social Issues:** During recent years, there has been a rising concern of the cultural and social effects of computer games on humans, especially on the youngsters. On the one hand, more and more people are getting addicted to computer game playing, particularly after the release of multiplayer online games, which likely has a bad effect to the academic performance of addicted student game players. It may also significantly reduce the amount of time for people to participate in social activities. On the other hand, the release of game titles with violent and sexual game content imposes negative ethical effect to the young people. Although there is not a consensus on the handling of the cultural and social issues of computer games, the game companies should try their best to maintain a good attitude in addressing these issues during the game development process.

GAME ENGINE

Making computer games is a complicated task. From the hardware perspective, when constructing a game, game developers may need to deal with a wide range of hardware and software platforms as well as work hard on a number of game components. More specifically, a computer game may need to be designed to run on different game platforms, including computers and game consoles, which are usually controlled by different operating systems. Even under the same hardware platform, the game may need to be rendered by different graphics accelerators and relied on different graphics application programming interfaces (APIs) to drive the graphics accelerators.

From the software perspective, when developing a computer game, game developers typically need to work on a number of game components, in which the most essential ones include game content management and rendering, game animation, game AI, and game physics. Working out these components generally involves much effort and is very time-consuming.

To minimize the complexity of game development by hiding the differences in various game platforms and to help game developers put their focus on developing high level game logics and game features, a game engine has been developed; it comprises a set of basic game building blocks and provides a high level and unified abstraction for both low level graphics APIs and hardware game platforms. With the help of a game engine, the investment of game development, in terms of time, manpower, and cost, can significantly be reduced. Reputable examples of game engines include Unreal Engine 3 (9) and RenderWare (10). In practice, there are not any standards or rules to govern the exact game components to be included in a game engine. Game engine developers have a great flexibility to select the appropriate set of components to make their own engines. However, there are some major game components that are essential to most of the games and, hence, should be included in the development of a game engine:

- **Game Content Management and Rendering:** Game content management and rendering is one of the most core parts of a computer game. It comprises techniques to manage game content in a way to support efficient content retrieval and processes for making the game content to be displayable on the output device of the game. For game content management, most of the game engines adopt a scene graph approach (11), where the game objects and the graphics primitives are hierarchically maintained with a tree structure. As such, tree structure implicitly links up the game objects according to their spatial relationship, it provides sufficient information for a game to pick out closely located game objects to support game rendering and game object interaction evaluation. On the other hand, for game content rendering, particularly when dealing with 3-D game content, there will be a significant amount of processes as well as a variety of options to go through for converting game content from the 3-D representation into the 2-D one for display. These processes and major options are shown as follows:
  1. *Standard graphics rendering processes*(12), such as perspective transformation, clipping, hidden surface, and back face removal, are fundamental to render 3-D game content into 2-D images for display. As these processes are usually natively come with the hardware graphics accelerators, game engine developers need not develop their own algorithm to support these processes. Instead, consider that the standard graphics rendering processes typically run on various hardware graphics accelerators, which are controlled by different low level graphics APIs, such as OpenGL (13) and DirectX (14), game engine developers may better work out high level unified abstraction on these graphics APIs and hardware to help reduce the effort of game developers to construct games for a variety of game platforms.
  2. *Shading and texturing* are the major processes to fix the appearance of individual game object. The shading process takes in the lighting and the object material information to evaluate the appearance of a game object by applying certain empirical lighting models. Common options of shading include flat shading, Gouraud shading, and Phong shading. They offer a different degree of realism to the rendered game objects. On the other hand, texture mapping adds or modifies detail on the game object surface. Basic options of texture mapping include basic texture mapping, multitexturing, and mipmapping, which arranges captured or artificial images to add on the surface of a game object. Advanced options of texture mapping include bumping mapping and displacement mapping. They make use of certain specially designed "texture maps," which comprise geometry modifiers rather than generic images, to modify the appearance of the geometric details over the surface of a game object.
  3. *Advanced rendering options* can be taken to further enhance the realism of the overall game environment. They include, but are not limited to, reflection, motion blur, and shadowing. Adopting these options definitely helps attract game players' interests as they make the rendered game environment look more realistic by adding details of some natural phenomenon to the game scene. However, as such rendering options generally require time-consuming add-on procedures to be executed on top of the standard graphics rendering processes, taking such options would likely degrade the rendering performance of a computer game significantly. Therefore, game developers should put these options as optional choices for game players to take rather than set them as mandatory game features.
- **Game AI:** Game AI (15) is a way to give "lives" to the nonperson-controlled game characters (NPCs) of the game environment, it directs the way of the NPC to interact with the game environment or other game objects. Putting different game AI to an NPC can assign different behaviors to the NPC. In fact, one of the major reasons for computer games to be so attractive is that game players can find different challenges and fun when they play against the NPCs inside the game environment. To implement game AI, two major options are available:
  1. *Reactive techniques* are widely adopted in many computer games, as they are fully deterministic. Examples of these techniques include scripts,

rule-based systems, and finite-state machines. Such techniques take in some given game parameters or game states, which are then evaluated through predefined rules to produce deterministic results. Practically, reactive techniques are good for implementing high level tactical decisions.

2. *Planning techniques*, in contrast, are nondeterministic. From a given situation, multiple actions can be taken depending on the current goal or some selected factors. A planning algorithm can scan through the possible options and find the sequence of actions that matches the goal or the selected factors. For instance, A^* is the most reputable example of planning techniques. Practically, planning techniques are good at helping to search the best possible path for a game object to navigate in the game environment.

- **Game Physics:** Game physics (16) is developed based on the laws of physics to govern how each individual game object reacts with the game environment or other game objects. It also offers a way to support the simulation of some natural phenomenon. Typically, the reaction of a game object can be determined using mass, velocity, friction, gravity, or some other selected physical properties. In practice, game physics can be natively applied to help generate a realistic response for the collision or interaction of the game objects. Alternatively, game physics can be applied to drive the motion of a large amount of tiny particles for simulating some natural phenomenon, such as the flow of smoke and water, fire blaze, snow, and cloud.
- **Animation:** Animation (3) is a technique to drive the motion of the body of the game characters. Typically, there are several ways to produce animation in computer games. They are shown as follows:

1. *Key-framing*(17) requires animators to define and draw key-frames of a motion sequence of the game character to be animated. However, manipulating and coordinating the limbs of a game character via key-framing is a complicated and tedious task. In addition, it is also difficult to produce realistic and natural looking motions with key-framing.
2. *Inverse kinematics*(18) computes the pose of a game character from a set of analytically constrained equations of motion. It can generally produce physically realistic motions.
3. *Motion capture*(8) acquires movement information from live objects. The captured position and orientation information from motion capture can then be applied to game characters to drive their motions. This approach has been widely accepted as it helps produce realistic and natural looking character animations.

MODERN GAME DESIGN ISSUES

Since the release of the first computer game, computer games have become one of our major entertainments. During the years, as game hardware becomes much more powerful, the game players' expectation on both the visual quality and the performance of computer games has then been increased significantly. Such expectation has partially been tackled by the release of new computer graphics algorithms and hardware. However, due to the advancement in computer network technologies and the popularity in multiplayer online games, game design issues are no longer restricted to computer graphics or input/output devices. Instead, the game design issues have become multidisciplined and are much more challenging than ever. The following shows the modern technical design issues that game developers should need to pay attention on when developing their new games:

- **Advancement in Hardware Graphics Accelerator:** Owning to the increase in the demand for high visual quality and high realism of the graphics output of computer games, a brand new type of hardware graphics accelerator—Graphics Processing Unit (GPU) (19) has been developing, which offers a dramatic improvement in the rendering performance at the game clients. Such improvement is due to the fact that GPU allows the major rendering tasks, including the standard and the advanced game rendering tasks as well as game physics, to be executed in parallel rather than carrying them out in a sequential manner as in the traditional hardware graphics accelerator. To take advantage of such advancement in hardware graphics accelerators, game engine developers should seek ways to parallelize the algorithms used for implementing the game engine components.
- **Game Controller (Input):** Game controller (20) is the primary input device for game players to issue commands or actions to drive game characters or interact in a computer game. Common game controllers include keyboard, joystick, and mouse. To use these controllers, game players need to convey their actions by getting in to their minds with controller-dependent operations. However, such operations may not always match well with the activities or interactions of one's game playing, especially for the games about human's real-life activities, such as driving, dancing, musical instrument playing, and shooting. Recently, game developers began to realize that the design of the game controllers could be one of the critical determinants for the success of a computer game. Hence, they have been working out many different game controllers to attract game players' interests and offer a better game playing experience. On the one hand, game-specific game controllers have been developed; examples include the steering wheel for driving or racing games, the dance platform for dancing games, and light guns for shooting games. Through these controllers, game players can act naturally as performing real-life activities during their game playing. On the other hand, game developers actively create new types of game controllers. Examples include the EyeToy of Playstations (21) and the wireless controller, Wii Remote by Wii (22).

Particularly, EyeToy makes use of a webcam to capture human motion, which virtually forms a game controller for a game player to control and interact with the game. Wii Remote is a wireless game controller with high degrees-of-freedom to let a game player generate a large variety of free motions or perform human's real-life actions for game playing. These game controllers also lead to the development of new game types.

- **Game Feedback (Output):** Game feedback is the way for a computer game to give game players responses to their actions. It is the one of the most direct ways for a game player to feel and to be impressed as to how good a computer game is. The primary ways for offering game feedback is to put different graphics effects on the screen and to generate sound effects for responding to game players' actions. However, as more game devices are released, a greater variety of game feedback is made available for game developers to consider putting in their games. Broadly speaking, we may have three methods of game feedbacks, includies tactile, audio, and visual feedbacks. For tactile feedback, the most common form can be found in most of the game pads of modern game consoles. It is typically done by generating vibrations to certain game events or game players' interaction. Another form is the resisting force used in the steering wheel for the car games. For audio feedback, as new sound compression and speaker technologies are emerging, multichannel sound feedback is now made available to provide higher sound quality and to support 3-D sound rendering. In particular, 3-D sound offers a new type of spatial feedback for objects or interactions in a game environment, which originally can be a provided by the visual feedback only. For visual feedback, a conventional approach focuses on rendering 3-D graphics and visual effects to display on a 2-D screen. But as stereo display technologies mature and become widely available, a 3-D display is made possible as an alternative option for visual feedback.
- **Scalability:** Multiplayer online games are the fastest growing game type in recent years. They allow a large amount of remote game players to get connected for playing in a shared game environment. Existing multiplayer online games have been implemented with distributed game servers. However, some of the games, such as Quake III Arena (23) and Diablo II (24), maintain an individual game state for each game server, which is not shared among the servers and is essentially a set of separated client–server systems running the same game and may not be considered as a real multiserver system. EverQuest (25), in contrast, divides the entire game environment into distinct zones and maintains these zones by individual game servers. EverQuest allows a client to travel from one zone (game server) to another freely. Ultima Online (26) and Asheron's Call (27) adopted an approach simlar to EverQuest, but they divided the entire game environment into visually continuous zones. The boundary of each zone is mirrored at a neighboring server to reduce the lag problem and to improve interactivity when a user crosses from one zone to another. In addition, Asheron's Call is technically more advanced in that it may dynamically transfer a portion of the zone controlled by a given game server to any other lightly loaded server. Unfortunately, game object coherency is not considered.
- **Network Latency:** A unique characteristic of multiplayer online games is that game updates are required to send to remote game players over the network throughout the game playing sessions to renew the states of the local copies of shared game objects at the game players. More than that, as multiplayer online games involve time-dependent game data and continuous user interaction, the state update events are therefore *continuous*(28) in nature. However, due to the existence of network latency, such updates are likely arrived at the remote game players after some delay, which leads to state discrepancy of the shared game objects among the game players. This fact opposes the sufficient condition for supporting game player interaction, in which the state updates need to be presented to remote game players either without any delay or at least within a very short period of time.

  To cope with the latency problem, adaptations could be performed at either the user side or the system side. For user-side adaptation, Final Fantasy XI (29), which is one of the popular online games currently available in the market, imposes restrictions to game player. In this game, a player usually receives position updates of other game players with almost a second delay. To reduce the effect of such delay, first, players can only attack enemy objects, but not each other. Second, the enemy objects are designed to move very little while they are under attack by a player. Such game rules significantly limit the game features and the type of games that can be developed. For system-side adaptation, a popular approach is to use dead reckoning (30). With this approach, the controlling game player of a game object runs a motion predictor for the object. Other game players accessing the object also run the same motion predictor to drive the motion of the local copies of the object. The controlling game player is required to keep track of the error between the actual and predicted motions of the object, and sends the updated motion information to the other game players when the error is higher than a given threshold. Although this approach is very simple, it does not guarantee that the state of a shared object could be synchronized among all game players.

  Recently, a trajectory preserving synchronization method (31) has been developed to tackle the problem. The method runs a reference simulator for each object on the game server. Each of the game players interested in the object, including those that access the objects as well as the owner of the object, will execute a gradual synchronization process on the local copy of the object to align its motion to that of the reference simulator running at the server. The method effectively reduces the discrepancy suffered by remote game players.

- **Game Cheating:** Game cheating generally refers to the activities that modify the game experience to give a game player an unfair advantage over the other players. Common ways of game cheating can be done in terms of user settings, exploits, and using external software. To cheat using user settings, one can change game data, game models, game environment, or game devices to make one to play a game in an easier way. Exploits, on the other hand, refer to the use of existing bugs of a game, or game features or tricks that are reserved for developers for testing purposes and are unintended to release to game players, to gain certain advantages for game playing. Other than the above, people may develop external software to help modify the normal behavior of a game to let a game player perform game cheating. To cheat in online games, packet tampering is common. It is performed by altering the game data sent between the game server and game clients. To prevent game cheating, game companies should take appropriate measures. Typical solutions may include the use of anticheating software and banning suspected cheaters from playing a game.

FUTURE TREND AND CONCLUSIONS

Based on the development in computer games, we enumerate and elaborate below a number of emerging issues on the future development of computer games, much of which serves the purpose of highlighting the direction for future research and development. In addition, it is anticipated that many new games will emerge in time as the technologies evolve. What we have pointed out here are only some important emerging issues and technologies, and we are leaving the readers to comment and decide which ones are the most important.

- **On-Demand Transmission:** Nowadays, multiplayer online games are usually set out with a large scale scene environment together with high quality game objects to attract more game players. A standard way to distribute such a large game content is to put it on CDROMs/DVDs for game players to purchase and subsequently install at their local machines. This scenario works fine if players' local machines have sufficient storage to hold the complete set of game content. Clearly, due to limited storage, it is difficult for handheld devices or portable game consoles to access multiplayer online games in this way. To loosen this limitation, a game-on-demand approach (32) can be adopted, which uses a prioritized content delivery scheme to help identify and deliver relevant game content in suitable quality to support users' game playing. In addition, the game-on-demand approach also serves games with very large scale game scenes and highly detailed game objects to be accessible to machines with any kind of network connection speeds. On the other hand, nowadays, game companies typically continue to develop new game scenes and game objects after they have released their games. It is not only because game companies want to gain revenue before they have finished developing their games, they also need to add new attractions to keep motivating people to play their games. With the game-on-demand approach, this process can be done totally transparently to the game players without requiring them to install patches as they do now, as new content can also be streamed progressively during game playing.
- **Web-Based Client Technologies:** Recently, there has been a dramatic development in client technologies for Web-based applications. Such technologies are emerging as important development tools and are critical for the future development of Web-based computer games. Typical examples include Flash, Java, and ActiveX. They help improve the user interface control of Web-based applications by getting through the problem of limited interaction control and web page reload when a user performs an action. More than that, client technology, like Flash, serves users with an interactive user interface as well as the multimedia support. The only requirement for adopting these client technologies is the preinstallation of application interpreters and plug-ins. In addition, a more advanced option, Asynchronous JavaScript and XML (AJAX) (33), has been made available since 2005 to provide more interaction control, such as context-sensitive menus and window interface, for developers to construct more serious Web-based application. AJAX also relaxes the need of preinstallation or maintenance of add-on plug-ins, and requires only a minimal amount of data transmission for application download, as they are driven by light-weighted JavaScript and XML codes. Finally, during run-time, AJAX can collect information from Web servers to update only a required part of the application content without the reloading web page.
- **Security Challenges:** Security control in existing computer games is partially addressed. However, as there is a trend that more and more computer games will be hosted on the Internet, the risk of these games being accessed by unauthorized people and being cracked for cheating will be increased as well. To address this problem, secure socket layer (SSL) could offer a good choice. It provides endpoint authentication and communications privacy over the Internet using cryptography. In particular, once a user has logged into a computer game, such data protection processes would be done transparently to the user.
- **Computer Games on Grid:** Grid Computing (34) is considered as a hot research area in recent years. Grid connects computers, storages, sensors, and services via fixed-line or wireless Internet (and other) networks. The goals of grid computing include resource sharing, coordinated problem solving, enabling dynamic virtual organizations/world, and thus service-oriented architecture is considered as a promising direction. Advantages of grid include data and performance reliability (avoid single point of failure), performance (avoid network and computation bottleneck), and resilience (guarantee existence of data and its

backup). In line with the change of new grid concepts, computer games can be considered as one important service to this emerging grid virtual world.

BIBLIOGRAPHY

1. Tile Based Game, Available: http://www.tonypa.pri.ee/tbw/index.html.
2. O. Deusen et al., The elements of nature: interactive and realistic techniques, *ACM SIGGRAPH 2004 Course Notes*, 2004.
3. R. Parent and R. Parent, *Computer Animation: Algorithms and Techniques*, New York: Elsevier Science, 2001.
4. The 3DO Company. Available: http://en.wikipedia.org/wiki/The_3D0_Company.
5. Nintendo, Available: http://www.nintendo.com/.
6. Nokia, Available: http://www.nokia.com/.
7. Sony, Available: http://www.sony.com/.
8. K. Meyer, H. Applewhite, and F. Biocca, A survey of position trackers, *Presence: Teleoperators and Virtual Environ.*, **1** (2): 173–200, 1992.
9. Unreal Engine 3, Available: http://www.unrealtechnology.com/html/technology/ue30.shtml.
10. RenderWare, Available: http://www.csl.com/.
11. *Computer Graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS)*, ISO/IEC 9592–1:1989, New York: American National Standards Institute, 1989.
12. D. Hearn and M. Baker, *Computer Graphics with OpenGL*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2004.
13. OpenGL, Available: http://www.opengl.org/.
14. DirectX, Available: http://www.microsoft.com/directx/.
15. I. Millington, *Artificial Intelligence for Games*, San Francisco, CA: Morgan Kaufman, 2005.
16. D. Eberly, *Games Physics*, San Francisco, CA: Morgan Kaufman, 2003.
17. N. Burtnyk and M. Wein, Interactive skeleton techniques for enhancing motion dynamics in key frame animation, *Commun. ACM*, **19** (10): 564–569, 1976.
18. T. Wang and C. Chen, A Combined optimization method for solving the inverse kinematics problems of mechanical manipulators, *IEEE Trans. on Robotics and Automation*, **7** (4): 489–499, 1991.
19. M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Reading, MA: Addison-Wesley Professional, 2005.
20. Game Controller. Available: http://en.wikipedia.org/wiki/Game controller.
21. EyeToy, Available: http://www.eyetoy.com/.
22. Wii, Available: http://wii.nintendo.com/.
23. Quake, Available: http://www.idsoftware.com/.
24. Diablo II, Starcraft, Available: http://www.blizzard.com/.
25. EverQuest, Available: http://everquest.station.sony.com/.
26. Ultima Online, Available: http://www.uo.com/.
27. Asheron's Call, Available: http://www.microsoft.com/games/zone/asheronscall/.
28. M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, Local-lag and timewarp: Providing consistency for replicated continuous applications, *IEEE Trans. on Multimed.*, **6** (1): 47–57, 2004.
29. Final Fantasy XI, Available: http://www.playonline.com/ff1 1/home/.
30. DIS Steering Committee, IEEE Standard for Distributed Interactive Simulation - Application Protocols, *IEEE Standard 1278*, 1998.
31. L. Li, F. Li, and R. Lau, A Trajectory preserving synchronization method for collaborative visualization, *IEEE Trans. Visualizat. Comp. Graph.***12** (5): 989–996, 2006.
32. F. Li, R. Lau, and D. Kilis, GameOD: An Internet Based Game-On-Demand Framework, *Proc. of ACM VRST*, 2004, pp. 129–136.
33. J. Eichorn, *Understanding AJAX*, Upper Saddle-River, NJ: Prentice Hall, 2006.
34. I. Foster and C. Kesselman, *The Grid 2*, San Francisco, CA: Morgan Kaufman, 2003.

Frederick W. B. Li
University of Durham,
Durham, United Kingdom

CROWD SIMULATION

Crowds are part of our everyday experience; nevertheless, in virtual worlds, they are still relatively rare. Despite achieving impressive levels of visual quality with nearly photorealistic look, virtual environments in many cases resemble "ghost towns," with a small number of virtual inhabitants in most cases only directly related to their scenarios. Computer graphics-generated crowds are rare in interactive virtual environments such as computer games or virtual reality educational and training systems.

The first generation of real-time virtual crowd simulations had to sacrifice both visual and behavioral details in order to increase the number of handled characters. Researchers, now, aim to go beyond earlier results to achieve high-fidelity believable groups and crowds integrated into complex virtual worlds, giving the user a possibility to interact with virtual characters. To achieve this goal, several topics have to be explored, including rendering, behavior computation, procedurally generated locomotion animation, interaction with objects, and scene management. New approaches taking into account requirements specific for crowd simulations, such as ability to produce variety and scalability, are needed. It is essential to investigate heterogeneous simulations where a smaller number of complex agents would coexist and interact with a less detailed larger crowd within the same virtual world. To optimize the amount of the computation needed for real-time crowd applications, levels-of-details techniques have to be explored taking into account the human perception in conjunction with an efficient scene management.

RELATED WORKS

Although collective behavior has been studied since as early as the end of the nineteenth century (1), attempts to simulate it by computer models are quite recent, with most of the work done only in the mid-and late-1990s. Recently, researchers from a broad range of fields such as architecture (2–4), computer graphics (5–8), physics (9), robotics (10), safety science (11, 12), training systems (13–15), and sociology (16) have been creating simulations involving collections of individuals.

We can distinguish two broader areas of crowd simulations. The first one is focusing on realism of behavioral aspects with usually simple two-dimensional (2D) visualizations like evacuation simulators, sociological crowd models, or crowd dynamics models. In this area, a simulated behavior is usually from a very narrow, controlled range (for example, people just flying to exit or people forming ring crowd structures) with efforts to quantitatively validate correspondence of results to real-world observations of particular situations (11). Ideally, simulation results would then be consistent with datasets collected from field observations or video footage of real crowds either by human observers (17) or by some automated image processing method (18). Visualization is used to help to understand simulation results, but it is not crucial. In most cases, a schematic representation, with crowd members represented by colored dots, or sticky figures, is enough, sometimes even preferable as it allows for highlighting important information.

In the second area, a main goal is high-quality visualization (for example, in movie productions and computer games), but usually the realism of the behavior model is not the priority. What is important is a convincing visual result, which is achieved partly by behavior models and partly by human intervention in the production process. A virtual crowd should both look well and be animated in a believable manner, the emphasis of the research being mostly on rendering and animation methods. Crowd members are visualized as fully animated three-dimensional figures that are textured and lit to fit into the environment (19). Here, behavior models do not necessarily aim to match quantitatively the real world; their purpose is more in alleviating the work of human animators, and to be able to respond to inputs in case of interactive applications.

SCALABILITY OF SIMULATION

Achieving high-fidelity crowds is a challenging task. All involved components such as behaviors, rendering, or animation should be able to run with a high level of detail. However, as there are always performance limits, only a smaller subset of the virtual population can be higher quality. Computational resources have to be distributed between a simulation of different parts of the population and a simulation of the environment.

Apart from high-level behavior, there are low-level consistency issues—for example, what should happen to parts of the scene that are not visible to the user? A common approach is to just ignore these parts of the scene; however, this causes inconsistent situations when the user's attention returns. As no simulation was done for this part, the system has to generate a new situation from scratch, which usually does not correspond to what the user remembers from his previous visit (e.g, a table was moved; however, when the user returns, the table is back in the original position and not where it was left before).

Of course, it is not desirable and often unfeasible to run a full simulation of the whole environment at all times. However, some minimal simulation has to be done to be able to maintain the consistency of the virtual world. This leads to the notion of levels of detail for simulation:

1) **Full simulation** for the parts of the scene, which are observed and being interacted with by the user(s).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

2) **Simplified simulation** for the parts that are farther away and not well visible (some animations can be simplified, locomotion is less accurate, collision detection is less accurate, etc.).
3) **Minimal simulation** for the parts of the world that are not visible and not being interacted with. The entities should still keep their internal state, and movements have to be computed; however, there is no need to properly animate them.

The following sections will detail how rendering, animation, navigation, and behavior should be improved to allow for real-time, better, more realistic, and less costly crowd simulations.

HIGH-FIDELITY VISUALIZATION

We want to be able to have a full close-up zoom of a human with the highest possible level of detail in geometry, skin details, shadows, and facial animations, while still keeping a large crowd of humans.

All current crowd rendering methods use precomputation to speed up the rendering (8,20,21). The problem is how to balance the time used for each human to render it and how much will be stored in memory in the form of precomputed animations and behaviors. If all humans are computed and displayed dynamically on-the-fly, we can reach around 1000 humans with real-time performance (25 frames per second). On the other hand, if all animations are precomputed, we can render 10,000 humans with the same performance. With the precomputed approach, fewer animations can be stored because memory starts to become a concern when the number of possible animations grows. Dynamically created animation allows much wider variety of motions, especially for interactive applications where motions can depend on actions of the user. The animations can be then generated, for example, by using inverse kinematics, rag doll physics, or motion capture.

Other issues that need to be addressed are shadows cast by the humans, such as very detailed shadows when we have a close view and less detailed ones or no shadows when the user sees the crowd at a distance. High detail shadows, for example, are those in the face and those cast on the ground and onto other objects in the environment.

Rendering

Tecchia et al. (8) used billboards or impostors for crowd rendering. The main advantage is that impostors are very lightweight to render once they are in the memory of the graphics card. The method requires building of all animations from all possible camera angles and storing these pictures in a texture. One such texture can hold one frame of animation in very low-resolution billboards, where every individual subframe is about 16 pixels tall and wide. This process can give good visuals because it is basically an image-based rendering approach, so even pictures of real humans could be incorporated. However, zooming on these billboards will produce aliasing artifacts, because the images on the billboards have to be small to fit in the graphics card's texture memory. As a result, billboarded humans serve as a good approach for far-away humans that do not need detailed views. More sophisticated impostors have been also proposed by Aubel et al. (22).

Another approach that unifies image-based and polygonal rendering is found in Ref. 20. They create view-dependent octree representations of every keyframe of animation, where nodes store information about whether it is a polygon or a point. These representations are also able to interpolate linearly from one tree to another so that in-between frames can be calculated. When the viewer is at a long distance, the human is rendered using point rendering; when zoomed in, using polygonal techniques; and when in between, a mixture of the two. It does take large amounts of data per keyframe and needs long preprocessing times because of its precise nature, but also it gives near-perfect interpolation between detail levels without "popping" artifacts that otherwise occur if one uses discrete detail levels.

The third alternative is to use vertex shaders to deform a conventional mesh using a skeleton. The disadvantage is that the pipeline would be constrained to shaders, and every interaction such as lighting, shadows, and other standard effects would then have to be programmed with shaders.

In the last few years, rendering of virtual humans has been drastically improved. This improvement is due, for example, to variety in clothes and skin, which has been achieved through vertex and fragment shaders; many different levels of detail have been created and used; and computationally efficient fake soft shadows have been developed for the highest levels of fidelity.

Variety

Variety in rendering is defined as having different forms or types and is necessary to create believable and reliable crowds in opposition to uniform crowds. For a human crowd, variation can come from the following aspects: gender, age, morphology, head, kind of clothes, color of clothes, and behaviors. We create as many color variations as possible from a single texture.

One approach (23,24) to variety in rendering is an extension to Ref. 8 to create color variety from a single texture to dynamically animated three-dimensional (3-D) virtual humans. We obtain a wide variety of colors and appearances by combining our texture and color variety. Each mesh has a set of interchangeable textures, and the alpha-channel of each texture is segmented in several zones: one for each body part. This segmentation is done using a desktop publishing software. Using this alpha map to vary body part colors could be done with a fixed function pipeline approach, i.e., all the computation on the CPU. However, to improve the execution speed, it is possible to explore the performance of high-end consumer graphics cards, allowing for shader programming. This way, each character can have up to 256 different alpha key areas, corresponding to a certain body part. Using the fixed function pipeline approach, this is completely unreasonable, as it would require 256 passes.

In the process of designing human color variety, we have to deal with localized constraints: Some body parts need very specific colors. One cannot simply randomly choose any color for any body part. This would result in green skin, yellow lips, and other frightening results. With a graphics user interface, the designer can set intuitive color ranges in which a randomly chosen value will not produce unrealistic results. For instance, the color variety for the skin of a virtual human is defined in a range of unsaturated shades with red and yellow dominance, almost deprived of blue and green.

Levels of Detail

For the rendering part of crowds, levels of details are essential to lower the computational cost and allow for highly detailed virtual humans at the forefront of the scene. For example, we can consider three main levels of detail as introduced in Ref. 23:

- The highest level of detail of a virtual human is represented by a deformable mesh that can be deformed in real time to perform a skeleton-based animation. This deformable mesh can be divided into several different levels of fidelity; for example, we may consider the highest is composed of about 5000 vertices, and the next two have a smaller number of vertices, i.e., about 1000 and 200 vertices, respectively.
- The second level of detail of a virtual human could be a rigid mesh: a mesh whose deformation for a specific animation has been precomputed. The precomputed deformations allow substantial gains in speed; however, they require memory space to store the results and thus constrain the animation variety.
- The final rendering level of detail is the billboard model, which represents a virtual human as a textured rectangle that always faces the camera. To create these models, images are sampled around the waist level of the character at 20 different angles, for each keyframe composing an animation. Rendering such billboards is very fast; unfortunately, the memory requirements for each animation to sample are also very high.

Figure 1 shows an example with the three levels. Through these different levels of detail, we pass from a computationally expensive process to a very fast but memory requiring approach. One could ask if it is then worthwhile to use all these levels of details. The answer is yes, because the costly approaches in terms of memory are used for virtual humans that are far away from the viewer. For these models, only a few animations need to be sampled, because a varying speed or walk style is unnoticeable from afar. In terms of performances, with the three-level approach, it is possible to have a real-time crowd with about 40,000 individuals.

ANIMATION

While designing an animation engine usable in crowd simulations, several criteria have to be taken into account:

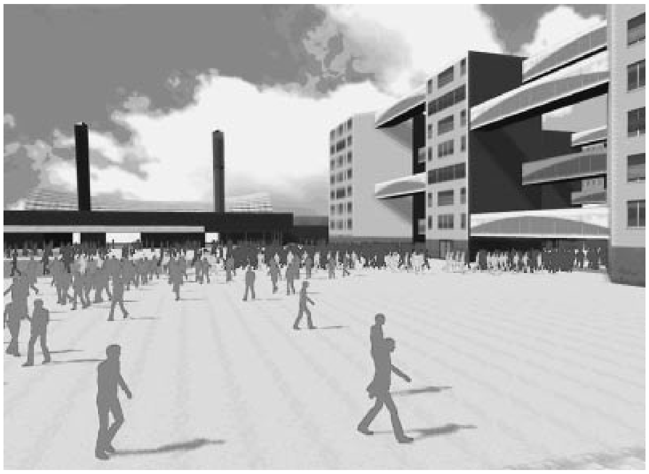


Figure 1. The three different rendering levels of detail: deformed meshes, rigid meshes, and billboards.

Animation computation should be efficient and scalable, it should also allow for variability, and it should be compatible with level-of-detail. To create flexible virtual humans with individualities, there are mainly two approaches:

- Motion capture and retargeting.
- Creation of computational models.

Motion Capture and Retargeting

The first approach consists in recording the motion using motion capture systems (magnetic or optical), and then trying to alternate such a motion to create this individuality. This process is tedious, and there is no universal method at this stage. Even if it is fairly easy to correct one posture by modifying its angular parameters (with an inverse kinematics engine, for instance), it becomes a difficult task to perform this over the whole motion sequence while ensuring that some spatial constraints are respected over a certain time range, and that no discontinuities arise. When one tries to adapt a captured motion to a different character, the constraints are usually violated, leading to problems such as the feet going into the ground or a hand unable to reach an object that the character should grab. The problem of adaptation and adjustment is usually referred to as the Motion Retargeting Problem.

Witkin and Popovic (25) proposed a technique for editing motions, by modifying the motion curves through warping functions, and produced some of the first interesting results. In a more recent paper (26), they extended their method to handle physical elements, such as mass and gravity, and described how to use characters with different numbers of degrees of freedom. Their algorithm is based on the reduction of the character to an abstract character, which is much simpler and only contains the degrees of freedom that are useful for a particular animation. The edition and modification are then computed on this simplified character and mapped again onto the end-user skeleton. Bruderlin and Williams (27) have described some basic facilities to change the animation, by modifying the motion parameter curves. They also introduced the notion of motion displacement map, which is an offset added to each motion curve. The Motion Retargeting Problem term was brought up by Michael Gleicher (28). He designed a space-time constraints solver, into which every constraint is added, leading to a big optimization problem. Bindiganavale and Badler (29) also addressed the motion retargeting problem, introducing new elements: using the zero-crossing of the second derivative to detect significant changes in the motion, visual attention tracking, and applying inverse kinematics to enforce constraints, by defining six subchains (the two arms and legs, the spine, and the neck). Finally, Lee and Shin (30) used in their system a coarse-to-fine hierarchy of B splines to interpolate the solutions computed by their inverse kinematics solver. They also reduced the complexity of the IK problem by analytically handling the degrees of freedom for the four human limbs.

Creation of Computational Models

The second approach consists in creating computational models that are controlled by a few parameters. One major problem is to find such models and to compose them to create complex motion.

The most important animation scheme used in crowd simulation, as well as games (31), is the locomotion, which is basically composed of walking and running motions.

The first idea is to generate different locomotion cycles online for each individual. Unfortunately, even though the engine could be very fast, we could not allow for spending precious computation time for such a task. Moreover, depending on the rendering levels of detail, e.g., billboards or rigid meshes, it is impossible to create an animation online.

A second approach is to use an animation database. At the beginning of a simulation, a virtual human walks at a certain speed. To find the corresponding animation, the framework formulates a request to the database. The database returns an animation at the closest speed. To create this database, we generate offline many different locomotion cycles, with varying parameters. A few representative locomotion cycles are selected to create the corresponding rigid mesh animations. An even smaller number of cycles are also selected to create billboard animations.

More recently, Glardon et al. (32) have proposed an approach to generate new generic human walking patterns using motion-captured data, leading to a real-time engine intended for virtual human animation. The method applies the principal component analysis (PCA) technique on motion data acquired by an optical system to yield a reduced dimension space where not only interpolation but also extrapolation are possible, controlled by quantitative speed parameter values. Moreover, with proper normalization and time warping methods, the generic presented engine can produce walking motions with continuously varying human height and speed with real-time reactivity.

This engine allows generating locomotion cycles, parameterized by a few user-defined values:

- Speed: the speed at which the human moves.
- Style: a value between 0 and 1, 0 being a walk motion, 1 being a run motion.
- Personification: a weight to blend among the five different locomotion styles of five different motions captured people.

Now when such a engine is fully integrated into a crowd framework, it is possible to generate many different locomotion cycles by simply varying the above parameters, thus, making each individual unique. This engine has been extended to curved walking (33) and dynamic obstacle avoidance (34).

NAVIGATION

The main goal is to simulate a large number of people navigating in a believable manner for entertainment applications. For this purpose, environments containing static obstacles are analyzed and preprocessed to identify collision-free areas (35,36). Collisions with dynamic obstacles, such as other pedestrians, are then avoided using local reactive methods based on a cell decomposition (37,38) or repulsive forces (9). Other approaches solve both navigation and pedestrian inter-collision problems simultaneously, using prioritized motion planning techniques (39, 40). In such approaches, navigation is planned successively for each entity, which then becomes a moving obstacle for the following ones—the growing complexity limits the number of people composing the crowd. The probabilistic roadmap (PRM) approach (41) can be adapted to plan individual locomotion (42,43). Recently, approaches tend to decompose environments in walkable corridors, whose width allows group navigation (44). None of these approaches automatically handle both uneven and multilayered environments, which are commonly encountered in virtual worlds. PRM-based approaches suit high-dimensional problems well. However, they are not the most efficient ones for navigation planning when the problem is reduced to three dimensions. Pettré et al. (23) propose a novel approach capable of decomposing such environments into sets of interconnected cylindrical navigable areas. Terrain analysis is inspired by Voronoï diagrams and by methods using graphics hardware to compute them (45). The resulting structure captures the environment topology and can be used for solving crowd navigation queries efficiently. To exploit the locomotion animations of the virtual humans, a navigation graph and path planner have been developed. The navigation graph supports both path planning and simulation by capturing the environment topology and by identifying navigable areas. These navigable areas (graph vertices) are modeled as cylindrical spaces with a variable radius. If two of these cylinders overlap, it is possible to pass from one to the other. A vertical gate (graph edge) is used to model each cylinder intersection as a connection. Figure 2 shows examples.

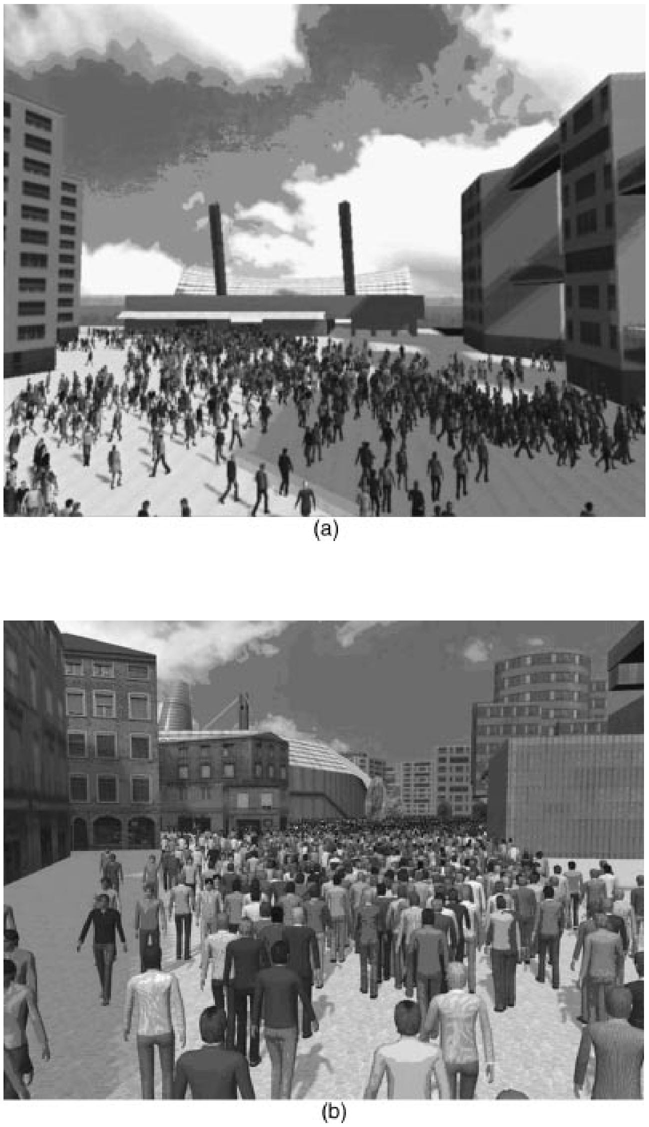


Figure 2. Two examples of large crowds.

HIGH-FIDELITY BEHAVIORS

To increase the number of simulated entities, the crowd simulation should be scalable. This function can be achieved, for example, by using behavior level-of-detail, where there are different computational demands for agents, depending on their relative position to the observer. The behavior model should then allow working with different fidelities, for example, by using iterative algorithms, or also heterogeneous crowds could be employed.

Behavior of animated characters has to obey some order. Humans have intentions and beliefs about their environment, and they almost never behave in a truly random manner. Most people plan their actions in advance with regard to the goal they are pursuing.

There are several possible approaches on how to simulate such behavior:

- **Scripting:** Scripting allows a very detailed level of control, but it is very inflexible. The virtual characters will not be able to react to the changes in their environment unless the scripts are extremely complex.
- **Reactive agents:** Virtual characters are not scripted, but they react to the changing environment according to the sets of rules (e.g., described by Badler in Ref. 46). There are few other popular techniques, e.g., BDI logic introduced by Bratman (47), cognitive modeling by Funge (48), or just simple finite state machines.
- **Planning:** Usually centralized, it tends to be inflexible in handling unexpected situations. To mitigate this to some extent, hierarchical planning is often used (49).

With the use of these three approaches, several difficulties arise: Scripting is problematic in the case of contingencies (e.g., unexpected obstacles in the path, objects being in unexpected places, etc.) and centralized planning tends to be complex, because it has to produce detailed plans for every character in the system and does not cope well with unexpected events. Reactive techniques perform usually well for single characters, but the lack of a global system state awareness hinders meaningful coordination if desired (e.g., a police squad).

These problems (contingencies, flexibility, and good level of control) can be addressed by a multilayered design, where:

- The low-level animation problems are handled by using state-of-the-art technology.
- The virtual characters have "brains," which handle the decision-making process by using classic planners. Planning on this level would allow for building goal-driven behaviors sensitive to the current situation.
- The virtual characters can communicate with each other by means of high-level communication primitives. The communication allows the virtual character to ask for help or for information. It also enables high-level communication with the user.

For example, in the ViCrowd system (50), crowds were modeled with various degrees of autonomy using a hierarchy of groups and individuals. Depending on the complexity of the simulation, a range of behaviors, from simple to complex rule-driven, were used to control the crowd motion with different degrees of autonomy (see Fig. 3). The crowd behavior is controlled in three different ways:

1) Using innate and scripted behaviors.
2) Defining behavioral rules, using events and reactions.
3) Providing an external control to guide crowd behaviors in real time.

To achieve the groups' and individuals' low-level behavior, three categories of information are used: knowledge that is used to represent the virtual environment's information; beliefs that are used to describe the internal

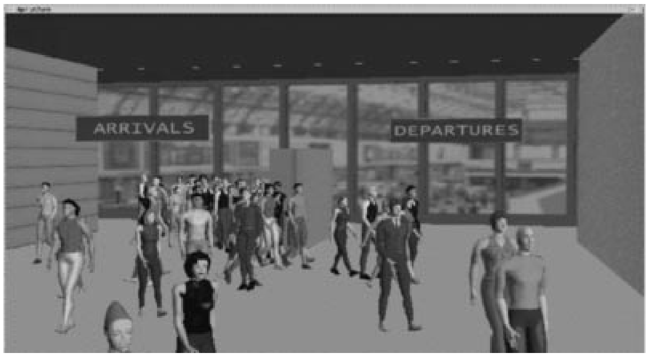


Figure 3. Crowd generated using the ViCrowd system.

status of groups and individuals; and intentions that represent the goal of a crowd or group.

Intelligence, memory, intention, and perception are focalized in the group structure. Also, each group can obtain one leader. This leader can be chosen randomly by the crowd system, defined by the user, or can emerge from the sociological rules. The crowd aims at providing autonomous, guided, and programmed crowds. Varying degrees of autonomy can be applied depending on the complexity of the problem. Externally controlled groups or guided groups no longer obey their scripted behavior, but they act according to the external specification. At a lower level, the individuals have a repertoire of basic behaviors that we call innate behaviors. An innate behavior is defined as an "inborn" way to behave. Examples of individual innate behaviors are goal-seeking behavior, the ability to follow scripted or guided events/reactions, and the way trajectories are processed and collision avoided. Although the innate behaviors are included in the model, the specification of scripted behaviors is done by means of a script language.

Perception

One key aspect in behavioral modeling is the perception of the virtual world by the virtual characters. For example, a decision should be based on the evaluation of the visual aspect of the scene as perceived by the virtual character. In a more general context, it is tempting to simulate perception by directly retrieving the location of each perceived object straight from the environment. This is of course the fastest solution (and has been extensively used in video games until the mid-1990s), but no one can ever pretend that it is realistic at all (although it can be useful, as we will see later on). Consequently, various ways of simulating visual perception have been proposed, depending on whether geometric or semantic information (or both) are considered. Conde and Thalmann (51) tried to integrate all multisensorial information from the virtual sensors of the virtual character.

Renault et al. (52) introduced first the concept of rendering-based vision, and then it was extended by several authors (53–57). In Ref. 52, it is achieved by rendering off-screen the scene as viewed by the virtual character. During the process, each individual object in the scene is assigned a different color, so that once the 2-D image has been computed, objects can still be identified: It is then easy to know which object is in sight by maintaining a table of correspondences between colors and objects' IDs. Furthermore, highly detailed depth information is retrieved from the view z-buffer, giving a precise location for each object. Rendering-based vision is the most elegant method, because it is the more realistic simulation of vision and addresses correctly vision issues such as occlusion, for instance. However, rendering the whole scene for each agent is very costly, and for real-time applications, one tends to favor geometric vision.

Bordeux et al. (58) have proposed a geometric vision consisting in a perception pipeline architecture into which filters can be combined to extract the required information. The perception filter represents the basic entity of the perception mechanism. Such a filter receives a perceptible entity from the scene as input, extracts specific information about it, and finally decides whether to let it pass through. However, the major problem with geometric vision is to find the proper formulas when intersecting volumes (for instance, intersecting the view frustum of the agent with a volume in the scene). One can use bounding boxes to reduce the computation time, but it will always be less accurate than rendering-based vision. Nevertheless, it can be sufficient for many applications, and as opposed to rendering-based vision, the computation time can be adjusted precisely by refining the bounding volumes of objects.

The most primitive approach is the database access. Data access makes maximum use of the scene data available in the application, which can be distributed in several modules. For instance, the object's position, dimensions, and shape are maintained by the rendering engine, whereas semantic data about objects can be maintained by a completely separate part of the application. Due to scalability constraints as well as plausibility considerations, the agents generally restrain their perception to a local area around them instead of the whole scene. This method is generally chosen when the number of agents is high, like in Reynolds' (7) flocks of birds or in Musse and Thalmanns (50) crowd simulation; human agents directly know the position of their neighbors and compute coherent collision avoidance trajectory.

AUTHORING

When increasing the number of involved individuals, it is becoming more difficult to create unique and varied content of scenarios with large numbers of entities. If we want to create or modify features of every individual one by one, it will soon become too laborious. If, on the other hand, we apply a set of features (either uniform or patterned) to many individuals at once, it could create unwanted artifacts on a larger scale, resulting in an "army-like" appearance with too uniform or periodic distributions of individuals or characteristics. Use of random distributions can alleviate such problems; however, it can be very difficult to capture the desired constraints into a set of mathematical equations,

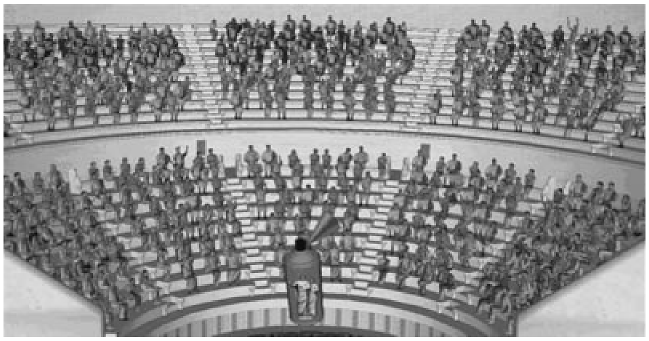


Figure 4. Crowdbrush application: The spray can is used to modify the class of Roman individuals.

especially considering integration into common art production pipelines.

Bottom-up approaches, such as local rule-based flocking (7), can create such complexity; however, they are difficult to control if we want to achieve particular end configurations (how to set local rules to get a global result). In the recent work, Anderson et al. (59) achieved interesting results for a particular case of constrained flocking animation. Nevertheless, the algorithm can get very costly when increasing the number of entities and simulation time.

Ulicny et al. (60) propose an approach that gives full creative power to designers using metaphors of artistic tools, operating on a 2-D canvas familiar from image manipulation programs working in WYSIWYG (What You See Is What You Get) mode, with a real-time view of the authored scene. The user controls the application using a mouse and a keyboard. These tools then affect the corresponding objects in a three-dimensional world space (see Fig. 4). Different tools have different visualizations and perform different effects on the scene, including creation and deletion of crowd members, changing of their appearances, triggering of various animations, setting of higher level behavioral parameters, setting waypoints for displacement of the crowd, or sending of events to a behavior subsystem. The mouse moves the visual representation of the brush tool (we used an icon of a spray can) on the screen, with the mouse buttons triggering different actions either on rendering or behavior subsystems. The keyboard selects different brushes and sets their parameters and switches between “navigate” and “paint” modes. In the “navigate” mode, the mouse controls position and orientation of the camera. In the “paint” mode, the camera control is suspended and the different areas on the screen are selected depending on the pressed mouse button. The selection areas can be, in principle, any arbitrary 2-D shape. This area is then further processed by the brush according to its particular configuration and a specific operator. For example, a stroke of the creation brush with the random operator would create a random mixture of entities or a stroke of the uniform color brush would set colors of affected individuals to the same value.

CONCLUSION

For many years, this was a challenge to produce realistic virtual crowds for special effects in movies. Now, there is a new challenge: the production of real-time truly autonomous virtual crowds. Real-time crowds are necessary for games, VR systems are necessary for training and simulation, and crowds are necessary in augmented reality applications. Autonomy is the only way to create believable crowds reacting to events in real time.

ACKNOWLEDGMENTS

The authors is grateful to people who have helped in the writing of this article, in particular, Pablo De Heras Ciechomski, Branislav Ulicny, Soraia Raupp Musse, Julien Pettré, Barbara Yersin, Jonathan Maim, and Pascal Glardon.

BIBLIOGRAPHY

1. G. LeBon, *Psychologie des Foules*, Paris, France: Alcan, 1895.
2. T. Schelhorn, D. O’Sullivan, M. Haklay, and M. Thurstain-Goodwin, STREETS: An agent-based pedestrian model, *Proc. Computers in Urban Planning and Urban Management*, Venice, Italy, 1999.
3. A. Penn and A. Turner, space syntax based agent simulation, In: M. Schreckenberg and S.D. Sharma (eds.), *Pedestrian and Evacuation Dynamics*, Berlin, Germany: Springer-Verlag, 2001.
4. A. Turner and A. Penn, encoding natural movement as an agent-based system: An investigation into human pedestrian behavior in the built environment, *Environ. Planning B: Planning Design*, **29**: 473–490, 2002.
5. E. Bouvier, P. Guilloteau, crowd simulation in immersive space management, *Proc. Eurographics Workshop on Virtual Environments and Scientific Visualization*, Berlin, Germany: Springer-Verlag, 1996, 104–110.
6. D. Brogan and J. Hodgins, Group behaviors for systems with significant dynamics, *Auto. Robots*, **4**: 137–153, 1997.
7. C.W. Reynolds, Flocks, herds, and schools: A distributed behavioral model, *Proc. SIGGRAPH*, 1987, pp. 25–34.
8. F. Tecchia, C. Loscos, and Y. Chrysanthou, Image-based crowd rendering, *IEEE Comp. Graphics Applicat.*, **22**(2): 36–43, 2002.
9. D. Helbing and P. Molnar, Social force model for pedestrian dynamics, *Phys. Rev. E*, **51**: 4282–4286, 1995.
10. P. Molnar and J. Starke, Control of distributed autonomous robotic systems using principles of pattern formation in nature and pedestrian behavior, *IEEE Trans. Syst. Man Cyb. B*, **31**: 433–436, 2001.
11. P. A. Thompson and E. W. Marchant, A Computer-model for the evacuation of large building population, *Fire Safety J.*, **24**: 131–148, 1995.
12. G. K. Still, Crowd Dynamics, PhD Thesis, Coventry, UK: Warwick University, 2000.
13. L. Bottaci, A direct manipulation interface for a user enhanceable crowd simulator, *J. Intell. Sys.*, **5**: 249–272, 1995.
14. D. Varner, D. R. Scott, J. Micheletti, and G. Aicella, UMSC small unit leader non-lethal trainer, *Proc. ITEC ’98*, 1998.

15. J. R. Williams, A Simulation Environment to Support Training for Large Scale Command and Control Tasks, PhD Thesis, Leeds, UK: University of Leeds, 1995.
16. C. W. Tucker, D. Schweingruber, and C. McPhail, Simulating arcs and rings in temporary gatherings, *Internat. J. Human-Computer Sys.*, **50**: 581–588, 1999.
17. D. Schweingruber and C. McPhail, A method for systematically observing and recording collective action, *Sociological Meth. Res.*, **27**(4): 451–498, 1999.
18. A. N. Marana, S. A. Velastin, L. F. Costa and R. A. LotufoAutomatic estimation of crowd density using texture, *Safety Sci.*, **28** (3): 165–175, 1998.
19. F. Tecchia, C. Loscos, and Y. Chrysanthou, visualizing crowds in real-time, *Comput. Graphics Forum*, **21**(4): 735–765, 2002.
20. M. Wand and W. Strasser, multi-resolution rendering of complex animated scenes, *Comput. Graphics Forum*, **21**(3): 483–491, 2002.
21. B. Ulicny, P. deHeras, and D. Thalmann, CrowdBrush: Interactive authoring of real-time crowd scenes, *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Grenoble, France, 2004.
22. A. Aubel, R. Boulic, and D. Thalmann, Real-time display of virtual humans: Level of details and impostors, *IEEE Trans. Circuits Syst. Video Technol.*, Special Issue on 3D Video Technology, 2000.
23. J. Pettre, P. deHears, J. Maim, B. Yersin, J. P. Laumaond, and D. Thalmann, Real-time navigating crowds: Scalable simulation and rendering, *Comp. Animation Virtual Worlds*, **16**, (3–4): 445–456, 2006.
24. B. Yersin, J. Maïm, P. de HerasCiechomski, S. Schertenleib, and D. Thalmann, Steering a virtual crowd based on a semantically augmented navigation graph, *VCROWDS*, 2005 pp. 169–178.
25. A. Witkin and Z. Popovic. Motion warping. *Proc. SIGGRAPH 95*, 1995, pp. 105–108.
26. Z. Popovic and A. Witkin, Physically based motion transformation. *Proc. SIGGRAPH 99*, 1999, pp. 11–20.
27. A. Bruderlin and L. Williams, Motion signal processing, *Proc. SIGGRAPH 95*, 1995 pp. 97–104
28. M. Gleicher, Retargeting motion to new characters, *Proc. SIGGRAPH 98*, 1998, pp. 33–42.
29. R. Bindiganavale, N. I. Badler, Motion abstraction and mapping with spatial constraints. In: N. Magnenat-Thalmann and D. Thalmann, (eds.), *Modeling and Motion Capture Techniques for Virtual Environments, Lecture Notes in Artificial Intelligence*, New York: Springer, 1998, pp. 70–82.
30. L. Jehee, S.Y. Shin, A hierarchical approach to interactive motion editing for human-like figures, *Proc. SIGGRAPH 99*, 1999, pp. 39–48.
31. R. Boulic, B. Ulicny, and D. Thalmann, Versatile Walk Engine, *J. Game Development*, **1** (1): 29–50, 2004.
32. P. Glardon, R. Boulic, and D. Thalmann, PCA-based walking engine using motion capture data, *Proc. Computer Graphics International*, 2004, pp. 292–298.
33. P. Glardon, R. Boulic, and D. Thalmann, Robust on-line adaptive footplant detection and enforcement for locomotion, *Visual Comput.*, **22** (3): 194–209, 2006.
34. P. Glardon, R. Boulic, and D. Thalmann, Dynamic obstacle clearing for real-time character animation, *Visual Comput.*, **22** (6): 399–414, 2006.
35. W. Shao and D. Terzopoulos, Autonomous pedestrians, SCA'05: *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005, pp. 19–28.
36. F. Lamarche and S. Donikian, Crowds of virtual humans: A new approach for real time navigation in complex and structured environments, *Comput. Graphics Forum*, **23** (3): 509–518, 2004.
37. P.G. Gipps and B. Marksjo, Micro-simulation model for pedestrian flows, *Math. and Comput. Simulation*, **27**: 95–105, 1985.
38. H. Klüpfel, T. Meyer-König, J. Wahle, and M. Schreckenberg, Microscopic simulation of evacuation processes on passenger ships. *Proc. Fourth Int. Conf. on Cellular Automata for Research and Industry*, 2000, 63–71.
39. M. Lau and J. Kuffner, Behavior planning for character animation, *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005, 271–280.
40. M. Sung, L. Kovar, and M. Gleicher, Fast and accurate goal-directed motion synthesis for crowds, *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005, Los Angeles, CA, pp. 291–300.
41. L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *Proc. IEEE Transactions on Robotics and Automation*, 1996, pp. 566–580.
42. M. G. Choi, J. Lee, and S.Y. Shin, Planning biped locomotion using motion capture data and probabilistic roadmaps, *Proc. SIGGRAPH'03: ACM Transactions on Graphics*, **22** (2): 182–203, 2003.
43. J. Pettré, J. P. Laumond, and T. Siméon, A 2-stages locomotion planner for digital actors. SCA'03: *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003, pp. 258–264.
44. A. Kamphuis and M.H. Overmars, Finding paths for coherent groups using clearance, SCA'04: *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004, pp. 10–28.
45. K. Hoff, J. Keyser, M. Lin, D. Manocha, and T. Culver, Fast computation of generalized voronoi diagrams using graphics hardware, *Proc. SIGGRAPH'99*, 1999, pp. 277–286.
46. N. Badler, LiveActor: A virtual training environment with reactive embodied agents, *Proc. Workshop on Intelligent Human Augmentation and Virtual Environments*, 2002.
47. M. E. Bratman, *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, 1987.
48. J. Funge, X. Tu, and D. Terzopoulos, Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters, *Proc. SIGGRAPH'99*, 1999.
49. J. Baxter and R. Hepplewhite, A hierarchical distributed planning framework for simulated battlefield entities, *Proc. PLANSIG'00*, 2000.
50. S. Raupp Musse, D. Thalmann, A behavioral model for real time simulation of virtual human crowds, *IEEE Trans. Visualization Comput. Graphics*, **l7** (2): 152–164, 2001.
51. T. Conde and D. Thalmann, An integrated perception for autonomous virtual agents: Active and predictive perception, *Comput. Animation and Virtual Worlds*, **16** (3–4): 457–468, 2006.
52. O. Renault, N. Magnenat-Thalmann, and D. Thalmann, A vision-based approach to behavioral animation, *J. Visualization Comp. Animation*, **1** (1): 18–21, 1990.
53. H. Noser, O. Renault, D. Thalmann, and N. Magnenat Thalmann, Navigation for digital actors based on synthetic vision, memory and learning, *Comput. Graphics*, **19** (1): 7–19, 1995.
54. X. Tu and D. Terzopoulos, Artificial fishes, physics, locomotion, perception, behavior, *Proc. SIGGRAPH '94*, 1994, pp. 43–50.

55. J. Kuffner, J. C. Latombe, Fast synthetic vision, memory, and learning models for virtual humans, *Proc. Computer Animation' 99*, 1999, pp. 118–127.

56. B. M. Blumberg, T. A. Galyean, Multi-level direction of autonomous creatures for real-time virtual environments, *Proc. SIGGRAPH 95*, 1995, pp. 47–54.

57. C. Peters and C. O'Sullivan, A memory model for autonomous virtual humans, *Proc. Third Irish Eurographics Workshop on Computer Graphics*, Dublin, Ireland, 2001, pp. 21–26.

58. C. Bordeux, R. Boulic, and D. Thalmann, An efficient and flexible perception pipeline for autonomous agents, *Proc. Eurographics '99*, 1999, pp. 23–30.

59. M. Anderson, E. McDaniel, and S. Chenney, Constrained animation of flocks, *Proc. ACM SIGGRAPH /Eurographics Symposium on Computer Animation*, 2003, pp. 286–297.

60. B. Ulicny, P. deHeras, and D. Thalmann, Crowdbrush: Interactive authoring of real-time crowd scenes, *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation '04*, 2004, pp. 243–252.

DANIEL THALMANN
Swiss Federal Institute
of Technology
Lausanne, Switzerland

HIGH-QUALITY TEXTURE MAPPING AND RENDERING OF POINT MODELS

POINT-BASED RENDERING

For over a decade since their introduction, points have been explored as alternatives to triangle meshes for geometry modeling and rendering (1). They have recently received increasing attention from the computer graphics community and gained a lot of popularity because of the wider availability of three-dimensional (3-D) acquisition devices. Based on their fundamental simplicity, points have been shown to be flexible and efficient in representing highly detailed features on complex objects. Directly working on point-based geometries greatly simplifies content creation, surface reconstruction, and rendering, as no connectivity or topological constraints exist.

Many point-based rendering algorithms focus on efficiency using hardware acceleration. Emphasis has also been placed on high-quality rendering. The surface splatting technique (2) and differential points (3) techniques have been proposed for high-quality surface rendering. Alexa et al. (4) control the fidelity of the representation by dynamically adjusting the density of the points. Zwicker et al. (5) introduce the elliptical weighted average (EWA) filter to increase the rendering quality for point-based rendering. Recently Botsch et al. (6) proposed Phong splatting to generate superior image quality, which bases the lighting of a splat on an associated linearly varying normal field. Schaufler and Jensen introduced ray tracing of point-based geometry (7). Their approach renders high-quality ray-traced images with global illumination using unstructured point-sampled data, thus avoiding the time-consuming process of reconstructing the underlying surface or any topological information. Intersections with the point-sampled geometry are detected by tracing a ray through the scene until the local density of points is above a predefined threshold. They then use all points within a fixed distance of the ray to interpolate the position, normal and any other attributes of the intersection. A deferred shading technique (8) has been proposed for high-quality point-based rendering, providing per-pixel Phong shading for dynamically changing geometries and high-quality anti-aliasing.

As in triangle-based rendering, texture mapping can be an effective technique to enrich visual reality in renderings of point models. Textures can be directly generated by a 3-D painting and editing system (9). In 3-D scanning, high-resolution photos are often acquired when lower resolution scans are conducted. Existing approaches couple geometry and texture information at each point; a point has both geometry coordinates and a color. In Surfels (2), Pfister et al. store prefiltered texture colors of the Surfel mipmap in the layered depth cube (LDC) tree and perform linear interpolation during rendering. As existing methods assign one color to each point, photos (textures) are downsampled to match the resolution of the point set. In this process, high-frequency texture information is permanently lost.

Ren et al. (10) propose the use of textured polygons when textures are mapped to each point splat and are blended through splatting. The proposed object space EWA splatting uses a two-pass rendering algorithm. In the first rendering pass, visibility splatting is performed by shifting opaque surfel polygons backward along the viewing rays, whereas in the second rendering pass, surface polygons are texture mapped, deformed, and rendered through view-dependent EWA prefiltering.

MOTIVATION OF HIGH-QUALITY TEXTURE MAPPING ON POINT-BASED MODELS

Texture mapping enhances the visual richness of rasterized images with a relatively small additional computation cost. It has been one of the most successful techniques developed in computer graphics for high-quality image synthesis since its birth in 1970s.

Textures were originally developed for mesh or polygonal models. In its basic form, a texture (an image) is mapped to a simple geometry shape, which allows arbitrarily complicated color details of the surface to be rendered without additional mesh resolutions. One typical example is mapping a brick texture to a simple rectangle to resemble the details of a wall.

Besides color, other values can be mapped such as specular reflection, normal vector perturbation (bump mapping), and surface displacement. For additional reading on texture mapping topic, readers can refer to Heckbert's survey paper and Wolberg's book in the reading list.

With the recent rapid advancement of laser range scanning techniques, acquisition of real objects (both geometry and appearance) is becoming common, leading to an increasing number of point-based models. When scanning small-to middle-size objects, it is practical to scan a large number of points to densely sample the object details. In this case, the point density of scans can match up with the pixel resolution of photographs that represent the details of the objects. When scans and photographs are registered together, color information is assigned to each point. However, for large outdoor environment scanning, because of the limitations of the scanning device and the constraints of the scanning process, the acquired point resolution is usually far below the typical resolution of a digital photograph. For example, the resolution of a photo can be more than ten times higher than that of a laser scan. In these cases, it is more desirable to perform authentic texture mapping on point models, i.e., assigning texture coordinates to each point so that texture colors in between points can be looked up, instead of directly assigning a color to each point.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

INTERPOLATION VS. COMPUTATION ORDER

In computer graphics, interpolation is an important operation across the entire rendering pipeline. Interpolation can be performed either before or after a function evaluation (or table lookup), corresponding to either preinterpolation or postinterpolation, respectively. In the following discussion, we review several important graphical operations or visualization procedures that have featured alternative methods depending on when the interpolation is conducted. Within this framework, our TSplatting performs interpolation on texture coordinates before color lookup, which is an alternative to directly interpolating colors, as is performed in conventional point splatting.

Preshaded vs. Postshaded Volume Rendering

Volume datasets consist of sample points defined at 3-D grids. In volume rendering, volumes are resampled and each sample's color is computed and contributed to the final image through compositing. Volume rendering algorithms can be classified as preshaded and postshaded according to where the volume resampling (interpolation) takes place. In preshaded algorithms (11), original volume samples at grids are first classified (through transfer function lookup) and shaded before they are resampled. Instead, postshaded volume rendering algorithms (12,13) first resample (interpolate) raw volume samples and then perform transfer function lookup and shading using the resampled values. Thus, postshaded methods conduct preinterpolation, whereas preshaded methods conduct postinterpolation. Because both transfer function lookup and shading are nonlinear functions, the resulting images from the two approaches are different. Although consensus still does not exist on which method is superior, in general, images generated from the preshaded methods result in blurry images and loss of fine geometry details, whereas the postshaded methods produce sharper features in generated images. On the other hand, preshaded methods can be more efficient when super-sampling of volumes is performed for generating higher resolution images because less effort is spent on transfer function lookup and shading.

Gouraud vs. Phong Shading

When rendering polygon meshes, Gouraud shading (14) and Phong shading (15) are two major shading algorithms for local illumination. Gouraud shading performs a lighting calculation at each vertex and then interpolates among vertices to obtain pixel colors within the projection of each polygon. Instead, Phong shading first interpolates normals across the polygon facet and then a lighting calculation is performed based on the interpolated normal to obtain pixel color. Here, Phong shading does preinterpolation, whereas Gouraud shading does postinterpolation. Because of the normal interpolation before the lighting evaluation, Phong shading produces specular highlights that are much less dependent on the underlying polygons than Gouraud shading does. On the other hand, Phong shading is more expensive than Gouraud shading.

Surfels vs. Phong Splatting for Point-Based Rendering

Similar to polygon-based rendering, point-based rendering can also be classified as preinterpolation versus postinterpolation methods. In Surfels (2), each point is treated as a flat disk with a single normal direction. Points are first flat-shaded before composition. Alternatively, Phong splatting (6) generates superior image quality by first interpolating normals among points using point splatting and then deferred shading by evaluating lighting at each pixel using the interpolated normals. In this case, Phong splatting is a preinterpolation method, whereas Surfels is a postinterpolation method. An in-between method is rendering using differential points (16), in which normals in the vicinity of a point are extrapolated based on the local differential geometry of the point and are then used to calculate the lighting of the point disk; these shaded point disks are finally blended together as done in Surfels.

Color Splatting versus Texture Splatting for Texture Mapping Points

The operation of mapping textures on points can be achieved in alternative approaches falling in the preinterpolation and postinterpolation framework. In conventional methods of texture mapping points, the color of each point is first looked up using its predefined texture coordinates and is then splatted and composited with adjacent point splats similarly computed. This method is referred to as color splatting by us. In our texture splatting (TSplatting), texture coordinates, instead of color, are splatted and composited followed by a deferred texture lookup of each screen pixel to obtain pixel colors. Based on the interpolation order, TSplatting belongs to preinterpolation while color splatting belongs to postinterpolation. We will see that TSplatting produces superior image quality than that of color splatting.

We summarize the afore-mentioned methods in Table 1. In general, postinterpolation methods are computationally less expensive but generate lower image quality than do preinterpolation methods. Next, we provide additional discussion on our TSplatting algorithm, which is a preinterpolation method.

TSPLATTING—A POINT-BASED HIGH-QUALITY TEXTURE MAPPING FRAMEWORK

Motivated by the above requirements, a novel framework of high-quality texture mapping of point-based models, *TSplatting*, is developed (17). The general two-pass point-based rendering pipeline is modified into a three-pass pipeline. The first two steps are very similar to a conventional point-based rendering pipeline except that the color of each point is replaced with its texture coordinates. The texture lookup operation is deferred to a third pass through a per-pixel texture lookup for all screen pixels. In addition, advanced techniques like bump mapping and texture-based lighting can be implemented. This method is capable of rendering point models with much higher texture resolution than that of the point model itself without resorting to meshing or triangulation. As a result, a

Table 1. Comparison of Algorithms with Order Switch of Computation and Interpolation in Different Operations

Operation	Variable	Function	Postinterpolation	Preinterpolation
Volume rendering	Volume density	Classification	Preshaded	Postshaded
Polygon shading	Normal	Lighting	Gauraud shading	Phong shading
Point shading	Normal	Lighting	Surfels	Differential points Phong splatting
Point-based texture mapping	Texture Coordinate	Texture Lookup	Color splatting	TSplatting

significant improvement in rendering quality occurs. This decoupling of geometry and texture facilitates perceptually guided point simplification by leveraging texture masking effects in achieving uncompromised final image quality (18). These texture mapping results can be potentially achieved in Botsch et al. 's deferred shading framework (8) by encoding texture coordinate information in the attribute pass and performing a texture lookup in the shading pass. Here we describe the explicit design and implementation of this framework. We then point out several important issues raised in this seemingly straightforward process and describe solutions to them.

FRAMEWORK OVERVIEW

The *TSplatting* framework maps high-resolution textures onto point models of lower spatial resolution. The pipeline of the framework is illustrated in Fig. 1.

As mentioned, point-based models are rendered in two passes. In the first pass, points are rendered as opaque disks to obtain a visibility mask. The points that pass the visibility test are then rendered using splatting techniques in the second pass (2,10). Adjacent splats are overlapped to ensure no holes are on the rendered surface. The color of each splat is blended with that of its neighbors based on the opacity values modulated with a low-pass Gaussian filter.

In *TSplatting* framework, a three-pass rendering is employed to texture map point-based models. Instead of splatting colors for each point, texture coordinates are used in the second pass and one additional pass for texture lookup is added. The texture coordinate interpolation in the second pass is the key for mapping high-resolution textures onto sparse points. The three rendering passes are illustrated in Fig. 1 and are explained below.

Pass 1: Visibility Computation

In the first pass, a visibility mask is generated by rendering point splats as opaque disks. To ensure hole-free surfaces, the projection size of each point is suitably calculated.

Pass 2: Texture Coordinate Splatting

In this pass, each point is splatted by replacing color with texture coordinates. Texture coordinates between points are blended and automatically interpolated by graphics hardware.

Pass 3: Per-pixel Texture Look-up

In this pass, texture coordinates encoded in each screen pixel are used to look up the high-resolution texture image. Lighting can also be incorporated by modulating the texel color with the lighting coefficients.

TEXTURE COORDINATES SPLATTING

One of the main contributions of the *TSplatting* framework is the use of deferred shading for splatting in the second rendering pass and the subsequent texture lookup. In the following discussion, we provide an analysis and justification of using texture coordinates splatting in achieving high texture mapping quality.

In point splatting (5,10), objects are represented by a set of points $\{P_k\}$ in object space. Each point is associated with a radially symmetric basis function r_k and color coefficient w_k. With texture mapping, each point also has a texture coordinate coefficient v_k. Elliptical Gaussians are normally chosen as basis function r_k, and the prefilter is h. The

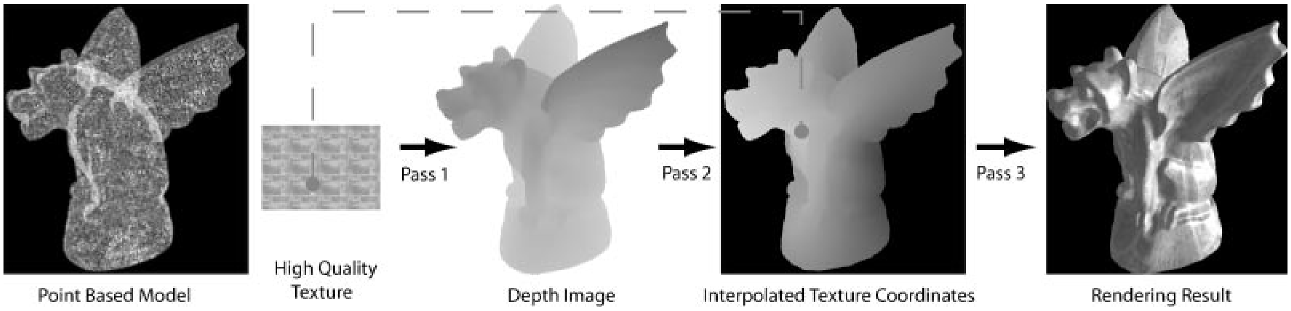


Figure 1. *TSplatting*. During rendering, texture coordinates between points are interpolated by graphics hardware. In the final step, a per-pixel texture look-up is performed to fetch colors from high-resolution textures.

splatting output $g_c(\mathbf{x})$ at screen position $\mathbf{x}$ can be expressed as a weighted sum of resampling filters $\rho_k(\mathbf{x})$:

$$g_c(\mathbf{x}) = \sum_{k \epsilon N} w_k \rho_k(\mathbf{x}) \tag{1}$$

$\rho k(\mathbf{x})$ is a convolution of the basis function r_k and prefilter h. The detailed definition of $\rho_k(\mathbf{x})$ can be found in Ref. 10. Color w_k is precomputed by texture prefiltering (5). According to the Nyquist theorem, the prefiltering results in a loss of texture details with frequency higher than the point model sampling rate.

Although the texture coordinate coefficient v_k normally varies gradually over the points, their corresponding textures may contain high-frequency details. In *TSplatting*, a reconstruction of texture coordinates $g_t(\mathbf{x})$ is first computed in the screen space (pass 2):

$$g_t(\mathbf{x}) = \sum_{k \epsilon N} v_k \rho_k(\mathbf{x}) \tag{2}$$

As the local parameterization is normally smooth, the loss of $g_t(\mathbf{x})$ during the splatting reconstruction is minimal.

In the subsequent stage (pass 3), each screen pixel obtains its color information through a resampling of the texture, in contrast with color splatting where each pixel color is obtained by a direct interpolation among colors of an adjacent point.

In the second rendering pass, texture coordinates are rendered as color attributes for each point. As the point size increases, texture splats overlap and smooth interpolation between splat centers is performed by assigning a Gaussian kernel (representing opacity) to each splat and the adjacent splats are composited together. Smoothly interpolated texture coordinates ensure a faithful mapping of the high-resolution texture. Note that, along boundaries, textures are deformed because of a lack of neighboring points for blending. A solution to this issue will be discussed later in this article.

Various experiments show that the *TSplatting* method (TS) provides much higher quality texture mapping than the normal color splatting methods (CS). In Fig. 2, a texture is mapped to point-based vase models with various resolutions. Figure 2(a) and (b) are vase models, with 10K and 35K points, respectively, rendered by the TS method. Figure 2(c)–(e) are rendered by traditional CS, with model sizes ranging from 10K to 560K. The rendering quality of the point model with only 35K points using the TS method [Fig. 2 (a)] is comparable with that of the 560K point model rendered with CS [Fig. 2 (e)]. Images in the second row are the close-ups (128×128) of the corresponding first row images. The last row shows the individual points of each model. Only when the point resolution is close to the image pixel resolution does the rendering quality of CS start to match up with the quality of the TS.

Figure 3 demonstrates the TS results by applying various high-resolution textures to a torus model. This torus model has 22,321 points. The texture resolutions are around 1024×1024. Per-pixel lighting is applied in the rendering. Figure 4 shows applying bump mapping with TS. The bump map shown in the upright corner is created using the NVIDIA Normal Map Filter.

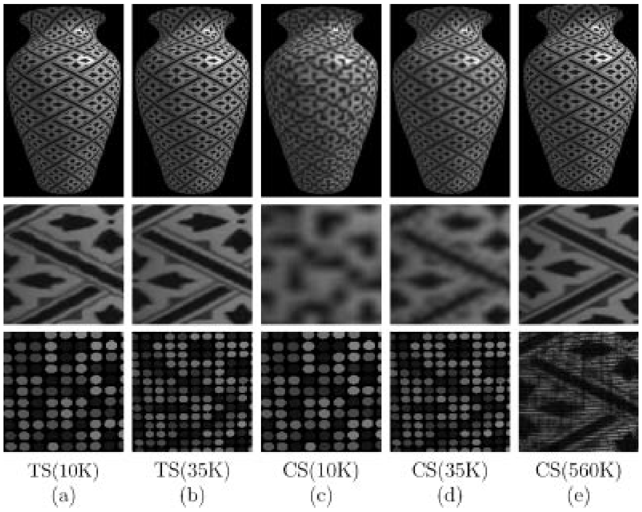


Figure 2. Comparison of *TSplatting* with traditional color splatting on points. Images (a) and (b) depict models rendered by the TS method and have 10K and 35K points, respectively. Images (c)–(e) are rendered by traditional CS. The image quality resulting from color splatting increases when the number of points increases (from 10K to 560K). Images in the second row are the corresponding close-ups of the rendered images in the first rows. The last row shows individual points in each model.

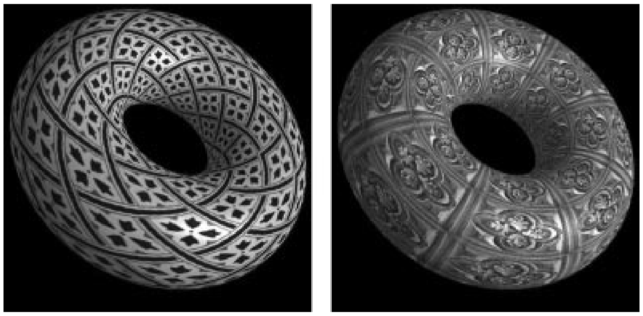


Figure 3. Various textures are applied to a torus model with the *TSplatting* method.

ADVANCED RENDERING ISSUES

We discuss solutions to several issues to achieve high-quality texture mapping on points at different viewing resolutions and texture/object boundaries.

Anti-aliasing—Mipmapping

Similar to mesh rendering, aliasing occurs when texels are minimized (i.e., one texel projects to less than one pixel size on screen) during texture mapping. The rendering must be appropriately anti-aliased to achieve high-quality results.

In the TS framework, conventional anti-aliasing methods can be naturally integrated. The mipmapping function (19) supported by OpenGL is used to achieve anti-alaising. To this end, an image pyramid is created for each input texture using the OpenGL command **gluBuild2DMipmaps** without explicit computation in the shader. In the last rendering pass, for each fragment, the appropriate level

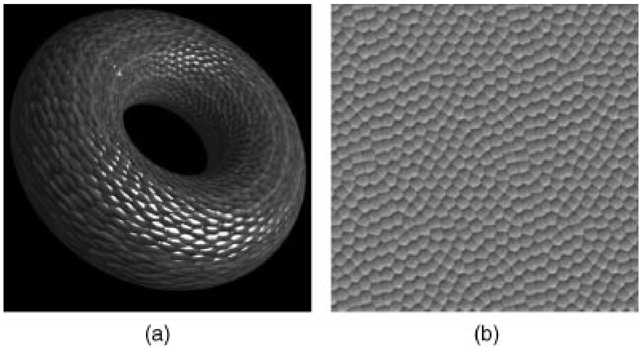


Figure 4. Bump mapping by *TSplatting*. (a) Rendered image. (b) Bump map texture.

of textures will be computed automatically by hardware and used for look-up. Effectively, no extra coding is needed for the anti-alisasing except changing the OpenGL status and building the texture pyramid.

Boundary Issues

When directly applying texture coordinate interpolation to a point model at the object boundaries, artifacts may appear in two situations: along the model boundaries because of lack of texture blending and at the locations where the texture coordinates change discontinuously [Fig. 5(b), marked by a yellow rounded rectangle]. Texture coordinates may be inappropriately interpolated along the discontinuity (boundary) of texture coordinates. Direct interpolation between the splats on the opposite sides of the texture coordinate discontinuity will result in a blur of the whole spectrum of the texture coordinates along the boundary.

To remove such rendering artifacts, it is necessary to limit the footprint of the boundary splats. We adapt the techniques to render sharp features by explicitly representing boundaries or discontinuity features as clip lines (20).

The mathematical setup of a clipping scheme is illustrated in Fig. 5(a). For each point, at most two clipping lines in each point's local texture coordinate are specified in the form of

$$A_i x + B_i y + C_i = 0 \tag{3}$$

A_i, B_i, and C_i (i = 1,2) are line parameters. The local texture coordinate is defined by the two-texture parameter vector du and dv. The point's center is the origin. Each line divides a plane containing the point splat into two parts. The region where

$$A_i x + B_i y + C_i < 0 \tag{4}$$

is defined to be clipped. Line parameters are passed to hardware together with other attributes of each point. During rendering, each fragment is checked with its value according to Equation (4). Figure 5(a) shows an example with two types of boundaries. The first is the point model boundary. The second is the texture coordinate discontinuity. Without clipping [Fig. 5 (b)], texture coordinates are blended across the boundary and artifacts are introduced. With clipping [Fig. 5 (c)], points along boundaries are represented as clipped points and are free from artifacts. With the solution for boundary problems, TS framework can also accommodate texture atlases.

CONCLUDING REMARKS

TS provides a novel framework for high-quality texture mapping on point-based models without connectivity information. It is capable of mapping a low-resolution point-based model with high-resolution textures. The texture coordinates of neighboring splats are interpolated, and the resulting smooth texture coordinate field in the screen space is used to look up the pixel colors. In this way, point

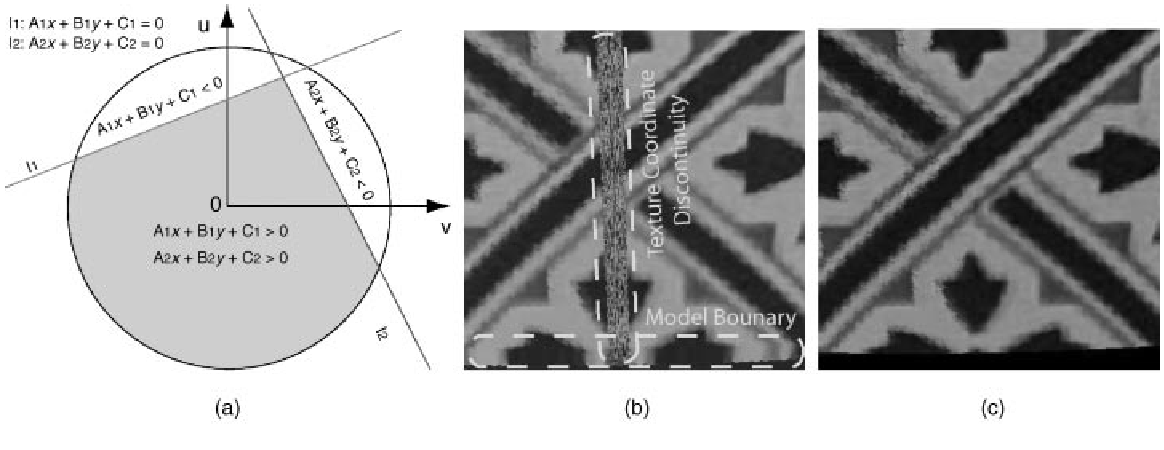


Figure 5. Clipping for points along boundaries. (a) For each point splat, up to two optional clipping lines are specified. Artifacts are marked by dotted yellow rectangles. (b) Clipping disabled. (c) Clipping enabled.

models with much higher texture resolution than that of the point model can be rendered.

BIBLIOGRAPHY

1. M. Levoy and T. Whitted, The use of points as display primitives. Technical Report TR 85-022, The University of North Carolina at Chapel Hill, 1985.
2. H. Pfister, M. Zwicker, J. van Baar, and M. Gross, Surfels: Surface elements as rendering primitives. *Proc. SIGGRAPH '00*. Reading, MA: ACM Press/Addison-Wesley, 2000, pp. 335–342.
3. A. Kalaiah and A. Varshney, Modeling and rendering of points with local geometry. *IEEE Trans. Visualization Comput. Graphics*, **9** (1): 30–42, 2003.
4. M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, Point set surfaces, *VIS '01: Proc. Conference on Visualization '01*, Washington, DC, 2001, pp. 21–28.
5. M. Zwicker, H. Pfister, J. van Baar, and M. Gross, Surface splatting. *Proc. SIGGRAPH '01*, 2001, pp. 371–378.
6. M. Botsch, M. Spernat, and L. Kobbelt, Phong splatting. *Proc. Eurographics Symposium on Point-Based Graphics*, 2004, pp. 25–32.
7. G. Schaufler and H. Wann Jensen, Ray tracing point sampled geometry. *Proc. Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, 2000, pp. 319–328.
8. M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt, High-quality surface splatting on today's GPUs. *Proc. Eurographics Symposium on Point-Based Graphics*, 2005, pp. 17–24.
9. M. Zwicker, M. Pauly, O. Knoll, and M. Gross, Pointshop 3D: An interactive system for point-based surface editing. *ACM Trans. Graph.*, **21** (3): 322–329, 2002.
10. L. Ren, H. Pfister, and M. Zwicker, Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Proc. Computer Graphics Forum (Eurographics 2002)*, 2002, pp. 461–470.
11. M. Levoy, Efficient ray tracing of volume data. *ACM Trans. Graph.*, **9** (3): 245–261, 1990.
12. U. Tiede, K. H. Hoehne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke, Investigation of medical 3D-rendering algorithms. *CGA*, **10** (2): 41–53, 1990.
13. R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang, Volvis: A diversified volume visualization system. *Proc. Visualization '94*, 1994, pp. 31–38.
14. H. Gouraud, Continuous shading of curved surfaces. *IEEE Trans. Comput.*, **C-20** (6): 623–629, 1971.
15. B. Tuong Phong, Illumination for computer-generated images, Ph.D. Dissertation, Department of Computer Science, University of Utah, Salt Lake City, July 1973.
16. A. Kalaiah and A. Varshney, Differential point rendering. *Proc. 12th Eurographics Workshop on Rendering Techniques*, London, UK, 2001, pp. 130–150.
17. X. Yuan, M. X. Nguyen, L. Qu, B. Chen, and G. W. Meyer, TSplatting: Mapping high quality textures on sparse point sets. *IEEE Trans. Visualization Comput. Graphics*. In press.
18. L. Qu, X. Yuan, M. X. Nguyen, G. Meyer, B. Chen, and J. Windsheimer, Perceptually guided rendering of textured point-based models. *Proc. Eurographics Symposium on Point-Based Graphics*, 2006, pp. 95–102.
19. L. Williams, Pyramidal parametrics. *SIGGRAPH '83: Proc. 10th Annual Conference on Computer Graphics and Interactive Techniques*, New York, 1983, pp. 1–11.
20. M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly, Perspective accurate splatting. *GI '04: Proc. 2004 Conference on Graphics Interface*, 2004, pp. 247–254.

FURTHER READING

G. Wolberg, *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA:, 1994.

P. S. Heckbert, Survey of texture mapping, *IEEE Comput. Graph. Applicat.*, **6** (11): 56–67, 1986.

XIAORU YUAN
MINH X. NGUYEN
BAOQUAN CHEN
University of Minnesota at Twin Cities
Minneapolis, Minnesota

L

LIGHTING

All we see around us is due to light. If we want to generate synthetic images that look like reality with a computer, we need to simulate how light behaves: How it leaves its sources, how it is reflected on objects, and how it is attenuated in the fog before it finally enters the eye and lands on the retina or enters the camera lens and excites the film. We need to perform *lighting* simulation. Studying the interaction of light with objects for the purpose of image generation has a long history in art and has been experiencing a huge revival in the last 40 years in the field of computer graphics. The more realistic and more quickly generated the synthetic images are, the more applications they find. Computer graphics is being used in instruction and training, industrial and artistic design, architectural walkthroughs, video games, the film industry, and scientific visualization.

The goal of lighting simulation is to generate images that look like reality. To achieve this effect, one needs to start with a faithful description of the environment being simulated, which is the stage of *modeling*, in which one has to specify the geometry of all objects in the scene (objects' shapes, positions, and sizes), optical properties of their surfaces (reflection, transmission, etc.), positions and emission properties of light sources, and, finally, the position and properties of the virtual camera. After the modeling stage, the virtual scene is ready for simulation of light transport (i.e., for simulation of the way light propagates in the scene). After simulation is done, we can take the scene's synthetic snapshots, which is also called image rendering. Often, the rendering stage and light transport simulation steps are combined into a single stage. In this case, the light simulation is restricted to finding how much light from the visible scene is ultimately reaching the sensors in a virtual camera.

The distribution of light in space, generated by various light sources, may vary significantly. Sun gives very intense rays of light originating from a single point. Such light will cause very definite shadows with hard edges. An overcast sky emits a very different light field—the light rays originate from all the positions over the sky dome and is distributed more or less uniformly in space. If such light generates any perceptible shadows at all, they will not manifest themselves as more than a dark and blurry smudge on the ground around an object.

Light goes through many interactions with scene objects before it reaches our eyes. Depending on the object shapes and their reflectance properties, light propagation creates various patterns and phenomena so well known from our everyday life. A matt wall reflects light *diffusely*, which means that it scatters light in all possible directions. Diffuse reflections often lead to the phenomenon called *color bleeding* (Fig. 1a, see the reddish tinge on the left part of the dragon's body), when a diffuse object lends its color tone to another diffuse object. *Caustics* are another well-known phenomenon, most often seen on a table under an interestingly shaped glass (Fig. 1b). It is formed by a light reflected by a shiny (specularly reflecting) object or refracted by a transparent object and focused on a diffuse receiver.

The goal of lighting is to simulate the light propagation in the scene to be able to reconstruct all the aforementioned phenomena such as hard and soft shadows, mirror reflections, refractions, color bleeding, and caustics. Today a large variety of lighting algorithms is available in computer graphics, differing by the degree to which they can faithfully simulate those phenomena. This article will give a short overview of those algorithms. Nowadays, there is a strong trend toward using physically plausible (or at least physically motivated) algorithms for lighting simulation. Therefore, before we delve into the workings of those algorithms, we will have to be more formal and define some physical quantities related to light propagation.

Lighting simulation is one of the tools used in a subfield of computer graphics called *image synthesis* or *rendering*. There are rendering techniques that do not use lighting simulation at all. Those techniques try to render objects in an expressive, artistic way, artificially stressing some of the objects' features. They are usually referred to as *non-photorealistic rendering* (NPR) techniques. Computer software for image synthesis is called a *renderer*.

A thorough treatment of lighting simulation with the focus on the underlying physics is given in Ref. 1. A more easily accessible, implementation-oriented description of lighting simulation is given in Refs. 2 and 3.

RADIOMETRY

Light is a form of energy. The amount of light energy in unit time is described by *radiant flux* Φ measured in Joules per second or Watts (W). To better localize the radiant flux in space, we use *irradiance* $E(P) = \frac{\mathrm{d}\Phi(P)}{\mathrm{d}A}$, which describes flux per unit area arriving at an infinitesimal surface area *around point* P from all possible directions. Irradiance is measured in Watts per square meter [$\mathrm{W{\cdot}m^{-2}}$]. *Radiant exitance* (or *radiosity*) B is the same as irradiance but describes the light *leaving* an infinitesimal surface area.

To be able to localize the flux not only in space but also in directions, we need to define solid angle and differential solid angle. Similar to its counterpart in plane, *solid angle* measure the span of a set of directions in space. Formally, the solid angle subtended by an object from a point P is the area of the projection of the object onto the unit sphere centered at P. *Differential solid angle* $\mathrm{d}\omega$ is an infinitesimally thin cone of directions around a considered direction. Radiant flux density in directions is given by *radiant intensity*, $I(\omega) = \frac{\mathrm{d}\Phi(\omega)}{\mathrm{d}\omega}$. Intensity, flux per unit solid angle, is mostly used to describe light leaving a point light source.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

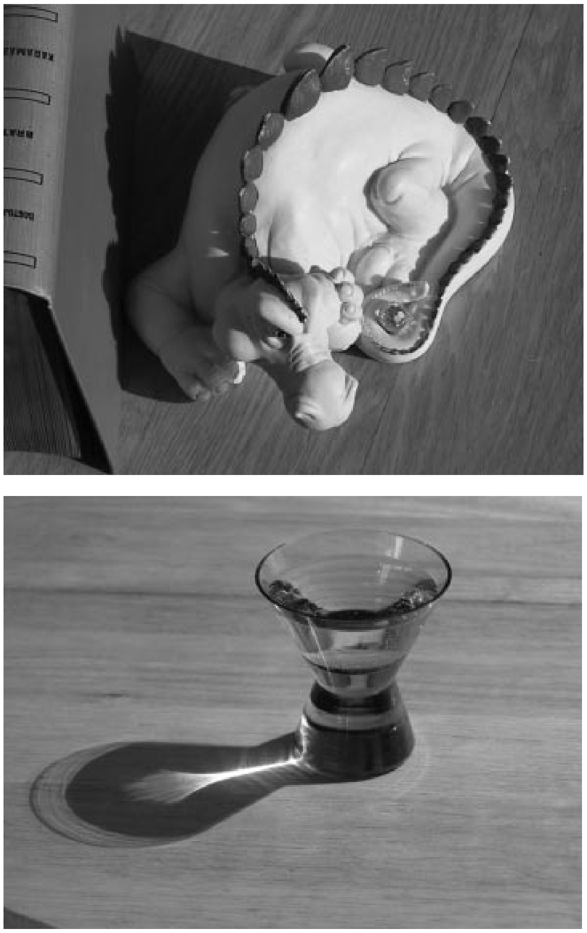


Figure 1. Complex light phenomena. *Color bleeding* (a) is due to diffusely reflected light landing on another diffuse surface. Here, the red book reflects light onto the dragon, effectively "transferring" the red color from the book to the back side of the dragon. *Caustics* (b) are patterns of light focused by reflections or refractions on a specular surface, landing on a diffuse object. Here, the green glass focuses the light onto the wooden table.

The physical quantity used to describe light energy distribution in space and in directions is *radiance* (Fig. 2), defined as $L(P,\omega) = \dfrac{\mathrm{d}\Phi(P,\omega)}{\mathrm{d}A\cos\theta\mathrm{d}\omega}$, that is radiant flux per unit *projected* area perpendicular to the considered direction, per unit solid angle. θ is the angle between the surface normal of the surface and the given direction. The term $\cos\theta$ in the denominator of the radiance equation converts the differential area to the *projected* differential area and makes a unit amount of radiance represent the same amount of light power reaching a surface independent of the surface's orientation.

Radiance is the most important quantity for lighting simulation for two reasons. First, it is the value of radiance and not any other radiometric quantity, which directly determines the intensity of light perceived by human observer or recorded on a photographic film.

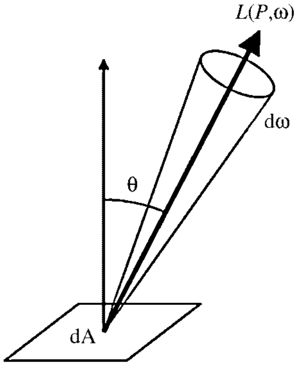


Figure 2. Radiance is radiant flux per unit projected area perpendicular to the considered direction per unit solid angle.

> *The ultimate goal of lighting simulation is to determine the value of outgoing radiance in the direction of the virtual camera for all visible points in the scene.*

Second, radiance is constant along rays in space (see Fig. 3).

> *The value of incoming radiance arriving at a point P from the direction ω is the same as the value of outgoing radiance leaving point Q (directly visible from P along ω) in the opposite direction $-\omega$.*

For these reasons, radiance is the most suitable quantity for computing energy transfer between surfaces in terms of straight rays transferring the light energy, which corresponds to the treatment of light propagation in *geometric optics* adopted by the vast majority of lighting simulation

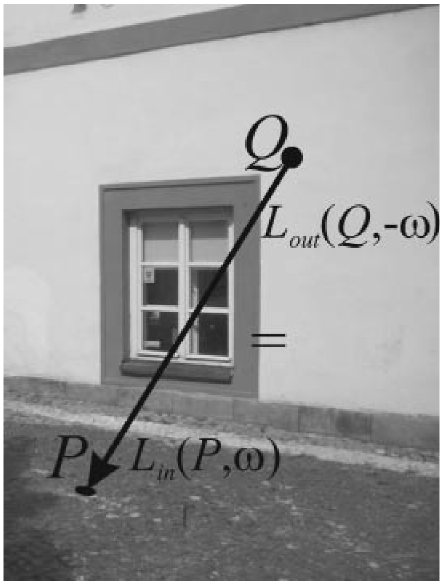


Figure 3. Radiance is constant along rays in space. The value of incoming radiance arriving at a point P from the direction ω is the same as the value of outgoing radiance leaving point Q (directly visible from P along ω) in the opposite direction $-\omega$.

algorithms: *Light propagates along straight lines until it hits an object and gets reflected*.

COLOR TREATMENT

Radiometric quantities are, in general, functions of light wavelength λ—they are spectral quantities. A spectrum of visible light is perceived as a color. The tri-chromatic nature of the human visual system (HVS) allows us to reproduce this color by a linear combination of three spectrally independent primary colors that are termed red, green, and blue (RGB) color (see the article on Color). In a lighting simulation software, the value of a radiometric quantity is thus represented by a 3-D vector corresponding to the three primary colors. This description of the spectral distribution of light energy is sufficient for simulating most of the perceivable lighting phenomena. However, some phenomena, such as dispersion, require performing the lighting simulation with the full spectrum representation and converting to RGB at the end of the simulation before the image display.

LIGHT SOURCES

Several different light source models are used for lighting simulation, some are completely abstract, others correspond to physical light emitters (see Fig. 4). *Point light*, a commonly used light source, emits all the light energy from a single point. The emitted light is expressed by *radiant intensity I*, which is the flux per unit solid angle in a given direction. The emission of a point light source is fully described by the *goniometric diagram*, a diagram of radiant intensity as the function of direction. An *omnidirectional point light* emits light equally in all directions (the goniometric diagram is constant). *Spot light* is a special type of point light source that emits only in a cone of directions around an axis. Even though point light sources do not exist in reality, they are very popular in lighting simulation, because they are very simple to handle and hence make the lighting simulation efficient. Another abstract light source type is *directional light*, which emits the light in parallel rays. It can be considered a special case of point light source where the point is at infinity. Point and directional light sources cast very sharp shadows.

Area light source refers to a 3-D object emitting light, a model that directly corresponds to physical luminaires. Emission of an area light source is described by the *emission distribution function* (EDF), which describes the outgoing radiance as a function of position on the light source and outgoing direction. Most common area light sources are rectangular or spherical lights with constant radiance over the surface position and direction. Area lights provide softer shadows than point light sources but are computationally more demanding.

Light probe image or *environment map* is a record of real world illumination at a point from all directions that can be used as a light source. Acquisition and illumination by light probe images is called image-based lighting and is described in a separate paragraph below.

LIGHT REFLECTION, BRDF, SHADERS

Light rays travel in straight lines in space until they hit the surface of an object, when the light gets either absorbed (changes into heat) or reflected. The way an object's surface reflects light defines the object's appearance: apparent color, glossiness, roughness, and so on. The color is a result of the spectral composition of the reflected light—some wavelengths get reflected more than others. Glossiness, roughness, specularity, and so on depend on the angular characteristics of the reflected light. Some materials reflect light in preferred directions (e.g., a mirror reflects only according to the Law of Reflection), others scatter the light in all directions (e.g., matte paints). The reflection behavior of a surface is described by the *bidirectional reflectance distribution function* (BRDF), which, for a given incoming direction ω_{in} and an outgoing direction ω_{out}, is the ratio of the differential irradiance due to the light incident from ω_{in} and the radiance reflected to ω_{out}, (Fig. 5a). Formally

$$f_r(P, \omega_{in}, \omega_{out}) = \frac{\mathrm{d}L_{out}(P, \omega_{out})}{L_{in}(P, \omega_{in}) \cos\theta_{in} d\omega_{in}}$$

In general, the BRDF is different for each point on a surface; it is a function of position: $f_r(P, \omega_{in}, \omega_{out})$. The position dependence of the BRDF creates the surface's characteristic visual texture.

BRDF defines the surface appearance and, therefore, it is at the heart of all rendering algorithms. It has to be evaluated for each visible point in the scene because it converts the light arriving at that point from the light sources (direction ω_{in}) into the light directed toward the eye (direction ω_{out}). Renderers make use of a small routine, called a *shader*, which uses the BRDF to compute the

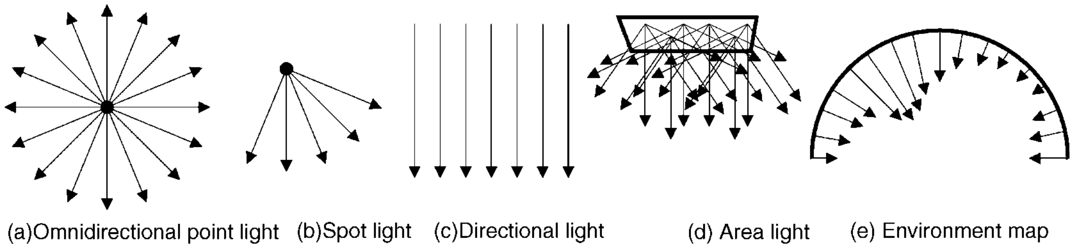
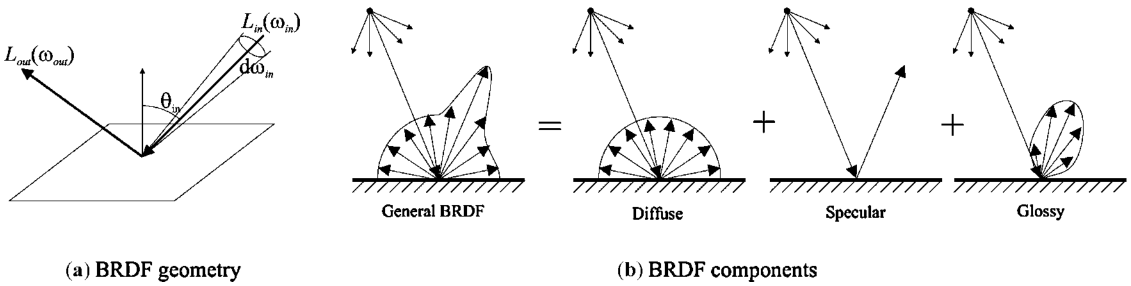


Figure 4. Various light source models are used for lighting simulation. (a) Omnidirectional point light, (b) spot light, (c) directional light, (d) area light, and (e) environment map.



(a) BRDF geometry

(b) BRDF components

Figure 5. (a) The geometry of light reflection at a surface. (b) A general BRDF is a sum of diffuse, specular, and glossy components.

amount of light reflected along an outgoing direction due to the light incident from one or more directions. Each object in the scene has a shader assigned to it depending on the desired reflectance properties.

BRDF Properties

A physically plausible BRDF must satisfy three fundamental properties. First, BRDF is a non-negative function. Second, BRDF is reciprocal, that is to say the BRDF value does not change if the incoming and outgoing directions are interchanged [i.e., $f_r(\omega_{in}, \omega_{out}) = f_r(\omega_{out}, \omega_{in})$]. Third, BRDF is energy conserving (i.e., a surface cannot reflect more than the amount of incident light).

BRDF Types

The BRDF functions are as wildly varying as are the appearances of objects in the real world. A *diffuse (Lambertian) BRDF* describes matte objects that reflect light uniformly in all directions. No matter where the light comes from, it is reflected uniformly along all directions in the hemisphere above the surface point. Diffuse BRDF is constant, $f_r^{diffuse}(\omega_{in}, \omega_{out}) = \frac{k_d}{\pi}$, where $0 \leq k_d \leq 1$. *Specular BRDF* describes a mirror surface, which only reflects light according to the Law of Reflection (angle of reflection is equal to the angle of incidence) and the reflected radiance is a constant fraction of the outgoing radiance. Apart from those two special cases, there is a wide range of *glossy* or *directional diffuse* BRDFs that can have completely arbitrary properties. Most commonly, the light is directed around the direction of perfect specular reflection, but it can also be reflected back toward the direction of incidence (retro-reflective materials). Real BRDFs are usually modeled by a sum of diffuse, specular, and glossy components (Fig. 5b).

BRDF Models

For the purpose of lighting simulation, general BRDFs are usually described by a mathematical model, parameterized by a small number of user settable parameters. The most common is the *Phong model*, whose physically plausible form is $f_r^{phong}(\omega_{in}, \omega_{out}) = k_d + k_s \cos^n \alpha$, where α is the angle between the mirror reflection direction ω_r of ω_{in} and outgoing direction ω_{out}. The user sets k_d (diffuse reflectivity—determines the surface color), k_s (specular reflectivity—determines the intensity of specular highlights), and n (specular exponent, or shininess—determines the sharpness of specular highlights). Many other BRDF models exist, see Refs. 1 or 3 for some examples.

BRDF Extensions

BRDF describes the reflection of light at one point. The model assumes that all light energy incident at a point is reflected from the same point, which is not true in the case of *subsurface scattering*, where the light energy enters the surface at one point, is scattered multiple times inside the material and leaves surface at a different point. This reflection behavior is exhibited by human skin (think of the frightening red color in one's face if he or she positions a flashlight under his or her chin), marble, milk, plant tissues, and so on. Reflection behavior of those materials is described by the *bidirectional surface scattering (reflectance) distribution function* (BSSDF).

DIRECT ILLUMINATION

Once the emission, geometry, and reflection functions of scene objects are specified, the light transport in the scene can be simulated. Real-time graphics (e.g., in video games) use the lighting simulation that disregards most of the interactions of light with scene objects. It only takes into account the so-called *direct illumination*. For each visible surface point in the scene, only the light energy coming *directly* from the light source is reflected toward the view point. The direct lighting simulation algorithm proceeds as follows.

For each image pixel, the scene point visible through that pixel is determined by a hidden surface removal algorithm [most commonly z-buffer algorithm (4)]. For a visible surface point P, the outgoing radiance $L_{out}(P, \omega_{out})$ in the direction of the virtual camera is computed by summing the incoming radiance contributions from the scene light sources, multiplied by the BRDF at P:

$$L_{out}(P, \omega_{out}) = \sum_{i=1}^{\#lights} L(Q_i \to P) \cdot f_r(P, Q_i \to P, \omega_{out}) \cdot \cos\theta_{in} \tag{1}$$

In this equation, Q_i is the position of the ith light source, $Q_i \to P$ denotes the direction from the ith light source toward the illuminated point P, and $L(Q_i \to P)$ is the radiance due to the ith light source, arriving at P. The equation assumes point light sources. Area light source can be approximated by a number of point lights with positions randomly picked from the surface of the area light source.

In the specific example of physically plausible Phong BRDF, the direct illumination would be computed using a formula similar to the following:

$$L_{out}(P,\omega_{out}) = \sum_{i=1}^{\#lights} I_i \cdot A_i \cdot [k_d + k_s \cos^n \alpha] \cdot \cos\theta_{in}$$

Here, I_i denotes the intensity of the light source. A_i is the light attenuation term, which should amount to the squared distance between the light source position and the illuminated point. In practical rendering, the attenuation might be only linear with distance or there might be no attenuation at all. The cosine term, $\cos\theta_{in}$, is computed as the dot product between the surface normal and direction ω_{in} (i.e., a unit vector in the direction of the light source).

In the basic form of the algorithm, all the light sources are assumed to illuminate all scene points, regardless of the possible occlusion between the illuminated point and the light source. It means that the determination of the outgoing radiance at a point is a local computation, as only the local BRDF is taken into account. Therefore, the algorithm is said to compute *local illumination*. All surfaces are illuminated exclusively by the light coming *directly* from the light sources, thus *direct illumination*. Direct illumination is very fast; using today's graphics hardware, it is possible to generate hundreds of images per second of directly illuminated reasonably complex scenes.

To include shadow generation in a direct illumination algorithm, the occlusion between the light source and the illuminated point has to be determined. To detect possible occluders, one can test all the scene objects for intersection with the ray between the illuminated point and the light source. However, this process is slow even with a specialized spatial indexing data structure. For real-time lighting simulation, shadows are generated using *shadow maps* or *shadow volume* (4).

GLOBAL ILLUMINATION

The simplistic direct illumination algorithm is generally not capable of simulating the interesting lighting effects mentioned in the introduction, such as specular reflections, color bleeding or caustics. The two main reasons are as follows. First, those effects are due to light reflected multiple times on its way from the source to the camera. In direct illumination, however, only a single reflection of light is taken into account (the reflection at the visible point P). Second, the point P under consideration is not only illuminated by light sources but also by the light reflected from all other surfaces visible from P. The light arriving at a point from other scene surfaces is called *indirect illumination* as opposed to direct illumination, which arrives directly from the light sources.

The total amount of light reflected by a surface at a point P in the direction ω_{out} is given by the following hemispherical integral referred to as the *Reflectance Equation* or *Illumination Integral*.

$$L_{out}(P,\omega_{out}) = \int_{\Omega_{in}} L_{in}(P,\omega_{in}) \cdot f_r(P,\omega_{in},\omega_{out}) \cdot \cos\theta_{in}\, \mathrm{d}\omega_{in}$$

Note the similarity between this equation and the direct illumination sum in Equation (1). The direct illumination sum only takes into account only those directions leading to the point light sources, whereas the Reflectance Equation takes into account all direction in the hemisphere, thereby translating the sum into an integral.

To compute the outgoing radiance L_{out} at the point P using the Reflectance Equation, one has to determine the incoming radiance L_{in} from every incoming direction ω_{in} in the hemisphere above P, multiply L_{in} by the BRDF and integrate over the whole hemisphere. The incoming radiance L_{in} from the direction ω_{in} is equal to the radiance leaving point Q (visible from P in direction ω_{in}) in the direction $-\omega_{in}$. That is to say, the incoming radiance at the point P from a single direction is given by the outgoing radiance at a completely different point Q in the scene.

The outgoing radiance at the Q is computed by evaluating the Reflectance Equation at Q, which involves evaluating the outgoing radiance at many other surface points R_i for which the Reflectance Equation has to be evaluated again. This behavior corresponds to the aforementioned multiple reflections of the light in the scene. The light transport is *global* in the sense that illumination of one point is given by the illumination of all other points in the scene.

By substituting the incoming radiance $L_{in}(P,\omega_{in})$ at P by the outgoing radiance L_{out} at the point directly visible from P in the direction ω_{in}, we get the *Rendering Equation* describing the *global light transport* in a scene:

$$L(P,\omega_{out}) = L^e(P,\omega_{out}) + \int_{\Omega_{in}} L(\Psi(P,\omega_{in}), -\omega_{in}) \cdot f_r(P,\omega_{in},\omega_{out}) \cdot \cos\theta_{in}\, \mathrm{d}\omega_{in}$$

The self-emitted radiance L^e is nonzero only for light sources and is given by the Emission Distribution Function. $\Psi(P,\omega_{in})$ is the ray casting function that represents the surface point visible from P in the direction ω_{in}. As all radiances in the Rendering Equation are outgoing, the subscript "out" was dropped. Global illumination algorithms compute an approximate solution of the Rendering Equation for points in the scene.

RAY TRACING

The simplest algorithm that computes some of the global illumination effects, namely perfect specular reflections and refractions, is ray tracing (5). The elementary operation in ray tracing is the evaluation of the ray casting function $\Psi(P,\omega)$ (also called *ray shooting*): Given a ray defined by its origin P and direction ω, find the closest intersection of the ray with the scene objects. With an acceleration space indexing data structure, the query can be evaluated in $O(\log N)$ time, where N is the number of scene objects.

Ray tracing generates images as follows. For a given pixel, a *primary ray* from the camera through that pixel is cast into the scene and the intersection point P is found. At the intersection point, *shadow rays* are sent toward the light sources to check whether they are occluded. Incoming radiance contributions due to each unoccluded light source are multiplied by the BRDF at P and summed to determine direct illumination. So far we only have direct illumination as described above. The bonus with ray tracing is the treatment of reflection on specular surfaces.

If the surface is specular, a *secondary ray* is cast *in the direction of the perfect specular reflection* (equal angle of incidence and reflection). The secondary ray intersects the scene at a different point Q, where the outgoing radiance in the direction $\omega = P - Q$ is evaluated and returned to the point P, where it is multiplied by the BRDF and added to the pixel color. The procedure to compute the outgoing radiance at Q is exactly the same as at P. Another secondary ray may be initiated at Q that intersects the surface at point R and so on. The whole procedure is recursive and is able to simulate multiple specular reflections. Refractions are handled similarly, but the secondary ray is cast through the object surface according to the index of refraction. This algorithm is also referred to as *classical* of *Whitted* ray tracing.

Ray tracing is a typical example of a *view-dependent* lighting computation algorithm—light, to be precise, the outgoing radiance, is computed only for the scene point visible from the current viewpoint of the virtual camera. In this way, ray tracing effectively combines image generation with lighting simulation.

MONTE CARLO

Ray tracing as described in the previous section handles indirect lighting on perfectly specular or transmitting surfaces, but still is not able to compute indirect lighting on diffuse or glossy ones. On those surfaces, one really has to evaluate the Reflectance Equation at each point. The problem on those surfaces is how to numerically evaluate the involved hemispherical integral. One possibility is to use Monte Carlo (MC) quadrature. Consider MC for evaluating the integral $I = \int_0^1 f(x)\mathrm{d}x$. It involves taking a number of uniformly distributed random numbers ξ_i from the interval $\langle 0, 1\rangle$, evaluating the function f at each of those points, and taking the arithmetic average to get an unbiased estimator of the value of integral $\langle I\rangle = \frac{1}{N}\sum_i f(\xi_i)$. The unbiased nature of the estimator means that the expected value of the estimator is equal to the value of the integral. The larger the number of samples $f(\xi_i)$, the more accurate the estimate is, but the expected error only decreases with the square root of the number of samples.

MONTE CARLO RAY TRACING

To apply MC to evaluating the hemispherical integral in the Reflectance Equation, one generates a number of random, uniformly distributed directions over the hemisphere. For each random direction, a secondary ray is cast in that direction exactly in the same way as in classic ray tracing. The secondary ray hits a surface at a point Q, the outgoing radiance in the direction of the secondary ray is computed (recursively using exactly the same procedure) and returned back. The contributions from all secondary rays are summed together and give the estimate of the indirect lighting. MC is only used to compute indirect illumination (the light energy coming from other surfaces in the scene). Direct illumination is computed using the shadow rays in the same way as in classic ray tracing. Figure 6 shows an image rendered with MC ray tracing.

Simply put, MC ray tracing casts the secondary rays in randomized directions, as opposed to classic ray tracing, which only casts them in the perfect specular reflection direction. By sending multiple randomized rays for each pixel and averaging the computed radiances, an approximation to the correct global illumination solution is achieved. A big disadvantage of MC ray tracing is the omnipresent noise, whose level only decreases with the square root of the number of samples.

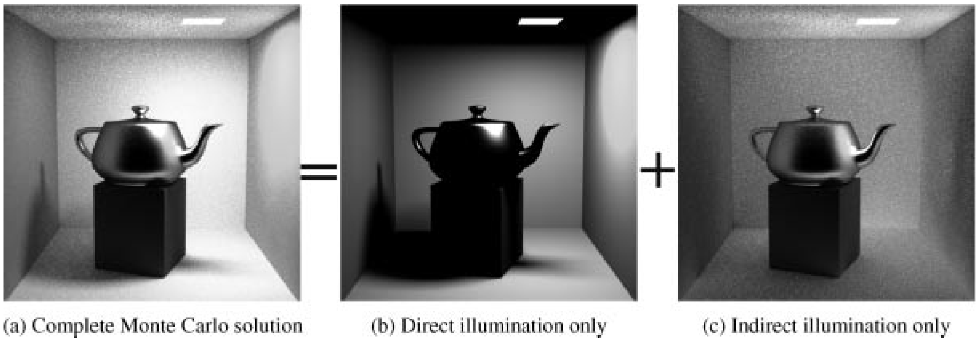

=
+
(a) Complete Monte Carlo solution
(b) Direct illumination only
(c) Indirect illumination only

Figure 6. Global illumination (a) is composed of the direct illumination (b) and indirect illumination (c). The indirect illumination is computed with MC quadrature, which may leave visible noise in images. Overall, 100 samples per pixel were used to compute the indirect illumination.

Classic ray tracing was able to compute indirect lighting only on specular surfaces. By applying MC to evaluating the Reflectance Equation, MC ray tracing can compute indirect lighting on surfaces with arbitrary BRDFs.

IMPORTANCE SAMPLING

Consider again evaluating the integral $I = \int_0^1 f(x)\mathrm{d}x$, but imagine this time that function *f* is near zero in most of the interval and only has a narrow peak around 0.5. If MC quadrature is run as described above by choosing the samples uniformly from the interval $\langle 0, 1\rangle$, many of the samples will not contribute to the integral estimate because they will miss the narrow peak, and the computational power put in them will be wasted. If no information is available about the behavior of *f*, if is probably the best one can do. However, if some information on the behavior of the integrand *f* is known, concentrating the samples around the peak will improve the estimate accuracy without having to generate excessive samples. *Importance sampling* is a variant of MC quadrature that generates more samples in the areas where the integrand is known to have larger values and reduces the variance of the estimate significantly. If the samples are generated with the probability density $p(x)$, the unbiased estimator is $\langle I\rangle = \frac{1}{N}\sum_i \frac{f(\xi_i)}{p(\xi_i)}$, where ξ_i's are the generated random numbers. Importance sampling significantly reduces the estimator variance if the probability density $p(x)$ follows the behavior of the integrand *f*(*x*). Going back to evaluating the Reflectance Equation, the integrand is $f_r(\omega_{in}, \omega_{out})L_{in}(\omega_{in})\cos\theta_i$. Incoming radiance $L_{in}(\omega_{in})$ is completely unknown (we are sending the secondary rays to sample it), but the BRDF is known. The direction ω_{out} is fixed (the integration is over the incoming directions), so we can use the known function $f_r^{\omega_{out}}(\omega_{in})\cos\theta_i$ as the probability density for generating the samples. For highly directional (glossy) BRDFs, the noise reduction is substantial, as shown in Fig.7.

PHOTON MAPS

Although importance sampling reduces the image noise a great deal, MC ray tracing still takes too long to converge to a noise-free image. Photon mapping (6) is a *two-pass* probabilistic lighting simulation method that performs significantly better.

In the first (photon tracing) pass, light sources emit photons in randomized directions. Brighter lights emit more photons, whereas dimmer ones emit less. Each time an emitted photon hits a surface, the information about the hit (hit position, photon energy, and incoming direction) is stored in the *photon map*, which is a spatial data structure specifically designed for fast nearest-neighbor queries. After storing the photon in the map, the tracing continues by reflecting the photon off the surface in a probabilistic manner (mostly using importance sampling according to the surface BRDF). The result of the photon tracing pass is a filled photon map, which roughly represents the distribution of indirect illumination in the scene.

In the second pass, lighting reconstruction and image formation are both done together. For each pixel, a primary ray is sent to the scene. At its intersection with a scene surface, the direct illumination is computed by sending the shadow rays as described above. Indirect illumination is reconstructed from the photon map. The map is queried for *k* nearest photons ($k = 50-200$), and the indirect illumination is computed from the energy and density of those photons with a procedure called *radiance estimate*. In this maner, the incoming radiance $L_{in}(\omega_{in})$ in the reflection equation is determined from the photon map without ever having to send any secondary rays.

Unlike ray tracing, photon mapping separates the lighting simulation stage from image generation. The first, photon tracing, pass is an example of a *view-independent*

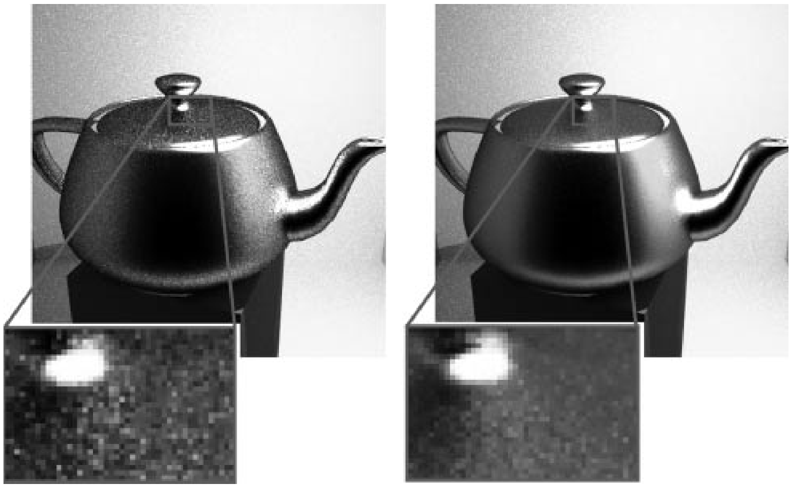


Figure 7. Importance can substantially reduce the noise level in the image. Both the left and right images were computed with MC ray tracing using 100 samples per pixel. Uniform hemisphere sampling was used for the image on the left; importance sampling was used for the image on the right.

lighting computation algorithm. The photons are traced and stored independently from the position of the camera. The image generation itself is performed in the second pass.

FINAL GATHERING, IRRADIANCE CACHING

As the photon map holds a very rough representation of illumination, the images generated with the reconstruction pass as described above do not usually have good quality because the indirect illumination randomly fluctuates over surfaces. Final gathering inserts one level of MC ray tracing into the lighting reconstruction pass of photon mapping and achieves high-quality images. For each primary ray hitting a surface, many secondary rays are sent (500–6000) to sample the indirect incoming radiance. At the points where the secondary rays hit a scene surface, the radiance estimate from the photon maps is used to determine indirect illumination. Because as many rough radiance estimates as there are secondary rays are averaged for each pixel, the image quality is good (the uneven indirect radiance is averaged out).

Final gathering is still very costly, because many secondary rays have to be sent for each pixel. However, on diffuse surfaces, the surface shading due to indirect illumination tends to change very slowly as we move over the surface, which is exploited in the *irradiance caching*(7) algorithm, where the costly final gathering is computed only at a few spots in the scene and interpolated for points in a neighborhood. Photon mapping with a form of final gathering is currently the most commonly used method for computing global illumination in production rendering software.

PARTICIPATING MEDIA—FOG, CLOUDS, AND SMOKE

All methods in the previous sections were built on one fundamental assumption—the value of radiance along a straight ray does not change until the ray hits a surface. However, if the space between the scene surfaces is filled with a volumetric medium like fog, cloud, or smoke, radiance can be altered by the interaction with the medium.

At each point of the medium the value of radiance along a straight line can be modified by a number of events. *Absorption* decreases the radiance due to conversion of the light energy into heat and is described by the volume absorption coefficient $\sigma_a[m^{-1}]$. *Scattering*, described by the volume scattering coefficient $\sigma_s[m^{-1}]$, is a process where a part of light energy is absorbed and then re-emitted in various directions. The angular distribution of the scattered light is described by the *phase function*. Scattering in a volume element is similar to reflection on a surface and the phase function corresponds to a surface BRDF. The total radiance loss due to absorption and scattering is described by the Beer's exponential extinction law: Radiance decreases exponentially with the optical depth of the medium. The optical depth is the line integral of $\sigma_a + \sigma_s$ along the ray. The value of radiance in a volume element does not only decrease, it can be increased due to emission and in-scattering. *In-scattering* is the radiance gain caused by the energy scattered into the ray direction from the neighboring volume elements. By integrating all the differential radiance changes along the ray, we get the integral form of the *Volume Light Transfer Equation* (see Ref. 3).

The light transfer in volumes is much more complicated than on surfaces and takes longer to compute. Fortunately, humans are not very skillful observers of illuminated volumes and large errors can be tolerated in the solution, which is often exploited by simplifying the transport—the most common simplification is the *single scattering* approximation, where only one scattering event is assumed on the way between the light source and the eye. It corresponds to direct illumination in surface lighting.

RADIOSITY METHOD

Radiosity (8) is a very different approach to computing global illumination on surfaces than MC ray tracing. It is based on the theory of radiative heat transfer and the solution is found with a finite element method. In the basic radiosity method, the scene surfaces are discretized into small surface elements (patches) of constant radiosity. Only diffuse BRDFs are considered. The energy exchange in the scene is described by the radiosity equation

$$B_i = B_i^e + \rho_i \sum_{j=1}^{N} F_{i,j} B_j$$

where B_i is the total radiosity of the ith patch, B_i^e is the radiosity due to self emission (non zero only for light sources), ρ_i is the surface reflectance, and $F_{i,j}$ is the form (or configuration) factor. The *form factor* $F_{i,j}$ is the proportion of the total power leaving the ith patch that is received by the jth patch. The whole radiosity equation means that the radiosity leaving the ith patch is the sum of the patch's self emitted radiosity and the reflection of the radiosities emitted by all other surface patches toward the ith patch. The radiosity emitted by a surface patch j toward the patch i is form factor $F_{i,j}$ times its total radiosity B_j. Knowing the form factors, the radiosity equations corresponding to all the patches of the scene can be cast in a system of linear equations with unknown patch radiosities and solved by standard linear algebra techniques. The details of the algorithm are given in a separate article.

In spite of a great deal of research that has gone into radiosity, the method did not prove to be practical. The most important disadvantage is that the method works correctly only with "clean" models (no overlapping triangles, no inconsistent normal, etc.), which is rarely the case in practice. Among computational disadvantages are large memory complexity due to the surface discretization, difficulty of computing form factors, and inability to handle general BRDFs.

Similar to the first pass of the photon mapping algorithm, radiosity gives a view-independent solution. It computes radiosity values for all surface patches regardless of the position of the camera. To generate the final image from a radiosity solution, final gathering is used because it masks the artifacts due to discretizing surfaces into patches.

IMAGE-BASED LIGHTING, HIGH DYNAMIC RANGE IMAGES

In spite of the success of modeling and rendering techniques to produce many realistic looking images of synthetic scenes, it is still not possible to model and render every naturally occurring object or scene. A compromise to this problem is to capture the real-world images and combine them with computer-rendered images of synthetic objects. This approach is commonly used for special effects in movies, in high-quality games, and in contents created for simulation and training. To achieve a seamless and convincing fusion of the virtual object with the images of the real world, it is essential that the virtual object be illuminated by the same light that illuminates other objects in the real scene. Thus, the illumination has to be measured from the real environment and applied during the lighting simulation on the synthetic object.

Image-based lighting (9) is the process of illuminating objects with images of light from the real world. It involves two steps. First, real-world illumination at a point is captured and stored as a *light probe image* (also called an *environment map*) (Fig. 8), an omnidirectional image in which each pixel corresponds to a discrete direction around the point. The image spans the whole range of intensities of the incoming light—it is a *high dynamic range (HDR) image*. An omnidirectional image can be taken by photographing a metallic sphere reflecting the environment or with a specialized omnidirectional camera. Multi-exposure photography is used to capture the high dynamic range. Each exposure captures one zone of intensities, which are then merged to produce the full dynamic range.

Once the light probe image is captured, it can be used to illuminate the object. To this end, the light probe might be replaced by a number (300–1000) of directional lights corresponding to the brightest pixels of the probe image.

IMAGE DISPLAY, TONE MAPPING

Once the physical radiance values are computed for a rectangular array of pixels, they have to be mapped to image intensity values (between 0 and 1) for display. One of the main problems that must be handled in this mapping process is that the emitting ranges of display pixel are limited and fixed, whereas the computed radiance values can vary by orders of magnitude. Another problem is that the displayed images are normally observed in illumination conditions that may be very different from the computed scene. *Tone reproduction* or *tone mapping* algorithms attempt to solve these problems. Reference 10 describes a number of such algorithms.

DISCUSSION

Lighting computation is an important branch of computer graphics. More than 30 years have been devoted to the research in lighting computation. Lighting engineers are

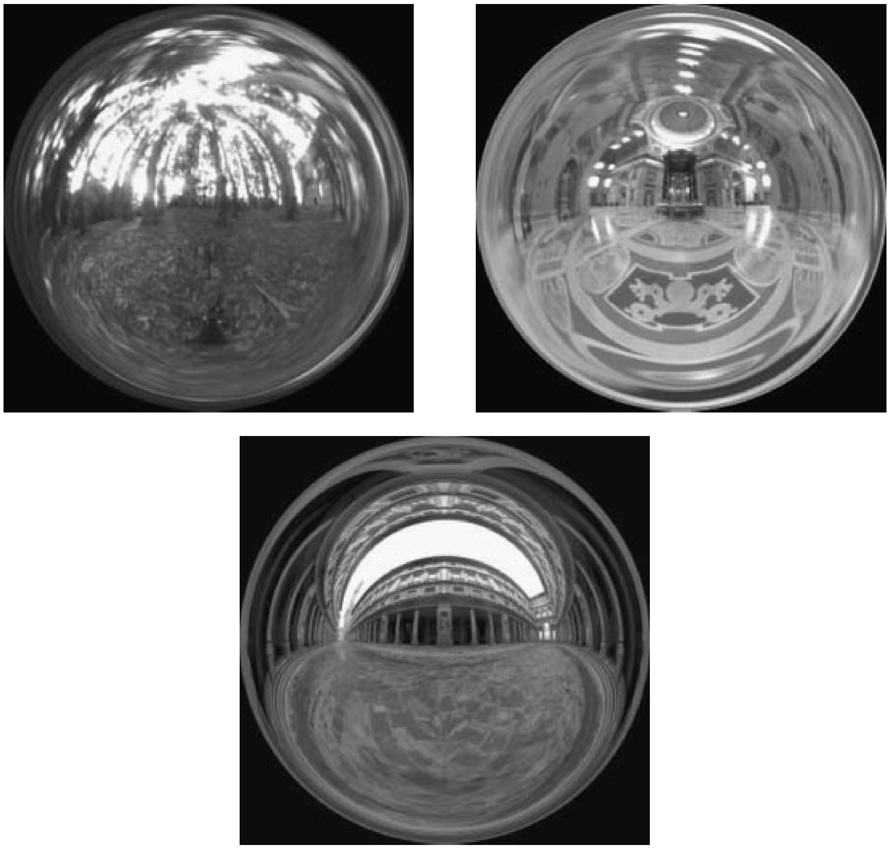


Figure 8. Example light probe images from the Light Probe Gallery at http://www.debevec.org/Probes. (Images courtesy of Paul Debevec.)

regularly using lighting computation to accurately predict the lighting distribution in architectural models. The entertainment industry is using lighting computation for producing convincing computer-generated images of virtual scenes.

Unfortunately, the computation time for global illumination simulation is still very high, prohibiting its full use in the entertainment industry, where thousands of images must be rendered within tight deadlines. Still, the industry is able to create images of stunning quality, mostly thanks to artists who know how light behaves and use their knowledge to “fake” the global illumination. As the indirect lighting would be missing in a direct illumination-only solution, lighting artists substitute indirect lighting by a number of *fill lights* to achieve a natural look of the object without having to resort to computationally expensive global illumination techniques. However, it takes a great deal of expertise to achieve a natural looking “global illuminated” scene using direct illumination-only techniques, and sometimes it is not possible at all. It is, therefore, desirable to make the global illumination available to all computer graphics artists. To that end, most of the current lighting research is being devoted to developing approximate but very fast global illumination algorithms, accurate enough to be imperceptible from the correct global illumination solution.

In the research community, MC techniques have been established as the method of choice for computing global illumination. Still, there are some important topics for future work. Efficient handling of light transport on glossy surfaces, adaptive sampling techniques exploiting a priori knowledge such as illumination smoothness are some examples. Finally, the computer-generated images will continue to look “computer generated” unless we start using reflection models that closely match reality. For this reason, more accessible and precise acquisition of BRDFs of real-world objects will be a topic of intensive research in future.

BIBLIOGRAPHY

1. A. S. Glassner, *Principles of Digital Image Synthesis*, San Francisco, CA: Morgan Kaufmann, 1995.
2. P. Dutré, P. Bekaert, and K. Bala, *Advanced Global Illumination*, Natick, MA: A. K. Peters Ltd., 2003.
3. M. Pharr and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, San Francisco, CA: Morgan Kaufmann, 2004.
4. T. Akenine-Möller and E. Haines, *Real-time Rendering*, 2nd ed.Natick, MA: A. K. Peters Ltd., 2002.
5. P. Shirley and R. K. Morley, *Realistic Ray Tracing*, 2nd ed. Natick, MA: A. K. Peters Ltd., 2003.
6. H. W. Jensen, *Realistic Image Synthesis Using Photon Mapping*, Natick, MA: A. K. Peters Ltd., 2001.
7. G. J. Ward and P. Heckbert, Irradiance gradients, *Proc. Eurographics Workshop on Rendering*, 85–98, 1992.
8. M. F. Cohen and J. R. Wallace, *Radiosity and Realistic Image Synthesis*, San Francisco, CA: Morgan Kaufmann, 1993.
9. P. Debevec, A tutorial on image-based lighting, *IEEE Comp. Graph. Appl.*, **Jan/Feb**: 2002.
10. E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec, *High Dynamic Range Imaging*, San Francisco, CA: Morgan Kaufman, 2005.

Jaroslav Křivánek
Czech Technical University
in Prague
Prague, Czech Republic
Sumanta Pattanaik
University of Central Florida
Orlando, Florida

PARAMETRIC SURFACE RENDERING

INTRODUCTION

Parametric surface (1) is a surface defined by a tensor-product of parametric curves, which are represented by parametric equations. As such a surface comes with a well-defined mathematical definition, it becomes one of the most common geometric tools for computer-aided geometric design and for the object modeling aids of many computer graphics applications. Typical examples of such a surface include Bézier surfaces (1) and non-uniform rational B-Spline (NURBS) surfaces (2). Unfortunately, because most existing graphics hardware only handle polygons, extra procedures must be carried out to convert a parametric surface into a polygon representation for rendering. This process introduces a significant bottleneck to graphics applications, which involve parametric surfaces, to assure interactiveness in terms of rendering performance.

ABOUT PARAMETRIC SURFACES

In the past, many different parametric surfaces have been developed. They include Bézier surfaces and NURBS surfaces, Bézier triangle, and multivariate objects. Such surfaces are mainly used to model smooth and curved objects, and particularly, to provide a good support for modeling deformable objects. For instance, Bézier surfaces and NURBS surfaces are typically used in computer-aided geometric design, whereas multivariate objects are mainly used in scientific visualization and in free-form deformation (3). A unique feature of the parametric surface is that its shape can be changed by modifying the position of its control points.

A parametric surface is modeled by taking a tensor-product on some parametric curves, which are formulated by parametric equations. In a parametric equation, each three-dimensional coordinate of a parametric curve is represented separately as an explicit function of an independent parameter u:

$$C(u) = (x(u), y(u), z(u)) \tag{1}$$

where $C(u)$ is a vector-valued function of the independent parameter u in which $a \leq u \leq b$. The boundary of the parametric equation is defined explicitly by the parametric intervals [a, b].

Bézier Surface

The Bézier surface (1) was developed by a French engineer named Pierre Bézier and was used to design Renault automobile bodies. The Bézier surface possesses several useful properties, such as endpoints interpolation, convex hull property, and global modification. Such properties make the Bézier surface highly useful and convenient to design curved and smooth objects. Hence, this surface is adopted widely in various CAD/CAM and in animation applications, and in general graphics programming packages, such as OpenGL and Performer. A Bézier surface is defined as

$$S_{Bez}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u) B_j^m(v) P_{ij} \quad \text{for} \quad 0 \leq u, v \leq 1 \tag{2}$$

where n and m are the degrees of the Bézier surface along u and v parametric directions, respectively. P_{ij} forms the control net. $B_i^n(u)$ and $B_j^m(v)$ are the basis functions, in which each is defined by a Bernstein polynomial:

$$B_i^n(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-1} \tag{3}$$

To evaluate a Bézier surface, we can apply the de Casteljau subdivision algorithm (4) to its Bernstein polynomials in both u and v parametric directions. For instance, in the u direction, we have

$$P_i^r(u) = (1-u) P_i^{r-1}(u) + u P_{i+1}^{r-1}(u) \tag{4}$$

for $r = 1, \ldots, n, i = 0, \ldots, n-r$ where n is the degree of the surface. Bernstein polynomials of the other parametric direction also are evaluated through a similar recursion process.

Although the Bézier surface provides a powerful tool in shape design, it has some limitations. Particularly, when modeling an object with complex shape, one may either choose to use a Bézier surface with prohibitively high degrees or a composition of pieces of low-degree Bézier surface patches by imposing an extra continuity constraint between the surface patches.

B-Spline Surface

The B-Spline surface (2) is formed by taking a tensor-product on B-Spline curves in two parametric directions, in which each of the B-Spline curves is a combination of several piecewise polynomials. The connecting points of the polynomial segments of a B-Spline curve are maintained automatically with C^{p-1} continuity, where p is the degree of the B-Spline curve. In addition, a B-Spline curve is composed of several spans. In the parametric domain, these spans are defined by a *knot vector*, which is a sequence *of knots*, or parameter values u_i, along the domain. A B-Spline polynomial is specified by a scaled sum of a set of basis functions. A B-Spline surface is defined as follows:

$$S_{Bsp}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u) N_j^q(v) P_{ij} \tag{5}$$

where n and m are the numbers of control points, and p and q are the degrees of surface in the u and v parametric

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

directions, respectively. P_{ij} forms the control net. $N_i^p(v)$ and $N_j^q(v)$ (v) are the basis functions. A basis function is defined by

$$N_i^p(u) = \begin{cases} 1 & \text{for} \quad (u_k \le u \le u_{k+1}) \\ 0 & \text{otherwise} \end{cases} \quad \textit{for} \quad p = 1 \tag{6}$$

$$N_i^p(u) = \frac{u - u_i}{u_{i+p} - u_i} N_i^{p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_i} N_{i+1}^{p-1}(u) \text{ for } p > 1 \tag{7}$$

The knot vectors of the B-Spline surface are defined as follows:

$$U = \{\underbrace{0, \ldots, 0,}_{p+1} u_{p+1}, \ldots, u_{r-p-1}, \{\underbrace{1, \ldots, 1,}_{p+1}\} \text{ for } r = n + p + 1 \tag{8}$$

$$V = \{\underbrace{0, \ldots, 0,}_{q+1} u_{q+1}, \ldots, u_{s-q-1}, \underbrace{1, \ldots, 1}_{q+1}\} \quad \text{for} \quad s = m + q + 1 \tag{9}$$

where U and V are the knot vectors along the u and v parametric directions, respectively. U has r knots, and V has s knots. The de Boor algorithm (5) was proposed to evaluate a B-Spline surface in parameter space with a recurrence formula of B-Spline basis functions. Other methods are proposed to accelerate the evaluation process, such as Shantz's adaptive forward differencing algorithm (6) and Silbermann's high-speed implementation for B-Splines (7).

Generally speaking, the B-Spline surface has similar properties to the Bézier surface. In addition, both surfaces are tightly related. For instance, a B-Spline surface can be reduced to a Bézier surface if it has both knot vectors in the following format:

$$\{\underbrace{0, \ldots, 0,}_{p+1} \underbrace{1, \ldots, 1}_{p+1}\} \quad \text{for} \quad p = \text{degree of B-Spline} \tag{10}$$

In addition, a B-Spline surface can be converted into several of Bézier surface patches through knot insertion (8).

NURBS Surface

A NURBS surface (2) is a rational generalization of the tensor product nonrational B-Spline surfaces. They are defined by applying the B-Spline surface Equation 5 to homogeneous coordinates rather than to the normal 3-D coordinates. The equation of a NURBS surface is defined as follows:

$$S_{Nrb}(u, v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} P_{i,j} R_{i,j}(u, v)}{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} R_{i,j}(u, v)} \tag{12}$$

where $w_{i,j}$ are the weights, $P_{i,j}$ form a control net, and $R_{i,j}(u,v)$ are the basis functions. NURBS surface not only shares all properties of nonrational B-Spline surface, but in addition, they possesses the following two useful features:

- They produce correct results under projective transformations, whereas nonrational B-Spline surfaces only produce a correct result under affine transformations.
- They can be used to represent lines, conics, planes, quadrics, and tori.

Similar to the Bézier surface, the NURBS surface is used widely in various CAD/CAM and animation applications, and in common graphics programming packages, such as OpenGL (9) and Performer (10), because such a surface can represent both analytic shapes and free-form surfaces with a common mathematical form. In addition, it comes with many useful geometric modeling toolkits, which include knot insertion, knot refinement, knot removal, degree elevation and degree reduction. Moreover, one can use a much smaller number of NURBS surface patches to construct objects with complex shapes in comparison with using a Bézier surface of the same degree, which helps to reduce effectively the effort in maintaining the visual continuity between surface patches.

Multivariate Objects

Bézier and B-Spline surfaces are used commonly in existing modeling and animation applications, because they are simple in geometric definition and possess a regular shape. In addition, their evaluation cost is low. However, such surfaces are difficult to use in modeling higher dimensional volumes or objects with very complicated shapes. Therefore, research has been conducted to explore higher dimensional multivariate objects (3,11,12), such as parametric volumes and hypersurfaces in R^n, n>3. The multivariate objects can be categorized mathematically into two major types. They are the multivariate objects formed by the multidimensional tensor-product of univariate Bernstein polynomials (S_A) and the generalized Bernstein polynomials over the barycentric coordinates (S_B). The defining equations of these two kinds of multivariate objects are shown as follows:

$$S_A(u, v, w, \ldots) = \sum_{i=0}^{l} \sum_{j=0}^{m} \sum_{k=0}^{n} \ldots B_i^l(u) B_j^m(v) B_k^n(w) \ldots P_{ijk\ldots} \tag{13}$$

$$S_B(u, v, w, \ldots) = \sum_{i+j+k\ldots=n} B_{ijk\ldots}^n(u, v, w, \ldots) P_{ijk\ldots} \tag{14}$$

where $B_i^l(u) B_j^m(v) B_k^n(w) \ldots$ and $B_{ijk\ldots}^n(u, v, w, \ldots)$ are the basis functions. $P_{ijk\ldots}$ form the control net. Note that the Bézier surface can be treated as a special case of the multidimensional tensor-product of univariate Bernstein polynomials, whereas the B-Spline surface or the NURBS surface can be represented in terms of the tensor-product of univariate Bernstein polynomials after applying appropriate knot insertion operations.

To evaluate a multivariate object defined by the multidimensional tensor-product of univariate Bernstein polynomials, we may subdivide it into a polygon model by applying the de Casteljau subdivision formula to all Berstein polynomials of different parametric directions. For example, in the u direction, we have

$$P^{rjk\ldots}_{ijk\ldots}(u) = (1-u)P^{r-1,j,k,\ldots}_{ijk\ldots}(u) + uP^{r-1,j,k,\ldots}_{i+1,j,k,\ldots}(u) \qquad (15)$$

for $r = 1, \ldots, l, i = 0, \ldots, l-r$ and all $j,k,\ldots\ldots$, where $(l, m, n,\ldots)$ is the degree of the surface. The other parametric directions have similar recursions. On the one hand, a multivariate object defined by the generalized Bernstein polynomials can be evaluated by applying the generalized de Casteljau subdivision formula. The generalized polynomials of degree n in generalized de Casteljau form are

$$P^{r}_{\mathrm{I}}(\mu) = uP^{r-1}_{\mathrm{I}+e1}(\mu) + vP^{r-1}_{\mathrm{I}+e2}(\mu) + wP^{r-1}_{\mathrm{I}+e3}(\mu) + \ldots \qquad (16)$$

for $r = 1,\ldots,n$ and $|\mathrm{I}| = n - r$, where $e1 = (1,0,0,\ldots)$, $e2 = (0,1,0,\ldots)$, $e3 = (0,0,1,\ldots)$, and $\mu = (u,v,w,\ldots)$.

On the other hand, multivariate objects may offer support to edit shapes of objects with arbitrary geometric representation. As a result, free-form deformation (FFD) (3) should be applied. It embeds an object to edit shapes in a regularly subdivided parallelepipedical 3-D parametric solid lattice, which is defined by a trivariate Bézier volume (a kind of multivariate object formed by the multidimensional tensor-product of univariate Bernstein polynomials). The solid lattice is referred to as the FFD lattice. Each sample point of the embedded object should be mapped to a parametric coordinate set of the FFD lattice. Shape editing of the embedded object can then be performed by moving the control points of the FFD lattice, in which the change in shape of the lattice would then be passed automatically to the embedded object. Based on FFD, extended free-form deformation (13) was proposed to relax the shape of the FFD lattice to arbitrary ones rather than sticking to the parallelepipedical one.

Properties of Parametric Surfaces

To examine the major properties of parametric surfaces that affect significantly the object modeling process, we broadly divide the parametric surfaces into two high-level classes by considering whether one is formulated by taking the tensor-product of just one polynomial or a several piecewise polynomials in each parametric direction. If just one polynomial is used in each parametric direction, the surfaces defined fal into the class of Bézier surfaces; otherwise, they belong to the class of B-Spline surfaces. The properties of these surfaces are depicted and compared as follows:

Local Modification Property. Local modification is proprietary to surfaces defined under the class of B-Spline, because any basis function $N_{i,p}(u)$ of a one-dimensional B-Spline would be evaluated as zero if we consider a parametric range outside the interval $[u_i, u_{i+p+1})$. Effectively, this approach implies a local control of each control point P_i of B-Spline surfaces. In other words, moving the control point P_k would only change a one-dimensional B-Spline within the parametric interval $[u_i, u_{i+p+1})$. For instance, in the case of a B-Spline surface, the control point $P_{i,j}$ affects only the shape of the surface within the parameter region $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$, where p and q are the degrees of the surface along u and v parameter directions, respectively.

Dependency between the Surface Degree and the Number of Control Points. Such dependency exists only on the surfaces defined under the class of Bézier surfaces, where the number of control points of a degree p Bézier curve is equal to $p + 1$. Essentially, when conducting object modeling using Bézier surfaces, if more control points are needed to model complex shapes, the degrees of such Bézier surfaces should then be increased accordingly. By doing this, the complexity in evaluating or in tessellating the Bézier surfaces would be increased significantly, which eventually leads to an undesired poor rendering performance. In contrast, surfaces defined under the class of B-Spline do not have such restriction. Effectively, the number of control points n depends on both the number of knots r and the degree p of a B-Spline, which equals

$$r = n + p + 1 \qquad (11)$$

whereas the degree of a B-Spline is fixed, one is allowed to choose any number of knots to define the B-Spline, which implies that the number of control points of a B-Spline is governed primarily by the number of knots. As the complexity in evaluating a B-Spline surface is mainly caused by the increase in the degree of the surface rather than in the increment in the number of knots, this property relaxes a B-Spline from experiencing very poor evaluation performance even if a lot of control points are added to the B-Spline to facilitate the modeling of complex shapes.

Continuity. Continuity (1,14) is a common property of both surfaces defined under the class of Bézier and B-Spline surfaces. It describes the smoothness at the junction of composite surface patches. Two types of continuity descriptors exist, namely *parametric continuity* C^n and *geometric continuity* G^n, where n typically indicates the level of continuity. Parametric continuity requires all derivatives of order 0 to $n-1$ at the junction of composite surface patches to be agreed, whereas geometric continuity imposes such agreement by considering the derivatives evaluated with respect to the arc-length around the junction, rather than to the parameterization at the junction. Parametric continuity suffers from the problem that it may not be held if the parameterization of the composite surface patches is changed, despite that the smoothness of the surface patches still is kept unchanged. In contrast, geometric continuity is appropriate for dealing with shape modeling as it allows one to modify the parameterization of a surface without affecting the smoothness of the surface. with regard to both classes of parametric surfaces, a B-Spline surface with degree (p,q) automatically maintains C^{p-1} and C^{q-1} continuity between polynomial patch segments in u and v directions, respectively. Comparing this with the composite

Bézier surface patches where an additional intersurface continuity constraint is required to be imposed, the B-Spline surface is thus preferred over the Bézier surface in the applications involving piecewise composite surface patches.

Degree Elevation and Reduction. Degree elevation and reduction (2) are tools used to adjust the degree of the parametric surfaces. They are applicable for both classes of Bézier and B-Spline surfaces. By altering the degree of a surface, the number of control points is increased or decreased. Hence, the degree of freedom for shape manipulation of the surface is adjusted as well. The most important feature of degree elevation and reduction is that both operations will not alter the shape of a surface, which provides the sufficient condition for one to freely change the degrees of parametric surfaces in an application, which is very useful. For instance, one may apply these tools to make all parametric surfaces in a modeling environment have the same degree. These surfaces can be stitched together to form a complex object, and the continuity condition of the object can be maintained.

GENERIC RENDERING METHODS

Pixel-Based Rendering

During the last two decades, various rendering methods have been developed for parametric surfaces. In the earlier stage, researchers focused on generating accurate images of parametric surfaces. Most methods developed were performed generally too slowly to support rendering parametric surface in real time. Catmull (15) presented a pixel-level subdivision method to render parametric surface by recursively subdividing a surface into smaller subpatches not bigger than a single screen pixel. Although Catmull derived an efficient subdivision algorithm for bicubic patches, the performance still is too slow to support an interactive display of surfaces because of the depth of subdivision. A more efficient method called scan-line display was developed and improved by several researchers (16–18). This method processes the screen pixels in the scan-line order. For each scan-line, the intersection of the scan plane with the surface forms a span. In practice, most scan-line based methods take advantage of spatial coherence to speed up the span computation. However, because of the inherent complexity of calculating scan-line intersections, these methods still do not perform fast enough for real-time display of large models. Whitted in Ref. 16 presented a method to render photo-realistic images of bicubic surfaces using ray tracing. The method subdivides repeatedly a surface intersected by a ray using Catmull's scheme (15) until the subpatch is small enough to approximate the point of intersection. This method was speed up by Kayjiya's numerical solution (19), where the calculation of ray patch intersection is reduced. Moreover, the performance can be additionally enhanced by Nishita et al.'s Bézier clipping algorithm (20), where the portion of a surface that does not intersect a ray is eliminated. However, the expensive ray intersection operations still make the interactive display of parametric surface difficult.

Polygon-Based Rendering

To speed up the rendering of parametric surface, many polygon-based rendering methods (5–8,21) have been developed. These methods subdivide a surface recursively into smaller patches until each patch is flat enough to be approximated by a polygon. Once the approximating polygons are computed, they can then be passed to and rendered by the hardware graphics accelerators. In contrast to the pixel-based methods, these methods use the polygon rendering capability available in existing graphics systems and hence may approach real-time rendering. The polygon-based rendering methods can be categorized into the *polynomial evaluation method, subdivision method*, and *frame-to-frame coherence method*, which are shown as follows:

- **Polynomial Evaluation Method:** A direct way to tessellate a parametric surface into polygons for rendering is performed by evaluating the surface equation for a succession of parametric pairs (u, v). The points obtained then form a set of polygons to approximate the parametric surface. The set of polygons is then passed to and processed by the hardware graphics accelerator. For example, the de Boor algorithm (5) was proposed to evaluate NURBS surfaces in the parameter space by using a recurrence formula of B-Spline basis functions. It is very useful for computer implementation, as one can implement this directly using a simple recursive function. An alternative approach to evaluate parametric surfaces is the Horner's algorithm (1). Instead of evaluating the surface equation along the (u, v) pairs in succession, it evaluates the surface polynomials in the form of a nested multiplication of monomials, which is generally faster. However, this method is numerically unstable because of the monomial form. Other methods are proposed to accelerate the evaluation process, such as Shantz's adaptive forward differencing algorithm (6) and Silbermann's high-speed implementation of NURBS (7). They propose alternative solutions to simplify the evaluation process of the parametric surface.
- **Subdivision Method:** Subdivision can be performed adaptively or uniformly. Adaptive subdivision subdivides recursively a parametric surface into surface patches until each patch is sufficiently flat or small enough to meet the screen-space projection threshold to assure a good display quality to the rendered surface. The surface patches created are held in a tree structure to ease the maintenance and tracing of the surface patches during the subdivision process. This approach can produce an optimized number of polygons for parametric surfaces with highly varying curvatures. Methods of this approach include those proposed by Clark (22), Barsky et al. (23) and Forsey et al. (21). However, extra care must be taken when using adaptive subdivision methods, as cracks may

appear in the generated polygon model. This problem occurs because resulting neighboring polygons may not be at the same resolution level. Hence, an additional crack prevention process is required to fix the cracks in the generated polygon model to ensure its visual continuity.

Uniform subdivision computes a constant step size along each parametric direction of a surface to generate a regular grid of polygons to represent a surface. Unlike adaptive subdivision, the polygon model created by uniform subdivision can be stored in an array instead of a tree structure, and the subdivision therefore is nonrecursive. On the other hand, although uniform subdivision can tessellate surfaces more efficiently than adaptive subdivision, usually it produces more polygons than necessary. Rockwood et al. (25) and Abi-Ezzi et al. (21,26) propose methods on uniform subdivision. In particular, Rockwood et al.'s method (25) subdivides a surface into a set of simple surface segments and tessellates these segments into a regular grid of polygons. A Coving and tiling process will then be conducted to handle the tessellation of the boundaries between the surface segments and that of the trimming regions of the surface. In practice, a variant of Rockwood et al.'s method (25) has been implemented in SGI GL and OpenGL libraries. Abi-Ezzi et al. (21,26) enhances this method by improving the computation of step size of polygonization and separating the compute-intensive and algorithm-intensive procedures.

- **Frame-to-Frame Coherence Method:** Kumar et al. (27) proposed a method to keep track of the user's viewpoint on a surface and to tessellate the surface according to the change of this viewpoint between successive frames. Similar to Rockwood et al.'s method, which subdivides a surface into a set of simple surface segments to facilitate additional processing, this method is based on the visibility of each surface segment in every next frame during run time and performs incremental tessellation on a surface segment or deletes one accordingly. Because there is usually only a small change in the viewpoint between two consecutive frames, this method minimizes the number of polygons generated within a given time frame.

Issues in Polygon-Based Rendering

The issues for applying the polygonal approximation methods to render parametric surfaces are summarized as follows:

- **Trimmed Surfaces:** If we restrict the domain of a parametric surface to a subset of its parametric space, the resultant surface is called a *trimmed surface*. We can trim a surface by enclosing a subset of area of the surface by a set of closed loops of directed curves called *trimming curves*. The enclosed area is called a *trimming region*. For rendering, a special triangulation procedure should be carried out around the boundary of the trimming region. Existing works for tackling this issue include Ref. 25 and 27.
- **Efficiency:** The method used to tessellate a parametric surface must be capable of calculating efficiently both the surface points and their corresponding normal vectors. The surface points form the vertices of the polygon model approximating the parametric surface. The normal vectors are used to shade and to support surface culling.
- **Display Quality:** The display quality is a very important criterion in surface rendering. To increase the display quality, one may generate a lot of polygons to represent the surface, although this is traded for the rendering performance. Nevertheless, generating a lot of polygons may not be necessary to correspond to a good visual quality for displaying a surface. Hence, to guarantee the display quality of a parametric surface, one needs to determine an optimal number of polygons based on some viewing criteria, such as the screen-space projection (25), to obtain a smooth image of the surface according to the current user's view point.
- **Sampling Distribution:** In addition to determining the number of polygons to represent a parametric surface, the method also should determine the distribution of these polygons. Two ways to handle this issue are *uniform tessellation* and *adaptive tessellation*. Uniform tessellation subdivides a surface evenly with a predetermined sampling size. Adaptive tessellation subdivides a surface with nonuniform resolution according to the local geometry, such as the curvature, to optimize the total number of polygons generated. In general, uniform tessellation produces more polygons for rendering than adaptive tessellation. However, extra computational time is needed for adaptive tessellation to determine the distribution of the polygons.
- **Crack Prevention:** When the adaptive tessellation is used, cracks may appear from the different sizes of neighboring polygons as shown in Fig. 1. Additional procedures must be performed to ensure the visual continuity of the image by removing the cracks (28).
- **Frame-to-Frame Coherence:** To accelerate the rendering of a parametric surface, the frame-to-frame coherence could reduce significantly the number polygons that are generated between successive frames. This coherence is helpful to accelerate the rendering performance when the viewing parameter to the surface is changing continuously.

RENDERING OF DEFORMABLE SURFACES

In real life, many objects are deformable, in which their shapes could be changed. Examples include human or animal characters, facial expressions, and soft objects such as clothes. The incorporation of such objects in computer graphics applications is particularly attractive as it is useful to enhance the realism of such applications. However, the rendering process of such objects generally is expensive. When an object deforms, a rendering process for the object is needed to re-run repeatedly from frame to frame to produce appropriate pixels or polygons for

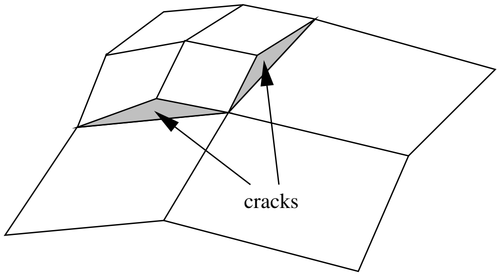


Figure 1. Cracks from adaptive tessellation.

graphics hardware to display the object. This process poses a significant computation burden on the graphics applications, which makes the real-time rendering of deformable objects difficult. Hence, deformable objects are seldom incorporated in interactive types of graphics applications.

To address this problem, an incremental surface rendering method (28,29) has been proposed, which is based on two fundamental techniques, *incremental polygon updating* and *resolution refinement*. The basic idea of incremental polygon model updating is to maintain two data structures of a deformable surface, the surface model and a polygon model representing the surface model. As the surface deforms, the polygon model is not regenerated through polygonization. Instead, it is updated incrementally to represent the deforming surface. This updating accelerates the rendering process of deformable surfaces by concerning the incremental evolution of such surfaces in successive frames. More specifically, we consider whenever a control point $P_{ijk}\ldots$ of a surface is moved to $\overline{P}_{ijk\ldots}$ with a displacement vector $\vec{V} = \overline{P}_{ijk} - P_{ijk\ldots}$, the incremental difference between two polygonal representations for a parametric surface before and after the control point movement can be represented as follows. For the multidimensional tensor-product of univariate Bernstein polynomials:

$$\overline{S}_A(u,v,w,\ldots) - S_A(u,v,w,\ldots)$$
$$= B(B_i^l(u)B_j^m(v)B_k^n(w)\ldots)(\overline{P}_{ijk\ldots} - P_{ijk}) = \alpha_{ijk\ldots}\vec{V} \quad (17)$$

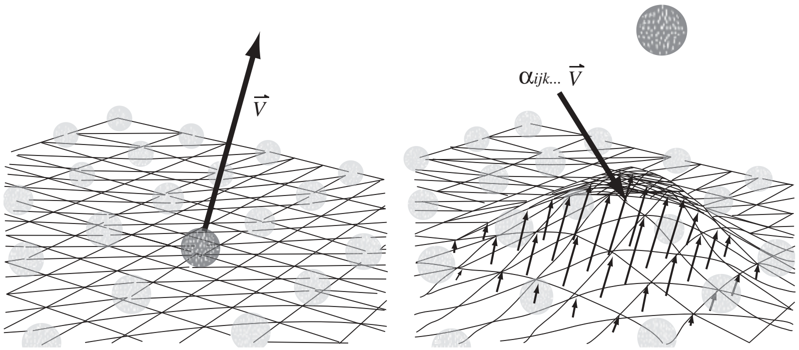
where $\alpha_{ijk\ldots} = B_i^l(u)B_j^m(v)B_k^n(w)\ldots$. For the generalized Bernstein polynomials over the barycentric coordinates:

$$\overline{S}_B(u,v,w,\ldots) - S_B(u,v,w,\ldots)$$
$$= (B_{ijk\ldots}^n(u,v,w,\ldots))(\overline{P}_{ijk\ldots} - P_{ijk\ldots}) = \beta_{ijk\ldots}\vec{V} \quad (18)$$

where $\beta_{ijk\ldots} = B_{ijk\ldots}^n(u,v,w,\ldots)$ It is obvious that the two deformation coefficients $\alpha_{ijk\ldots}$ and $\beta_{ijk\ldots}$ are constants for each particular set of $(u,v,w,\ldots)$ parameter values. Hence, if the resolution of the polygon model representing the surface remains unchanged before and after deformation, one may precompute the deformation coefficients and update the polygon model incrementally by the deformation coefficients and the displacement vector of the moving control point. In the implementation, the incremental polygon model updating is carried out in two stages: the *preprocessing* stage and the *run-time* stage. In the preprocessing stage, a surface is tessellated to obtain a polygon model and a set of deformation coefficients $\alpha_{ijk\ldots}$ or $\beta_{ijk\ldots}$ for each control point is evaluated. As the surface deforms during run time, the polygon model is updated incrementally with the set of deformation coefficients and the displacement vector of the moving control point. Figure 2(a) shows a surface deformation by moving a control point with displacement $\vec{V}$. Figure 2(b) shows the incremental updating of the affected polygon vertices. With the incremental polygon model updating technique, a surface point on the deformed surface $\overline{s}_{ijk\ldots}$ can be calculated by

$$\overline{s}_{ijk\ldots} = s_{ijk\ldots} + \alpha_{ijk\ldots}\vec{V} \quad (19)$$

where $s_{ijk\ldots}$ and $\overline{s}_{ijk\ldots}$ are the surface points before and after deformation, respectively. $\vec{V}$ is the displacement vector of the current moving control point, and $\alpha_{ijk\ldots}$ is the deformation coefficient associating with the surface point s and the current moving control point. This technique is efficient. First, only one vector addition and one scalar-vector multiplication are found on each affected vertex of the polygon model to produce the deformed one. Second, the precomputed deformation coefficients are constant, and hence, no recomputation is needed. Second, since a surface point on the deformed surface is calculated by Equation (19) regardless



(a) (b)

Figure 2. Incremental polygon model updating.

of the type of deforming surfaces to handle, the computational complexity is therefore independent of the complexity of the defining equation of the deforming surface. In other words, this technique has a constant and low computational complexity for all types of deformable parametric surfaces.

On the other hand, when a surface deforms, its curvature also is changed. If the curvature is increased or decreased by a large amount during the deformation process, the resolution of the corresponding polygon model may become too coarse or too high to represent the deformed surface, respectively. To overcome this problem, a resolution refinement technique has been proposed to refine incrementally the resolution of the polygon model and to generate the corresponding deformation coefficients according to the change in the local curvature of the surface and some animation parameters, such as the viewer-object distance or the screen projection size of the object. More specifically, resolution refinement considers the fact that either a surface is modeled by the multidimensional tensor-product of univariate Bernstein polynomials or the generalized Bernstein polynomials over the barycentric coordinates, and it can be subdivided through de Casteljau formula as shown in Equations (15) and (16), respectively. By subtracting a subdivided deformed surface generated by the de Casteljau formula with its nondeformed counterpart, one can easily deduce that deformation coefficients incrementally can also be generated through de Casteljau formula. As a result, the de Casteljau formula provides a means to refine incrementally the polygon model representing the deformable surface and to generate corresponding deformation coefficients for newly added polygon vertices to support incremental polygon model updating.

All in all, the incremental surface rendering method (28,29) provides a unique solution to allow deformable parametric surfaces to be rendered interactively. Rendering deformable parametric surfaces with this method can be roughly 3 to 15 times faster than applying generic rendering methods. In addition, an extended version of this method has been published in Ref. 30 to cover trimmed parametric surfaces.

CONCLUSION

In conclusion, a parametric surface offers several advantages for object modeling. First, it has a well-defined mathematical definition, which provides a concise and systematic way to represent objects. Second, a parametric surface produces scalable geometry; i.e., all fine detailed features of the modeled object could be reserved without any loss even if the object is undergoing arbitrary zooming. Third, the control points provide a native aid to support object–shape modification. Despite the advantages, the rendering process of parametric surfaces is typically time consuming, as such a process is not natively supported by existing hardware graphics accelerators. Another major obstruction to providing native hardware support in rendering parametric surface is that several different kinds of parametric surfaces exist, each of them should be evaluated differently. To date, a unified way for surface evaluation still is not yet available. Another factor that hinders the rendering performance of parametric surface is that if a parametric surface is deforming, a generic rendering process must be re-run for every time frame to generate the updated pixel- or polygon-based representation to render the shape changed surface.

To improve the rendering performance, incremental update always is an effective approach, which helps to minimize the computational overhead for surface rendering. For instance, one can refine or prune only certain subregions of a parametric surface. More specifically, the subregions should be those experiencing visibility changes or undergoing deformation. On the other hand, incremental polygon updating, as in Refs. 28 and 29, accelerates the rendering of a deformable parametric surface by maintaining a set of deformation coefficients and uses the coefficients to update incrementally the polygonal representation of the surface. In the future, a major effort will be focused on the development of new surface rendering methods by taking advantage of the parallel processing power of the graphics processing unit to accelerate significantly the surface rendering process. Developing such methods, however, is not straightforward. In particular, the irregularity of trimming curves/regions makes parallelizing the rendering method for trimmed parametric surfaces difficult. A preliminary attempt to address this issue can be found in Ref. 31. On the other hand, the methods developed also should take care of the handling of object deformation. To this end, when supporting object deformation, one should minimize the memory usage and the amount of data transfer to and from the texture memory or to other kinds of memory storages.

BIBLIOGRAPHY

1. G. Farin, *Curves and Surfaces for CAGD* (5th ed.): *A Practical Guide*, London: Academic Press, 2002.
2. L. Piegl and W. Tiller, *The NURBS Book* (2nd ed.), New York: Springer-Verlag, 1997.
3. T. Sederberg and S. Parry, Free-form deformation of solid geometric models, *Proc. of ACM SIGGRAPH*, 1986 pp. 151–160.
4. P. de Casteljau, *Shape Mathematics and CAD*, London: Kogan Page, 1986.
5. C. deBoor, On calculating with B-splines, *J. Approx. Theory*, **6**: 50–62, 1972.
6. M. Shantz and S. L. Chang, Rendering trimmed NURBS with adaptive forward differencing, *Proc. of ACM SIGGRAPH*, 1988, pp. 189–198.
7. M. Silbermann, High speed implementation of nonuniform rational B-splines (NURBS), *SPIE Vol. 1251 Curves and Surfaces in Computer Vision and Graphics*, 1990, pp. 338–345.
8. W. Boehm, Inserting new knots into B-spline curves, *Computer -Aided Design*, **12** (4): 199–201, 1980.
9. OpenGL. Available http://www.opengl.org/.
10. SGI Performer. Available http://www.sgi.com/products/software/performer/.
11. P. Alfeld, Scattered data interpolation in three or more variables, in T. Lyche and L. L. Schumaker, *Mathematical Methods in Computer Aided Geometric Design*, San Diego, CA: Academic Press, 1989, pp. 1–34.

12. J. Hoschek, and D. Lasser, *Fundamentals of Computer Aided Geometric Design*, Natick, MA: A. K. Peters Ltd., 1993.
13. S. Coquillart, Extended free-form deformation: a sculpturing tool for 3D geometric modeling, *Proc. of ACM SIGGRAPH*, 1990, pp. 187–193.
14. B. Barsky and T. DeRose, Geometric Continuity of Parametric Curves, *Tech. Rep. UCB / CSD 84 / 205*, Dept. of Computer Science, University of California Berkeley, 1984.
15. E. Catmull, A Subdivision Algorithm for Computer Display of Curved Surfaces, PhD Thesis, Salt Lake City, UT: University of Utah, 1974.
16. J. Whitted, A scan line algorithm for computer display of curved surfaces, *Proc. of ACM SIGGRAPH*, **12** (3): 8–13, 1978.
17. J. Blinn, Computer Display of Curved Surfaces, PhD Thesis, Salt Lake City, UT: University of Utah, 1978.
18. J. Lane, L. Carpenter, J. Whitted and J. Blinn, Scan line methods for displaying parametrically defined surfaces, *commun. ACM*, **23** (1): 1980, pp. 23–34,
19. J. Kajiya, Ray tracing parametric patches, *Proc. of ACM SIGGRAPH*, 1982, pp. 245–254.
20. T. Nishita, T. Sederberg and M. Kakimoto, Ray tracing trimmed rational surface patches, *Proc. of ACM SIGGRAPH*, 1990, pp. 337–345.
21. S. Abi-Ezzi and L. Shirman, Tessellation of curved surfaces under highly varying transformations *Proc. of Eurographics*, 1991, pp. 385–397.
22. J. Clark, A fast algorithm for rendering parametric surfaces, *Proc. of SIGGRAPH*, 1979, pp. 289–99.
23. B. Barsky, A. D. DeRose and M. Dippé, An Adaptive Subdivision Method with Crack Prevention for Rendering Beta-spline Objects, *Tech. Rep. UCB/CSD 87/348*, Dept. of Computer Science, University of California Berkeley, 1987.
24. D. Forsey and R. Klassen, An adaptive subdivision algorithm for crack prevention in the display of parametric surfaces, *Proc. of Graphics Interface*, 1990, pp. 1–8.
25. A. Rockwood, K. Heaton and T. Davis, Real-time rendering of trimmed surfaces, *Proc. of ACM SIGGRAPH*, 1989, pp. 107–117.
26. S. Abi-Ezzi and S. Subramaniam, Fast dynamic tessellation of trimmed NURBS surfaces, *Proc. of Eurographics*, 1994, pp. 107–126.
27. S. Kumar and D. Manocha, Efficient rendering of trimmed NURBS surfaces, *Comp. Aided Design*, **27** (7): 509–521, 1995.
28. F. Li, R. Lau and M. Green, Interactive rendering of deforming NURBS surfaces, *Proc. of Eurograph.*, 1997, 47–56.
29. F. Li and R. Lau, Real-time rendering of deformable parametric free-form surfaces, *Proc. of ACM VRST*, 1999, pp. 131–138.
30. G. Cheung, R. Lau and F. Li, Incremental rendering of deformable trimmed NURBS surfaces, *Proc. of ACM VRST*, 2003, pp. 48–55.
31. M. Guthe, Á. Balázs, and R. Klein, GPU-based trimming and tessellation of nurbs and T-spline surfaces, *ACM SIGGRAPH 2005 Sketches*, 2005, pp. 1016–1023.

FREDERICK W. B. LI
University of Durham
Durham, United Kingdom

RADIOSITY

INTRODUCTION

Radiosity, a radiometric quantity, is the amount of light flux leaving unit surface area.[1] The unit of this quantity is watts/m^2. In the computer graphics field, the radiosity is mostly used to describe an object-space method for computing accurate radiosities on the object surfaces in a synthetic environment. In the rest of the article, we will describe this radiosity method.

Radiosity method is the very first physically based global illumination computation method. It was introduced into the computer graphics field by the researchers from Cornell University in 1984 (1). The basic idea is to first discretize the scene into patches, then set up a linear system for unknown radiosities of the patches, and finally solve the linear system to compute these radiosity values. The linear system describes the radiosity propagation between patches in equilibrium. Once the computation is done, it is possible to render the scene from any viewing point using any of the standard rendering techniques with radiosity[2] as the color of the patch. Radiosity is widely used in realistic rendering and real-time visualization of synthetic scenes.

Radiosity method is mostly used to compute global illumination solution in scenes containing diffusely reflecting surfaces and illuminated by diffuse light sources. It successfully simulates multiple inter-reflections. Such inter-reflection between diffuse surfaces is very difficult to simulate using other methods, like ray tracing. Radiosity method provides a solution for every surface patch in the scene, and the solution is independent of the viewer position, thus called a *view-independent object-space* technique.

To get a general idea about radiosity, it is helpful to observe how light propagates in a simple scene. Figure 1 shows an intuitive scenario of light propagation in a very simple scene made up of three patches. Light propagates between every pair of patches. In Fig. 1, patch 1 illuminates patch 2 and patch 3; patch 2 illuminates patch 1 and patch 3; patch 3 illuminates patch 1 and patch 2. Concave patches will also illuminate themselves. One can see from Fig. 2 that, during this light propagation process, only a fraction of light flux leaving patch *i* reaches another patch *j*. For diffuse surfaces, this fraction depends only on the form (size, orientation, and distance) of the pair of patches and hence is known as form factor. Form factor is denoted by $F_{i\to j}$, where subscript $i\to j$ represents light reaching patch *j* from patch *i*. A total of nine form factors are needed to describe the light propagation in the scene in Fig. 1. They are shown in Fig. 3. When the light propagation between the patches in Fig. 1 reaches equilibrium (i.e., light flux leaving or reaching each patch remains unchanged), the light flux leaving a patch can be expressed as the light flux emitted from the patch plus the reflected part of the light flux incident on the patch. The incident flux is the sum of the flux arriving from all other patches in the scene. Thus, we can write an expression for the equilibrium flux leaving a patch in terms of the light flux leaving all other patches of the scene. In Equation (1), we give the expressions for the three patches of the scene shown in Fig. 1.

$$\begin{aligned}\Phi_1 &= \Phi_{e,1} + \rho_1\Phi_1 F_{1\to 1} + \rho_1\Phi_2 F_{2\to 1} + \rho_1\Phi_3 F_{3\to 1}\\ \Phi_2 &= \Phi_{e,2} + \rho_2\Phi_1 F_{1\to 2} + \rho_2\Phi_2 F_{2\to 2} + \rho_2\Phi_3 F_{3\to 2}\\ \Phi_3 &= \Phi_{e,3} + \rho_3\Phi_1 F_{1\to 3} + \rho_3\Phi_2 F_{2\to 3} + \rho_3\Phi_3 F_{3\to 3}\end{aligned} \tag{1}$$

Each of these equations is linear, and together they form a linear system. A general form of the light flux equation in an arbitrary scene is given in Equation (2).

$$\Phi_i = \Phi_{e,i} + \rho_i \sum_{j=1}^{n} \Phi_j F_{j\to i} \text{ for } i = 1\ldots n \tag{2}$$

where Φ_i, $\Phi_{e,i}$, and ρ_i are the equilibrium light flux, the emitted light flux, and the diffuse reflectance of the surface patch *i*, respectively; $F_{j\to i}$ is the form factor between patch *j* and patch *i*; and *n* is the total number of patches in the scene.

Equation (2) is known as the light flux transport equation. $\Phi_{e,i}$ and ρ_i in the equation are the surface properties and are known quantities. $F_{j\to i}$ depends only on the geometry of surface patches *i* and *j*, and hence can be computed independent of the lighting condition. Given the values of $\Phi_{e,i}$, ρ_i, and $F_{j\to i}$, the unknown Φ_is can be computed by solving the linear system formed from Equation (2).

FORM FACTOR

The form factor $F_{i\to j}$ is defined as the fraction of light flux leaving patch *i* and reaching patch *j*. Equation (3) provides a mathematical expression of the form factor between two diffuse surfaces with uniform light flux over their area.

$$F_{i\to j} = \frac{1}{A_i}\int_{A_j} dA_y \int_{A_i} dA_x \frac{\cos\phi_x \cos\phi_y}{\pi r_{xy}^2} V(x,y) \tag{3}$$

where ϕ_x and ϕ_y are the angles between the normal's of differential patches dA_x and dA_y, respectively with the line connecting the two differential patches; r_{xy} is the distance between dA_x and dA_y; and A_i and A_j are areas of patches *i* and *j*. Figure 4 illustrates these parameters. *V(x,y)* is the visibility between the two differential patches and is

[1] Refer to the article on “Lighting” for the definition of various radiometric quantities.

[2] For a Lambertian surface, color is a constant times its radiosity.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

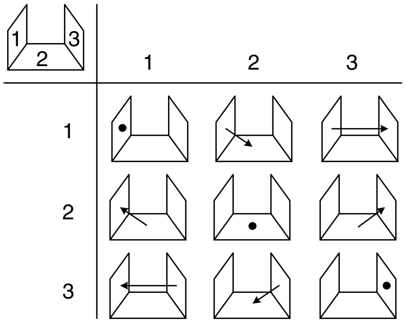


Figure 1. Light interaction.

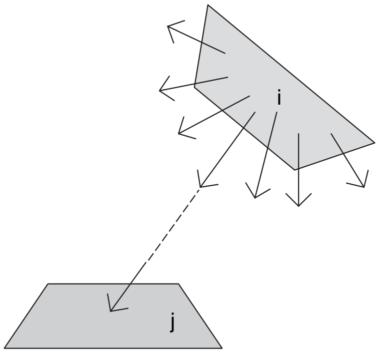


Figure 2. Light propagation.

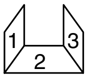
	1	2	3
1	$F_{1\to1}$	$F_{1\to2}$	$F_{1\to3}$
2	$F_{2\to1}$	$F_{2\to2}$	$F_{2\to3}$
3	$F_{3\to1}$	$F_{3\to2}$	$F_{3\to3}$

Figure 3. Form factors.

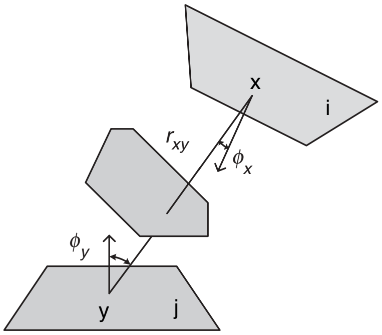


Figure 4. Form factor parameters.

expressed in Equation (4).

$$V(x,y) = \begin{cases} 0, \text{ if there is occlusion between } dA_x \text{ and } dA_y \\ 1, \text{ otherwise} \end{cases} \tag{4}$$

If we change the relationship between the patches i and j and wish to write an expression for $F_{j\to i}$, then it will be as follows:

$$F_{j\to i} = \frac{1}{A_j}\int_{A_j} dA_y \int_{A_i} dA_x \frac{\cos\phi_x \cos\phi_y}{\pi r_{xy}^2} V(x,y) \tag{5}$$

From these two form factor equations we can derive a useful property of form factor:

$$A_i F_{i\to j} = A_j F_{j\to i} \text{ or } \frac{F_{i\to j}}{F_{j\to i}} = \frac{A_j}{A_i}$$

In other words, the ratio of the two form factors between any two patches is inversely proportional to the ratio between the areas of the two patches.

From the definition of the form factors itself, we get another property: The sum of all form factors from any patch is no greater than 1 (i.e., $\sum_j F_{i\to j} \leq 1$).

RADIOSITY EQUATION

In the beginning of this article, we defined radiosity as the flux per unit area. If we use symbol B to denote radiosity and assume that the light is distributed uniformly over the area of the patch, then we have $B_i = \frac{\Phi_i}{A_i}$. Using this definition of radiosity and the property of form factor, we derive the radiosity transport equation (or *radiosity equation* for short) in Equation (6) from the flux transport equation given in Equation (2).

$$\begin{aligned}\frac{\Phi_i}{A_i} &= \frac{\Phi_{e,i}}{A_i} + \rho_i \sum_{j=1}^{n} \Phi_j \frac{F_{j\to i}}{A_i} = \frac{\Phi_{e,i}}{A_i} + \rho_i \sum_{j=1}^{n} \frac{\Phi_j}{A_j}\frac{A_j F_{j\to i}}{A_i} \\ &= \frac{\Phi_{e,i}}{A_i} + \rho_i \sum_{j=1}^{n} \frac{\Phi_j}{A_j} F_{i\to j} B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{i\to j}\end{aligned} \tag{6}$$

The derivation of Equation (6) uses the inverse area form factor relationship $\frac{A_j F_{j\to i}}{A_i} = F_{i\to j}$ described in the previous section.

Figure 5 summarizes the various steps for the radiosity computation algorithm. The process begins by inputting the model, followed by computing form factors and solving a linear system, and finally by displaying the scene from any arbitrary viewpoint. The input geometry is often discretized into smaller patches in the first step. The last step is often a hardware walk-through in the scene. Note that in this walk-through step, the change in view direction does not require the re-evaluation of any other step of the algorithm. Change in the surface property (diffuse reflectance and emissivity), however, does require repetition of the linear system solution step. But, form factors are not

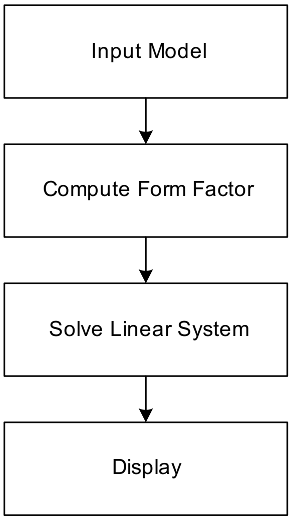


Figure 5. Radiosity algorithm.

affected by this change and hence they do not have to be recomputed.

SOLVING RADIOSITY EQUATION

By rearranging Equation (6), we can get the following matrix form of the radiosity system:

$$(I - M)B = E \tag{7}$$

where; I is the identity matrix, M =

$$\begin{bmatrix} \rho_1 F_{1\to 1} & \rho_1 F_{1\to 2} & \cdots & \rho_1 F_{1\to n} \\ \rho_2 F_{2\to 1} & \rho_2 F_{2\to 2} & \cdots & \rho_2 F_{2\to n} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_n F_{n\to 1} & \rho_n F_{n\to 2} & \cdots & \rho_n F_{n\to n} \end{bmatrix}, B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix}, \text{ and}$$

$$E = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}.$$

For convenience, we may denote (I–M) in Equation (7) with K, a square matrix with coefficients k_{ij}.

$$K \equiv I - M = (k_{ij})_{n\times n} \tag{8}$$

where

$$k_{ij} = \begin{cases} 1 - \rho_i F_{ii}, & \text{for } i = j \\ -\rho_i F_{ij}, & \text{for } i \neq j \end{cases}$$

With this change, the radiosity system is now represented by Equation (9), as follows:

$$KB = E \tag{9}$$

Given this matrix formulation, the radiosity system can be solved by inverting K and computing $B = K^{-1}E$. However, the size of the linear system of the radiosity equations for any nontrivial scene makes inverting the matrix K impractical. A common environment could have tens of thousands or millions of patches.

From the law of conservation of energy, $\rho_i \leq 1$, and the property of the form factor mentioned in the earlier section, we get the relation $\sum_{j=1}^{n}\rho_i F_{ij} \leq 1$, or in other words $\sum_{j=1,j\neq i}^{n}\rho_i F_{ij} \leq 1 - \rho_i F_{ii}$. Thus, each row of the matrix K satisfies the property $\sum_{j=1,j\neq i}^{n}|k_{ij}| \leq |k_{ii}|$, which makes K a diagonally dominant matrix, and hence guarantees that the radiosity equation has a unique solution, which also guarantees that an iterative method will finally converge to this unique solution. Hence, iterative methods or relaxation methods are commonly used for the radiosity solution. We describe two popular solution methods in the next section.

Two different classes of iterative methods have been used to solve the radiosity system [Equation (9)]. They include: *gathering methods* and *shooting methods*. These methods start from an initial estimation for the unknowns, and then they are iteratively updated based on the previous estimation until the linear system reaches convergence. To describe these methods, we go back to the original radiosity equation given in Equation (6).

Gathering Methods

A basic gathering method for solving the radiosity system is the iterative refinement of the radiosity values of the patches from the values computed at the previous iteration. The iteration scheme is shown in Equation (10). It is a reformulated version of Equation (6).

$$B_i^{k+1} = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j^k \tag{10}$$

where $B_j^0 = E_j$ and B_j^k is the result after evaluating Equation (10) for k iterative steps.

This iterative method is the same as the well-known *Jacobi* iteration method for solving a linear system. A simple extension of this method is to always compute B_i^{k+1} using the latest values of B, as shown in Equation (11).

$$B_i^{k+1} = E_i + \rho_i \sum_{j=1}^{i-1} F_{ij} B_j^{k+1} + \rho_i \sum_{j=i+1}^{n} F_{ij} B_j^k \tag{11}$$

This method is well known as the *Gauss–Seidel* method. It converges faster than the *Jacobi* method. An additional advantage of this method is that it is no longer required to maintain the previous and current sets of B values.

Both the *Jacobi* and *Gauss–Seidel* methods seem to be gathering radiosity values from all the patches of the scene to compute the radiosity of a single patch, which is why these methods are known as *gathering methods*.

Shooting Methods

This class of methods is based on another iterative reformulation of Equation (6). The reformulation is given in Equation (12). In this reformulation, the radiosity of a patch j is distributed or shot to all other patches i of the scene to update their radiosity values.

$$\begin{aligned} B_i^{k+1} &= B_i^k + \Delta Rad \\ \Delta B_i^{k+1} &= \Delta B_i^k + \Delta Rad \end{aligned} \tag{12}$$

where $\Delta Rad = \rho_i \Delta B_j^k F_{ij} = \rho_i \Delta B_j^k \dfrac{A_j}{A_i} F_{ji}$ and is the *unshot* radiosity.

This method starts with $B_i^0 = \Delta B_i^0 = E_i$. A patch j with maximum unshot flux (i.e., $\Delta B_j^k A_j = \max\limits_{1 \le m \le n} \{\Delta B_m^k A_m\}$) is chosen to shoot the radiosity to all other patches. After the shooting is done, the patch's unshot radiosity ΔB_j^k is set to zero. This process of choosing the patch and shooting its unshot radiosity is repeated until converged.

The total number of iterative steps required using iterative shooting methods is not any less than that required for the gathering method. However, in the shooting method, the radiosity values of the patches approach faster to the equilibrium value. Both the iterative methods are rarely run to achieve the full convergence of the solution. They are only computed for a fixed number of iterations in practice. Thus, the shooting method often creates a solution closer to the equilibrium solution, hence the rendering created from the partial radiosity solution obtained using a reasonably small number of iterations is visually preferable. Cohen et al. (2) first proposed this method for solving the radiosity system and called it the progressive refinement method. They computed a display radiosity value $B_i^{display}$ from the partial iterative solution, as shown by the equation

$$B_i^{display} = B_i^k + B_{ambient}^k \tag{13}$$

where $B_{ambient}^k$ is the ambient radiosity approximated by multiple bounces of average radiosity $\overline{\Delta B}^k$ with average diffuse reflectance $\bar{\rho}$, and their expressions are shown below.

$$B_{ambient}^k = \sum_{j=0}^{\infty} \bar{\rho}^j \overline{\Delta B}^k = \frac{1}{1-\bar{\rho}} \overline{\Delta B}^k \tag{14}$$

where

$$\overline{\Delta B}^k = \sum_{j=1}^{n} \Delta B_j^k A_j \Big/ \sum_{j=1}^{n} A_j \text{ and } \bar{\rho} = \sum_{j=1}^{n} \rho_j A_j \Big/ \sum_{j=1}^{n} A_j$$

FORM FACTOR COMPUTATION

Form factor must be computed before solving the radiosity system. Form factor computation is the most complex step of the radiosity method. In general, it takes 60–80% of the computation time in total. There are two classes of methods to compute form factors: analytical methods and numerical methods.

Analytical Methods

The analytical methods are useful only in the absence of the occlusion between two patches. The most often used analytical method is from Nishita and Nakamae (3). It formulates the form factor between a differential patch dS_i of the patch S_i to patch S_j as a line integral. The formulation is given in Equation (15). This method does not consider occlusion between dS_i and S_j.

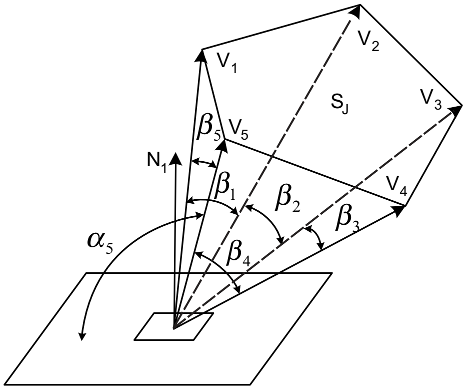


Figure 6. dS_i and S_j.

$$F_{dS_i \to S_j} = \frac{1}{2} \sum_{k=1}^{m} \beta_k \cos \alpha_k \tag{15}$$

where $\cos \beta_k = \dfrac{QV_{k+1}}{|QV_{k+1}|} \cdot \dfrac{QV_k}{|QV_k|}$, $\cos \alpha_k = \dfrac{N_i}{|N_i|} \cdot \dfrac{QV_k \times QV_{k+1}}{|QV_k \times QV_{k+1}|}$, and V_k-s are the vertices of patch S_j and Q is the center of the differential patch dS_i (see Fig. 6 for illustration).

Numerical Methods

Numerical methods are often used in the form factor computation. Hemicube method (4) is one of the first proposed and most popular numerical methods. This method also computes form factor $F_{dS_i \to S_j}$ between a differential patch dS_i to all the patches S_j in the scene. In this method, a virtual unit hemicube is set up around dS_i. The faces of the hemi-cube are discretized into a number of rectangular pixels (See Fig. 7 for illustration). The form factors ΔF_q of pixel q from dS_i to the pixel q on the virtual hemicube is precomputed analytically. All the patches of the scene are scan converted with visibility resolution onto the five faces of the hemicube. $F_{dS_i \to S_j}$ is finally computed by summing up the ΔF_q-s of all the pixels on the hemicube that are occupied by the patch S_j.

This method can take advantage of hardware Z-buffer rendering to speed up the form factor computation. Z-buffering allows for handling the visibility of patches.

The patch-to-patch form factor $F_{i \to j}$ is either approximated to be equal to $F_{dS_i \to j}$ computed from the center of the patch S_i or approximated as an average of the $F_{dS_i \to j}$-s at the vertices of the patch S_i.

ΔF_q-s are precomputed analytically using the equations given below.

For a pixel q on the top face of the hemicube, the following is the equation for ΔF_q:

$$\Delta F_q = \frac{\cos \phi_i \cos \phi_j}{\pi r_{ij}^2} \Delta A_{top} = \frac{1}{\pi (1 + x^2 + y^2)^2} \Delta A_{top} \tag{16}$$

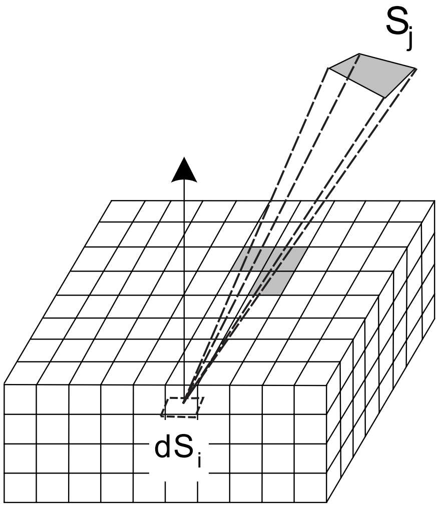


Figure 7. Hemicube.

In this equation, ΔA_{top} is the area of the pixel. Given the local coordinate system setup as shown in Fig. 8, the coordinate of the center of the pixel is $(x,y,1)$, and we also have $\cos\phi_i = \cos\phi_j = 1/r_{ij}$ and $r_{ij} = \sqrt{1+x^2+y^2}$.

For a pixel on the left side face of the hemicube, its ΔF_q is computed as follows:

$$\Delta F_q = \frac{1}{\pi(1+y^2+z^2)^2}\Delta A_{side} \tag{17}$$

where $(-1, y, z)$ is the coordinate of the center of the pixel and ΔA_{side} is the area of the pixel. Equations for the pixels on the other side faces are similar to Equation (17), except for the denominator, which depends on the coordinate of the pixel.

EXTENSIONS TO CLASSIC RADIOSITY

Classic radiosity methods assume that the radiosity is uniform over the area of the surface. This condition is rarely true over larger surfaces of the scene. So the very first step of the radiosity method is scene discretization. Regular discretization of scene surface causes light and shadow leakage at discontinuities due to shadow boundaries and touching objects. A discontinuity meshing technique (5) has been proposed to address this issue. This technique identifies discontinuities and splits the surfaces along the discontinuities first. The split surfaces are further discretized to smaller patches.

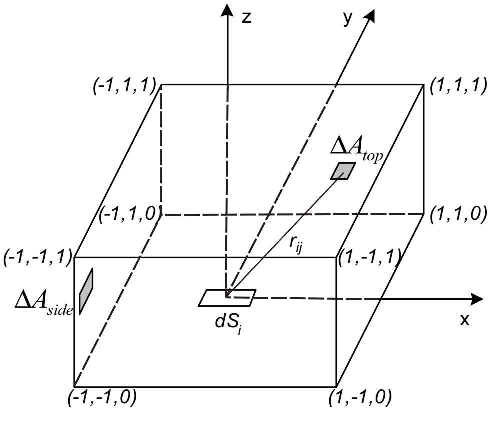


Figure 8. Coordinate system of ΔF_q.

The radiosity computed for patches of a surface gives discrete approximation of a smooth radiosity function over the surface. Direct rendering of such a function gives a faceted appearance to the surface. Hence, a smooth reconstruction of the radiosity function over the surface is desirable before it is used for display. A commonly used reconstruction method is to first compute the weighted average radiosity on vertices of the scene using radiosity of adjacent patches. Weights are the relative areas of the patches. The radiosity on any point on a patch is then interpolated from the vertices of the patch giving a smooth appearance to the scene surfaces.

The assumptions imposed in radiosity formulation often limit the application of the classic radiosity method. The assumptions are: Lambertian surface, uniform light distribution over the patch, planar patches, nonparticipating medium, and so on. There has been much research aimed at overcoming the limitations. Wavelet (6) and finite element methods (7) have been proposed to remove the uniform light distribution assumption and thus to reduce the amount of discretization. The hierarchical radiosity method (8,9) has been proposed to accelerate the computation by reducing the number of form factor computations with very little loss of accuracy. Extensions to support nondiffuse surfaces (10), textured surfaces (11), curved surfaces (12), nondiffuse light sources (13), participating media (14), furry surfaces (15) and fractals (16) have also been proposed.

The radiosity method will remain the primary physically based method for computing realistic lighting in synthetic scenes. More than 15 years of research has been devoted to solving problems related to this method. Findings of this research are being used regularly in many rendering software solutions developed for commercial and noncommercial use. Despite so many years of research efforts, the scene discretization and the extension to nondiffuse environments remain hard problems.

BIBLIOGRAPHY

1. C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, Modeling the interaction of light between diffuse surfaces, *Comput. Grap.*, **18** (3): 212–222, 1984.
2. M. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg, A progressive refinement approach to fast radiosity image generation, *Comput. Graph.*, **22** (4): 75–84, 1988.
3. T. Nishita and E. Nakamae, Continuous tone representation of three-dimensional objects taking account of shadows and interreflections, *Comput. Graph.*, **19** (3): 23–30, 1985.
4. M. Cohen and D. P. Greenberg, The hemi-cube: A radiosity solution for complex environments, *Comput. Graph.*, **19** (3): 31–40, 1985.

5. D. Lischinski, F. Tampieri, and D. P. Greenberg, Combining hierarchical radiosity and discontinuity meshing, *ACM SIGGRAPH '93 Proc.*, 1993, pp. 199–208.
6. F. Cuny, L. Alonso, and N. Holzschuch, A novel approach makes higher order wavelet really efficient for radiosity, *Comput. Graph.Forum (Proc. of Eurographics 2000)*, **19** (3): 99–108, 2000.
7. R. Troutman and N. L. Max, Radiosity algorithm using higher order finite element methods, *ACM SIGGRAPH 1993 Proc.*, 1993, pp. 209–212.
8. M. Cohen, D. P. Greenberg, D. S. Immel, and P. J. Brock, An efficient radiosity approach for realistic image synthesis, *IEEE Comput. Graph. Appl.*, **6** (3): 26–35, 1986.
9. P. Hanrahan, D. Salzman, and L. Upperle, A rapid hierarchical radiosity algorithm, *Comput. Graph.*, **25** (4): 197–206, 1991.
10. J. R. Wallace, M. F. Cohen, and D. P. Greenberg, A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods, *Comput. Graph.*, **21** (4): 311–320, 1987.
11. H. Chen and E. H. Wu, An efficient radiosity solution for bump texture generation, *Comput. Graph.*, **24** (4): 125–134, 1990.
12. H. Bao and Q. Peng, A progressive radiosity algorithm for scenes containing curved surfaces, *Comput. Graph.Forum* (Eurographics '93), **12** (3): C399–C408, 1993.
13. E. Languenou and P. Tellier, Including physical light sources and daylight in global illumination, *Third Eurographics Workshop on Rendering*, pp. 217–225, 1992.
14. H. E. Rushmeier and K. E. Torrance, The zonal method for calculating light intensities in the presence of a participating medium, *Comput. Graph.*, **21** (4): 293–302, 1987.
15. H. Chen and E. Wu, Radiosity for furry surfaces, *Proc. of EUROGRAPHICS'91*, in F. H. Post and W. Barth, (eds.) North-Holland: Elsevier Science Publishers B. V. 1991, pp. 447–457.
16. E. Wu, A radiosity solution for illumination of random fractal surfaces, *J. Visualization Comput. Animation*, **6** (4): 219–229, 1995.

FURTHER READING

I. Ashdown, *Radiosity: A Programmer's Perspective*, New York: John Wiley & Sons, Inc., 1994.

M. F. Cohen and J. R. Wallace, *Radiosity and Realistic Image Synthesis*, Boston, MA: Academic Press Professional, 1993.

F. X. Sillion, *Radiosity and Global Illumination*, San Francisco, CA: Morgan Kaufmann Publishers, 1994.

Ruifeng Xu
Sumanta N. Pattanaik
University of Central Florida
Orlando, Florida

R

RENDERING

INTRODUCTION

In the real world, light sources emit photons that normally travel in straight lines until they interact with a surface or a volume. When a photon encounters a surface, it may either be absorbed, reflected, or transmitted. Some of these photons may hit the retina of an observer where they are converted into a signal that is then processed by the brain, thus forming an image. Similarly, photons may be caught by the sensor of a camera. In either case, the image is a 2-D representation of the environment.

The formation of an image as a result of photons interacting with a 3-D environment may be simulated on the computer. The environment is then replaced by a 3-D geometric model and the interaction of light with this model is simulated with one of a large number of algorithms. The process of image synthesis by simulating light behavior is called *rendering*.

As long as the environment is not altered, the interaction of light and surfaces gives rise to a distribution of light in a scene that is in equilibrium (i.e., the environment does not get lighter or darker).

As all rendering algorithms model the same process, it is possible to summarize most rendering algorithms by a single equation, which is known as the *rendering equation*. The underlying principle is that each point x on a surface receives light from the environment. The light that falls on a point on a surface may be coming directly from a light source, or it may have been reflected one or more times by other surfaces.

Considering a point x on some surface that receives light from all directions, the material of the surface determines how much of this light is reflected, and in which directions. The reflective properties of a surface are also dependent on wavelength, which gives each surface its distinctive color. A material may therefore be modeled using a function that describes how much light incident on a point on a surface is reflected for each incoming and each outgoing direction. Such functions are generally known as *bidirectional reflectance distribution functions* (BRDFs), and are denoted here as $f_r(\mathrm{x}, \Theta_i, \Theta_o)$. This function is dependent on the position on the surface x, as well as the angle of incidence Θ_i and the outgoing direction Θ_o.

To determine how much light a surface reflects into a particular direction, we can multiply the BRDF for each angle of incidence with the amount of incident light $L_i(\mathrm{x}, \Theta_i)$ and integrate these pairwise multiplications, which yields a quantity for one specific outgoing direction.

A point on a surface may also emit light, which is denoted with a non-zero term $L_e(\mathrm{x}, \Theta_o)$. This term is dependent on position on the surface (e.g., a television screen emits light that is spatially varying in intensity), and may also be directionally varying (e.g., spot lights emit more light in some directions than in others).

Thus, the amount of light that leaves a point x on a surface in a particular direction Θ_o may be modeled as follows:

$$L_o(\mathrm{x}, \Theta_o) = L_e(\mathrm{x}, \Theta_o) + \int_{\Omega_i} g(\mathrm{x}, \Theta_i, \Theta_o) d\omega_i \tag{1}$$

$$g(\mathrm{x}, \Theta_i, \Theta_o) = f_r(\mathrm{x}, \Theta_i, \Theta_o) L_i(\mathrm{x}, \Theta_i) \cos \Theta_i \tag{2}$$

This equation is known as the *rendering equation*. To compute how much light is reflected into a particular direction, we need to integrate over all incident directions (a hemisphere of directions Ω_i if we assume that the surface is not transparent). Thus, the above equation will have to be recursively evaluated for each point in the environment that is visible from x.

To compute an image by simulating light in the above manner, we would have to evaluate the rendering equation for each pixel separately (multiple times if we were to apply *antialiasing* in the process). It should be clear that the number of computations required to evaluate this equation even once is astronomical. For practical problems, the computational cost of evaluating the rendering equation directly is too high. However, there are many ways to simplify this equation, for example, by removing parts of the computation that do not contribute significantly to the final solution.

It is, for instance, possible to only account for the direct contribution of light sources, and ignore all reflected light. Such algorithms fall in the class of *local illumination* algorithms. If indirect illumination (i.e., illumination after one or more reflections or transmissions) is accounted for, then we speak of *global illumination* algorithms.

Finally, the rendering equation is known as a Fredholm equation of the second kind, which implies that no analytical solutions are known. We, therefore, have to resort to numerical approximations to evaluate the rendering equation. In particular, this equation is routinely discretized, turning its evaluation into a sampling problem.

In summary, rendering involves the creation of images by simulating the behavior of light in artificial scenes. Such scenes consist of descriptions of surfaces and light sources (the *geometry*). In addition to having a position in space and a particular shape, surfaces are characterized by the manner in which they interact with light (*material properties*). In the following sections, geometry and materials are discussed in greater detail, followed by a brief explanation of the more prominent local and global illumination algorithms.

GEOMETRY

The shape of an object can be modeled with a collection of simple primitives, including polygon and triangle meshes, spline surfaces, and point-based representations.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

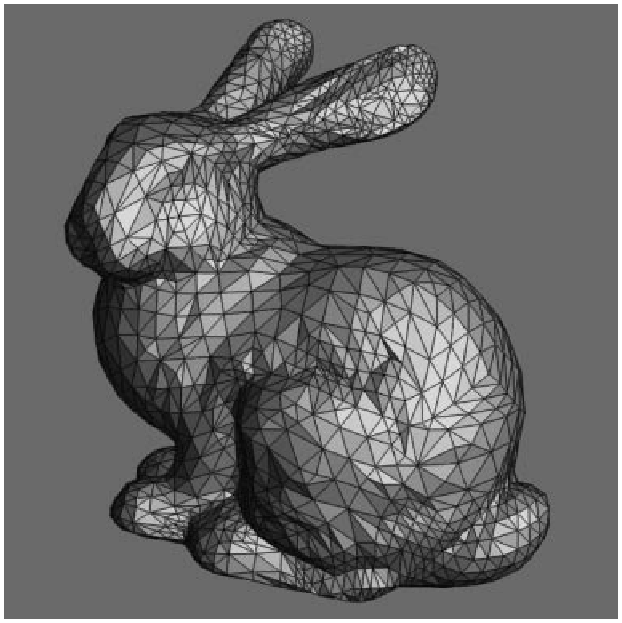


Figure 1. Mesh representation of a model of a bunny. Image courtesy of Hugues Hoppe (1).

Geometric representations can either be modeled by hand using modeling software, such as Alias Wavefront, or objects can be scanned with a laser scanner.

A frequently used representation is a mesh (Fig. 1). A mesh is made up of one or more simple polygonal shapes, for example, triangles. Some polygons share boundaries with other polygons in the mesh and together produce the structure of the object. Of course, polygons will only approximate the shape of the actual object. The larger the number of polygons used (and therefore the smaller their size), the closer the approximation will be to the actual shape of the object. The number of polygons also determines the time it takes to render the object, thereby affording a trade-off between quality and computation time.

For efficiency purposes, large meshes may be reduced in size. One technique to reduce the number of polygons representing an object's surface is *displacement mapping*. In this technique, the surface of the object is represented by fewer, larger polygons, and the small-scale features are captured in a depth map. Points on the surface, which is represented by polygons, are then displaced according to the displacement map. In this way, the object shape retains fine features, but the number of polygons used to represent the object is smaller.

Another way of reducing rendering time is to use *level of detail* algorithms. These algorithms ensure that the object is represented by as many primitives as necessary, dependent on distance to the viewer. If the object is far from the viewpoint, most of the fine details will not be visible, and thus the object's shape may be represented by fewer polygons. If the viewpoint approaches the object, the object needs to be represented by a larger number of polygons so that the fine-scale features, which are now visible, are adequately visualized.

The shape of an object may also be described by parametric equations such as Bézier curves and B-splines. The parametric surface has certain advantages over the simpler polygonal model. First, the representation is much more concise. If an object has fine features, the mesh will require many polygons to represent the object. However, patches of the same surface, when represented paramet-rically, will be fewer, which is because each patch can represent a curved surface segment, whereas triangles and polygons are flat.

Scanners can be used to determine the shape of existing objects. The output from a scanner is a dense set of points. Typically-these points define the vertices of a triangle mesh. Relatively recently, algorithms have been developed to render point clouds directly, obviating the need for triangulation. This approach also lends itself to a simpler level of detail algorithms because altering the number of points is more straightforward than altering the number, size, and shape of polygons or patches representing the object shape.

MATERIALS

The micro-structure of the object determines the way light interacts with it, and hence it determines the appearance of the object. This micro-structure is represented by a material description, such as the BRDF f_r introduced in the Introduction.

If a surface scatters light equally in all directions, we call the material *diffuse* or *Lambertian*, leading to a BRDF that is a constant function (i.e., $f_r = \rho/\pi$) where ρ is a measure of how much light is reflected. Other materials may reflect light more in some directions than others, which is a function of the direction of incidence. For instance, a mirror reflects almost all light in the reflected direction. In between lie glossy materials that scatter light into a cone centered around the direction of mirror reflection.

The angles Θ_i and Θ_o can each be decomposed into an elevation angle ϕ and an azimuthal angle θ in the plane of the surface, for instance, $\Theta_i = (\phi_i, \theta_i)$. If the material's reflective properties depend only on ϕ_i, ϕ_o, and $\theta_i - \theta_o$, then reflections are invariant to rotation around the surface normal, and the material is called isotropic. On the other hand, if f_r depends on $\theta_i, \theta_o, \phi_i$, and ϕ_o independently, then rotation around the surface normal will alter the reflection, and the material is called anisotropic (brushed aluminium is an example).

Real materials can be measured, or BRDFs may be modeled empirically. In the latter case, reciprocity and conservation of energy are considered important features of any plausible BRDF. Reciprocity refers to the fact that f_r should return the same result if Θ_i and Θ_o are reversed. Conservation of energy means that light is either reflected or absorbed, but not lost in any other way.

Extensions to basic BRDFs include models for transparency (e.g., glass), translucency, and spatial variance. Translucency stems from light scatter inside a surface, as shown in Fig. 2. Wax, skin, fruit, and milk all display some degree of translucency. An example of a spatially varying material is woodgrain, which is normally modeled using texture mapping. A texture map can be created by taking a photograph of the desired material, and then

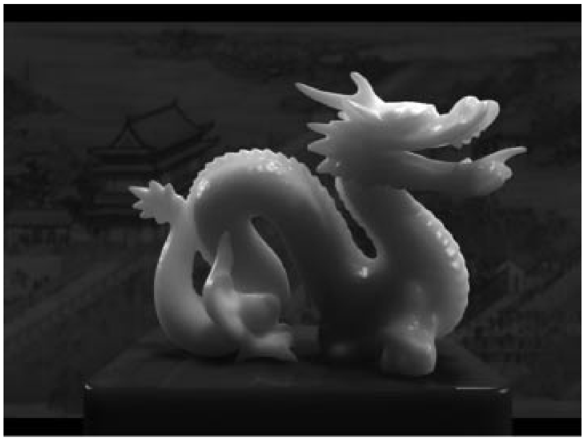


Figure 2. An example of an object rendered with a translucent material. Image courtesy of Rui Wang, university of Massachusetts, Amherst.

mapping it onto the surface of the object. Texture maps and BRDFs may be combined to yield spatially variant BRDFs, or *bidirectional texture functions* (BTFs).

LOCAL ILLUMINATION

Images may be rendered by projecting all the geometry onto a plane that represents the screen in 3-D space, thus implementing a local illumination model. For each pixel, the nearest object may be tracked using a *z-buffer*. This buffer stores for each pixel the distance between the view point and the currently nearest object. When a new object is projected, its distance is tested against the distances stored in the z-buffer. If the new object is closer, it is drawn and the z-buffer is updated. The color assigned to the pixel is then derived from the object's color using a simple shading algorithm.

The simplicity of projective algorithms makes them amenable to hardware implementation. As a result, most graphics cards implement a graphics pipeline based on z-buffering. To maximize performance, geometry is typically limited to simple shapes such as triangle and polygonal meshes. Only simple materials are supported.

However, modern graphics cards incorporate two programmable stages that allow vertices and pixels to be manipulated respectively, providing flexibility in an otherwise rigid hardware environment. Programming these two stages is achieved through APIs such as *OpenGL* or *DirectX*. The (limited) ability to program graphics cards has given rise to many extensions to the basic z-buffer algorithm, such as *shadow maps*, which compute shadows.

RAY TRACING AND RAY CASTING

One of the basic operations in rendering is to compute which (part of an) object is visible from a given point in space and a given direction. Such sampling of the scene is often accomplished with a technique called *ray casting*. A ray is a half-line starting at a specified point in space (its *origin*) and aimed at a particular *direction*. There may be many objects located along the line of sight of such a ray, and to compute which object is closest to the ray origin, the ray is intersected with each object. The point on the surface of the nearest object where the ray intersects, is called the *intersection point*. Functions for ray intersection calculations are available for a wide variety of geometric primitives, including triangles, polygons, implicit surfaces, and splines—a distinct advantage of any ray casting-based algorithm.

An image of a scene may be created by specifying a camera position, and casting (primary) rays starting at this position into different directions associated with the pixels that make up the image. This process computes for each pixel the nearest object. The color of the nearest object is then assigned to its corresponding pixel. Such a ray caster may be extended to a full ray tracer by also shooting secondary rays. These rays start at the intersection points of the primary rays and are aimed into specific directions based on which type of lighting effect is desired.

For instance, rays may be traced from an intersection point toward the light sources. Such shadow rays are useful for computing shadows, because the shading of the intersection point can be adjusted based on whether it was in shadow or not.

If an intersection point belongs to an object with a specular material, an additional ray may be shot into the reflected direction. This direction is computed by mirroring the incident ray into the *surface normal*, a vector that specifies the surface orientation at the intersection point. The reflected ray is then recursively traced and its returned color is assigned to the intersection point, which is in turn used to color a pixel. The same procedure is followed for transmitted rays in the case of transparent objects. A typical ray tracing example is shown in Fig. 3.

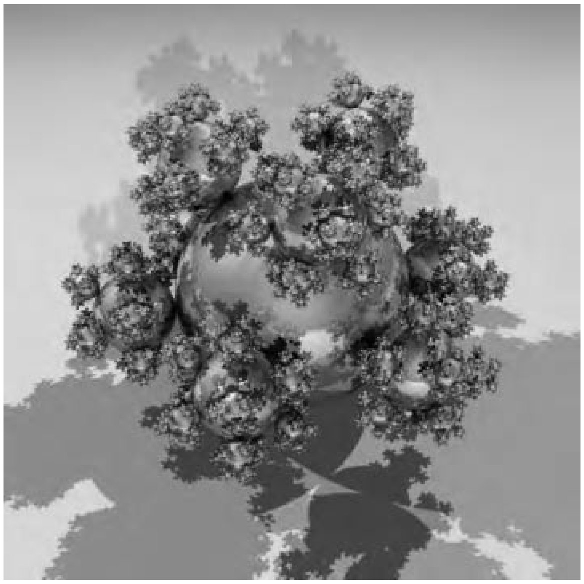


Figure 3. A typical ray traced image, consisting of reflective spheres and sharp shadows. Image courtesy of Eric Haines. The model is part of the Standard Procedural Database, as set of models for testing rendering algorithms; see *http://www.acm.org/tog/resources/SPD/*.

Thus, ray tracing is a recursive algorithm based on casting rays. Starting from the view point, it is called *eye ray tracing*. It is a relatively straightforward way to evaluate a simplified version of the rendering equation [Equation (1)], known as the ray tracing equation:

$$\begin{aligned} L_o(\mathrm{x},\Theta_o) = {} & L_e(\mathrm{x},\Theta_o) \\ & + \sum_{L} \int_{x_i \in L} v(\mathrm{x},\mathrm{x}_i)\, f_{r,d}(\mathrm{x}) L_e(\mathrm{x}_i,\Theta_i) \cos\Theta_i d\omega_i \\ & + \int_{\Theta_s \in \Omega_s} f_{r,s}(\mathrm{x},\Theta_s,\Theta_o) L(\mathrm{x}_s,\Theta_s) \cos\Theta_s d\omega_s \\ & + \rho_d(\mathrm{x}) L_a(\mathrm{x}) \end{aligned}$$

The four terms on the right-hand side are the emission term, followed by a summation of samples shot toward the light sources. The visibility term $v(\mathrm{x},\mathrm{x}_i)$ is 1 if position x_i on the light source is visible from point x and 0 otherwise. The integration in the second term is over all possible positions on each light source. The third term accounts for specular reflections, and the fourth term is the *ambient term*, which is added to account for everything that is not sampled directly.

Thus, in ray tracing, only the most important directions are sampled, namely the contributions of the light sources and mirror reflections, which represents a vast reduction in computational complexity over a full evaluation of the rendering equation, albeit at the cost of a modest loss of visual quality. Finally, ray tracing algorithms can now run at interactive rates and, under limited circumstances, even in real-time.

RADIOSITY

Both ray tracing and the local illumination models discussed earlier are view-point-dependent techniques. Thus, for each frame, all illumination will be recomputed, which is desirable for view-point-dependent effects, such as specular reflection. However, diffuse reflection does not visibly alter if a different viewpoint is chosen. It is therefore possible to preprocess the environment to compute the light interaction between diffuse surfaces, which may be achieved by employing a radiosity algorithm (Fig. 4). The result can then be used to create an image, for instance, by ray tracing or with a projective algorithm.

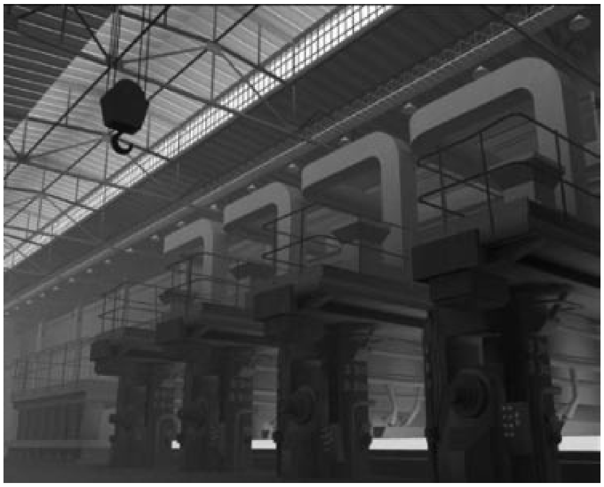


Figure 4. An early example of a scene preprocessed by a radiosity algorithm. Image courtesy of Michael Cohen (2).

The surfaces in the environment are first subdivided into small patches. For computational efficiency, it is normally assumed that the light distribution over each patch is constant. For small enough patches, this assumption is fair. Each patch can receive light from other patches and then diffusely reflect it. A common way to implement radiosity is to select the patch with the most energy and distribute this energy over all other patches, which then gain some energy. This process is repeated until convergence is reached. As all patches are assumed to be diffuse reflectors, the rendering equation [Equation (1)] can be simplified to not include the dependence on outgoing direction Θ_o:

$$L(\mathrm{x}) = L_e(\mathrm{x}) + \rho_d(\mathrm{x}) \int_{\mathrm{x}} L(\mathrm{x}') \frac{\cos\Theta_i \cos\Theta_o'}{\pi \|\mathrm{x}' - \mathrm{x}\|^2} v(\mathrm{x},\mathrm{x}') dA'$$

where $v(\mathrm{x},\mathrm{x}')$ is the visibility term, as before. This equation models light interaction between pairs of points in the environment. As radiosity operates on uniform patches rather than points, this equation can be rewritten to include a *form factor*F^{ij}, which approximates the fraction of energy leaving one patch and reaching another patch:

$$L^i = L_e^i + \rho_d^i \sum_j L^j F^{ij}$$

$$F^{ij} = \frac{1}{A^i} \int_{A^i} \int_{A^j} \frac{\cos\Theta_i \cos\Theta_j}{\pi r^2} \delta^{ij} dA^j dA^i$$

where the visibility term v between points is replaced with δ^{ij}, which denotes the visibility between patches i and j. The form factor depends on the distance between the two patches, as well as their spatial orientation with respect to one another (Fig. 5). In practice, the computation of form factors is achieved by ray casting.

The radiosity algorithm can be used to model diffuse inter-reflection, which accounts for visual effects such as

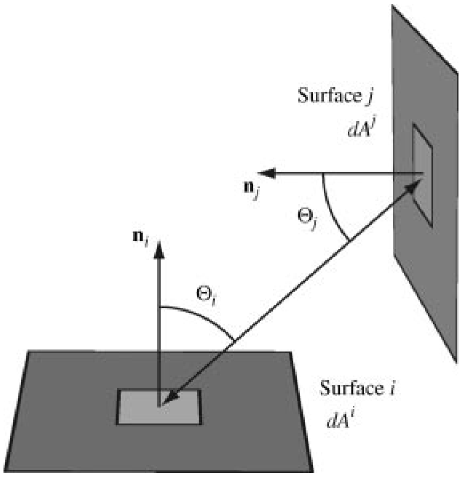


Figure 5. Geometric relationship between two patches.

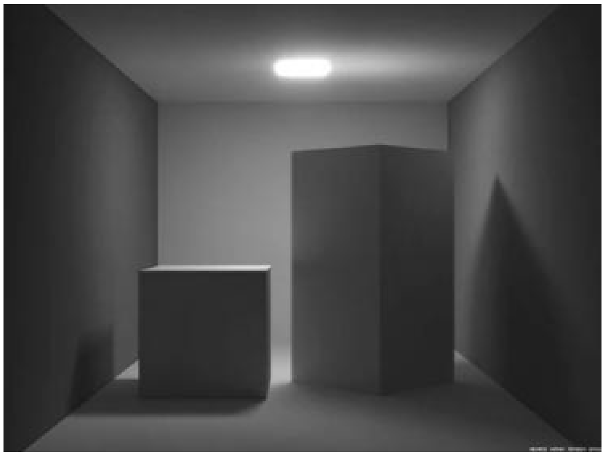


Figure 6. Example of color bleeding. In particular, the gray object on the right has a tinge of yellow and blue, caused by light that was first reflected off the yellow and blue surfaces. Image courtesy of Henrik Wann Jensen.

color bleeding (the colored glow that a surface takes on when near a bright surface of a different color, as shown in Fig. 6).

MONTE CARLO SAMPLING

The integral in the rendering equation [Equation (1)] may be evaluated numerically. However, as both the domain Ω_i and the integrand [Equation (1)] are complex functions, a very large number of samples would be required to obtain an accurate estimate. To make this sampling process more efficient, a stochastic process called Monte Carlo sampling may be employed. The environment is then sampled randomly according to *a probability density function* (pdf) $p(\omega_i)$:

$$\int_{\Omega_i} g(\omega_i, \Theta_i, \Theta_o) d\omega_i \approx \frac{1}{N} \sum_{i=1}^{N} \frac{g(\omega_i, \Theta_i, \Theta_o)}{p(\omega_i)}$$

The number of sample points N can be set to trade speed for accuracy. Typically, to evaluate each $g(\omega_i, \Theta_i, \Theta_o)$, a ray is traced into the environment. For efficiency, the pdf should be chosen to follow the general shape of the integrand $g(\omega_i, \Theta_i, \Theta_o)$. There are many ways to choose the pdf, a process known as *importance sampling*.

In addition, it is possible to split the integral into disjunct parts for which a simpler pdf may be known. This process is called *stratified sampling*. One could view ray tracing as a form of stratified sampling, because instead of sampling a full hemisphere around each intersection point, rays are only directed at the light sources and the reflected and transmitted directions. Both importance sampling and stratified sampling will help reduce the number of samples N required for an accurate evaluation of the rendering equation [Equation (1)].

PHOTON MAPPING

Certain complex types of illumination such as the caustic patterns created by light refracted through transparent objects are not efficiently sampled by Monte Carlo sampling alone. Rendering algorithms, such as ray tracing, radiosity, as well as local illumination models, expressly omit the sampling that would be required to capture caustics.

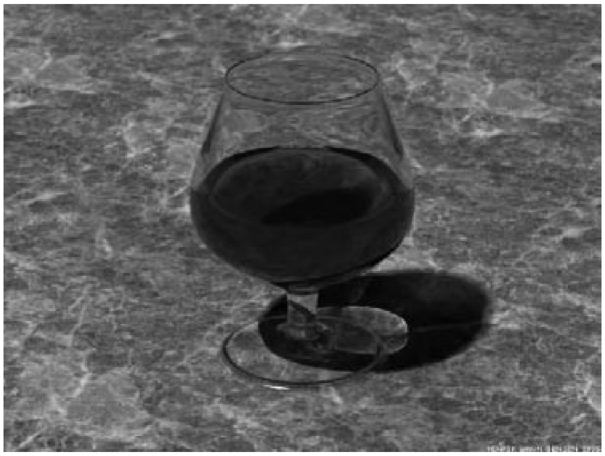


Figure 7. Light refracted through the transparent glass creates a caustic on the table. Image courtesy of Henrik Wann Jensen.

To enable the rendering of caustics, as shown in Fig. 7, as well as make rendering of other light interactions such as diffuse inter-reflection more efficient, photons may be tracked starting at the light source (known as *photon ray tracing*), rather than tracing photons backwards starting at the viewpoint (as in eye ray tracing). They can then be deposited on diffuse surfaces after having undergone one or more refractions through dielectric (transparent) objects. Thus, photons are stored in a data structure called *a photon map*, which represents the distribution of light over the surfaces in an environment.

An image may then be created using conventional ray tracing. Whenever an intersection with a diffuse surface is detected, the photon map is used to determine how much light is present at the intersection point. The photon map may therefore be seen as a data-structure to connect the initial light pass with the subsequent rendering pass.

Regarding efficiency, photon maps need to be created only once as long as the scene is static. The rendering pass can be repeated for any desired viewpoint.

IMAGE-BASED RENDERING

To avoid the expense of modeling a complicated scene, it is sometimes more convenient to photograph a scene from different viewpoints. To create images for novel viewpoints that were not photographed, an interpolation scheme may be applied. Rendering using images as a modeling primitive is called *image-based rendering*. Such techniques attempt to compute a continuous representation of the *plenoptic* function, given some discrete representation of it. The plenoptic function is defined as the intensity of light rays passing through the camera center at every camera location (V_x, V_y, V_z), orientation (θ, ϕ), and for every wavelength (λ) and time (t), that is:

$$P_7 = P(V_x, V_y, V_z, \theta, \phi, \lambda, t) \tag{3}$$

Thus, the plenoptic function may be considered a representation of the scene, such that, when input parameters like camera location and orientation are altered, the scene represented by the function changes accordingly. Simplified versions of the plenoptic function exist. For instance, if we assume that the environment is constant, we may remove the parameter t. The simplest plenoptic function is a 2-D panoramic view of the scene with a fixed viewpoint. θ and ϕ are the only two input parameters in this case.

If instead of a full panoramic view, we captured several images that are a part of this panoramic view, then these images would be a discrete representation of the plenoptic function. Image-based rendering techniques take these discrete representations as input and provide a continuous representation, for example, the complete panoramic view in the above case. A technique might take two images with different viewpoints as input and produce a set of images that have viewpoints that lie in between the two original viewpoints.

There are many image-based rendering techniques, and they may be broadly classified into three groups. The first group requires complete information of scene geometry, for example, in the form of a depth map of the scene. This information along with one or more images is sufficient to render scenes from a viewpoint close to the viewpoint of the given image(s). 3-D warping techniques belong to this category.

The second group of image-based rendering techniques uses only input images of the scene to render another image of the same scene from a different viewpoint. There is no reliance on any given information of the scene geometry. Examples include *light field rendering* and *lumigraph* systems.

The third group lies somewhere in between the previous groups. This group requires several input images as well as further geometric information in the form of correspondence features in the two images (for example, points). Given this correspondence, the scene may be rendered from all viewpoints between the two viewpoints of the original input images. *View morphing* (Fig. 8) and *interpolation* techniques fall under this category.

Images may also be used to represent the lighting of the scene alone, whereas geometry and materials are represented directly. This process is called *image-based lighting* (IBL, see Fig. 9). Here, the first step is to create the image that will represent the lighting of the scene. An image of a mirrored ball placed in the scene may be used to represent this lighting. Images typically have a limited range of pixel values (0 to 255), which cannot represent the lighting of an arbitrary scene. *High dynamic range* (HDR) images are used instead as their pixel values are not limited to 256 values and are proportional to the actual illumination of the scene. The captured image is then mapped to a sphere and the object is placed within it before rendering.

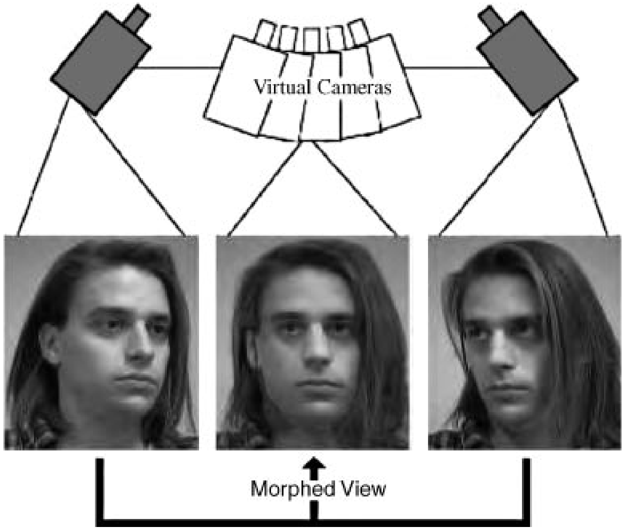


Figure 8. Two images are used to produce a third using image-based rendering. Image courtesy of Steven Seitz (3).

FURTHER READING

Rendering is an important part of the field of computer graphics. There are many excellent books, as well as a vast number of papers. Examples of general graphics books are in Refs. (5–9). References (10–13) are books specifically for ray tracing. Global illumination is covered in Ref. 14. Radiosity is explained in detail in Refs. 15–17. Photon mapping is described in Ref. 18. For local illumination models as well as using the OpenGL API, see Ref. 19. Image-based lighting is a relatively new rendering technique, described in Ref. 20. For real-time rendering, see Ref. 21. Parallel rendering is covered in Ref. 22. The notation used for the equations in this article are based on Arjan Kok's thesis (23).

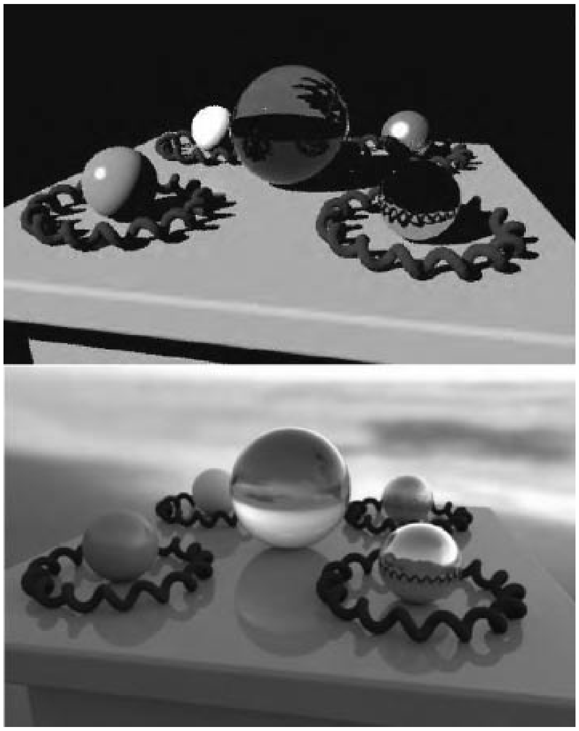


Figure 9. Scene rendering without image based lighting (top), and with image-based lighting (bottom). Image courtesy of Paul Debevec (4).

The latest research on rendering is published in a variety of forums. The most relevant conferences are *ACM SIGGRAPH*, the *Eurographics Symposium on Rendering*, and the *Eurographics main* conference. In addition, several journals publish rendering papers, such as *ACM Transactions on Graphics*, *IEEE Transactions on Visualization and Computer Graphics*, *Eurographics Forum*, and the *Journal of Graphics Tools*.

ACKNOWLEDGMENTS

We thank Hugues Hoppe, Rui Wang, Eric Haines, Michael Cohen, Henrick Wann Jersen, Steven Seitz, and Paul Debevec for kindly allowing us to reproduce some of their images.

BIBLIOGRAPHY

1. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95*,1995,173–182.
2. M. F. Cohen, S. Chen, J. R. Wallace, and D. P. Greenberg, A progressive refinement approach to fast radiosity image generation. In *SIGGRAPH '88*, 1988, 74–84.
3. S. M. Seitz and C.R. Dyer, View morphing. In *SIGGRAPH '96*, 1996, 21–30.
4. P. Debevec, E. Reinhard, G. Ward, and S. Pattanaik, High dynamic range imaging. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, 2004.
5. P. Shirley, M. Ashikhmin, S. R. Marschner, E. Reinhard, K. Sung, W. B. Thompson, and P. Willemsen, *Fundamentals of Computer Graphics*, 2nd ed. Natick, MA: A.K. Peters, 2005.
6. M. Pharr and G. Humphreys, *Physically Based Rendering*, San Fransisco, CA: Morgan Kaufmann, 2004.
7. A. S. Glassner, *Principles of Digital Image Synthesis*, San Fransisco, CA: Morgan Kaufmann, 1995.
8. J. Foley, A. Van Dam, S. Feiner, and J. Hughes, *Computer Graphics, Principles and Practice*, 2nd ed. Reading Addison-Wesley, 1990.
9. A. Watt and M. Watt, *Advanced Animation and Rendering Techniques, Theory and Practice*, Wokingham, UK: Addison-Wesley, 1992.
10. A. S. Glassner, ed. *An Introduction to Ray Tracing*, San Diego, CA: Academic Press, 1989.
11. G. Ward Larson and R. A. Shakespeare, *Rendering with Radiance*, San Francisco, CA: Morgan Kaufmann, 1998.
12. P. Shirley and R. K. Morley, *Realistic Ray Tracing*, 2nd ed.Natick, MA: A.K. Peters, 2003.
13. K. Suffern, *Ray Tracing from the Ground Up*, Natick, MA: A.K. Peters, 2007.
14. P. Dutré, P. Bekaert, and K. Bala, *Advanced Global Illumination*, Natick, MA: A.K. Peters, 2003.
15. M. F. Cohen and J. R. Wallace, *Radiosity and Realistic Image Synthesis*, Cambridge, MA: Academic Press, 1993.
16. F. X. Sillion and C. Puech, *Radiosity and Global Illumination*, San Francisco, CA: Morgan Kaufmann, 1994.
17. I. Ashdown, *Radiosity: A Programmer's Perspective*, New York: John Wiley & Sons, 1994.
18. H. W. Jensen, *Realistic Image Synthesis using Photon Mapping*, Natick, MA: A.K. Peters, 2001.
19. D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 3rd ed.Upper Sadle River, NJ: Pearson Prentice Hall, 2004.
20. E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec, *High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting*, San Francisco, CA: Morgan Kaufmann, 2005.
21. T. Akenine-Möller and E. Haines, *Real-time Rendering*, 2nd ed. Natick, MA: A.K. Peters, 2002.
22. A. Chalmers, T. Davis, and E. Reinhard, eds. *Practical Parallel Rendering*. Natick, MA: A.K. Peters, 2002.
23. A. J. F. Kok, *Ray Tracing and Radiosity Algorithms for Photorealistic Image Synthesis*, PhD thesis, Delft University of Technology, The Netherlands. Delft University Press, ISBN 90-6275-981-5.

Erik Reinhard
University of Bristol
Bristol, United Kingdom
Erum Khan
Ahmet Oğuz Akyüz
University of Central Florida
Orlando, Florida

S

SOLID MODELING

Solid modeling is the technique for representing and manipulating a complete description of physical objects. A complete description of a solid object is a representation of the object that is sufficient for answering geometric queries algorithmically. This requires a mathematical formulation that defines rigorously the characteristics of a solid object. Based on this mathematical formulation, different schemes for representing physical objects are developed.

MATHEMATICAL FORMULATION

A solid is described using the concept of point-set topology, whereas the boundary of a solid is characterized by using the concept of algebraic topology as discussed below. A solid is a closed point set in the three-dimensional Euclidean space E^3. For example, a unit cube is denoted as $S = \{\mathbf{p} : \mathbf{p} = (x,y,z),$ such that $x \in [0,1], y \in [0,1], z \in [0,1]\}$. A point set in E^3 denoting a solid is rigid and regular (1). A point set is rigid, which implies that it remains the same when being moved from one location to another. A point set is regular, which implies that the point set does not contain isolated points, edges, or faces with no material around them. To ensure a point set is regular, a regularization process is applied to a point set as defined below.

Definition 1. Regularization of a Point Set. Given a point set S, the regularization of S is defined as $r(S) = c(i(S))$, where $c(S)$ and $i(S)$ are, respectively, the closure and interior of S

The regularization process discards isolated parts of a point set, which is then enclosed with a tight boundary resulting in a regular point set. A point set S is regular if $r(S) = S$, and a regular set S is usually referred to as an r-set.

BOOLEAN OPERATIONS ON POINT SETS

Boolean operations on r-sets may result in a nonregular point set. For example, if $A = \{\mathbf{p} : \mathbf{p} = (x,y,z),$ such that $x \in [0,1], y \in [0,1], z \in [0,1]\}$ and $B = \{\mathbf{p} : \mathbf{p} = (x,y,z),$ such that $x \in [1,2], y \in [1,2], z \in [1,2]\}$, then,

$$A \cap B = \{\mathbf{p} : \mathbf{p} = (x,y,z), \text{ such that } x \in [0,1] \cap [1,2], y \in [0,1] \cap [1,2], z \in [0,1] \cap [1,2]\}$$

or,

$$A \cap B = \{\mathbf{p} : \mathbf{p} = (x,y,z), \text{ such that } x = 1, y = 1, z = 1\},$$

$A \cap B$ is thus a single point and is not regular. To ensure the result of Boolean operations on point sets are regular point sets, the concept of regularized Boolean operation is adopted. In the above example, $A \cap B = (1,1,1)$, the interior of $A \cap B$ is empty; i.e., $i(A \cap B) = \phi$. As the closure of $i(\phi)$ is empty, applying a regularization on $A \cap B$ gives $r(A \cap B) = c(i(A \cap B)) = \phi$. Based on the concept of regularization on point sets, regularized Boolean operation is defined as follows.

Definition 2. Regularized Boolean Operations. Denote $\cup^*$, $\cap^*$, and -* as the regularized union, intersect, and difference operations, respectively. They are defined as follows:

$$A \cup^* B = c(i(A \cup B) \tag{1a}$$

$$A \cap^* B = c(i(A \cap B) \tag{1b}$$

$$A -^* B = c(i(A - B) \tag{1c}$$

A point set describing a solid must be finite and well defined. The surfaces of a solid are thus restricted to be algebraic or analytic surfaces.

THE BOUNDARY OF A SOLID

The boundary of a solid is a collection of faces connected to form a closed skin of the solid. The concept of algebraic topology (2) is usually adopted for characterizing the boundary of a solid.

The surface composing the boundary of a solid is a 2-manifold in the two-dimensional space E^2. A 2-manifold is a topological space in which the neighborhood of a point on the surface is topologically equivalent to an open disk of E^2. However, there are cases when the boundary of a solid is not a 2-manifold as shown in Fig. 1. In addition, some 2-manifolds do not constitute a valid object in E^3 (e.g., the Klein bottle). To ensure the boundary of an object encloses a valid solid, the faces of the boundary must be orientable with no self-intersection. An object is invalid (or is a nonmanifold object) if its boundary does not satisfy the Euler–Poincare characteristic as described below.

THE EULER–POINCARE CHARACTERISTIC

The Euler–Poincare characteristic states that an object is a nonmanifold object if its boundary does not satisfy the equation

$$v - e + f = 2(s - h) + r$$

where v-number of vertices, e-number of edges, f-number of faces, s-number of shells, h-number of through holes, and r-number of rings.

The Euler–Poincare formula is a necessary but not a sufficient condition for a valid solid. An object with a boundary that does not satisfy the Euler–Poincare formula is an invalid solid. On the contrary, an object satisfying the Euler–Poincare formula may not be a valid solid. Figure 2

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

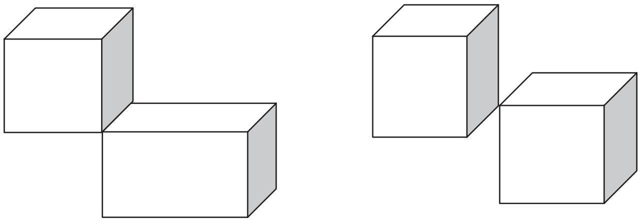


Figure 1. Nonmanifold object.

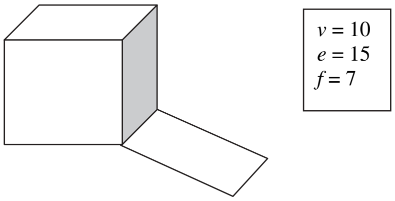


Figure 2. An invalid solid statisfies Euler characteristic.

illustrates an invalid solid satisfying the Euler–Poincare formula.

REPRESENTATION SCHEMES

Denote M as the modeling space of solids. That is, a solid X is an element of M. A solid representation (1) of an object is a collection of symbols specifying the solid object. There are various representation schemes for modeling solids defined as point sets in E^3. In general, the properties of a solid representation scheme can be described as follows:

Geometric coverage — the objects that can be described using the representation scheme.

Validity — a representation scheme must designate a valid solid (Fig. 3)

Completeness — a representation must provide enough data for any geometric computation. For example, given a point **p**, an algorithm must exist for deciding if **p** is *in, on,* or *out* of the solid.

Uniqueness — a representation scheme is unique if there is only one representation for a given solid object.

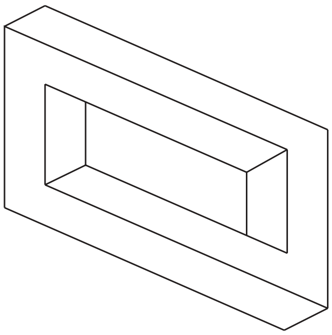


Figure 3. An invalid solid.

Unambiguous — a representation scheme is unambiguous if a representation designates exactly one solid object.

Conciseness — the space (computer storage) required for a valid representation.

Closure of operation — whether operations on solids preserve the validity of the representation.

Computational requirement and applicability — the algorithms that can be applied to the representation scheme and the complexity of these algorithms.

Among the various solid representation schemes, the constructive solid geometry and the boundary representation are the most popular schemes as discussed in the following.

CONSTRUCTIVE SOLID GEOMETRY (CSG)

Constructive solid models consider solids as point sets of E^3. The basic elements of CSG model are simple point sets that can be represented as simple half-spaces. Given a mapping $f : E^3 \rightarrow R$ that maps the Euclidean space to the real axis, the function $f(\mathbf{p})$, where $\mathbf{p}$ is a point in E^3, divides the three-dimensional space into two halves. They are the space $f(\mathbf{p}) > 0$ and its complement $f(\mathbf{p}) \leq 0$. More complex objects are obtained by combining half-spaces with Boolean operations. Figure 4 shows a cylinder constructed with three half-spaces. An object modeled with constructive solid geometry can be represented with a CSG tree. A CSG tree is a binary tree structure with half-spaces or primitive solids at the leaf nodes. Figure 5 shows a set of solid primitives commonly used in a CSG modeler. All internal nodes are Boolean operations or transformations. An example is shown in Figure 6, where an L-shaped block is constructed with a CSG modeler. Figure 7 gives an example for modeling a more complicated object using a CSG modeler.

POINT MEMBERSHIP CLASSIFICATION AND BOOLEAN OPERATIONS

A basic characteristic of a solid representation scheme is to be capable of providing sufficient information for deciding

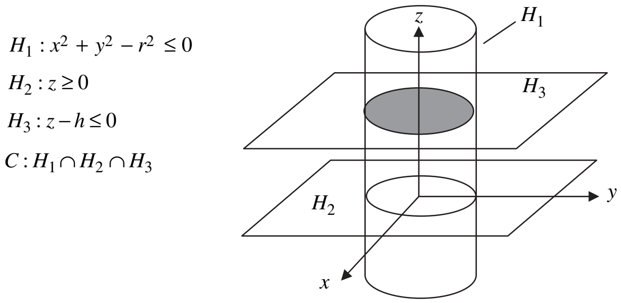


Figure 4. A cylinder constructed with three half-spaces.

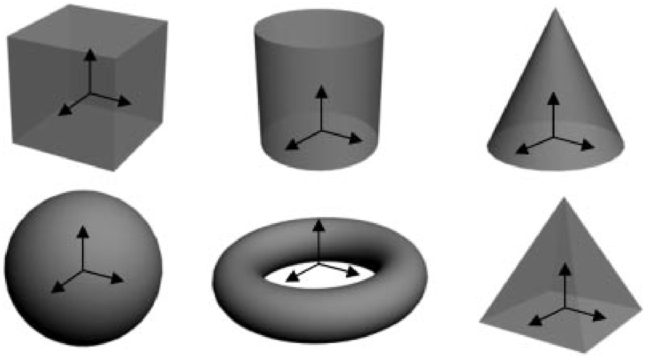


Figure 5. Commonly used solid primitives.

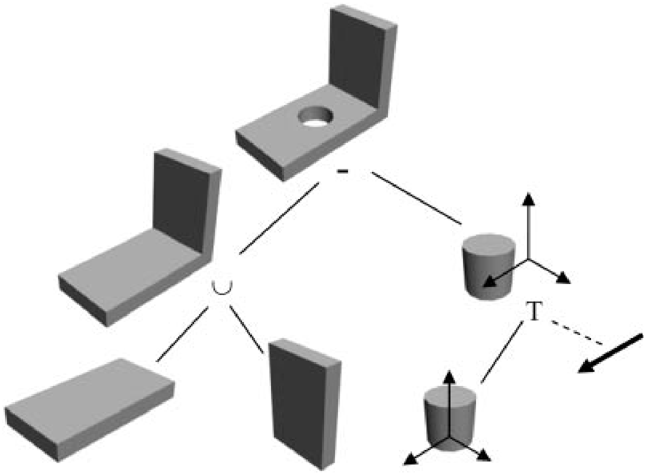


Figure 6. The CSG tree of an L-shaped block.

if a given point **p** is *in*, *on*, or *out* of a solid using a suitable point membership classification (PMC) (3,4) algorithm. Using a CSG representation scheme, point membership classification is performed with a divide-and-conquer approach. Starting from the root node of the CSG tree, the point **p** is passed down the tree. Whenever a Boolean operation node is encountered, **p** is passed to both the left and right subnodes of the current node. If a transformation node is encountered, an inverse transformation is applied to **p** and the result is passed to the left subnode. Whenever a leaf node is encountered, the point (possibly transformed) is classified against the half-space or primitive solid of the node. The result is propagated upward to the parent node. In this upward propagation, the results of the left and right subtree of a Boolean operation node are combined. The upward propagation ends at the root node where the result obtained will be the result of classifying p against the whole object.

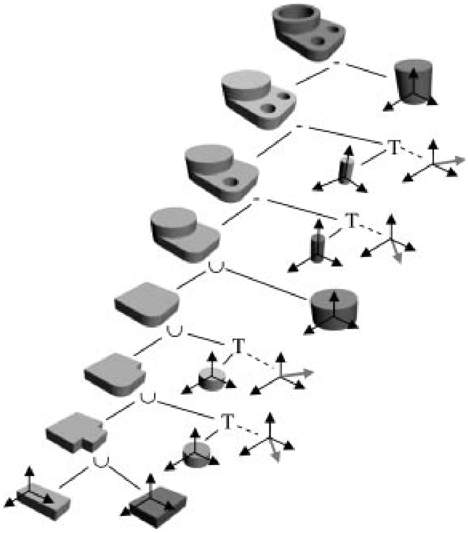


Figure 7. Object modeling with constructive solid geometry.

Combining the PMC results at a Boolean operation node requires special consideration. Assume R_L and R_R as the PMC results of the left and right subtrees, respectively. If $R_L \neq \text{on}$ or $R_R \neq \text{on}$, then the result of R_L *.op.* R_R (where $op \in \{\cup, \cap, -\}$) can be easily obtained according to Table 1. In all three cases, if a point is *on* both A and B, the classification result is undetermined. For instance, in Fig. 8, the point **p** is *on* both A and B. However, **p** may be *in* or *on* $A \cup B$ depending on the relative position of A and B. To obtain an accurate classification result, information regarding the local geometry of the solids in the vicinity of the point is required for the classification.

NEIGHBORHOOD

The Neighborhood $N(\mathbf{p}, S)$ of a point $\mathbf{p} = (x, y, z)$ with respect to a solid S is the intersection of an open ball with S, where the open ball is a sphere with an infinitesimal

Table 1. Classifications in Boolean operations

		B		
	$A \cup B$	*in*	*on*	*out*
	in	*in*	*in*	*in*
A	*on*	*in*	*?*	*in*
	out	*in*	*on*	*out*

		B		
	$A \cap B$	*in*	*on*	*out*
	in	*in*	*on*	*out*
A	*on*	*on*	*?*	*out*
	out	*out*	*out*	*out*

		B		
	A–B	*in*	*on*	*out*
	in	*out*	*on*	*in*
A	*on*	*out*	*?*	*on*
	out	*out*	*out*	*out*

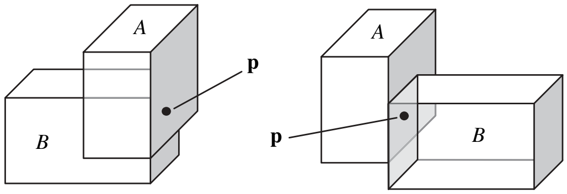


Figure 8. A point lying *on* both A and B may be *on* or *in* $A \cup B$.

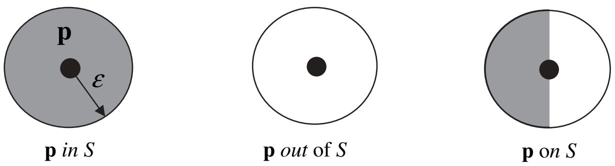


Figure 9. Three possible cases of the neighborhood at a point.

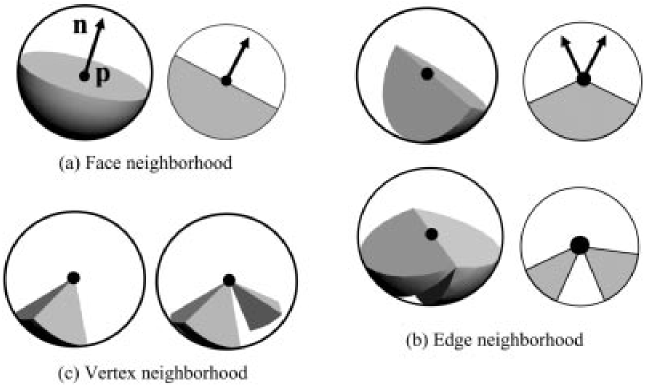
radius ε centered at **p**; i.e.,

$$N(\mathbf{p}, S) = S \cap \{(x\prime, y\prime, z\prime) | (x\prime - x)^2 + (y\prime - y)^2 + (z\prime - z)^2 < \varepsilon^2\} \qquad (2)$$

Using the notion of neighborhood, the following three cases can be established (Fig. 9):

1. A point **p** is *in* S iff N(**p**, S) is a full ball.
2. A point **p** is *out* of S iff N(**p**, S) is empty.
3. A point **p** is *on* S iff N(**p**, S) is not full and not empty.

If **p** lies in the interior of a face, its neighborhood can be represented by the equation of the surface and is oriented such that the face normal points to the exterior of S (Fig. 10a). If **p** lies in the interior of an edge, the neighborhood is represented by a set of sectors in a plane containing **p** that is perpendicular to the edge at **p** (Fig. 10b). A vertex neighborhood is inferred from the set of face normals of the faces incident at **p** (Fig. 10c). In a Boolean operation between two objects A and B, the neighborhoods of a point **p** relative to A and B are combined. Whether the combined neighborhood is full, empty, or not full nor empty determines whether **p** is *in, out*, or *on* the combined solid. In addition, by interpreting whether the neighborhood is a face neighborhood, edge neighborhood, or vertex neighborhood, the point **p** can be classified to be lying on a face, an edge, or is a vertex. For example, in a union node, if the neighborhood N_L, N_R of **p** relative to both the left and the right subtrees are face neighborhoods with face normals $\mathbf{n}_L$ and $\mathbf{n}_R$, respectively, then the union N of N_L and N_R is an edge neighborhood when $|\mathbf{n}_L \bullet \mathbf{n}_R| \neq 1$. If $\mathbf{n}_L \bullet \mathbf{n}_R = 1$, then N is a face neighborhood. If $\mathbf{n}_L \bullet \mathbf{n}_R = -1$, N is full (Fig. 11). If N_L and N_R are edge neighborhoods, the union of N_L and N_R may be a vertex neighborhood, an edge neighborhood, a face neighborhood, or it is full (Fig. 12).



(a) Face neighborhood

(b) Edge neighborhood

(c) Vertex neighborhood

Figure 10. Representation of a neighborhood.

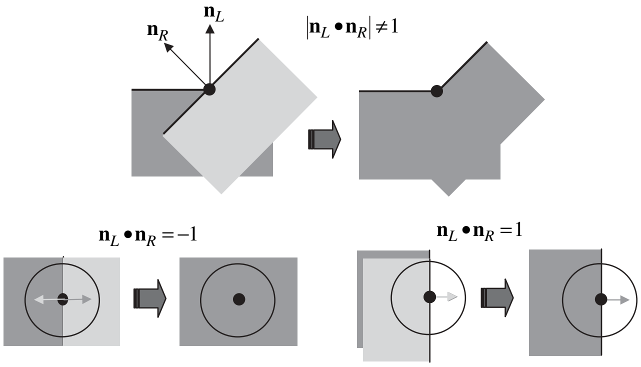

$\mathbf{n}_L$
$\mathbf{n}_R$
$|\mathbf{n}_L \bullet \mathbf{n}_R| \neq 1$
$\mathbf{n}_L \bullet \mathbf{n}_R = -1$
$\mathbf{n}_L \bullet \mathbf{n}_R = 1$

Figure 11. Combining face neighborhoods at a union node.

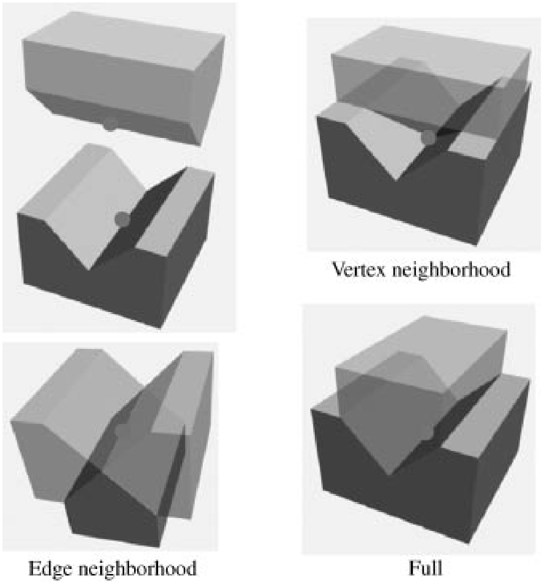


Figure 12. The union of edge neighborhoods.

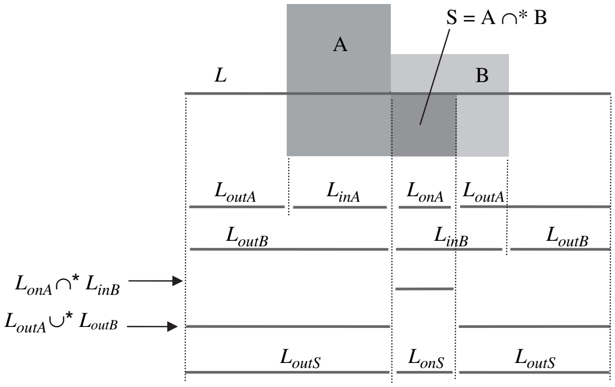


Figure 13. Edge-solid classification.

BOUNDARY EVALUATION

CSG representation provides a concise description of a solid. However, there is no explicit information for display purposes. To obtain this explicit information, a boundary evaluation process is required. This requires classifying all edges and faces of all primitive solids (or half-spaces) against the CSG representation of the solid. These are performed with the edge-solid classification and the face-solid classified algorithms.

In the edge-solid classification process, an edge L is partitioned into segments by intersecting L with all primitives of the solid. These segments are then classified against the given solid. A simple approach for classifying segments is to apply a point membership classification at the midpoint of each segment. Segments that are classified as lying *on* the solid are combined to give edges of the solid (Fig. 13).

In the face-solid classification process, a face is intersected with all primitives of the solid. The intersection edges and the boundary edges of the face are then classified against the given solid using the edge-solid classification algorithm. Edges lying on the solid are then connected to form boundaries of the face (Fig. 14).

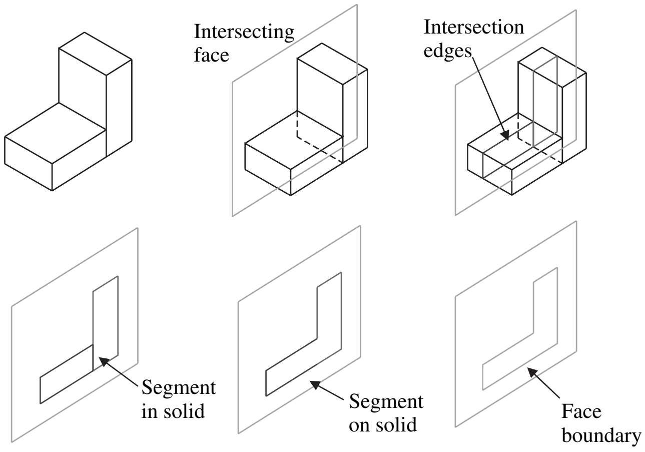


Figure 14. Face-solid classification.

PROPERTIES OF CSG MODELS

The geometric coverage of a CSG modeler depends on the type of primitive solids and half-spaces used. Provided all primitives are *r*-sets, and regularized set operations are adopted for the construction of more complex solids, CSG models are always valid solids. CSG models are unambiguous but not unique. A CSG model can be described precisely using a binary tree. CSG representations are thus relatively concise. Algorithms for manipulating CSG models (e.g., boundary evaluation) may be computationally expensive. With suitable algorithms, CSG models can be converted to other types of solid representations.

BOUNDARY REPRESENTATION

A boundary representation (B-Rep) of a solid describes a solid by its boundary information. Boundary information in a B-rep includes the geometric information and the topological relationship between entities of the boundary. Geometric information refers to the vertices, positions, surfaces, and curves equations of the bounding faces and edges. Topological relationship refers to the relationship among the faces, edges, and vertices so that they conform to a closed volume. For instance, the boundary of an object can be modeled as a collection of faces that forms the complete skin of the object. Each face is bounded by a set of edge loops, and each edge is, in general, bounded by two vertices. The boundary of an object can thus be described as a hierarchy of faces and edges as depicted in Fig. 15.

B-REP DATA STRUCTURE

Among the various data structures for implementing B-Rep, the most popular one is the winged-edge structure (5), which describes a manifold object with three tables of vertices, faces, and edges. In the winged-edge structure, each face is bounded by a set of disjoint edge loops or cycles. Each vertex is shared by a circularly ordered set of edges. For each edge, there are two vertices bounding the edge, which also defines the direction of the edge. There are two

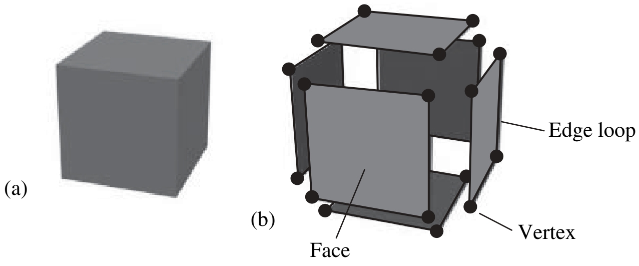


Figure 15. Boundary data: (a) A cube and (b) boundary data of a cube.

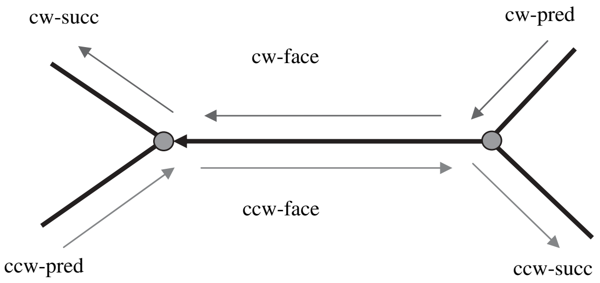


Figure 16. The winged-edge data structure.

faces, the left and the right adjacent faces, sharing an edge. There are two edge cycles sharing an edge. One edge cycle is referred to as in the clockwise direction, and the other is in the counterclockwise direction. The edge cycle on a face is always arranged in a clockwise direction as viewed from outside of the solid. In each direction, there is a preceding and a succeeding edge associated with the given edge. Figure 16 illustrates the relations among vertices, edges, and faces in a winged-edge structure. Table 2 gives an example of a winged-edge representation of the tetrahedron shown in Fig. 17.

VALIDITY OF B-REP SOLID AND EULER OPERATORS

Given a boundary representation specifying a valid solid, the set of faces of a boundary model forms the complete skin of the solid with no missing parts. Faces of the model do not intersect each other except at common vertices or edges. Face boundaries are simple polygons or closed contours that do not intersect themselves. To avoid the construction of invalid B-Rep solid, Euler operators (4,5) are usually employed in the object construction process. Euler operators keep track of the number of different topological entities in a model such that it satisfies the Euler–Poincare characteristic. Euler operators are functions defined as strings of symbols in the character set {M, K , S, J, V, E, F, S, H, R}. Each of the symbols M, K, S, and J denotes an operation on the topological entities V, E, F, S, H, and R. Descriptions of these symbols are listed below:

M—make, K—kill, S—split, J—join,
V—vertex, E—edge, F—face, S—solid, H—hole, R—ring.

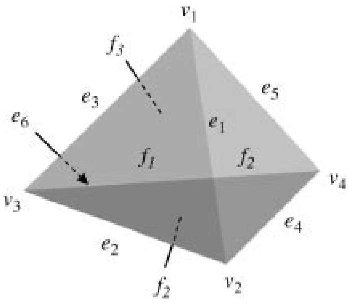


Figure 17. A tetrahedron.

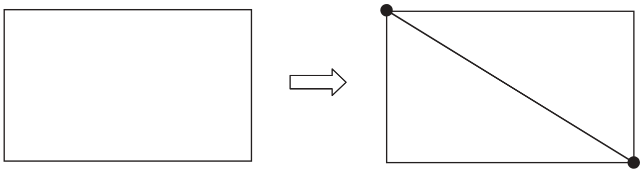


Figure 18. The MEF operator.

For example, an operator MEF denotes a function to "Make Edge and Face." As shown in Fig. 18, inserting an edge in the model also creates another face at the same time. In Fig. 19, the operator KEMR removes an edge while creating a ring in the object. The use of Euler operators always produces topologically valid boundary data. However, an object with a topologically valid boundary may be geometrically invalid. As shown in Fig. 20, moving the vertex **p** of a valid solid upward causes the interior of the object to be exposed and thus becomes an invalid object.

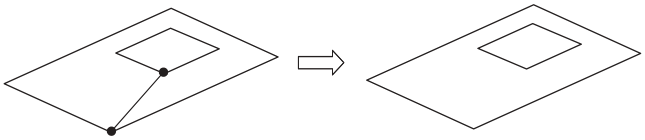


Figure 19. The KEMR operator.

Table 2. Winged-edge data.

	Vertices		Faces		CW		CCW	
Edge	Start	End	CW	CCW	Pred	Succ	Pred	Succ
e_1	v_1	v_2	f_1	f_2	e_3	e_2	e_4	e_5
e_2	v_2	v_3	f_1	f_4	e_1	e_3	e_6	e_4
e_3	v_3	v_1	f_1	f_3	e_2	e_1	e_5	e_6
e_4	v_1	v_4	f_2	f_3	e_5	e_1	e_2	e_6
e_5	v_4	v_2	f_2	f_4	e_1	e_4	e_6	e_3
e_6	v_4	v_3	f_4	f_3	e_3	e_5	e_4	e_2

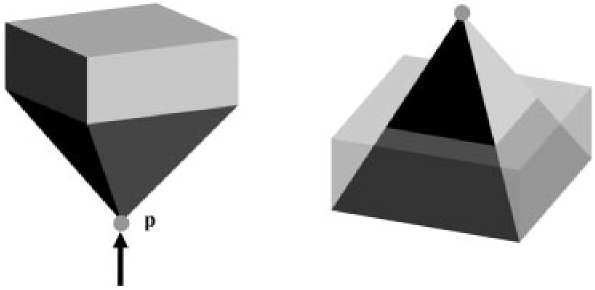


Figure 20. A topologically valid solid becomes geometrically invalid.

PROPERTIES OF B-REP

The geometric coverage of the boundary representation is determined by the types of surfaces being used. Early works in boundary representation are mainly confined to the manipulation of polyhedral solid models (6). Contemporary B-Rep modelers adopt trimmed NURBS surfaces for describing the geometry of an object boundary. This allows freeform solids to be modeled. B-Rep is unique but ambiguous; that is, one representation always refers to one solid. However, there may be more than one representation for one object. For example, a cube can be represented by 6 squares, or it can be represented by 12 triangles. As vertices, edges, and faces information are readily available, fast interactive display of B-Rep solid can be achieved. However, a complex data structure is required for maintaining a B-Rep. Nevertheless, B-Rep is attractive as it allows local modifications of objects. Local modification refers to a change in the geometry of an object. For instance, a change in the position of a vertex of a cube causes a plane of the cube to become a bilinear surface (Fig. 21).

HYBRID REPRESENTATION

As there is no single representation scheme that is superior, hybrid representation is usually adopted. In most popular hybrid representation schemes, objects are represented basically in B-Rep. Boolean operations are allowed for constructing solids. In some cases, sweep operations (7) are also adopted for constructing solids. A history of operations is recorded that is an extension of a CSG representation. The history of operations allows the modeled object to be modified by adjusting the location or shape parameters of the primitive solids. It also serves as the basis for implementing the “Undo” operation. As a B-Rep exists for each object, local modifications can be performed on the solid, whereas set operations may also be applied on objects. This provides a basis for the parametric or constraint-based modelers (8), which are widely adopted in commercial CAD systems.

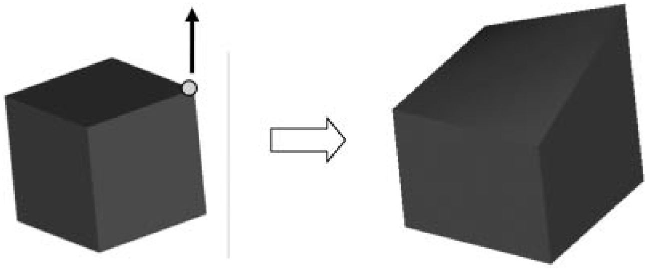


Figure 21. Local modification of a cube.

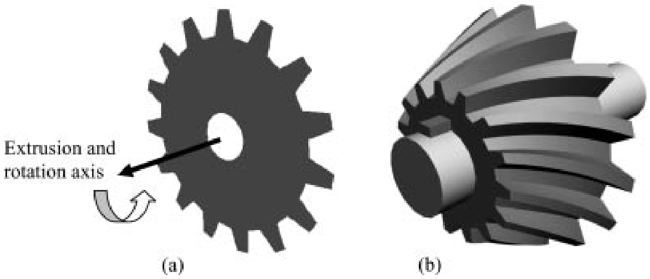


Figure 22. A bevel gear created with a sweep operation: (a) The gear contour, the extrusion, and rotation axis for the sweep. (b) Union of the swept solid, a cylinder, and a block.

OTHER REPRESENTATION SCHEMES

Besides the most popular CSG and B-Rep representation schemes, other representation schemes help to widen the geometric coverage or the scope of applications of solid modelers. Schemes that widen the geometric coverage of CSG modeler include the Sweep-CSG (7) and the Constructive Shell Representation (9). Sweep-CSG technique allows objects created with sweep operations to be used as primitives in a CSG modeler. An example is shown in Fig. 22 where a bevel gear is created by sweeping a planar contour as shown in Fig. 22a. The sweep operation is an extrusion of the contour, whereas the contour is being rotated and scaled. The Constructive Shell Representation adopts truncated tetrahedrons (Trunctet) as basic primitives. A trunctet is a tetrahedron truncated by a quadric surface. By maintaining continuity between trunctets, free-form solids can be modeled with CSG representation (Fig. 23).

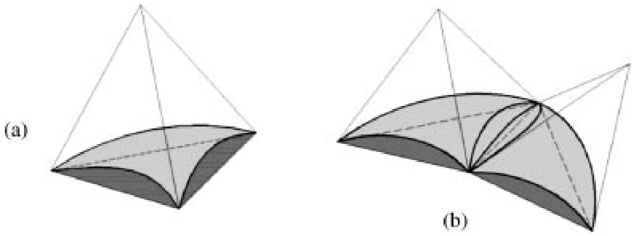


Figure 23. Trunctet and the Constructive Shell Representation: (a) A trunctet and (b) an object constructed with four smoothly connected trunctets.

Applications such as various stress and heat analysis using the finite element method requires an object to be represented as a collection of connected solid elements. This solid representation scheme is usually referred to as Cell Decomposition (10). Spatial Enumeration and Octree (10) are solid representation schemes widely used in applications where only a coarse representation of an object is required (e.g., collision detection). Although the most popular B-Rep and CSG schemes may not be suitable for these applications, algorithms exist for converting B-Rep and CSG models into other representation schemes, and hence, they remain a primary representation in existing solid modeling systems.

REMARKS

Solid modeling is a technique particularly suitable for modeling physical objects and is widely adopted in most commercial CAD systems. A product model built on top of a solid model provides the necessary information for the design analysis and production of the model. For instance, the rapid prototyping process requires an object to have a closed boundary for its processing. Manufacturing process planning of a product requires identifying the manufacturing features of the object. This can be obtained from the CSG tree of the object or the topological relationship between the entities in a B-Rep model. The bills of materials for a product require volumetric information of the product, which is readily available in the solid model of the product. A major limitation of solid modeling is its complexity in terms of storage and the algorithms required for its processing.

BIBLIOGRAPHY

1. A. G., Requicha Representations of solid objects—theory, methods, and systems, *ACM Comput. Surv.*, **12**(4): 437–464, 1980.
2. J., Mayer *Algebraic Topology*, Englewood Cliffs, NJ: Prentice-Hall, 1972.
3. B. R., Tilove Set membership classification: A unified approach to geometric intersection problems, *IEEE Trans. Comput.*, **29**(10): 874–883, 1980.
4. C. M., Hoffmann *Geometric and Solid Modeling: An Introduction*, San Francisco, CA: Morgan Kaufmann, 1989.
5. B., Baumgart A Polyhedron Representation for Computer Vision, *Proc. National Computer Conference*, 1975, 589–596.
6. I. C., Braid The synthesis of solids bounded by many faces, *Commun. ACM*, **18**(4): 209–216, 1975.
7. K. C., Hui S. T., Tan Display techniques and boundary evaluation of a sweep-CSG modeller, *Visual Comp.* **8**(1): 18–34, 1991.
8. J. J., Shah M., Mantyla *Parametric and Feature Based CAD/CAM*, New York: John Wiley & Sons, Inc., 1995.
9. J., Menon Guo Baining, Free-form modeling in bilateral brep and CSG representation schemes, *Internat. J. Computat. Geom. Appl.* **8**(5–6): 537–575, 1998.
10. M., Mantyla *An Introduction to Solid Modeling*, Rockville, MD: Computer Science Press, 1988.

K. C. Hui
The Chinese University
of Hong Kong
Shatin, Hong Kong

S

SURFACE DEFORMATION

Surface deformation is a fundamental tool for the construction and animation of three-dimensional (3-D) models. Deforming a surface is to adjust the shape of surface, which requires adjusting the parameters in the representation of a surface. As the defining parameters of different types of surfaces are different, different shape control techniques are available for different surface types. Alternatively, general deformation tools can be used for deforming surfaces. However, the deformation effect depends on how these tools are applied to the surfaces. In general, deformation tools can be classified into the geometric- and physics-based approaches. The geometry-based approach refers to the deformation of surfaces by adjusting geometric handles such as lattices and curves. The physics-based approach associates material properties with a surface such that the surface can be deformed in accordance with physical laws. They are discussed in detail in the following sections.

REPRESENTATION AND DEFORMATION OF SURFACE

The early work on the deformation technique developed by Alan Barr (1) uses a set of hierarchical transformations (stretching, bending, twisting, and tapering) for deforming an object. This technique determines the surface normal of a deformed object by applying a transformation to the surface normal vector of the undeformed surface. It requires a representation of the surface of which the Jacobian matrix can be determined. In fact the representation of a surface determines how a surface can be deformed. For surfaces that are represented in terms of geometric entities such as points and curves, deformation of the surface can be easily achieved through adjusting the locations of the points or modifying the shape of the curves. However, for surfaces where there is no direct geometric interpretation for the defining parameters, special techniques have to be employed to associate geometric entities with its representation. In order not to restrict the pattern of deformation on a surface, free-form surfaces or surfaces that can be used to model arbitrary object shapes are usually considered for surface deformation.

Deforming Algebraic Surfaces

An algebraic surfaces *f* is a function of the mapping *f*: $E^3 \rightarrow R$, such that all points **p** in E^3 satisfying *f*(**p**) = *k* lie on the surface, where *k* is a constant. Deforming an algebraic surface is to modify the function *f* such that a different set of points satisfies the modified function. As the defining parameters of an algebraic surface are the coefficients of the function *f*, they do not provide intuitive control over the shape of the surface. Alternative means are adopted for deforming algebraic surfaces. One approach to provide intuitive shape control on algebraic surface is to express an algebraic surface with a set of control points (2–4). Consider a tetrahedron T with vertices $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, $\mathbf{v}_4$; a point **p** lying within T can be expressed as

$$\mathbf{p} = s\mathbf{v}_1 + t\mathbf{v}_2 + u\mathbf{v}_3 + v\mathbf{v}_4, \qquad s + t + u + v = 1 \tag{1}$$

where (*s, t, u, v*) is the barycentric coordinate of **p** relative to the vertices $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, $\mathbf{v}_4$. The barycentric coordinates of $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, $\mathbf{v}_4$ are thus (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), and (0, 0, 0, 1), respectively. Using Gua's approach (3), a quadric algebraic patch interpolating three vertices of T is given by

$$\begin{aligned} a_b(s,t,u,v) &= w_{2000}s^2 + w_{0200}t^2 + w_{0020}u^2 + w_{0002}v^2 \\ &\quad + 2w_{1100}st + 2w_{1001}sv + 2w_{0011}uv + 2w_{0110}tu \\ &\quad + 2w_{1010}su + 2w_{0101}tv \\ &= 0 \end{aligned} \tag{2}$$

where w_{ijk} are the weights associated with the control points on the tetrahedron as shown in Fig. 1. Assuming the surface passes through the vertices $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, the weights w_{2000}, w_{0200}, w_{0020} are zero. Normalizing Equation 2, or setting w_{0002} to 1, gives

$$\begin{aligned} a_b(s,t,u,v) &= v^2 + 2w_{1100}st + 2w_{1001}sv + 2w_{0011}uv \\ &\quad + 2w_{0110}tu + 2w_{1010}su + 2w_{0101}tv \\ &= 0 \end{aligned} \tag{3}$$

The shape of a surface ·patch is thus defined by the weights w_{1100}, w_{0110}, w_{1010}, w_{1001}, $w_{0101,}$ w_{0011}, which are determined by the surface normals $\mathbf{n}_{\mathrm{i}}$ of the surface at the base vertices of the tetrahedron as given by the equations (3).

$$\begin{aligned} &w_{1100} = \frac{1}{2}(\mathbf{v}_2 - \mathbf{v}_1)\cdot\mathbf{n}_1,\ w_{0110} = \frac{1}{2}(\mathbf{v}_3 - \mathbf{v}_2)\cdot\mathbf{n}_2, \\ &w_{1010} = \frac{1}{2}(\mathbf{v}_1 - \mathbf{v}_3)\cdot\mathbf{n}_3, \\ &w_{1001} = \frac{1}{2}(\mathbf{v}_4 - \mathbf{v}_1)\cdot\mathbf{n}_1,\ w_{0101} = \frac{1}{2}(\mathbf{v}_4 - \mathbf{v}_2)\cdot\mathbf{n}_2, \\ &w_{0011} = \frac{1}{2}(\mathbf{v}_4 - \mathbf{v}_3)\cdot\mathbf{n}_3, \end{aligned} \tag{4}$$

The shape of an algebraic surface can thus be changed by adjusting the locations of the control points on the tetrahedron.

Deforming Tensor Product Surfaces

Parametric surfaces are usually represented with a set of geometric entities. Consider a tensor product surface $\mathbf{s}(u, v)$ expressed in terms of a set of control polygon vertices (e.g., Bezier surface and NURBS surfaces) (5)

$$\mathbf{s}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m}\alpha_{i,k}(u)\beta_{j,l}(v)\mathbf{b}_{i,j} \tag{5}$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

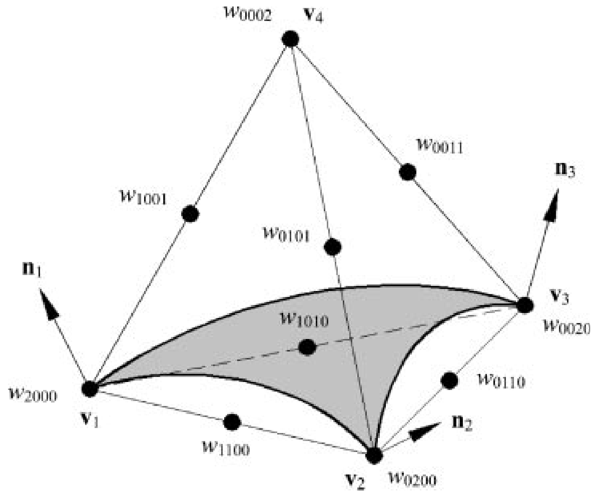


Figure 1. An algebraic surface patch.

where $\alpha_{i,k}(u)$, $\beta_{j,l}(v)$ are blending functions or basis functions associated with each control polygon vertices of the surface and k and l are orders of the basis functions in the u and v directions, respectively. The shape of the surface $\mathbf{s}(u, v)$ can be easily modified by adjusting the location of the control polygon vertices $\mathbf{b}_{i,j}$. For tensor product surfaces, $\mathbf{s}(u, v)$ approximates the control points except at the corners where $\mathbf{s}(u, v)$ interpolates the corner vertices. By subdividing $\mathbf{s}(u, v)$ into a set of C^{k-1}, and C^{l-1} continuous surface patches $\mathbf{s}_i(u, v)$ along the u and v directions, deformation of $\mathbf{s}(u, v)$ can be controlled by adjusting the corner vertices of $\mathbf{s}_i(u, v)$. This allows points on the surface to be used as handles for the deformation. However, special consideration has to be taken to maintain the continuity of the surface at these corner vertices (5). This is also true for surfaces that are defined by its corner vertices and tangents, e.g., parametric cubic surfaces; deformation of the surface can be achieved through adjusting the corner vertices and tangents.

Trimming Tensor Product Surfaces

In general, adjusting the control polygon vertices or surface points of a surface modifies the curvature of a surface. Although the boundary of a surface can be adjusted by modifying the control vertices on the boundary of a surface, this may also affect the curvature of a surface. To adjust the boundary of a surface without affecting the surface curvature, the concept of trimmed surface is usually adopted.

A trimmed surface is a surface with irregular boundaries. Consider a region of a surface defined by a series of trimming curves on a surface $\mathbf{s}(u,v)$ as shown in Fig. 2. A popular approach for representing trimming curves of a surface is to specify the curves on the parametric space of $\mathbf{s}(u, v)$. For example, in a trimmed NURBS surface, a trimming curve is expressed as

$$\mathbf{c}(t) = (u(t), v(t)) = \sum_{i=0}^{n} R_{i,k}(t)\mathbf{b}_i \tag{6}$$

where $\mathbf{b}_i = (u_i, v_i)$ are points on the parametric space of $\mathbf{s}(u, v)$,

$$R_{i,k}(t) = \frac{w_i N_{i,k}(t)}{\sum_{j=0}^{n} w_j N_{j,k}(t)}$$

$$N_{i,k}(t) = \frac{(t - t_i)N_{i,k-1}(t)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - t)N_{i+1,k-1}(t)}{t_{i+k} - t_{i+1}},$$

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } t_i \le t \le t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

and w_i are the weights associated with $\mathbf{b}_i$. The trimmed curve in 3D space is thus $\mathbf{s}(u(t), v(t))$. Trimming curves are closed loops in the parametric space of $\mathbf{s}(u, v)$. To identify the valid region of the trimmed surface, a popular approach is to use the direction of the trimming curve loop to indicate the region of the surface to be retained (6). For instance, a counter-clockwise loop denotes the outer boundary of the surface, whereas a clockwise loop indicates a hole of the surface.

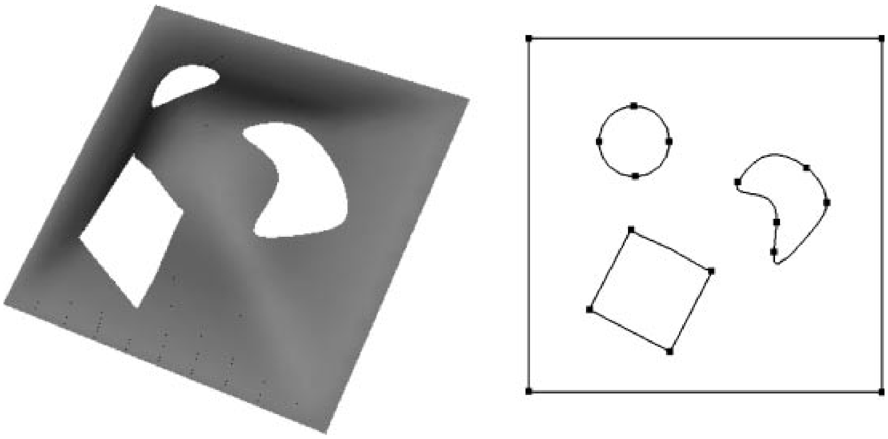


Figure 2. A trimmed surface and its trimming curves in the parametric space.

Deforming Subdivision and Mesh Surfaces

Subdivision surfaces and simple polygon meshes are also popular choices for deformable surfaces as they are defined by a set of points $\mathbf{b}_{i,j}$ constituting the base mesh or polygon mesh of the surface. Deformation can thus be achieved through adjusting the locations of $\mathbf{b}_{i,j}$. However, manipulating individual vertices of a polygon mesh is a tedious process especially for surfaces with a high mesh density. Deformation tools are thus usually adopted for this purpose as discussed below.

GEOMETRY-BASED DEFORMATION TOOL

Popular tools for geometry-based deformation include the lattice-based deformation, the curve-based deformation, and the constraints-based deformation.

Free-Form Deformation (FFD)

The free-form deformation technique introduced by Thomas W. Sederberg and Scott R. Parry (7) deforms an object by embedding the object in a parallelepiped region. Defining a local coordinate system at a point $\mathbf{p}_0$ in the parallelepiped region as shown in Fig. 3, a point $\mathbf{p}$ of the object can be expressed as

$$\mathbf{p} = \mathbf{p}_0 + s\mathbf{l} + t\mathbf{m} + u\mathbf{n} \tag{7a}$$

where $\mathbf{l}$, $\mathbf{m}$, $\mathbf{n}$ are the coordinate axes at $\mathbf{p}_0$ and their corresponding coordinates are

$$s = \frac{(\mathbf{p}-\mathbf{p}_0)\cdot\mathbf{l}}{|\mathbf{l}|^2}, \quad t = \frac{(\mathbf{p}-\mathbf{p}_0)\cdot\mathbf{m}}{|\mathbf{m}|^2}, \quad u = \frac{(\mathbf{p}-\mathbf{p}_0)\cdot\mathbf{n}}{|\mathbf{n}|^2} \tag{7b}$$

so that $0 \le s \le 1$, $0 \le t \le 1$, and $0 \le u \le 1$ for $\mathbf{p}$ lying in the parallelepiped. By imposing a grid of control points $\mathbf{q}_{ijk}$ on parallelepiped, where

$$\mathbf{q}_{ijk} = \mathbf{p}_o + \frac{i}{l}\mathbf{l} + \frac{j}{m}\mathbf{m} + \frac{k}{n}\mathbf{n}, \tag{8}$$

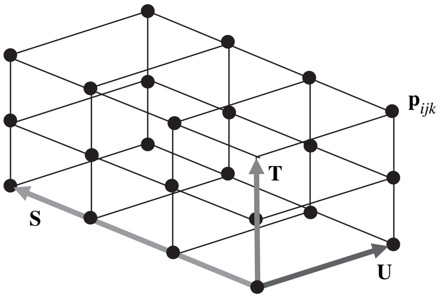


Figure 3. An FFD lattice.

and l, m, n are the dimensions of the grid in the $\mathbf{l}$, $\mathbf{m}$, $\mathbf{n}$ direction respectively, the point $\mathbf{p}$ can be expressed as

$$\mathbf{p} = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n}\binom{l}{i}\binom{m}{j}\binom{n}{k}(1-s)^{l-i} \times s^i(1-t)^{m-j}t^j(1-u)^{n-k}u^k\mathbf{q}_{ijk} \tag{9}$$

A change in the location of the control points $\mathbf{q}_{ijk}$ thus causes the point $\mathbf{p}$ and hence the object to be deformed. Denote F_1 as a mapping $F_1\colon E^3 \to R^3$, such that $\mathbf{p} \in E^3$, $\mathbf{p}_L \in R^3$, and $\mathbf{p}_L = (s, t, u)$, so that $\mathbf{p}_L = F_1(\mathbf{p})$ is the function in Equation 7b. Define F_2 as the mapping $F_2\colon R^3 \to E^3$, where F_2 is the function in Equation 9. A free-form deformation is a mapping $F = F_1 \circ F_2$, where $F : E^3 \to E^3$ transforms every point of an object S to that of the deformed object $F(S)$. As F is independent of the representation of S, the deformation can be applied to an object irrespective of its representation. However, applying the deformation to surface points or the control points of the surface may result in different surfaces as shown in Fig. 4. An example illustrating the application of FFD on the algebraic surface (8) discussed in Section 2.1 is shown in Fig. 5.

Extended Free-Form Deformation

The control lattice as constructed with Equation 8 is parallelepiped. This contrained the shape and size of the region to be deformed by FFD. Coquillart (9) proposed an Extended FFD that allows the use of a non-parallelepiped or non-prismatic control lattice. Predefined lattices in EFFD include the parallelepipedical and cylindrical lattices (Fig. 6). These lattices can be edited and combined to give more complex lattices. An EFFD lattice thus consists of a set of standard FFD lattices that may include degenerated vertices. A surface or a region of a surface is associated with the EFFD lattice. Given a point $\mathbf{p}$ in the region of a surface to be deformed, the corresponding FFD lattice

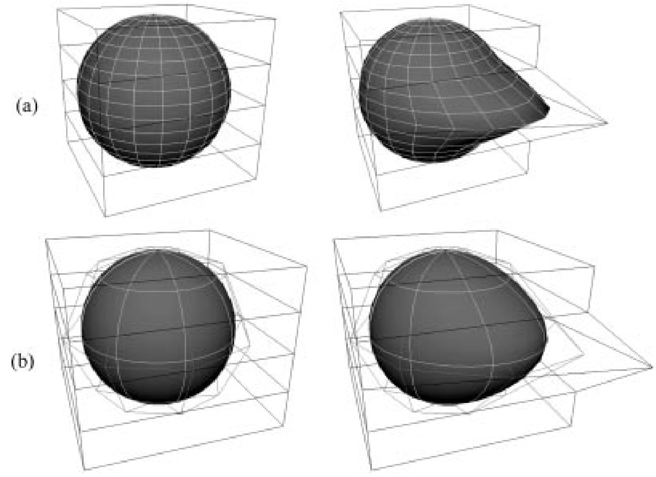


Figure 4. Deformation of a sphere using FFD. (a) Applying the deformation to the polyhedral model of a sphere. (b) Applying the deforming to the control polygon of a NURBS sphere.

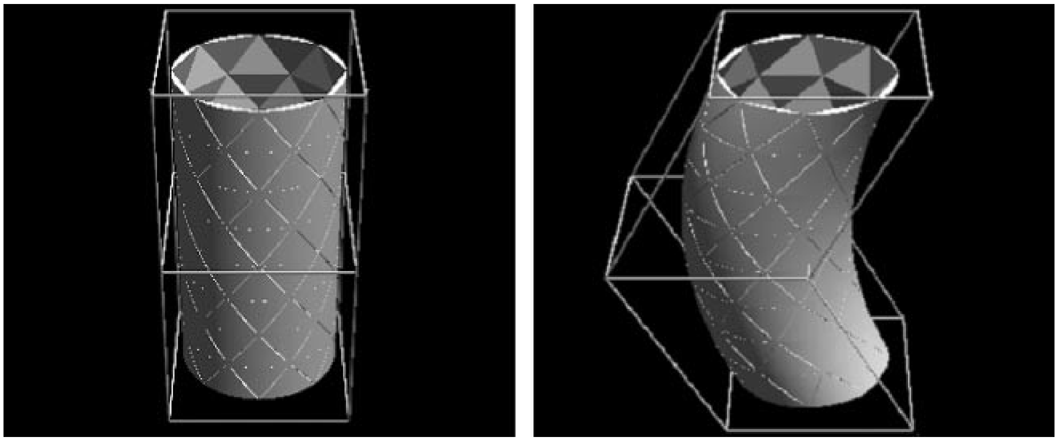


Figure 5. Applying FFD on an algebraic surface.

containing **p** is located. The lattice coordinate (s, t, u) of **p** relative to the corresponding FFD lattice is then determined using Newton approximation. Whenever there is a change in the locations of the lattice vertices, the position of **p** in E^3 can be computed using Equation 9. This allows the shape of the surface covered by the lattice to be modified by adjusting the EFFD lattice control points.

Direct Free-Form Deformation

Free-form deformation allows an object to be deformed by adjusting the lattice control points. A more intuitive approach for controlling the deformation is to allow object points to be adjusted directly. The corresponding positions of the lattice control vertices are then determined (10). This in turn is used to determine the deformation of the other object points. According to Equation 8, a deformed object point **p** can be expressed as **p** = **BQ**, where **Q** is the column matrix of the lattice control points and **B** is a function of s, t, u. Assuming a deformation of the lattice control points $\Delta\mathbf{Q}$ causes a deformation of the object point by $\Delta\mathbf{p}$, then

$$\Delta\mathbf{p} = \mathbf{B}\Delta\mathbf{Q} \tag{10}$$

Given $\Delta\mathbf{p}$, the deformation of the lattice control points $\Delta\mathbf{Q}$ is to determined by solving Equation 10. As there are $(m+1)(n+1)(l+1)$ control points in the FFD lattice, $\Delta\mathbf{Q}$ is thus a $(m+1)(n+1)(l+1)$ by 3 matrix and **B** is a row matrix with $(m+1)(n+1)(l+1)$ elements. As **B** cannot be inverted, the pseudoinverse $\mathbf{B}^+$ of **B** can be used to obtain $\Delta\mathbf{Q}$, i.e.

$$\Delta\mathbf{Q} = \mathbf{B}^+\Delta\mathbf{p} \tag{11}$$

where $\mathbf{B}^+ = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T$.

Using the pseudoinverse for solving Equation 10 gives the best solution, in a least-squares sense, for the required deformation.

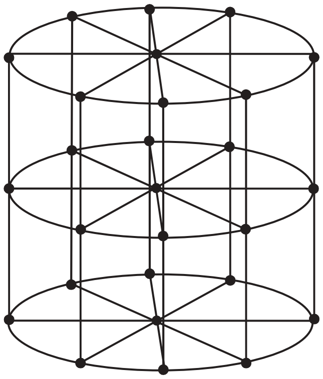


Figure 6. A cylindrical lattice.

Other Free-Form Deformation Techniques

There are several other variations of FFD, including the use of non-uniform rational B-splines for the control lattice, which provides more control on the deformation (11). The use of lattice with arbitrary topology constructed with subdivision surfaces (12) or triangles (13) allows irregularly shaped lattice to be used for the deformation. Free-form deformation is also applied for animation (14). A more recent study on geometric wraps and deformation can found in Ref. 15.

Axial Deformation

Axial deformation (16) deforms an object by associating a curve with an object. Deformation of the object can be achieved by manipulating the curve. Axial deformation is a mapping α: $E^3 \to E^3$, and $\alpha = f \circ g$, where $g: E^3 \to R^4$, and f: $R^4 \to E^3$, where the function g converts an object point **p** in E^3 to the axial curve space defined by a given curve $\mathbf{c}(t)$. The function f maps a point in axial space to E^3. Converting a point in E^3 to the axial space of a curve is to specify the given point relative to a local coordinate frame on $\mathbf{c}(t)$. The local coordinate frame of a curve is a set of normalized orthogonal vectors $\mathbf{u}(t)$, $\mathbf{v}(t)$, and $\mathbf{w}(t)$ defined at the point $\mathbf{c}(t)$. A popular approach is to specify the local coordinate frame using the Frenet Frame, which is expressed as

$$\mathbf{w} = \frac{\mathbf{c}'(t)}{|\mathbf{c}'(t)|} \tag{12a}$$

$$\mathbf{u} = \frac{\mathbf{c}'(t) \times \mathbf{c}''(t)}{|\mathbf{c}'(t) \times \mathbf{c}''(t)|} \tag{12b}$$

$$\mathbf{v} = \frac{\mathbf{w} \times \mathbf{u}}{|\mathbf{w} \times \mathbf{u}|} \tag{12c}$$

The Frenet Frame of a curve is defined by the tangent, normal, and binormal of $\mathbf{c}(t)$. The local coordinate frame is thus completely defined by the curve $\mathbf{c}(t)$, and there is no user control on the orientation of the frame, which may not be desirable. In addition, $\mathbf{c}''(t)$ may vanish so that the normal and hence the coordinate frame is not defined. An alternative is to use the rotation minimizing frame (17).

$$\mathbf{w} = \frac{\mathbf{c}'(t)}{|\mathbf{c}'(t)|} \tag{13a}$$

$$\mathbf{u}'(t) = -\frac{(\mathbf{c}''(t) \cdot \mathbf{u}(t)\mathbf{c}'(t))}{|\mathbf{c}'(t)|^2} \tag{13b}$$

$$\mathbf{v}'(t) = \frac{(\mathbf{c}''(t) \cdot \mathbf{v}(t))\mathbf{c}'(t)}{|\mathbf{c}'(t)|^2} \tag{13c}$$

Given an initial coordinate frame, the set of differential equations (13a–13c) can be solved to obtain a rotation minimized frame along the curve $\mathbf{c}(t)$. As only one local coordinate frame is specified, this technique cannot be used for obtaining a smooth transition between user-defined local coordinate frames. That is, the twist of a curve and hence the twist of the object cannot be fully controlled.

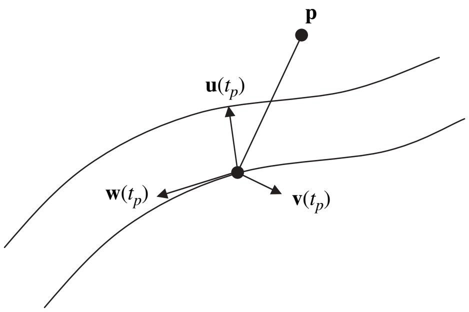


Figure 7. Axial curve pair.

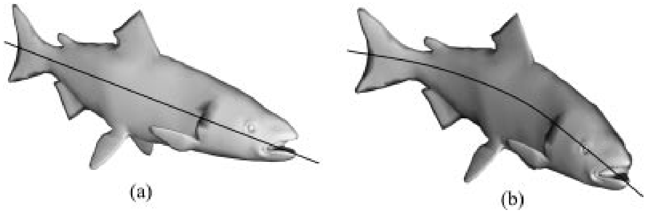


Figure 8. Axial deformation. (a) Object with an undeformed axial curve. (b) Object with the deformed axial curve.

The use of a curve-pair (18) for specifying the local coordinate frame provides complete control over the twist of a curve. A primary curve specifies the location of the local coordinate frame, and an orientation curve is used to define the rotation of the frame relative to the curve (Fig. 7). In this case, a special technique is required for synchronizing the change in shape of both the primary and the orientation curve. With a well-defined local coordinate frame on an axial curve $\mathbf{c}(t)$, and a given point $\mathbf{p}$ on an object, $\mathbf{p}$ can be expressed relative to the curve $\mathbf{c}(t)$ as

$$\mathbf{p} = \mathbf{c}(t_p) + \lambda\mathbf{u}(t_p) + \beta\mathbf{v}(t_p) + \gamma\mathbf{w}(t_p) \tag{14}$$

where

$$\lambda = (\mathbf{p} - \mathbf{c}(t_p)) \cdot \mathbf{u}(t_p) \tag{15a}$$

$$\beta = (\mathbf{p} - \mathbf{c}(t_p)) \cdot \mathbf{v}(t_p) \tag{15b}$$

$$\gamma = (\mathbf{p} - \mathbf{c}(t_p)) \cdot \mathbf{w}(t_p) \tag{15c}$$

and $\mathbf{c}(t_p)$ is the point on $\mathbf{c}(t)$ corresponding to $\mathbf{p}$. Usually, $\mathbf{c}(t_p)$ is the point on $\mathbf{c}(t)$ closest to $\mathbf{p}$. A point in the axial space of $\mathbf{c}(t)$ corresponding to $\mathbf{p}$ is thus $(t_p, \lambda, \beta, \gamma)$. By specifying object points in the axial space of $\mathbf{c}(t)$, the shape of the oe deformed by adjusting $\mathbf{c}(t)$. Figure 8 shows an example of applying an axial deformation to an object. Figure 9 shows the bending and twisting of an object using the curve-pair approach. A zone of influence can be associated with the axial curve to constraint the region of influence of the axial curve. The zone of influence is defined as the region between two general cylinders around the axial curve. Denote $r_{\min}$ and $r_{\max}$ as the radii of the zone of influence; an object point $\mathbf{p}$ will only be deformed if $r_{\min} \leq |\mathbf{p} - \mathbf{c}(\mathbf{t_p})| \leq r_{\max}$. This allows an object to be

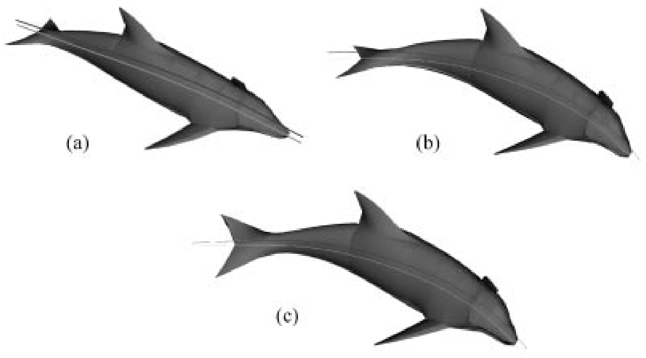


Figure 9. Curve-pair based axial deformation. (a) Object with an undeformed axial curve-pair. (b) Effect of bending an axial curve-pair. (c) Effect of twisting an axial curve-pair.

deformed locally as all object points lying outside of the zone of influence are not deformed.

The curve-based deformation technique also includes the "Wire" deformation technique (19) in which a deformation is defined with a tuple (*W*, *R*, *s*, *r*,*f*), where *W* is the wire curve, *R* is a reference curve, *r* specifies the zone of influence (the distance from the curve), and *s* is the scale factor. By adjusting the wire curve *W*, the shape of *W* relative to the reference curve *R* determines the deformation on the associated object.

Constraint-Based Deformation

The constraints-based approach (20) deforms a surface by defining a set of point displacement constraints. Deformations satisfying these constraints are determined by solving a system of equations satisfying the constraints. Given a point $\mathbf{u} = [u_1, u_2, \ldots, u_n]^{\mathrm{T}}$, and its displacement $d(\mathbf{u}) = [d_1(\mathbf{u}), d_2(\mathbf{u}), \ldots, d_n(\mathbf{u})]^{\mathrm{T}}$, the deformation at a point can be expressed as

$$d(\mathbf{u}) = \mathbf{M}\, f(\mathbf{u}) \tag{16}$$

where *f* is a function of the mapping $f: R^n \to R^m$, $m \geq n$, and $\mathbf{M}$ is the matrix of a linear transformation $T : R^m \to R^n$. Assume there are n_c constraint points, then $d(\mathbf{u}_i) = [d_1(\mathbf{u}_i), d_2(\mathbf{u}_i), \ldots, d_n(\mathbf{u}_i)]^T$, $j = 1, \ldots, n_c$. With a given function *f*, the matrix $\mathbf{M}$ is obtained by solving the system of $n \times n_c$ equations

$$[d(\mathbf{u}_1) \quad \ldots \quad d(\mathbf{u}_{n_c})]^T = \mathbf{M}[f(\mathbf{u}_1) \quad \ldots \quad f(\mathbf{u}_{n_c})]^T \tag{17}$$

In general, n may not be the same as n_c, and $\mathbf{M}$ is obtained using pseudoinverse. Once $\mathbf{M}$ is obtained, the displacement of any point of the object can be computed using Equation 16. Based on the same concept, the Simple Constrained Deformation (21) associates a desired displacement and radius of influence for each constraint point. A local B-spline basis function centered at the constraint point is determined that falls to zero for points beyond the radius of influence. The displacement of an object point is obtained by a linear combination of the basis functions such that all constraints are satisfied.

Physics-Based Deformation

Physics-based deformation techniques are deformation methods based on continuum mechanics, which accounts for the effects of materials properties, external forces, and environmental constraints on the deformation of objects.

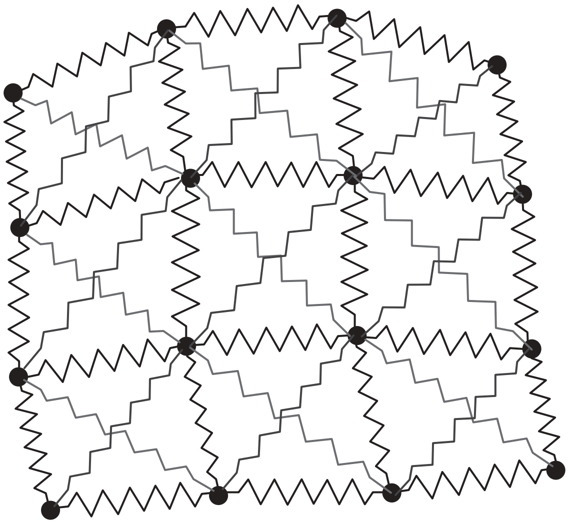


Figure 10. A surface modeled with a mass-spring system.

Mass-Spring Model

The mass-spring model is one of the most widely used techniques for physics-based surface deformation (22). A surface is modeled as a mesh of point masses connected by springs as shown in Fig. 10. The springs are usually assumed to be linearly elastic, although in some cases, nonlinear springs are used to model inelastic behavior. In general, a mass point at the position $\mathbf{x}_i$ in a mass spring system is governed by the Newton's Second Law of motion

$$m_i \underset{i}{\ddot{\mathbf{x}}} = -\gamma_i \underset{i}{\dot{\mathbf{x}}} + \sum_{j=0}^{n_g} \mathbf{g}_{ij} + \mathbf{f}_i \tag{18}$$

where m_i is the mass of the mass point, $\gamma_i \dot{\mathbf{x}}_i$ is the velocity-dependent damping force, $\mathbf{g}_{ij}$ are the spring forces acting at m_i, and $\mathbf{f}_i$ is the external force at m_i. Assuming there are n mass points in the system, applying Equation 18 at each of the mass points gives

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{f} \tag{19}$$

where $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ are the $3n \times 3n$ mass, damping, and stiffness matrices, respectively, and $\mathbf{f}$ is a $3n \times 1$ column vector denoting the total external forces on the mass point. Equation 19 describes the dynamics of a mass-spring system and is thus commonly used for modeling surfaces that are deformed dynamically. Given the initial position $\mathbf{x}$ and velocity $\mathbf{v}$ of the mass points, and expressing Equation 19 as

$$\mathbf{v} = \mathbf{M}^{-1}(-\mathbf{C}\mathbf{v} - \mathbf{K}\mathbf{x} + \mathbf{f}) \tag{20a}$$

$$\dot{\mathbf{x}} = \mathbf{v} \tag{20b}$$

the velocity $\mathbf{v}$ of the mass point at subsequent time instances can be obtained. The corresponding location of the mass point can be obtained with various numerical integration techniques. Mass-spring systems are easy to construct and can be animated at interactive speed. However, it is a significant simplification of the physics in a continuous body. A more accurate physical model for surface deformation is to use the finite/boundary element method.

Deformation using Finite/Boundary Element Method

Using the finite element method (FEM), an object is approximated with a set of discrete elements such as triangles, quadrilaterals, tetrahedrons, and cuboids. Solid elements (e.g., tetrahedrons and cubes) are used for closed surfaces, whereas plate or shell elements are used for nonclosed surfaces. As an object is in equilibrium when its total potential energy is at a minimum, an equilibrium equation is established by considering the total strain energy of a system and the work due to external forces. The boundary element method (BEM) approximates the closed surface of a volume with a set of elements such as triangles and quadrilaterals. No solid element is thus required for BEM. Assuming the deforming object is linearly elastic, both FEM and BEM result in a simple matrix equation

$$\mathbf{F} = \mathbf{K}\mathbf{U} \tag{21}$$

where $\mathbf{F}$ denotes the external forces acting on element nodes of the object, $\mathbf{U}$ is the displacement at the element nodes, and $\mathbf{K}$ is the stiffness matrix. Detailed discussion on the derivation of Equation 21 can be found in Refs. 22 and 23 and in standard texts on FEM/BEM. Given the material properties of an object, the variables at an element node are the external force acting on the node and the corresponding displacement. In general, one of the variables is known, whereas the other one is unknown. By partitioning $\mathbf{F}$ and $\mathbf{U}$ into known and unknown elements, Equation 21 can be rewritten as

$$\begin{bmatrix} \mathbf{F}_k \\ \mathbf{F}_u \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{00} & \mathbf{K}_{01} \\ \mathbf{K}_{10} & \mathbf{K}_{11} \end{bmatrix} \begin{bmatrix} \mathbf{U}_k \\ \mathbf{U}_u \end{bmatrix} \tag{22}$$

where $\mathbf{F}_k$, $\mathbf{F}_u$ denotes the known and unknown nodal forces and $\mathbf{U}_k$, $\mathbf{U}_u$ denotes the known and unknown nodal displacements.

Hence,

$$\mathbf{F}_k = \mathbf{K}_{00}\mathbf{U}_k + \mathbf{K}_{01}\mathbf{U}_u \tag{23a}$$

$$\mathbf{F}_u = \mathbf{K}_{10}\mathbf{U}_k + \mathbf{K}_{11}\mathbf{U}_u \tag{23b}$$

In general, for graphics applications, the diplacement at certain nodes is specified or constrained, and the forces at the free or unconstrained nodes are zero (i.e., $\mathbf{F}_k = 0$). Assume there are n nodes in the object with known displacement at k of the nodes. The free boundary will be composed of $n - k$ nodes. The dimensions of $\mathbf{K}_{00}$ and $\mathbf{K}_{01}$ are thus $(n - k) \times k$, and $(n - k) \times (n - k)$. Equation 23a gives

$$\mathbf{U}_u = -\mathbf{K}_{01}^{-1}\mathbf{K}_{00}\mathbf{U}_k \tag{24}$$

Given the deformation at certain nodes of a surface, the deformation of the other nodes can be obtained using Equation 24.

In general, FEM and BEM technique can only be applied for solid objects or closed surfaces. Standard plate or shell elements can be used for open surfaces by assuming a constant thickness for the surface. By considering the potential energy in the stretching and bending of a surface, an energy functional can be established for a surface (24,25). Minimizing the energy functional gives an equation in the form of Equation 21. Standard FEM techniques can then be used for deforming the surface. A mass and a damping term can also be incorporated into Equation 21 to allow for the simulation of dynamic effects.

REMARKS

Geometry-based deformation tools are usually employed for deforming surfaces in the object construction process in which the deformation does not need to obey the laws of physics. For the purposes of animation, the use of geometry-based deformation for generating realistic motions relies very much on the skill and experience of the animator in using the deformation tool. On the contrary, physics-based deformation is capable of generating an animated surface deformation with little user interactions and thus plays an important role in animation.

BIBLIOGRAPHY

1. A. H. Barr, Global and local deformations of solid primitives, *ACM Comput. Graphics*, **18**(3): 21–30, 1984.
2. T. W. Sedergerg, Piecewise algebraic surface patches, *Comput.-Aided Geometric Design*, **2**: 53–59, 1985.
3. B. Guo, Representation of arbitrary shapes using implicit quadrics, *Visual Comput.*, **9**(5): 267–277, 1993.
4. C. L. Bajaj, Free-form modeling with implicit surface patches, In: J. Bloomenthal and B. Wyvill (eds.), *Implicit Surfaces*, San Francisco, CA: Morgan Kaufman Publishers, 1996.
5. G. Farin, *Curves and Surfaces for CAGD, a Practical Guide*, 4th ed.San Diego, CA: Academic Press, 1997.
6. OpenGL Architecture Review Board, D. Shreiner, M. Woo, and J. Neider, *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, Version 2 (5th Edition), 2005.
7. T. W. Sederberg, and S. R. Parry, Free-form deformation of solid geometric models, *ACM Computer Graphics*, **20**(4): 151–160, 1986.
8. K. C. Hui, Free-form deformation of constructive solid model, *Comput.-Aided Design*, **35**(13): 1221–1224, 2003.
9. S. Coquillart, Extended free-form deformation: A sculpturing tool for 3D geometric modeling, *ACM Comput. Graphics*, **24**(4): 187–196, 1990.
10. W. M. Hsu, J. F. Hughes, H. Kaufmann, Direct manipulation of free-form deformations, *ACM Comput. Graphics*, **26**(2): 177–184, 1992.
11. H. L. Lamousin, W. N. Waggenspack, NURBS-based free-form deformations, *IEEE Comput. Graphics Applicat.*, **14**(6): 59–65, 1994.
12. R. MacCracken, and K. I. Joy, Free-form deformations with lattices of arbitrary topology, *Proc. SIGGRAPH*, 1996, pp. 181–188.
13. K. G. Kobayashi, and K. Ootsubo, t-FFD: Free-form deformation by using triangular mesh, *Proc. 8th ACM Symposium on Solid Modeling and Applications*, 2003, pp. 226–234.
14. S. Coquillart, and P. Jancene, Animated free-form deformation: An interactive animation technique, *ACM Comput. Graphics*, **25**(4): 23–26, 1991.

15. T. Milliron, R. J. Jensen, R. Barzel, and A. Finkelstein, A framework for geometric warps and deformations, *ACM Trans. Graphics*, **21**(1): 20–51, 2002.

16. F. Lazarus, S. Coquillart, and Jancene P. , Axial deformations: An intuitive deformation technique, *Comput.-Aided Design*, **26**(8): 607–613, 1994.

17. F. Klok, Two moving coordinate frames for sweeping along a 3D trajectory, *Comput.-Aided Geometric Design*, **3**: 217–229, 1986.

18. K. C. Hui, Free-form design using axial curve-pairs, *Comput.-Aided Design*, **34**(8): 583–595, 2002.

19. K. Singh, and E. Fiume, Wires: A geometric deformation technique, *Proc. SIGGRAPH*, 1998, pp. 405–414.

20. P. Borrel, and D. Bechmann, Deformation of N-dimensional objects, *Internat. J. Computat. Geomet. Applicat.*, **1**(4): 427–453, 1991.

21. P. Borrel, and A. Rapporort, Simple constrained deformations for geometric modeling and interactive design, *ACM Trans. Graphics*, **13**(2): 137–155, 1994.

22. S. F. F. Gibson, and B. Mirtich, A survey of deformable modeling in computer graphics, *TR-97-19, MERL-A Mitsubishi Electric Research Laboratory*, Available: http://www.merl.com.

23. D. L. James and D. K. Pai, ARTDEFO: Accurate Real Time Deformable Objects, *Proc. SIGGRAPH*, 1999, pp. 65–72.

24. G. Celniker, and D. Gossard, Deformable curve and surface finite-elements for free-form shape design, *ACM Comput. Graphics*, **25**(4): 257–266, 1991.

25. D. Terzopoulos, H. Qin, Dynamic NURBS with geometric constraints for interactive sculpting, *ACM Trans. Graphics*, **13**(2): 103–136, 1994.

K. C. Hui
The Chinese University of
Hong Kong
Shatin, Hong Kong

S

SURFACE MODELING

INTRODUCTION

Surface is a fundamental element in describing the shape of an object. The Classic method for specifying a surface is to use a set of orthogonal planar projection curves to describe the surface. With the advance of computer graphics and computer-aided design technologies, the wire frame models has evolved and is now capable of describing three-dimensional (3-D) objects with lines and curves in 3-D space. However, the wire frame model is ambiguous and does not provide a complete 3-D description of a surface. P. Bezier and P. de Casteljau independently developed the Bezier curves and surfaces in the late 1960s, which revolutionized the methods for describing surfaces. In general, surface modeling is the technique for describing a 3-D surface. There are three popular ways for specifying a surface, namely, implicit, parametric, and subdivision. They are described in the following sections.

IMPLICIT SURFACE

An implicit surface is a set of points $\mathbf{p} = (x, y, z)$ in space such that $f(\mathbf{p}) = k$, where k is constant. In general, an implicit surface is a function of the mapping $f : E^3 \rightarrow R$, where E^3 is the 3-D Euclidean space and R is the real axis. Popular implicit surfaces include the quadric surfaces and toruses. Implicit surfaces are unbounded. Extra constraints are required to specify a valid range for the surface. Direct display of an implicit surface can be performed with a ray tracing process. This process requires performing a ray/surface intersection, which is a time-consuming process. An alternative is to approximate the surface with a polygon mesh, which requires performing a polygonization process on $f(\mathbf{p})$. Adjusting the shape of an implicit surface requires modifying the function f, which is not intuitive for general users. However, implicit surfaces are useful for modeling objects that can be described with implicit or algebraic functions, e.g., blends between objects. The following discussion focuses on parametric surfaces that are more commonly used in existing geometric modeling systems.

PARAMETRIC SURFACES

A parametric surface is a mapping $f : E^2 \rightarrow E^3$ that maps a point (u, v) in the parametric space to the 3-D Euclidean space, (Fig. 1):

$$\mathbf{s}(u,v) = f(u,v) = \lfloor\, f_x(u,v) \quad f_y(u,v) \quad f_z(u,v)\,\rfloor \tag{1}$$

Keeping u (or v) constant, and varying v (or u), an isoparametric curve is obtained. The tangents along the u and v directions are defined as

$$\mathbf{s}_u(u,v) = \frac{\partial \mathbf{s}(u,v)}{\partial u} = \begin{bmatrix} \dfrac{\partial f_x(u,v)}{\partial u} & \dfrac{\partial f_y(u,v)}{\partial u} & \dfrac{\partial f_z(u,v)}{\partial u} \end{bmatrix} \tag{2}$$

$$\mathbf{s}_v(u,v) = \frac{\partial \mathbf{s}(u,v)}{\partial v} = \begin{bmatrix} \dfrac{\partial f_x(u,v)}{\partial v} & \dfrac{\partial f_y(u,v)}{\partial v} & \dfrac{\partial f_z(u,v)}{\partial v} \end{bmatrix} \tag{3}$$

These two tangent vectors define a tangent plane at $\mathbf{s}(u, v)$ and the normal to the parametric surface is the normal to the tangent plane and is expressed as

$$\mathbf{n}(u,v) = \frac{\mathbf{s}_u(u,v) \times \mathbf{s}_v(u,v)}{|\mathbf{s}_u(u,v) \times \mathbf{s}_v(u,v)|} \tag{4}$$

The twist vector of a surface $\frac{\partial^2 \mathbf{s}(u,v)}{\partial u \partial v}$ measures the rate of change of a u-tangent (v-tangent) in the v-direction (u-direction).

Commonly used parametric surfaces include the sweep surfaces, ruled surfaces, Coon's surfaces, Bezier surfaces, B-spline surfaces, and NURBS surfaces as described below.

SWEEP SURFACE

A sweep surface is a surface obtained by transforming a curve in space. Given a curve $\mathbf{p}(u)$ defined in its local coordinate frame, a 3×3 transformation matrix $\mathbf{M}(v)$, and a translation vector $\mathbf{T}(v)$, a sweep surface is defined as

$$\mathbf{s}(u,v) = \mathbf{p}(u)\mathbf{M}(v) + \mathbf{T}(v) \tag{5}$$

The curve $\mathbf{p}(u)$ is sometime referred to as the sweep contour. Surfaces such as the surfaces of revolution, tabulated cylinders, or extrusion surface are particular cases of sweep surfaces. In a surface of revolution, $\mathbf{M}(v)$ is a rotation. In an extrusion surface, $\mathbf{M}(v)$ is an identity or a scaling transformation if a tapered extrusion is performed. To allow the construction of more general sweep transformation, $\mathbf{M}(v)$ can be defined with a rail curve $\mathbf{c}(v)$, $0 \leq v \leq 1$, such that $\mathbf{M}(v)$ is a rotation about an axis $\dot{\mathbf{c}}(0) \times \dot{\mathbf{c}}(v)$ through an angle $\theta = \cos^{-1}\left(\frac{\dot{\mathbf{c}}(0) \bullet \dot{\mathbf{c}}(v)}{|\dot{\mathbf{c}}(0)||\dot{\mathbf{c}}(v)|}\right)$. An example is shown in Fig. 2. If the curve $\mathbf{p}$ is allowed to change in the sweep, or $\mathbf{p}$ is also a function of v, i.e., $\mathbf{p} = \mathbf{p}(u, v)$, a more complex surface form can be obtained. This is often specified with two sweep contours $\mathbf{p}_0(u)$ and $\mathbf{p}_1(u)$, where $\mathbf{p}_0(u)$ denotes the sweep contour at $v = 0$ and $\mathbf{p}_1(u)$ denotes the sweep contour at $v = 1$. The sweep contour along the sweep is determined by blending $\mathbf{p}_0(u)$ and $\mathbf{p}_1(u)$, i.e.,

$$\mathbf{p}(u,v) = \alpha(v)\mathbf{p}_0(u) + \beta(v)\mathbf{p}_1(u) \tag{6}$$

where $\alpha(v)$ and $\beta(v)$ are blending functions of $\mathbf{p}_0(u)$, and $\mathbf{p}_1(u)$, respectively, such that $\alpha(v) + \beta(v) = 1$[1]. A popular set

[1] $\alpha(v) + \beta(v) = 1$ confines $\mathbf{p}(u,v)$ to lie between $\mathbf{p}_0(u)$ and $\mathbf{p}_1(u)$.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

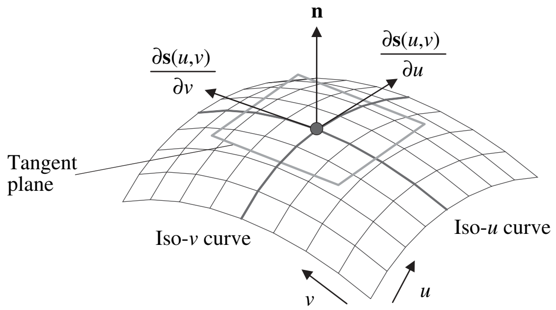

n
$\frac{\partial \mathbf{s}(u,v)}{\partial v}$
$\frac{\partial \mathbf{s}(u,v)}{\partial u}$
Tangent plane
Iso-v curve
Iso-u curve
v
u

Figure 1. A parametric surface.

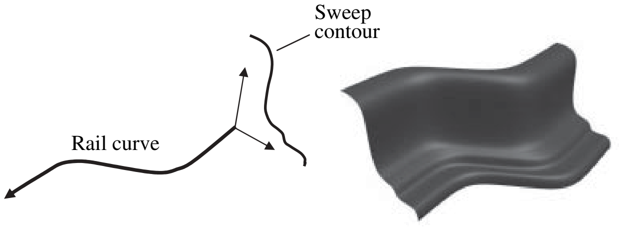


Figure 2. A sweep surface.

of blending functions are $\alpha(v) = 1 - v$, and $\beta(v) = v$. To provide a more flexible sweep operation, the sweep transformation can be defined with two rail curves $\mathbf{c}_1(v)$, $\mathbf{c}_2(v)$; $0 \le v \le 1$ as shown in Fig. 3. The vectors $\mathbf{d}(0) = \mathbf{c}_2(0) - \mathbf{c}_1(0)$ and $\mathbf{d}(v) = \mathbf{c}_2(v) - \mathbf{c}_1(v)$ define the orientation, location, and the size of $\mathbf{p}(u,v)$. The transformation $\mathbf{M}(v)$ is a rotation about an axis $\mathbf{d}(0) \times \mathbf{d}(v)$ through an angle $\theta = \cos^{-1}(\frac{\mathbf{d}(0) \bullet \mathbf{d}(v)}{|\mathbf{d}(0)||\mathbf{d}(v)|})$. Using the rail curves to define the sweep transform, the sweep contour is expressed as

$$\mathbf{p}(u,v) = k(v)[\alpha(v)(\mathbf{p}_0(u) - \mathbf{c}_1(0)) + \beta(v)(\mathbf{p}_1(u) - \mathbf{c}_1(0))] \quad (7)$$

where $k(v) = \frac{|\mathbf{c}_2(v) - \mathbf{c}_1(v)|}{|\mathbf{c}_2(0) - \mathbf{c}_1(0)|}$. The sweep surface is thus obtained with the expression

$$\mathbf{s}(u,v) = \mathbf{p}(u,v)\mathbf{M}(v) + \mathbf{T}(v) \quad (8)$$

where $\mathbf{T}(v) = \mathbf{c}_1(v)$.

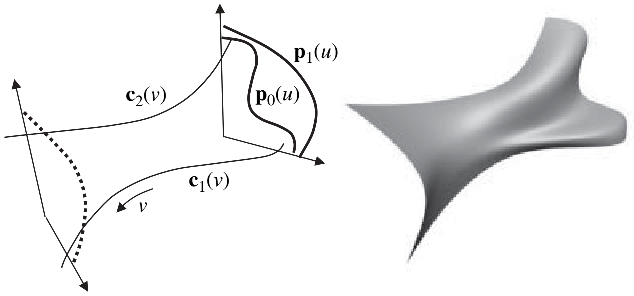


Figure 3. A sweep surface with two rail curves.

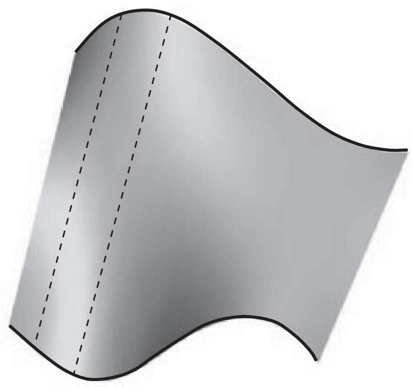


Figure 4. A ruled surface.

RULED SURFACE

A ruled surface is the surface obtained by linearly interpolating between two given curves. Assuming the curves are $\mathbf{p}(u)$ and $\mathbf{q}(u)$, where $0 \le u \le 1$, a linear interpolation between $\mathbf{p}(u)$ and $\mathbf{q}(u)$ gives

$$\mathbf{s}(u,v) = (1 - v)\mathbf{p}(u) + v\mathbf{q}(u) \quad (9)$$

An example of a ruled surface is shown in Fig. 4. If $\mathbf{p}(u)$, $\mathbf{q}(u)$ are straight lines, so that $\mathbf{p}(u) = (1 - u)\mathbf{p}_0 + u\mathbf{p}_1$ and $\mathbf{q}(u) = (1 - u)\mathbf{q}_0 + u\mathbf{q}_1$, then

$$\mathbf{s}(u,v) = (1 - u)(1 - v)\mathbf{p}_0 + u(1 - v)\mathbf{p}_1 + v(1 - u)\mathbf{q}_0 + uv\mathbf{q}_1 \quad (10)$$

The surface define with Equation (10) is specified with four points and is usually referred to as a bilinear surface (Fig. 5).

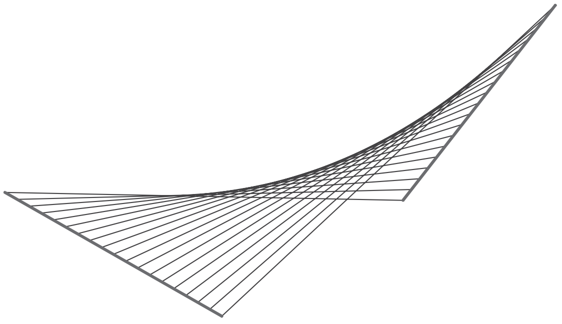


Figure 5. A bilinear surface.

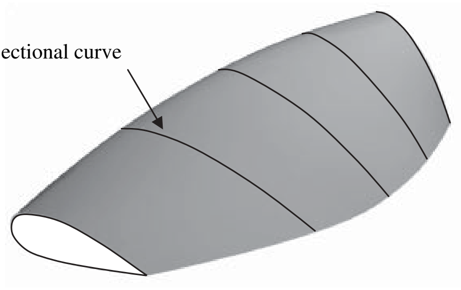


Figure 6. A lofted surface.

LOFTED SURFACE

A lofted surface is a surface interpolating a series of sectional curves as shown in Fig. 6. Given a set of curves $\mathbf{p}_i(u)$, $0 \le u \le 1$, a surface $\mathbf{s}_i(u, v)$ interpolating $\mathbf{p}_i(u)$ and $\mathbf{p}_{i+1}(u)$ can be expressed as

$$\mathbf{s}_i(u,v) = \alpha(v)\mathbf{p}_i(u) + \beta(v)\mathbf{p}_{i+1}(u) \tag{11}$$

where $0 \le u \le 1$ and $\alpha(v)$, $\beta(v)$ are blending functions of $\mathbf{p}_i(u)$ and $\mathbf{p}_{i+1}(u)$, respectively, such that $\alpha(v) + \beta(v) = 1$[2]. To maintain the continuity of the lofted surface, continuity across the boundaries of consecutive $\mathbf{s}_i(u, v)$ has to be maintained. This can be easily achieved by using a cubic blending functions (1), $\alpha(v) = 3v^2 - 2v^3$, and $\beta(v) = 1 - 3v^2 + 2v^3$.

COON'S BILINEAR SURFACE

A Coon's bilinear surface defines a surface with four contiguous boundary curves. Given four curves $\mathbf{p}_0(u)$, $\mathbf{p}_1(u)$, $\mathbf{q}_0(w)$, $\mathbf{q}_1(w)$, $0 \le u \le 1$, $0 \le v \le 1$, such that $\mathbf{p}_0(0) = \mathbf{q}_0(0)$, $\mathbf{q}_0(1) = \mathbf{p}_1(0)$, $\mathbf{p}_1(1) = \mathbf{q}_1(1)$, $\mathbf{q}_1(0) = \mathbf{p}_0(1)$ as shown in Fig. 7(a), a Coon's bilinear surface interpolates all boundary curves and corner points and is expressed as

$$\begin{aligned}\mathbf{s}(u,v) &= (1-v)\mathbf{p}_0(u) + v\mathbf{p}_1(u) + (1-u)\mathbf{q}_0(v) + u\mathbf{q}_1(v)\\ &-(1-u)(1-v)\mathbf{p}_0(0) - u(1-v)\mathbf{p}_0(1) - v(1-u)\mathbf{p}_1(0) - uv\mathbf{p}_1(1)\end{aligned} \tag{12}$$

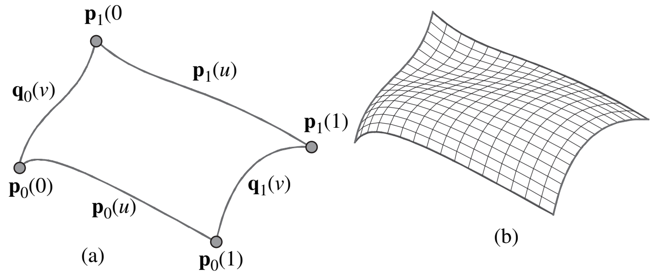


Figure 7. A Coon's bilinear surface: (a) corner vertices and boundary curves and (b) the Coon's bilinear surface.

Expressing Equation (12) in matrix form gives

$$\mathbf{s}(u,v) = \begin{bmatrix} 1-u & u & 1 \end{bmatrix} \begin{bmatrix} -\mathbf{p}_0(0) & -\mathbf{p}_1(0) & \mathbf{q}_0(v) \\ -\mathbf{p}_0(1) & -\mathbf{p}_1(1) & \mathbf{q}_1(v) \\ \mathbf{p}_0(u) & \mathbf{p}_1(u) & 0 \end{bmatrix} \begin{bmatrix} 1-v \\ v \\ 1 \end{bmatrix} \tag{13}$$

A Coon's bilinear surface is usually interpreted as the result of subtracting a bilinear surface defined by the corner points $\mathbf{p}_0(0)$, $\mathbf{p}_0(1)$, $\mathbf{p}_1(0)$, $\mathbf{p}_1(1)$ from the sum of two ruled surfaces defined by $\mathbf{p}_0(u)$, $\mathbf{p}_1(u)$ and $\mathbf{q}_0(v)$, $\mathbf{q}_1(v)$.

BICUBIC SURFACE

A bicubic surface is a surface in which the isoparametric curves are cubic curves and is given by the expression

$$\mathbf{s}(u,v) = \mathbf{F}(u)\mathbf{G}\mathbf{F}^{\mathrm{T}}(v), \tag{14}$$

where

$$\begin{aligned}\mathbf{F}(u) &= [F_1(u) \quad F_2(u) \quad F_3(u) \quad F_4(u)]\\ \mathbf{F}^{\mathrm{T}}(v) &= [F_1(v) \quad F_2(v) \quad F_3(v) \quad F_4(v)]^{\mathrm{T}}\end{aligned}$$

$$\mathbf{G} = \begin{bmatrix} \mathbf{s}(0,0) & \mathbf{s}(0,1) & \mathbf{s}_w(0,0) & \mathbf{s}_w(0,1) \\ \mathbf{s}(1,0) & \mathbf{s}(1,1) & \mathbf{s}_w(1,0) & \mathbf{s}_w(1,1) \\ \mathbf{s}_u(0,0) & \mathbf{s}_u(0,1) & \mathbf{s}_{uw}(0,0) & \mathbf{s}_{uw}(0,1) \\ \mathbf{s}_u(1,0) & \mathbf{s}_u(1,1) & \mathbf{s}_{uw}(1,0) & \mathbf{s}_{uw}(1,1) \end{bmatrix}$$

and

$$\begin{aligned}F_1(u) &= 1 - 3u^2 + 2u^3, F_2(u) = 3u^2 - 2u^3\\ F_3(u) &= u - 2u^2 + u^3, F_4(u) = -u^2 + u^3\\ F_1(v) &= 1 - 3v^2 + 2v^3, F_2(v) = 3v^2 - 2v^3\\ F_3(v) &= v - 2v^2 + v^3, F_4(v) = -v^2 + v^3\end{aligned}$$

Equation (14) can be rewritten as

$$\mathbf{s}^c(u,v) = \mathbf{U}\mathbf{N}_c\mathbf{G}\mathbf{N}_c{}^{\mathrm{T}}\mathbf{V} \tag{15}$$

where

$$\mathbf{U} = [u^3 \quad u^2 \quad u \quad 1], \mathbf{V} = [v^3 \quad v^2 \quad v \quad 1],$$

$$\mathbf{N}_c = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The matrix $\mathbf{G}$ can be partitioned into four 2 × 2 matrices

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_{1,1} & \mathbf{G}_{1,2} \\ \mathbf{G}_{2,1} & \mathbf{G}_{2,2} \end{bmatrix}$$

[2] $\alpha(v) + \beta(v) = 1$ confines $\mathbf{s}(u,v)$ to lie between $\mathbf{p}_i(u)$ and $\mathbf{p}_{i+1}(u)$.

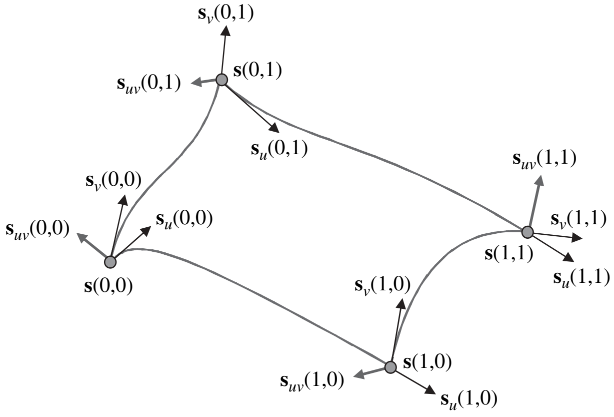


Figure 8. Parameters defining a Bi-cubic surface.

where

$$\mathbf{G}_{1,1} = \begin{bmatrix} \mathbf{s}(0,0) & \mathbf{s}(0,1) \\ \mathbf{s}(1,0) & \mathbf{s}(1,1) \end{bmatrix}, \qquad \mathbf{G}_{1,2} = \begin{bmatrix} \mathbf{s}_v(0,0) & \mathbf{s}_v(0,1) \\ \mathbf{s}_v(1,0) & \mathbf{s}_v(1,1) \end{bmatrix}$$

$$\mathbf{G}_{2,1} = \begin{bmatrix} \mathbf{s}_u(0,0) & \mathbf{s}_u(0,1) \\ \mathbf{s}_u(1,0) & \mathbf{s}_u(1,1) \end{bmatrix}, \quad \mathbf{G}_{2,2} = \begin{bmatrix} \mathbf{s}_{uv}(0,0) & \mathbf{s}_{uv}(0,1) \\ \mathbf{s}_{uv}(1,0) & \mathbf{s}_{uv}(1,1) \end{bmatrix}$$

These submatrices $\mathbf{G}_{1,1}$, $\mathbf{G}_{1,2}$, $\mathbf{G}_{2,1}$, and $\mathbf{G}_{2,2}$ specifies the positions, the v-tangents, the u-tangents, and the twist vectors at the corner vertices of the surface. These parameters at the corner vertices of the surface determine the shape of the bicubic surface (Fig. 8). A change in these parameters except the twist vectors will affect the boundary of the surface, whereas a change in the twist vector will only affect the interior of the surface as shown in Fig. 9.

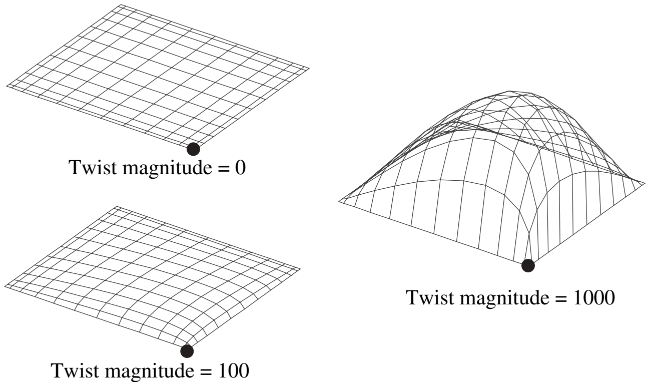


Figure 9. Effect of the twist vector at a vertex on a Bicubic surface.

GENERAL PROPERTIES OF TENSOR PRODUCT SURFACES

Bezier surfaces, B-spline surfaces, and NURBS surfaces are usually referred to as tensor product surfaces. These surfaces are in general represented as the weighted sum of a set of points $\mathbf{p}_{i,j}$ in E^3, i.e.,

$$\mathbf{s}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \alpha_i(u)\beta_j(v)\mathbf{p}_{i,j} \tag{16}$$

where $\sum_{i=0}^{n}\alpha_i(u) = 1$, $\sum_{j=0}^{m}\beta_i(u) = 1$, and hence, $\sum_{i=0}^{n}\sum_{j=0}^{m}\alpha_i(u)\beta_j(v) = 1$. The surface $\mathbf{s}(u,v)$ is thus a barycentric combination of $\mathbf{p}_{i,j}$ and is invariant under affine transformation, i.e., applying an affine transformation to $\mathbf{s}(u,v)$ is the same as applying the transformation to $\mathbf{p}_{i,j}$. The surface $\mathbf{s}(u,v)$ is also a convex combination of $\mathbf{p}_{i,j}$, or $\mathbf{s}(u,v)$ always lies in the convex hull defined by the points $\mathbf{p}_{i,j}$. A detailed description of the mathematical properties of parametric surfaces can be found in Ref. 2.

BEZIER SURFACE

A Bezier surface is defined with a polygon mesh of control vertices $\mathbf{b}_{ij,}$ $i = 0,\ldots,n, j = 0,\ldots,m$ shown in Fig.10. Using the de Casteljau algorithm (2), a point on the surface defined by $\mathbf{b}_{i,j}$ is obtained by applying a series of linear interpolations on the control points iteratively. P. Bezier (3) adopted a different approach and expressed the surface explicitly as

$$\mathbf{s}(u,w) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{b}_{i,j}J_{n,i}(u)K_{m,j}(v) \tag{17}$$

where $0 \le u \le 1, 0 \le v \le 1$, and $J_{n,i}(u)$, $K_{m,j}(v)$ are Bernstein basis functions

$$J_{n,i}(u) = \binom{n}{i}u^i(1-u)^{n-i}, \quad \binom{n}{i} = \frac{n!}{i!(n-i)!}$$

$$K_{m,j}(v) = \binom{m}{j}v^j(1-v)^{m-j}, \quad \binom{m}{j} = \frac{m!}{j!(n-j)!}$$

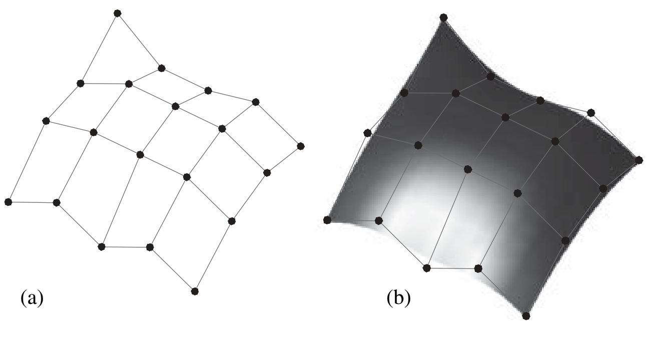


Figure 10. A Bezier surface and its control polygon mesh: (a) control polygon mesh and (b) the Bezier surface.

As $J_{n,i}(u)$, $K_{m,j}(v)$ are Bernstein basis functions $\sum_{i=0}^{n} J_{n,i}(u) \equiv 1$, and $\sum_{j=0}^{m} K_{m,j}(v) \equiv 1$; hence, $\sum_{i=0}^{n}\sum_{j=0}^{m} J_{n,i}(u) K_{m,j}(v) \equiv 1$. A Bezier surface is thus a barycentric combination of its control polygon vertices $\mathbf{b}_{i,j}$ and is determined by the basic properties of a tensor product surface as discussed in the previous section.

The degree of the surface in each parametric direction is determined by the number of control polygon vertices in that direction. The degree of the surface in the u direction is n, and the degree of the surface in the v direction is m. The continuity of the surface is thus C^{n-1} and C^{m-1} in the u and v directions, respectively. A Bezier surface passes through the corner points of the defining polygon mesh, and its shape generally follows the shape of the defining polygon mesh.

Consider a Bezier surface $\mathbf{s}_b(u, w)$ constructed with a 4×4 polygon mesh; i.e., $\mathbf{s}^b(u,v) = \sum_{i=0}^{4}\sum_{j=0}^{4} \mathbf{b}_{i,j} J_{4,i}(u) K_{4,j}(v)$. Expending and converting into matrix form gives

$$\mathbf{s}^b(u,w) = \mathbf{U}\mathbf{N}_b\mathbf{B}\mathbf{N}_b{}^{\mathrm{T}}\mathrm{V} \tag{18}$$

where

$$\mathbf{U} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}^{\mathrm{T}}$$

$$\mathbf{N}_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{b}_{0,0} & \mathbf{b}_{0,1} & \mathbf{b}_{0,2} & \mathbf{b}_{0,3} \\ \mathbf{b}_{1,0} & \mathbf{b}_{1,1} & \mathbf{b}_{1,2} & \mathbf{b}_{1,3} \\ \mathbf{b}_{2,0} & \mathbf{b}_{2,1} & \mathbf{b}_{2,2} & \mathbf{b}_{2,3} \\ \mathbf{b}_{3,0} & \mathbf{b}_{3,1} & \mathbf{b}_{3,2} & \mathbf{b}_{3,3} \end{bmatrix}$$

If this 4×4 Bezier surface is the same as a bicubic surface, then from Equation (15)

$$\mathbf{U}\mathbf{N}_c\mathbf{G}\mathbf{N}_c{}^{\mathrm{T}}\mathbf{V} = \mathbf{U}\mathbf{N}_b\mathbf{B}\mathbf{N}_b{}^{\mathrm{T}}\mathbf{V}$$

and hence,

$$\mathbf{G} = \mathbf{N}_c^{-1}\mathbf{N}_b\mathbf{B}\mathbf{N}_b^{\mathrm{T}}[\mathbf{N}_c^{\mathrm{T}}]^{-1}$$

This gives

$$\mathbf{G} = \begin{bmatrix} \mathbf{b}_{0,0} & \mathbf{b}_{0,3} & 3(\mathbf{b}_{0,1} - \mathbf{b}_{0,0}) & 3(\mathbf{b}_{0,3} - \mathbf{b}_{0,2}) \\ \mathbf{b}_{3,0} & \mathbf{b}_{3,3} & 3(\mathbf{b}_{3,1} - \mathbf{b}_{3,0}) & 3(\mathbf{b}_{3,3} - \mathbf{b}_{3,2}) \\ 3(\mathbf{b}_{1,0} - \mathbf{b}_{0,0}) & 3(\mathbf{b}_{1,3} - \mathbf{b}_{0,3}) & 9(\mathbf{b}_{0,0} - \mathbf{b}_{1,0} - \mathbf{b}_{0,1} + \mathbf{b}_{1,1}) & 9(\mathbf{b}_{0,0} - \mathbf{b}_{1,0} - \mathbf{b}_{0,1} + \mathbf{b}_{1,1}) \\ 3(\mathbf{b}_{3,0} - \mathbf{b}_{2,0}) & 3(\mathbf{b}_{3,3} - \mathbf{b}_{2,3}) & 9(\mathbf{b}_{2,0} - \mathbf{b}_{3,0} - \mathbf{b}_{2,1} + \mathbf{b}_{3,1}) & 9(\mathbf{b}_{2,2} - \mathbf{b}_{3,2} - \mathbf{b}_{2,3} + \mathbf{b}_{3,3}) \end{bmatrix}$$

The control polygon vertices on the boundary thus specify the positions and tangent vectors at the corner of the surface, whereas the interior polygon vertices control the twist at the corner of the surface.

B-SPLINE SURFACE

A B-spline surface is defined with a mesh of control polygon vertices $\mathbf{b}_{i,j}$, $i = 0, \ldots, n$, $j = 0, \ldots, m$:

$$\mathbf{s}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{b}_{i,j} N_{i,k}(u) N_{j,l}(v) \tag{19}$$

where k and l are the orders of the surface in the u and v directions, respectively; u and v lie in the range defined by the knot sequences $\{u_0, u_1, \ldots, u_{n+k+1}\}$ and $\{v_0, v_1, \ldots, v_{m+l+1}\}$, respectively, such that $u_i \le u_{i+1}$, $v_j \le v_{j+1}$, and

$$\begin{aligned} N_{i,k}(u) &= \frac{(u - u_i)N_{i,k-1}(u)}{u_{i+k-1} - u_i} + \frac{(u_{i+k} - u)N_{i+1,k-1}(u)}{u_{i+k} - u_{i+1}}, \\ N_{i,k}(u) &= \begin{cases} 1 & \text{if } u_i \le u \le u_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ N_{j,l}(v) &= \frac{(v - v_j)N_{j,l-1}(v)}{v_{j+l-1} - v_j} + \frac{(v_{j+l} - v)N_{j+1,l-1}(v)}{v_{j+l} - v_{j+1}}, \\ N_{j,l}(v) &= \begin{cases} 1 & \text{if } v_j \le v \le v_{j+1} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The basis functions are partition of unity; i.e.,$\sum_{i=0}^{n} N_{i,k}(u) \equiv 1$, $\sum_{j=0}^{l} N_{j,l}(u) \equiv 1$, and hence, $\sum_{i=0}^{n}\sum_{j=0}^{l} N_{i,k}(u) N_{j,l}(v) \equiv 1$. It follows that a B-spline surface is a barycentric combination of its control polygon vertices $\mathbf{b}_{i,j}$ and thus possesses the basic properties of a free-form surface.

In general, three types of knot sequences affect the shape of the surface. These knot sequences are the periodic, open uniform, and open nonuniform sequence. Consider the knot sequence in the u direction. The knot sequence is Periodic, if $\Delta u = u_{i+1} - u_i$ is a constant. In an open uniform knot sequence, assuming Δu is a constant,

$$\begin{aligned} u_i &= u_0, & 1 &\le i \le k \\ u_i &= u_{i-1} + \Delta u, & k + 1 &\le i \le n + 2 \\ u_i &= u_{n+2}, & n + 3 &\le i \le n + k + 1 \end{aligned}$$

For an open nonuniform knot sequence,

$$\begin{aligned} u_i &= u_0, & 1 &\le i \le k \\ u_i &\le u_{i+1}, & k + 1 &\le i \le n + 2 \\ u_i &= u_{n+2}, & n + 3 &\le i \le n + k + 1 \end{aligned}$$

The use of a periodic knot sequence results in a surface that does not pass through the corner vertices. Figure 11(b) shows an example in which a B-spline surface is constructed with a periodic knot sequence in both directions. Figure 11(a) shows the B-spline surface using a periodic knot sequence in one direction, and an open uniform knot sequence in the other direction. Figure 11(c) illustrates a

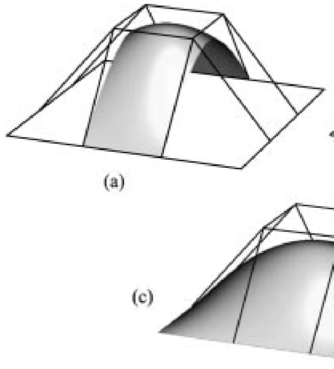


Figure 11. A B-spline surface: (a) using periodic knot sequence in one direction, (b) using periodic knot sequence in both directions, and (c) using open uniform knot sequences in both directions.

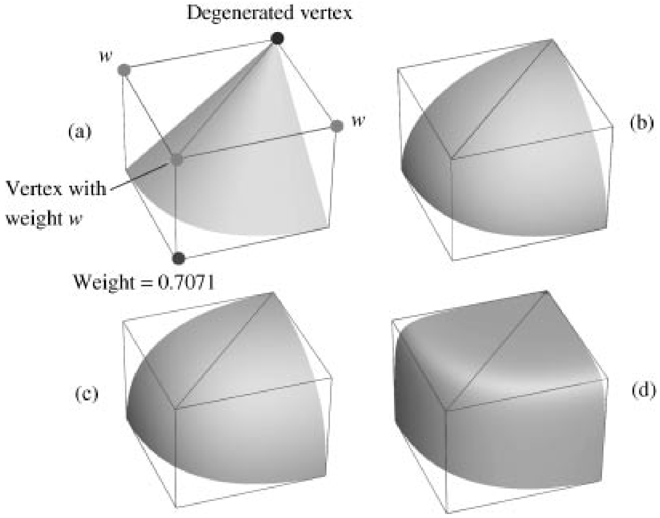


Figure 12. Constructing Quadric surfaces with NURBS: (a) sector of a cone, $w = 0$,(b) sector of a paraboloid, $w = 1$,(c) sector of a sphere, $w = 0.7071$, and (d) sector of hyperboloid, $w = 10$.

B-spline surface using an open uniform knot sequence in both directions.

The maximum order of a B-spline surface in each parametric direction is the number of control polygon vertices in that direction. Given a B-spline surface $\mathbf{s}(u, v)$ of order k in the u -direction and order l in the v direction, the maximum continuity of $\mathbf{s}(u, v)$ is C^{k-2} in the u direction, and C^{l-2} in the v direction. A B-spline surface constructed with open uniform knot sequences in both u and v directions reduces to a Bezier surface when the orders of the surface in the u and v directions are the same as the number of control polygon vertices in the corresponding directions.

RATIONAL B-SPLINE SURFACE

A rational B-spline surface (4) is an extension of a B-spline surface and is thus defined over a mesh of control polygon vertices. A rational B-spline surface constructed with non-uniform knot sequences is commonly referred to as a non-uniform rational B-spline surface (NURBS). Using the same notations as in the description of B-spline surface, a rational B-spline surface is expressed as

$$\mathbf{s}(u,v) = \frac{\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{m} w_{i,j}N_{i,k}(u)N_{j,l}(v)\mathbf{b}_{i,j}}{\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{m} w_{i,j}N_{i,k}(u)N_{j,l}(v)} \tag{20}$$

where $w_{i,j}$ is a weight associated with a control polygon vertex $\mathbf{b}_{i,j}$. By setting all $w_{i,j}$ the same, a rational B-spline surface reduces to a B-spline surface. The use of a weight for each polygon vertex provides extra control over the shape of the surface. With suitable weight adjustment and knot sequence, a rational quadric B-spline surface can be reduced to a quadric surface. Figure 12 shows a rational B-spline surface constructed with a 3×3 control polygon mesh. A third-order open uniform basis function is assumed for both parametric directions of the surface. The weights associated with all control vertices except those indicated in Fig. 12(a) are set to one. By adjusting the weights w associated with the vertices as indicated, various quadric surfaces can be obtained as shown in Fig. 12(b)–(d). In general, increasing the weight of a vertex pulls the surface toward the vertex.

The shape of a Bezier, B-spline, or rational B-spline surface can be easily adjusted by modifying the order of the basis function and the knot sequence and by relocating the control polygon vertices. Given a surface of order $k \times l$, a sharp edge exists on the surface if the multiplicity of the vertices on a row of the mesh is $k - 1$, or the multiplicity of the vertices on a column of the mesh is $l - 1$ (Fig.13).

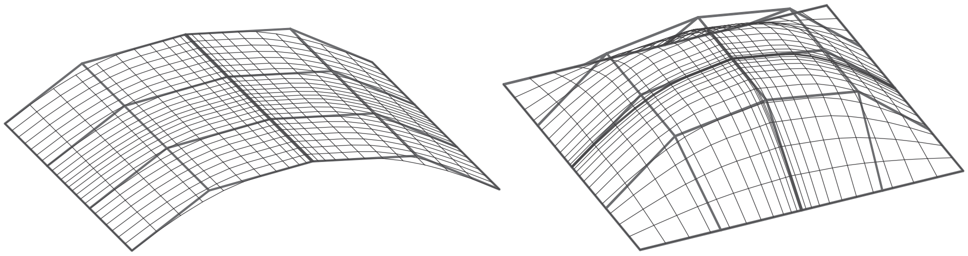


Figure 13. A fourth-order B-spline surface with multiple coincident net lines.

Multiplicity of a vertex refers to the number of repeated occurrence of the vertex at the same location.

COMPOSITE PATCHES AND CONTINUITY

Objects with a complex shape are usually constructed with a composite of surfaces. This process often requires maintaining the continuity between adjacent surface patches. There are two different types of continuity, namely, the parametric continuity and the geometric continuity. Parametric continuity refers to the continuity of derivatives across the common surface boundaries. Given two surfaces $\mathbf{g}(u,\ v)$, $0 \le u \le 1, 0 \le v \le 1$, and $\mathbf{h}(s,\ t)$, $0 \le s \le 1$, $0 \le t \le 1$, such that $\mathbf{g} = (1, v) = \mathbf{h}(0, t)$, and denotes $\mathbf{f}^n_u(u, v)$ as the n-th derivative of $\mathbf{f}(u, v)$ relative to u, then $\mathbf{s}$ and $\mathbf{r}$ are C^n continuous if $\mathbf{g}^n_u(1, v) = \mathbf{h}^n_s(0, t)$ and $\mathbf{g}^n_v(1, v) = \mathbf{h}^n_{\mathrm{t}}(0, t)$. That is, two surfaces are C^n continuous if the n-th derivates of their ccorresponding isoparametric lines are the same at the common boundary point.

In general, C^1 or C^2 is sufficient for most applications. In cases in which only the visual or geomentic continuity is considered, only the directions of the derivatives are considered. For instance, G^1 continuity between $\mathbf{g}$ and $\mathbf{h}$ requires continuity of the surface normal along the common boundaries, which can be expressed as

$$\mathbf{g}_u(1, v) \times \mathbf{g}_v(1, v) = \lambda \mathbf{h}_s(0, t) \times \mathbf{h}_t(0, t) \tag{21}$$

where λ is an arbitrary constant. A singularity arises when degenerated vertices exist in a control polygon mesh. For instance, in Fig. 14, the surface normal is not well defined at the degenerated vertex. In this case, a unique surface normal can only be guaranteed if all polygons sharing the degenerated vertex are coplanar.

SUBDIVISION SURFACE

Tensor product surfaces are in general restricted to model surfaces with a rectangular topology. To model objects with irregular topology, control polygons with degenerated vertices are usually required. Because of the singularity of the surface at the degenerated vertices, this is often undesirable. The subdivision surface, which was introduced in the late 1970s, is capable of modeling surfaces with irregular topology.

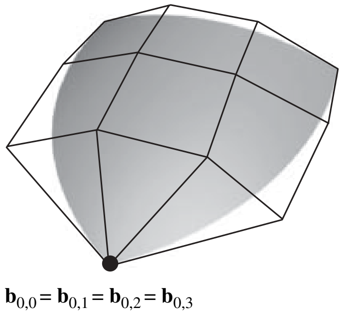

$\mathbf{b}_{0,0} = \mathbf{b}_{0,1} = \mathbf{b}_{0,2} = \mathbf{b}_{0,3}$

Figure 14. A surface with a degenerated vertex.

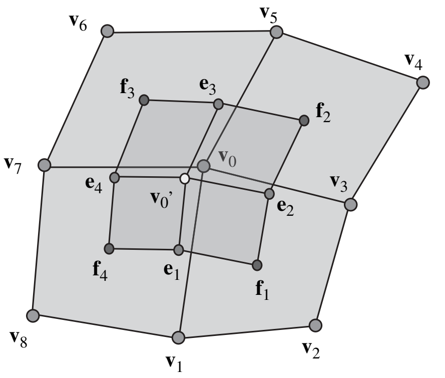


Figure 15. A Catmull–Clark subdivision surface.

Classic subdivision schemes include the Catmull–Clark scheme (5) that generates bicubic B-spline surfaces and the Doo–Sabin scheme (6) for producing quadric B-spline surfaces. Other popular subdivision schemes include the Loop scheme (7) that works on triangular meshes and the interpolative Butterfly scheme (8). A description of the Catmull–Clark scheme is given below. Details on the other schemes can be found in Ref. 9.

Consider the subdivision of a mesh as shown in Fig. 15 using the Catmull–Clark scheme. Assume there are four edges incident to a vertex $\mathbf{v}_0$. The mesh sharing $\mathbf{v}_0$ is subdivided by generating a set of face points, edge points, and vertex points. New face points are placed at the center of each face as expressed below:

$$\begin{aligned} \mathbf{f}_1 &= \frac{1}{4}(\mathbf{v}_0 + \mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3), \quad \mathbf{f}_2 = \frac{1}{4}(\mathbf{v}_0 + \mathbf{v}_3 + \mathbf{v}_4 + \mathbf{v}_5) \\ \mathbf{f}_3 &= \frac{1}{4}(\mathbf{v}_0 + \mathbf{v}_5 + \mathbf{v}_6 + \mathbf{v}_7), \quad \mathbf{f}_3 = \frac{1}{4}(\mathbf{v}_0 + \mathbf{v}_7 + \mathbf{v}_8 + \mathbf{v}_1) \end{aligned} \tag{22}$$

New edge points are inserted at

$$\begin{aligned} \mathbf{e}_1 &= \frac{1}{2}\left[\frac{(\mathbf{v}_0 + \mathbf{v}_1)}{2} + \frac{(\mathbf{f}_4 + \mathbf{f}_1)}{2}\right], \quad \mathbf{e}_2 = \frac{1}{2}\left[\frac{(\mathbf{v}_0 + \mathbf{v}_3)}{2} + \frac{(\mathbf{f}_1 + \mathbf{f}_2)}{2}\right] \\ \mathbf{e}_3 &= \frac{1}{2}\left[\frac{(\mathbf{v}_0 + \mathbf{v}_5)}{2} + \frac{(\mathbf{f}_2 + \mathbf{f}_3)}{2}\right], \quad \mathbf{e}_4 = \frac{1}{2}\left[\frac{(\mathbf{v}_0 + \mathbf{v}_7)}{2} + \frac{(\mathbf{f}_3 + \mathbf{f}_4)}{2}\right] \end{aligned} \tag{23}$$

and new vertex points are inserted at

$$\mathbf{v}_0' = \frac{\mathbf{Q}}{4} + \frac{\mathbf{R}}{2} + \frac{\mathbf{v}_0}{4} \tag{24}$$

where

$$\mathbf{Q} = \frac{1}{4}(\mathbf{f}_1 + \mathbf{f}_2 + \mathbf{f}_3 + \mathbf{f}_4)$$

and

$$\mathbf{R} = \frac{1}{4}\left(\frac{\mathbf{v}_0 + \mathbf{v}_1}{2} + \frac{\mathbf{v}_0 + \mathbf{v}_3}{2} + \frac{\mathbf{v}_0 + \mathbf{v}_5}{2} + \frac{\mathbf{v}_0 + \mathbf{v}_7}{2}\right)$$

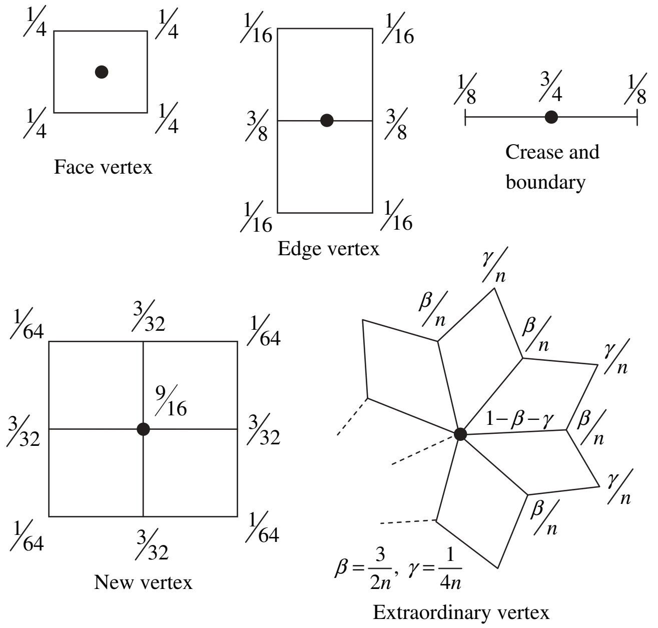


Figure 16. Masks for Catmull–Clark subdivision.

In general, for meshes with irregular mesh topology, a new face point is the average of all old points defining the face. A new edge points is the average of the midpoints of the old edge and the average of the two new face vertices of the faces sharing the edge. A new vertex point is obtained with the expression

$$\frac{\mathbf{Q}}{n}+\frac{2\mathbf{R}}{n}+\frac{\mathbf{S}(n-3)}{n} \tag{25}$$

where n is the number of edges sharing the vertex, $\mathbf{S}$ is the old vertex point, $\mathbf{Q}$ is the average of the new face points of all faces sharing the old vertex point, and $\mathbf{R}$ is the average of the midpoints of all old edges incident to the old vertex point.

An edge can be tagged as creased such that it is not smoothed in the subdivision process. In this case, a new face point remains the average of the vertices bounding the face. A new edge point is the midpoint of the edge. Assuming there are n creased edges incident to a vertex $\mathbf{v}_0$, the new vertex point of $\mathbf{v}_0$ is generated according to the following rules:

1. If $n \le 1$, the new vertex point is computed with Equation (25).
2. If $n = 2$, and the vertices of the creased edges are $(\mathbf{v}_0, \mathbf{v}_a), (\mathbf{v}_0, \mathbf{v}_b)$, the new vertex point is given by $\mathbf{v}_0' = \frac{1}{8}\mathbf{v}_a + \frac{3}{4}\mathbf{v}_0 + \frac{1}{8}\mathbf{v}_b$.
3. if $n{>}2$, then $\mathbf{v}_0' = \mathbf{v}_0$.

The subdivision rule for a subdivision scheme is often described using the mask notation.Figure 16 gives the mask notation for the Catmull–Clark subdivision rules. The new position of the highlighted vertex is given by the sum of the products of the mask weights and the corresponding vertices.

The number of edges incident to a vertex is usually referred to as the valency of a vertex. In a regular Catmull–Clark subdivision mesh, the valency is four. Vertices with valency other than four are called extraordinary vertices. By applying one subdivision to a base mesh, the mesh will be composed of all quadrilateral faces. The total number of extraordinary vertices is hence fixed after one subdivision. Figure 17 shows an example of a Catmull–Clark subdivision surface at different subdivision levels.

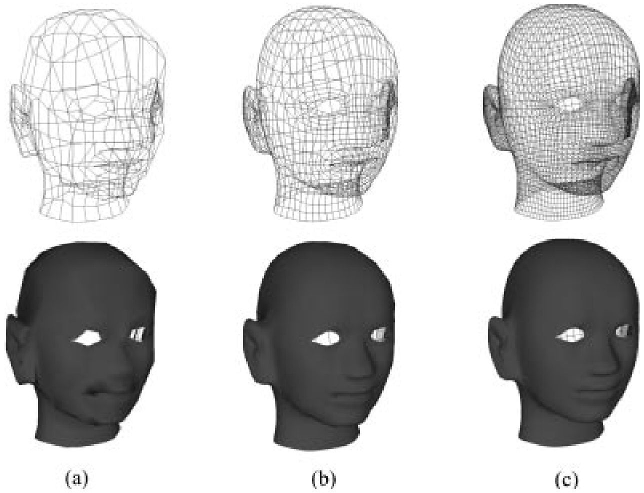


Figure 17. A head model constructed with the Catmull–Clark subdivision scheme: (a) base mesh, (b) subdivision level = 1, and (c) subdivision level = 2.

REMARKS

Surface modeling is an essential technique for the construction of 3-D objects. A wide variety of different surface construction techniques have been developed. The description given above in no way covers all different surfaces types. It only attempts to describe those popularly used surfaces and their characteristics. For other surfaces, e.g., Bezier triangles (2) and Gregory patch (10), details can be found in the corresponding references.

REFERENCE

1. I. D. Faux and M. J. Pratt, *Computational Geometry for Design and Manufacture*. Chichester, U.K.: Ellis Horwood, 1980.
2. G. Farin, *Curves and Surfaces for CAGD, a Practical Guide*, 4th ed. New York: Academic Press, 1997.
3. P. Bezier, *Numerical Control: Mathematics and Applications*. New York: Wiley, 1972.
4. L. Piegl and W. Tiller, *The NURBS Books*, 2nd ed. New York: Springer, 1997.
5. E. Catmull and J. Clark, Recursively generated B-spline surfaces on arbitrary topological meshes, *Comput.-Aided Design*, **9**, (6): 350–355, 1978.
6. D. Doo and M. Sabin, Behaviour of recursive subdivision surfaces near extraordinary points, *Comput-Aided Design*, **9**, (6): 356–360, 1978.
7. C. Loop, Smooth subdivision surfaces based on triangles, Master's Thesis, Saltlake city: University of Utah, Department of Mathematics, 1987.
8. N. Dyn, D. Levin, and J. A. Gregory, A butterfly subdivision scheme for surface interpolation with tension control, *ACM Trans. Graphics*, **9** (2) 160–169, 1990.
9. D. Zorin and P. Schroder, Subdivision for modeling and animation, *SIGGRAPH*, *2000 Course Notes*.
10. P. Charrot and J. Gregory, A pentagonal surface patch for computer aided geometric design, *Comput. Aided Geometric Design*, **1** (1): 87–94, 1984.

K.C. Hui
The Chinese University of
Hong Kong
Shatin, Hong Kong

VIRTUAL CLOTHING

INTRODUCTION

Virtual garment simulation is the result of a large combination of techniques that have dramatically evolved during the last decade. Unlike the mechanical models used for existing mechanical engineering for simulating deformable structures, several new challenges are associated with the highly versatile nature of cloth. The central pillar of garment simulation is the development of efficient mechanical simulation models that can reproduce accurately the specific mechanical properties of cloth. However, cloth is by nature highly deformable and specific simulation problems exist from this fact. First, the mechanical representation should be accurate enough to deal with the nonlinearities and large deformations occurring at any place in the cloth, such as folds and wrinkles. Moreover, the garment cloth interacts strongly with the body that wears it, as well as with the other garments of the apparel. This strong interaction requires advanced methods to detect efficiently the geometrical contacts constraining the behavior of the cloth, and to integrate them in the mechanical model (collision detection and response). All of these methods require advanced and complex computational methods where the most important issues are computation speed and efficiency. For real-time applications, however, only specific approximation and simplification methods allow the computation of garment animation, giving up some of the mechanical accuracy toward a more realistic.

Garment simulation, which started in the late 1980s with very simple models such as Weil's approach (1), has benefited from the increase in performance of computer hardware and tools as well as from the development of specific simulation technologies that have led to impressive applications not only in the field of simulation of virtual worlds but also in the fashion garment industry. In Fig. 1, we can see complex clothes models done from sketches of famous designers of the sixties.

EARLY DEVELOPMENTS IN VIRTUAL GARMENT SIMULATION

In the field of computer graphics, the first applications for mechanical cloth simulation appeared in 1987 with the work of Terzopoulos et al. (2,3) in the form of a simulation system that relies on the Lagrange equations of motion and elastic surface energy. Solutions were obtained through finite difference schemes on regular grids, which allowed simple scenes involving cloth to be simulated, such as the accurate simulation of a flag or the draping of a rectangular cloth. However, the first applications that simulated garments realistically started in 1990 (Fig. 2) with the considerations of many other technologies complementing cloth simulation (4,5), such as body modeling, body animation, and collision detection and response (6). These applications innovated the fashion industry by providing the first virtual system allowing virtual garment patterns to be sewed together around a character.

Since then, most developments were aimed toward optimizing the accuracy and efficiency of the methods for simulating cloth accurately and efficiently, along with the developments of actual applications and commercial products.

MECHANICAL MODELS

Two major stages are to be considered in the design of an accurate mechanical simulation system:

1. The characterization and measurement of the mechanical properties of the material to be simulated, which includes the identification of the main factors that characterize the material and their quantitative measurement through a set of mechanical parameters, or behavior curves, possibly with their analytical modelization.
2. The reproduction of these mechanical properties in a numerical resolution system that uses state-of-the-art numerical methods and algorithms to reproduce accurately the resulting static or the dynamic behavior.

MECHANICAL CHARACTERIZATION OF CLOTH

The mechanical properties of deformable surfaces can be grouped into four main families:

1. Elasticity: the internal stress resulting from a given geometrical strain.
2. Viscosity: the internal stress resulting from a given strain rate.
3. Plasticity: how the properties evolve according to the deformation history.
4. Resilience: the limits at which the structure will break.

Elastic properties are the main contributor of mechanical effects in the usual contexts where cloth objects are used. Deformations are often small and slow enough to make the effects of viscosity, plasticity, and resilience insignificant. One major hypothesis is that quasistatic models in the domain of elastic deformations will suffice for models intended to simulate the rest position of the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

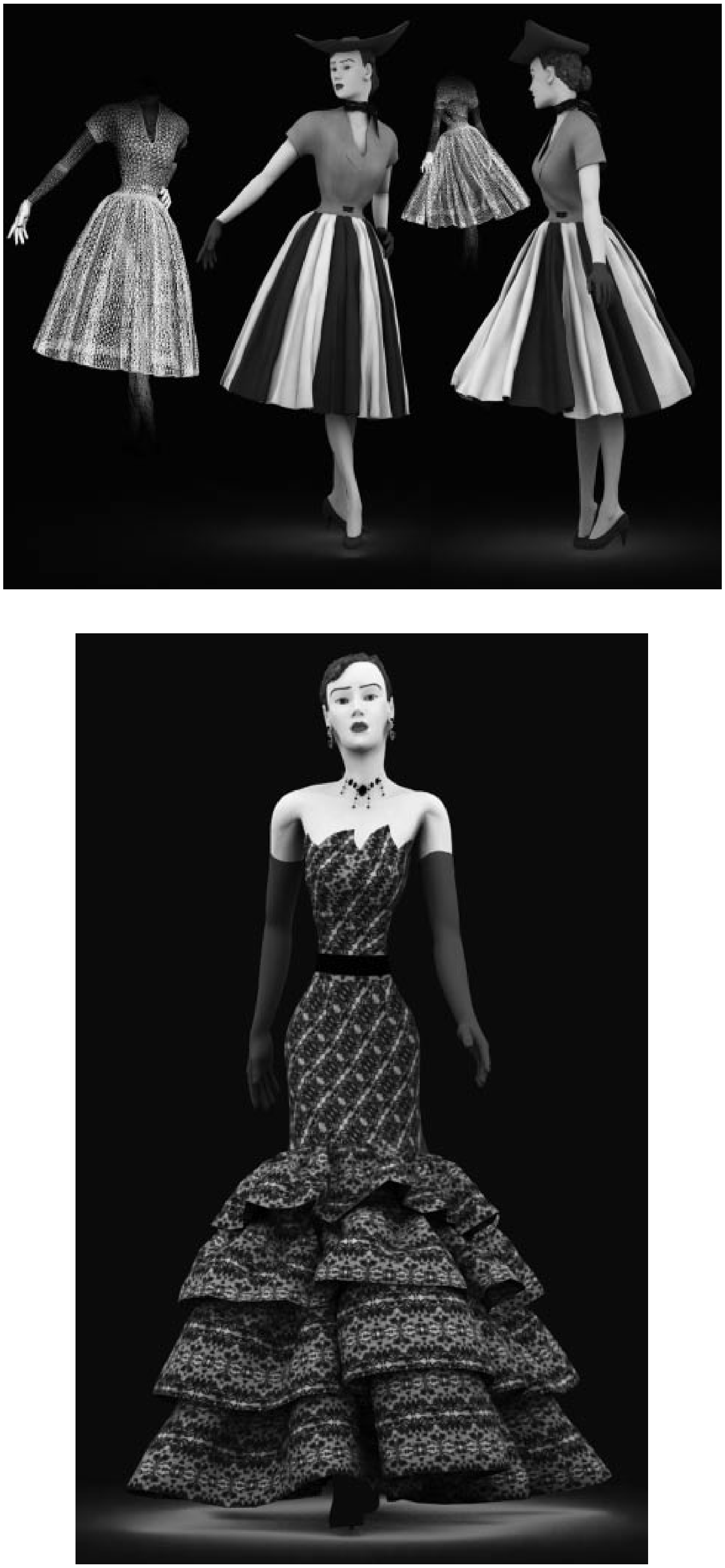


Figure 1. From the award-winning film "High Fashion in Equations," MIRALab–University of Geneva.

Figure 1. (*Continued*)

garment on an immobile mannequin (draping). However, when a realistic animation is needed, the parameters relating energy dissipation through the evolution of the deformation are also needed, and complete dynamic models, including viscosity and plasticity, should be used.

Depending on the amplitude of the mechanical phenomena under study, the curves expressing mechanical properties exhibit shapes of varying complexity. If the amplitude is small enough, these shapes may be approximated by straight lines. This linearity hypothesis is a common way to simplify the characterization and modeling of mechanical phenomena. However, in general, the nonlinear properties of cloth are captured through strain–stress curves. More complex models also consider more complex phenomena, such as the plasticity behavior, appearing as hysteresis in the strain–stress curves.

It is common in elasticity theory to consider that the orientation of the material has no effect on its mechanical properties (isotropy). This, however, is inappropriate for cloth, as its properties depend considerably on their orientation relative to the fabric thread.

Elastic effects can be divided into two main contributions:

1. Metric elasticity: deformations along the surface plane.
2. Bending elasticity: deformations orthogonally to the surface plane.

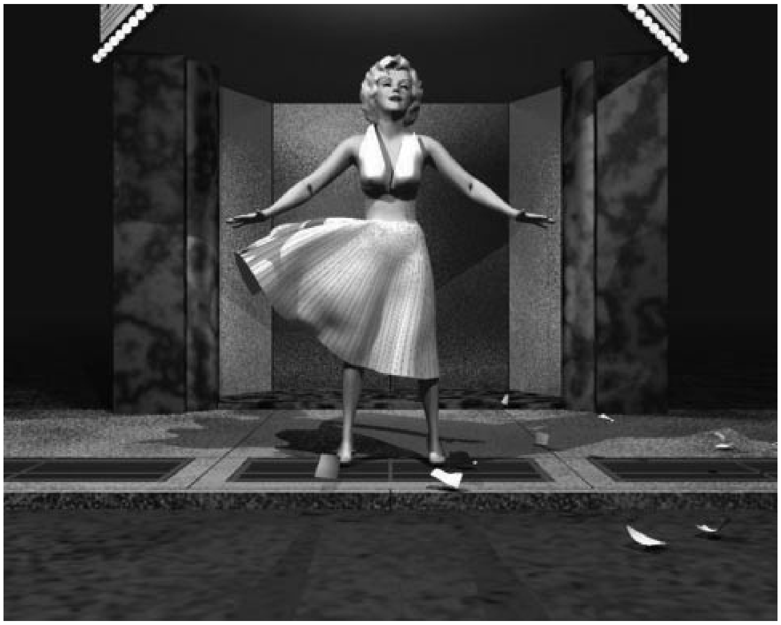


Figure 2. “FlashBack”: Early virtual garments used context-dependent simulation of simplified cloth models (image courtesy of MIRALab–University of Geneva).

The garment industry needs the measurement of major fabric mechanical properties through normalized procedures to guarantee consistent information exchange between garment industry and cloth manufacturers. The Kawabata evaluation system for fabric (KES) is a reference methodology for the experimental observation of the elastic properties of the fabric material. Using five experiments, 15 curves are obtained, which determine 21 parameters for the fabric.

Four standard tests are part of KES to determine the mechanical properties of cloth, using normalized measurement equipment (Fig. 3). The *tensile test* measures the force/deformation curve of extension for a piece of fabric of normalized size along weft and warp directions and allows the measurement of tensile elongation strain–stress behavior along with other parameters assessing nonlinearity and hysteresis. The *shearing test* is the same experiment using shear deformations, which allows the measurement of tensile shear strain–stress behavior. The *bending test* measures the curves for bending deformation in a similar way and measures the bending strain–stress behavior. Finally the *compression test* and the *friction test* measure parameters related to the compressibility and the friction coefficients. Allthough the KES measurements determine parameters assessing the nonlinearity of the behavior curves and some evaluation of the plasticity, other methodologies, such as the FAST method, use simpler procedures to determine the linear parameters only.

Whereas KES measurements and similar systems summarize the basic mechanical behaviors of fabric material, the visual deformations of cloth, such as buckling and wrinkling, are a complex combination of these parameters with other subtle behaviors that cannot be characterized and measured directly.

To take these effects into account, other tests focus on more complex deformations. Among them, the draping test considers a cloth disk of a given diameter draped onto a smaller horizontal disk surface. The edge of the cloth will fall around the support and produce wrinkling. The wrinkle pattern can be measured (number and depth of the wrinkles) and used as a validation test for simulation models.

Tests have also been devised to measure other complex deformation of fabric material, mostly related to bending, creasing, and wrinkling.

Methods for Mechanical Cloth Simulation

Continuum Mechanics Models. Well known in mechanical engineering, the finite element method measures the cloth surface as discretized in interpolation patches for a given order (bilinear, trilinear, quadrilinear) and measures an associated set of parameters (degrees of freedom) that give the actual shape to the interpolation surface over the element. From the mechanical properties of the material, the mechanical energy is computed from the deformation of the surface for given values of the interpolation parameters. An equation system based on the energy variation is then constructed with these degrees of freedom. Surface continuity between adjacent elements imposes additional constraint relationships. A large, sparse linear system is

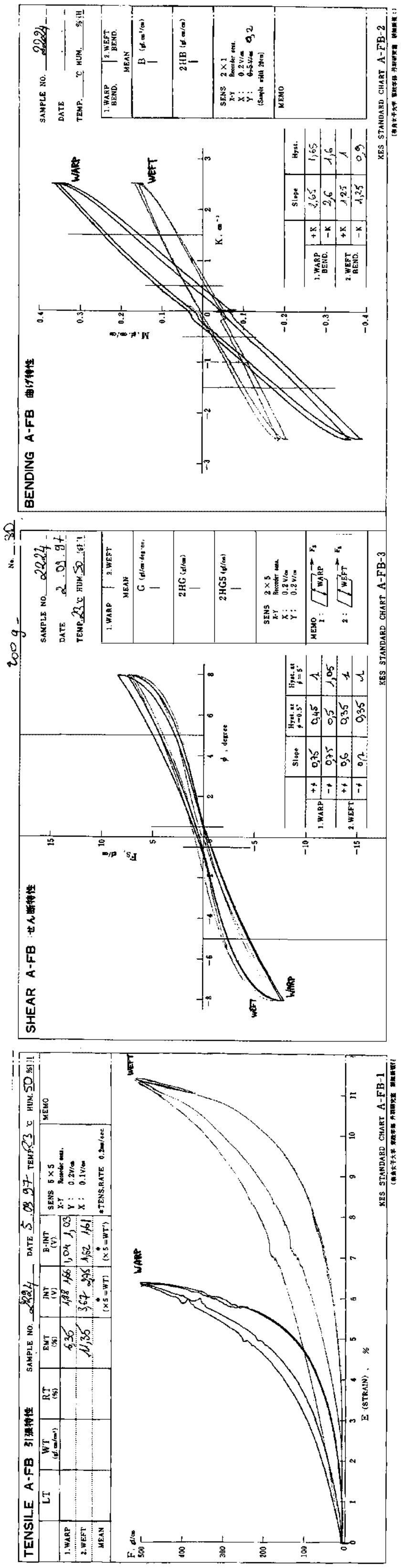


Figure 3. Examples of elongation, shear and bending strain–stress curves measured during KES evaluation of a fabric sample.

built by assembling successively the contributions of all elements of the surface and then it is solved using optimized iterative techniques, such as the conjugate gradient method.

In the beginning, finite elements had only a marginal role in cloth simulation. The main attempts are described in Refs. (7-9). Most implementations focus on the accurate reproduction of mechanical properties of fabrics, but they restrict the application field to the simulation of simple garment samples under elementary mechanical contexts, mostly because of the huge computational requirements of these models. Furthermore, accurate modeling of highly variable constraints (large nonlinear deformations, highly variable collisions) is difficult to integrate into the formalism of finite elements, and this sharply reduces the ability of the model to cope with the very complicated geometrical contexts that develop in real-world garment simulation on virtual characters.

Some recent developments have attempted to speed up the computation times required for finite elements. These developments have been used particularly in the context of interactive simulation for virtual surgery systems (10,11). For instance, preinverting the resolution matrix (as done in Ref. 12 for particle systems) may speed up the computation (13), but it restricts the application field to linear models and to very small mechanical systems (14). The “explicit finite elements” described in Ref. 15 come close to a good computational charge compromised by locally approximating the resolution of each element (16). These models also rely on simple linear formulations of the strain tensor that are inappropriate for the large deformations or displacements encountered in cloth simulation. Alleviating this problem through the use of a linearized form of the Green Lagrange strain tensor has led recently to more accurate models (10,17). A coordinate rotation is used to ensure accuracy of the linearized form in the context of large deformations (18-20). However, none of these methods has been implemented in the context of general nonlinear strain–stress mechanical behaviors.

Particle Systems. An easier and more pragmatic way to perform cloth simulation is to use of particle systems. Particle systems consider the cloth to be represented only by the set of vertices that constitute the polygonal mesh of the surface. These particles are moved through the action of forces that represent the mechanical behavior of the cloth, which are computed from the geometric relationships between the particles that measure the deformation of the virtual cloth. Among the different variations of particle systems, the spring-mass scheme is the simplest and most widely used. It considers the distance between neighboring particle pairs as the only deformation measurement and interaction source representing the internal elasticity of the cloth.

The simplest approach is to construct the springs along the edges of a triangular mesh describing the surface. This approach, however, leads to a model that cannot accurately display the anisotropic strain–stress behavior or the bending behavior of the cloth material. More accurate models are constructed on regular square particle grids describing the surface. Whereas elongation stiffness is modeled by springs along the edges of the grid, shear stiffness is modeled by diagonal springs, and bending stiffness is modeled by leapfrog springs along the edges. This model still is inaccurate because of the unavoidable cross-dependencies between the various deformation modes relative to the corresponding springs. Also, it is inappropriate for nonlinear elastic models and large deformations. More accurate variations of the model consider angular springs rather than straight springs to represent shear and bending stiffness, but the simplicity of the original spring-mass scheme is then lost (Fig. 4).

Particle systems are among the simplest and most efficient ways to define rough models that compute highly deformable mechanical systems, such as cloth, with computation times small enough to integrate them into systems to simulate complete garments on virtual bodies. Among the main contributions on particle system models, early works considered simple viscoelastic models on regular grids with applications for draping problems with simple

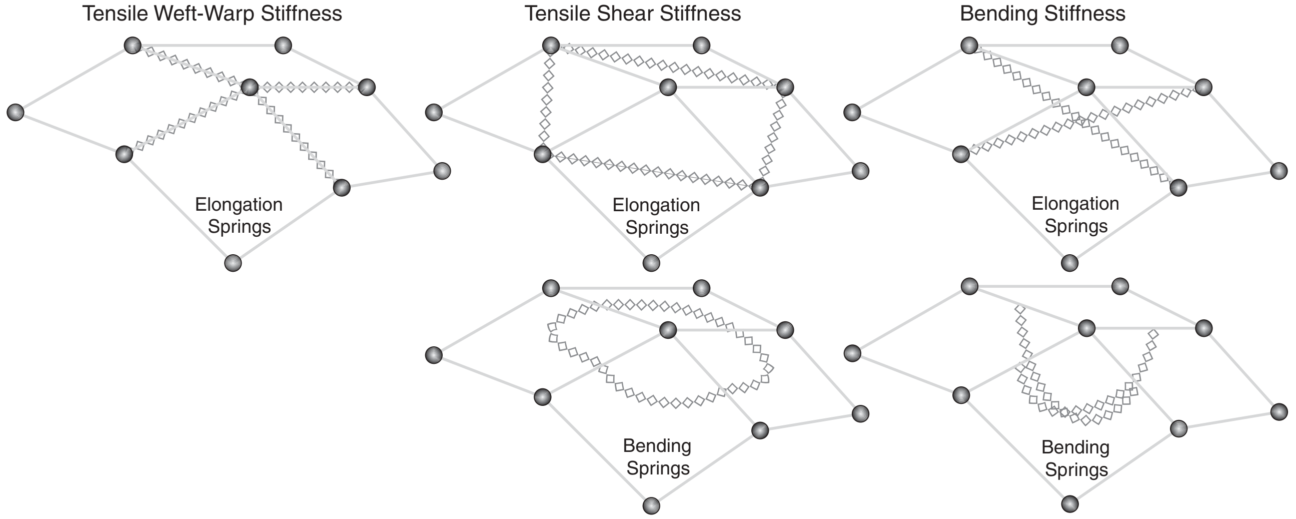


Figure 4. Various structures of spring-mass systems used in cloth simulation.

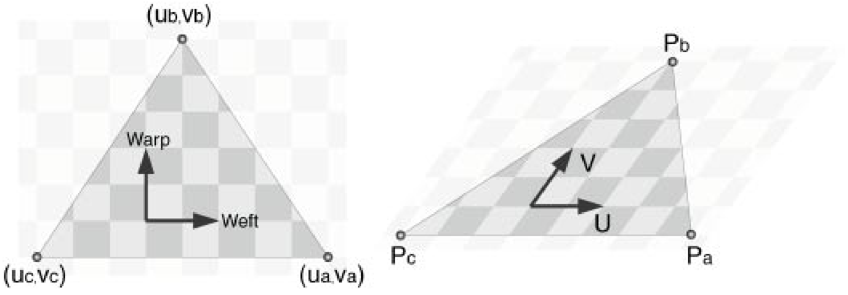


Figure 5. Accurate representation of tensile elasticity using particle systems: A triangle of cloth element defined on the 2-D cloth surface (left) is deformed in 3-D space (right), and its deformation state is computed from the deformation of its weft-warp coordinate system.

numerical integration schemes (21). Accurate models started with Breen et al. (22) modeling the microstructure of cloth using parameters derived from KES behavior curves and integration based on energy minimization. However, such accurate models required a lot of computation for solving problems that were restricted to draping. On the other hand, more recent models trade accuracy for speed, such as the grid model detailed by Provot (23), which includes geometric constraints for limiting large deformation of cloth. Additional contributions from Eberhardt et al. (24) include the simulation of KES parameters and the comparison of the efficiency between several integration methods. Advanced surface representations were used in Ref. 25, where the simulation model and collision detection takes advantage of the hierarchical structure of subdivision surfaces. Modeling animated garments on virtual characters is the specific aim of the work described by Volino et al. (26,27), which investigates improved spring-mass representations for better accuracy of surface elasticity modeling on irregular meshes. Finally, advanced particle systems that describe accurately the deformable behavior of elastic materials (17) also can describe accurately the anisotropic and nonlinear behavior of cloth materials (28). Such models can represent surface properties as accurately as first-order finite elements, offering a very good compromise between accuracy and computation speed (Fig. 5).

Simulating Bending Stiffness. Tensile stiffness only involves computing deformations and forces within mesh elements. On the other hand, the simulation of bending stiffness necessitates the action of out-of-plane forces between several adjacent mesh elements.

Several solutions have been proposed in the literature, representing two main approaches (Fig. 6). The first solution is to use crossover springs that extend the surface, opposing transversal bending (23,24). The second is to evaluate precisely the angle between adjacent mesh elements and to create between them normal forces that oppose this angle through opposite bending momentum (26,29–31). This approach can reach similar accuracy as grid continuum-mechanics (2) and grid particle system derivatives (32), which are complex to evaluate. Another solution proposed in Ref. (33) is to evaluate bending as a second-order discrete derivative vector computed linearly from the vertex positions, and to compute bending forces directly from it. It offers the advantage of a completely linear formulation that can efficiently be integrated by numerical algorithms.

Numerical Integration Methods. Although various models can be used to compute the force applied on each particle given their position and velocity, these forces must be integrated along time to obtain the position and velocity of the particle for the following time-steps using methods related to the integration of ordinary differential equation systems. Most recent contributions focus on improvements of the numerical integration methods in order to improve the efficiency of the simulation.

Explicit integration methods are the simplest methods available to solve first-order ordinary differential systems. They predict the future system state directly from the value of the derivatives. The best known techniques are the Runge–Kutta methods. Among them, the fast but unstable and inaccurate first-order Euler method, used in many

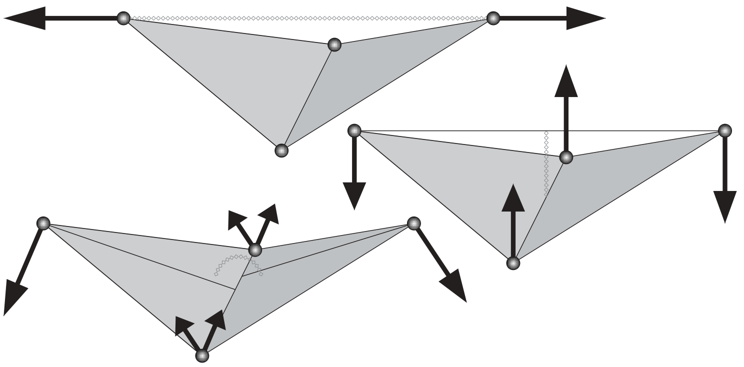


Figure 6. Three ways for creating bending stiffness in a triangle mesh: Using tensile crossover springs over mesh edges (top), using forces along triangle normals (bottom), and using forces linearly evaluated from a weighted sum of vertex positions (right).

early implementations, extrapolates directly the future state from the current state and from the derivative. Higher order and more accurate methods also exist, such as the second-order Midpoint method, used for instance in early models by Volino et al.(26), and the very accurate fourth-order Runge–Kutta method, used for instance by Eberhardt et al. (24).

In addition to considerations for accuracy, stability and robustness are other key factors to consider. For most situations encountered in cloth simulation, the numerical stiffness of the equations (stiff elastic forces, small surface elements) require the simulation time-steps to be small enough to ensure the stability of the system, and this limits the computation speed much more than accuracy considerations. Adequate time-step control is therefore essential for an optimal simulation. A common solution is to use the fifth-order Runge–Kutta algorithm detailed in Ref. (34) which embeds integration error evaluation used for tuning the time-step adaptively (29).

To circumvent the problem of instability, implicit numerical methods are used. For cloth simulation, this was outlined first by Baraff et al. (32). The most basic implementation of the implicit method is the Euler step, which finds the future state for which "backward" Euler computation would return the initial state. It performs the computation using not the derivative at the current time-step but the predicted derivative for the next time-step. Besides the inverse Euler method other, more accurate, higher order implicit methods exist, such as the inverse Midpoint method, which quite simple but exhibits some instability. A simple solution is to interpolate between the equations of the Euler and the Midpoint methods, as proposed by Volino et al. (40). Higher order methods, such as the Rosenbrook method, do not exhibit convincing efficiencies in the field of cloth simulation. Multistep methods, which perform a single-step iteration using a linear combination of several previous states, are good candidates for a accuracy–stability compromise. Among them, the second-order backward differential formula has shown some interesting performances, as used by Eberhardt,(35) Hauth et al. (36) and Choi et al. (37).

Whatever variation chosen, the major difficulty with implicit integration methods is that they involve the resolution of a large and sparse linear equation system for each iteration, constructed from the Jacobian matrix of the particle forces against their position and velocity. A commonly used simplification involves linearization of the mechanical model so as to obtain a linear approximation of the matrix that does not evolve along time, and on which initial construction and preprocessing allows an efficient resolution method to be used, as for example in Kang et al. (38), or even the matrix inverse to be precomputed as done by Desbrun et al. (12). Another simplification is to suppress completely the need for computing the matrix using an adapted approximation embedded directly in an explicit iteration. A big drawback of all these methods results from the approximation of the matrix that cannot take into account the nonlinearities of the model (mostly those resulting from the change in orientation of the surface elements during the simulation). Although this change is acceptable draping applications, animations usually behave poorly because of excessive numerical damping, which also increases as the time-step becomes large.

The best numerical method to resolve the linear system seems to be the conjugate gradient method, as suggested by Baraff et al. (32), with many variations and preconditioning schemes depending on how the mechanical model is formulated and on how the geometrical constraints of the cloth are integrated.

Most models that use implicit integration schemes restrict themselves to using spring-mass systems as their simple formulation to ease the process of defining the linear system to be resolved. However, implicit integration methods also can be used to integrate accurate surface-based particle systems as the one described above, from derivation of the particle force expressions relative to the particle positions and velocities (39). This approach is integrated simply into the implicit formulations described by Volino et al. (40) and is extended toward other advanced methods as by Hauth et al. (36). These formulations blur the line between particle systems and finite element methods, as the described particle system is indeed a first-order finite element method where the implicit resolution scheme corresponds to the energy minimization scheme of finite elements and the build of the linear system matrix corresponds to the assembly process of elements into the global system to be resolved. This key idea to design a new system combines the accuracy of finite elements with the efficiency of the techniques used for creating a particle system.

Collision Processing. Collision detection is one of the most time-consuming tasks when it comes to simulating virtual characters wearing complete garments (41). The usual complexity of collision detection processes result from the large number of primitives that describe these surfaces. Most of the collision detection applications need to compute which polygons of large meshes do actually collide. In most of the cases, these meshes are animated (through user interaction or simulation processes) and collision detection must be involved at each step of the animation process to ensure immediate and continuous feedback to the animation control.

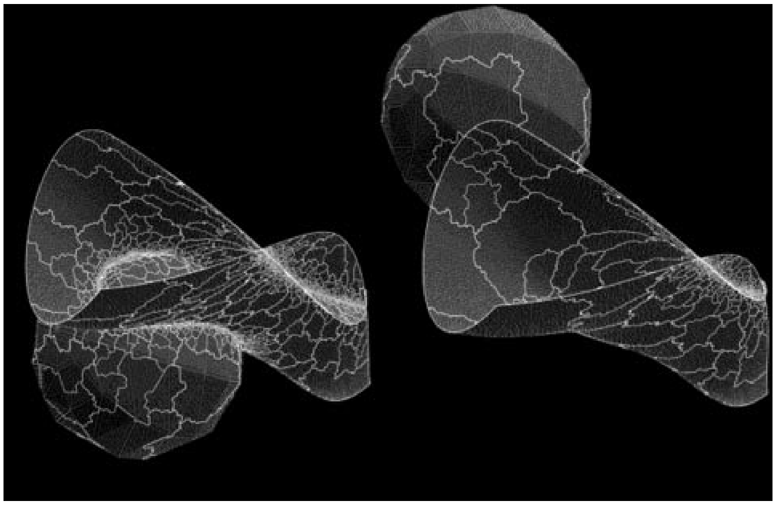
Most of the efficient collision detection algorithms take advantage of a hierarchical decomposition of the complex scheme, which allows for avoiding the quadratic time of testing extensively collisions between all possible couples of primitives. Two major ways of constructing such hierarchies are as follows:

1. Space subdivision schemes: the space is divided in to a hierarchical structure, typically octree methods. Using such structure, a reduced number of geographical neighbors of a given primitive are found in log(n) time (the depth of a hierarchy separating geographically n primitives) and tested for collisions against it.
2. Object subdivision schemes: the primitives of the object are grouped in to a hierarchical structure. These methods are based on bounding volume hierarchies. Using such structure, large bunches of primitives may

be discarded in log(n) time (the depth of a well-constructed hierarchy tree of n primitives) through simple techniques such as bounding-volume evaluations.

In the context of cloth simulation, object subdivision schemes are the most appropriate, as they take advantage of the constant topology of the mesh, which defines a near-constant, local geometric proximity relationship between mesh elements. This task is performed through an adapted bounding-volume hierarchy algorithm, which can use a constant discrete-orientation-polytope hierarchy constructed on the mesh and an optimization for self-collision detection using curvature evaluation on the surface hierarchy (29,42). This algorithm is fast enough to detect full collision and self-collision between all objects of the scene with acceptable impact on the processing time) (Fig. 7). Thus, body and cloth meshes are handled symmetrically by the collision detection process, ensuring perfect versatility of the collision handling between the body and the several layers of garments.

Collision response may either be force-based, using strong nonlinear penalty forces that simulate contact forces, or impulse-based, using a geometrical scheme based on correction of the mesh position, velocity, and acceleration. Allthough force-based models ensure good integration with the mechanical simulation process, the high nonlinearity of the forces degrades the performance of the numerical resolution. Impulse-based methods are more difficult to integrate into the mechanical model, but they offer a more controllable geometric constraint enforcement that does not affect the numerical resolution too much (27,40) (Fig. 8).



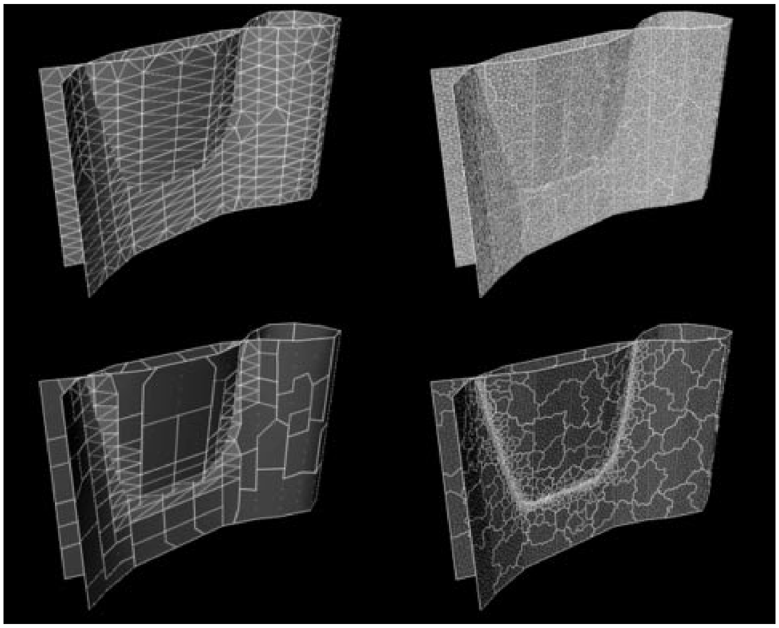


Figure 7. Collision detection using bounding-volume hierarchies: Only colliding regions are subdivided for detecting colliding mesh elements.

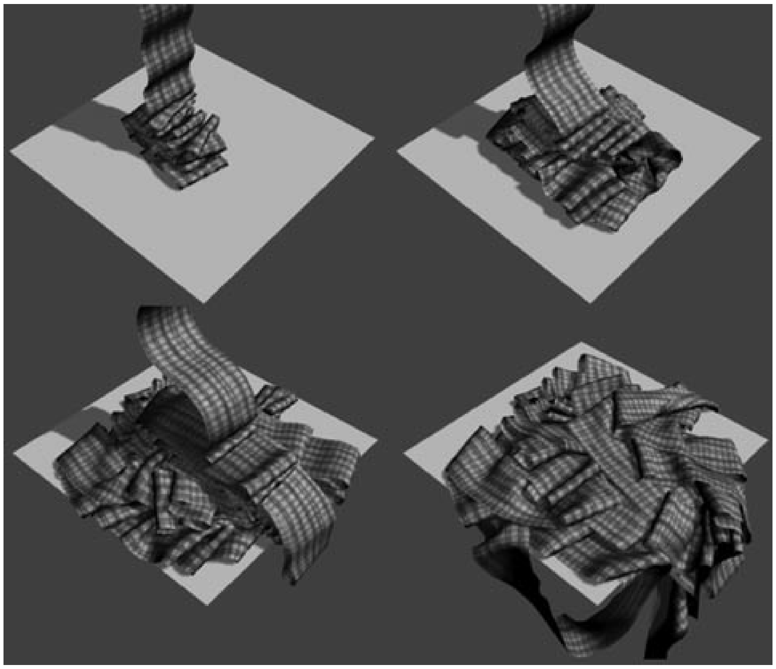


Figure 8. Advanced collision methods are required for solving such challenging situations in cloth simulation.

GARMENT DESIGN AND SIMULATION

Since the first developments of simulated garments on virtual characters (4,5), cloth simulation and garment animation has made its way not only into computer research (27), but also into commercial products aimed both for 3-D computer design and the garment industry.

A System for Pattern-Based Garment Design

Allthough essential, computational techniques alone are not sufficient to produce a powerful tool to allow accurate and convenient creation and prototyping of complex garments. All state-of-the-art techniques have to be integrated into a garment design and simulation tool aimed at prototyping and virtual visualization, to allow fashion designers to experiment virtually on new collections with high-quality preview animations, as well as to allow pattern makers to adjust precisely the shape and measurements of the patterns to fit the body optimally for best comfort (Fig. 9).

The most intuitive and natural approach for building garments takes its inspiration from the traditional garment industry, where garments are created from two-dimensional patterns and then seamed together (Fig. 10). Working with 2-D patterns is the simplest way to keep an accurate, precise, and measurable description and representation for a cloth surface. In the traditional

Figure 9. From the award-winning film “High Fashion in Equations,” MIRALab–University of Geneva.

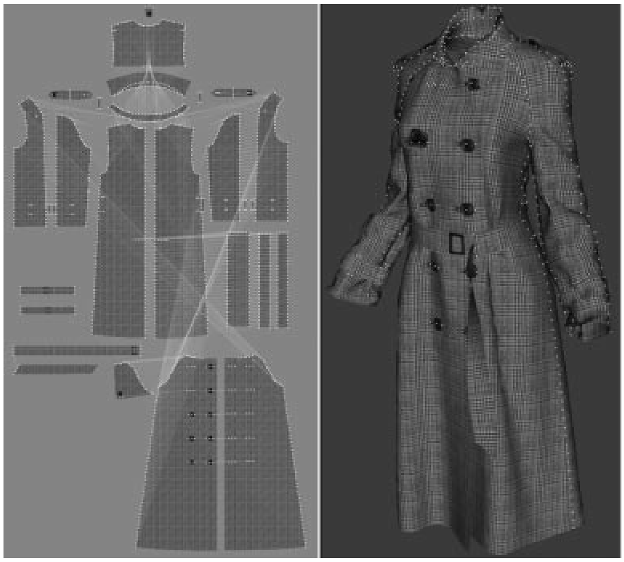


Figure 10. Between real and virtual: A garment design system should offer high-quality garment simulation, along with highly interactive pattern 2-D–3-D design and preview tools allowing complex garment models to be designed efficiently with many features such as seams, buttons, pockets, and belts.

garment and fashion design approach, garments usually are described as a collection of cloth surfaces, tailored in fabric material, along with the description of how these patterns should be seamed together to obtain the final garment. Many computer tools already are available in the garment industry for this purpose. A powerful virtual garment design system reproduces this approach by providing a framework to design the patterns accurately with the information necessary for their correct seaming and assembly. Subsequently, these patterns are placed on the 3-D virtual bodies and animated along with the virtual actor's motion.

Garment Prototyping

Combined with the accuracy and the speed of state-of-the-art mechanical simulation techniques, tasks such as comfortability evaluations are open to the garment designer, through the addition of several visualization tools, such as:

- Preview of fabric deformations and tensions along any weave orientation.
- Preview of pressure forces of the garment on the body skin.
- Immediate update of these evaluations according to pattern reshaping and sizing, fabric material change, and body measurements and posture change.

These tools allow the pattern designer to virtually test and adjust the measurements of complex garment patterns to the body size and postures of numerous different "virtual mannequins," assessing the strain and stress of the cloth, as well as the pressure exerted on the skin, assessing how the garment feels and slides on the body as it moves, and detecting eventual gesture limitations resulting from particular garment features (Fig. 11).

Commercial Products

Two kinds of virtual garment design products currently are available: those created for general cloth simulation and animation, and those specialized for draping and fitting garment models on virtual mannequins. The first category offers tools to simulate any kind of deformable surface

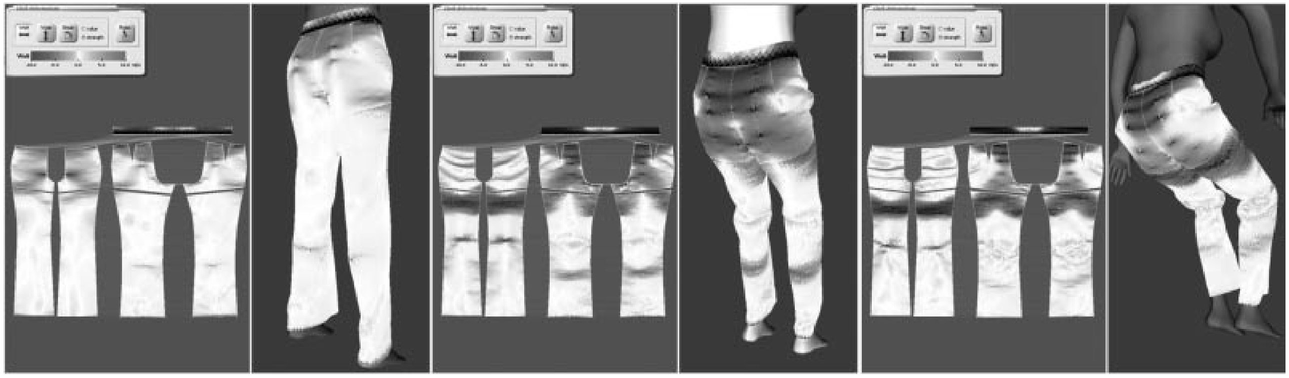


Figure 11. Virtual prototyping: Displaying weft constraints on an animated body (from standing to sitting).

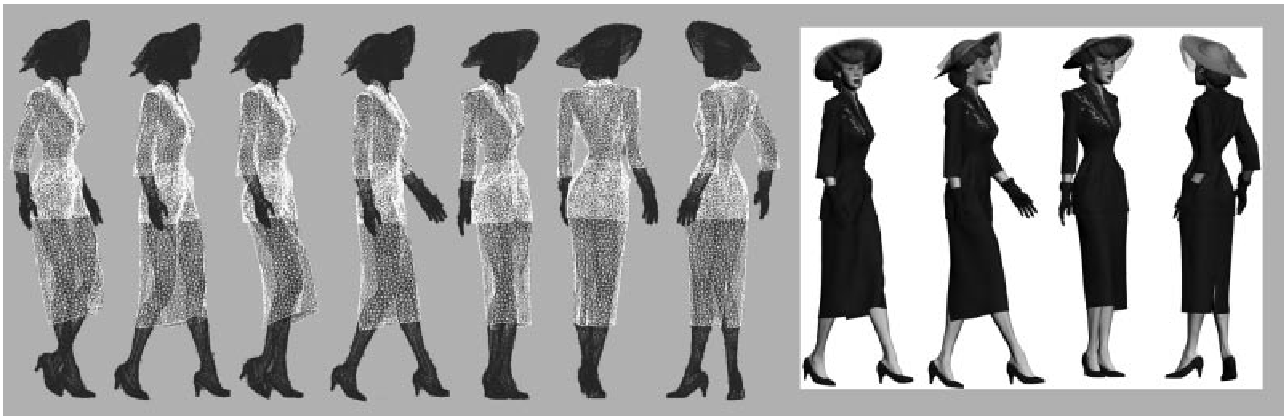


Figure 12. An animation sequence from the film “High Fashion in Equations.”

mechanically. These products usually offer a simple mechanical model containing only the basic mechanical parameters of cloth (stiffness, viscosity, bending, and gravity) modeled as a spring-mass particle system and simulated using state-of-the-art integration techniques. They allow the computation of realistic cloth animation, but they do not provide any tool for designing garments. Also these products offer general collision detection schemes for interaction with any other objects. These tools are usually integrated as plug-ins into 3-D design and animation frameworks.

The second category focuses on garment draping on virtual mannequins for visualization (virtual fashion, web applications) and prototyping purposes (garment design applications). The CAD applications specialize in the simulation of pattern assembly and of garment draping using accurate mechanical models of fabrics, whereas the visualization application takes advantage of geometric techniques for quickly generating realisticsally dressed mannequins out of design choices. Both applications use pattern models imported from professional pattern design tools. These tools also provide a stand-alone environment for setting up the simulation and for visualizing the results (Fig. 12).

BIBLIOGRAPHY

1. J. Weil, The synthesis of cloth objects, *Computer Graphics, SIGGRAPH 86 Conference Proceedings*, **20**: 49–54, 1986.
2. D. Terzopoulos, J.C. Platt, and H. Barr, Elastically deformable models, *Computer Graphics (SIGGRAPH'97 Proceedings)*, 1987, pp. 205–214.
3. D. Terzopoulos and K. Fleischer, Modeling inelastic deformation: viscoelasticity, plasticity, fracture, *Computer Graphics (SIGGRAPH'88 proceedings)*, 1988, pp. 269–278.
4. B. Lafleur, N. Magnenat-Thalmann, and D. Thalmann, Cloth animation with self-collision detection, *IFIP Conference on Modeling in Computer Graphics proceedings*, 1991, pp. 179–197.
5. M. Carignan, Y. Yang, N. Magnenat-Thalmann, and D. Thalmann, Dressing animated synthetic actors with complex deformable clothes, *Computer Graphics (SIGGRAPH'92 Proceedings)*, **26**(2): 99–104, 1992.
6. Y. Yang and N. Magnenat-Thalmann, An improved algorithm for collision detection in cloth animation with human body, *First Pacific Cont. on Computer Graphics and Applications*1993, pp. 237–251.
7. J.R. Collier, B.J. Collier, G. O'toole, and S.M. Sargand, Drape prediction by means of finite-element analysis, *J. Textile Institute*, **82** (1): 96–107, 1991.
8. L. Gan, N.G. Ly, and G.P. Steven, A study of fabric deformation using non-linear finite elements, *Textile Res. J.*, **65** (11): 660–668, 1995.
9. J.W. Eischen, S. Deng, and T.G. Clapp, Finite-element modeling and control of flexible fabric parts, *IEEE Computer Graph. Applicat.*, **16** (5): 71–80, 1996.
10. G. Desbrunne, M. Desbrun, M.P. Cani, and A.H. Barr, Dynamic real-time deformations using space & time adaptive sampling, *Computer Graphics (SIGGRAPH'01 proceedings)*, 2001, pp. 31–36.
11. M. Hauth, J. Gross, and W. Strasser, Interactive physically-based solid dynamics, *Eurographics Symposium on Computer Animation*, 2003, pp. 17–27.
12. M. Desbrun, P. Schröder, and A. Barr, Interactive animation of structured deformable objects, *Proceedings of Graphics Interface*, 1999.
13. M. Bro-Nielsen and S. Cotin, Real-time volumetric deformable models for surgery simulation using finite elements and condensation, *Eurographics 1996 proceedings*, 1996, pp. 21–30.
14. D. James and D. Pai, Accurate real-time deformable objects, *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, 1999, pp. 65–72.
15. J. O'Brien and J. Hodgins, Graphical modeling and animation of brittle fracture, *Computer Graphics (SIGGRAPH'99 Proceedings)*, ACM Press, 1999, pp. 137–146.
16. S. Cotin, H. Delingette, and N. Ayache, Real-time elastic deformations of soft tissues for surgery simulation, *IEEE Trans. Visualizat. Comp. Graph.*, **5**(1): 62–73, 1999.
17. O. Etzmuss, J. Gross, and W. Strasser, Deriving a particle system from continuum mechanics for the animation of deformable objects, *IEEE Trans. on Visualizat. and Comp. Graph.*, **9** (4): 538–550, 2003.
18. M. Muller and M. Gross, Interactive virtual materials, *Proceedings of Graphics Interface, Canadian Human-Computer Communications Society*, 2000, pp. 239–246.
19. M. Muller, J. Dorsey, L. Mcmillan, R. Jagnow, and B. Cutler, Stable real-time deformations, *Proceedings of the Eurographics Symposium on Computer Animation*, 2002, pp. 49–54.

20. O. Etzmuss, M. Keckeisen, and W. Strasser, A fast finite-element solution for cloth modeling, *Proceedings of the 11th Pacific Conference on -Computer Graphics and Applications*, 2003, pp. 244–251.

21. Y. Sakagushi, M. Minoh, and K. Ikeda, A dynamically deformable model of dress, *Trans. Society of Electron., Informat. Commun.*, 1991, pp. 25–32.

22. D.E. Breen, D.H. House, and M.J. Wozny, Predicting the drape of woven cloth using interacting particles, *Computer Graphics Proceedings*, 1994, pp. 365–372.

23. X. Provot, Deformation constraints in a mass-spring model to describe rigide cloth behavior, *Graphics Interface'95 proceedings*, 1995, pp. 147–154.

24. B. Eberhardt, A. Weber, and W. Strasser, A fast, flexible, particle-system model for cloth draping, *IEEE Computer Graphics and Applications*, **16** (5): 52–59, 1996.

25. T. Derose, M. Kass, and T. Truong, Subdivision surfaces in character animation, *Computer Graphics (SIGGRAPH'98 Proceedings)*, 1998, pp. 148–157.

26. P. Volino, M. Courchesne, and N. Magnenat-Thalmann, Versatile and efficient techniques for simulating cloth and other deformable objects, *Computer Graphics (SIGGRAPH'95 proceedings)*, 1995, pp. 137–144.

27. P. Volino and N. Magnenat-Thalmann, Developing simulation techniques for an interactive clothing system, *Virtual Systems and Multimedia (VSMM'97 proceedings)*, Geneva, Switzerland, 1997, pp. 109–118.

28. P. Volino and N. Magnenat-Thalmann, Accurate garment prototyping and simulation, *Computer-Aided Design Appl.*, **2** (5): 645–654, 2005.

29. E. Grinspun, A. Hirani, M. Desbrun, and P. Schröder, Discrete shells, *ACM Symposium on Computer Animation*, 2003.

30. R. Bridson, S. Marino, and R. Fedkiw, Simulation of clothing with folds and wrinkles, *Eurographics-SIGGRAPH Symposium on Computer Animation*, 2003, pp. 28–36.

31. B. Thomaszewski and M. Wacker, Bending models for thin flexible objects, *WSCG Short Commun. Proceedings*, **9** (1), 2006.

32. D. Baraff and A. Witkin, Large steps in cloth simulation, *Computer Graphics Proceedings*, **32**: 106–117, 1998.

33. P. Volino and N. Magnenat-Thalmann, Simple linear bending stiffness in particle systems, *SIGGRAPH-Eurographics Symposium on Computer Animation*, 2006.

34. W.H. Press, W.T. Vetterling, S.A. Teukolsky, and B.P. Flannery, *Numerical recipes in C*, 2nd ed., Cambridge, UK: Cambridge University Press, 1992.

35. B. Eberhardt, O. Etzmuss, and M. Hauth, Implicit-explicit schemes for fast animation with particles systems, *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, 2000, pp. 137–151.

36. M. Hauth and O. Etzmuss, A high performance solver for the animation of deformable objects using advanced numerical metds, *Eurographics 2001 proceedings*, 2001.

37. K.J. Choi, H.S. Ko, Stable but responsive cloth, *Computer Graphics (SIGGRAPH'02 Proceedings)*, 2002.

38. Y.M. Kang, J.H. Choi, H.G. Cho, D.H. Lee, and C.J. Park, Real-time animation technique for flexible and thin objects, *WSCG'2000 proceedings*, 2000, pp. 322–329.

39. P. Volino and N. Magnenat-Thalmann, Implicit midpoint integration and adaptive damping for efficient cloth simulation, *Computer Animation and Virtual Worlds*, **16** (3–4): 163–175, 2005.

40. P. Volino and N. Magnenat-Thalmann, Implementing fast cloth simulation with collision response, *Computer Graphics International Proceedings*, 2000, pp. 257–266.

41. J. Metzger, S. Kimmerle, and O. Etzmuss, Hierarchical techniques in collision detection for cloth animation, *J. WSCG*, **11** (2): 2003, 322–329.

42. P. Volino and N. Magnenat-Thalmann, Efficient self-collision detection on smoothly discretised surface animation using geometrical shape regularity, *Computer Graphics Forum (Eurographics'94 proceedings)*, **13** (3): 155–166, 1994.

FURTHER READING

T. Agui, Y. Nagao, and M. Nakajma, An expression method of cylindrical cloth objects-an expression of folds of a sleeve using computer graphics, *Trans. of Soc. Electron., Informat. and Communicat.*, **J73-D-II**: 1095–1097, 1990.

D. Baraff, A. Witkin, and M. Kass, Untangling cloth, *Computer Graphics Proceedings*, Addison-Wesley, 2003.

G. Bergen, Efficient collision detection of complex deformable models using AABB trees, *J. Graphics Tools*, **2** (4): 1–14, 1997.

R. Bridson, R. Fedkiv, and J. Anderson, Robust treatment of collisions, contact, and friction for cloth animation, *Computer Graphics Proceedings*, 2002.

U. Cugini and C. Rizzi, 3D design and simulation of men garments, *WSCG Workshop Proceedings*, 2002.

F. Cordier and N. Magnenat-Thalmann, Real-time animation of dressed virtual humans, *Eurographics 2002 Proceedings*, 2002.

F. Cordier, H. Seo, and N. Magnenat-Thalmann, Made-to-measure technologies for online clothing store, *IEEE Computer Graphics Appl.***23**: 38–48, 2003.

G. Debunne, M. Desbrun, M.P. Cani, and A. Barr, Adaptive simulation of soft bodies in real-time, *Computer Animation, Annual Conference Series*, IEEE Press, 2000.

S.A. Ehmann, M.C. Lin, Accurate and fast proximity queries between polyhedra using convex surface decomposition, *Computer Graphics Forum*, 2001, pp. 500–510.

A. Fuhrmann, C. Gross, and V. Luckas, Interactive animation of cloth including self-collision detection, *Journal of WSCG*, **11** (1): 141–148, 2003.

A. Fuhrmann, C. Gross, V. Luckas, and A. Weber, Interaction-free dressing of virtual humans, *Computer & Graphics*, **27** (1): 71–82, 2003.

B.K. Hind and J. Mccartney, Interactive garment design, *Visual Computer*, **6**: 53–61, 1990.

P. Hubbard, Approximating polyhedra with spheres for time-critical collision detection, *ACM Trans. Graphics*, **15** (3): 179–210, 1996.

S. Gottschalk, M.C. Lin, and D. Manosha, OOBTree: a hierarchical structure for rapid interference detection, *SIGGRAPH 96 Conference Proceedings*, 1996, pp. 171–180.

S. Hadap, E. Bangarter, P. Volino, and N. Magnenat-Thalmann, Animating wrinkles on clothes, *IEEE Visualization '99*. San Francisco, CA, 1999, pp. 175–182.

Y. M. Kang, J. H. Choi, H. G. Cho, and D. H. Lee, An efficient animation of wrinkled cloth with approximate implicit integration, *Visual Comp. J.*, **17** (3): 147–157, 2001.

Y.M. Kang and H.G. Cho, Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles, *Computer Animation 2000 proceedings*, 2002, pp. 203–211.

J.T. Klosowski, M. Held, and J.S.B. Mitchell, Efficient collision detection using bounding volume hierarchies of k-dops, *IEEE Trans. on Visualizat. Comp. Graph.*, **4** (1): 21–36, 1998.

T. Larsson, T. Akinine-Möller, Collision detection for continuously deformable bodies, *Proceedings of Eurographics*, Short Presentations, 2001, pp. 325–333.

M. Meyer, G. Debunne, M. Desbrun, and A. H. Barr, Interactive animation of cloth-like objects in virtual reality, *J. Visualizat. Comp. Animat.*, **12** (1): 1–12, 2001.

H. Ng and R.L. Grimsdale, GEOFF-A geometrical editor for fold formation, *Lecture Notes in Computer Science Vol. 1024*: Image Analysis Applications and Computer Graphic, New York: Springer-Verlag, 1995, pp. 124–131.

M. Oshita and A. Makinouchi, Real-time cloth simulation with sparse particles and curved faces, *Proceedings of Computer Animation*, Seoul, Korea, 2001.

T. Vassilev and B. Spanlang, Fast cloth animation on walking avatars, *Eurographics Proceedings*, 2001.

P. Volino and N. Magnenat-Thalmann, Fast geometrical wrinkles on animated surfaces, *WSCG'99 Proceedings*, 1999.

P. Volino and N. Magnenat-Thalmann, Comparing efficiency of integration methods for cloth simulation, *Computer Graphics International Proceedings*, 2001.

G. Zachmann, Minimal hierarchical collision detection, *Proc. ACM Symposium on Virtual Reality*, 2002, pp. 121–128.

C. Luible, P. Volino, and N. Magnenat-Thalmann, “High Fashion in Equations,” *International Conference on Computer Graphics and Interactive Techniques, ACM Siggraph 2007*, sketches, San Diego, session: Vogue, Article No. 36 and Film Selected at the electronic Theater, SIGGRAPH' 2007.

Pascal Volino
Christiane Luible
Nadia Magnenat-Thalmann
University of Geneva
Geneva, Switzerland

VOLUME GRAPHICS AND VOLUME VISUALIZATION

Volume graphics is concerned with graphics scenes, where models are defined using volume representations instead of, or in addition to, traditional surface representations. It is a study of the input, storage, construction, manipulation, display, and animation of volume models in a true three-dimensional (3-D) form. Its primary aim is to create realistic and artistic computer-generated imagery from graphics scenes comprising volume objects (see Figs. 1 and 2), and to facilitate the interaction with these objects in graphical virtual environments. A generalized specification of a volume model is a set of scalar fields, $\boldsymbol{F}_1(p)$, $\boldsymbol{F}_2(p)$, ..., $\boldsymbol{F}_\mathrm{k}(p)$, which define the geometrical and physical attributes of every point p in 3-D space. Scalar fields related to a specific attribute are usually grouped together to form a vector or tensor field. Unlike a surface model or a surface-bounded solid model, a volume model does not normally have an explicit geometric boundary and its physical attributes are not defined homogeneously within its bounding volume. As true 3-D representations of graphical models, volume representations possess more descriptive power than surface representation. They provide an effective means for modeling objects with complex internal structures (such as human bodies) as well as objects without well-defined geometry (such as fires and smoke). Many modern data acquisition technologies are capable of capturing volumetric attributes of such objects in volume representations, facilitating physically faithful modeling of the real world.

Volume visualization is also concerned with volume data representations that are used to store measured physical attributes of real-world objects and phenomena, or to represent computer-generated models and their attributes in volumetric forms. Although it is typical and conventional for volume datasets (such as in computed tomography) to correspond spatially to the 3-D physical world, it is also common in many visualization applications to use volume datasets to store nonspatial physical data as well as abstract information. As a volume representation gives a full 3-D description of everywhere in a 3-D volume, it is usually difficult to comprehend a volume dataset visually when it is projected directly onto a two-dimensional (2-D) display. We can appreciate such difficulties by imagining viewing a photograph (i.e., a 2-D dataset) horizontally at the eye level (i.e., a one-dimensional (1-D) projection). Therefore, the primary aim of volume visualization is to extract important information from volume data and convey such information visually to reviewers. This aim justifies deflection from creation of realistic imagery, which is the primary aim of volume graphics, and allows simplifications and embellishments, if they improve the desired understanding. Otherwise, in many ways, the subject of volume visualization encompasses most aspects of volume graphics. Nevertheless, the development of the subject has been heavily influenced by many applications, including medical imaging and scientific computation.

Since the emergence of computer graphics in the 1960s, visual realism and real-time interaction have been the two main driving forces behind its development. As for many objects in the real world we observe normally only their surfaces, in general, it is computationally more economic to deal with geometric specifications in surface representations by assuming empty or homogeneous object interiors. Hence, in traditional computer graphics, most existing modeling and rendering methods deal with graphics models specified as surfaces or surface-bounded solids, and this focus has led to the dominance of triangular meshes in the state-of-the-art graphics hardware and software. This collection of methods is often referred to as *surface graphics* techniques. The primary deficiencies of surface graphics include its inability to encapsulate the internal description of a model and the difficulties in modeling and rendering amorphous phenomena.

Various volumetric techniques, including hypertextures and clouds modeling, have been proposed to address the shortcomings of surface graphics. Driven by several applications, there have also been significant advances in volume visualization, yielding numerous methods for processing and rendering volume datasets, many of which can be performed interactively. Coupled with the rapid increase in the processing power and storage capacity of computers over the past few decades, volume visualization now provides an indispensable means in science, engineering, and medicine, assisting in the observation, measurement, modeling, experimentation, abstraction, and analysis of the physical world. In the meantime, volume graphics is offering some striking visual realism in commercial animation production, for instance, in the modeling and rendering of animal fur. These developments have led to beliefs that volume-based techniques have the potential to match and overtake surface-based techniques in computer graphics. In 1993, Kaufman, et al. (1) first outlined the framework of volume graphics as a subfield of computer graphics. Since then, considerable progress has been made in the field.

VOLUME MODELS AND DATA REPRESENTATIONS

A volume model represents a graphical object by defining its geometrical and physical attributes at every point p in 3-D Euclidean space $\mathbb{E}$ or a volumetric subdomain $\mathbb{D}(\mathbb{D} \subseteq \mathbb{E})$. With scalar fields as its underlying concept, it provides a consistent means for specifying the geometry and physical properties of a spatial entity intrinsically in a true 3-D manner. In particular, volume modeling represents conceptually an important extension to surface-based modeling by allowing the specification of the internal structures of objects and amorphous phenomena. All volume rendering integrals assume that optical properties of a volume model are not homogeneously defined in $\mathbb{D}$. Most volume

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

rendering algorithms are designed to handle volume models only.

Similar to surface representations, a volume model can be specified procedurally or using sampled datasets. Procedurally defined models typically allow more accurate computation of various geometric properties of the models, but they may not be suitable for representing complex real-world objects. Volume models that are defined upon sampled data have been the main focus of volume graphics and visualization, largely because several digitization technologies (see **ADVANCED TOPICS**) exist for acquiring various physical attributes in volume data representations. A collection of such data for representing a volume model is referred as a *volume dataset*. A scheme that defines the data types of data primitives and governs the inter-relationship between different types of data primitives is referred to as a *data representation*.

Spatially Sampled Data Representations

The basic notion of a *spatially sampled volume dataset* is a set of samples $\mathbf{V} = \{(p_i, v_i) | i = 1, 2, \ldots, n\}$, where v_i is a scalar value that represents some property (such as luminance intensity) at each sampling location p_i in 3-D Euclidean space $\mathbb{E}$. In most applications, we are only interested in a subdomain of $\mathbb{E}$ that encloses the set of sample points $p_i (i = 1, 2, \ldots, n)$ specified in $\mathbf{V}$. We denote such a subdomain as $\mathbb{D}(\mathbb{D} \subseteq \mathbb{E})$, and $\mathbb{D}$ in effect defines an *object domain*, that is, the valid spatial domain of a volume object. As the underlying model of the dataset $\mathbf{V}$ is a continuous scalar field $\boldsymbol{F}(p)$ defined in $\mathbb{D}$, it is necessary to define a scalar value for every point in $\mathbb{D}$, especially for those points that are not specified in $\mathbf{V}$. Here we consider only a single scalar field, and the concept can easily be generalized to multiple scalar fields. The typical methods for obtaining a specification of $\boldsymbol{F}(p)$ from a volume dataset depend mainly on the following three aspects:

1. The *interpolation function* that derives a value v at an arbitrary point p in $\mathbb{D}$ from several known point-value pairs. Hence it is necessary for such an interpolation function to have the knowledge of a subset of point-value pairs in $\mathbf{V}$, which should have influence on the value of any given point p in $\mathbb{D}$. A large number of interpolation functions require the subdomain $\mathbb{D}$ to be further divided into elementary volumes such that all sample points $p_i (i = 1, 2, \ldots, n)$ are only located at the boundary of these elementary volumes. This restricts the influence of each known point-value pair (p_i, v_i) to those elementary volumes, to which p_i is connected. Normally elementary volumes do not overlap with each other except along the shared boundaries. For some interpolation functions that do not require a nonoverlapping partition of $\mathbb{D}$, it is common to determine a subset of known point-value pairs in $\mathbf{V}$ for a given p in $\mathbb{D}$ based on proximity, or by simply including all point-value pairs in $\mathbf{V}$. Usually, a volume dataset is not fixed with a specific interpolation function, and an appropriate interpolation is normally selected at the rendering or resampling stage according to various application needs such as accuracy and performance.
2. The *geometrical positioning* of the set of sample points $p_i (i = 1, 2, \ldots, n)$ in $\mathbf{V}$. Such information may be defined *explicitly* or *implicitly* in a volume dataset. In those data representations with an implicit geometry specification, all sampling points are organized with regular and recurring elementary structures, such as a 3-D regular grid with cubic cells; and sampling locations can thereby be derived from some mathematical formulas.
3. The *topological relationship* or *connectivity* between the known sample points in $\mathbf{V}$. Such information may be defined *explicitly* or *implicitly* in a volume dataset. As mentioned, it is common to use an interpolation function in conjunction with a nonoverlapping spatial partition of $\mathbb{D}$. In an explicit connectivity specification, additional elementary structures, such as edges and cells, are included in a dataset, and they are normally defined by connecting known sample points in $\mathbf{V}$ to form an elementary structure. In an implicit specification, the elementary volumes are presupposed to have a regular and recurring structure in relation to the geometrical locations of the sample points in $\mathbf{V}$. In some cases, the connectivity information is neither explicitly nor implicitly defined for a dataset but requires to be derived *dynamically* from the geometrical information in $\mathbf{V}$.

Many volume data representations feature different types of specifications for geometrical positioning, topological connectivity, and interpolation function. Some of the most commonly used volume data representations are listed below.

3-D Regular Grid. This data representation is the most popular, where samples are taken at regularly spaced intervals along three orthogonal axes (e.g., the clouds in Fig. 1, the chess pieces in Fig. 2, the CT head in Fig. 3, and the four objects in Fig. 4). The sample points are commonly referred to as *voxels* (volume elements), as the 3-D analog of pixels. The straight lines, which link consecutive voxels in the three axial directions, collectively form a *regular grid*. Such a grid with a constant spacing in all three directions is said to be *isotropic* and, otherwise, *anisotropic*. The grid inherently subdivides its rectangular object domain $\mathbb{D}$ into many elementary cells in the form of cubes (in an isotropic grid) or cuboids (in an anisotropic grid). The value at any point inside such a cell is typically obtained using tri-linear interpolation of the values of the eight neighboring voxels. As the values at sample points can be stored by using a 3-D array (commonly called a *3-D raster* or a *volume buffer*) with implicit geometrical and topological specifications, this data representation is economic to store and efficient to process.

Note that the original definition of voxels, for instance, in *spatial occupancy enumeration* (see **VOLUME AND SURFACE**), implies that each voxel occupies a small cubic domain. Nevertheless, it is common nowadays to consider that a voxel is simply a discrete sample in a continuous volumetric domain.

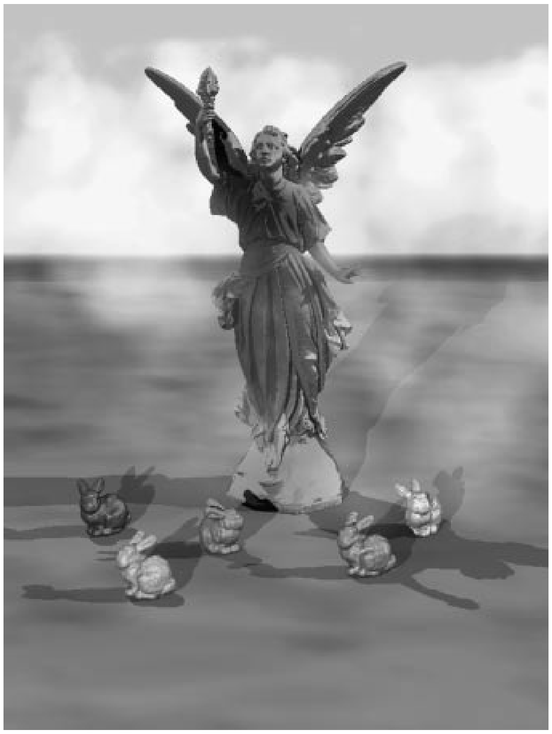


Figure 1. This is a volumetric scene, where the clouds (from the Universität Erlangen-Nürnberg) are represented by a 3D regular grid, and the Lucy and Bunny models (both from Stanford University) are point-based volume objects specified with radial basis functions. The image was produced by M. Chen (for details, see D. Chisnall, M. Chen and C. Hansen, "Ray-driven dynamic working set Rendering," *The Visual Computer*, **23**(3):167–179, 2007).

Tetrahedral Mesh. Data representations that require explicit geometrical and topological information are collectively referred to as *irregular grids*. One such data representations is a *tetrahedral mesh* that typically comprises a list of point-value pairs (as in the general notion) and a list of elementary cells in the form of tetrahedra. The convex hull of all sample points defines the valid spatial domain $\mathbb{D}$ of the dataset. Each tetrahedral cell is specified by four sampling points as its vertices, and the value of any point p inside the cell is typically determined by using the barycentric coordinates of p with respect to the four vertices.

Sometimes a volume dataset contains only a list of scattered samples without an explicit specification of topological connectivity. We can construct a tetrahedral mesh using a *tetrahedralization* algorithm that subdivides the convex hall of the given sample points into a set of tetrahedral cells with sample points as vertices. One such algorithm is the *3-D Delaunary triangulation* that ensures that no sample point falls inside the circumsphere of any tetrahedral cell.

Radial Basis Functions. Some volume data representations do not demand the partitioning of the object domain $\mathbb{D}$. One approach is to associate each point-value pair $(p_{i,} v_i)$ with a spherical *radial basis function* (RBF) that defines the influence of (p_i, v_i) upon any arbitrary point p in $\mathbb{E}$, which is normally in inverse proportion to the distance between p and p_i. Let $\boldsymbol{\omega}(p,\ p_i,\ r_i)$ be a radial basis function, where $r_i \in [0,\ \infty]$ is called the *radius of influence* that defines a sphere such that for any point p that falls outside of the sphere, $\boldsymbol{\omega}(p,\ p_i,\ r_i) = 0$. In many cases, a constant radius of influence is applied to all sample points; whereas in the others, each sample point is associated with an individual r_i that typically reflects the confidence or accuracy of the sampling process. A scalar field $\boldsymbol{F}(p)$ can therefore be - obtained as the sum of all $\boldsymbol{\omega}(p,\ p_i,\ r_i) \cdot v_i\ (i = 1,\ 2,\ \ldots,\ n)$ (e.g., the Lucy and Bunny models in Fig. 2). Many proposed radial basis functions can be used in conjunction with a volume data representation, including the Gaussian function that assumes an infinite radius of influence for every sample point, and several polynomial functions that approximate the Gaussian while facilitating the control of the radius of influence.

In general, it is not essential for a radial basis function to define the influence of p_i solely based on distance. Other considerations can be featured in $\omega(p,\ p_i,\ r_i)$. For example, when the set of voxels $\mathbf{V}$ is known to represent samples on a surface, one may use an ellipsoidal function to reduce the influence along the normal at p_i. We call a $\omega(p,\ p_i,\ r_i)$, with which the influence of p_i falls away from p_i at different rates in different directions, as an *anisotropic* or *nonuniform radial basis function*.

Nonspatial Data Representations

The 3-D Fourier transformation has been used to represent volume data in the frequency domain, which offers sensitive detection of spatial frequency components of the intensity variations along each axis. In the Fourier domain, an object with a given intensity texture will contribute the same Fourier frequency components independent of its position in the volume, thus enabling a unique shape description. It is also useful in constructing high-pass filters for boundary enhancement and low-pass filters for noise reduction. In addition to the qualities of Fourier transformation, 3-D wavelet transformation also facilitates a multiresolution decomposition and scale-invariant interpretation of volume data in the wavelet domain. The 3-D wavelets have successfully been used in several applications of volume visualization.

As a 3-D representation of spatial information, volume datasets may demand substantial storage space and are slow to navigate. Several compressed data representations have been proposed to overcome these difficulties.

VOLUME AND SURFACE

The most intrinsic representation of a volume model is a scalar field $\boldsymbol{F}(p)$ in 3-D Euclidean space $\mathbb{E}$. Conceptually we can consider $\mathbb{E}$ being the valid spatial domain of the volume model. The most intrinsic representation of a surface model, in a form related to the scalar field, is $\boldsymbol{F}(p) = \tau$ such that its valid spatial domain $\mathbb{D}$ contains only those points where $\boldsymbol{F}(p)$ is equal to a specific scalar value τ, that is, $\mathbb{D} = \{p | p \in \mathbb{E},\ \boldsymbol{F}(p) = \tau\}$. At least notionally, $\boldsymbol{F}(p)$ contains more information than $\boldsymbol{F}(p) = \tau$, whereas $\boldsymbol{F}(p) = \tau$ is an abstraction of $\boldsymbol{F}(p)$, hence potentially a more compact representation.

In traditional computer graphics, most existing modeling and rendering methods deal with graphics models specified as surfaces. Most graphics rendering pipelines are designed to support the display of surfaces. Surface representations, especially triangular meshes, are often the only acceptable form of input to many traditional graphics pipelines. Against this background, much of the early effort in volume visualization has been made to approximate a volume model by a surface model that can then be rendered using a surface-based graphics system. Such a rendering process is usually referred to as *indirect volume rendering*. There has also been effort for converting surface models to volume models in order to take advantage of the extensive collections of surface models available in the public domain, and to render such surface models using *direct volume rendering* (see **VOLUME RENDERING**).

Surface Extraction

Surface extraction is a process of generating a surface representation **S** from a volume representation **V**, where the underlying specification of **V** is a scalar field $\boldsymbol{F}(p)$, and that of **S** is $\boldsymbol{F}(p) = \tau$, which defines the set of all points in a scalar field with a specific scalar value τ. The scalar value τ is referred to as an *iso-value*, and $\boldsymbol{F}(p) = \tau$ an *iso-surface* (also called a *level surface* or a *a level-set*). Often the extraction process is also referred to as *iso-surfacing, surface reconstruction, surface tiling*, and *surface tracking*, some of which were used only in the context of a specific group of algorithms.

Notionally, deriving $\boldsymbol{F}(p) = \tau$ from $\boldsymbol{F}(p)$ seems to be a trivial process. In fact, both volume and surface representations, **V** and **S**, are normally defined in different forms of discrete data representations, and not all points on the iso-surface can easily be identified in **V** or stored in **S**. This gives rise to several groups of surface extraction algorithms.

Marching Cubes. Consider a common requirement for extracting an iso-surface in the form of a triangular mesh from a volume model in the form of a 3-D regular grid, where the tri-linear interpolation function is used to define the underlying scalar field. The most popular method for addressing this requirement is the *marching cubes algorithm* (2). Given a regular grid of $n_x \times n_y \times n_z$ voxels, and an iso-value τ, there are $(n_x - 1) \times (n_y - 1) \times (n_z - 1)$ cubic cells, each bounded by eight neighboring voxels. The algorithm examines these cubic cells one by one. For each cell, it first determines whether the iso-surface intersects with the cell. If there is an intersection, it creates a triangle or a few triangles to represent the part of the iso-surface within the cell. These triangles can then be organized into a triangular mesh (or a few disjoint meshes) to be displayed by a surface-based graphics system. As the iso-surface within such a cell is usually a curved surface because of the tri-linear interpolation, the triangular representation is mostly only an approximation.

The core of the marching cubes algorithm is to determine the number of triangles in each cell and their topological arrangement in relation to the cell boundary and to each other. There are 256 possible cases (including two cases of non-intersection), if we classify each cell by considering each of its eight voxels as either "$\geq \tau$" or "$< \tau$". Through three types of symmetrical transformations (i.e., complementary, rotational, and reflectional transformations), or a combination of a series of them, the 256 cases can be reduced to 14 basic topological cases, only 1 of which indicates that there is no intersection between the cell and the iso-surface.

However, what complicates the marching cubes algorithm is the fact that many basic cases are ambiguous; that is, the binary classification of the eight voxels alone does not always uniquely determine a topological structure for the triangular representation within the cell. Some ambiguous cases may have up to 7 possible variants and some may involve a less desirable structure called tunnels. In fact a similar but much simpler ambiguity problem exists in a class of 2-D contouring algorithms that extract contour lines from 2-D regular grids. The 2-D ambiguity can be resolved by a method called *asymptotic decider*, which analyzes the asymptotes of the hyperbola representing the bilinear interpolation of a square cell. One observation is that all ambiguous cubic cells involve one or more faces that are considered to be ambiguous in 2-D contouring. By applying the asymptotic decider to all ambiguous faces on a cube, one can determine the external edges of the triangles to be constructed. In some cases, these edges can adequately define the topology of these triangles, whereas in others, additional computation of some internal properties of the cell is necessary for further discriminatory analysis.

Extracting other Geometry Descriptions. In addition to surfaces, several algorithms have been developed for extracting a set of points on a medial surface and on a line-like skeleton representing the central axis of a volume model. Algorithms have also been proposed for extracting multiple iso-surfaces from volumetric datasets and, in particular, for constructing a tetrahedral mesh representing an *interval volume*, $\tau_1 \leq F(p) \leq \tau_2$, which is a collection of all iso-surfaces defined by iso-values in the range $[\tau_1, \tau_2]$. Such a representation is particularly useful in describing real-life surface structures that do not have the properties of perfect mathematical surfaces (e.g., zero or uniform thickness). It is also a fundamental data type used in rapid prototyping and, in particular, the layered manufacturing process.

Voxelization

Voxelization is a process for converting from a surface model (or a surface-bounded solid model) to a discrete volume data representation. The target volume data representation is usually in the form of a 3-D regular grid, and thus, the process is also sometimes referred to as *3-D rasterization* and *3-D scan-conversion*. If we draw an analogy between a surface specification $\boldsymbol{F}(p)$=0 and a continuous 3-D signal, voxelization is essentially a process of digitization, which takes samples in a spatial domain, measures the relationship between each sampling position and the surface specification concerned, and records the measurement in a volume data representation.

Spatial-Occupancy Enumeration. This is one of the early schemes for object decomposition in computer graphics, and it is nowadays referred to as *binary voxelization* in volume graphics and visualization. Given a surface or a surface-bounded solid model, a binary voxelization algorithm generates a cellular representation that best approximates the spatial occupancy. The cellular primitives are normally organized as an array of cubes in a 3-D regular grid, resulting in the most basic discrete representation of a volume model. The term "voxel" (volume element) is believed to be coined in association with *spatial-occupancy enumeration,* where it refers to such a cellular primitive. Algorithms have been developed for obtaining binary voxel representations for a range of objects, including lines, circles, curves, polygons, polyhedra, quadric objects, implicit solids, and constructive solid geometry.

A critical consideration in binary voxelization is to ensure that the geometrical connectivity between voxels in the voxelized model reflects with the continuity of the original surface model, and the two models feature the same topology. The geometrical and topological properties of such voxelized models are part of the studies of *discrete geometry and topology*, which are often referred to as *digital geometry and topology* in the context of computer graphics and image processing.

Multivalued Voxelization. With the limited resolution of a volume buffer, binary voxelization often results in discrete volume objects that exhibit noticeable object space aliasing. Similar to anti-aliasing in image processing, one effective approach for combating the object space aliasing is to increase the depth of voxel values from the binary domain to the integer or real domain. However, unlike anti-aliasing in the image space that focuses largely on an optical illusion of a smooth object boundary, the main objective of anti-aliasing in the object space is to obtain a better approximation of a continuous object, and to facilitate more accurate sampling during volume rendering. One approach is to use the value at each voxel to encode the intersected area (or volume) between the corresponding cellular primitive and the original surface model (or surface-bounded solid model). Another approach is to apply smoothing convolution filters to a binary volume representation by treating the value at each voxel represents the signal level at a point in space. The most popular multivalued volume representation for approximating a surface model is the *distance field* model.

Distance Field. A *distance field*(3) $\boldsymbol{D}_{\mathrm{X}}(p)$ is a scalar field that defines the closest distance from every point p in 3-D Euclidean space $\mathbb{E}$ to a given point set $\mathbf{X}$. Typically $\mathbf{X}$ is specified as a set of all points on a continuous surface, or inside a surface-bounded solid, although conceptually $\mathbf{X}$ can also be a discontinuous point set. For a closed surface $\mathbf{S}$ that separates $\mathbb{E}$ to two disjoint subdomains, $\mathbb{X}$ and $\mathbb{E}$–$\mathbb{X}$, where $\mathbb{X}$ contains all points inside or on $\mathbf{S}$, a *signed distance field* $\boldsymbol{D}_{\mathrm{S}}(p)$ for $\mathbf{S}$ associates a sign to the distance at each point p to indicate whether p belongs to $\mathbb{X}$, conventionally positive for $p \notin \mathbb{X}$, and negative for $p \in \mathbb{X}$. Hence, the process of rendering $\mathbf{S}$ in surface graphics is transformed to that of rendering the iso-surface $\boldsymbol{D}_{\mathrm{S}}(p)= 0$ in volume graphics.

Like other volume models, a distance field can be represented by a spatially sampled volume dataset, mostly in the form of an isotropic regular grid. Such a dataset is customarily referred to as a *distance volume* (e.g., the chess pieces in Fig. 2) . Although a distance volume dataset may be obtained by sampling every voxel p_i within a bounding volume $\mathbb{D}(\mathbb{X} \subseteq \mathbb{D} \subset \mathbb{E})$ against the specification of $\mathbf{S}$, this approach can often be computationally costly, when it is not straightforward to identify the closest point $x \in \mathbb{X}$ for an arbitrarily given voxel $p_i \in \mathbb{D}$. A number of methods have been proposed for accelerating the identification of the closest point or a small region that contain the closest point, typically by spatially partitioning $\mathbb{D}$ based on the primitives of $\mathbf{S}$ (e.g., triangles in a triangular mesh) or in relation to the parameter space of $\mathbf{S}$ (e.g., in the case of many parametric surfaces). These methods facilitate an efficient search for the closest point by exploiting the precomputed correlation between the subdivisions of $\mathbb{D}$ and the components or parameters of $\mathbf{S}$, as well as the spatial coherence within the subdivisions of $\mathbb{D}$.

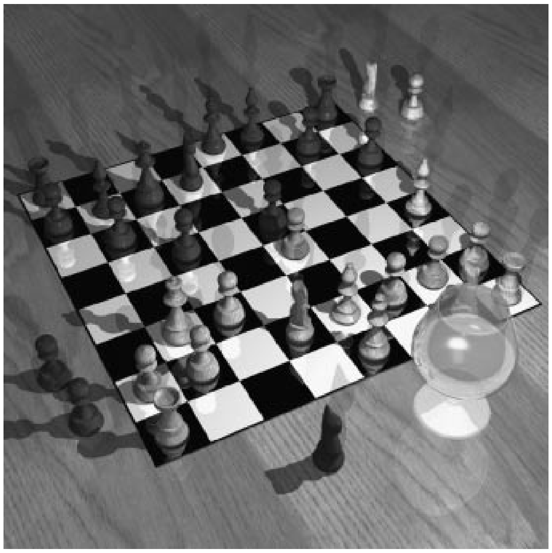


Figure 2. This scene contains over 30 volume objects. The wine glass is a volume object constructed by applying a rotational sweeping to an image (for details, see A. S. Winter and M. Chen, "Image-swept volumes," *Computer Graphics Forum*, **21**(3):441–456, 460, 2002). The chess pieces are distance field models voxelized by M. W. Jones (for details, see M. W. Jones, "The production of volume data from triangular meshes using voxelisation," *Computer Graphics Forum*, **15**(5), 311–318, 1996). The scene was constructed and rendered by A. S. Winter.

An alternative approach to the direct sampling of every voxel in $\mathbb{D}$ is to approximate the distance computation for most voxels using *distance transform*. For many types of surface models, it is relatively easy to determine their spatial occupancy in $\mathbb{D}$, for instance, using a binary voxelization method. Hence a distance volume can be initialized as 0 for occupied voxels and ∞ for unoccupied voxels. It is often desirable to improve this initial distance volume to further classify the occupied voxels, and sometimes additional voxels in their close neighborhood, with more accurate distance calculation. The finite distances calculated are then propagated to the entire volume by systematically evaluating all voxels with an initial distance of ∞. For each of such voxels, p_i, its distance to $\mathbf{S}$ is estimated based on the known distances of the neighboring voxels. Distance transform is normally an iterative process, where a voxel may be evaluated more than once, and the distance volume records only the smallest distance estimated at each voxel.

VOLUME RENDERING

Volume rendering is a computational process for synthesizing 2-D images from volume models. Nowadays the term "volume rendering" usually implies *direct volume rendering,* in which the rendering algorithm processes a volume model directly without the need for extracting an intermediate surface model. As mentioned, a generalized specification of a volume model is a set of scalar fields, $\boldsymbol{F}_1(p), \boldsymbol{F}_2(p), \ldots, \boldsymbol{F}_k(p)$, which define the geometrical and physical attributes of every point p in 3-D space. An ideal volume model would assemble all such attributes for specifying how the model, at each point p in its spatial domain $\mathbb{D}$, would interact with lights coming from all directions. An ultimate volume rendering process would combine all lights that arrive at each pixel to be rendered, taking into account their traversal paths through, and interaction with, the volume model. However, this is computationally intractable, and often, for example in volume visualization, not necessary. In practice, a volume rendering algorithm is confined to evaluating only a specific set of attributes of a volume model, a limited number of light paths, and certain types of interaction between the lights and the volume model.

In volume visualization, a volume data representation often does not contain the geometrical and physical attributes required by a volume rendering algorithm. It is therefore necessary to map the values of the known scalar fields in the data representations to the required attributes. Such a mapping function is referred to as a *transfer function*. Here we assume that all necessary attributes are available to a volume rendering algorithm. Algorithms for direct volume rendering fall into two main categories, namely *image-order* methods and *object-order* (or *volume-order*) methods, which indicate whether a rendering process is executed according to the order of elements of an image or those of a volume representation.

Volume Rendering Integrals

Given a path U along which a light ray passes through a volumetric medium, the light transport between the two endpoints of this path involves a Riemann integral of a function Ξ over the position variable $u \in [\mathrm{a}, \mathrm{b}]$ on the path. This function Ξ defines the interaction between light and material. Such an integral is called *a volume rendering integral* (4,5). For computationally efficiency, many commonly used volume rendering algorithms are confined to evaluate only those paths in a straight line, although it is not necessary for a light ray to follow a straight line. Some of the most commonly used volume rendering integrals are given below.

Emission-Only Integral. This is perhaps the simplest volume rendering integral, which presupposes that the volume model concerned is fully transparent and every point in the object domain $\mathbb{D}$ may potentially emit some light uniformly in all directions. Hence the volume model can be represented by a scalar field $\boldsymbol{E}(p, \lambda)$, which specifies the radiative power emitted at every point $p \in \mathbb{D}$ in the form of a *spectral power distribution* (SPD), where λ is the wavelength within the radiation band concerned. It is common to limit this range to the visible spectrum $\lambda \in [380\,\mathrm{nm}, 770\,\mathrm{nm}]$, or often a narrower range, $\lambda \in [400\,\mathrm{nm}; 700\,\mathrm{nm}]$, to which human eyes are more sensitive. We can also approximate the light using other color representations, such as the RGB color representation, yielding a volume model with three scalar fields, $\boldsymbol{R}(p)$, $\boldsymbol{G}(p)$, and $\boldsymbol{B}(p)$. To maintain the generality, we do not draw explicit distinction between different color representations in the following discussions. For example, we use $\boldsymbol{E}(p)$ as an abstraction for both the spectral and the RGB representations of emitted light.

Because of the assumption of a fully transparent volumetric medium, the light emitted by every point along the light ray will reach the end of the ray. This results in an accumulated light intensity I, which can be expressed by a simple volume rendering integral as shown in Table 1. This integral offers a reasonably accurate approximation of a class of volumetric display hardware, namely *emissive displays*, although the absence of absorption specification restricts its deployment in volume graphics and visualization.

Absorption-Only Integral. This integral is concerned with a translucent volumetric medium with no internal light emitting source. The volume model features essentially only an absorptivity field $\boldsymbol{A}(p)$. In physics, the absorptivity of a homogeneous or infinitesimal volume is normally specified in a spectral representation. However, in volume graphics and visualization, it is commonly approximated by a single scalar value that defines a uniform absorptivity across the visible color spectrum. For a light ray passing through a homogeneous volume with a constant absorptivity α, the light intensity of the ray decreases exponentially with the path length Δu. Let L and I be the intensity of the light entering and leaving the volume, respectively. We have $I = L \cdot e^{-\alpha \cdot \Delta u}$, where $\alpha \cdot \Delta u$ is also referred to as *internal optical density*. This is known as *Lambert's* or *Bouguer's law*. For a volume model with inhomogeneous absorptivity

Table 1. Summary of Commonly Used Volume Rendering Integrals

Name	Attribute Field	Volume Rendering Integral
Emission-only	Emissive Intensity $\boldsymbol{E}(p)$	$I = \int_a^b \boldsymbol{E}(u)du$
Absorption-only	Absorptivity $\boldsymbol{A}(p)$	$I = L \cdot e^{-\int_a^b A(u)du}$ or $\ln L - \ln I = \int_a^b \boldsymbol{A}(u)du$
Absorption and emission	Emissive Intensity $\boldsymbol{E}(p)$, Absorptivity $\boldsymbol{A}(p)$	$I = \int_a^b \boldsymbol{E}(u) \cdot e^{-\int_u^b A(t)dt} du$

defined by a scalar field $\boldsymbol{A}(p)$, the relationship between L and I involves the second volume rendering integral in Table 1. The approximation results from the assumption that the volume model has a constant refractive index, allowing the omission of the partial backreflection in the integral.

In volume visualization, this particular integral is usually used in conjunction with a directional light source placed at the back of a volume model. The light, of an initial intensity L, transmits through the volume model, registering the remaining intensity on the synthesized image, which usually bears a strong resemblance to an inversed x-ray image.

Absorption and Emission Integral. This inevitably leads to the consideration of a volume model with both emission and absorption attributes. Considering an infinitesimal path length du, the emission at each point u along the path of the light is attenuated by the absorption taken place between u and the end of the path. This results in the third volume integral in Table 1. This integral can be used to simulate some imaging devices, such as in nuclear medicine imaging. However, most objects in the real world do not emit light, and uniform emission in all directions cannot convey the shape of an object effectively. This leads to the introduction of the following popular volume rendering integral.

Approximating Volume Rendering Integrals

The above-mentioned volume rendering integrals are usually approximated by a Riemann sum for the corresponding function Ξ and a partition defined by a series of samples $\{u_1, u_2, \ldots, u_n\}$ along a path U. In volume visualization, it is also common to introduce simplifications (e.g., in maximum intensity evaluation) and embellishments (e.g., replacing emissive intensity with rendered color), in order to facilitate the desired system performance and user understanding.

Opacity and Color Integral. This volume rendering integral is based on the absorption and emission integral, but it replaces the emission specification $\boldsymbol{E}(p)$ with a computed reflection specification $\boldsymbol{C}(p)$. The computation of $\boldsymbol{C}(p)$ usually involves one or more external point light sources and considers both the light reflection from and transmission through a volume medium. It makes a number of computationally useful, but conceptually crude, assumptions. For instance, it normally assumes that an external light can reach any point inside the object domain $\mathbb{D}$ without considering the absorption along the light path (i.e., the soft shadow effect). It does not take any secondary lighting into account. It can accommodate refractive transmission, but it usually confines to only specular transmission. Despite its relatively crude assumptions, the introduction of reflection enables more effective depiction of the shape of level-surfaces within a volume model, especially through some familiar visual effects such as the combination of diffuse and specular reflection.

The absorption calculation within the light ray toward the viewer is also simplified by first approximating the inner integral with a corresponding Riemann sum and by sampling discretely between u and b with an interval Δt, resulting in $\prod_{t_1=u+\Delta t}^{t_m=b} e^{-A(t_j)\Delta t}$. We then substitute each exponential function in the product with the first two terms of its Maclaurin's expansion, resulting in $\prod_{t_1=u+\Delta t}^{t_m=b}(1-\boldsymbol{A}(t_j)\Delta t)$, where $\boldsymbol{A}(t_j)\Delta t$ and $1-\boldsymbol{A}(t_j)\Delta t$ are commonly referred to as the *opacity* and *transparency*. Finally, we approximate the whole integral with a Riemann sum of a partition with a series of samples $\{u_1, u_2, \ldots, u_n\}$ along U. This gives an approximated piecewise integral as

$$I \approx \sum\nolimits_{i=1}^{n} \boldsymbol{C}(u_i) \cdot \prod\nolimits_{j=i+1}^{n} (1-\boldsymbol{A}(u_j)\cdot \Delta u)\cdot \Delta u$$

Here we consider an uncomplicated case, where U is a straight line and $\Delta t = \Delta u = u_i - u_{i-1}$ is a constant. We also explicitly make $\prod_k^k (1 - A(u_k)\Delta u) = 1$ because Δu is 0 in this case.

This volume rendering integral is normally used in conjunction with a volume model that features an opacity field $\boldsymbol{\alpha}(p)$ and an object color field $\boldsymbol{c}(p)$. The former is a colloquial reference to the specification of absorptivity and is usually used to attenuate $\boldsymbol{c}(p)$ by assuming that all absorbed energy is transformed to out-scattering energy without change in wavelength. Thus, $\boldsymbol{\alpha}(p)\cdot\boldsymbol{c}(p)$ corresponds to the reflectance of a material independent of any light source, and the result of the product is colloquially referred to as an *opacity-weighted color*. At an arbitrary point $p \in \mathbb{D}$, we determines the reflection of a level-surface at p in relation to the external light sources using an illumination model, such as the Phong and Blinn–Phong models. The computed reflection intensity gives the specification of $\boldsymbol{C}(p)$, which is colloquially referred to as the *rendered color* at p.

Maximum Intensity Evaluation. Given a scalar field $\boldsymbol{B}(p)$ specifying the brightness at every point $p \in \mathbb{D}$, often a volume rendering algorithm is only interested in the maximum brightness value along a light path U. $\boldsymbol{B}(p)$ can be computed from other color specifications, such as emission $\boldsymbol{E}(p)$ and reflected color $\boldsymbol{C}(p)$. More often it relates directly to the grayscale intensity specification of a captured volume dataset, for instance, in medical imaging. To obtain this maximum value, one has to invoke a search, instead of an integration, along U. Although strictly this is not an integral and has little basis in physics, it offers a simpler and faster alternative to the above-mentioned volume rendering integrals. We thereby include this concept here as a "pseudo-integral" because it is applicable to all volume rendering algorithms discussed hereinafter. A volume rendering algorithm based on maximum intensity evaluation is customarily referred to as *maximum intensity projection* (MIP), although it does not necessarily imply the use of an object-order algorithm based on voxel projection.

Transfer Functions

In volume visualization, the physical attributes required by a volume rendering integral, such as absorption and emission, are often not present in a volume model, where the given scalar fields, $\boldsymbol{F}_1(p), \boldsymbol{F}_2(p), \ldots, \boldsymbol{F}_k(p)$, usually represent some captured properties (e.g., sonic reflection,

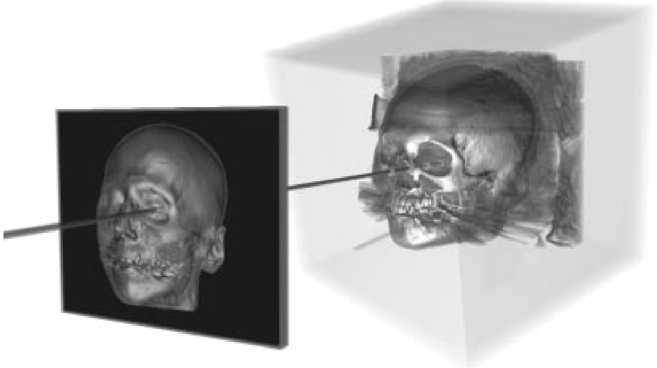


Figure 3. This is an illustration of the ray casting method for direct volume rendering. The scene is comprised of several volume objects, including a CT head dataset (University of North Carolina, Chapel Hill), and a pre-rendered image (a 2D regular grid) containing a visualization of the CT head. Both the scene and the visualization image were themselves rendered using ray casting with different transfer functions. The scene was constructed and rendered by M. Chen.

temperature) that are not relevant to the integral. Hence it is necessary to create the required scalar fields, such as $\boldsymbol{A}(p)$ and $\boldsymbol{E}(p)$ in the case of the absorption and emission integral, by mapping from the given scalar fields, $\boldsymbol{F}_1(p)$, $\boldsymbol{F}_2(p)$, $\ldots$, $\boldsymbol{F}_k(p)$. Such a mapping function is referred to as a *transfer function*. Simple transfer functions typically define a mapping from every possible value in an input scalar field to an appropriate value that is meaningful to the output scalar field. For example, for visualizing a computed tomography dataset, one can create scalar fields for color and opacity from a given scalar field for x-ray attenuation, where different attenuation levels encode different materials (e.g., bones and soft tissues). It is also common to implement such a transfer function using a look-up table.

However, designing an effective transfer function is usually not a trivial task. In some more sophisticated methods, regional properties (e.g., gradient vectors) are used in the design of a transfer function to highlight specific visual features (e.g., material boundary) in the synthesized imagery. Many recent developments in this area have been focused on automatic and semi-automatic construction of transfer functions guided by high-level information, such as a histogram or a contour tree, about a given dataset.

Ray Casting

Ray casting is the principal algorithm for direct volume rendering (6), and it realizes a volume rendering integral by approximating it with a corresponding Riemann sum. It is a nonrecursive variant of the ray tracing method commonly used in computer graphics and is a typical example of image-order methods. The algorithm can be used in conjunction with continuous volume models and sampled volume representations as well as some high-level representations such as volume scene graphs. In principle, it can easily be extended to realize recursive ray tracing and photon ray tracing. Here we consider only a simple case of casting a single eye-ray to realize a volume rendering integral (as illustrated in Fig. 3).

To synthesize an image, the algorithm casts an imaginary ray from a viewing position (i.e., center of projection), through each pixel in the image (i.e., image plane), into the scene containing volume models. Let U be a section of a light path passing through a volume model. The algorithm takes samples of the relevant scalar fields at a series of discrete locations, $\{u_1, u_2, \ldots, u_n\}$, along U. As an example, we use the popular *opacity and color integral* in the form of $\boldsymbol{I} \approx \sum_{i=1}^{n} \boldsymbol{C}(u_i) \cdot \prod_{1}^{j=i}(1 - \boldsymbol{\alpha}(u_j) \cdot \Delta u) \cdot \Delta u$ for computing the light intensity at u_1. Note the changes of upper and lower limits of both the summation and the product. This is because we take samples in the reverse direction of the light from a viewing position. One implementation of this algorithm for a single ray is described by the iterative pseudocode in Table 2. This is normally referred to *as front-to-back ray casting*. It facilitates so-called *early ray termination*, allowing the ray casting to complete whenever the ray has accumulated a sufficient amount of opacity.

An alternative implementation, referred to as *back-to-front ray casting*, is to accumulate the intensity from u_n to u_1, in the same way as the light travels. As shown in Table 2, this provides relatively simpler operations in the iteration without the need for accumulating the opacity, but it loses the advantage of early ray termination. When $\boldsymbol{C}(u_i)$ is derived from $\alpha(u_i) \cdot c(u_i)$, the main operation for computing the compositing color in each iteration is essentially the so-called *alpha blending* operation frequently used for combining multiple layers of images. As the alpha blending operation is widely supported by graphics hardware primarily for texture mapping, the back-to-front approach provides the basis for a particular class of accelerated volume rendering algorithms, namely *texture-based volume rendering*.

Voxel Projection

This class of algorithms is designed to render volume models in spatially sampled data representations. In contrast with the image-order methods that synthesize an

Table 2. Two Alternative Implementations of Ray Casting with the Opacity and Color Integral

Step	Accumulated Intensity	Accumulated Opacity
front-to-back ray casting		
1. Initialization	$I_0 = \text{null}$	$O_0 = 0$
2. Iteration, $i = 1, 2, \ldots, n$	$I_i = I_{i-1} + \boldsymbol{C}(u_i) \cdot (1 - O_{i-1}) \cdot \Delta u$	$O_i = O_{i-1} + \boldsymbol{\alpha}(u_i) \cdot (1 - O_{i-1}) \cdot \Delta u$
3. Early ray termination	**`if`** $O_i \geq (1 - \varepsilon)$ **`then`** $I_n = I_i,\ O_n = O_i$, the ray casting completes **`else`** continue from step 2	
4. Background compositing	$I = I_n + I_{background} \cdot (1 - O_n)$	
	back-to-front ray casting	
1. Initialization	$I_{n+1} = I_{background}$	
2. Iteration, $i = n, \ldots, 2, 1$	$I_i = I_{i+1} \cdot (1 - \alpha(u_i) \cdot \Delta u) + C(u_i) \cdot \Delta u$	

image pixel by pixel, object-order methods process a volumetric dataset voxel by voxel and project those displayable voxels onto one or more pixels in the image plane. The voxels in a volume are normally traversed in either a back-to-front or a front-to-back manner in relation to the distances from voxels to the image plane.

Primitive Projection. Most of the early work in this category involves the projection of an opaque 2-D primitive approximating the projected image of a voxel or parts of its cellular representation, such as a point or a quadrilateral, onto the image plane. Coloring or shading can be applied to each primitive. When several voxels are projected onto the same pixel, the back-to-front approach enables the pixel value of a later voxel (i.e., a voxel in the front) to overwrite that of an earlier voxel (i.e., a voxel at the back). The front-to-back approach often facilitates less drawing operations but requires the support of a z-buffer.

Splatting. As each displayable voxel does not contribute equally to all pixels within its projection on the image plane, visualization generated by projecting opaque primitives often lacks in accuracy and realism. The most effective solution to this problem is the *splatting* algorithm (7). For each voxel, the algorithm evaluates a 3-D function that defines the potential contribution of the voxel to every point in $\mathbb{E}$. Hence, if represents a reconstruction of the original signal available to the data acquisition process. Given a voxel at p_i, the general form of this 3-D function is a radial basis function $\boldsymbol{\omega}$, which is referred to as a *volume reconstruction kernel* or *interpolation kernel*. Note that the influence of p_i often does not have a uniform distribution solely based on the distance to p_i. Several factors that commonly affect the specification of $\boldsymbol{\omega}$.

In perspective projection, for example, we may moderate the influence of p_i in the directions perpendicular to the viewing direction according to the distance from p_i to the view plane, facilitating sharp definition for close-by voxels, and anti-aliasing for distant voxels. With some data representations, such as an anisotropic 3-D grid, the voxel positions that are fed into a volume rendering pipeline are usually specified in the grid coordinates, representing a deformed 3-D Euclidean space. It is thereby necessary to use a nonuniform radial basis function to correct the distortion (see **also VOLUME MODELS AND DATA REPRESENTATIONS**).

Given a voxel p_i and its reconstruction kernel $\boldsymbol{\omega}$, we can synthesize an image of $\boldsymbol{\omega}$ that represents the total amount of contribution, which will be projected from p_i onto the image plane. For each pixel η in this image, the light ray U_η, which passes through $\boldsymbol{\omega}$ and arrives at η, records the contribution of p_i as $\boldsymbol{\Omega}(\eta) = \int_{u_a}^{u_b} \boldsymbol{\omega}(u,\ p_i)du$, where u_a and u_b are the two endpoints of the intersection between U_η and the bounding volume of $\boldsymbol{\omega}$. This image of $\boldsymbol{\omega}$ is called *a footprint*, and a reconstruction kernel with a finite radius of influence has a finite footprint. Let p_i be associated with an intensity value v_i. The projection of this voxel on the image plane is thereby a "splat" that can be computed from the footprint as $v_i \cdot \boldsymbol{\Omega}(\eta)$, for all η in the footprint. One of the central technical issues of the splatting algorithm is the pre-computation of a footprint table, independent from any particular voxels, in order to reduce the cost of computing integration during the rendering. Another is the combination of different splats in the image plane. For an *order-independent* integral, such as the emission-only or absorption-only integral, it is not necessary to process the voxels in any particular order. However, for an *order-dependent* integral, such as the combined absorption and emission integral or the opacity and color integral, we need to process the projected splats in an order that is consistent within the ordering of the corresponding voxels. The compositing of consecutive splats in relation to a particular pixel is similar to the compositing of consecutive samples along a ray in ray casting.

Illumination

For the opacity and color integral, a commonly adopted approach is to involve one or more light sources in the computation of $\boldsymbol{C}(p)$. In theory, the illumination at p depends on not only the optical properties sampled at p and the intensity of each light source, but also indirect light reflected toward p from another part of the medium (i.e., scattering) as well as the absorptivity of the medium that determines how much light can eventually arrive at p (i.e., shadows). Such an illumination model is referred to as *global illumination*. To avoid costly computation with a global illumination model, it is common to adopt a *local illumination* model where $\boldsymbol{C}(p)$ is estimated based only on the optical properties sampled at p and the intensity of each light source. In many applications, a local illumination model is normally adequate for rendering a single iso-surface within a volume. When handling multiple iso-surfaces, or amorphous regions, one needs to be aware of the limitation of such a model and the potential perceptual discrepancy due to the omission of shadows and indirect lighting.

Note that in traditional computer graphics, the terms "global" and "local" can sometimes lead to ambiguous interpretation in volume graphics. Some commonly used volume rendering integrals can produce some typical effects usually associated with global illumination only. For example, the absorption-only integral is in effect a shadow algorithm for back-lit objects. The opacity and color integral, in conjunction with back-to-front ray casting, takes into account indirect light reflected toward a sample from all previously sampled points on the ray. Frequently, the two terms are considered to be the two extremes of a scale, and all illumination models fall somewhere on the scale in a subjective manner. For consistency, in volume graphics, we use the term "global illumination" to imply an illumination model that requires the rendering algorithm to gather indirect light dynamically during rendering at each point to be illuminated, and "local illumination" for one that does not require the gathering of indirect light dynamically, but can use pre-stored luminance at each point due to indirect light. Hence, a model that involves a precomputed shadow volume is a local illumination model (or more precisely with precomputed global illumination data), whereas a model that computes soft shadows by tracing a ray through volumetric medium toward a light source during rendering is a global illumination model. Hence, the challenge is to use a local illumination model to produce as much global illumination effects as possible, with a practicable space requirement and sufficient scene dynamics.

Classic Illumination Models. Given a light source L, one can estimate the reflection at a sampling point locally by using one of the empirical or physically based illumination models designed for surface geometry, such as the Phong, Phong–Blinn, and Cook–Torrance models. When such a model is used in volume rendering, it is assumed that each sampling position, p, is associated with a level-surface or microfacet. This assumption allows us to compute the surface normal at p, which is required by almost all surface-based illumination models. In volume models, surface geometry is normally not explicitly defined, and in many situations, models do not even assume the existence of a surface. Hence, the computation of surface normals is usually substituted by that of gradient vectors. Although for some parametric or procedurally defined volume models, it is possible to derive gradient vectors analytically, in most applications, especially where discrete volumetric models are used, gradient vectors are estimated, for example, using the *finite differences* method for rectangular grids, and *4-D linear regression* for both regular and irregular grids. The commonly used *central differences* method is a reduced form of finite differences based on the first two terms of the Taylor series. For a given point $p = (x, y, z)$, and a small volume domain defined by $[-\delta_x, \delta_x] \times [-\delta_y, \delta_y] \times [-\delta_z, \delta_z]$, the gradient at p can be obtained from a scalar field $\boldsymbol{F}$ as

$$\left(\frac{F(x+\delta_x, y, z) - F(x-\delta_x, y, z)}{2\delta_x},\right.$$
$$\frac{F(x, y+\delta_y, z) - F(x, y-\delta_y, z)}{2\delta_y},$$
$$\left.\frac{F(x, y, z+\delta_z) - F(x, y, z-\delta_z)}{2\delta_z}\right)$$

Many other gradient estimation methods exist, including schemes that involve more or less neighboring samples and schemes where the discrete volume models are first convolved using a high-order interpolation function. Gradients are computed as the first derivative of the interpolation function.

Measured and Precomputed BRDFs. The light reflected from a point on a surface can be described by a *bidirectional reflection distribution function* (BRDF). Hence, it is feasible to obtain a BRDF in sampled form by either measurement or computer simulation(8). The measurements of a BRDF are usually made using a goniophotometer in a large number of directions, in terms of polar and azimuth angles, uniformly distributed on a hemisphere about a source. In computer graphics, it is also common to precompute discrete samples of a BRDF on a hemisphere surrounding a surface element. Given n sampling points on a hemisphere, and n possible incident directions of light, a BRDF can be represented by an $n \times n$ matrix. Given an arbitrary incident light vector, and an arbitrary viewing vector, one can determine the local luminance along the viewing vector by performing two look-up operations and interpolating up to 16 samples.

One major advantage of using measured or precomputed BRDFs is that the rendering algorithm does not require a complex illumination model. One can use measured data to compensate for the lack of an appropriate illumination model that accounts for a range of physical attributes or use precomputed data for a complicated and computationally intensive illumination model. Similar to a BRDF, the light transmitted at a point on a surface can be described by a *bidirectional transmittance distribution function* (BTDF). The combination of BRDF and BTDF provides a discrete specification of a *phase function*.

Phase Functions. A *phase function*, $\boldsymbol{P}(p, \psi, \phi)$, defines a probability distribution of scattering at point p in direction ψ with respect to the direction ϕ of the incident light. The fundamental difference between such an illumination model and those mentioned above is that it is entirely volumetric and does not assume the existence of a surface or microfacet at every visible point in space. Although phase functions are largely used in the context of global illumination, they can be used for local illumination in a perhaps rather simplified manner. Despite the omission of the multiple scattering in local illumination, phase functions allow a volumetric point to be lit by light from any direction. On the contrary, classic illumination models and BRDFs consider only light in front of the assumed surface or microfacet defined at the point concerned.

Multiple Scattering. The most referenced global illumination model is Kajiya's rendering equation, which defines the light transport from point p to point q as $\boldsymbol{I}(q, p) = v(q, p) \cdot (\boldsymbol{E}(q, p) + \int_{x \in \Sigma_S} \boldsymbol{R}(q, p, x)\boldsymbol{I}(p, x)dx)$, where $\boldsymbol{v}$ defines the visibility between p and q, $\boldsymbol{E}$ specifies the light emitted from p in the direction toward q, and $\boldsymbol{R}$ is a bidirectional reflectivity function defining a probability distribution of scattering in the direction from p to q for energy arriving at p from x. The integral is over $\boldsymbol{\Sigma}_S$, which is the set of all points on all surfaces in the scene, which specifies the indirect light reflected from p toward q.

We can modify this rendering equation for global illumination in volume graphics. Since any volume rendering integral featuring absorption would intrinsically take care of the visibility calculation betweenp p and q, we can remove v. We also need to replace Σ_S with a set of all volumetric points in the scene. Since we can cast a ray from p in every direction ϕ and use an appropriate volume rendering integral to gather all indirect light from direction ψ, we can in fact substitute Σ_S with the set of all directions $\boldsymbol{\Phi}$ from a unit sphere towards its center p. This results in Max's rendering equation (5) as

$$\boldsymbol{C}(p, \Psi) = \boldsymbol{E}(p, \Psi) + \int_{\phi \in \boldsymbol{\Phi}} \boldsymbol{\beta}(p)\boldsymbol{P}(p, \Psi, \phi)\boldsymbol{I}(p, \phi)d\phi$$

where $\boldsymbol{C}$ is the light transport from p toward direction Ψ, $\boldsymbol{E}$ is a volumetric emission function that specifies the light emitted from p in direction Ψ, $\boldsymbol{\beta}$ defines the probability light being scattered instead of being absorbed (which is called *albedo)* at p, $\boldsymbol{P}$ is a phase function as mentioned above, and $\boldsymbol{I}$ is the light transport arriving at p from direction ϕ. As we consider every point in space can potentially emit light, we can use the absorption and emission integral to compute $\boldsymbol{I}$. For each ray from p in direction $-\phi$, we have $I(p, \phi) = \int_0^\infty \boldsymbol{C}(p - u\phi) \cdot e^{-\int_I^0 A(p-t\phi)dt} du$. Let q be a pixel on the image plane. We can thus compute its intensity as $I = \int_{\psi \in \Psi} \boldsymbol{I}(q, \psi) d\psi$, where $\boldsymbol{\Psi}$ is the set of all directions from a unit hemisphere in front of the image plane toward its center q.

Max's rendering equation represents a complete solution for global illumination in volume graphics and visualization. However, solving this equation is not a trivial task. Much effort has been made to approximate the equation with various assumptions and simplifications.

One-Dimensional Rediosity

An optical model proposed by Kubelka and Munk considers both absorption and scattering but only in the directions of an incident flux and a reflected flux. It assumes that a volumetric colorant layer can be divided into a large number of homogeneous elementary layers. The optical properties of the volume thus depend on one direction. The two fluxes I_i and I_r flow in opposite directions. Given a volume model with two scalar fields, $\boldsymbol{K}(p)$ and $\boldsymbol{S}(p)$, representing the absorptivity and scattering coefficients, respectively, we can derive the reflectance R and transmittance T of a thin layer around p as

$$R = \frac{\sinh(b \cdot \boldsymbol{S}(p) \cdot \Delta u)}{a \cdot \sinh(b \cdot \boldsymbol{S}(p) \cdot \Delta u) + b \cdot \cosh(b \cdot \boldsymbol{S}(p) \cdot \Delta u)},$$

$$T = \frac{b}{a \cdot \sinh(b \cdot \boldsymbol{S}(p) \cdot \Delta u) + b \cdot \cosh(b \cdot \boldsymbol{S}(p) \cdot \Delta u)}$$

where Δu is the thickness of the layer, $a = 1 + \boldsymbol{K}(p)/\boldsymbol{S}(p)$, and $b = \sqrt{a^2 - 1}$. Given the reflectance and transmittance of a series of consecutive layers, (R_1, T_1), (R_2, T_2), ..., the infinite process of interaction among these layers can be realized using a front-to-back ray casting algorithm (9), as shown in Table 3. Such as infinite process of interaction can be considered as one-dimensional radiosity within the two fluxes, which is a typical effect of global illumination. As the rendering algorithm shown in Table 3 does not require gathering indirect light in addition to the computation of the volume rendering integral for reflectance and transmittance accumulation, it can be classified as a local illumination model based on the previous definition.

ADVANCED TOPICS

Research in volume visualization and volume graphics started in the late 1970s and early 1980s. Since then, significant advances have been made in the following areas.

Volume Modeling

Volume modeling is a process for constructing models of 3-D objects and phenomena using volume data representations, which can be specified procedurally or using sampled datasets. The ultimate aim is to provide users with efficient and effective tools for building complex scenes with such volumetric models. Technical advances in this area include constructive modeling of complex objects and scenes; interactive software systems for sculpting volume objects (see Figs. 1 and 2); data structures for space partitioning, multiresolution representation and data compression; and algorithms for antialiasing in object space.

Volumetric techniques are essential to the modeling and synthesizing of atmospheric and gaseous effects, such as clouds, smoke, and fire. *Volumetric textures*, including solid textures and hyper-textures, have provided vital support to achieve photo-realism in traditional surface graphics. In recent years, volumetric textures also played a central role in modeling and animating realistic hair.

High Performance Hardware and Software Systems

Parallel and distributed computation provided volume graphics and visualization with an indispensable means to achieve real-time performance until recently. Nowadays most volume rendering algorithms can be implemented on consumer PC hardware (10). However, with the increasing size of volume datasets, infrastructure-based computation will continue to play an important role in many applications of volume graphics and visualization. The technical issues to be considered in infrastructure-based volume rendering include *data partitioning and distribution, external memory management, task assignment and load balancing, image composition, collaborative visualization*, and *autonomic infrastructure management*.

Volume Manipulation, Deformation and Animation

Volume manipulation refers to the application of elementary processing operations to volume models usually in the form of sampled datasets. Any manipulation of a volume model will likely lead to changes of sampled values in the

Table 3. The Implementation of Ray Casting Based on the Kubelka and Munk Theory.

Step	Accumulated Reflectance	Accumulated Transmittance
1. Initialization	$\mathbf{R}_0 = \text{null}$	$\mathbf{T}_0 = \text{full}$
2. Iteration, $i = 1, 2, \ldots, n$	$\mathbf{R}_i = \mathbf{R}_{i-1} + \dfrac{\mathbf{T}_{i-1}^2 \cdot R_i}{1 - \mathbf{R}_{i-1} \cdot R_i}$	$\mathbf{T}_i = \dfrac{\mathbf{T}_{i-1} \cdot T_i}{1 - \mathbf{R}_{i-1} \cdot R_i}$
3. Early ray termination	`if` *energy*$(\mathbf{T}_i) \leq \varepsilon$ `then` $\mathbf{R} = \mathbf{R}_i$, $\mathbf{T} = \mathbf{T}_i$ the ray casting completes `else` continue from step 2	
4. Add opaque background or no background	$\mathbf{R}_{with-bg} = \mathbf{R} + \dfrac{\mathbf{T}^2 \cdot R_{background}}{1 - \mathbf{R} \cdot R_{background}}$	*No background*
5. Post-illumination	$I = L_{front-light} \cdot \mathbf{R}_{with-bg}$	$I = L_{front-light} \cdot \mathbf{R} + L_{back-light} \cdot \mathbf{T}$

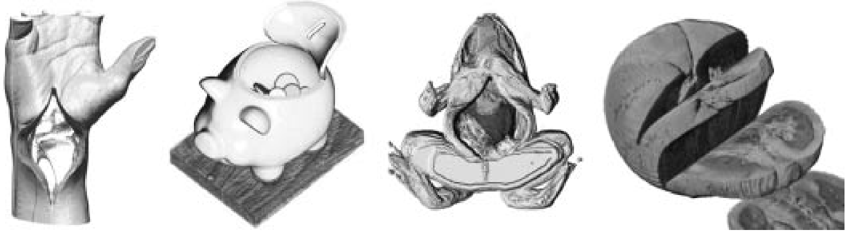


Figure 4. Different deformation operations, such as opening, peeling, and slicing, can be performed on volume models in real time using GPU-assisted techniques. These images were produced by C. Correa (for details, see C. Correa, D. Silver and M. Chen, "Feature aligned volume manipulation for illustration and visualization," *IEEE Transactions on Visualization and Computer Graphics*, **12**(5):1069–1076, 2006). The volume objects are from Lawrence Berkeley Laboratory and University of Erlangen.

datasets and may thereby result in alterations to geometrical, topological, and semantic attributes of the object(s) defined by the model. In general, techniques for manipulating volume models have reached a relatively mature status, with many well-studied technical problems and solutions, including *surface extraction, skeletonization, filtering, volume morphing, segmentation*, and *registration* (11).

Volume deformation refers to the intended change of geometric shape of a volume object under the control of some external influence such as a force. Applications of deformation techniques include computer animation, object modeling, computer-aided illustration (see Fig. 4), and surgical simulation. Techniques for volume deformation fall into two main categories, *empirical deformable models* and *physically based deformation models*. Recent advances in this area include hardware-assisted real-time deformation and mesh-free deformation techniques.

Volume animation refers to the simulation of motion and deformation of digital characters represented by volume models. Although the overall effort made in this area is so far limited, there are several major breakthroughs. Two types of control techniques, namely *block-based* and *skeleton-based*, have been used in volume animation.

Volume Data Capture and Reconstruction

Digitization is a family of technologies for acquiring volumetric models of real-life objects or phenomena. These technologies are based on measuring various physical properties, resulting in a wide range of modalities, including *computed tomography, magnetic resonance imaging*, *3-D ultrasonography, and positron emission tomography* in medical imaging; *seismic measurements* in geosciences; *confocal microscopy* in biology; and *electron microscopy* in chemistry.

Although some modalities involve a volumetric sampling process, which takes a collection of samples at discrete 3-D positions within an object domain, many can only take samples outside the object domain, hence require a *reconstruction* process to build volumetric models from the captured external samples. For example, in computed tomography, techniques have been developed for reconstructing an "intensity" volume from a set of x-ray images. These include *filtered back-projection* for transmission tomography and *algebraic reconstruction technique* and *maximum-likelihood expectation maximization* for emission tomography.

BIBLIOGRAPHY

1. A. Kaufman, D. Cohen, and R. Yagel, Volume graphics, *IEEE Comput.*, **26**(7): 51–64, 1993.
2. W. E. Lorensen and H. E. Cline, Marching cubes: a high resolution 3-D surface construction algorithm, *ACM SIGGRAPH Comput. Graph.*, **21**(4): 163–169, 1987.
3. M. W. Jones, J. A. Bækrentzen, and Milos Sramek, 3-D distance fields: a survey of techniques and applications, *IEEE Trans. Visualization Comput. Graph.*, **12**(4): 581–599, 2006.
4. J. T. Kajiya and B. P. von Herzen, Ray tracing volume densities, *ACM SIGGRAPH Comput. Graph.*, **18**(3), 165–174, 1984.
5. N. Max, Optical models for direct volume rendering. *IEEE Trans. Vis. Comput. Graph.*, **1**(2): 99–108, 1995.
6. M. Levoy, Volume rendering: display of surfaces from volume data, *IEEE Comput. Graph. Applicat.*, **8**(3): 29–37, 1988.
7. L. Westover, Footprint evaluation for volume rendering, *ACM SIGGRAPH Comput. Graph.*, **24**(4): 367–376, 1990.
8. X. D. He, K. E. Torrance, F. X. Sillion, and D. P. Greenberg. A comprehensive physical model for light reflection, *ACM SIGGRAPH Comput. Graph.,* **25**(4): 175–186, 1991.
9. A. Abdul-Rahman and M. Chen, Spectral volume rendering based on the Kubelka-Munk theory, *Comput. Graph. Forum*, **24**(3): 2005.
10. K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf, *Real-time Volume Graphics*, Wellesley, MA: A K Peters, 2006.
11. M. Chen, C. Correa, S. Islam, M. W. Jones, P.-Y. Shen, D. Silver, S. J. Walton, and P. J. Willis. Manipulating, deforming and animating sampled object representations. *Comput. Graph. Forum.* **27**(4): 824–852, 2007.

FURTHER READING

M. Chen, A. E. Kaufman, and R. Yagel (eds.), *Volume Graphics*. New York: Springer, 2000.

K. Mueller and A. Kaufman (eds.), *Proc. Volume Graphics*. New York: Springer, 2001.

I. Fujishiro, K. Mueller, and A. Kaufman (eds.), *Proc. Volume Graphics*. Eurographics, 2003.

E. Gröller, I. Fujishiro, K. Mueller, and T. Ertl (eds.), *Proc. Volume Graphics*. Eurographics, 2005.

T. Möller, R. Machiraju, M. Chen, and T. Ertl (eds.), *Proc. Volume Graphics*. Eurographics, 2006.

B. G. Blundell and A. J. Schwarz, The classification of volumetric display systems: characteristics and predictability of the image space, *IEEE Trans. Vis. Comput. Graph.*, **12** (4): 581–599, 2006.

J. Blinn, Light reflection functions for simulation of clouds and dusty surfaces, *ACM SIGGRAPH Comput. Graph.*, **16** (3): 21–29, 1982.

M. Chen and J. V. Tucker, Constructive volume geometry, *Comput. Graph. Forum*, **19** (5): 281–293, 2000.

J. T. Kajiya, The rendering equation, *ACM SIGGRAPH Comput. Graph.*, **20** (4), 143–150, 1984.

Appropriate articles published in various conferences and journals, including: *Proceedings of ACM SIGGRAPH*, 1976–present.

Proceedings of IEEE Visualization, 1990–present.

Proceedings of Eurographics/IEEE VGTC Data Visualization, 1998–present.

IEEE Transactions on Visualization and Computer Graphics.

ACM Transactions on Graphics.

Eurographics Computer Graphics Forum.

MIN CHEN
Swansea University
Swansea, Wales,
United Kingdom

WARPING AND MORPHING

OVERVIEW

Warping and morphing are the techniques of synthesizing a novel graphical object by deforming given objects. Whereas warping is a purely geometric transformation, morphing (or metamorphosis) interpolates two or more graphical objects. Given two images of different persons as shown on the left and right in Fig. 1, an image morphing technique generates the intermediate images between them, so that the shape and appearance of the faces are transformed as if one person *evolves* into the other. In this article, the state-of-the-art techniques of morphing are introduced, and warping techniques are discussed in the context of the morphing techniques.

Consider the situation where an animator is given two images I_{src} and I_{dst}, and must make the morphing animation between them. The first step of generating a morphing sequence is determines feature correspondences between the images. The correspondences are specified by geometric primitives such as points, line segments, and/or mesh nodes.

The sparse correspondences between images are then converted into a mapping, which is referred to as a *warping function*, that spatially relates all pixels in the images. The warping function defines the smooth deformation of given images in a common coordinate system. The warped images are finally interpolated by a *blending function* into in-between images. The feature specification typically involves user interaction, but the rest of the process can be automated.

Techniques of warping and morphing have been used in both industry and academia. The application includes visual effects in television and film production, fluid and nonrigid body simulation, and visualization of time-series data. The range of application has been extended into various types of media formats used in computer graphics. In the next section, the morphing techniques for two-dimensional (2-D) images are first reviewed. The applications to volumes, three-dimensional (3-D) surface models, and light fields are introduced in the final section.

WARPING AND MORPHING OF IMAGES

Smooth transition between given images can be achieved through simple cross dissolving. The result is, visually poor, however, due to double-imaging effects apparent in misaligned regions. Morphing techniques generate a smooth transformation from one image to another by using a warping process followed by a cross-dissolving process. In this section, we explain several morphing algorithms, including those based on mesh warping, field morphing, scattered data interpolation, energy minimization, and free-form deformations. Some techniques toward automatic morphing are also reviewed. Readers should refer to the survey by Wolberg (1) for the comprehensive review of morphing techniques.

Mesh Warping

The technique of image warping was first developed by the film industry (2). Figure 2 illustrates the two-pass mesh warping algorithm. The top and bottom row are the warping processes of two different images, I_{src} and I_{dst}. In mesh warping, we define a 2-D mesh structure in each of the image coordinate systems. The mesh has the same topology and defines the warping function between the images. Image warping is then performed by deforming the grids between the two. The degree of warping can be controlled by a warping parameter $t \in [0,1]$ shown on a horizontal axis in Fig. 2. A sequence of morphing images is generated by using a linear blending function of two warped images with t as a blending parameter.

The algorithm of mesh warping can be summarized into three-step image processing in Algorithm 1. Some attempts to extend this algorithm to nonlinear interpolation exist. For instance, the Catmull–Rom spline interpolation for mesh warping is demonstrated by Wolberg (1).

Field Morphing

One drawback to mesh warping is that a user has to specify the feature points between images by a grid. Defining a grid structure for given images is not a trivial task, and the result of warping is affected by the mesh structure. Beier and Neely (3) propose a field morphing technique that allows a user to specify the correspondence between images by a sparse set of line segments. A globally smooth warping function is generated according to the distance to each segment. Points and curved lines can also be used for the primitives to define the correspondence. The final warping function is a weighted sum of the warping generated by all primitives. Figure 3 shows an example of field morphing.

Algorithm 1. Mesh Warping

```
for all $t \in [0, 1]$ do
  Warp $I_{src}$ into $W_{src}$ using $t$ as a warping parameter
  Warp $I_{dst}$ into $W_{dst}$ using $1 - t$ as a warping parameter
  Blend $W_{src}$ and $W_{dst}$ into $I_{morph}$ using $t$ as a blending parameter
end for
```

Scattered Data Interpolation

The most generic primitive for feature correspondence is a point, because lines and curves can be point sampled. The generic algorithm of image blending is then considered as the interpolation between two 3-D points, $(x_{src}, y_{src}, t_{src})$ and $(x_{dst}, y_{dst}, t_{dst})$, where a point (x_{src}, y_{src}) in an image I_{src} is warped with a warping parameter t_{src} and a point (x_{dst}, y_{dst}) in another image I_{dst} is warped with t_{dst}.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

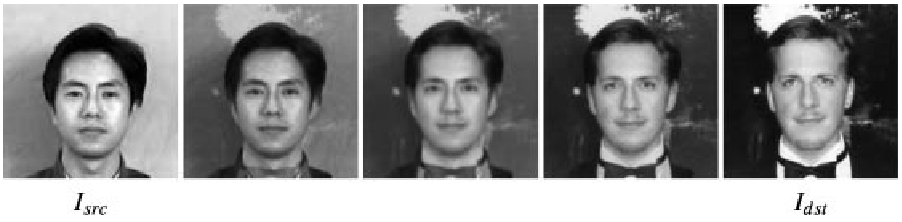


Figure 1. Image morphing: Given two images of objects, I_{src} and I_{dst}, an animation sequence between them is generated in the way that I_{src} deforms gradually into I_{dst}.

This formulation is investigated by Ruprecht and Miller (4). A framework using thin plate splines is developed by Lee et al. (5). Hassanien and Nakajima (6) propose a method of facial image metamorphosis that uses Navier splines. Arad et al. (7) use radial basis functions for the same application. Figure 4 shows the warping by radial basis functions. The techniques based on scattered data interpolation can generate a globally smooth warping from a coarse set of point correspondences. The computational cost is low, and the algorithm is generally more stable than field morphing.

Bijective Warping

Field morphing and scattered data interpolation do not guarantee one-to-one correspondences between two images and, therefore, do not have any physical meaning in the deformation. Lee et al. (8) proposed an energy minimization method for deriving one-to-one warp functions. They further extended their method and developed a more effective method (9) by combining an energy minimization approach with a free-form deformation (FFD) method proposed by Sederberg and Parry (10) in a multiresolution structure (see Fig. 5). This method is inspired by the nonrigid deformation in physics simulation, which enables natural-looking deformation in a morphing sequence.

Automated Morphing

The most time-consuming part in a morphing process is specifying correspondence between images. Suppose two images are sufficiently similar; then the techniques of image registration methods (11) can be used to generate the correspondences. When the images are different views of an object, the reconstruction of camera geometry gives us geometrically valid correspondences (12). For nonrigid objects, optical flow methods (13) and feature trackers (14) can be used.

The methods mentioned above assume that two images are captured under similar conditions and, therefore, that only small displacements must be recovered. This assumption, however, does not hold in general morphing applications where two images can be vastly different. Shinagawa and Kunii (15) propose a method of finding correspondences between images without using any constraints except image intensity. Yamazaki et al. (16) propose a linear filter bank to exploit the multilevel image structure.

Although automatic morphing can drastically reduce the work for a user, user interaction is essential for defining semantic correspondence or for remedying artifacts caused by erroneous correspondences. Gao and Sederberg (17) propose a hybrid system that allows a user to improve the correspondences generated by image-matching algorithms.

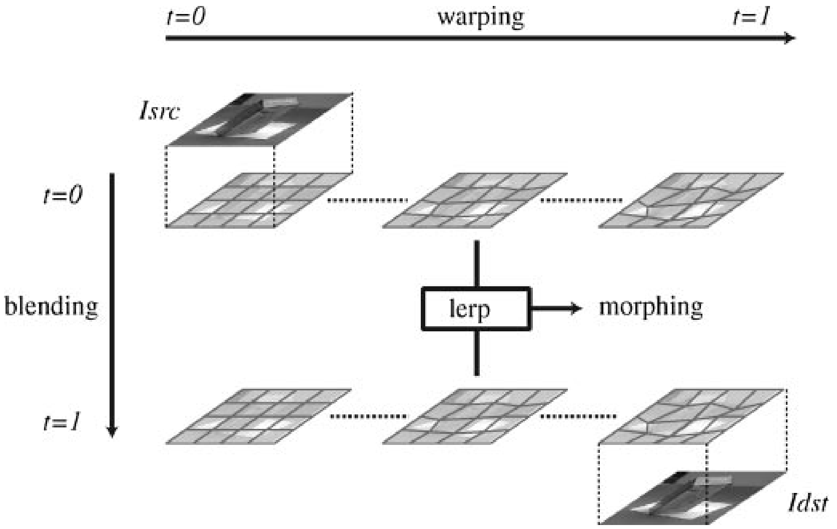


Figure 2. Mesh warping.

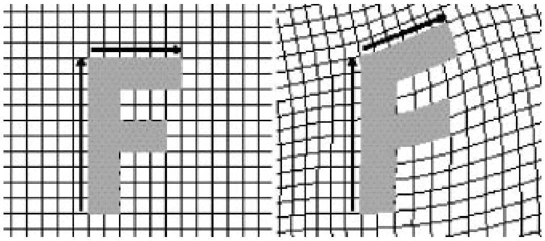


Figure 3. Field morphing: Two sides of an "F" character are specified by line segments. The weighted sum of linear transformation defined by each line segment gives us a globally smooth warping function [images by Beier and Neely (3)].

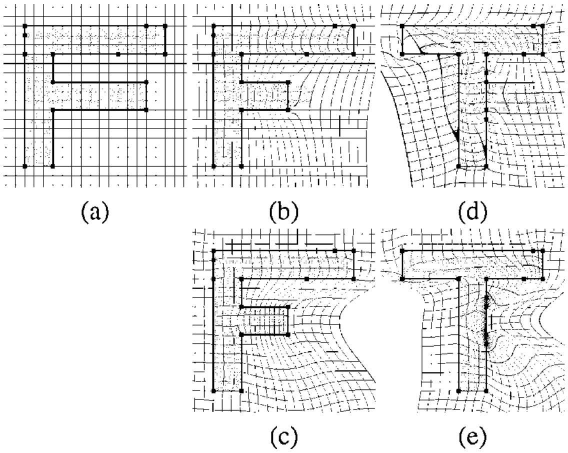


Figure 5. Free-form deformation from "F" to "T" characters. The top row shows results obtained by field morphing. The bottom row is the results by multilevel free-form deformation [images by Lee et al. (9)].

The user can specify some feature points as constraints on the warping between images, and the system then can determine the best image matching by solving a constrained optimization.

APPLICATIONS TO OTHER GRAPHICAL OBJECTS

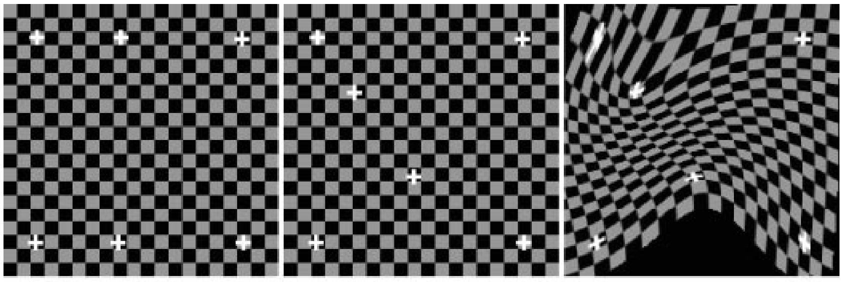
Following the success of image morphing techniques shown in the previous section, many researchers have developed the techniques of morphing 3-D objects (18). Generally, a morphing of 3-D models includes the interpolation of their shapes as well as an interpolation of their attributes such as color, texture, or appearance of the surface. The challenge in these techniques is how to define an intrinsic morphing sequence between any two objects in arbitrary structure.

Volumetric Representation of 3-D Shape

Implicit surface (19), level set (20), and voxelized objects are commonly used representations of 3-D solids and 2-D closed surfaces. The shape of an object is defined by a set of points p such that $f(p) = c$ for some function f and shape attribute c. It is then a straightforward task to extend and generalize the techniques of 2-D image morphing to 3-D shapes or higher dimension.

Pasko and Savchenko (21) developed a warping algorithm for 3-D shapes represented in the scalar functions. This method, however, often suffers from unnecessary distortion or change in topology such as creation of many connected components. Cohen-Or et al. (22) solve this problem by combining a straightforward interpolation with a signed distance transformation that allows the algorithm to deform the whole space continuously. Figure 6 shows the morphing of 3-D shapes that have different topology. The signed distance representation of 3-D shapes is a mathematical abstraction of geometric properties, such as continuity or genus, and therefore, it allows continuous transition between different geometry without concerning the explicit properties in the sequence of morphing animation.

A specific user interface for designing a morphing sequence of two volumes is proposed by Lerios et al. (23) Their system provides users a graphical interface to specify feature correspondences by using simple geometric primitives such as points, lines, and boxes in the volumes in the same spirit as the Beier and Neely method (3). Figure 7 shows an example of the graphics user interface where 37 geometric primitives are used for feature correspondences.



(a) I_{src} (b) I_{dst} (c) W_{src}

Figure 4. Warping by scattered data interpolation: Image warping by radial basis functions. (a) Source image and feature points. (b) Destination image and corresponding feature points. (c) Source image warped by thin plate spline radial basis [images by Arad et al. (7)].

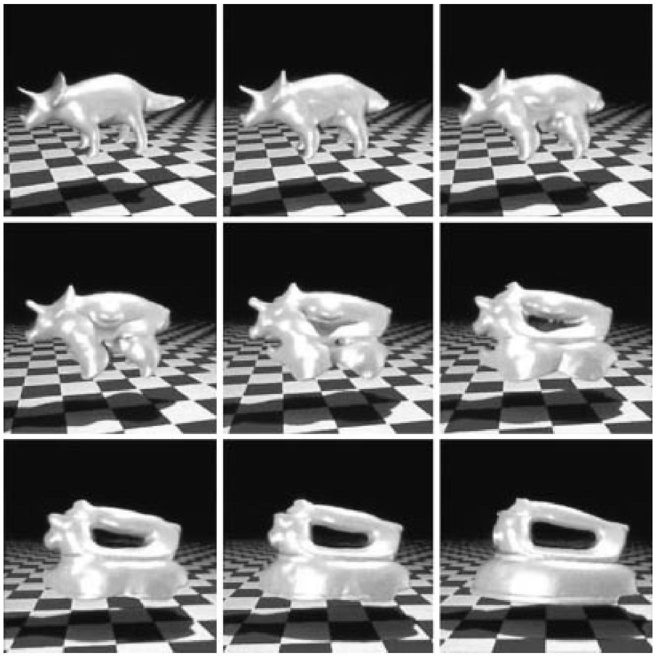
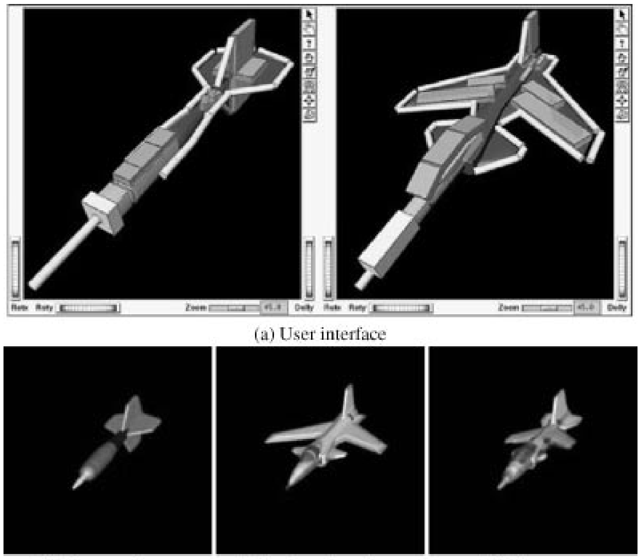


Figure 6. 3-D shape morphing between objects with different topology [images by Cohen-Or et al. (22)].



(a) User interface

(b) Source volume

(c) Destination volume

(d) In between

Figure 7. Feature-based volume morphing between a dart and an X-29 space ship [images by Lerios et al. (23)].

Boundary Representation of 3-D Shape

Boundary representations of 3-D shape are very popular in the computer graphics community. A large number of models and data structures have been proposed to represent objects by their boundaries. The polygonal surfaces and the parameterized surfaces are the two main models.

The use of boundary representations has several advantages: efficient data structure, capability of texture mapping, and intuitive representation. As a counterpart, this representation is implicitly constrained both geometrically and topologically. The algorithm of morphing of 3-D shapes in boundary representation has to consider these constraints during the entire process of morphing.

Specifying corresponding points between two surface models is essential to constructing a single mesh with two geometric instantiations: one for each source and destination object. This single mesh can be obtained by merging the meshes (24) or by creating a new common mesh (25). The existence of a common mesh for two different models implies that they have the same topology.

DeCarlo and Gallier (26) propose a method of dealing with degenerated geometric instantiations of the common mesh where an edge or a face can be embedded onto a single point or edge (see Fig. 8). They use a sparse control mesh on each surface in order to define a mapping between the input objects. This method applies to general (triangulated) polyhedral surfaces.

Light Field

The light field (27, 28) is the representation of the appearance of 3-D objects without explicit geometry. The dataset is typically composed of a large number of images captured from various viewpoints. Each pixel in the images is regarded as a sample of rays passing in the 3-D space. Given a viewpoint at rendering, a corresponding view is synthesized by interpolating the sampled rays.

Zhang et al. (29) applied a feature-based morphing approach to the light field that is similar to the Beier and Neely system (3). Given two light field data, a user selects representative viewpoints for each light field and then specifies the correspondence that defines a common mesh structure on 2-D images of rendered light fields. The system then propagates the correspondence to other viewpoints that are not selected by the user, taking into account the occlusion.

In addition to the change of viewpoints, the variable light source is also modeled in surface light field rendering (30). This method assumes that the 3-D shape of an object of interest is given and that the variation of the appearance of all points of the surface is represented efficiently. Jeong et al. (31) applied the feature-based morphing technique to the surface light field. Because the surface appearance changes drastically, they propose a dynamic change of mesh structure defined by a user so that the highlights are not blurred by interpolation.

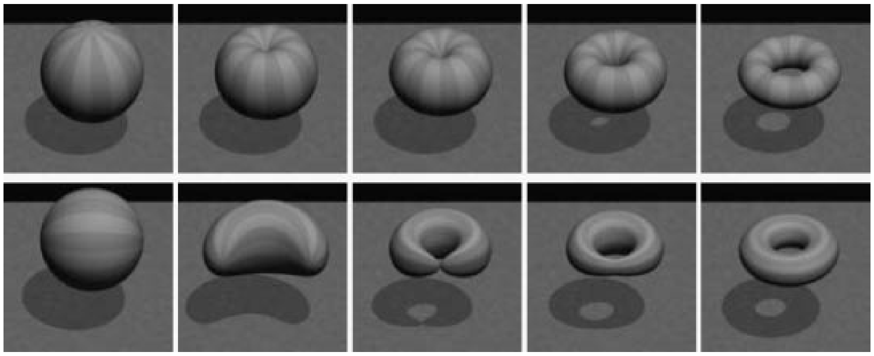


Figure 8. Topological evolution of 3-D surfaces [images by DeCarlo and Gallier (26)].

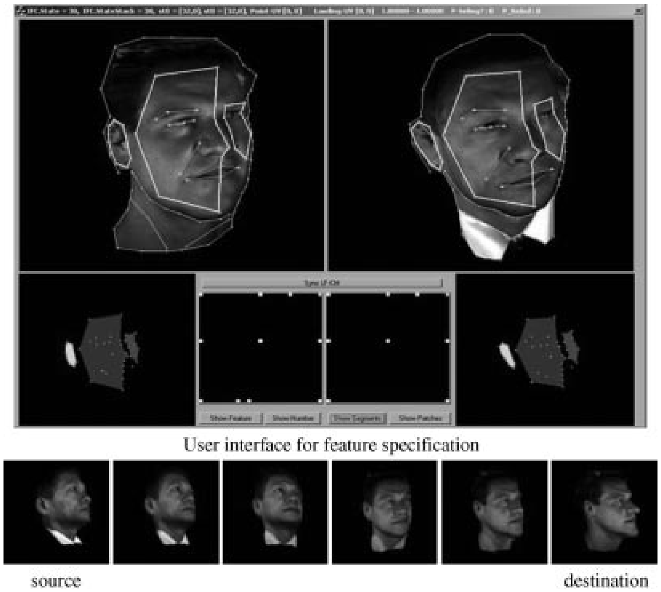


Figure 9. Feature based light field morphing [images by Zhang et al. (31)].

SUMMARY

In this article, some representative work for warping and morphing of images and 3-D object models are presented. The warping is an underlying process of morphing algorithms, although the warping techniques themselves have a wide range of applications in the context of image registration (11). The morphing can be used not only for visualization purposes but also for data compression (32). The biggest issue in warping and morphing processes is how to efficiently build correspondences between data. Several intuitive user interfaces that enable a user to specify complicated feature correspondences have been proposed, as well as a few algorithms that attempt to find correspondences automatically. Designing easy-to-use and powerful systems for general morphing purposes is still an open problem.

BIBLIOGRAPHY

1. G. Wolberg, Image morphing: A survey, *The Visual Computer*, **14**(8): 360–372, 1998.
2. D. B. Smythe, *A two-pass mesh warping algorithm for object transformation and image interpolation*, Technical Report 1030, Industrial Lights and Magics, 1990.
3. T. Beir and S. Neely, Feature-based image metamorphosis, *Proc. SIGGRAPH '92*, ACM, June 1992, p. 35.
4. D. Ruprecht and H. Müller, Deformed cross-dissolves for image interpolation in scientific visualization, *J. Visualization and Computer Animation*, **5**(3): 167–181, 1994.
5. S.-Y. Lee, K.-Y. Chwa, J. Hahn, and S. Y. Shin, Image morphing using deformable surfaces, *Proc. Computer Animation*, 1994, pp. 31–39.
6. A. E. Hassanien and M. Nakajima, Image morphing of facial images transformation based on navier elastic body splines, *Proc. the Computer Animation*, 1998, pp. 119–125.
7. N. Arad, N. Dyn, D. Reisfeld and Y. Yeshurun, Image warping by radial basis functions: applications to facial expressions, *CVGIP: Graphical Models and Image Processing*, **56**(2): 161–172, 1994.
8. S.-Y. Lee, K.-Y. Chwa, and S. Y. Shin, Image metamorphosis using snakes and free-form deformations, *Computer Graphics*, **29**: 439–448, 1995.
9. S.-Y. Lee, K.-Y. Chwa, J. Hahn, and S. Y. Shin, Image morphing using deformation techniques, *J. Visualization and Computer Animation*, **7**(1): 3–24, 1996.
10. T. W. Sederberg and S. R. Parry, Free-form deformation of solid geometric models, *Proc. SIGGRAPH '86*, 1986, pp. 151–160.
11. B. Zitova and J. Flusser, Image registration methods: A survey, *Image and Vision Computing*, **24**: 977–1000, 2003.
12. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed.Cambridge, U.K.: Cambridge University Press, 2004.
13. B. K. P. Horn and B. G. Schunk, Determining optical flow, *Artificial Intell*, **17**: 185–203, 1981.
14. B. D. Lucas and T. Kanade, An iterative image registration technique with an application to stereo vision, in P. J. Hayes, (ed.), *Proc. 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*. William Kaufmann, August 1991, pp. 674–679.
15. Y. Shinagawa and T. L. Kunii, Unconstrained automatic image matching using multiresolutional critical-point filters, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**(9): 994–1010, 1998.

16. S. Yamazaki, K. Ikeuchi, and Y. Shingawa, Determining plausible mapping between images without a priori knowledge, *Proc. Asian Conference on Computer Vision 2004*, 2004, pp. 408–413.

17. P. Gao and T. W. Sederberg, A work minimization approach to image morphing, *The Visual Computer*, **14**(8-9): 390–400, 1998.

18. F. Lazarus and A. Verroust, 3D metamorphosis: A survey, *The Visual Computer*, **8-9**(14): 373–389, 1998.

19. J. Bloomenthal, *Introduction to Implicit Surfaces*, San Francisco, CA Morgan Kaufmann, 1997.

20. J. Sethian, *Level set methods and fast marching methods*. Cambridge, U.K.: Cambridge University Press, 1996.

21. A. A. Pasko and V. V. Savchenko, Constructing functionally defined surfaces, *Proc. First International Workshop on Implicit Surfaces, Grenoble*, 1995, pp. 97–106.

22. D. Cohen-Or, A. Solomovic and D. Levin, Three-dimensional distance field metamorphosis, *ACM Transactions on Graphics*, **17**(2): 116–141, 1998.

23. A. Lerios, C. D. Garfinkle, and M. Levoy, Feature-based volume metamorphosis, *Proc. SIGGRAPH '95*, 1995, pp. 449–456.

24. E. W. Bethel and S. P. Uselton, Shape distortion in computer-assisted keyframe animation, *Proc. Computer Animation '89*, 1989, pp. 215–224.

25. F. Lazarus and A. Verroust, Metamorphosis of cylinder-like objects, *J. Visualization and Computer Animation*, **8**(3): 131–146, 1997.

26. D. DeCarlo and J. Gallier, Topological evolution of surfaces, *Proc. Graphics Interface '96*, 1996, pp. 194–203.

27. M. Levoy and P. Hanrahan, Light field rendering, *Proc. SIGGRAPH '96*, 1996, pp. 31–42.

28. S. Gortler, R. Grzeszczuk, R. Szeliski and M. Cohen, The lumigraph, *Proc. SIGGRAPH '96*, 1996, pp. 43–54.

29. Z. Zhang, L. Wang, B. Guo, and H.-Y. Shum, Feature-based light field morphing, *ACM Transactions on Graphics*, **21**(3): 457–464, 2002.

30. D. Wood, D. Azuma, W. Aldinger, B. Curless, T. Duchamp, D. Salesin and W. Steutzle, Surface light fields for 3D photography, *Proc. SIGGRAPH 2000*, 2000, pp. 287–296.

31. E. Jeong, M. Yoon, Y. Lee, M. Ahn, S. Lee, and B. Guo, Feature-based surface light field morphing, *Proc. Pacific Graphics 2003*, 2003, pp. 215–223.

32. F. Galpin, R. Balter, L. Morin and K. Deguchi, 3D models coding and morphing for efficient video compression, *Proc. Computer Vision and Pattern Recognition 2004*, Vol. 1, 2004, pp. 331–334.

SHUNTARO YAMAZAKI
National Institute of
Advanced Industrial
Science and Technology
Tokyo, Japan

ARTIFICIAL INTELLIGENCE LANGUAGES

The process of programming a solution to a problem is inherently difficult. This has been recognized by conventional programmers for many years and has been one of the motivating forces behind both structured and object-oriented programming techniques. The problem seems to be that the human brain does not have the capacity to handle the complexity of the programming task for nontrivial problems. The solution has been to first use structured and then object-oriented techniques that break the problem into manageable "chunks." However, this "*divide et impera*" technique did not solve the problem of the imperative (procedural, commanding) description of the solution, i.e., of the explicit ordering of the actions leading to the solution. Moreover, the sequence of statements in imperative language also implies the need to have explicit commands to alter the sequence, for example, control structures such as "while…do," "repeat…until," or even "goto." Many errors in imperative languages are introduced because the specified sequencing is not correct. On the other hand, in declarative languages used mainly for AI programming, we describe the problem rather than the explicit way to solve it, or the order in which things must be done. The explicit ordering has been replaced by the implicit ordering, conditioned by the relationships between the objects. The lack of explicit sequence of control relieves the user of the burden of specifying the control flow in the program.

Declarative programming is the umbrella term that covers both functional programming and relational programming. Although the two approaches do have many superficial similarities—both classes of languages are nonprocedural and, in their pure forms, involve programming without side-effects—they do have different mathematical foundations. In writing functional programs, the programmer is concerned with specifying the solution to a problem as a collection of many-to-one transformations, which corresponds closely to the mathematical definition of a function. On the other hand, a relational program specifies a collection of many-to-many transformations. Thus, in relational programming languages, there is a set of solutions to a particular application rather than the single solution that is produced from a function application. Although the execution mechanisms that have been proposed for relational programming languages are radically different from the approaches for a functional programming language, both approaches have been widely used in artificial intelligence programming.

To provide AI-related comparison, we have included two equally popular AI-language alternatives, a functional language Lisp and relational language Prolog. From the beginning, Lisp was the language of choice for American AI researchers. The reasons are many, but primarily they result from the strong mathematical roots of the language, symbolic rather than numeric processing, and its ability to treat its own code as data. Researchers have exploited this capability of the Lisp program to modify themselves at runtime for research in machine learning, natural-language understanding, and other aspects of AI. Moreover, artificial intelligence programming requires the flexibility, extensibility, modularity, and underlying data structures and data abstraction facilities that Lisp provides. Although Lisp is one of the older programming languages in use, it has remained the most widely used language in AI programming.

The logic programming language Prolog has been growing in popularity since it was originally introduced in Europe in the early 1970s. Prolog is most easily matched to tasks involving logic and proof-like activities. A Prolog program is essentially a description of objects and relations between them. A subset of formal logic (called Horn clause logic) is used to specify the desired conditions. Prolog's adherent believes that it is easier to learn and use than Lisp. They say that it uses less memory and is more easily moved from one computer to another.

Although both Lisp and Prolog have been supported with almost religious intensity by passionate advocates, the dilemma of Prolog versus Lisp has softened over the years, and many now believe in a combination of ideas from both worlds. New AI languages can be roughly divided into Lisp-like languages, Prolog-like languages, hybrid languages, object-oriented languages, agent-oriented languages, semantic Web languages, and specialized AI programming languages.

Before we discuss specific AI programming paradigms and languages, it would be useful to underline the specific features that facilitate the production of AI programs as distinct from a language for writing other types of applications. Apart from the features that are now needed for building almost any kind of complex systems, like possessing a variety of data types, a flexible control structure, and the ability to produce efficient code, the features that are particularly important in building AI systems are as follows (1–4):

- Good symbol manipulation facilities, because AI is concerned with symbolic rather than numeric processing.
- Good list manipulating facilities, because lists are the most frequently used data structures in AI programs.
- Late binding times for the object type or the data structure size, because in many AI systems, it is not possible to define such things in advance.
- Pattern matching facilities, both to identify data in the large knowledge base and to determine control for the execution of production systems.
- Facilities for performing some kind of automatic deduction and for storing a database of assertions that provide the basis for deduction.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

- Facilities for building complex knowledge structures, such as frames, so that related pieces of information can be grouped together and assessed as a unit.
- Mechanisms by which the programmer can provide additional knowledge (meta-knowledge) that can be used to focus the attention of the system where it is likely to be the most profitable.
- Control structures that facilitate both goal-directed behavior (top-down processing or backward-chaining) and data-directed (or bottom-up processing or forward chaining).
- The ability to intermix procedural and declarative knowledge in whatever way best suits a particular task.
- Good programming environment, because AI programs are among the largest and most complex computer systems ever developed and present formidable design and implementation problems.

No existing language provides all of these features. Some languages do well at one at the expense of others, and some hybrid languages combine multiple programming paradigms trying to satisfy as many of these needs as possible. However, the main differentiator between various AI programming languages is their ability to represent knowledge clearly and concisely.

LOGIC PROGRAMMING

Logic programming began in the early 1970s as a direct outgrowth of earlier work in automatic theorem proving and artificial intelligence. It can be defined as the use of symbolic logic for the explicit representation of problems, together with the use of controlled logical inference for the effective solution of those problems.

The credit for the introduction of logic programming goes mainly to Kowalski (5), (6) and Colmerauer et al. (7), although Green (8) and Hayes (9) should also be mentioned in this regard. In 1972, Kowalski and Colmerauer were led to the fundamental idea that logic can be used as a programming language. The acronym PROLOG [PROgramming in LOGic) was conceived, and the first PROLOG interpreter was implemented in the language ALGOL-W by Roussel in 1972.

The idea that first-order logic, or at least substantial subsets of it, can be used as a programming language was revolutionary, because until 1972, logic had only ever been used as a specification language in computer science. However, it has been shown that logic has a procedural interpretation, which makes it very effective as a programming language. Briefly, a program clause $A < = B_1, \ldots, B_n$ is regarded as a procedure definition. If $< = C_1, \ldots, C_k$ is a goal clause, then each C_j is regarded as a procedure call. A program is run by giving it an initial goal. If the current goal is $< = C_1, \ldots, C_k$, a step in the computation involves unifying some C_j with the head A of a program clause $A < = B_1, \ldots, B_n$ and thus reducing the current goal to the goal $< = (C_1, \ldots, C_{j-1} B_1, \ldots, B_n, C_{j+1}, \ldots, C_k)\theta$, where θ is the unifying substitution. Unification thus becomes a uniform mechanism for parameter passing, data selection, and data construction. The computation terminates when the empty goal is produced.

One of the main ideas of logic programming, which is due to Kowalski, is that an algorithm consists of two disjoint components, the logic and the control. The logic is the statement of *what* the problem is that has to be solved. The control is the statement of *how* it is to be solved. The ideal of logic programming is that the programmer should only have to specify the logic component of an algorithm. The logic should be exercised solely by the logic programming system. Unfortunately, this ideal has not yet been achieved with current logic programming system because of two broad problems. The first of these is a control problem. Currently, programmers need to provide a lot of control information, partly by the ordering of clauses and atoms in clauses and partly by extra-logical control features, such as *cut*. The second problem is the negation problem. The Horn clause subset of logic does not have sufficient expressive power, and hence the Prolog system allows negative literals in the bodies of clauses.

Logic has two other interpretations. The first of these is the database interpretation. Here a logic program is regarded as a database. We thus obtain a very natural and powerful generalization of relational databases, which correspond to logic programs consisting solely of ground unit clauses. The concept of logic as a uniform language for data, programs, queries, views, and integrity constraints has great theoretical and practical potential.

The third interpretation of logic is the process interpretation. In this interpretation, a goal $< = B_{-1}, \ldots B_n$ is regarded as a system of concurrent processes. A step in the computation is the reduction of a process to a system of processes. Shared variables act as communication channels between processes. Now several Prologs are based on the process interpretation. This interpretation allows logic to be used for operating system applications and object-oriented programming.

It is clear that logic provides a single formalism for apparently diverse parts of computer science. Logic provides us with a general-purpose, problem-solving language, a concurrent language suitable for operating systems and a foundation for database systems. This range of applications together with the simplicity, elegance, and unifying effect of logic programming assures it of an important and influential future.

One of the most important practical outcomes of the research in logic programming has been the language Prolog, based on a *Horn clause* subset of logic. Most of logic programming systems available today are either Prolog interpreters or compilers. Most use the simple computation rule, which always selects the leftmost atom in a goal. However, logic programming is by no means limited to a Prolog. It is essential not only to find more appropriate computation rules, but also to find ways to program in a larger subset of logic, not just in a clausal subset. In this entry we will also briefly cover a database query language based on logic programming, Datalog, and several hybrid languages supporting the logic programming paradigm (together with some other paradigms, functional, for instance).

Prolog

Prolog stands for programming in logic—an idea that emerged in the early 1970s to use logic as a programming language. The early developers of this idea included Robert Kowalski at Edinburgh (on the theoretical side), Maarten van Emden at Edinburgh (experimental demonstration), and Alain Colmerauer at Marseilles (implementation). The curent popularity of Prolog is largely due to David Warren's efficient implementation at Edinburgh in the mid-1970s.

Prolog has rapidly gained popularity in Europe as a practical programming tool. The language received impetus from its selection in 1981 as the basis for the Japanese Fifth Generation Computers project. On the other hand, in the United States its acceptance began with some delay, due to several factors. One was the reaction of the "orthodox school" of logic programming, which insisted on the use of pure logic that should not be marred by adding practical facilities not related to logic. Another factor was a previous American experience with the Microplanner language, also akin to the idea of logic programming, but inefficiently implemented. And the third factor that delayed the acceptance of Prolog was that for a long time Lisp had no serious competition among languages for AI. In research centers with strong Lisp tradition, there was therefore a natural resistance to Prolog.

The language's smooth handling of extremely complex AI problems and ability to effect rapid prototyping has been a big factor in its success, even in the United States. Whereas conventional languages are procedurally oriented, Prolog introduces the descriptive or declarative view, although it also supports the procedural view. The declarative meaning is concerned only with the *relations* defined by the program. This greatly alters the way of thinking about problem and makes learning to program in Prolog an exciting intellectual challenge. The declarative view is advantageous from the programming point of view. Nevertheless, the procedural details often have to be considered by the programmer as well.

Apart from this dual procedural/declarative semantics, the key features of Prolog are as follows (10–12):

- Prolog programming consists of defining *relations* and querying about relations.
- A program consists of *clauses*. These are of three types: *facts*, *rules*, and *questions*.
- A relation can be specified by *facts*, simply stating the n-tuples of objects that satisfy the relation, or by stating *rules* about the relation.
- A *procedure* is a set of clauses about the same relation.
- Querying about relations, by means of *questions*, resembles querying a database. Prolog's answer to a question consists of a set of objects that satisfy the question.
- In Prolog, to establish whether an object satisfies a query is often a complicated process that involves logical inference, exploring among alternatives and possibly *backtracking*. All this is done automatically by the Prolog system and is, in principle, hidden from the user.

Different programming languages use different ways of representing knowledge. They are designed so that the kind of information you can represent, the kinds of statements you can make, and the kinds of operations the language can handle easily all reflect the requirements of the classes of problems for which the language is particularly suitable. The key features of Prolog that give it its individuality as a programming language are as follows:

- Representation of knowledge as *relationships* between objects; the core representation method consists of relationships expressed in terms of a predicate that signifies a relationship and arguments or objects that are related by this predicate.
- The use of *logical rules* for deriving implicit knowledge from the information explicitly represented, where both the logical rules and the explicit knowledge are put in the *knowledge base* of information available to the Prolog.
- The use of *lists* as a versatile form of structuring data, although not the only form of structuring data that is used in Prolog.
- The use of *recursion* as a powerful programming technique.
- Variables acquire values by a process of *pattern matching* in which they are instantiated or bound to various values.

The simplest use of Prolog is to use it as a convenient system for retrieving the knowledge explicitly represented, i.e., for interrogating or queering the knowledge base. The process of asking a question is also referred to as "setting a GOAL for the system to satisfy." You type the question, and the system searches the knowledge base to determine whether the information you are looking for is there.

The next use of Prolog is to supply the system with part of the information you are looking for and to ask the system to find a missing part.

In both cases above Prolog works fundamentally by pattern matching. It tries to match the pattern of our question to the various pieces of information in the knowledge base.

The third case has a distinguishing feature. If your question contains variables (a word beginning with an uppercase letter), Prolog also has to find what are the particular objects (in place of variables) for which the goal are satisfied. The particular instantiation of variables to these objects are shown to the user.

One advantage of using Prolog is that Prolog interpreter is in essence a built-in inference engine that draws logical conclusions using the knowledge supplied by the facts and rules.

To program in Prolog, one specifies some facts and rules about objects and relationships and then asks questions about objects and relationships. For instance, if one entered the following facts:

```
likes(peter, mary)
likes (paul, mary)
likes (mary, john)
```

and then asked

```
?-likes (peter, mary)
```

Prolog would respond by printing

```
yes.
```

In this trivial example, the word `likes` is the *predicate* that indicates that such a relationship exists between one object, `peter`, and a second object, `mary`. In this case Prolog says that it can establish the truth of the assertion that “Peter likes Mary” based on the three facts it has been given. In a sense, computation in Prolog is simply controlled logical deduction. One simply states the facts that one knows, and Prolog can tell whether any specific conclusion could be deduced from those facts. In knowledge engineering terms, Prolog’s control structure is logical inference.

Prolog is the best current implementation of logic programming, although a programming language cannot be strictly logical, because input and output operations necessarily entail some extralogical procedures. Thus, Prolog incorporates some basic code that controls the procedural aspects of its operations. However, this aspect is kept at a minimum, and it is possible to conceptualize Prolog strictly as a logical system.

Indeed, there are two Prolog programming styles: a *declarative* style and a *procedural* style. In declarative programming, one focuses on telling the system what it should know and relies on the system to handle the procedures. In procedural programming, one considers the specific problem-solving behavior the computer will exhibit. For instance, knowledge engineers who are building new expert system concerns themselves with procedural aspects of Prolog. Users, however, need not to worry about procedural details and are free simply to assert facts and ask questions.

One of the basic demands that AI language should satisfy is a good list processing. A *list* is virtually the only complex data structure that Prolog has to offer. Lists is said to have a head and a tail. The head is the first list item. The tail is the list composed of all remaining lists. The atom on the left of vertical bar is the list head, and the part to the right is the list tail.

The following example illustrates the way the list appending operation is performed in Prolog:

```
  append ([ ] , L,L).
  append ([ X|L1] ,L2, [ X|L3] )
      :- append (L1,L2,L3).
```

This simple Prolog program consists of two relations. The first says that the result of appending the empty list ([ ]) to any list L is simply L. The second relation describes an inference rule that can be used to reduce the problem of computing the result of an append operation involving a shorter list. Using this rule, eventually the problem will be reduced to appending the empty list, and the value is given directly in the first relation. The notation [X | L1] means the list whose first element is X, and the rest of which is L1. So the second relation says that the result of appending [X|L1] to L2 is [X|L3] provided that it can be shown that the result of appending L1 to L2 is L3.

FUNCTIONAL PROGRAMMING

Historically, the most popular AI language, Lisp (13–15), has been classified as a functional programming language in which simple functions are defined and then combined to form more complex functions. A function takes some number of arguments, binds those arguments to some variables, and then evaluates some forms in the context of those bindings.

Functional languages became popular within the AI community because they are much more problem-oriented than conventional languages. Moreover, the jump from formal specification to a functional program is much shorter and easier, so the research in the AI field was much more comfortable.

Functional programming is a style of programming that emphasizes the evaluation of expressions, rather than the execution of commands. The expressions in this language are formed by using functions to combine basic values. A functional language is a language that supports and encourages programming in a functional style.

For example, consider the task of calculating the sum of the integers from 1 to 10. In an imperative language such as C, this might be expressed using a simple loop, repeatedly updating the values held in an accumulator variable `total` and a counter variable `i`:

```
total = 0;
for (i=1; i<=10; ++i)
  total += i;
```

In a functional language, the same program would be expressed without any variable updates. For example, in Haskell, a non-strict functional programming language, the result can be calculated by evaluating the expression:

```
    sum[ 1..10]
```

Here `[ 1..10]` is an expression that represents the list of integers from 1 to 10, whereas `sum` is a function that can be used to calculate the sum of an arbitrary list of values.

The same idea could be used in strict functional languages such as SML or Scheme, but it is more common to find such programs with an explicit loop, often expressed recursively. Nevertheless, there is still no need to update the values of the variables involved as follows.

SML:

```
let fun sum i tot = if i=0 then tot else sum (i-1)
  (tot+i)
in sum 10 0
  end
```

Scheme:

```
  (define sum
    (lambda (from total)
      (if (= 0 from)
        total
        (sum (- from 1) (+ total from)))))
(sum 10 0)
```

It is often possible to write functional-style programs in an imperative language, and vice-versa. It is then a matter of opinion whether a particular language can be described as functional. It is widely agreed that languages like Haskell and Miranda are "purely functional," whereas SML and Scheme are not. However, there are some small differences of opinion about the precise technical motivation for this distinction. One definition that has been suggested says that "purely functional" languages perform all their computations via function application. This is in contrast to languages, such as Scheme and SML, that are predominantly functional but also allow computational effects caused by expression evaluation that persist after the evaluation is completed. Sometimes, the term "purely functional" is also used in a broader sense to mean languages that might incorporate computational effects, but without altering the notion of "function" (as evidenced by the fact that the essential properties of functions are preserved). Typically, the evaluation of an expression can yield a "task," which is then executed separately to cause computational effects. The evaluation and execution phases are separated in such a way that the evaluation phase does not compromise the standard properties of expressions and functions. The input/output mechanism of Haskell, for example, is of this kind.

There is also much debate in the functional programming community about the distinction and the relative merits of strict and non-strict functional programming languages. In a strict language, the arguments to a function are always evaluated before it is invoked, whereas in a non-strict language, the arguments to a function are not evaluated until their values are actually required. It is possible, however, to support a mixture of these two approaches like in some versions of the functional language Hope.

It is not possible to discuss the mathematical foundation of functional programming without a formal notation for function definition and application. The usual notation that is used in applicative functional languages is so-called λ–(lambda) calculus. It is a simple notation and yet powerful enough to model all of the more esoteric features of functional languages. The basic symbols in the λ–calculus are the variable names, λ, dot (·), and open and closed brackets. The general form for a function definition is

$$\lambda x.M$$

which denotes the function F such that for any value of x, $F(x) = M$, and the value of F can be computed on an argument N by substituting N into defining equation. A valid -expression, described in BNF notation, is as follows:

```
Expression : : = Variablename |
           Expression Expression |
           λ Variable_name_list. Expression |
           ( Expression )
```

The primary relevance of the λ–calculus to artificial intelligence is through the medium of Lisp. Lisp's creator McCarthy used λ–calculus as the bases of Lisp's notation for procedures. Since that time, other programming languages used the λ–calculus in a more pervasive way. However, from the point of view of artificial intelligence, the most important among functional languages is definitely Lisp, which will be given more attention in the following paragraphs.

Lisp

Lisp (*List* processing) is a family of languages with a long history. Early key ideas in Lisp were developed by John McCarthy during the Darthmouth Summer Research Project on Artificial Intelligence in 1956. Of the major programming languages still in use, only FORTRAN is older then Lisp. Since then it has grown to be the most commonly used language for Artificial Intelligence and Expert Systems programming, McCarthy's motivation was to develop an algebraic list processing language for artificial intelligence work.

John McCarthy, the language creator, describes the key ideas in Lisp as follows (13):

- Computing with symbolic expressions rather than numbers; that is, bit patterns in a computer's memory and registers can stand for arbitrary symbols, not just those of arithmetic.
- List processing, that is, representing data as linked-list structures in the machine and as multilevel lists on paper.
- Control structure based on the composition of functions to form more complex functions.
- Recursion as a way to describe processes and problems.
- Representation of LISP programs internally as linked lists and externally as multilevel lists, that is, in the same form as all data are represented.
- The function EVAL, written in LISP itself, serves as an interpreter for LISP and as a formal definition of language.

One major differences between Lisp and conventional programming languages (such as FORTRAN, Pascal, Ada, and C) is that Lisp is a language for symbolic rather than for numeric processing. Although it can manipulate numbers as well, its strength lies in being able to manipulate *symbols* that represent arbitrary objects from the domain of interest. Processing pointers to objects and altering data structures comprising other such pointers is the essence of symbolic processing. *Symbols*, also called *atoms* because of the analogy to the smallest indivisible units, are the most important data types in Lisp. Their main use is as a way of describing programs and data for programs. Symbol is a Lisp object. It has a name associated with them and several aspects or uses. First, it has a value, which can be accessed or altered using exactly the same forms that access or alter the value of a lexical variable. In fact, the methods of naming symbols are the same as those used for naming a lexical variable. In addition to a value, a symbol can have a property list, a package, a print name, and possibly a function definition associated with it. A property list is simply a list of indicators and values used to store properties associated with some objects that the symbol is defined by the programmer to represent. A print name is usually the string of characters that constitutes the identifier. A package is a structure that establishes a mapping between an identifier and a symbol. It is usually a hash table containing symbols. A function is normally associated with a lexical variable or a symbol. The symbol printed representation is as a sequence of alphabetic, numeric, pseudo-alphabetic, and special characters.

Other typical data types are lists, trees, vectors, arrays, streams, structures, and so on. Out of these data structures can be built representations for formulas, real-world objects, natural-language sentences, visual scenes, medical concepts, geographical concepts, and other symbolic data (even other Lisp programs). It is important to note that in Lisp it is data objects that are typed, not variables. Any variable can have any Lisp object as its value.

Historically, list processing was the conceptual core of Lisp (the name was taken from **Lis**t **p**rocessing). Lists in Lisp are reprinted in two basic forms. The external, visible, form of list is composed of an opening parenthesis followed by any number of symbolic expressions followed by a closing parenthesis. A symbolic expression can be a symbol or another list. Internally, a list is represented as a chain of CONS cells. The CONS cell is the original basic building block for Lisp data structures. Each CONS cell is composed of a CAR (the upper half, the “data” part) and CDR (the lower half, the “link” part). Lists are represented internally by linking CONS cells into chains by using the CDR of each cell to point to the CAR of the next cell. *CONS cells* can be linked together to form data structures of any desired size or complexity. NILL is the Lisp symbol for an “empty list,” and it is used to represent the Boolean value “false.”

The list appending function in Lisp would be as follows:

```
(DE APPEND (L1 L2)
    (COND (( NULL L1) L2)
    (( ATOM L1) (CONS L1 L2)
    ( TRUE (CONS (CAR L1) (APPEND (CDR L1) L2)0000
```

The Lisp function returns a list that is the result of appending L1 to L2. It uses the Lisp function CONS to attach one element to the front of a list. It calls itself recursively until all elements of L1 have been attached. The Lisp function CAR returns the first element of the list it is given and the function CDR returns the list it is given minus the first element. ATOM is true if its argument is a single object rather than a list.

Lisp relies on dynamic allocation of space for data storage. Memory management in Lisp is completely automatic, and the application programmer does not need to worry about assigning storage space. It manages storage space very efficiently and frees the programmer to create complex and flexible programs.

Lisp is a very good choice for an artificial intelligence project programming for several reasons. Most AI projects involve manipulation of symbolic rather than of numeric data, and Lisp provides primitives for manipulating symbols and collections of symbols. Lisp also provides automatic memory-management facilities so you do not need to write and debug routines to allocate and reclaim data structures. Lisp is extensible and contains a powerful macro facility that allows layers of abstraction.

Lisp’s lists can be of any size and contain objects of any data types (including other lists), so that programmers can create very complex data structures for representing abstract concepts such as object hierarchies, natural-language parse trees, and expert-systems rules. A collection of facts about an individual object can easily be represented in the property list that is associated with the symbol representing the concept. The property list is simply a list of attribute-value pairs. The fact that both data and procedures are represented as lists makes it possible to integrate declarative and procedural knowledge into a single data structure such as a property list.

Although symbols and lists are central to many artificial intelligence programs, other data structures such as arrays and strings are also often necessary.

The most natural Lisp control structure is recursion, which often represents the most appropriate control strategy for many problem-solving tasks.

Moreover, Lisp has the ability to treat its code as data. Researchers have exploited this capability of the Lisp program to modify themselves at runtime for research in machine learning, natural-language understanding, and other aspects of AI. Lisp implementation also encourages an interactive style of development ideally suited to exploring solutions for difficult or poorly specified problems. This is of crucial importance in an AI application area where the problems are too hard to be solved without human intervention.

Perhaps the most successful artificial intelligence application in the business world is expert system technology. Lisp is tailored to expert system creation because the language is rich, with its flexible list data type and excellent support for recursion, because it is extensible, and has facilities for rapid prototyping, which lets the implementer experiment with design and customize the expert system. Programmers can use the built-in list data type for easy creation of the data structures necessary to represent parameters, rules, premise clauses, conclusion actions, and other objects that constitute the knowledge base. Even the expert systems that could process a wealth of expertise about such esoteric disciplines as chemistry, biology, and avionics and handle a complex and rapidly changing process in real time have been built into Lisp. A Lisp-based expert system shell G2, manufactured by Gensym of Cambridge, MA, has been used all around the world for building real-time expert systems. Even the most complex applications like space-shuttle fault-diagnosis or launch-operation support have been Lisp-based (15).

It is important to remember that Lisp can be an explorative language rather than a product producing one. Lisp is a marvelous research language that gives a programmer the ability to create and experiment without paying attention to the data types of variables or the way memory is allocated.

Lisp has strengths and weaknesses. It has had some real successes but also some real problems that still have to be solved (14). Nevertheless, it should be part of the toolkit of any professional AI programmer, particularly of those who routinely construct a very large and complex expert system.

NEW AI LANGUAGES

AI languages that emerged after Lisp and Prolog can be roughly classified into several categories, with no strict boundaries between them (i.e., the same language can belong to more than one group):

- Lisp-like languages
- Prolog-like languages
- Hybrid languages
- Object-oriented languages
- Agent-oriented languages
- Semantic Web languages
- Specialized programming languages

Being the first AI programming language and one of the first programming languages at all, Lisp became an ancestor of numerous incarnations of this popular language [Allegro CL (16), ECoLisp (17), Kali Scheme (18), RScheme (19), Screamer (20), etc.]. Allegro CL powered by Common Lisp is an object-oriented development system used for dynamic servers, Web services, knowledge management, data mining, smart data integration, and manufacture control. ECoLisp (Embeddable Common Lisp) is an implementation of Common Lisp designed for embedding into C-based applications. Kali Scheme is a distributed implementation of Scheme dialect of Lisp that permits efficient transmission of higher order objects such as closures and continuations. The integration of distributed communication facilities within a higher order programming language engenders several new abstractions and paradigms for distributed computing. RScheme is another object-oriented, extended version of the Scheme language that can be translated to C easily, and then compiled with a standard C compiler to generate machine code. Screamer is an extension of Common Lisp augmented with practically all of the functionality of both Prolog and constraint logic programming languages.

Prolog has also been extended and augmented numerous times [Ciao Prolog (21), GNU Prolog (22), Amzi! Prolog (23), etc.]. Ciao Prolog is a complete Prolog system with a novel modular design that allows both restricting and extending of the language. Ciao Prolog extensions currently include feature terms (records), higher order functions, constraints, objects, persistent predicates, a good base for distributed execution (agents), and concurrency. Libraries support Web programming, sockets, and external interfaces (C, Java, TCL/Tk, relational databases, etc.). GNU Prolog offers various extensions useful in practice (global variables, OS interface, sockets, etc.). In particular, it contains an efficient constraint solver over finite domains (FDs). This facilitates constraint logic programming, combining the power of constraint programming with the declarative nature of logic programming. The key feature of the GNU Prolog solver is the use of a single (low-level) primitive to define all (high-level) FD constraints. Amzi! Prolog offers a variety of rule-based components (configuration and pricing rules for products and services, government regulations for industry, legal and tax rules for forms filing, workflow rules for optimal customer service, diagnostic rules for problem-solving, integrity-checking rules with databases, parsing rules for documents, tuning rules with performance-sensitive applications, advisory rules with help systems, business rules with any commercial application) that can be easily embedded into Web applications.

Although Prolog is the first and most popular logic programming language, some other languages also fall into this category [e.g. Gödel (24), Mercury (25), etc.]. Gödel is a declarative, general-purpose programming language belonging to the family of logic programming languages. It is a strongly typed language, where the type system is based on many-sorted logic with parametric polymorphism. Gödel supports infinite precision integers, infinite precision rational numbers, and floating-point numbers. It can solve constraints over finite domains of integers and linear rational constraints, and it supports processing of finite sets. It also has a flexible computation rule and a pruning operator that generalizes the commit of the concurrent logic programming languages. Mercury is a new logic programming language, which is purely declarative. Like Prolog and many other logic programming languages, it is a high-level language that allows programmers to concentrate on the problem rather than on the low-level details such as memory management. Unlike Prolog, which is oriented toward exploratory programming, Mercury is designed for the construction of large, reliable, efficient software systems by teams of programmers.

A few hybrid AI programming languages combine different programming paradigms, such as JEOPS (26), Kiev (37), CLIPS (28), and Jess (29). JEOPS, The Java Embedded Object Production System, is a programming language intended to give Java the power of production systems. JEOPS extends Java by adding forward chaining, first-order production rules through a set of classes designed to provide this language with some kind of declarative programming. These features enable the development of intelligent applications, such as software agents or expert systems. The Kiev programming language is a backward-compatible extension of Java that includes support for lambda-calculus closures (i.e., functional programming) and Prolog-like logic programming. CLIPS is an expert system tool that provides a complete environment for the construction of rule-and/or object-based expert systems. It provides a cohesive tool for handling a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented, and procedural. Jess is a rule engine and scripting environment written entirely in Java, originally inspired by CLIPS. It extends CLIPS by including backward chaining, working memory queries, and the ability to manipulate and reason upon Java objects.

Object-oriented programming languages [Eiffel, C++, Java, Python (30), Jython (31), etc.] are based on abstract data structures called classes. A class is used to represent data but also a behavior of that class, defined by its main operations often called methods. One of the most important characteristics of object-oriented programming languages is the ability to build hierarchies of classes and subclasses, where subclasses can inherit properties of their superclasses, which supports modularity and reusability. Among object-oriented languages, Java has recently become very popular in some areas of AI, especially for agent-based application, Internet search engines (and Internet applications in general), and data mining. Java supports automatic garbage collection and a multithreading mechanism, which makes it interesting from an AI perspective. Other popular object-oriented languages are Python and its Java version, Jython. These languages have been used in natural language processing, machine learning, constraint satisfaction, genetic algorithms, and expert systems. They are portable, interpretable, object-oriented languages. They use modules, classes, exceptions, very high-level dynamic data types, and dynamic typing. New built-in modules written in C, C++, or Java can be easily added.

During the last decade, Web applications and Web programming have been constantly gaining in popularity, and the AI community did not stay away from this trend. New fields such as intelligent agents and semantic Web emerged with the corresponding supporting programming languages. In the field of intelligent agents there are several programming languages such as APRIL (32) and Mozart (33). APRIL is a symbolic programming language, which is designed for writing mobile, distributed, and agent-based systems especially in a Web environment. It has the following advanced features such as a macro sublanguage, asynchronous message sending and receiving, code mobility, pattern matching, higher order functions, and strong typing. The Mozart Programming System is an advanced development platform for intelligent, distributed applications, which provides support in two areas: open distributed computing and constraint-based inference. Mozart is based on Oz language, which supports declarative programming, object-oriented programming, constraint programming, and concurrency as a part of a coherent whole, suitable for developing multiagent systems. Mozart is used for both general-purpose distributed applications as well as for hard problems requiring sophisticated optimization and inference abilities. It can be used to develop applications in different domains such as scheduling and time-tabling, placement and configuration, natural language and knowledge representation, multiagent systems, and sophisticated collaborative tools.

In the area of semantic Web several new ontology and schema languages such as XOL (34), SHOE (35), OML (36), RDFS (37), DAML+OIL (38), and OWL (39) have appeared. XOL is an XML-based ontology-exchange language, which was initially designed for exchange of bioinformatics ontologies, but it can also be used for ontologies in any domain. XOL is a language with the semantics of object-oriented knowledge representation systems but with XML syntax. SHOE (Simple HTML Ontology Extensions) is a small extension to HTML, which allows Web page authors to annotate their Web documents with machine-readable knowledge. OML (Ontology Markup Language) is based on description logics and conceptual graphs and allows representing concepts, organized in taxonomies, relations, and axioms in first-order logic. RDFS (RDF Schemas) is an extension of the RDF (Resource Description Framework). It is a declarative representation language influenced by ideas from knowledge representation (e.g., semantic nets, frames, and predicate logic) as well as database schema specification languages and graph data models. DAML+OIL is a semantic markup language for Web resources. It was built on earlier W3C standards such as RDF and RDF Schema, and it extends these languages with richer modeling primitives. DAML+OIL provides modeling primitives commonly found in frame-based languages. The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies on the Web. OWL is developed as a vocabulary extension of RDF and is derived from the DAML+OIL.

The OWL, Web Ontology Language, is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. OWL has more facilities for expressing semantics than XML, RDF, and RDF Schema (RDF-S), and thus OWL goes beyond these languages in its ability to represent machine-interpretable content on the Web. OWL is a revision of the DAML+OIL Web ontology language incorporating lessons learned from the design and application of DAML+OIL.

OWL is part of the growing stack of W3C recommendations related to the semantic Web:

- XML provides a surface syntax for structured documents but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with data types.
- RDF is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model, and these data models can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations amang classes (e.g., disjointedness), cardinality (e.g., "exactly one"), equality, richer typing of properties, characteristics of properties (e.g., symmetry), and enumerated classes.

OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users.

- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints.

- *OWL DL* supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time).
- *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. Ontology developers adopting OWL should consider which sublanguage best suits their needs. The choice between OWL Lite and OWL DL depends on the extent to which users require the more expressive constructs provided by OWL DL. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modeling facilities of RDF Schema (e.g., defining classes of classes or attaching properties to classes). When using OWL Full as compared with OWL DL, reasoning support is less predictable because complete OWL Full implementations do not currently exist. OWL Full can be viewed as an extension of RDF, whereas OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF. Every OWL (Lite, DL, Full) document is an RDF document, and every RDF document is an OWL Full document, but only some RDF documents will be a legal OWL Lite or OWL DL document.

Specialized programming languages, such as DHARMI (40), ECLiPSe (41), Esterel (42), and Shift (43), are used to solve specific problems in different AI domains. DHARMI is a high-level spatial language whose components are transparently administered by a background process called the Habitat. The language was designed to make modeling prototypes and handle living data. Programs can be modified while running, which is accomplished by blurring the distinction among source code, program, and data. ECLiPSe is a programming language for the cost-effective development and deployment of constraint programming applications, e.g., in the areas of planning, scheduling, resource allocation, timetabling, and transport. It contains several constraint problem-solver-libraries, a high-level modeling and control language, interfaces to third-party solvers, an integrated development environment, and interfaces for embedding into host environments. Esterel is a synchronous programming language dedicated to control-dominated reactive systems, such as control circuits, embedded systems, human–machine interface, or communication protocols. Shift is a programming language aimed at describing dynamic networks of hybrid automata. Such systems consist of components that can be created, interconnected, and destroyed as the system evolves. Components exhibit hybrid behavior, consisting of continuous-time phases separated by discrete-event transitions. Components may evolve independently, or they may interact through their inputs outputs, and exported events. The interaction network may also evolve.

Many of the recently emerged AI languages could not be covered here because of the imposed constraints on the length of the contribution.

BIBLIOGRAPHY

1. A. Barr, P. Cohen and E. Feigenbaum (eds.), *The Handbook of Artificial Intelligence*. Reading, MA: Addison Wesley, 1990.
2. E. Rich, *Artificial Intelligence*. New York: McGraw-Hill, 1990.
3. S. J. Russel, P. Norvig, *Artificial Intelligence: A Modern Approach*, Englewood Cliffs, NJ: Prentice Hall, 2002
4. P. Norvig, *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Philadelphia, PA: Morgan Kaufmann, 1991.
5. R. A. Kowalski and D. Kuehner, Linear Resolution with Selection Function. *Artif. Intell.*, **2**: 227–260, 1971.
6. R. A. Kowalski, *Logic for Problem Solving*. New York: Elsevier North Holland, 1979.
7. A. Colmerauer, H. Kanoui, P. Russel, and R. Passero, *Un Systeme de Communication Homme-Machine en Francais*. Marseille, Francec: Groupe de Recherche en Intelligence Artificielle, Universite d'Aix-Marseille, 1973
8. C. Green, Applications of Theorem Proving to Problem Solving, *Proc. IJCAI '69 Conference*, 1969, pp. 219–239.
9. P. J. Hayes, Computation and Deduction, *Proc. MFCS Conference*, 1973.
10. W. F. Clocksin and C. S. Mellish, *Programming in Prolog*. New York: Springer-Verlag, 1984.
11. L. Sterling and E. Shapiro, *The Art of Prolog, Second Edition: Advanced Programming Techniques (Logic Programming)*. Combridge, MA: MIT Press, 1994.
12. I. Bratko, *Prolog Programming for Artificial Intelligence*. Reading, MA: Addison-Wesley, 1986.
13. J. McCarthy, History of Lisp, in D. Wexelblat (ed.), *History of Programming Languages*. New York: Academic Press, 1978.
14. R. Gabriel, LISP: good news, bad news, how to win big, *AI Expert*, **6** (6): 31–40, 1991.
15. J. Keyes, LISP: The great contender, *AI Expert*, **7** (1): 24–28, 1992.
16. Franz Inc., Allegro CL. Available: http://www.franz.com.
17. ECoLisp. Available: http://www.di.unipi.it/~attardi/software.html.
18. H. Cejtin, S. Jagannathan, and R. Kelsey Higher-order distributed objects, *ACM Trans. Programming Languages Syst.*, 1995.
19. RScheme Development Group, RScheme. Available: http://www.rscheme.org.
20. Screamer. Available: http://www.cis.upenn.edu/~screamer-tools/home.htm.
21. D. Cabeza and M. Hermenegildo, Distributed WWW programming using (Ciao-)Prolog and the PiLLoW Library, *Theory Practice Logic Programming*, **1**(3): 251–282, 2001.
22. Daniel Diaz, GNU Prolog. Available: http://gnu-prolog.inria.fr.
23. Amzi, Amzi! Prolog. Available: http://www.amzi.com/products/prolog_products.htm.
24. P. M. Hill and J. W. Lloyd, *The Gödel Programming Language* Combridge, MA: MIT Press, 1994
25. D. Overton, Z. Somogyi, and P. Stuckey, Constraint-based mode analysis of Mercury, *Proc. of the Fourth International Conference on Principles and Practice of Declarative Programming*, Pittsburgh, PA, Oct 2002, pp. 109–120.
26. C. S. da Figueira Filho and G. L. Ramalho, 'JEOPS — The Java Embedded Object Production System', in *Lecture Notes in Artificial Intelligence*, vol. 1952. New York: Springer Verlag 2000, pp. 52–61.

27. Kiev compiler joined Open Source community, Kiev. Available: http://kiev.forestro.com/index.html.

28. J. Giarratano and G. Riley, *Expert Systems: Principles and Programming* 3rd ed Boston, MA: PWS Publishing, 1998.

29. E. Friedman-Hill, *Jess in Action*. Manning Publications, 2003.

30. A. Gauld, *Learn to Program Using Python*, Reading, MA: Addison-Wesley, 2001.

31. R. Hightower, *Python Programming with the Java Class Libraries*, Reading, MA: Addison-Wesley, 2002.

32. SourceForge.net, APRIL. Available: http://sourceforge.net/projects/networkagent/.

33. P. Van Roy, 'General overview of Mozart/Oz', *Proc. Second International Mozart/Oz Conference*, Charleroi, Belgium, 2004.

34. R. Karp et al., XOL: An XML Based Ontology Exchange Language (version 0.4). Available: www.ai.sri.com/~pkart/xol.

35. J. Heflin et al., SHOE: A Knowledge Representation Language for Internet Applications, Technical Report, CS-TR-4078 (UMIACS TR-99-71), Dept. of Computer Science, University of Maryland, 1999.

36. R. Kent, Conceptual Knowledge Markup Language (version. 0.2). Available: www.ontologies.org/CKML/CKML%200.2html.

37. D. Brickley and R. V. Guha, Resource Description Framework (RDF) Schema Specification, *W2C Proposed Recommendation*. Available: www.w3.org/TR/PR-rdf-schema .

38. Reference description of the DAML+OIL ontology markup language. Available: http://www.daml.org/2001/03/reference.

39. W3C, OWL Web Ontology Language Reference. Available: http://www.w3.org/TR/owl-ref/.

40. E. Wolf, DHARMI. Available: http://megazone.bigpanda.com/~wolf/DHARMI.

41. J. Schimpf and C. Gervet, ECLiPSe Release 5.7, *ALP Newsl.*, **17**(2), 2004.

42. G. Berry, The foundations of esterel, in G. Plotkin, C. Stirling, and M. Tofte (eds.), *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Combridge, MA: MIT Press, 2000.

43. California PATH UC. Berkeley, Shift. Available: http://path.berkeley.edu/SHIFT.

SANJA VRANES
The Mihailo Pupin Institute
Belgrade, Serbia and
Montenegro

AUTONOMY-ORIENTED COMPUTING (AOC)

INTRODUCTION

Programming Paradigms

In computer science, several major programming paradigms exist, including *imperative, functional, logic*, and *object-oriented* paradigms. The imperative paradigm embodies computation in terms of a program state and statements for updating the program state. An imperative program specifies a sequence of commands for a computer to perform. In contrast to the imperative paradigm, the functional paradigm handles computation by means of evaluating a series of functional expressions rather than the execution of commands. Both the imperative and the functional paradigms emphasize the mapping between inputs and outputs. On the other hand, the logic paradigm represents a set of rules and facts and then finds a solution based on automated theorem proving. The object-oriented paradigm may be regarded as an extension of the imperative paradigm by encapsulating variables and their operations into classes. In the field of multiagent systems. Shoham proposed an *agent-oriented programming paradigm*(1) where the basic unit is an agent characterized by a set of mental attributes, such as beliefs, commitments, and choices.

Autonomy-Oriented Computing (AOC): A New Paradigm

This article describes a new programming paradigm called *autonomy-oriented computing* (AOC), which focuses on the construct of synthetic autonomy in locally interacting entities, and uses the aggregated effects of entity interactions to generate desired global solutions or systems dynamics. The fundamental working mechanism of *self-organization* that underlies the AOC paradigm offers the advantages of natural formulation as well as scalable performance to characterize complex systems or to handle computationally hard problems that are distributed and large scale in nature.

AOC versus OOP and AOP. Table 1 presents a brief comparison among object-oriented programming (OOP), agent-oriented programming (AOP), and AOC. We elaborate their essential differences as follows.

1. *Components:* Unlike OOP and AOP, AOC builds on the basic units of autonomous entities and the environment in which they reside. Autonomous entities are characterized by their internal state and goals, and are equipped with an evaluation function, self-organizing behavior, and behavioral rules. Here, *autonomous* means that an entity behaves and makes decisions to change its internal state without control from other entities or an external *commander*. The autonomous entities in AOC are not as complex, i.e., having mental attributes, as agents in AOP.
2. *Computation Philosophy:* In both OOP and AOP, the computation is embodied as a process of message passing and responses among objects or agents. Particularly, in AOP, the computation involves some techniques of artificial intelligence, such as knowledge representation, inference, and reasoning mechanisms. AOP is suitable for modeling distributed systems (e.g., workflow management) and for solving distributed problems (e.g., transport scheduling) (2).

The computation in AOC, on the other hand, relies on the self-organization of autonomous entities. Entities directly or indirectly interact with each other or with their environment to achieve their respective goals. As entities simultaneously behave and interact, the results of entity interactions will be aggregated nonlinearly. The aim of an AOC system is to find a solution through the nonlinear aggregation of local interactions. For instance, in the case of computational problem solving, the emergent AOC system states may correspond to the solutions of a problem at hand. Generally speaking, AOC works in a bottom-up manner, somewhat like the working mechanism of nature. It is one of the reasons why AOC is well suited to characterize the behavior of complex systems and to solve computationally difficult problems.

MOTIVATIONS AND GOALS OF AUTONOMY-ORIENTED COMPUTING

Nature is full of complex systems that exhibit interesting behavior (3,4). Scientists and researchers are interested in modeling complex systems primarily for two reasons: (*1*) to discover and to understand the underlying working mechanism of a complex system concerned, and (*2*) to simulate and to use the observed complex behavior to formulate problem-solving strategies such as global optimization. In general, one wants to be able to *explain, predict, reconstruct*, and *deploy* a complex system. Computer scientists and mathematicians have, in the past, developed various nature-inspired algorithms to solve their problems at hand.

Common techniques for complex systems modeling can be divided broadly into top-down and bottom-up approaches. Top-down approaches start from the high-level system and use various techniques such as ordinary and partial differential equations (5–7). These approaches generally simplify the behavior of individuals in a complex system and tend to model the average case well, where local variations in the behavior are minimal and can be ignored (8). However, such approaches are not always applicable. For instance, the distribution of antibodies in the human

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Table 1. A Comparison Among OOP, AOP [Shoham (1)], and AOC

	OOP	AOP	AOC
Basic unit	object	agent	autonomous entity and environment
Attributes of basic unit	member variables and member functions	beliefs, decisions, capabilities, and obligations	states, evaluation function, goals, self-organizing behavior, and behavioral rules
Computation	message passing and response methods	message passing and response methods	(i) self-organization of autonomous entities and (ii) self-aggregation of entities behavior and interaction
Interaction	inheritance and messages among objects	messages among agents, including inform, request, offer, promise, decline, etc.	(i) interaction between entities and their environment and (ii) direct/indirect interaction among entities
Constraints on methods	none	honesty, consistency, etc.	behavioral rules
Suitability	modeling and decomposition	(i) modeling distributed systems and (ii) solving distributed problems (2)	(i) characterizing the behavior of complex systems and (ii) solving computationally difficult problems

immune system tends to be heterogeneous. Therefore, the use of differential equations cannot accurately describe the emergent behavior and dynamics in such a biologic system (9). On the other hand, bottom-up approaches (10–13) start from the smallest element of a complex system and take into consideration the following characteristics of entities in the system:

- *Autonomous:* Entities are individuals with bounded rationality that will act independently. No central controller exists to direct and coordinate individual entities.
- *Distributed:* Autonomous entities with localized reactive or decision-making capabilities are distributed in a heterogeneous environment, and interact locally among themselves to exchange their state information or to affect the states of others.
- *Emergent:* Distributed autonomous entities collectively exhibit complex (purposeful) behavior that is not present nor predefined in the behavior of entities in a system.
- *Adaptive:* Entities often change their behavior in response to changes in the environment in which they are situated.
- *Self-organized:* Through entity interactions, the system self-aggregates and amplifies certain outcomes of entity behavior. In other words, autonomous entities can self-organize to evolve some system-level emergent behavior.

AOC is a bottom-up approach to characterize the behavior of complex systems and to solve computationally difficult problems. Specifically, AOC has three goals: The first goal is *to reproduce life-like behavior in computation.* With detailed knowledge of the underlying mechanism, simplified life-like behavior can be used as the model for general-purpose problem-solving techniques. The second goal is *to study the underlying mecnanism of a real-world complex system* by hypothesizing and repeated experimentation. The end product of these simulations is a better understanding of, or explanations to, the real working mechanism of the modeled system. The third goal concerns *the emergence of a problem solver in the absence of human intervention*. In other words, self-adaptive, self-discovery algorithms are desired.

THREE GENERAL APPROACHES TO AUTONOMY-ORIENTED COMPUTING

To build an AOC-based model, we need to carry out the following steps:

1. Observe the macroscopic behavior of a natural system.
2. Design entities with desired synthetic behavior as well as an environment where entities reside.
3. Observe the macroscopic behavior of the artificial system.
4. Validate the behavior of the artificial system against the natural counterpart.
5. Modify (*2*) in view of (*4*).
6. Repeat (*3*)–(*5*) until the result is satisfactory.
7. Find out a model/origin of (*1*) in terms of (*2*) or apply the derived model to solve problems.

AOC is intended to reconstruct, to explain, and to predict the behavior of complex systems, which is hard to model or compute using top-down approaches. Local interactions between autonomous entities are the primary driving force of AOC. Formulation of an autonomy-oriented computational system involves an appropriate analogy, which normally comes from nature. Employing such an analogy, therefore, requires identification, abstraction, and reproduction of a certain natural phenomenon. The process of abstraction inevitably involves certain simplification of the natural counterpart. An abstracted version of some natural phenomena is the starting point of AOC, such that the problem at hand can be recasted. In the following subsections, we present three main approaches of AOC in detail.

AOC-By-Fabrication

AOC-by-fabrication is intended to replicate certain self-organizing behavior observable in the real world to form a general-purpose problem solver. The operating mechanism is more or less known and may be simplified during the modeling process. Research in artificial life (Alife) (10) is related to this AOC approach up to the behavior replication stage. Nature-inspired techniques such as ant systems (14) and evolutionary algorithms (15,16) are typical examples of such an extension.

Building on the experience of complex systems modeling, AOC algorithms are used to solve computationally hard problems. For example, in the commonly used version of genetic algorithms (16) that belong to the family of evolutionary algorithms, the process of sexual evolution is simplified to selection, recombination, and mutation, without the explicit identification, of male and female in the gene pool. Evolutionary programming (15) and evolution strategy (17), on the other hand, are closer to asexual reproduction with the addition of constraints on mutation and the introduction of mutation operator evolution, respectively. Despite these simplifications and modifications, evolutionary algorithms still capture the essence of natural evolution and are proven global optimization techniques.

As a demonstration of the AOC-by-fabrication approach, Liu et al. have developed an autonomy-oriented algorithm for image segmentation which identifies homogeneous regions within an image (18,19). Autonomous entities are deployed to a 2-D grid representation of an image. Each entity is equipped with the ability to assess the homogeneity of the region within a predefined locality. When a nonhomogeneous region is found, the autonomous entity will diffuse to another pixel in a certain direction within the local region. In contrast, when an entity locates a homogeneous region within the range of the pixel it currently resides, it replicates (breeds) itself to give rise to a certain number of offspring entities, and delivers them to its local region in a certain direction. The breeding behavior enables the newly created offspring to be distributed near the pixels where the region is found to be homogeneous, so that it is more likely to find the extension to the current homogeneous region. Apart from breeding, the entity will also label the pixel found to be homogeneous. If an autonomous entity fails to find a homogeneous region during its lifespan (a predefined number of steps) or wanders off the search space during diffusion, it will be marked inactive. In essence, the stimuli from the pixels will direct the autonomous entities to two different behavioral tracts: breeding and pixel labeling, or diffusion and decay. The directions of breeding and diffusion are determined by their respective behavioral vectors, which contain weights (between 0 and 1) of all possible directions. The weights are updated by considering the number of successful siblings in the respective directions. An entity is considered to be successful if it has found one or more pixels that are within a homogeneous region during its lifetime. A similar technique also has been applied to a feature extraction task such as border tracing and edge detection (20).

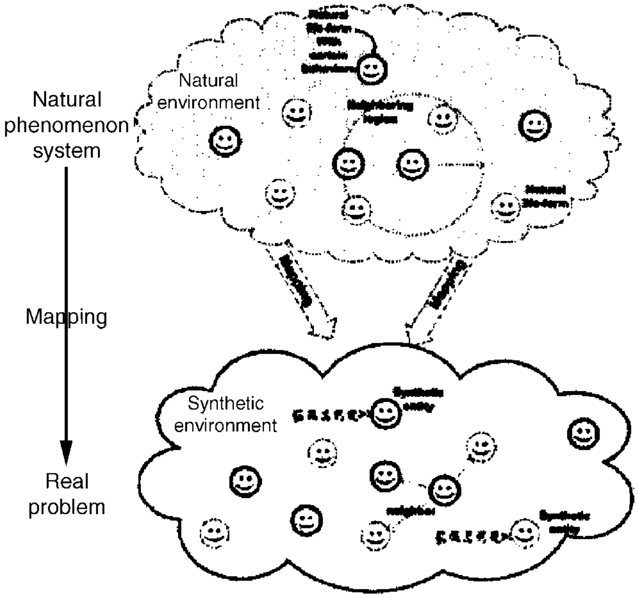


Figure 1. A schematic diagram of the AOC-by-fabrication approach, which is intended to build a mapping between a real problem and a natural phenomenon/system (21). In the figure, entities are characterized by a set of attributes (e.g., G, B, S, F, and R).

In general, the AOC-by-fabrication approach focuses on building a mapping between a real problem and a natural phenomenon/system, the working mechanism behind which is usually more or less known (see Fig. 1). In the mapping, the synthetic entities and their parameters (e.g., states or behavior) correspond, respectively, to the natural life-forms and their properties. Ideally, some special states of the natural phenomenon/system correspond to the solutions of the real problem. In particular, the AOC-by-fabrication approach has the following common characteristics:

1. There is a population of individuals, each of which is mainly characterized by its state, goals, self-organizing behavior, and behavioral rules. Individuals may be homogeneous or heterogeneous. Even in the homogeneous case, individuals may differ in certain detailed parameters.
2. The composition of the population may change over time, by the process analogous to birth (amplification of the desired behavior) and death (elimination of the undesired behavior). But, in some applications, the population has the fixed number of entities.
3. Tne interactions between individuals are local; neither global information nor central executive to control behavior or interactions is needed.

4. The environment acts as the center of information relating to the current status of the problem and as a place holder for information sharing between individuals.
5. The local goals of individuals drive the selection of local behavior at each step.
6. The global goal of the whole system is implicitly represented by the combination of all individuals' local goals or a universal fitness function, which measures the progress of the computation.

AOC-by-Prototyping

AOC-by-prototyping is a common AOC approach to understand the operating mechanism underlying a complex system. It models the system by simulating certain observed behavior, through characterizing the construct of synthetic autonomy in entities. Usually, AOC-by-prototyping involves a trial-and-error process to eliminate the difference between a prototype and its natural counterpart. Examples of this approach include the study of Internet ecology, traffic jams, and web log analysis. This AOC approach relates to multiagent approaches to complex systems in the field of distributed artificial intelligence.

With the help of a blueprint, we can build a model of a system in an orderly fashion. With insufficient knowledge about the mechanism of how the system works, it is difficult, if not impossible, to build such a model. Assumptions about the unknown workings have to be made to get the process started. We can verify the model by comparing its behavior with some observed behavior of the desired system. This process will have to be repeated several times before a good, probably not perfect, prototype is found. Apart from obtaining a working model of the desired system, an important by-product of the process is the discovery of the mechanisms that are unknown when the design process first started. This view is shared by researchers developing and testing theories about human cognition and social phenomena.

Liu et al. (22) and (23) have illustrated the AOC-by-prototyping approach to characterize the regularities of web surfing. They propose a web surfing model, which takes into account the characteristics of users, such as interest profiles, motivations, and navigation strategies. They view users as *information foraging entities* inhabiting the web space. The web space is a collection of websites connected by hyperlinks. Each website contains certain information contents, and each hyperlink between two websites signifies certain content similarity between them. The contents contained in a website are characterized by using a multidimensional *consent vector* where each component corresponds to the relative information weight on a certain topic. The web space is generated by assigning topics to each web page according to a certain statistical distribution. The variance of the distribution controls the similarity of the pages.

Liu and Zhang (23) have simulated users in the system by associating with them an *interest vector*, again generated randomly based on a statistical distribution with certain variance. Therefore, a parameter controls the degree of overlap in interest between users. When an information foraging entity finds certain websites in which the content is close to its interested topic(s), it will get more *motivated* to *surf* deeper. On the other hand, when the entity does not find any interesting information after some foraging steps or it has found enough contents to satisfy its interests, it will stop foraging and leave the web space. To model such a motivation-driven foraging behavior, they introduce a support function, which serves as the driving force for an entity to forage further. When the entity has found some useful information, it will get rewarded, and thus the support value will be increased. As the support value exceeds a certain threshold, which implies that the entity has obtained a sufficient amount of useful information, the entity will stop foraging. On the contrary, if the support value is too low, the entity will lose its motivation to forage further and thus leave the web space.

In summary, AOC-by-prototyping is used to uncover the working mechanism behind a natural phenomenon/system. In doing so, at the beginning, a preliminary prototype will be built to characterize or to simulate the natural counterpart. Then, by observing the difference between the natural phenomenon/system and the synthetic prototype, a trial-and-error process will be involved to fine-tune the prototype, especially the related parameters, to adjust the behavior of the synthetic entities. Figure 2 presents a schematic diagram of the process of AOC-by-prototyping.

In a sense, AOC-by-prototyping can be observed as an iterated application of AOC-by-fabrication with the addition of parameter tuning at each iteration. The difference between the desired behavior and the actual behavior of a prototype is the guideline to parameter adjustment. The process can be summarized, with reference to the summary of AOC-by-fabrication, as follows:

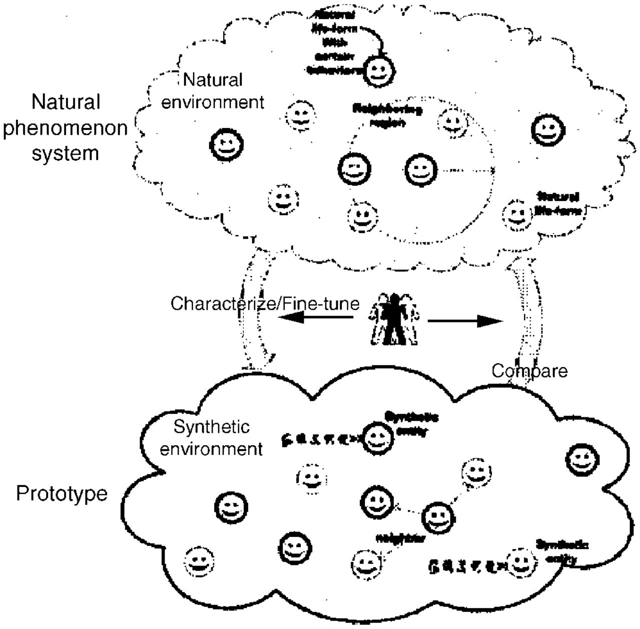


Figure 2. A schematic diagram of the process of AOC-by-prototyping, where the trial-and-error process, i.e., repeated *fine-tune* and *compare* steps, will be manually operated (as symbolized in the figure) (21).

1. The state, evaluation function, goals, self-organizing behavior, and behavioral rules of an entity can be changed from one prototype to the next.
2. The definition of the environment can also be changed from one version to the next.
3. There is an additional step to compare the synthetic model with the natural counterpart.
4. A new prototype is built by adopting (*1*) and (*2*) above, and by repeating the whole process.

AOC-by-Self-Discovery

AOC-by-self-discovery emphasizes the ability of an AOC-based computational system to find its own way to achieve what AOC-by-prototyping can do. The ultimate goal is to have a fully automated algorithm that can adjust its own parameters for different application domains. In other words, the AOC becomes autonomous. Some evolutionary algorithms that exhibit a self-adaptive capability are examples of this approach.

As inspired by diffusion in nature, Tsui and Liu (24,25) have developed an AOC-based method, called *evolutionary diffusion optimization* (EDO), to tackle a global optimization task. A population of entities is used to represent candidate solutions to the optimization problem at hand. The goal is to build a collective view of the landscape of the search space by sharing information among entities. Specifically, each entity performs a search in its local proximity, and captures the "direction" information of the landscape in a *probability matrix*—the likelihood estimate of success with respect to the direction of search in each object variable.

EDO defines three types of local behavior for each entity, namely *diffuse*, *reproduce*, and *aging*. Free-ranging entities that are searching for a position better than their birthplaces are called *active* entities. Those entities that already have become parents are called *inactive* entities.

Entities in EDO explore uncharted locations in the solution space by *diffusion*. *Rational move* refers to the kind of diffusion where an entity modifies its object vector by drawing a random number for each dimension of the object vector. Each random number is then used to choose between the set of fixed steps according to the probability matrix. An adjustment is then made by adding the product of the number of steps and the size of a step to the entry in question in the object vector. This process is repeated until all dimensions of the object vector are covered. The updated object vector then becomes the new position of the entity in the solution space. As an entity becomes older, it becomes more eager to find a better position. Therefore, it will decide probabilistically to act wild and to take a *random walk*. An entity will first choose randomly a direction of diffusion for each dimension, i.e., either no change, toward the upper bound, or toward the lower bound. In case a move is to be made, a new value between the chosen bound and the current value is then picked randomly. The process ends when all dimensions of the object vector are updated.

At the end of an iteration, the fitness of all active entities are compared with that of their parents, which have (temporarily) become inactive. All entities with higher fitness will *reproduce* via asexual reproduction—a reproducing entity replicates itself a number of times and sends the new entities off to new locations by rational moves. Parents and their offspring share the same probability matrix. Only when an entity becomes a parent then a new probability matrix will be created for it, which is an exact copy of the parent's updated one. Sharing the probability matrix between parents and siblings enables entities from the same family to learn from each other's successes as well as failures.

Aging is the process by which consistently unsuccessful entities are eliminated from the system. It is controlled by a *lifespan* parameter. Exceptions are granted to those entities whose ages have reached the set limit but that have the above-average fitness. On the other hand, entities whose fitness is at the lower 25% of the population will be eliminated before their lifespan expires.

Search algorithms need a scheme to implement the strategy that says "good" moves need to be rewarded while "bad" moves should be discouraged. All entities in EDO maintain a close link between parents and offspring via sharing the probability matrix. Therefore, it is very easy for EDO to implement the above strategy, and EDO has two feedback mechanisms for updating the probability matrix of the parent. *Positive feedback* increases the value of the entry in the probability matrix that corresponds to the "good" move. In contrast, *negative feedback* reduces the relevant probabilities that relate to the "bad" move. Although negative feedback is exercised after each diffusion, positive feedback can take place only after an entity has become a parent. Note also that all probabilities are normalized using their respective sum after updating.

EDO also adapts the *step-size* parameter, which determines the amount of change during *diffusion*, over time based on the performance measurement of the population. *Step-size* is reduced if the population has not improved over a period of time. Conversely, if the population has been improving continuously for some time, *step-size* is increased. The rationale is that the entities in the neighborhood of a minimum value need to make finer steps for careful exploitation, whereas using a large *step-size* during a period of continuous improvement attempts to speed up the search.

AOC-by-self-discovery can be used not only to build a mapping between a real problem and a certain natural phenomenon/system, but also to reveal the operating mechanism behind a natural phenomenon/system. In general, it combines the uses of AOC-by-fabrication and AOC-by-prototyping. In its implementation, AOC-by-self-discovery is the same as AOC-by-prototyping except that the process of trial-and-error in AOC-by-self-discovery is automated (see Fig. 3). The full automation of the prototyping process is achieved by having an autonomous entity to control another level of autonomous entities. The example described above shows that AOC-by-self-discovery is indeed a viable proposition. The steps for engineering this kind of AOC algorithm is the same as those stated, with the addition of one rule:

- System parameters are self-adapted according to some performance measurements.

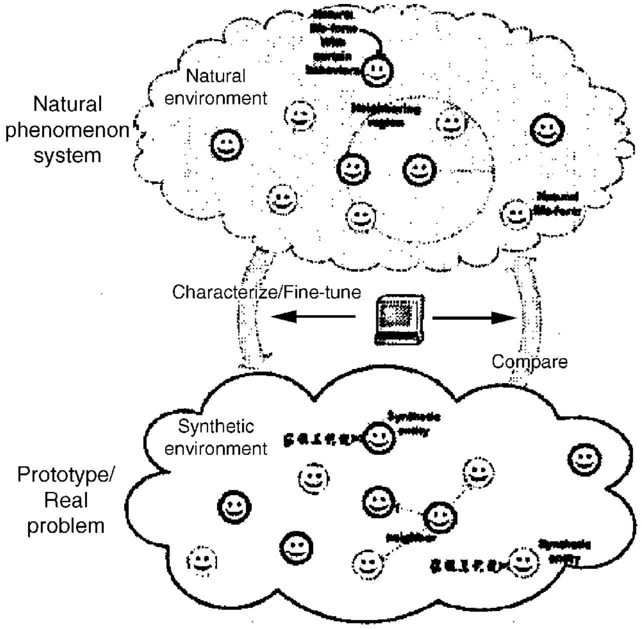


Figure 3. A schematic diagram of the process of AOC-by-self-discovery (21). As compared with AOC-by-prototyping (see Fig. 2), here the trial-and-error process, i.e., repeated *fine-tune* and *compare* steps, is automatically implemented by the system (as symbolized in the figure).

IMPORTANT CONCEPTS REVISITED

In this section, we will summarize the article by highlighting some of the key modeling concepts that we have introduced in the preceding descriptions of AOC approaches.

The Environment

As one of the main components in an AOC system, the environment usually plays three roles. First, the *environment* serves as the domain in which autonomous entities roam. This is a static view of the environment. Second, the environment acts as the *noticeboard* where the autonomous entities can read and/or post local information. In this sense, the environment can also be regarded as an indirect communication medium among entities. This is the dynamic view of the environment. Third, the environment also keeps the central clock that helps synchronize the behavior of all autonomous entities, if necessary.

Autonomous Entities

An autonomous entity possesses a way to find out what is going on with other entities as well as with the environment. As a result, it will modify its own state, exert changes to the environment, and/or affect other entities. Central to an autonomous entity is its *local behavior* and *behavioral rules* that govern how it should act or react to the information collected from the environment and its neighbors. The local behavior and behavioral rules determine to which state the entity will transit.

In different AOC systems, the neighbors of an entity can be fixed or dynamically changed.

At each moment, an entity is in a certain state. It, according to its behavioral rules, selects and performs its behavior to achieve certain goals with respect to its state. In doing so, it needs to interact with its neighbors and/or its environment to get the necessary information.

Interactions in an AOC System

The emergent behavior of an AOC system originates from its internal interactions. Generally speaking, there are two types of interactions, namely, *interactions between entities and their environment* and *interactions among entities*.

The interactions between an entity and its environment are implemented through state transitions as caused by the entity's self-organizing behavior.

Different AOC systems may have different ways of interactions among their entities. Those interactions can be categorized further into two categories: direct and *indirect* interactions. Direct interactions are implemented through direct state information exchanges among entities. In an AOC system with direct interactions, each entity can interact with its *neighbors*. Indirect interactions are implemented through the communication medium role of the environment. They can be carried out in two stages: (*1*) through the interactions between an entity and its environment, the entity will "transfer" its information to the environment, and (*2*) other entities will consider the information that has been "transfered" to the environment by the previous entity.

For further readings on AOC, e.g., more comprehensive surveys of related work, formal descriptions of the AOC approaches and formulations, and detailed discussions of examples as mentioned in this article, please refer to Refs. (21) and (26).

BIBLIOGRAPHY

1. Y. Shoham, Agent-oriented programming, *Artif. Intell.*, **60**(1): 51–92, 1993.
2. R. Kuhnel, Agent oriented programming with Java, I. Plander, editor, *Proceedings of the Seventh International Conference on Artificial Intelligence and Information - Control Systems of Robots (AIICSR'97)*, Singapore: World Scientific Publishing, 1997.
3. S. Kauffman, *At Home in the Universe: the Search for Laws of Complexity*, Oxford UK: Oxford University Press, 1996.
4. S. Rihani, *Complex Systems Theory and Development Practice: Understanding Non-linear Realities*, London: Zed Books, 2002.
5. K. Vajravelu, ed., *Differential Equations and Nonlinear Mechanics*, Dordrecht, the Netherlands: Kluwer Academic Publishers, 2001.
6. A. M. Blokhin, *Differential Equations and Mathematical Modelling*, Nova Science Publishers, 2002.
7. J. H. Vandermeer and D. E. Goldberg, *Population Ecology: First Principles*, Princeton, NJ: Princeton University Press, 2003.
8. J. Casti, *World Be Worlds: How Simulation is Changing the Frontiers of Science*, New York: John Wiley & Sons, 1997.
9. Y. Louzoun, S. Solomon, H. Atlan, and I. R. Cohen. The emergence of spatial complexity in the immune system. Los

Alamos Physics Archive, 2000. Available: (http://xxx.lanl.gov/html/cond-mat/0008133).

10. C. G. Langton, Artificial life, in C. G. Langton, ed., *Artificial Life*, Volume VI of SFI Studies in the Sciences of Complexity, Redwood City, CA: Addison-Wesley, 1989.
11. M. Resnick, *Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds*, Cambridge, MA: MIT Press, 1994.
12. J. Doran and N. Gilbert, Simulating societies: An introduction, in N. Gilbert, and J. Doran, ed., *Simulating Societies: The Computer Simulation of Social Phenomena*, London: UCL Press, 1994, pp. 1–18.
13. E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford UK: Oxford University Press, 1999.
14. M. Dorigo, V. Maniezzo, and A. Colorni, The ant system: Optimization by a colony of cooperative agents. *IEEE Trans. Syst. Man, and Cybernetics-Part B*, **26**(1): 1–13, 1996.
15. L. Fogel, A. J. Owens, and M. Walsh, *Artificial Intelligence Through Simulated Evolution*, New York: John Wiley & Sons, 1966.
16. J. H. Holland, *Adaptation in Natural and Artificial Systems*, Cambridge, MA: MIT Press, 1992.
17. H. P. Schwefel, *Numerical Optimization of Computer Models*, New York: John Wiley & Sons, 1981.
18. J. Liu, *Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization, and Adaptive Computation*, Singapore: World Scientific Publishing, 2001.
19. J. Liu, Y. Y. Tang, and Y. C. Cao, An evolutionary autonomous agents approach to image feature extraction. *IEEE Trans. on Evolut. Comput.*, **1**(2): 141–158, 1997.
20. J. Liu and Y. Y. Tang, Adaptive image segmentation with distributed behavior-based agents. *IEEE Trans. on Pattern Anal. and Machine Intell.,* **21**(6): 544–551, 1999.
21. J. Liu, X. Jin, and K. C. Tsui, *Autonomy Oriented Computing: From Problem Solving to Complex Systems Modeling*, Pordrect, the Netherland: Kluwer Academic Publishers/Springer, 2004.
22. J. Liu, S. Zhang, and J. Yang, Characterizing Web usage regulartities with information foraging agents. *IEEE Trans. on Knowledge and Data Engineering*, **16**(6): 566–584, 2004.
23. J. Liu and S. Zhang, Unveiling the origin of Web surfing regularities, in *Proceedings of iNET 2001*, 2001.
24. K. C. Tsui and J. Liu, Evolutionary diffusion optimization, Part I: Description of the algorithm, in X. Yao, ed., *Proceedings of the Congress on Evolutionary, Computation (CEC 2002)*, 2002, pp. 169–174.
25. K. C. Tsui and J. Liu, Evolutionary diffusion optimization, Part II: Performance assessment, in X. Yao, ed., *Proceedings of the Congress on Evolutionary Computation (CEC 2002)*, 2002, pp. 1284–1290.
26. J. Liu, X. Jin, and K. C. Tsui, Autonomy oriented computing (AOC): Formulating computatinal systems with autonomous components. *IEEE Trans. Sys. Man, and Cybernetics, Part A*, **35**(6): 879–902, 2005.

JIMING LIU
XIAOLONG JIN
KWOK CHING TSUI
Hong Kong Baptist University
Hong Kong

B

BIOINFORMATICS

INTRODUCTION

Almost all genetic information is stored in genome sequences. Genome sequences have been determined for many species, including humans, and thus huge amounts of sequence data have been obtained. Furthermore, a large amount of related data such as three-dimensional protein structures and gene expression patterns have also been produced. To analyze these data, we need new computational methods and tools. One major goal of *bioinformatics* is to develop such methods and tools, whereas another major goal of bioinformatics is to discover new biological knowledge using such kinds of tools. *Computational biology* is regarded as almost synonymous with bioinformatics. Although the difference between these two terms is very unclear, it seems that computational biology focuses on computational methods and on the actual process of analyzing and interpreting data.

Here, we overview important topics in bioinformatics: comparison of sequences, motif discovery, hidden Markov models (HMMs), protein structure prediction, kernel methods for bioinformatics, and analysis of gene expression patterns. Readers interested in more details may refer to the following textbooks (1–3) and handbook (4).

COMPARISON OF SEQUENCES

Comparison of two or multiple sequences is a fundamental and important problem in bioinformatics (1–3) because if two sequences of DNA or protein are similar to each other, it is expected that these DNAs or proteins have similar functions. Although there are many variants, we define here a basic version (global multiple alignment under the Sum-of-Pairs scoring scheme with linear gap costs) of the problem formally. Let $s_1, s_2, \ldots, s_k$ be sequences (i.e., strings) over a fixed alphabet Σ, where $k > 1$. Σ is usually either the set of bases {A, C, G, T} or the set of amino acids (i.e., $|\Sigma| = 20$). An *alignment* for $s_1, s_2, \ldots, s_k$ is obtained by inserting *gap symbols* (denoted by "– ") into or at either end of s_i such that the resulting sequences $s'_1, s'_2, \ldots, s'_k$ are of the same length l. Introduction of gaps is important because gaps correspond to insertions and deletions of bases (in DNA) or residues (in protein) that occur in the process of evolution. For example, consider three sequences CGCCAGTG, CGAGAGG, and GCCGTGG. Then examples of alignments are as follows:

M1	M2	M3
CGCCAGTG-	CGCCAGT-G	CGCCAGT-G-
CG--AGAGG	CG--AGAGG	-CGA-G-AGG
-GCC-GTGG	-GCC-GTGG	-GCC-GT-GG

In an alignment, letters in the same column correspond to each other: Bases or residues in the same column are regarded to have the same origin.

Let $f(x,y)$ be a function from $\Sigma' \times \Sigma'$ to $\mathcal{R}$ that satisfies $f(x,y) = f(y,x)$ and $f(x,-) = f(-,y) = -d$ for all $x, y \in \Sigma$ and $f(-,-) = 0$, where R denotes the set of real numbers and $\Sigma' = \Sigma \cup \{-\}$. The score for an alignment $\mathcal{M}$ is defined by

$$\mathrm{score}(\mathcal{M}) = \sum_{1 \le p < q \le k} \sum_{i=l}^{l} f(s'_p[i], s'_q[i])$$

where $s'_p[j]$ denotes the jth letter of s'_p. Then we define an *optimal alignment* to be an alignment with the maximum score. If we define $f(x,x) = 1$ and $f(x,-) = f(-,x) = -1$ for $x \in \Sigma$, $f(x,y) = -1$ for $x \neq y$, and $f(-,-) = 0$, the scores of $\mathcal{M}_1$ and $\mathcal{M}_2$ are both 3, and the score of $\mathcal{M}_3$ is -5. In this case, both $\mathcal{M}_1$ and $\mathcal{M}_2$ are optimal alignments. The alignment problem is to find an optimal alignment. It is called the *pairwise alignment* problem if $k = 2$, and otherwise, it is called the *multiple alignment* (multiple sequence alignment) problem.

An optimal alignment for two sequences can be computed in $O(n^2)$ time using a simple *dynamic programming* algorithm (1), where n is the larger length of the two input sequences. The following procedure gives the core part of the algorithm:

$$\begin{aligned} D[i][0] &= -i \cdot d, \quad D[0][j] = -j \cdot d \\ D[i][j] &= \max(D[i-1][j] - d, \quad D[i][j-1] - d, \\ &\quad D[i-1][j-1] + f(s_1[i], s_2[j])) \end{aligned}$$

where $D[i][j]$ corresponds to the optimal score between $s_1[1] \ldots s_1[i]$ and $s_2[1] \ldots s_2[j]$. An optimal alignment can also be obtained from this matrix by using the *traceback* technique (1). Many variants are proposed for pairwise alignment, among which local alignment (the Smith–Waterman algorithm) with affine gap costs is most widely used (1,3).

This algorithm is fast enough to compare two sequences. However, in the case of a *homology search* (search for homologous genes or proteins), it is required to find sequences in a database that are similar to a given sequence. For example, suppose that one determines a new DNA sequence of some gene in some organism and wants to know the function of the gene. He or she tries to find similar sequences in other organisms using a database (such as GenBank 5), which stores all known DNA sequences. If a similar sequence whose function is known is found, then he or she can infer that the new gene has a similar function. Thus, in a homology search, pairwise alignment between a query sequence and all sequences in a database should be performed. Since more than several hundreds of thousands of sequences are usually stored in a database, simple application of pairwise alignment would take a lot of time. Therefore, several heuristic methods have been proposed to speed up a database search, among which FASTA and BLAST are widely used (3). Most heuristic methods employ the following strategy: Candidate sequences having fragments (short length substrings) that are the same as (or very similar to) a fragment of the query sequence are first searched, and then

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

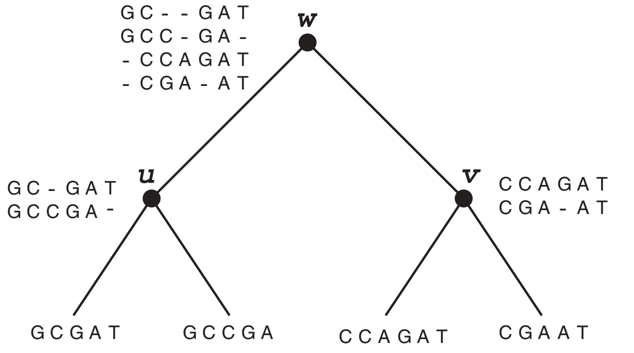


Figure 1. Progressive alignment. Pairwise sequence alignment is performed at nodes u and v, whereas profile alignment is performed at node w.

pairwise alignments are computed using these fragments as anchors. Using these methods, a homology search against several hundreds or thousands of sequences can be done in around a few minutes.

The dynamic programming algorithm above can be extended for cases of $k > 2$, but it is not practical because it takes ($O(n^k)$ time or more. Indeed, multiple alignment is known to be NP-hard if k is a part of the input (i.e., k is not fixed) (6). Thus, a variety of heuristic methods have been applied to multiple alignment that include simulated annealing, evolutionary computation, iterative improvement, branch-and-bound search, and stochastic methods (1,7).

The most widely used method employs the *progressive strategy*(1,8). In this strategy, we need an alignment between two profiles, where a profile corresponds to the result of an alignment. Alignment between profiles can be computed in a similar way to pairwise alignment: Each column is treated as if it were a letter in pairwise alignment. An outline of the progressive strategy used in CLUSTAL-W (8) is as follows (see also Fig. 1):

(i) Construct a distance matrix for all pairs of sequences by pairwise sequence alignment, followed by conversion of alignment scores into distances using an appropriate method.
(ii) Construct a rooted tree whose leaves correspond to input sequences, using a method for phylogenetic tree construction.
(iii) Progressively perform sequence–sequence, sequence–profile, and profile–profile alignment at nodes in order of decreasing similarity.

Although we have assumed that score functions were given, derivation or optimization of score functions is also important. Score functions are usually derived by taking log-ratios of frequencies (9). Since score functions obtained in this manner are not necessarily optimal, some methods have been proposed for optimizing score functions (10).

MOTIF DISCOVERY

It is very common that sequences of genes or proteins with a common biological function have a common pattern of sequences. For example, promoter regions of many genes in Eukaryotes have “TATAA” as a subsequence. Such a pattern is called a *motif* (more precisely, a sequence motif) (11,12). Motif discovery from sequences is important for inference of functions of proteins and for finding biologically meaningful regions (such as transcription factor binding sites) in DNA sequences. Although there are various ways of defining motif patterns, these can be broadly divided into *deterministic patterns* and *probabilistic patterns*(11).

Deterministic patterns are usually described using syntax similar to regular expressions. For example, “[AG]-x(2,5)-C” is a pattern matching any sequence containing a substring starting with A or G, followed by between two and five arbitrary symbols, followed by C. Deterministic patterns are usually discovered from positive examples (sequences having a common function) and negative examples (sequences not having the function). Although discovery of deterministic patterns is computationally hard (NP-hard) in general, various machine learning techniques have been applied (11).

Probabilistic patterns are considered to be more flexible than deterministic patterns, although deterministic patterns are easier to interpret. Probabilistic patterns are represented using statistical models. For example, *profiles* (also known as *weight matrices* or *position-specific score matrices*) and *hidden Markov models* are widely used (1). Here, we introduce profiles. A profile is a function $w(x, j)$ from $\Sigma \times [1\ldots L]$ to $\mathcal{R}$, where L denotes the length of subsequences corresponding to a motif, and $[1\ldots L]$ denotes the set of integers between 1 and L. It should be noted in this case that the lengths of motif regions (i.e., subsequences corresponding to a motif) must be the same and that gaps are not allowed in the motif regions. A profile can be represented by a two-dimensional matrix of size $L \cdot |\Sigma|$. A subsequence $s[i]\ldots s[i + L{-}1]$ of s is regarded as a motif if $\Sigma_{j=1,\ldots,L} w(s[i + j - 1], j)$ is greater than a threshold θ.

Various methods have been proposed in order to derive a profile from sequences $s_1, s_2, \ldots, s_k$ having a common function. One common approach is to select a subsequence t_i from each sequence s_i such that the relative entropy score (the average information content) is maximized (see Fig. 2). The relative entropy score is defined by

$$\frac{1}{L}\sum_{j=1}^{L}\sum_{a\in\Sigma} f_j(a)\log_2\frac{f_j(a)}{p(a)}$$

where $f_j(a)$ is the frequency of appearances of symbol a at the$j$th position in the subsequences (i.e., $f_j(a) = |\{i|t_i|[j] = a\}|/k$) and$p_a$ is the background probability of symbol a. In a simplest case, we may use $p(a) = \frac{1}{|\Sigma|}$.

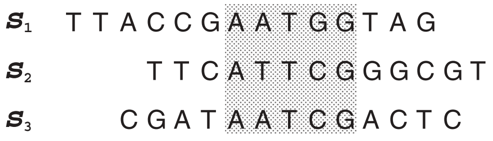


Figure 2. Motif discovery based on relative entropy score. Shaded regions correspond to t_1,t_2 and t_3 (L = 5).

Maximization of this relative entropy score is known to be NP-hard (13). On the other hand, several heuristic algorithms have been proposed based on statistical algorithms such as the expectation maximization (EM) method (14) and Gibbs sampling (12).

HIDDEN MARKOV MODELS

HMMs were originally developed in the areas of statistics and speech recognition. In the early 1990s, HMMs were applied to multiple sequence alignment (15) and protein secondary structure prediction (16). After that, the HMM and its variants were applied to solve various problems in bioinformatics. For example, HMMs have been applied to gene finding (identification of subsequences in DNA that encode genes), motif finding, and recognition of protein domains (1). One advantage of HMMs is that they can provide more detailed generative models for biological sequences than sequence alignment, although HMMs usually require longer CPU time than sequence alignment, and HMMs often need to be trained. Here, we briefly review the HMM and its application to bioinformatics. Readers interested in the details may refer to Ref. 1.

An HMM is defined by quadruplet (Σ, Q, A, E), where Σ is an alphabet (a set of symbols), $Q = \{q_0, \ldots, q_m\}$ is a set of states, $A = (a_{kl})$ is an $(m + 1) \times (m + 1)$ matrix of state transition probabilities, and $E = (e_k(b))$ is an $(m + 1) \times |\Sigma|$ matrix of emission probabilities. To be more precise, a_{kl} denotes the transition probability from state q_k to q_l, and $e_k(b)$ denotes the probability that a symbol b is emitted at state q_k. Θ denotes the collection of parameters of an HMM [i.e., $\Theta = (A, E)$], where we assume that Σ and Q are fixed based on the nature of the problem.

A *path*$\pi = \pi[1] \ldots \pi[n]$ is a sequence of (indices of) states. The probability that both π and a sequence $s = s[1] \ldots s[n]$ over Σ are generated under Θ is defined by

$$P(s, \pi | \Theta) = \prod_{i=1}^{n} a_{\pi[i-1]\pi[i]} e_{\pi[i]}(s[i]),$$

where $\pi[0] = 0$ is introduced as a fictitious state, a_{0k} denotes the probability that the initial state is q_k, and $a_{k0} = 0$ for all k.

There are three important algorithms for using HMMs: the *Viterbi algorithm*, the *forward algorithms*, and the *Baum–Welch algorithm*. The Viterbi algorithm computes the most plausible path for a given sequence. Precisely, it computes $\pi^*(s)$ defined by

$$\pi^*(s) = \arg\max_{\pi} P(s, \pi | \Theta)$$

when sequence s is given. The forward algorithm computes the probability that a given sequence is generated. It computes

$$P(s|\Theta) = \sum_{\pi} P(s, \pi | \Theta)$$

when sequence s is given. Both the Viterbi and forward algorithms are based on the dynamic programming technique. Each of these algorithms works in $O(nm^2)$ time for fixed Σ.

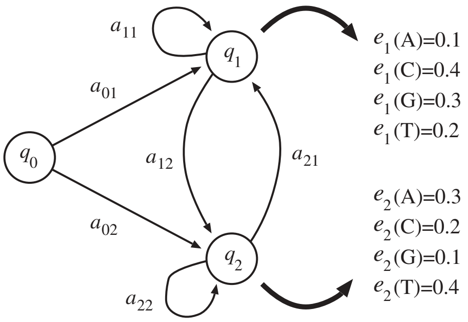


Figure 3. Example of an HMM.

Figure 3 shows an example of an HMM. Suppose that $a_{kl} = 0.5$ for all $k, l \, (l \neq 0)$. Then, for sequence $s =$ ATCGCT, we have $\pi^*(s)$-0221112 and $P(s, \pi^*|\Theta) = 0.5^6 \cdot 0.4^4 \cdot 0.3^2$.

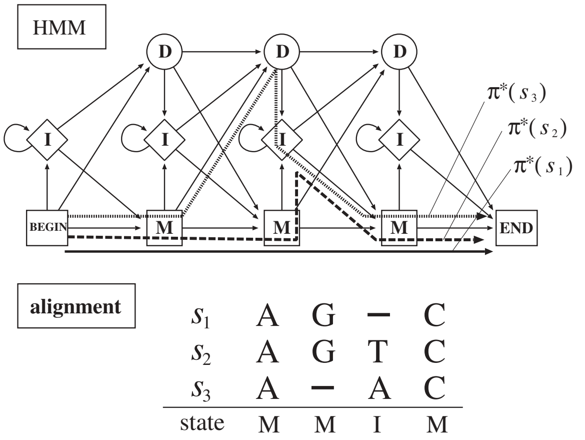
We assumed in both algorithms that Θ was fixed. However, it is often required to train HMMs from sample data. The Baum–Welch algorithm is used to estimate Θ when a set of sequences is given. Suppose that a set of k sequences $\{s_1, \ldots, s_k\}$ is given. The likelihood of observing these k sequences is defined to be $\prod_{j=1}^{k} P(s_j|\Theta)$ for each Θ. Based on the maximum likelihood method, we want to estimate Θ, which maximizes this product (likelihood). That is, the goal is to find an optimal set of parameters Θ^* defined by

$$\Theta^* = \arg\max_{\Theta} \prod_{j=1}^{k} P(s_j|\Theta)$$

However, it is computationally difficult to find an optimal set of parameters. Therefore, various heuristic methods have been proposed for finding a locally optimal set of parameters. Among them, the Baum–Welch algorithm is most widely used. It is a kind of EM algorithm, and it computes a locally optimal set of parameters using an iterative improvement strategy. How to determine the architecture of the HMM is also an important problem. Although several approaches were proposed to automatically determine the architectures from data (see Sections 3.4 and 6.5 of Ref. 1), the architectures are usually determined manually based on knowledge about the target problem.

HMMs are applied to bioinformatics in various ways. One common way is the use of *profile HMMs*. Recall that a profile is a function $w(x, j)$ from $\Sigma \times [1..L]$ to $\mathcal{R}$, where L denotes the length of a motif region. Given a sequence s, the score for s was defined by $\Sigma_{j=1,\ldots,L} w(s[j], j)$. Although profiles are useful for detecting short motifs, they are not so useful for detecting long motifs or remote homologs (sequences having weak similarities) because insertions or deletions are not allowed. A profile HMM is considered to be an extension of a profile in which insertions and deletions are allowed.

A profile HMM has a special architecture as shown in Fig. 4. The states are classified into three types: *match states* (M), *insertion states* (I), and *deletion states* (D). A match state corresponds to one position in a profile. A symbol b is emitted from a match state q_j with probability



s_1	A	G	–	C
s_2	A	G	T	C
s_3	A	–	A	C
state	M	M	I	M

Figure 4. Computation of multiple alignment using a profile HMM.

$e_j(b)$. A symbol b is also emitted from any insertion state q_i with probability $p(b)$, where $p(b)$ is the background frequency of occurrence of the symbol b. No symbol is emitted from any deletion state. Using a profile HMM, we can also obtain multiple alignment of sequences by combining $\pi^*(s_j)$ for all input sequences s_j. Although alignments obtained by profile HMMs are not necessarily optimal, they are meaningful from a biological viewpoint (1).

Many variants and extensions of HHMs have also been developed and applied in bioinformatics. For example, stochastic context-free grammar was applied to prediction of RNA secondary structures (1).

PROTEIN STRUCTURE PREDICTION

Protein structure prediction is the problem of a given protein sequence (target sequence), inferring its three-dimensional structure (17,18). This problem is important since determination of the three-dimensional structure of a protein is much harder than determination of its sequence, and the structure provides useful information on the function and interactions of the protein, which cannot be observed directly from the sequence. Various kinds of approaches exist for protein structure prediction (18), where the major approaches (to be explained below) include *ab initio, homology modeling, secondary structure prediction,* and *protein threading*.

Since many methods have been proposed, a type of contest or meeting called CASP (community-wide experiment on the critical assessment of techniques for protein structure prediction) has been held every two years since 1994 (18). CASP has been playing an important role in the progress of protein structure prediction technologies.

Ab Initio

This approach tries to predict protein structures based on basic principles of physics. For example, energy minimization and molecular dynamics have been applied. However, this approach is currently limited to prediction of small protein structures because it requires enormous computational power. A combination of an ab initio approach and a statistical approach has also been studied (19).

Homology Modeling

Two proteins tend to have similar structures if their sequences are similar enough (although there are exceptional cases). Based on this fact, we have an outline of the structure (backbone structure) from the result of a sequence alignment between the target sequence and the template sequence whose structure is known and that is similar enough to the target sequence. After obtaining a backbone structure, methods such as energy minimization or molecular dynamics are applied to predicting a detailed structure.

Secondary Structure Prediction

In secondary structure prediction, each amino acid of a protein structure is predicted to be one of three classes: α-helix, β-strand, or other, depending on its local shape. Since it is a simple classification problem, many methods in artificial intelligence have been applied. It is easy to see that random prediction (randomly output one of three classes) will achieve 33.3% accuracy. The best existing methods achieve 70~80% accuracy, some of which are based on artificial neural networks (20).

Protein Threading

It is useful in protein structure prediction to measure the compatibility between an input protein sequence and a known protein structure. For that purpose, we usually compute an *alignment between a sequence and a structure* (see Fig. 5). This problem is called *protein threading*. Many algorithms have been proposed for protein threading (3,18,21). Based on score functions, these can be grouped into two classes: threading with profiles and threading with pair score functions.

Threading with Profiles

The score function for this type of threading does not explicitly include the pairwise interaction preferences so that score functions are treated as profiles. A simple dynamic programming algorithm can be used to compute an optimal threading as in the case of pairwise sequence

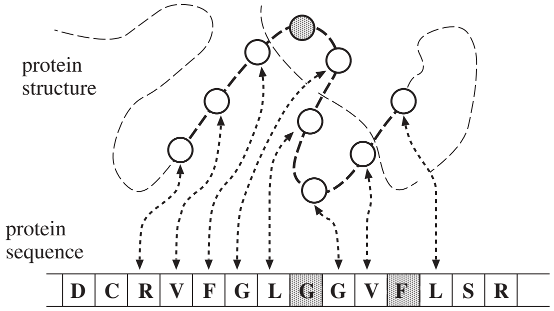


Figure 5. In protein threading, an alignment between a query sequence and a template structure is computed. Shaded parts correspond to gaps.

alignment. However, this method is not so useful unless there is a structure whose sequence has some similarity with an input sequence.

Threading with Pair Score Functions

The score function for this type of threading includes the pairwise interaction preferences. Since protein threading is proven to be NP-hard (22), various methods have been proposed based on heuristics, which include double dynamic programming, frozen approximation, Monte-Carlo sampling, and evolutionary computation (21). Although these methods are not guaranteed to find optimal solutions, several other methods have been proposed in which optimal solutions are guaranteed to be found under some assumptions (e.g., gaps are not allowed in α-helices or β-strands). The first practical algorithm with guaranteed optimal solutions was proposed by employing an elaborated branch-and-bound procedure (23). However, it could not be applied to large protein structures. In 2003, a protein threading method (with pairwise interaction preferences) formulated as a large-scale *integer programming* (IP) was proposed (21). The IP formulation is then relaxed to a linear programming (LP) problem. Finally, an optimal solution is obtained from the LP by using a branch-and-bound method. Surprisingly, the relaxed LP programs generated integral solutions (i.e., optimal solutions) directly in most cases.

From Generative to Discriminative Models

Most methods described in the previous sections are generative: Such objects as alignments and predicted structures are generated. On the other hand, many problems require discriminative approaches: It is required for predicting to which class a given object belongs. For that purpose, various techniques in pattern recognition, statistics, and artificial intelligence have been applied, including but not limited to, neural networks and decision trees. Among these techniques, *support vector machines* (SVMs) and kernel methods (24,25) are beginning to be recognized as one of the most powerful approaches to discriminative problems in bioinformatics (25,26), since the prediction accuracies are in many cases better than other methods and it is easy to apply SVMs; once a suitable kernel function is designed, efficient software tools for SVMs are available. Thus, in this section, we focus on SVMs and kernel methods (see also Fig. 6).

SVMs are basically used for binary discrimination. Let *POS* and *NEG* be the sets of *positive examples* and *negative examples* in a training set, where each example is represented as a point in d-dimensional Euclidean space. Then an SVM tries to find an optimal hyperplane h such that the distance between h and the closest point to h is the maximum (i.e., the margin is maximized) under the condition that all points in *POS* lie above h and all points in *NEG* lie below h. Once such h is obtained, a new test data point is predicted as positive (respectively negative) if it lies above h (respectively below h). If h does not exist, which completely separates *POS* from *NEG*, it is required to optimize the *soft margin*, which is a combination of the margin and the classification error. In order to apply an SVM effectively, it is important to design a *kernel function* suitable for an application problem, where a kernel takes two objects (e.g., two sequences) as inputs and provides a measure of similarity between these objects. Kernel functions can also be used in principal component analysis (PCA) and canonical correlation analysis (CCA) (25–27). In the rest of this section, we briefly review the kernel functions developed for biological sequence analysis.

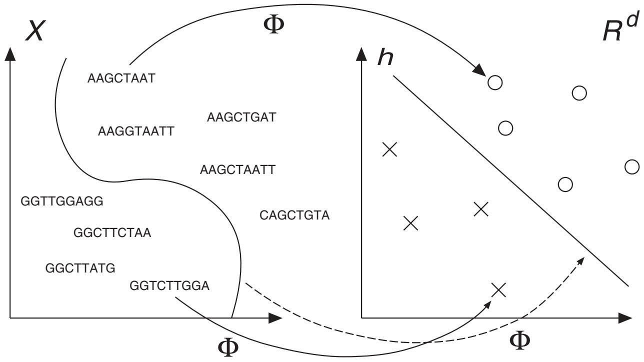


Figure 6. Kernel function and support vector machine. In the right figure, circles denote positive examples and crosses denote negative examples.

We consider a space $\mathcal{X}$ of objects. For example, $\mathcal{X}$ can be a set of DNA or protein sequences. We also consider a *feature map* ϕ from $\mathcal{X}$ to $\mathcal{R}^d$, where $d \in \{1,2,3,\ldots\}$ (we can even consider infinite-dimensional space (Hilbert space) instead of $\mathcal{R}^d$). We define a kernel K from $\mathcal{X} \times \mathcal{X}$ to $\mathcal{R}$ by

$$K(x,y) = \phi(x) \cdot \phi(y)$$

where $\phi(x) \cdot \phi(y)$ is the inner product between vectors $\phi(x)$ and $\phi(y)$. It is known that if a function K from $\mathcal{X} \times \mathcal{X}$ to $\mathcal{R}$ is symmetric [i.e., $K(x,y) = K(y,x)$] and positive definite (i.e., $\Sigma_{i=1}^{n}\Sigma_{j=1}^{n} a_i a_j K(x_i,x_j) \geq 0$ holds for any $n > 0$, for any $(a_1,\ldots,a_n) \in \mathcal{R}$, and for any $(x_1,\ldots,x_n) \in \mathcal{X}^n$), K is a valid kernel (i.e., some $\phi(x)$ exists such that $K(x,y) = \phi(x) \cdot \phi(y)$).

In bioinformatics, it is important to develop kernel functions for sequence data. One of the simplest kernel functions for sequences is the *spectrum kernel* (28). Let k be a positive integer. We define a feature map $\phi_k(x)$ from a set of sequences over Σ to $\mathcal{R}^{|\Sigma^k|}$ by

$$\phi_k(x) = (occ(s,x))_{s \in \Sigma^k}$$

where $occ(s, x)$ denotes the number of occurrences of substring s in string x. The k-spectrum kernel is then defined as $K(x,y) = \phi_k(x) \cdot \phi_k(y)$. Although the number of dimensions of $\mathcal{R}^{|\Sigma^k|}$ is large, we can compute $(K(x,y)$ efficiently (in $(O(kn)$ time) using a data structure named *suffix trees* without computing $\phi_k(x)$ (28). Here, we consider the example case of $k = 2$ and $\Sigma = \{A, C\}$. Then we have $\phi_2(x) = (occ(AA,x), occ(AC,x), occ(CA,x), occ(CC,x))$. Thus, for example, we have $K(ACCAC, CCAAAC) = 4$ since $\phi_2(ACCAC) = (0,2,1,1)$ and $\phi_2(CCAAAC) = (2,1,1,1)$. The spectrum kernel was extended to allow small mismatches (*mismatch kernel*) (29) and to use motifs in place of substrings (*motif kernel*) (30).

Several methods have been proposed that combine HMMs with SVMs. The *SVM-Fisher kernel* is one such kernel (31). To use the SVM-Fisher kernel, we first train a profile HMM with positive training data using the Baum–Welch algorithm. Then we compute a feature vector for each input sequence s as follows. Let m be the number of match states in the profile HMM. $E_i(a)$ denotes the expected number of times that $a \in \Sigma$ is observed in the ith match state for s, $e_i(a)$ denotes the emission probability of $a \in \Sigma$, and θ_{al} is the coordinate corresponding to a of the lth ($l \in \{1,\ldots,9\}$) Dirichlet distribution (1). It is known that $E_i(a)$ can be computed using the forward and backward algorithms (1,3). Then the feature vector $\phi_F(s)$ is defined by

$$\phi_F(s) = \left(\sum_{a \in \Sigma} E_i(a)[\frac{\theta_{al}}{e_i(a)} - 1] \right)_{(l,q_i) \in \{1,\ldots,9\} \times Q_{\mathrm{MATCH}}}$$

which is finally combined with the radial basis function kernel. As another approach to combining HMMs and SVMs, the local alignment kernel was developed based on the pair HMM model (a variant of the HMM) (32).

Kernels for other objects have also been proposed. The *marginalized kernel* was developed based on the expectation with respect to hidden variables (33). The marginalized kernel is defined in a very general way, and thus, it can be applied to nonsequence objects. For example, the marginalized graph kernel was developed and applied to classification of chemical compounds (34).

ANALYSIS OF GENE EXPRESSION PATTERNS

Genetic information stored in genes is used to synthesize proteins. Each gene usually encodes one or a few kinds of proteins. Genes are said to be *expressed* if a certain amount of corresponding proteins are synthesized. *DNA microarray* and *DNA chip* technologies enabled observation of expression levels of several thousands of genes simultaneously. Precisely, the amount of mRNA (messenger RNA) corresponding to each gene is estimated by observing the amount of cDNA that is obtained from mRNA via reverse transcription. Since proteins are synthesized from mRNA, the amount of mRNA estimated via DNA microarray or DNA chip is considered to approximately indicate the expression level of the gene. Analysis of gene expression patterns and time-series data of gene expression patterns has recently become an important topic in bioinformatics. Although various problems have been considered, this section focuses on the three important problems of *clustering of gene expression patterns, classification of tumor types* using gene expression patterns, and *inference of genetic regulatory networks*.

Clustering of Gene Expression Patterns

This problem is important for classification and prediction of functions of genes because it is expected that genes with similar functions have similar gene expression patterns (35,36). Suppose that we have a vector of gene expression levels $(g_i(1), g_i(2), \ldots, g_i(t))$ for each gene, where $g_i(j)$ denotes the gene expression level (real number) of the ith gene under the jth environmental constraint or at the jth time step. We would like to divide a set of several thousands genes into several or several tens of clusters according to similarities of vectors of gene expression levels (see Fig. 7). Clustering of real vectors is a well-studied topic in artificial intelligence and statistics, and many methods have been proposed. Various clustering methods have been applied to clustering of gene expression patterns, which include hierarchical clustering, self-organizing maps, k-means clustering, and EM-clustering (35–37).

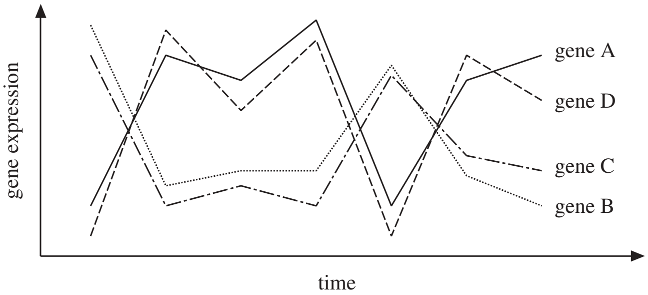


Figure 7. Clustering of gene expression patterns. In this case, genes are clustered into two groups: {A,D} and {B,C}.

Classification of Tumor Types

This problem may be the most important because it has many potential applications in medical and pharmaceutical sciences. Suppose that we have expression patterns for samples of tumor cells from patients and we would like to classify samples into more detailed tumor classes. Golub et al. considered two problems: *class discovery* and *class prediction* (38). Class discovery defines previously unrecognized tumor subtypes, whereas class prediction assigns particular tumor samples to predefined classes.

Golub et al. applied the *self-organizing map* (a kind of clustering method) to class discovery. In this case, they considered a vector $\mathbf{g}_j = (g_1^j, g_2^j, \ldots, g_m^j)$ for each patient, where g_i^j denotes the gene expression level of the ith gene of the sample obtained from the jth patient. They classified the set of samples into a few classes based on similarities of vectors. They also employed *weighted voting* for class predictions, where the weight for each gene was learned from training samples and each test sample was classified according to the sum of the weights.

In their experiments, not all genes were used for weighted votes, but only several tens of genes relevant to class distinction were selected and used. Use of selected genes seems better for several reasons. For example, cost for measurement of gene expression levels will be much lower if only selected genes are used. Golub et al. called these selected genes *informative genes*. Although they used a simple method to select genes, many methods have been proposed for selecting informative genes. Using terminologies in artificial intelligence, class discovery, class prediction, and selection of informative genes correspond to clustering, learning of discrimination rules, and feature selection, respectively. Many methods have been developed for these three problems in artificial intelligence (37,39–41).

Although it is still unclear which method is the best for tumor classification, the SVMs explained here have been effectively applied to class prediction (40,41). For example, consider the case of predicting whether a given sample belongs to a particular tumor class. We regard gene expression profile $\mathbf{g}_j$ corresponding to the jth sample as an example (i.e., a point in m-dimensional Euclidean space), where $\mathbf{g}_j$ is regarded as a positive example if the sample belongs to the tumor class, and as a negative example otherwise. Then we can simply apply an SVM to this problem, where many variants and extensions, which include multiple tumor class prediction, have been proposed (40,41). SVMs can also be applied to selection of informative genes in combination with *recursive feature elimination*(39,42). In this method, genes are ranked based on the weight (effect on classification) of each gene, and the gene with the smallest rank is recursively removed, where SVM-learning is executed at each recursive step.

Inference of Genetic Regulatory Networks

In order to understand the detailed mechanism of organisms, it is important to know which genes are expressed, when they are expressed, and to what extent. Expressions of genes are regulated through genetic regulatory systems structured by networks of interactions among DNA, RNA, proteins, and chemical compounds. Gene expression data are expected to be useful for revealing these genetic regulatory networks. Therefore, many studies have been done in order to infer the architectures of genetic regulatory networks from gene expression data. Usually, mathematical models of networks are required to infer genetic regulatory networks. Extensive studies have been done using such models as Boolean networks, Bayesian networks, and differential equations (4,43–46).

Here we briefly describe the Boolean network model and its relation with the Bayesian network model. The Boolean network is a very simple model (47). Each gene corresponds to a node in a network. Each node takes either 0 (inactive) or 1 (active), and the states of nodes change synchronously according to regulation rules given as Boolean functions. In a Boolean network, the state of node v_i at time t is denoted by $v_i(t)$, where $v_i(t)$ takes either 0 or 1. A node v_i has k_i incoming nodes $v_{i1}, \ldots, v_{ik_i}$, and the state of v_i at time $t + 1$ is determined by

$$v_i(t+1) = f_i(v_{i_1}(t), \ldots, v_{i_{k_i}}(t))$$

where f_i is a Boolean function with k_i input variables. This rule means that gene v_i is controlled by genes $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$.

For example, consider a very simple network in which there three nodes exist (i.e., genes) v_1, v_2, v_3, and the regulation rules are given as follows:

$$\begin{aligned} v_1(t+1) &= v_2(t), \\ v_2(t+1) &= v_1(t) \wedge v_3(t), \\ v_3(t+1) &= \overline{v_1(t)}, \end{aligned}$$

where $x \wedge y$ means the conjunction (logical AND) of x and y, and $\bar{x}$ means the negation (logical NOT) of x. Suppose that the states of genes at time 0 are $(v_1(0), v_2(0), v_3(0)) = (1, 1, 1)$. Then the states of genes change as follows:

$$(1,1,1) \Rightarrow (1,1,0) \Rightarrow (1,0,0) \Rightarrow (0,0,0) \Rightarrow (0,0,1) \Rightarrow (0,0,1) \Rightarrow \cdots$$

This sequence of state transitions corresponds to time-series data of gene expression patterns.

Under this model, inference of a gene regulatory network is defined as a problem of inferring regulation rules (i.e., input genes and Boolean functions) for all genes from a set of state transition sequences (43). Although gene regulation rules are deterministic in Boolean networks, the Boolean network model was extended to the probabilistic Boolean network model (48), in which multiple Boolean functions can be assigned to one gene and one Boolean function is randomly selected for each gene at each time step according to some probability distribution. Probabilistic Boolean networks are almost equivalent to dynamic Bayesian networks with a binary domain (49). In practice, Bayesian networks have been more widely applied to inference of genetic networks than Boolean networks since Bayesian networks are considered to be more flexible. Furthermore, many variants of Bayesian networks and their inference algorithms have been proposed for modeling and inference of genetic networks (43–46).

BIBLIOGRAPHY

1. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis. Probabilistic Models of Proteins and Nucleic Acids*, Cambridge, UK: Cambridge University Press, 1998.
2. N. C. Jones and P. A. Pevzner, *An Introduction to Bioinformatics Algorithms*, Cambridge, MA: The MIT Press, 2004.
3. D. W. Mount, *Bioinformatics: Sequence and Genome Analysis*, Cdd Spring Harbor, NY: Cold Spring Harbor Laboratory Press, 2001.
4. Aluru, S. (ed.), *Handbook of Computational Molecular Biology*, Boca Raton, FL: CRC Press, 2006.
5. D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler, GenBank, *Nucleic Acids Res.*, **34**: D16–D20, 2006.
6. L. Wang and T. Jiang, On the complexity of multiple sequence alignment, *J. Computat. Biol.*, **1**: 337–348, 1994.
7. C. Notredame, Recent progresses in multiple sequence alignment: A survey, *Pharmacogenomics*, **3**: 131–144, 2002.
8. J. Thompson, D. Higgins, and T. Gibson, CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting position-specific gap penalties and weight matrix choice, *Nucl. Acids Res.*, **22**: 4673–4390, 1994.
9. A. Henikoff and J. G. Henikoff, Amino acid substitution matrices from protein blocks, *Proc. Natl. Acad. Sci. USA*, **89**: 10915–10919, 1992.
10. M. Kann, B. Qian, and R. A. Goldstein, Optimization of a new score function for the detection of remote homologs, *Proteins: Struc. Funct. Genetics*, **41**: 498–503, 2000.
11. A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert, Approaches to the automatic discovery of patterns in biosequences, *J. Computat. Biol.*, **5**: 279–305, 1998.

12. C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment, *Science*, **262**: 208–214, 1993.
13. T. Akutsu, H. Arimura, and S. Shimozono, On approximation algorithms for local multiple alignment, *Proc. 4th Int. Conf. Comput. Molec. Biol.*, 1–7, 2000.
14. T. L. Bailey and C. Elkan, Fitting a mixture model by expectation maximization to discover motifs in biopolymers, *Proc. Second International Conf. on Intelligent Systems for Molecular Biology*, 28–36, 1994.
15. A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler, Hidden Markov models in computational biology. Applications to protein modeling, *J. Molec. Biol.*, **235**: 1501–1531, 1994.
16. K. Asai, S. Hayamizu, and K. Handa, Prediction of protein secondary structure by the hidden Markov model, *Compu. Applicat. Biosci.*, **9**: 141–146, 1993.
17. M. Levitt, M. Gernstein, E. Huang, S. Subbiah, and J. Tsai, Protein folding: The endgame, *Ann. Rev. Biochem.*, **66**: 549–579, 1997.
18. J. Moult, K. Fidelis, B. Rost, T. Hubbard, and A. Tramontano, Critical assessment of methods of protein structure prediction (CASP) - Round 6, *Proteins: Struc. Funct. Genet.*, **61**(S7): 3–7, 2005.
19. P. Bradley, S. Chivian, J. Meiler, K. M. Misuras, A. Rohl, and W. R. Schief, et al., Rosetta predictions in CASP5: Successes, failures, and prospects for complete automation, *Proteins: Struc. Funct. Genet.*, **53**: 457–468, 2003.
20. G. Pollastri, D. Przybylski, B. Rost, and P. Baldi, Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles, *Proteins: Struct. Funct. Genet.*, **47**: 228–235, 2002.
21. J. Xu, M. Li, D. Kim, and Y. Xu, RUPTOR: Optimal protein threading by linear programming, *Journal of Bioinformatics and Computational Biology*, **1**: 95–117, 2003.
22. R. H. Lathrop, The protein threading problem with sequence amino acid interaction preferences is NP-complete, *Protein Engin.*, **7**: 1059–1068, 1994.
23. R. H. Lathrop and T. F. Smith, Global optimum protein threading with gapped alignment and empirical pair score functions, *J. Molec. Biol.*, **255**: 641–665, 1996.
24. C. Cortes and V. Vapnik, Support vector networks, *Mach. Learning*, **20**: 273–297, 1995.
25. J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge, UK: Cambridge Univ. Press, 2004.
26. B. Schölkopf, K. Tsuda, and J.-P. Vert, (eds.), *Kernel Methods in Computational Biology*, Cambridge, MA: The MIT Press, 2004.
27. Y. Yamanishi, J.-P. Vert, A. Nakaya, and M. Kanehisa, Extraction of correlated gene clusters from multiple genomic data by generalized kernel canonical correlation analysis, *Bioinformatics*, **19**: i323–i330, 2003.
28. C. Leslie, E. Eskin, and W. E. Noble, The spectrum kernel: A string kernel for svm protein classification, *Proc. Pacific Symp. Biocomput. 2002*, **7**: 564–575, 2002.
29. C. Leslie, E. Eskin, J. Wetson, and W. E. Noble, Mismatch string kernels for svm protein classification, *Advances in Neural Information Processing Systems 15*. Cambridge, MA: The MIT Press, 2003.
30. A. Ben-Hur and D. Brutlag, Remote homology detection: A motif based approach, *Bioinformatics*, **19**: i26–i33, 2003.
31. T. Jaakola, M. Diekhans, and D. Haussler, A discriminative framework for detecting remote protein homologies, *J. Computat. Biol.*, **7**: 95–114, 2000.
32. H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu, Protein homology detection using string alignment kernels, *Bioinformatics*, **20**: 1682–1689, 2004.
33. K. Tsuda, T. Kin, and K. Asai, Marginalized kernels for biological sequences, *Bioinformatics*, **18**: S268–S275, 2002.
34. H. Kashima, K. Tsuda, and A. Inokuchi, Marginalized kernels between labeled graphs, *Proc. 20th Int. Conf. Machine Learning*, Menlo Park, CA: AAAI Press, 2003, pp. 321–328.
35. M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein, Cluster analysis and display of genome-wide expression patterns, *Proc. Natl. Acad. Sci. USA*, **95**: 14863–14868, 1998.
36. K. Y. Yeung, C. Fraley, A. Murua, A. E. Raftery, and W. L. Ruzzo, Model-based clustering and data transformations for gene expression data, *Bioinformatics*, **17**: 977–987, 2001.
37. A. Thalamuthu, I. Mukhopadhyay, X. Zheng, and G. C. Tseng, Evaluation and comparison of gene clustering methods in microarray analysis *Bioinformatics*, **19**: 2405–2412, 2006.
38. T. R. Golub, S. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeck, and J. P. Mesirov, et al., Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring, *Science*, **286**: 531–537. 1999.
39. F. Li and Y. Yang, Analysis of recursive gene selection approaches from microarray data, *Bioinformatics*, **21**: 3741–3747, 2005.
40. G. Natsoulis, *et al.*, Classification of a large microarray data set: Algorithm comparison and analysis of drug signatures, *Genome Res.*, **15**: 724–736, 2005.
41. A. Statnikov, C. F. Aliferis, I. Tsamardinos, D. Hardin, and S. Levy, A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis, *Bioinformatics*, **21**: 631–643, 2005.
42. I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, Gene selection for cancer classification using support vector machines, *Mach. Learning*, **46**: 389–422, 2002.
43. T. Akutsu, S. Miyano, and S. Kuhara, Inferring qualitative relations in genetic networks and metabolic pathways, *Bioinformatics*, **16**: 727–734, 2000.
44. H. deJong, Modeling and simulation of genetic regulatory systems: a literature review, *J. Computat. Biol.*, **9**: 67–103, 2002.
45. N. Friedman, M. Linial, I. Nachman, and D. Pe'er, Using Bayesian networks to analyze expression data, *J. Computat. Biol.*, **7**: 601–620, 2000.
46. S. Kim, S. Imoto, and S. Miyano, Inferring gene networks from time series microarray data using dynamic Bayesian networks, *Brief. Bioinformat.*, **4**: 228–235, 2003.
47. S. A. Kauffman, *The Origins of Order: Self-organization and Selection in Evolution*, Oxford, UK: Oxford Univ. Press, 1993.
48. I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks, *Bioinformatics*, **18**: 261–274, 2002.
49. H. Lähdesmäki, S. Hautaniemi, I. Shmulevich, and O. Yli-Harja, Relationships between Probabilistic Boolean networks and dynamic Bayesian networks as models of gene regulatory networks, *Signal Process.*, **86**: 814–834, 2006.

TATSUYA AKUTSU
Kyoto University
Kyoto, Japan

BIOLOGICALLY INSPIRED NEURAL COMPUTATION

THE BIOLOGICAL NEURON

In order to fully understand the relationship of artificial neural network components to their biological analogues, an overview of the biological neuron and its function within networks of interconnected neurons is presented here. In particular, the electrical behavior of the neuron membrane, the influence of synaptic junctions, and the computational aspects of simple neural networks are presented.

It should be emphasized that there is no mathematical model that exactly describes the behavior of the biological neuron. Instead, there is a plethora of models and algorithms, some of which are closer to the true biological behavior than others, as verified by experimentation. Moreover, as a general rule, those models that are more true to neuron physiology, henceforth called *biological*, have more computationally demanding solutions than those that are less physiologically accurate. Hence, the more simplified models, including the perceptron and its derivative models, will henceforth be called *computational*.

Neurons

Nerve cells (or neurons) are fundamental components of the human nervous system. They relay and process information that governs our movement and perception. Moreover, the number of neurons in the human brain itself is estimated to be on the order of 10^{11}(1). Recently, glial cells, which provide physical support to neurons in the brain and outnumber the neurons by a ratio of 10:1, have also been reported to perform some computational tasks through chemical signaling (2). However, their influence shall be omitted here for practical purposes.

The physical dimensions of neurons vary significantly depending on the location and function of the particular neuron [see Fig. 1(3)]. However, all neurons are encapsulated by a thin membrane, in the order of 50 nm in thickness separating an inner axoplasm from the outer environment. Also, all neurons can be thought of as being comprised of three components: the soma, or cell body of the neuron, typically spheroidal in shape; the dendrites, or thin extensions of the cell body that receive input stimuli; and the axon, a larger and longer extension of the cell body that transports information to other neurons in the form of electrical pulses. Moreover, typical neurons in the human brain have a soma from 4 μm to 100 μm in diameter and an axon from several millimeters to over a meter in length, as in the case of some motor neurons that extend down the spine.

The excitable nerve cells differ from other cells in that they communicate with each other through a distinct use of electrical and chemical signaling. That is, local changes in electric potential and current density of the cell membrane propagate along the neuronal cell conveying information in an *intra*cellular manner, whereas chemicals called neurotransmitters are emitted from one cell and affect the electrical behavior of adjacent cells, thus conveying information in an *inter*cellular manner.

Given the unique electrical and chemical behavior of neurons, it is possible to envision how groups of neurons are capable of executing computational tasks. In fact, it is well established that neurons are responsible for the detection and interpretation of exogenous stimuli (through sensation), as well as the regulation of muscle contractions (or movement). For example, in the human retina, sensory stimuli (light waves) are converted to electrical impulses by photo-receptor cells. Then, networks of neurons process the exogenous information through nonlinear methods such as lateral inhibition (4). Also, impulses originating from motor neurons in the basal ganglia can activate muscle cell contractions, thus governing movement in humans and nonhuman primates (4).

Electrical Behavior

The electrical behavior of neurons is due to the concentration gradient and movement of specific ions across the cell membrane. In particular, the electric potential across the membrane (or transmembrane potential V_m) changes in proportion to the presence of positive ions within the cell. Also, as ions are transported across the membrane, they effectively establish a transmembrane current I_m. Moreover, at equilibrium conditions, $V_m \approx -70\,\text{mV}$ and $I_m \approx 0$.

The forces that transport the ions across the cell membrane are due to diffusion, permeability, and a biological mechanism of active transport. In particular, a biological "ion pump" actively transports ions across the membrane in order to maintain a particular concentration gradient for each ion. Also, when the permeability of the membrane to a particular ion changes, the diffusion forces establish a transmembrane current for that ion. Moreover, when V_m is below some threshold V_{th}, the relation between V_m and I_m is practically linear with constant permeability, whereas when V_m approaches V_{th}, ion permeability begins to change with respect to time, thus defining a time-varying and nonlinear relationship between V_m and I_m that lasts for a finite duration.

The nonlinear behavior of the neuron consists of a chain reaction of shifts in permeability that governs the influx and exit of charged ions across the cell membrane over time. That is, first the sodium (Na^+) permeability increases for a brief instant (roughly 1 ms in duration) causing Na^+ ions to diffuse into the cell from the higher concentration of Na^+ outside the cell, thus raising V_m. Next, the potassium (K^+) permeability immediately increases followed by a mass exit of K^+ ions from the higher K^+ concentration inside the cell, which, in turn, pulls V_m down to a hyperpolarized state V_{hyp} until the K^+ permeability returns to equilibrium. At the same time, the ion pump mechanism is constantly restoring the concentrations of Na^+ and K^+ to the original levels.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

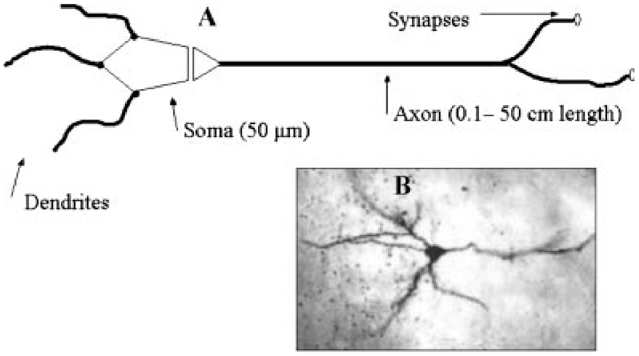


Figure 1. Graphic illustration of a neuron (A). Image of a pyramidal neuron from the cerebral cortex (B) from Ref. 3.

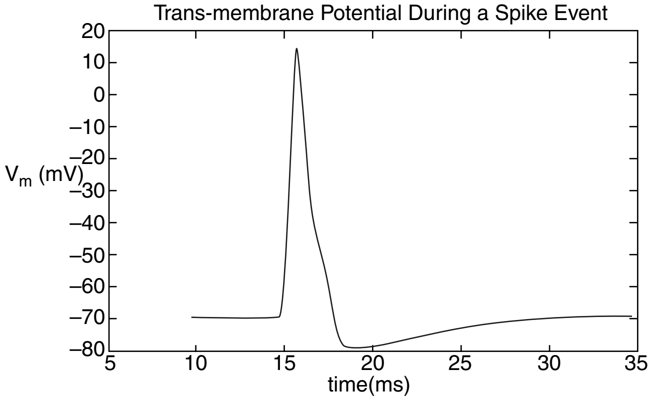


Figure 3. A typical action potential waveform.

Figure 2 illustrates the behavior of the cell membrane when a Na$^+$ channel is open while a K$^+$ channel is closed. Also, relative concentrations of Na$^+$ and K$^+$ are shown.

The course that V_m follows throughout the duration of the nonlinear behavior is characterized by three phases. Namely, these phases are the depolarization, repolarization, and hyperpolarization. In particular, during depolarization, V_m suddenly increases from its equilibrium to roughly 20 mV due to the influx of Na$^+$. Then, the membrane immediately enters the repolarization phase, where V_m suddenly decreases due to the exit of K$^+$, which is then followed by hyperpolarization, where V_m swings below −70 mV to roughly −80 mV. Together, the three phases comprise an "action potential" waveform or "spike" illustrated in Fig. 3. Moreover, the action potential duration is roughly 2 ms for the depolarization and repolarization phases, whereas the hyperpolarization phase lasts somewhat longer. Furthermore, due to the closed Na$^+$ channels immediately after depolarization, it is virtually impossible to elicit a new action potential event in the neuron. This effect is called inactivation and its duration is the *absolute refractory period*. However, during the period of hyperpolarization, it is possible, although somewhat difficult, to achieve threshold as compared with rest conditions. This process is called de-inactivation and the duration is the *relative refractory period* (1).

It is believed that the action potential is generated at the point where the axon meets the neuron soma, a region dubbed "axon hillock." Subsequently, ion currents entering the cell force adjacent locations of the cell membrane to depolarize, thus propagating the action potential across the cell axon at roughly 1 m/s (10 m/s in myelinated cells), depending on the dimensions of the cell.

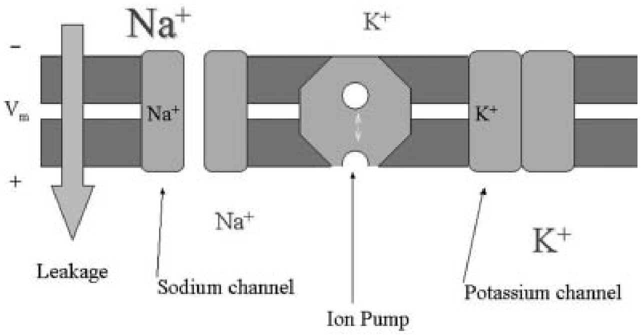


Figure 2. Ion permeability in a neuron membrane.

For a more rigorous mathematical description of the nerve cell dynamics, the interested reader may study the "core conductor" model, which pairs the Hodgkin–Huxley equations with the "transmission line" or "cable equations" (4,5).

Synaptic Transmission

The synaptic junction is the computational link between adjacent neurons. It regulates communication between adjacent neurons through short-term and long-term processes. The long-term processes are believed to regulate learning ability, whereas short-term processes are involved in the transmission and processing of real-time information.

In the short-term, communication at the synapse is achieved through the release of neurotransmitters by one neuron, and the effect on ion permeability that those neurotransmitters have in the membrane of adjacent neurons. In particular, when the action potential waveform travels along the length of the axon, then branches into a dendritic branch, it finally terminates at a synaptic junction where biochemical processes not yet well understood cause the release or exocytosis of neurotransmitters into a space between the presynaptic neuron and postsynaptic neuron. Next, as the neurotransmitters come into contact with the postsynaptic membrane, they alter the ionic permeability of that membrane and cause an inflow or outflow of current, depending on the type of neurotransmitter and the particular ion channels affected on the postsynaptic neuron. Moreover, depending on whether the resulting net membrane current is inward or outward, the synaptic connection is said to be *excitatory* or *inhibitory*.

From a systems perspective, the synapse acts as an integrator because of the first-order response of induced local postsynaptic current upon arrival of a presynaptic action potential. Moreover, this result is likely because of the slowly decaying amount of neurotransmitter released in response to the arrival of the action potential. In particular, the impulse response of the system can be modeled as

$$I_m(t) = \frac{J}{\tau_s} e^{-(t-t_k)/\tau_s} u(t - t_k) \tag{1}$$

where $I_m(t)$ represents the local current induced by the synapse over time, J represents the strength or efficacy of the synapse, τ_s represents the decay constant, t_k is the time of the incident spike, and $u(t)$ represents the unit step function (6). Furthermore, notice how Equation (1) omits the effect of action potential intensity and instead focuses only on the time of incidence t_k of the incoming spike.

Random processes within the presynaptic membrane cause spontaneous release of neurotransmitter even in the absence of an incident spike. Moreover, Fatt and Katz have shown that neurotransmitter is actually released in bundles where the number of bundles follows a Poisson process. Thus, the cumulative effect of this release over many synapses is to cause occasional random spike generation in the postsynaptic membrane.

FROM BIOLOGY TO COMPUTER SCIENCE

Simplifications of the biological model of the neuron can produce the early "perceptron" models and threshold units that influenced the vast proliferation of artificial neural networks in the latter part of the twentieth century. In particular, a linear relationship can be established between the summation of synaptic activity in a neuron and the frequency of presynaptic spike trains. Also, a distinct nonlinear function can be established between the resulting spike frequency of a particular neuron and the summation of its synaptic contributions.

From Synapse to Summation

As shown previously, synapses integrate the spikes and collectively cause an aggregation of positive charge in the cell body of the neuron. This temporal and spatial integration is known as "summation" in neuroscience literature. There are also inhibitory synapses that retain the temporal integration, but elicit an *exit* of positive charge from the postsynaptic membrane, thus contributing a negative component to the overall summation.

The temporal integration is described by Equation (1) above. Moreover, solving Equation (1) with respect to local postsynaptic transmembrane current I_l for some synapse l during the arrival of a presynaptic spike train of frequency f shows a near-linear relationship between I_l and f. In particular, assuming the decay rate τ_s of Equation (1) is large compared with the duration of an action potential, the response of I_l with respect to the presynaptic transmembrane potential is just an impulse response

$$h(t) = \begin{Bmatrix} \frac{J}{\tau_s} e^{-\frac{t}{\tau_s}} & \text{for } t \geq 0 \\ 0 & \text{otherwise} \end{Bmatrix} \tag{2}$$

Next, assuming there is an arrival of a spike train of frequency f, or an interarrival time of $\Delta t = 1/f$, also assuming the spike train has a long duration (ignoring transients), the input can be represented as

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\Delta t) \tag{3}$$

Thus, in a linear system sense, the local postsynaptic current that will result is given by the convolution equation

$$I_l(t) = \int_{-\infty}^{+\infty} h(T - t)s(T)dT \tag{4}$$

Furthermore, solving for $I_l(t)$ yields

$$I_l(t) = \frac{J}{\tau_s} \frac{e^{\frac{1}{\tau_s}(t - \lceil \frac{t}{\Delta t} \rceil \Delta t)}}{1 - e^{-\frac{\Delta t}{\tau_s}}} \tag{5}$$

As can be gleaned from Equation (5), the dependence of I_l on time t is constrained within limits that depend on τ_s. In particular, the range of I_l can be described as

$$\frac{J}{\tau_s\left(e^{+\frac{\Delta t}{\tau_s}} - 1\right)} < I_l < \frac{J}{\tau_s\left(1 - e^{-\frac{\Delta t}{\tau_s}}\right)} \tag{6}$$

Furthermore, taking the limit as $\tau_s \to \infty$ yields the remarkable result that

$$I_l = \frac{J}{\Delta t} = Jf \tag{7}$$

Which can be thought of as a neural rate-coding theorem in that it shows a linear relationship between the incident spike frequency at a synapse and the resulting induced local current in the postsynaptic neuron membrane. Moreover, it is noteworthy that the synaptic efficacy J retains the same value and meaning in the transition from the biological model in Ref. 6 to the computational realm of the perceptron. Also, it is evident that as τ_s becomes larger, the dependency of I_l on f becomes stronger than its dependency on t. Thus, the temporal variations of the incident spike train become less significant with larger τ_s as Fig. 4 illustrates.

In turn, the "spatial" integration that is achieved by a neuron is simply the aggregate effect of all the synapses connected to the neuron. In mathematical terms, using the linear relation shown in Equation (7), the total transmembrane current due to synaptic connections can be described as

$$I_s = \sum_{l=1}^{L} J_l f_l \tag{8}$$

where J_l and f_l represent the efficacy and incident spike frequency of a particular synapse l.

The results of this section suggest that the mechanism of summation used in popular models of artificial neural networks is closely related to the more biological model of the synapse, as has been alluded to often in the scientific literature. That is, the overall effect of synaptic connections

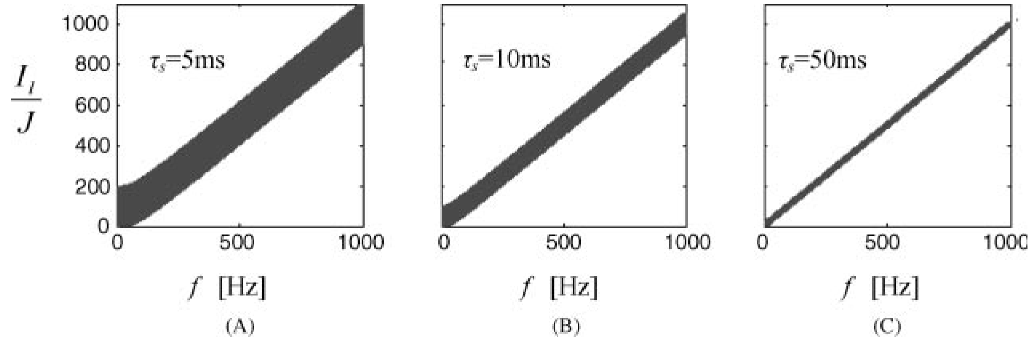


Figure 4. The shaded regions show the possible range of local current at a given frequency of an incident spike train for $\tau_s = 5\,\mathrm{ms}$ (A), $\tau_s = 10\,\mathrm{ms}$ (B), $\tau_s = 50\,\mathrm{ms}$ (C). Moreover, the values of current are normalized with respect to J. As can be seen, the relationship between postsynaptic local current I_l and incident spike frequency f is approximately linear with an uncertainty that grows inversely proportional to the synaptic decay rate τ_s.

on a neuron can be described by a linear relationship—the weighted sum of the incident spiking frequencies, where each weight represents the efficacy of a particular synapse. Also, the decay rate of a synapse affects the accuracy or fuzziness of the relationship in that relatively larger decay rates constrain the time-varying aspects of the effect. Moreover, practical ranges for τ_s that were used in this study were 5 ms, 10 ms, and 50 ms, similar to the synaptic types found in Ref. 6.

From Voltage-Gated Channels to Activation Functions

As described previously, the voltage-gated behavior of the neuron membrane establishes a threshold between the linear and nonlinear modes of operation. In particular, when the transmembrane voltage V_m is sufficiently less than the threshold V_{th}, the relationship between V_m and the transmembrane current I_m is practically a linear one. However, when V_m reaches V_{th}, a nonlinear event known as the action potential is generated.

The biological model describes the dynamics of the neuron behavior with respect to time. However, what is the relationship between the biological model and the computational model of the perceptron? In other words, how can the perceptron model be derived from the biological analogue?

The analysis can begin by describing the subthreshold dynamics of the neuron membrane. In particular, given nominal membrane resistance R_m and capacitance C_m, the linear model relating V_m to the total transmembrane current I_m is

$$R_m C_m \frac{dV_m}{dt} + V_m - R_m I_m = 0 \tag{9}$$

For the case when I_m consists of a step function with amplitude $\bar{I}_m$, and V_{rest} is V_m at $t = 0$, Equation (9) can be solved for V_m yielding the result

$$V_m(t) = V_{rest} + \bar{I}_m R_m \left(1 - e^{-\frac{t}{R_m C_m}}\right) \tag{10}$$

At this point, a change of variables can make the derivations simpler. In particular, introducing the relative transmembrane voltage $\Delta V_m = V_m - V_{rest}$ into Equation (1) yields

$$\Delta V_m = \bar{I}_m R_m \left(1 - e^{-\frac{t}{R_m C_m}}\right) \tag{11}$$

Now, introducing the relative threshold $\Delta V_{th} = V_{th} - V_{rest}$, and replacing t with Δt, the time required to reach threshold is

$$\Delta t = -R_m C_m \ln\left(1 - \frac{\Delta V_{th}}{\bar{I}_m R_m}\right) \tag{12}$$

It is apparent that Δt will be finite for only certain values of $\bar{I}_m$. In particular, the minimum value of $\bar{I}_m$ required to achieve threshold (called the rheobase current) (5) is

$$I_{rh} = \frac{\Delta V_{th}}{R_m} \tag{13}$$

The implications for the spike frequency $f = \Delta t^{-1}$ are that f remains essentially zero until I_s (the summation current) surpasses the rheobase current. Furthermore, this implication confirms the older perceptron models with the "hard-limiting" function, or the activation function with a discontinuity. However, what are the exact values of f as I_s extends past the rheobase current?

To answer this question, Equation (12) may seem like the likely candidate. However, this equation predicts that f will grow unboundedly with I_s, a relation that is not, in fact, practical. In contrast, a more realistic scenario includes the refractory period t_{ref} of the neuron. The reason is that t_{ref} plays a significant role in bounding the upper limits of spike frequency f that are attainable by a neuron.

As stated previously, t_{ref} is caused by the inactivation of ion channels in the cell membrane and the prolonged opening of the K+ channels. Specifically, when V_m swings to V_{hyp} during hyperpolarization, any activating current has to be strong enough to drive V_m from V_{hyp} to V_{th}, which is a greater leap than driving V_m from V_{rest} to V_{th}(1).

For all practical purposes, this activity would mean that ΔV_{th} is nearly infinite during the inactivation phase of the sodium channels, and then decays exponentially from

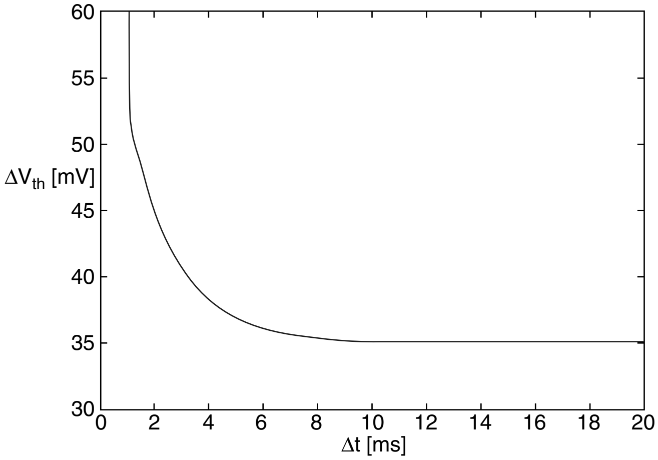


Figure 5. The relative threshold ΔV_{th} as a function of the time elapsed Δt since depolarization for $\alpha = 575.6$, $v_\Theta = 35\,\text{mV}$, $V_o = 66.6\,\text{mV}$, $T_{abs} = 1\,\text{ms}$, and $A = 3.2 \times 10^{-16}$.

ΔV_{hyp} to its nominal value as the potassium channels close. Moreover, this kind of description is closely in keeping with the models of threshold voltage mentioned in Ref. 4. In particular, assuming the inactivation phase of the sodium channels ends at time T_{abs}, a very steep function of Δt could model ΔV_{th} when $t < T_{abs}$. Then, for $t > T_{abs}$, the relation could be a decaying exponential. Thus, if v_Θ is ΔV_{th} at rest and A is a constant chosen to keep the curve continuous, then

$$\Delta V_{th}(\Delta t) = \left\{ \begin{array}{ll} v_\Theta + \dfrac{A}{\Delta t^{10}}, & 0 \leq \Delta t < T_{abs} \\ v_\Theta + (v_0 - v_\Theta)e^{-\alpha \Delta t}, & T_{abs} < \Delta t \\ v_\Theta, & \text{otherwise} \end{array} \right\} \tag{14}$$

Figure 5 shows a graphic representation of Equation (14) for appropriate values of v_Θ, T_{abs}, and α.

Substituting Equation (14) into Equation (12) yields a relation that cannot easily be solved for Δt. However, numerical methods can be used to obtain a consistent answer. Thus, using the ALOPEX algorithm (15), the end result is a series of points describing the relation of Δt to I_s. Furthermore, inverting this result yields a relation of f to I_s (shown in Fig. 6) that is in keeping with the "activation function" of the perceptron.

The activation function is a central theme of the perceptron and most of the derivative artificial neuron models in that it limits the possible output levels of a neuron. In this sense, it is the defining factor that places artificial neurons and neural networks into the category of nonlinear adaptive systems (8).

A popular approximation to the activation function of a neuron is the sigmoidal function. For example, using the sigmoidal function, the output of a neuron given the summation I_s is

$$F(I_s) = \frac{2}{\left(1 + e^{-\beta I_s}\right)} - 1 \tag{15}$$

Figure 6 shows the results of a numerical solution with a sigmoidal function optimized to fit the numerical solution.

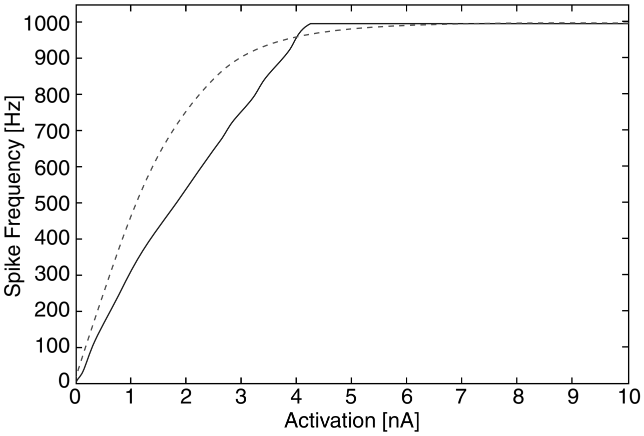


Figure 6. Activation function (solid) and sigmoid (--). Sigmoid parameter was $\beta = 0.99 \times 10^9$. Subthreshold neuron parameters were $R_m = 414\,\text{M}\Omega$ and $C_m = 78.5\,\text{pF}$ for a neuron of diameter 50 μm.

Rate-Coding in Neurons and Networks

The synapse can be thought of as a *decoder* of spike trains into levels of activity, whereas the neuron itself is viewed as a modulator or *encoder* of aggregate synaptic activity into a spike train. This aspect of neuron function has been called *rate-coding* because the information transmitted by a particular neuron is thought to be carried by the firing rate itself. Also, pulse frequency modulation (PFM) has been used to characterize this model (4). Accordingly, Fig. 7 shows the decoding, summation, and spike generation aspects of a neuron where $H_e(S)$ and $H_i(S)$ are transfer functions of excitatory and inhibitory synapses, respectively, and the component labeled "H-H" denotes a Hodgkin–Huxley-type spike generation mechanism.

From Synaptic Plasticity to Learning Algorithms

The cornerstone of the theory on synaptic plasticity is the "Neurophysiological Postulate," made by Donald Hebb in 1949 (9,10), which states

> When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased.

A simple mathematical description of this postulate is suggested by Haykin et al. (11). In particular, given the

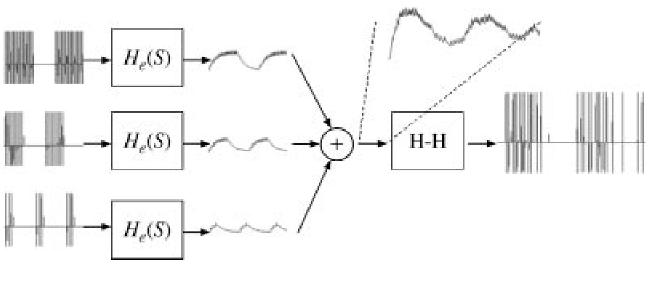


Figure 7. A neuron as a pulse frequency modulator.

activity in two adjacent neurons is $x_a(t)$ and $x_b(t)$ and a "learning rate" parameter is η, then the change in synaptic efficacy would be

$$\Delta\theta_{ab}(t) = x_a(t)x_b(t) \tag{16}$$

However, the value representing synaptic efficacy θ_i can potentially grow without bound. Thus, to curb this behavior, Oja introduced a learning rule that will guarantee that the norm of θ_is will have unity magnitude (12). In particular, given the output of a neuron is V and the input to synapse i is ξ_i, then

$$\Delta\theta_i = \eta V(\xi_i - V\theta_i) \tag{17}$$

Both the Hebbian and Oja rules are considered to be in the category of "unsupervised" learning in that a network of such neurons adjusts itself to encode or interpret the data at the input. However, when the synaptic weights of a network are modified to minimize some external cost function, the learning rules are in the category of "supervised" learning (13).

One such learning rule is the "perceptron learning rule," introduced by Rosenblatt (14). In this case, given the desired output ζ_k of neuron k and some threshold N, the weight of synapse i is updated as

$$\Delta\theta_{k,i} = \eta \mathrm{U}\left(N - \zeta_k \sum_j \theta_{k,j}\xi_{k,j}\right)\zeta_k\xi_{k,i} \tag{18}$$

where U is the unit step function.

A similar innovation on this theme of supervised learning is the ALOPEX algorithm that introduces stochastic components in the updating of synaptic weights in order to minimize a global cost function (15). In particular, given global error E_n at iteration n, zero-mean random variable $r_{i,n}$, and rate parameters γ and σ, the algorithm can be described in the following steps

$$\Delta E_n = E_n - E_{n-1} \tag{19}$$

$$\Delta\theta_{i,n+1} = \gamma\Delta E_n \Delta\theta_{i,n} + \sigma r_{i,n} \tag{20}$$

THE PERCEPTRON AND OTHER NEURAL NETWORKS

The derivations of synaptic summation and the neuron activation function outlined above show how the perceptron model, introduced by Rosenblatt (14), is very similar to the more biological descriptions of the neuron and its synapses. In particular, using the standard notation for perceptrons, the output of some neuron i with synaptic weights given by w_{ik}, inputs ξ_k, and threshold μ_i, is given by

$$O_i = g\left(\sum_k w_{ik}\xi_k - \mu_i\right) \tag{21}$$

Here, the activation function $g(\bullet)$ can be thought of as the activation function derived from Equations (12) and (14), whereas the argument of $g(\bullet)$ is the summation mentioned in Equation (8). Also, the threshold can be thought of as the rheobase current mentioned in Equation 13.

For a "hard-limiting" activation function, the summation can be thought of as being sufficiently large so that the transition of $g(\bullet)$ from zero to peak (~1000 Hz) is negligible. In this case, the model resembles the McCulloch–Pitts (16) unit, so that, for some neuron unit i at iteration n and threshold μ_i, the next output is given as

$$O_i(n+1) = U\left(\sum_j w_{ij}O_j(n) - \mu_i\right) \tag{22}$$

where U($\bullet$) is the unit step function.

In 1982, Hopfield (17) popularized another neuron model that was originally developed by Cragg and Temperley in 1954 and later modified by Caianiello in 1961. Moreover, it is similar to the McCulloch–Pitts model except that the step function is replaced by the signum function and the threshold is dropped from the equation. In this manner, the dynamics of a network of such neurons are described by equations that are familiar in statistical mechanics. In particular, the output of a single neuron i is

$$S_i = sgn\left(\sum_j w_{ij}S_j\right) \tag{23}$$

Networks composed of Hopfield units are known as "associative memories" because of their ability to associate an unknown input to some stored information. Also, the stored patterns are known as "attractors" because of the trajectory that an unknown input will follow until it reaches the correct association (12).

Three-Dimensional Neural Networks

A three-dimensional, time-dependent (though synchronized) artificial neural network (ANN) is presented with the purpose to better simulate a complex biological system and thereby be useful in the understanding of how memories are stored and recollected. In emulating the biological model, this ANN also functions to aid in visualizing how neural damage of various kinds affects mental activity.

The ANN is three-dimensional and time-dependent (yet synchronized). It differs from the classic pattern recognition system in four ways:

1. It is a three-dimensional network (3D-NN), rather than a one-dimensional neural network. In this sense, it consists of two-dimensional planes of neurons stacked atop one another in the third dimension. The connectivity of a network is determined by this 3D configuration, with neurons that are closer to one another having a greater chance of being connected to one another.
2. The network is dependent on time. It takes one unit of testing time for neurotransmitters to jump across any synapse. The exception to this role is the

connections going from the external stimuli to the receptor neurons, which transmit signals instantaneously; these connections do not represent a synaptic gap, but rather the detection of external stimuli.

3. The connections can act in lateral and feedback manners, as well as a feed-forward manner (which is the type presented most often in the classic ANN). Lateral and feedback connections are present within the biological system, and these here help to make the network more flexible and closer to the biological model (See Fig. 1).
4. The classic ANN has the purpose of pattern recognition. The 3D-NN can be trained in an unsupervised manner, however, via a version of Hebb's rule, where external stimuli may or may not be present, and the network generates particular sets of weights in response. Without the introduction of known templates, it tries to simulate learning templates for the first time, as a child does in real life. This creation of memories is then used in conjunction with a pattern recognition system when recollection of these memories is desired.

Damage of the network can be assessed by destroying neurons or connections within specific areas and then visualizing how the activity of the entire network is affected. Various conditions can be simulated. For example, a stroke or concussion is modeled by destroying neurons within a finite area on a certain layer. Parkinson's disease is simulated by destroying connections between specific layers (such as the destruction of inhibition in this condition that can lead to tremors). Dementia is simulated by damaging connection strengths with random noise.

This type of neural network allows analysis of these conditions through visualization of the activity of the network at various times and places. Through such visualization, the ANN attempts to aid in greater understanding of its biological counterpart. An example of such a network is given in Fig. 8, where we show the concept of the ANN, where predefined areas of the first layer (in physiological terms these areas are the so-called Receptive Fields), are connected to neurons in the second layer after their output has undergone some filtering. The outputs of the second-layer neurons connect to the third layer and so on. This network, however, differs from the classic perceptron network in two main points: each neuron can send an output to any neuron in the previous layers or the next layers and, in addition, it can make connections laterally (i.e., with neurons on the same plane); and information is transferred in quanta of time from one layer to the next.

Figure 9 shows this propagation of signals from layer to layer and for six consecutive times. The columns correspond to time intervals (1 through 6) (Fig. 9 a, b, c) and rows correspond to neuronal layers (1 though 3). The X, Y coordinates specify the position of a neuron on the plane, whereas the Z coordinate indicates the activity of the neuron.

The reader should notice that at $t = 1$ (Fig. 9a), only the first layer has activity, because this layer is where the stimulus is originating. The second and third layers do not have any activity. At $t = 2$ (Fig. 9a), the activity has also reached the second layer, whereas at $t = 3$ (Fig. 9a), all three layers show some activity. Because of the feedback connectivity as well as the lateral connections, the distribution of activity on each plane is not changing in a linear manner (Fig. 9 b, c).

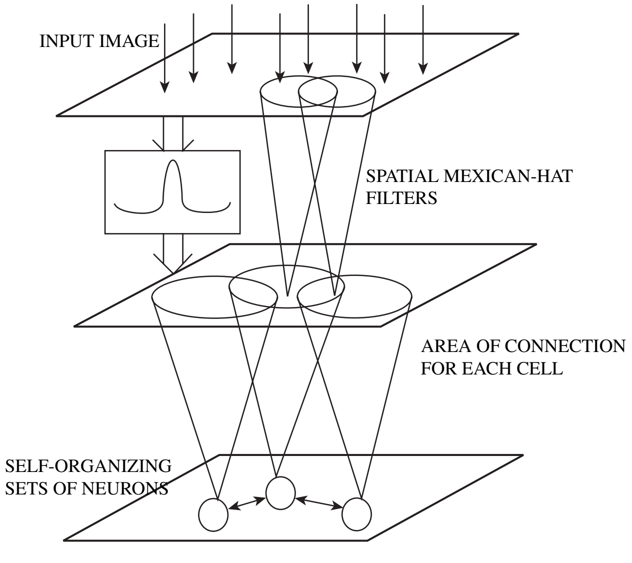

INPUT IMAGE
SPATIAL MEXICAN-HAT FILTERS
AREA OF CONNECTION FOR EACH CELL
SELF-ORGANIZING SETS OF NEURONS

Figure 8. Schematic representation of a 3D-neural network. Stimulus is applied on the first layer. Neurons can connect to any other neuron on any level, with specified connection strengths, gains, and timing characteristics such as delays.

The network, as in the classic ANNs, consists of an input (stimulus) layer, layers of neurons, and connections. The stimulus layer can contain as many different stimuli as desired. Each stimulus consists of a specified number of nodes, each of which shall have a specified activation. The neural layers all contain the same number of neurons, but the spatial locations of neurons on each plane are random and unique to that plane. The neurons themselves can have activation functions (AFs) of four different types. The linear AF takes the form $y_j = \tau \sum_i x_i w_{ij}$, where y_j is the output of neuron j, x_i is the output of neuron i at the beginning of a connection that terminates at neuron j, w_{ij} is the connection strength between these two neurons, and τ is the slope of the AF. The sigmoid is expressed as $y_j = \frac{2}{1+\exp\left[-\frac{\left(\sum_i x_i w_{ij}\right)}{\tau}\right]} - 1$; the multiplier of 2 and subtraction of 1 function to give the sigmoid bounds of -1 and 1, with an input of 0 giving 0 output; without these two factors, the bounds are 0 and 1, which is a reasonable scenario, but an input of 0 creates an output of 1/2, which then propagates through the network, creating output even though there is no initial stimulus, which is unacceptable. The biological neuron uses a threshold-type activation function, which the sigmoid mimics, except for the addition of a continuous first derivative. The linear AF is useful in making a network simple, as it is then easier to trace what one knows an output should be. In trying to simulate the biological system, however, the sigmoid is more accurate. Unfortunately, the exponential in the sigmoid can only

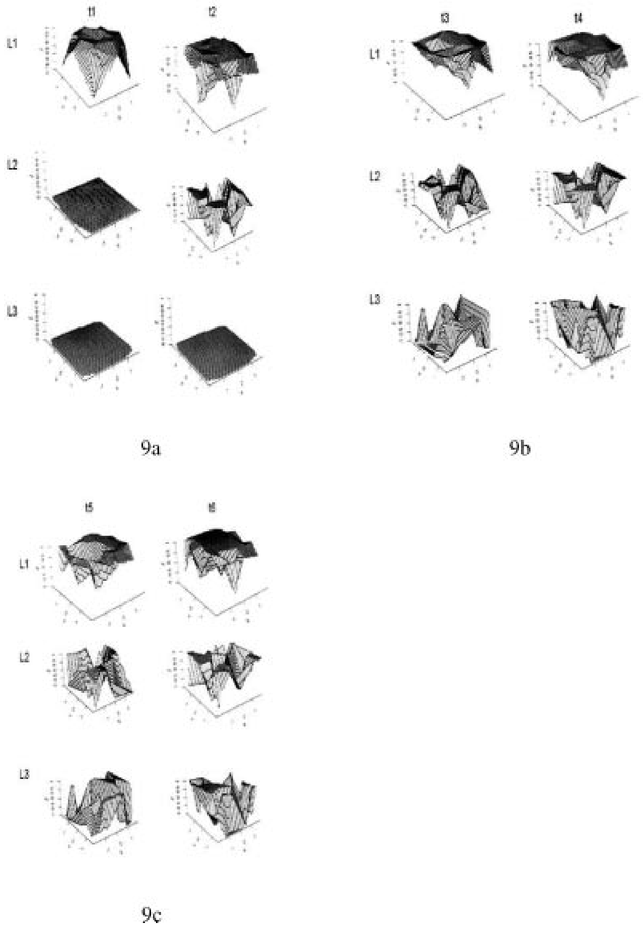


Figure 9. Progress of the workings of the 3D spatiotemporal neural network. Columns correspond to time slots. Rows indicate layers of neurons in a planar form, with given X, Y coordinates for neuronal positions on the plane. The Z axis represents the activity of neurons at that time. Notice that at $t = 1$, layers 2 and 3 have not received any inputs yet and therefore have no activity, At $t = 2$, layer 2 has some activity but layer 3 does not.

handle arguments up to a finite size, and too great an input will cause the network to fail; luckily, however, this problem usually only occurs when the stimuli are given activation that exceeds normal values, and it is easily controlled; because neural outputs are limited to a maximum of 1 (apart from the activation of input layer nodes), this overflow is only a problem when a neuron has too large a receptive field (greater than 1000 inputs, maybe), which then inundates this neuron with a greater input than the exponential function can handle. A problem with the linear AF manifests itself when feedback and lateral connections are introduced into a network; in this case, a signal, via these non-feed-forward connections, can grow indefinitely, and the output of a neuron possibly can, over time, approach infinity. A linear AF, however, can be given bounds of minimum and maximum activation. Bounding the linear AF is effective in preventing the unwanted blowup of activation seen in unbounded linear nets with feedback. A fourth type of AF is the step function, with which a neuron will fire with an output of 1 if the input exceeds a certain threshold, and it will remain inactive (output of 0) if the input is less than this threshold. The step function emulates the biological system in the closest fashion, but it does not give as much flexibility in neural outputs as the linear and sigmoid functions do, because the step function lacks the intermediate values otherwise possible.

Connections can stem from the input layer to any number of other layers, and also among all other layers. The only limitation on connections is that, with respect to the stimulus layer, only feed-forward connections may exist, which is biologically reasonable. Among all other layers, feed-forward, feedback, and lateral connections are viable. Connections are created on the basis of a connective neighborhood. The radius of this neighborhood may represent one of two things, either it is the boundary of the circle within which all neurons are connected to (by the neuron of origination) or it is the standard deviation of a Gaussian probability curve, with closer neurons having a greater probability of being connected. The initial weights of these connections are random, except for the synapses between the inputs and the first neural layer, which can either be initially random (and thereafter trained along with the other synapses) or initially (and thereafter) a constant 1.

Testing the network simply consists of exposing it to specified stimuli and allowing the network to run through a specified number of time units. The external stimuli can be either constant in value (with a delay for activation, if desired) or dynamic. The dynamic input is a sine curve, which varies the intensity of the nodes in a stimulus sinusoidally as a function of time. Testing the ANN will generate activation of neural layers.

Training the ANN can be done in one of two ways: unsupervised or supervised. The former uses predetermined stimuli, but it does not match these stimuli with known memories of any sort. A set of stimuli are presented to the network, and the outputs of all neurons are calculated. Then, the weights are updated according to a specified training rule. With the new synaptic strengths, the network is then run through again while being presented with the desired stimulus. The weights are again changed. This process continues iteratively for the desired number of iterations. It is seen that one iteration of training here is equal to one unit of network running time (as opposed to the case of supervised training, below, where one iteration of training usually consists of many time units of running through the network).

During unsupervised training, as during testing, the input layer can be either static or dynamic. Static inputs can be either homogeneous (each node in a stimulus taking the same value) or heterogeneous. Dynamic inputs can be sinusoidal (as in the testing case) or prespecified. Many training rules are available for the purpose of updating weights without supervision.

Supervised training requires stimuli-output pairs to be known, where the stimuli are the features of a specific template, and the ANN matches these features with the output neural layer corresponding to this same template. This layer must take a known form, after a specified amount of running time, for specific stimuli. The ALOPEX training algorithm then proceeds to iteratively alter the connection strengths of the network until the introduction of a certain set of stimuli creates the correct scenario of activation for the output layer. The activation of the output layer (to correspond to a specific set of features) can be set in one of two ways. One is to specify areas of output neurons to be active (positive) when a certain template is introduced, with other output neurons having negative activation. The other way for the output layer to be set is by creating data files of the activation of this layer via the testing of the network. In this latter case, a network can be trained in an unsupervised manner, and this trained network is then tested with the same stimuli used to train it, thus creating a data file containing the activation of the output neural layer after a preset amount of time. This data file can then be used as the desired activation of the output layer in a supervised network, and ALOPEX will try to force an identical network to attain this activation when the same stimuli are introduced.

Supervised training can try to take many different stimuli-output pairs and match them up in the same network, which may be used to simulate the recollection of stored memories when situations encountered are similar to those that brought about such a memory in the first place. As in most pattern recognition ANNs, however, convergence problems limit the applicability of this ANN in its desired task.

Damage to a network is a concern that may be visualized in like manner. A network can be damaged by deactivation of a group of connections between specified layers. Taking all connections away from a particular layer in effect kills all neurons on that layer. Another option is to destroy connections only within specified areas on two layers, or to simply add a Gaussian noise to all weights.

A network damaged as described earlier is easily visualized or retrained in the same manner as before and, by doing so, the effects of damage on neural activity become apparent.

These ANNs, although they are three-dimensional and time-dependent, behave in a synchronized manner, as do their classic counterparts. It may be of use, for future development, to construct nets that transmit signals in a manner where the time it takes a signal to propagate from one neuron to the next is proportional to the distance between these two neurons. Hardware ANNs do so naturally, but a digital one can be constructed in a manner where it is also possible. A synchronized ANN is still viable for this application if the time increments are very small and a signal takes a number of time units proportional to this distance to travel across the synapse. Would this method better simulate the biological system? Might it also better simulate the biological model if a frequency-modulated signal is simulated? These have been questions for future research that can prove their validity.

Biologically Inspired Modular Neural Networks

The idea of building modular networks comes from the analogy with biological systems, in which a brain (as a common example) consists of a series of interconnected substructures, like auditory, vestibular, and visual systems, which, in turn, are further structured on more functionally independent groups of neurons. Each level of signal processing performs its unique and independent purpose, such that the complexity of the output of each subsystem depends on the hierarchical level of that subsystem within the whole system. For instance, in the striate cortex (area 17), simple cells provide increased activity when a bar or slit of light stimulates a precise area of the visual field at a precise orientation. Their output is further processed by complex neurons, which respond best to straight lines moving through the receptive field in a particular direction with a specific orientation. The dot-like information from ganglion and Lateral Geniculate (LG) cells is, therefore, transformed in the occipital lobe into information about edges and their position, length, orientation, and movement. Although this information represents a high degree of abstraction, the visual association areas of the occipital lobe serve as only an early stage in the integration of visual information.

The usage of modular neural networks is most beneficial when there are cases of missing pieces of data. As each module takes its input from several others, a missing connection between modules would not significantly alter that module's output.

The anticipation is that the greater the number of features per input module, the more advantageous is the usage of modular neural networks in case of missing features. One possible application of this approach can be used in face recognition, when certain parts of a face image (like nose or eyes) are not available for some images (20).

For further improvement of the algorithm, different schemes can be used to compute the local or global error factor in the ALOPEX optimization (see Appendix 1), as well as a more reliable way for adjusting the noise with respect to the global error.

As stated earlier, one type of modular neural network is a multilayer perceptron that is not fully connected. However, just deleting random connections does not make a modular neural network. Haykin (21) defines a modular neural network as follows:

> A neural network is said to be modular if the computation performed by the network can be decomposed into two or more modules (subsystems) that operate on distinct inputs without communicating with each other. The outputs of the modules are mediated by an integrating unit that is not permitted to feed information back to the modules. In particular, the integrating unit both (1) decides how the outputs of the modules should be combined to form the final output of the system, and (2) decides which modules should learn which training patterns.

The idea of modular neural networks is analogous to biological systems (22). Our brain has many different subsystems that process sensory inputs and then feed these results to other central processing neurons in the brain. For instance, consider a person who meets someone they have not seen in a long time. To remember the identity of this person, multiple sensory inputs may be processed. Foremost perhaps is the sense of sight whereby one processes what the person looks like. That may not be enough to recognize the person, as the person may have changed over the course of a number of years. However, their looks coupled with the person's voice, the sensory input from the ears may be enough to provide an identity. If those two are not enough, perhaps the person wears a distinctive cologne or perfume that the olfactory senses will process and add an input to the central processing. In addition, the sense of touch may also provide more information if the person has a firm handshake or soft hands. In this way, our biological system makes many different observations each processed first by some module and then the results sent to be further processed at a central location. Indeed, there may be several layers of processing before a final result is achieved.

In addition to different modules processing the input, the same sensor may process the input in two different ways. For example, the ears process the sound of a person's voice. The pitch, tonality, volume, and speed of a person's voice are all taken into account when one is identifying someone. However, perhaps more important is what that person says. For instance, they may tell you their name—a piece of data that is highly critical to identification. These data would be passed to the central processing to be used to match that name with the database of peoples' names that one has previously met. It is easy to postulate that what someone says is processed differently, and perhaps feeds to a different module in the next layer, than how they say it, even though the same raw data are used.

Although the concept of a modular neural network is based on biological phenomena, it also makes sense from a purely practical viewpoint. Many real-world problems have a large amount of data points. Using this large number of points as input to a fully connected multilayer perceptron results in a very large number of weights. Just blindly trying to train a network with this approach most often results in poor performance of the network, not to mention long training times because of slow convergence (23). Sometimes there are feature extraction methods, which will reduce the number of data points. However, as was the case in this project, there are times when even then the amount of data is large. As it is desirable to have the minimum number of weights that will yield good performance, a modular neural network may be a good solution. Each module is effectively able to compress its data and extract subfeatures, which then are used as input to a fully connected neural network. Without this modularity, the number of weights in the network would be far greater.

SUMMARY

Biologically inspired neural networks in computational intelligence have been proven to be more efficient in pattern recognition tasks (24). Several examples exist that prove the notion of "every neuron connected to every neuron in the network" might not be the best approach. The feed-forward approach with a huge number of inputs and many layers of neurons does not seem to be the best and most efficient way of doing computations, simply because the number of weights that have to be optimized is prohibitively large.

We presented new types of architectures that have the ability of overcoming the above-mentioned shortcomings. In addition, mathematical models of certain biological events and processes were also presented.

BIBLIOGRAPHY

1. M. F. Bear, B. W. Connors, and M. A. Pardiso, *Neuroscience: Exploring the Brain*, Philadelphia, PA: Lippincott Williams & Wilkins, 2001.
2. B. Kast, Best supporting actors, *Nature*, **412** (6848): 674, 2001.
3. Chudler, E.H., Available: http://faculty.washington.edu/chudler/cellpyr.html
4. S. Deutsch, and E. Micheli-Tzanakou, *Neuroelectric Systems*, New York: New York University Press, 1987.
5. J. Malmivuo, and R. Plonsey, *Bioelectromagnetism, Principles and Applications of Bioelectric and Biomagnetic Fields*, Oxford, U.K.: Oxford University Press, 1994.
6. R. Moreno-Bote and N. Parga, Role of synaptic filtering on the firing response of simple model neurons, *Phys. Rev. Lett.* **92** (2): 281021–281024, 2004.
7. S. Leondopulos, E. Micheli-Tzanakou, A polynomial approximation to the neuronal action potential as governed by the Hodgkin-Huxley equations, *Proc. of the 30th IEEE Northeast Bioengineering Conference*, **30**: 75–76, 2004.
8. S. Haykin, *Adaptive Filter Theory*, Englewood Cliffs, NJ.: Prentice Hall 2002.
9. T.J. Sejnowski, The book of Hebb, *Neuron* **24**: 773–776, 1999.
10. G. Bi and M. Poo, Synaptic modification by correlated activity: Hebb's postulate revisited, *Annual Rev. in Neurosci.*, **24**: 139–66, 2001.
11. S. Haykin, Z. Chen., S. Becker, Stochastic Correlative Learning Algorithms, *IEEE Trans. on Signal Process.*, **52** (8): 2004.
12. J. Hertz, A. Krogh, R.G. Palmer, Introduction to the theory of neural computation, Boston, MA: Addison-Wesley, 1991.
13. E. Micheli-Tzanakou, *Supervised and Unsupervised Pattern Recognition Feature Extraction and Computational Intelligence*, Bocat Raton, F.L.: CRC Press, 2000.
14. F. Rosenblatt, *Principles of Neurodynamics*, New York: Spartan, 1962.
15. E. Harth and E. Tzanakou, ALOPEX: A stochastic method for determining visual receptive fields, *Vision Research*, **14**: 1475–1482, 1974.
16. W. S. McCulloch, and W. H. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Math. Biophys.*, **5**: 115–133, 1943.
17. J.J. Hopfield, Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proc. of the National Academy of Sciences*, USA, **79**: 2554–2558, 1982.
18. A. L. Hodgkin and A. F. Huxle, A quantitative description of membrane current and its application to conduction and excitation in nerve, *J. of Physiol.*, **117**: 500–544, 1952.
19. D. O. Hebb, *Organization of Behavior: A Neurophysiological Theory*, New York: Wiley, 1949.
20. E. Micheli-Tzanakou, E. Uyeda, R. Ray, A. Sharma, R. Ramanujan, and J. Doug, Comparison of Neural Network Algorithms for Face Recognition, *Simulation*, **64** (1): 15–27, 1995.
21. S. Haykin, *Neural Networks: A Comprehensive Foundation*, New York: Macmillan College Publishing Company, 1994.
22. T. Hrycej, *Modular Learning in Neural Networks*, New York: Wiley, 1992.
23. C. Rodriguez, S. Rementeria, J. Martin, A. Lafuente, J. Muguerza, and J. Perez, A Modular Neural Network Approach

to Fault Diagnosis, *IEEE Trans. on Neural Net.*, **7** (2): 326–340, 1996.

24. J. Webster (ed.), Wiley encyclopedia of Electrical and Electronics Engineering. New York: Wiley. Available: http://www.wiley.com.

FURTHER READING

P. Fatt, and B. Katz, Spontaneous subthreshold activity at motor nerve endings, *J. of Physiol.*, **117**: 109, 1952.

W. Bialek, and A. Zee, Coding and computation with neural spike trains, *J. of Stat. Phy.*, **59**: 103–115, 1990.

D. M. MacKay, and W. S. McCulloch, The limiting information capacity of a neuronal link, *Bull. of Math. Biophys.*, **14**: 127–135, 1952.

F. C. Hoppensteadt, E. M. Izhikevich, Thalamo-cortical interactions modeled by weakly connected oscillators: Could brain use FM radio principles?, *Biosystems*, **48**: 85–94, 1998.

G. S. Berns, T. J. Sejnowski, A computational model of how the basal ganglia produce sequences, *J. of Cog. Neurosci.*, **10** (1): 108–121, 1998.

D. Noble, A modification of the Hodgkin-Huxley equations applicable to Purkinje fibre action and pacemaker potentials, *J. of Phys.*, **160**: 317–352, 1962.

R. FitzHugh, Impulses and physiological states in theoretical models of nerve membrane, *Biophys. J.*, **1**: 1961.

F. B. Hanson and H. C. Tuckwell, Diffusion Approximation for Neuronal Activity Including Reversal Potentials, *J. of Theoret. Neurobiol.*, **2**: 127–153, 1983.

J. L. Hindmarsh, R. M. Rose, A model of neuronal bursting using three coupled first order differential equations, *Proc. of the Royal Soc. of London B: Biol. Sci.*, **221** (1222): 87–102, 1984.

S. Wolpert, E. Micheli-Tzanakou, A neuromime in VLSI, *IEEE Trans. on Neural Networks*, **7** (2): 1996.

S. Shinomoto, and Y. Kuramoto, Phase transitions in active rotator systems, *Prog. in Theoret. Phys.*, **75**: 1105–1110, 1986.

A. D. Coop, and G. N. Reeke Jr., The composite neuron: A realistic one-compartment Purkinje cell model suitable for large-scale neuronal network simulations, *J. of Computat. Neurosci.*, **10** (2): 173–186, 2001.

J. Feng, Is the integrate-and-fire model good enough?-A review, *Neural Networks*, **14**: 955–975, 2001.

J. Feng, and P. Zhang, Behavior of integrate-and-fire and Hodgkin-Huxley models with correlated inputs, *Phys. Rev. E*, **63**: 051902.

E. M. Izhikevich, Weakly pulse-coupled oscillators, FM interactions, synchronization, and oscillatory associative memory, *IEEE Trans. on Neural Net.*, **10** (3): 1999.

E. M. Izhikevich, Which Model to Use for Cortical Spiking Neurons? *IEEE Trans. on Neural Net.*, **15**: 1063–1070, 2004.

E. Oja, A simplified neuron model as a principal component analyzer, *J. Math. Biol.*, **15**: 267–273, 1982.

B. G. Cragg, and H. N. V. Temperley, The organization of neurones: A cooperative analogy, *EEG and Clinical Neurophys.*, **6**: 85–92, 1954.

E. R. Caianiello, Outline of a theory of thought-processes and thinking machines, J. of Theoretical Biol., **1**: 204–235, 1961.

Stathis Leondopulos
Evangelia Micheli-Tzanakou
Rutgers University
Piscataway, New Jersey

COGNITIVE SYSTEMS AND COGNITIVE ARCHITECTURES

INTRODUCTION

Cognitive systems refer to computational models and systems that are in some way inspired by human (or animal) cognition as we understand it, which is a broad class of systems, not always well defined or clearly delineated. There is a variety of forms of cognitive systems. They have been developed for a variety of different purposes and in a variety of different ways. We will describe two broad categories below.

In general, computational cognitive modeling explores the essence of cognition through developing computational models of mechanisms (including representations) and processes of cognition, thereby producing realistic cognitive systems. In this enterprise, a cognitive architecture is a domain-generic and comprehensive computational cognitive model that may be used for a wide range of analysis of behavior. It embodies generic descriptions of cognition in computer algorithms and programs. Its function is to provide a general framework to facilitate more detailed computatonal modeling and understanding of various components and processes of the mind. Cognitive architectures occupy a particularly important place among all kinds of cognitive systems, as they aim to capture all basic structures and processes of the mind, and therefore are essential for broad, multiple-level, multiple-domain analyses of behavior. Developing cognitive architectures has been a difficult task. In this article, the importance of developing cognitive architectures, among other cognitive systems, will be discussed, and examples of cognitive architectures will be given.

Another common approach toward developing cognitive systems is the logic-based approach. From the logical point of view, a cognitive system is first and foremost a system that, through time, adopts and manages certain attitudes toward propositions, and reasons over these propositions, to perform the actions that will secure certain desired ends. The most important propositional attitudes are *believes that* and *knows that.* (Our focus herein will be on the latter. Other propositional attitudes include *wants that* and *hopes that*.) A propositional attitude is simply a relationship holding between an agent (or system) and one or more propositions, where propositions are declarative statements.

We can think of a cognitive system's life as being a cycle of sensing, reasoning, acting; sensing, reasoning, acting; . . ., and so on. In a cognitive system, this cycle repeats *ad infinitum*, presumably with goal after goal achieved along the way. In a logic-based cognitive system, the knowledge at the heart of this cycle is represented as formulas in one or more logics, and the reasoning in question is also regimented by these logics.

The eventual objective of cognitive systems research is to construct physically instantiated cognitive systems that can perceive, understand, and interact with their environment, and evolve and learn to achieve human-like performance in complex activities (often requiring context-specific knowledge). The readers may look into Refs. 1–4 for further information.

COGNITIVE ARCHITECTURES

In this section, we describe cognitive architectures. First, the question of what a cognitive architecture is is answered. Next, the importance of cognitive architectures is addressed. Then an example cognitive architecture is presented.

What is a Cognitive Architecture?

As mentioned earlier, a cognitive *architecture* is a comprehensive computational cognitive model, which is aimed to capture the essential structure and process of the mind, and can be used for a broad, multiple-level, multiple-domain analysis of behavior (5,6).

Let us explore this notion of architecture with an analogy. The architecture for a building consists of its overall framework and its overall design, as well as roofs, foundations, walls, windows, floors, and so on. Furniture and appliances can be easily rearranged and/or replaced and therefore they are not part of the architecture. By the same token, a cognitive architecture includes overall structures, essential divisions of modules, essential relations between modules, basic representations and algorithms within modules, and a variety of other aspects (2,7). In general, an architecture includes those aspects of a system that are relatively invariant across time, domains, and individuals. It deals with componential processes of cognition in a structurally and mechanistically well-defined way.

In relation to understanding the human mind (i.e., in relation to cognitive science), a cognitive architecture provides a concrete framework for more detailed computational modeling of cognitive phenomena. Research in computational cognitive modeling explores the essence of cognition and various cognitive functionalities through developing detailed, process-based understanding by specifying corresponding computational models of mechanisms and processes. It embodies descriptions of cognition in concrete computer algorithms and programs. Therefore, it produces runnable computational models of cognitive processes. Detailed simulations are then conducted based on the computational models. In this enterprise, a cognitive architecture may be used for broad, multiple-level, multiple-domain analyses of cognition.

In relation to building intelligent systems, a cognitive architecture specifies the underlying infrastructure for intelligent systems, which includes a variety of capabilities, modules, and subsystems. On that basis, application

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

systems may be more easily developed. A cognitive architecture also carries with it theories of cognition and understanding of intelligence gained from studying human cognition. Therefore, the development of intelligent systems can be more cognitively grounded, which may be advantageous in many circumstances (1,2).

Existing cognitive architectures include Soar (8), ACT-R (9), CLARION (6), and many others.

For further (generic) information about cognitive architectures, the readers may turn to the following websites:

http://www.cogsci.rpi.edu/˜rsun/arch.html

http://books.nap.edu/openbook.php?isbn=0309060966

as well as the following websites for specific individual cognitive architectures (Soar, ACT-R, and CLARION):

http://www.cogsci.rpi.edu/˜rsun/clarion.html

http://act-r.psy.cmu.edu/

http://sitemaker.umich.edu/soar/home

Why Are Cognitive Architectures Important?

For cognitive science, the importance of cognitive architectures lies in the fact that they are beneficial to understanding the human mind. In understanding cognitive phenomena, the use of computational simulation on the basis of cognitive architectures forces one to think in terms of process and in terms of detail. Instead of using vague, purely conceptual theories, cognitive architectures force theoreticians to think clearly. They are, therefore, critical tools in the study of the mind. Researchers who use cognitive architectures must specify a cognitive mechanism in sufficient detail to allow the resulting models to be implemented on computers and run as simulations. This approach requires that important elements of the models be spelled out explicitly, thus aiding in developing better, conceptually clearer theories. It is certainly true that more specialized, narrowly scoped models may also serve this purpose, but they are not as generic and as comprehensive and thus they are not as useful (1).

An architecture serves as an initial set of assumptions to be used for further computational modeling of cognition. These assumptions, in reality, may be based on either available scientific data (for example, psychological or biological data), philosophical thoughts and arguments, or ad hoc working hypotheses (including computationally inspired such hypotheses). An architecture is useful and important precisely because it provides a comprehensive initial framework for further modeling in a variety of task domains. Different cognitive architectures, such as Soar, ACT-R, or CLARION, embody different sets of assumptions (see an example later).

Cognitive architectures also provide a deeper level of explanation. Instead of a model specifically designed for a specific task (often in an ad hoc way), using a cognitive architecture forces modelers to think in terms of the mechanisms and processes available within a generic cognitive architecture that are not specifically designed for a particular task, and thereby to generate explanations of the task that are not centered on superficial, high level features of a task (as often happens with specialized, narrowly scoped models), that is, to generate explanations of a deeper kind. To describe a task in terms of available mechanisms and processes of a cognitive architecture is to generate explanations centered on primitives of cognition as envisioned in the cognitive architecture (e.g., ACT-R or CLARION), and therefore such explanations are deeper explanations. Because of the nature of such deeper explanations, this style of theorizing is also more likely to lead to unified explanations for a large variety of data and/or phenomena, because potentially a large variety of tasks, data, and phenomena can be explained on the basis of the same set of primitives provided by the same cognitive architecture. Therefore, using cognitive architectures leads to comprehensive theories of the mind (5,6,9), unlike using more specialized, narrowly scoped models.

Although the importance of being able to reproduce the nuances of empirical data from specific psychological experiments is evident, broad functionality in cognitive architectures is also important (9), as the human mind needs to deal with the full cycle that includes all of the following: transducing signals, processing them, storing them, representing them, manipulating them, and generating motor actions based on them. There is clearly a need to develop generic models of cognition that are capable of a wide range of functionalities to avoid the myopia often resulting from narrowly-scoped research (in psychology in particular).

In all, cognitive architectures are believed to be essential in advancing the understanding of the mind (5,6,9). Therefore, developing cognitive architectures is an important enterprise in cognitive science.

On the other hand, for the fields of artificial intelligence and computational intelligence (AI/CI), the importance of cognitive architectures lies in the fact that they support the central goal of AI/CI—building artificial systems that are as capable as human beings. Cognitive architectures help us to reverse engineer the best existing intelligent system—the human mind. They constitute a solid basis for building intelligent systems, because they are well motivated by, and properly grounded in, existing cognitive research. The use of cognitive architectures in building intelligent systems may also facilitate the interaction between humans and artificially intelligent systems because of the similarity between humans and cognitively based intelligent systems.

It is also worth noting that cognitive architectures are the antithesis of "expert systems": Instead of focusing on capturing performance in narrow domains, they are aimed to provide broad coverage of a wide variety of domains (2). Business and industrial applications of intelligent systems increasingly require broad systems that are capable of a wide range of intelligent behaviors, not just isolated systems of narrow functionalities. For example, one application may require the inclusion of capabilities for raw image processing, pattern recognition, categorization, reasoning, decision making, and natural language communications. It may even require planning, control of robotic devices, and interactions with other systems and devices. Such requirements accentuate the importance of research on broadly scoped cognitive architectures that perform a wide range of cognitive functionalities across a variety of task domains (as opposed to more specialized systems).

An Example of a Cognitive Architecture

An Overview. As an example, we will describe a cognitive architecture: CLARION. It has been described extensively in a series of previous papers, including Refs. 6,10–12. The reader is referred to these publications for further details.

Those who wish to know more about other cognitive architectures in existence (such as ACT-R or Soar) may want to see Refs. 8 and 9.

CLARION is an integrative architecture, consisting of a number of distinct subsystems, with a dual representational structure in each subsystem (i.e., implicit versus explicit representations; more later). Its subsystems include the action-centered subsystem (the ACS), the nonaction-centered subsystem (the NACS), the motivational subsystem (the MS), and the meta-cognitive subsystem (the MCS). The role of the action-centered subsystem is to control actions, regardless of whether the actions are for external physical movements or for internal mental operations. The role of the nonaction-centered subsystem is to maintain general knowledge (either implicit or explicit). The role of the motivational subsystem is to provide underlying motivations for actions in terms of providing impetus and feedback (e.g., indicating whether outcomes are satisfactory). The role of the meta-cognitive subsystem is to monitor, direct, and modify the operations of the action-centered subsystem dynamically as well as the operations of all the other subsystems.

Each of these interacting subsystems consists of two “levels” of representation (i.e., a dual representational structure): Generally, in each subsystem, the top level encodes explicit knowledge and the bottom level encodes implicit knowledge. The distinction of implicit and explicit knowledge has been amply argued for before (6,13–15). The two levels interact, for example, by cooperating in actions, through a combination of the action recommendations from the two levels respectively, as well as by cooperating in learning through a bottom-up and a top-down process (to be discussed below). See Fig. 1.

It has been intended that this cognitive architecture satisfy some basic requirements as follows. It should be able to learn with or without a priori domain-specific knowledge to begin with (unlike most other existing cognitive architectures) (11,13). It also has to learn continuously from ongoing experience in the world (as indicated by Refs. 16 and 17, and others, human learning is often gradual and ongoing). As suggested by Refs. 13 and 14, and others, there are clearly different types of knowledge involved in human learning. Moreover, different types of learning processes are involved in acquiring different types of knowledge (9,11,18). Furthermore, it should include both situated actions/reactions and cognitive deliberations (6). It should be able to handle complex situations that are not amenable to simple rules. Finally, unlike other existing cognitive architectures, it should more fully incorporate motivational processes as well as meta-cognitive processes. Based on the above considerations, CLARION was developed.

Some Details. ***The Action-Centered Subsystem.*** First, let us look into the action-centered subsystem (the ACS) of CLARION. The overall operation of the action-centered subsystem may be described as follows:

1. Observe the current state x.
2. Compute in the bottom level the Q-values of x associated with each of all the possible actions a_i's: $Q(x, a_1)$, $Q(x, a_2)$, ..., $Q(x, a_n)$.

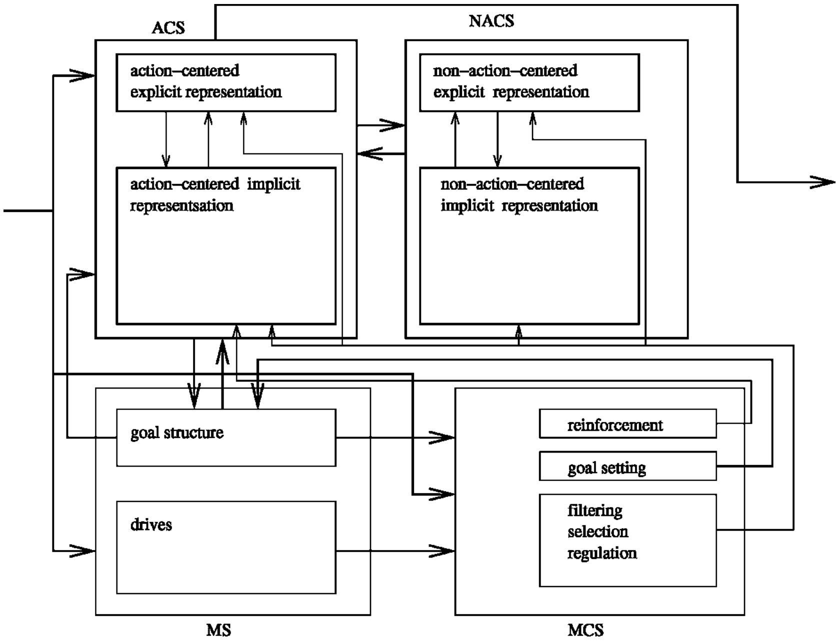

ACS
NACS
action–centered explicit representation
action–centered implicit represensation
non–action–centered explicit representation
non–action–centered implicit representation
goal structure
drives
reinforcement
goal setting
filtering
selection
regulation
MS
MCS

Figure 1. The CLARION architecture.

3. Find out all the possible actions ($b_1, b_2, \ldots, b_m$) at the top level, based on the input x (sent up from the bottom level) and the rules in place.
4. Compare or combine the values of the selected a_is with those of b_js (sent down from the top level), and choose an appropriate action b.
5. Perform the action b, and observe the next state y and (possibly) the reinforcement r.
6. Update Q-values at the bottom level in accordance with the *Q-Learning-Backpropagation* algorithm.
7. Update the rule network at the top level using the *Rule-Extraction-Refinement* algorithm.
8. Go back to Step 1.

In the bottom level of the action-centered subsystem, implicit reactive routines are learned: A Q-value is an evaluation of the “quality” of an action in a given state: $Q(x, a)$ indicates how desirable action a is in state x (which consists of some sensory input). An action may be chosen in any state based on Q-values in that state. To acquire the Q-values, the *Q-learning* algorithm (19) may be used, which is a reinforcement learning algorithm (see the articles on learning algorithms in this encyclopedia). It basically compares the values of successive actions and adjusts an evaluation function on that basis. It thereby develops reactive sequential behaviors or reactive routines [such as navigating through a body of water or handling daily activities, in a reactive way (6,12)]. Reinforcement learning is implemented in modular (multiple) neural networks. Due to such networks, CLARION is able to handle very complex situations that are not amenable to simple rules.

In the top level of the action-centered subsystem, explicit symbolic conceptual knowledge is captured in the form of explicit symbolic rules; see Ref. 12 for details. There are many ways in which explicit knowledge may be learned, including independent hypothesis-testing learning and “bottom-up learning” as discussed below.

Humans are generally able to learn implicit knowledge through trial and error, without necessarily using a priori knowledge. On top of that, explicit knowledge can be acquired also from ongoing experience in the world, possibly through the mediation of implicit knowledge (i.e., bottom-up learning; see Refs. 6,18, and 20). The basic process of bottom-up learning (which is generally missing from other existing cognitive architectures and distinguishes CLARION from others) is as follows: If an action implicitly decided by the bottom level is successful, then the agent extracts an explicit rule that corresponds to the action selected by the bottom level and adds the rule to the top level. Then, in subsequent interaction with the world, the agent verifies the extracted rule by considering the outcome of applying the rule: If the outcome is not successful, then the rule should be made more specific and exclusive of the current case; if the outcome is successful, the agent may try to generalize the rule to make it more universal (21).[1] After explicit rules have been learned, a variety of explicit reasoning methods may be used. Learning explicit conceptual representation at the top level can also be useful in enhancing learning of implicit reactive routines at the bottom level (11).

Although CLARION can learn even when no a priori or externally provided explicit knowledge is available, it can make use of it when such knowledge is available (9,22). To deal with instructed learning, externally provided knowledge, in the forms of explicit conceptual structures such as rules, plans, categories, and so on, can 1) be combined with existent conceptual structures at the top level, and 2) be assimilated into implicit reactive routines at the bottom level. This process is known as top-down learning (12).

The Non-action-Centered Subsystem. The nonaction-centered subsystem (NACS) may be used for representing general knowledge about the world (23), for performing various kinds of memory retrievals and inferences. The nonaction-centered subsystem is under the control of the action-centered subsystem (through its actions).

At the bottom level, “associative memory” networks encode nonaction-centered implicit knowledge. Associations are formed by mapping an input to an output (such as mapping “2 + 3” to “5”). For example, the regular backpropagation learning algorithm can be used to establish such associations between pairs of inputs and outputs (24).

On the other hand, at the top level of the nonaction-centered subsystem, a general knowledge store encodes explicit nonaction-centered knowledge (25). In this network, chunks are specified through dimensional values (features).[2] A node is set up in the top level to represent a chunk. The chunk node connects to its corresponding features (represented as individual nodes) in the bottom level of the nonaction-centered subsystem (25). Additionally, links between chunks encode explicit associations between pairs of chunks, known as associative rules. Explicit associative rules may be formed (i.e., learned) in a variety of ways (12).

Different from most other existing cognitive architectures, during reasoning, in addition to applying associative rules, similarity-based reasoning may be employed in the nonaction-centered subsystem. During reasoning, a known (given or inferred) chunk may be automatically compared with another chunk. If the similarity between them is sufficiently high, then the latter chunk is inferred (12,25).

As in the action-centered subsystem, top-down or bottom-up learning may take place in the nonaction-centered subsystem, either to extract explicit knowledge in the top level from the implicit knowledge in the bottom level or to assimilate explicit knowledge of the top level into implicit knowledge in the bottom level.

The Motivational and the Meta-Cognitive Subsystem. The motivational subsystem (the MS) is concerned with why an

[1] The detail of the bottom-up learning algorithm can be found in Ref. 10.

[2] The basic form of a chunk is as follows: *chunk-id*$_i$: (dim_{i_1}, val_{i_1})(dim_{i_2}, val_{i_1}) ...... (dim_{i_n}, val_{i_n}), where *dim* denotes a particular state/output dimension and *val* specifies its corresponding value. For example, *table-1:* (*size*, *large*) (*color*, *white*) (*number-of-legs*, *four*) specifies a large, four-legged, white table.

agent does what it does. Simply saying that an agent chooses actions to maximizes gains, rewards, reinforcements, or payoffs leaves open the question of what determines these things. The relevance of the motivational subsystem to the action-centered subsystem lies primarily in the fact that it provides the context in which the goal and the reinforcement of the action-centered subsystem are set. It thereby influences the working of the action-centered subsystem, and by extension, the working of the nonaction-centered subsystem.

A dual motivational representation is in place in CLARION. The explicit goals (such as "finding food") of an agent (which is tied to the working of the action-centered subsystem) may be generated based on internal drive states (for example, "being hungry"; see Ref. 12 for details).

Beyond low level drives (concerning physiological needs),[3] there are also higher level drives. Some of them are primary, in the sense of being "hard-wired".[4] Although primary drives are built-in and relatively unalterable, there are also "derived" drives, which are secondary, changeable, and acquired mostly in the process of satisfying primary drives.

The meta-cognitive subsystem (the MCS) is closely tied to the motivational subsystem. The meta-cognitive subsystem monitors, controls, and regulates action-centered and nonaction-centered processes for the sake of improving performance (26,27). Control and regulation may be in the forms of setting goals for the action-centered subsystem, setting essential parameters of the action-centered subsystem and the nonaction-centered subsystem, interrupting and changing ongoing processes in the action-centered subsystem and the nonaction-centered subsystem, and so on. Control and regulation can also be carried out through setting reinforcement functions for the action-centered subsystem. All of the above can be done on the basis of drive states and/or goals in the motivational subsystem. The meta-cognitive subsystem is also made up of two levels: the top level (explicit) and the bottom level (implicit).

Accounting for Cognitive Data. Like some other cognitive architectures (ACT-R in particular), CLARION has been successful in accounting for and explaining a variety of psychological data. For example, a number of well-known psychological tasks have been simulated using CLARION that span the spectrum ranging from simple reactive skills to complex cognitive skills. The simulated tasks include serial reaction time tasks, artificial grammar learning tasks, process control tasks, categorical inference tasks, alphabetic arithmetic tasks, and the Tower of Hanoi task (6). Among them, serial reaction time and process control tasks are typical implicit learning tasks (mainly involving implicit reactive routines), whereas Tower of Hanoi and alphabetic arithmetic are high level cognitive skill acquisition tasks (with a significant presence of explicit processes). In addition, extensive work has been done on a complex minefield navigation task, which involves complex sequential decision making (10,11). Work has also been done on an organizational decision task (28), and other social simulation tasks, as well as meta-cognitive tasks. While accounting for various psychological data, CLARION provides explanations that shed new light on cognitive phenomena.

In all of these cases of simulations, the use of the CLARION cognitive architecture forces one to think in terms of process, and in terms of details, as envisaged in CLARION. The use of CLARION also provides a deeper level of explanations. It is deeper because the explanations were centered on lower level mechanisms and processes (1,6). Due to the nature of such deeper explanations, this approach is also likely to lead to unified explanations, unifying a large variety of data and/or phenomena. For example, all the afore-mentioned tasks have been explained computationally in a unified way in CLARION.

LOGIC-BASED COGNITIVE SYSTEMS

We now give an account of logic-based cognitive systems, mentioned in broad strokes earlier.

Logic-Based Cognitive Systems in General

At any time t during its existence, the cognitive state of a cognitive system S consists in what the system knows at that time, denoted by Φ_S^t. (To ease exposition, we leave aside the distinction between what S knows versus what it merely believes.) We assume that as S moves through time, what it knows at any moment is determined, in general, by two sources: information coming directly from the external environment in which S lives, through the transducers in S's sensors that turn raw sense data into propositional content, and from reasoning carried out by S over its knowledge.

For example, suppose you learn that Alvin loves Bill, and that everyone loves anyone who loves someone. Your goal is to determine whether or not everyone loves Bill, and whether or not Katherine loves Dave. The reasoning needs to be provided in the form of an explicit series of inferences (which serves to guarantee that the reasoning in question is "surveyable").

Your knowledge (or *knowledge base*) now includes that Alvin loves Bill. (It also includes 'Everyone loves anyone who loves someone'.) You know this because information impinging upon your sensors has been transduced into propositional content added to your knowledge base. We can summarize the situation at this point is as follows:

$$\Phi_S^{t_{n+1}} = \Phi_S^{t_n} \cup \{\texttt{Loves(alvin,bill)}\}$$

Generalizing, we can define a ternary function *env* from timepoint-indexed knowledge bases, and formulas generated by *trans* applied to raw information hitting sensors, to a new, augmented knowledge base at the next timepoint. So we have:

$$\Phi_S^{t_{n+1}} = env(\Phi_S^{t_n}, trans(raw))$$

where $trans(raw) = \texttt{Loves(alvin,bill)}$.

[3] Low level drives include, for example, *need for food, need for water, need to avoid danger,* and so on (12).

[4] A few high level drives include: *desire for domination, desire for social approval, desire for following social norms, desire for reciprocation, desire for imitation* (of certain other people), and so on (12).

Now consider the second source of new knowledge, viz., reasoning. On the basis of reasoning over the proposition that Alvin loves Bill, we know that someone loves Bill, that someone loves someone, that someone whose name starts with ‘A’ loves Bill, and so on. These additional propositions can be directly deduced from the single one about Alvin and Bill; each of them can be safely added to your knowledge base.

Let $\mathcal{R}[\Phi]$ denote an augmentation of Φ via some mode of reasoning $\mathcal{R}$. Then your knowledge at the next timepoint, t_{n+2}, is given by

$$\Phi_S^{t_{n+2}} = \mathcal{R}[env(\Phi_S^{t_n}, trans(raw))]$$

As time flows on, the environment’s updating, followed by reasoning, followed by changes the cognitive system makes to the environment (the system’s actions), define the cognitive life of S.

But what is $\mathcal{R}$, and what is the structure of propositions returned by *trans*? This point is where logic enters the stage. In a logic-based cognitive system, propositions are represented by formulas in a logic, and a logic provides precise machinery for carrying out reasoning.

Knowledge Representation in Elementary Logic

In general, when it comes to any logic-based system, three main components are required: one is syntactic, one is semantic, and one is metatheoretical in nature.

The syntactic component includes specification of the alphabet of a given logical system, the grammar for building well-formed formulas (wffs) from this alphabet, and, more importantly, a proof theory that precisely describes how and when one formula can be inferred from a set of formulas. The semantic component includes a precise account of the conditions under which a formula in a given system is true or false. The metatheoretical component includes theorems, conjectures, and hypotheses concerning the syntactic component, the semantic component, and connections between them.

The simplest logics to build logic-based cognitive systems are the propositional calculus and the predicate calculus (or first-order logic, or just FOL).

The alphabet for propositional logic is an infinite list

$$p_1, p_2, \ldots, p_n, p_{n+1}, \cdots$$

of propositional variables and the five familiar truth-functional connectives $\neg$, $\rightarrow$, $\leftrightarrow$, $\wedge$, $\vee$. (The connectives can at least provisionally be read, respectively, as ‘not,’ ‘implies’ (or ‘if then’), ‘if and only if,’ ‘and,’ and ‘or.’) To say that ‘if Alvin loves Bill, then Bill loves Alvin, and so does Katherine,’ we could write

$$a_l \rightarrow (b_1 \wedge k_l)$$

where b_l and k_l are the propositional variables.

We move up to first-order logic when we allow the quantifiers $\exists x$ (‘there exists at least one thing x such that …’) and $\forall x$ (‘for all x …’); the first is known as the *existential* quantifier, and the second is known as the *universal*. We also allow a supply of variables, constants, relations, and function symbols. Using this representation, the proposition that ‘Everyone loves anyone who loves someone’ is represented as

$$\forall x \forall y (\exists z Loves(y,z) \rightarrow Loves(x,y))$$

Deductive Reasoning

The hallmark of deductive reasoning is that if the premises are true, then that which is deduced from them must be true as well. In logic, deduction is formalized in a *proof theory*. Such theories (versions of which were first invented and presented by Aristotle) are often designed not to model the reasoning of logically untrained humans, but rather to express ideal, normatively correct human deductive reasoning targeted by the logically trained. To canvass other proof theories explicitly designed to model the deductive reasoning of logically untrained humans, interested readers may consult Ref. 29.

A number of proof theories are possible (for either of the propositional or predicate calculi). When the goal is to imitate human reasoning and to be understood by humans, the proof theory of choice is *natural deduction* rather than *resolution*. The latter approach to reasoning (whose one and only rule of inference, in the end, is that from $\varphi \vee \psi$ and $\neg \varphi$ one can infer ψ), while used by a number of automated theorem provers (e.g., Otter, which, along with resolution, is presented in Ref. 30), is generally impenetrable to humans.

On the other hand, suppositional reasoning is at the heart of natural deduction. For example, one such common suppositional technique is to assume the opposite of what one wishes to establish, to show that from this assumption some contradiction (i.e., an absurdity) follows, and to then conclude that the assumption must be false. The technique in question is known as *reductio ad absurdum,* or indirect proof, or proof by contradiction. Another natural rule is that to establish that some conditional of the form $\varphi \rightarrow \psi$ (where φ and ψ are any formulas in a logic L), it suffices to suppose φ and derive ψ based on this supposition. With this derivation accomplished, the supposition can be discharged, and the conditional $\varphi \rightarrow \psi$ established. The needed conclusion from the previous example (i.e., whether or not everyone loves Bill, and whether or not Katherine loves Dave) follows readily from such reasoning. (For an introduction to natural deduction, replete with proof-construction and proof-checking software, see Ref. 31.)

Nonmonotonic Reasoning

Deductive reasoning is monotonic. That is, if φ can be deduced from some knowledge base Φ of formulas (written $\Phi \vdash_D \phi$), then for any formula $\psi \notin \Phi$, it remains true that $\Phi \cup \{\psi\} \vdash_D \phi$ In other words, when $\mathcal{R}$ is deductive in nature, new knowledge never invalidates prior reasoning.

This process is not how human cognition works in real life. For example, at present, I know that my house is standing. But if, later in the day, while away from my home and working at RPI, I learn that a vicious tornado passed over RPI, and touched down in the town of Brunswick where my house is located, I have new information that probably leads me to at least suspend judgment as to whether or not my house still stands. Or to take the much-used example from AI, if I know that Tweety is a bird, I will probably deduce that Tweety can fly, on the strength of a general principle saying that birds can fly. But if I learn that Tweety is a penguin, the situation must be revised: that Tweety can fly should now not be in my knowledge base. Nonmonotonic reasoning is the form of reasoning designed to model, formally, this kind of *defeasible* inference.

There are many different logic-based approaches that have been designed to model defeasible reasoning—default logic, circumscription, argument-based defeasible reasoning, and so on. (The *locus classicus* of a survey can be found in Ref. 32.) In the limited space available in the present chapter, we can only briefly explain one of these approaches—argument-based defeasible reasoning, because it seems to accord best with what humans do as they adjust their knowledge through time.

Returning to the tornado example, what is the argument that supports the belief that the house stands (while one sits within it)? Here is Argument 1:

(1) I perceive that my house is still standing.
(2) If I perceive ϕ, ϕ holds.
$\therefore$(3) My house is still standing.

Later on, we learned that the tornado had touched down in Brunswick, and devastating damage to some homes has come to pass. At this point (t_2), if one was pressed to articulate the current position on (3), one might offer something like this (Argument 2):

(4) A tornado has just (i.e., at some time between t_1 and t_2) touched down in Brunswick, and destroyed some houses there.
(5) My house is located in Brunswick.
(6) I have no evidence that my house was *not* struck to smithereens by a tornado that recently passed through the town in which my house is located.
(7) If a tornado has just destroyed some houses in town T, and house h is located in T, and one has no evidence that h is not among the houses destroyed by the tornado, then one ought not to believe that h was not destroyed.
$\therefore$(8) I ought not to believe that my house is still standing (i.e., I ought not to believe (3).

The challenge is to devise formalisms and mechanisms that model this kind of mental activity through time. The argument-based approach to nonmonotonic reasoning does this. Although the details of the approach must be left to outside reading (33), it should be easy enough to see that the main point is to allow one argument to shoot down another (and one argument to shoot down an argument that shoots down an argument, which revives the original, etc.), and to keep a running tab on which propositions should be believed at any particular time.

Argument 2 above rather obviously shoots down Argument 1. Should one then learn that only two houses in Brunswick were leveled, and that they are both located on the other side of the town, Argument 2 would be defeated by a third argument, because this third argument would overthrow (6). With Argument 2 defeated, (3) would be reinstated, and back in my knowledge base. Notice that this ebb and flow in argument-versus-argument activity is far more than just straight deductive reasoning. (Logic can be used to model nondeductive reasoning that is not only nonmonotonic, but also inductive, abductive, probabilistic, model-based, and analogical, but coverage of these modes of inference is beyond the scope of the present entry). For coverage of the inductive and probabilistic modes of reasoning, see Ref. 34. For coverage of model-based reasoning, which is not based solely on purely linguistic formulas, but rather on models, which are analogous to states of affairs or situations on which linguistic formulas are true or false (or probable, indeterminate, etc.), see Ref. 35.

Modal Logics

Logics can be used to represent knowledge, but advanced logics can also be used to represent knowledge about knowledge, and reasoning about knowledge about knowledge. Modeling such knowledge and reasoning is important for capturing human cognition, and in light of the fact that heretofore the emphasis in psychology of reasoning has been on modeling simpler reasoning that does not involve modals, the level of importance only grows. Consider the Wise Man Puzzle below as an illustration of modal reasoning to be captured:

> Suppose there are three wise men who are told by their king that at least one of them has a white spot on his forehead; actually, all three have white spots on their foreheads. We assume that each wise man can see the others' foreheads but not his own, and thus each knows whether the others have white spots. Suppose we are told that the first wise man says, "I do not know whether I have a white spot," and that the second wise man then says, "I also do not know whether I have a white spot." Now we would like to ask you to attempt to answer the following questions:
>
> 1. Does the third wise man now know whether or not he has a white spot?
> 2. If so, what does he know, that he has one or doesn't have one?
> 3. And, if so, that is, if the third wise man does know one way or the other, provide a detailed account (showing all work, all notes, etc.; use scrap paper as necessary) of the reasoning that produces his knowledge.

The logic able to answer these questions is a modal propositional epistemic logic; we refer to it simply as $\mathcal{L}_{KT}$. This logic is produced by adding to the propositional calculus the modal operators $\Box$ (traditionally interpreted as "necessarily") and $\Diamond$ (traditionally interpreted as "possibly"), with subscripts on these operators to refer to cognitive systems. Because we are here concerned with what cognitive systems believe and know, we will focus on the box, and will rewrite $\Box_\alpha$ as $\mathbf{K}_\alpha$ [i.e., cognitive system α knows (something)]. So, to represent that 'Wise man A knows he doesn't have a white spot on his forehead,' we can write $\mathbf{K}_A$ ($\neg$White(A)). Here's the grammar for $\mathcal{L}_{KT}$.

1. All wffs in the propositional calculus are wffs.
2. If ϕ is a closed wff, and α is a constant, then $\Box_\alpha\phi$ is a wff. Since we are here concerned with doxastic matters, that is, matters involving believing and knowing, we say that $\mathbf{B}_\alpha\phi$ is a wff, or, if we are concerned with 'knows' rather than 'believes,' that $\mathbf{K}_\alpha\phi$ is a wff.
3. If ϕ and ψ are wffs, then so are any strings that can be constructed from ϕ and ψ by the usual propositional connectives (e.g., $\rightarrow, \Lambda, \ldots$).

Next, here are some key axioms and rules of inference:

K $\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi)$

T $\Box\phi \rightarrow \phi$

LO ("logical omniscience") Where $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, from $\Phi \vdash_D \psi$ and $\mathbf{K}_\alpha\phi_1$, $\mathbf{K}_\alpha\phi_2, \ldots$ infer $\mathbf{K}_\alpha\psi$

The first rule says that if one knows a conditional, then if one knows the antecedent of the conditional, one knows the consequent. The second says that if one knows some proposition, that proposition is true. The inference rule LO says that the agent α knows that which can be deduced from what she knows. This rule of inference, without restrictions placed on it, implies that if a knows, say, the axioms of set theory (which are known to be sufficient for deductively deriving all of classical mathematics from them), α knows all of classical mathematics, which is not cognitively plausible. Fortunately, LO allows for the introduction of parameters that more closely match the human case. For example LO^n would be the rule of inference according to which α knows the consequences of what she knows, as long as the length of the derivations (in some fixed proof theory) of the consequences does not exceed n steps.

To ease exposition, we restrict the solution to the two-wise man version. In this version, the key information consists in these three facts:

1. A knows that if A does not have a white spot, B will know that A does not have a white spot.
2. A knows that B knows that either A or B has a white spot.
3. A knows that B does not know whether or not B has a white spot.

Here is a proof in $\mathcal{L}_{KT}$ that solves this problem:

1. $\mathbf{K}_A(\neg\text{White(A)} \rightarrow \mathrm{K}_{\mathrm{B}}(\neg\text{White(A)}))$ (first fact)
2. $\mathbf{K}_A(\mathbf{K}_B(\neg\text{White(A)} \rightarrow \text{White(B)}))$ (second fact)
3. $\mathbf{K}_A(\neg\mathbf{K}_B(\text{White(B)}))$ (third fact)
4. $\neg\text{White(A)} \rightarrow \mathbf{K}_B(\neg\text{White(A)})$ 1, T
5. $\mathbf{K}_B(\neg\text{White(A)} \rightarrow \text{White(B)})$ 2, T
6. $\mathbf{K}_B(\neg\text{White(A)}) \rightarrow \mathbf{K}_B(\text{White(B)})$ 5, K
7. $\neg\text{White(A)} \rightarrow \mathbf{K}_B(\text{White(B)})$ 4, 6
8. $\neg\mathbf{K}_B(\text{White(B)}) \rightarrow \text{White(A)}$ 7
9. $\mathbf{K}_A(\neg\mathbf{K}_B(\text{White(B)}) \rightarrow \text{White(A)})$ 4–8, 1, LO
10. $\mathbf{K}_A(\neg\mathbf{K}_B(\text{White(B)})) \rightarrow \mathbf{K}_A(\text{White(A)})$ 9, K
11. $\mathbf{K}_A(\text{White(A)})$ 3, 10

The foregoing solution closely follows that provided by Ref. 32; this solution lacks a formal semantics for the inference rules in question. For a fuller version of a solution to the arbitrarily iterated n-wise man version of the problem, replete with a formal semantics for the proof theory used, and a real-life implementation that produces a logic-based cognitive system, running in real time, that solves this problem; see Ref. 36.

Examples of Logic-Based Cognitive Systems

There are many logic-based cognitive systems that have been engineered. It is important to know that they can be physically embodied, have to deal with rapid-fire interaction with the physical environment, and still run efficiently.

For example, Amir and Maynard-Reid (37) built a logic-based robot able to carry out clerical functions in an office environment; similar engineering has been carried out in Ref. (38). For a set of recent examples of readily understood, small-scale logic-based cognitive systems doing various things that humans do; see Ref. 39.

There is insufficient space to put on display an actual logic-based cognitive system of a realistic size here. So see the afore-mentioned references for further details.

CONCLUDING REMARKS

In recent decades, the research on cognitive systems has progressed to the extent that we can start to build computational systems that mimic the human mind to some degree, although there is a long way to go before we can fully understand the architecture of the human mind and thereby develop computational cognitive systems that replicate its full capabilities.

Some example cognitive systems have been presented here. Yet, it is still necessary to explore more fully the space of possible cognitive systems (40,41), to further advance the state of the art in cognitive systems, in cognitive modeling, and in cognitive science in general. It will also be necessary to enhance the functionalities of cognitive systems so that they can be capable of the full range of intelligent behaviors. Many challenges and issues need to be addressed (1,2). We can expect that the field of cognitive systems will have a significant and meaningful impact on cognitive science and on computer science both in terms of understanding cognition and in terms of developing artificially intelligent systems. The goal of constructing embodied systems that can perceive, understand, and interact with their environment to achieve human-like performance in various activities drives this field forward.

BIBLIOGRAPHY

1. R. Sun, The importance of cognitive architectures: An analysis based on CLARION, *J. Experimen. Theoret. Artif. Intell.*, **19** (2): 159–193, 2007.

2. P. Langley, J. Laird, and S. Rogers, Cognitive architectures: Research issues and challenges, *Cog. Sys. Res.*, In press.
3. R. W. Pew and A. S. Mavor (eds), *Modeling Human and Organizational Behavior: Application to Military Simulations*. Washington, D.C.: National Academy Press, 1998.
4. F. Ritter, N. Shadbolt, D. Elliman, R. Young, F. Gobet, and G. Baxter, *Techniques for Modeling Human Performance in Synthetic Environments: A Supplementary Review*. Dayton, OH: Human Systems Information Analysis Center, Wright-Patterson Air Force Base, 2003.
5. A. Newell, *Unified Theories of Cognition*, Cambridge, MA: Harvard University Press, 1990.
6. R. Sun, *Duality of the Mind*, Mahwah, N.J.: Lawrence Erlbaum Associates, 2002.
7. R. Sun, Desiderata for cognitive architectures, *Philosoph. Psych.*, **17** (3): 341–373, 2004.
8. P. Rosenbloom, J. Laird, and A. Newell, *The SOAR Papers: Research on Integrated Intelligence*. Cambridge, MA: MIT Press, 1993.
9. J. Anderson and C. Lebiere, *The Atomic Components of Thought*. Mahwah, NJ: Lawrence Erlbaum Associates, 1998.
10. R. Sun and T. Peterson, Autonomous learning of sequential tasks: experiments and analyses, *IEEE Trans. Neural Networks*, **9** (6): 1217–1234, 1998.
11. R. Sun, E. Merrill, and T. Peterson, From implicit skills to explicit knowledge: A bottom-up model of skill learning, *Cog. Sci.*, **25** (2): 203–244, 2001.
12. R. Sun, *A Tutorial on CLARION*. Technical report, Cognitive Science Department, Rens-selaer Polytechnic Institute. Available: http://www.cogsci.rpi.edu/~rsun/sun.tutorial.pdf.
13. A. Reber, Implicit learning and tacit knowledge, *J. Experimen. Psych.: General*, **118** (3): 219–235, 1989.
14. C. Seger, Implicit learning, *Psycholog. Bull.*, **115** (2): 163–196, 1994.
15. A. Cleeremans, A. Destrebecqz and M. Boyer, Implicit learning: News from the front. *Trends in Cog. Sci.*, **2** (10): 406–416, 1998.
16. D. Medin, W. Wattenmaker, and R. Michalski, Constraints and preferences in inductive learning: An experimental study of human and machine performance, *Cog. Sci.*, **11**: 299–339, 1987.
17. R. Nosofsky, T. Palmeri, and S. McKinley, Rule-plus-exception model of classification learning, *Psycholo. Rev.*, **101** (1): 53–79, 1994.
18. A. Karmiloff-Smith, From meta-processes to conscious access: Evidence from children's metalinguistic and repair data, *Cognition*, **23**: 95–147, 1986.
19. C. Watkins, *Learning with Delayed Rewards*. Ph.D Thesis, Cambridge, UK: Cambridge University, 1989.
20. W. Stanley, R. Mathews, R. Buss, and S. Kotler-Cope, Insight without awareness: On the interaction of verbalization, instruction and practice in a simulated process control task, *Quart. J. Experimen. Psych.*, **41A** (3): 553–577, 1989.
21. R. Michalski, A theory and methodology of inductive learning, *Artif. Intell.*, **20**: 111–161, 1983.
22. W. Schneider and W. Oliver, An instructable connectionist/control architecture, in K. VanLehn (ed.), *Architectures for Intelligence*, Hillsdale, NJ: Erlbaum, 1991.
23. M. R. Quillian, Semantic memory, in M. Minsky (ed.), *Semantic Information Processing*. Cambridge, MA: MIT Press, 1968, pp. 227–270.
24. D. Rumelhart, J. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Micro structures of Cognition*. Cambridge, MA: MIT Press, 1986.
25. R. Sun, Robust reasoning: Integrating rule-based and similarity-based reasoning. *Artif. Intell.*, **75** (2): 241–296, 1995.
26. T. Nelson, (ed.) *Metacognition: Core Readings.* Allyn and Bacon, 1993.
27. J. D. Smith, W. E. Shields, and D. A. Washburn, The comparative psychology of uncertainty monitoring and metacognition, *Behav. Brain Sci.*, **26** (3): 317–339, 2003.
28. R. Sun and I. Naveh, Simulating organizational decision making with a cognitive architecture CLARION, *J. Artif. Soc. Social Simulat.*, **7** (3): 2004. *http://jasss.soc.surrey.ac.uk/7/3/5.html*
29. L. Rips, *The Psychology of Proof*. Cambridge, MA: MIT Press, 1994.
30. L. Wos, R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning: Introduction and Applications*. New York: McGraw Hill, 1992.
31. J. Barwise and J. Etchemendy, *Language, Proof and Logic*, New York: Seven Bridges, 1999.
32. M. Genesereth and N. Nilsson, *Logical Foundations of Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann, 1987.
33. J. L. Pollock, How to reason defeasibly, *Artif. Intell.*, **57** (1): 1–42, 1992.
34. B. Skyrms, *Choice and Chance: An Introduction to Inductive Logic*. Belmont, CA: Wadsworth, 1999.
35. P. Johnson-Laird, *Mental Models*. Harvard, MA: Harvard University Press, 1983.
36. K. Arkoudas and S. Bringsjord, Metareasoning for multi-agent epistemic logics. *Fifth International Conference on Computational Logic In Multi-Agent Systems (CLIMA 2004)*, Lecture Notes in Artificial Intelligence (LNAI), **3487**: 111–125, 2005.
37. E. Amir and P. Maynard-Reid, LiSA: A robot driven by logical subsumption, *Proc. of the Fifth Symposium on the Logical Formalization of Commonsense Reasoning*, AAAI Press, 2001.
38. S. Bringsjord, S. Khemlani, K. Arkoudas, C. McEvoy, M. Destefano, and M. Daigle, Advanced Synthetic Characters, Evil, and E, in M. Al-Akaidi and A. El Rhalibi (eds.), *Game-On 2005, 6th International Conference on Intelligent Games and Simulation*, Ghent-Zwijnaarde, Belgium: European Simulation Society, 2005, pp 31–39.
39. E. Mueller, *Commonsense Reasoning*. San Francisco, CA: Morgan Kaufmann, 2006.
40. A. Sloman and R. Chrisley, More things than are dreamt of in your biology: Information processing in biologically-inspired robots. *Cog. Sys. Res.*, **6** (2): 145–174, 2005.
41. R. Sun and C. Ling, Computational cognitive modeling, the source of power and other related issues. *AI Magazine*, **19** (2): 113–120, 1998.

FURTHER READING

A. Newell, *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press, 1990.

A. Newell and H. Simon, Computer science as empirical inquiry: Symbols and search. Commun. of ACM, **19**: 113–126, 1976.

D. Rumelhart, J. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*. Cambridge, MA: MIT Press, 1986.

R. Sun, *Duality of the Mind*. Mahwah, N.J.: Lawrence Erlbaum Associates, 2002.

R. Sun, P. Slusarz, and C. Terry, The interaction of the explicit and the implicit in skill learning: A dual-process approach, *Psycholog. Rev.*, **112** (1): 159–192, 2005.

R. Sun, *Integrating Rules and Connectionism for Robust Commonsense Reasoning*. New York: John Wiley & Sons, 1994.

Ron Sun
Selmer Bringsjord
Rensselaer Polytechnic
Institute
Troy, New York

DIMENSIONALITY REDUCTION

INTRODUCTION

A dimension refers to a measurement of a certain aspect of an object. Dimensionality reduction is the study of methods for reducing the number of dimensions describing the object. Its general objectives are to remove irrelevant and redundant data to reduce the computational cost and avoid data over-fitting (1) and to improve the quality of data for efficient data-intensive processing tasks such as pattern recognition and data mining. Dimensionality reduction is an effective solution to the problem of “curse of dimensionality.” When the number of dimensions increases linearly, experiments have shown that the required number of examples for learning increases exponentially (2). Figure 1 shows an example of the curse of dimensionality.

In practice, researchers and practitioners interchangeably use dimension, feature, variable, and attribute. Similarly, we will interchangeably use object, example, vector, and instance. Consider an application in which a system processes data (speech signal, images, or patterns in general) in the form of a collection of vectors. For a particular application, it is more often than not that a subset of features is relevant, and in some cases, a large number of features are irrelevant. This problem can be caused by factors such as (i) many dimensions will have variation smaller than the measurement noise and thus will be *irrelevant*, and (ii) many dimensions will be correlated (through linear combinations or functional dependence) to others and thus will be *redundant*. Therefore, in many situations, it is recommended to remove the irrelevant and redundant dimensions, producing a more economical representation of the data (6).

Dimensionality reduction is a research area at the intersection of several disciplines, including statistics, databases, data mining, text mining, pattern recognition, machine learning, artificial intelligence, visualization, and optimization. Each of these areas has a way of looking at the problem. For example, in pattern recognition the problem of dimensionality reduction is to extract a small set of features that recovers most of the variability of the data. In text mining, however, the problem is defined as selecting a small subset of words or terms (not new features that are combination of words or terms). Use of this important technique also varies with the application domain. Examples of applications of dimensionality reduction techniques include mining of text documents, gene structure discovery, image processing, statistical learning, and exploratory data analysis. Different applications need to be treated with different techniques. Depending on the application, new features may be extracted as in the case of exploratory analysis, or a small subset of original features are selected as in the case of gene structure discovery.

Dimensionality reduction has been a subject of much research currently and over the past several decades [some good overviews are available (7–10)]. In particular, the pioneering work of Sammon (11) has given inspiration to today’s information processing systems. Sammon, in the late 1970s, combined dimensionality reduction with issues such as classification and interactive visual data analysis. Recently, there is a renewed interest in this topic due to massive data of large dimensionality created in data mining, data warehousing, and knowledge discovery applications. Other applications such as genome project, text mining, and web mining also require efficient dimensionality reduction methods.

Dimensionality reduction methods can be grouped in various ways: (i) feature selection or feature extraction, (ii) linear or nonlinear, (iii) supervised or unsupervised, and (iv) local or global. Dimensionality reduction methods are often classified into feature selection or feature extraction. In feature selection, a subset of original features are selected in the end. In feature extraction, new features are extracted using some mapping (linear or nonlinear) from the original set of features. Linear methods such as principal components analysis (PCA) use a linear mapping to extract new features from original features (9). Similarly, nonlinear methods such as Sammon’s mapping (7), locally linear embedding (12), and ISOMAP (13) use a nonlinear mapping to extract new features. Supervised methods can take advantage of any class information present in the data, whereas unsupervised methods do not use this class information. One limitation of the supervised methods is that characteristic variables that describe examples of infrequent classes tend to be easily removed as a result of dimensionality reduction making use of the class distribution. Typically, supervised dimensionality reduction methods can be further divided into local or global methods. In a local method, features are selected for each category of the class feature; in the case of a global method, features are selected for all categories. Among these different ways of categorizing dimensionality reduction methods, we will mainly describe various methods of dimensionality reduction methods in terms of feature extraction or feature selection.

In the following two sections, we introduce the basic concepts and key techniques of feature extraction and selection, respectively. We then discuss some dimensionality reduction methods in practice.

FEATURE EXTRACTION

Feature extraction can be defined as follows: Given a set of features $S = \{v_1, v_2, \ldots, v_D\}$, find a new set of features S' derived from a linear or nonlinear mapping of S. The cardinality of $|S'| = d$ and $J(S') \geq J(T)$ for all derived set of features T with $|T| = d$, where J is the evaluation function. Here d or some other parameter that can determine d

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

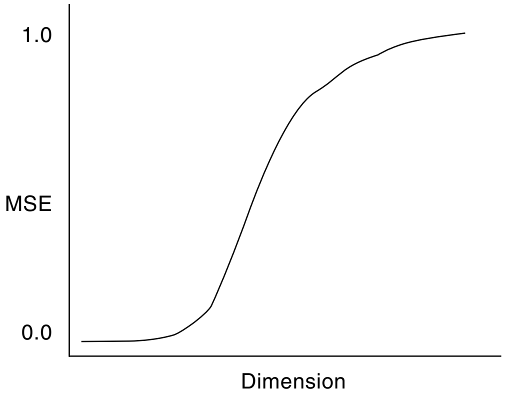


Figure 1. An example of curse of dimensionality: MSE is the mean-squared error of a 1-nearest neighbor rule (3, 4). Each dimension is generated uniformly on $[-1, 1]$. As the dimensionality increases, the MSE increases very sharply until it levels off at 1.0. This happens as early as dimensionality $= 10$. See Ref. 5 for further details.

(e.g., a threshold eigen value) is usually specified by the user.

When all existing features are recombined to yield new features, then we are dealing with feature extraction. Hence a mapping is defined that transforms any original D dimensional feature vector into a new d-dimensional feature vector. Ideally the mapping conserves or even enhances the discriminatory information while reducing the dimensionality of the feature vector. Mapping can be linear or nonlinear. Figure 2 depicts this pictorially. The following descriptions of a sample of classic feature extraction methods will bring out the methodical difference between feature transformation and selection (Fig. 3).

D original features
Extract New Features by Linear/Nonlinear Mapping
d extracted features

Figure 2. Feature extraction process.

Principal Components Analysis (PCA)

It is the most widely used linear feature extraction method (14). It is also called the Karhunen–Loeve transform in signal processing literature. The class information is not taken into consideration in this method. The objective of this method is to find a set of d orthogonal basis vectors that maximally captures the relationship between the original dimensions. It can be shown that the *jth* principal component direction is along an eigenvector direction of the global covariance matrix $C = \frac{1}{N-1}\sum_{j=1}^{N}[(x_j - \mu)^T(x_j - \mu)]$ of the feature vector x and its global mean μ. The first d eigenvectors e_j then define a $D \times d$ matrix M with the eigenvectors as columns that transforms the original sample x to the extracted sample $y = M^T x^T$.

The eigenanalysis can be done using standard mathematical software, e.g., Ref. 15. PCA is an information conserving transform. Using all d' ($d \leq d' \leq D$) nonzero eigenvectors conserves the information contained in the

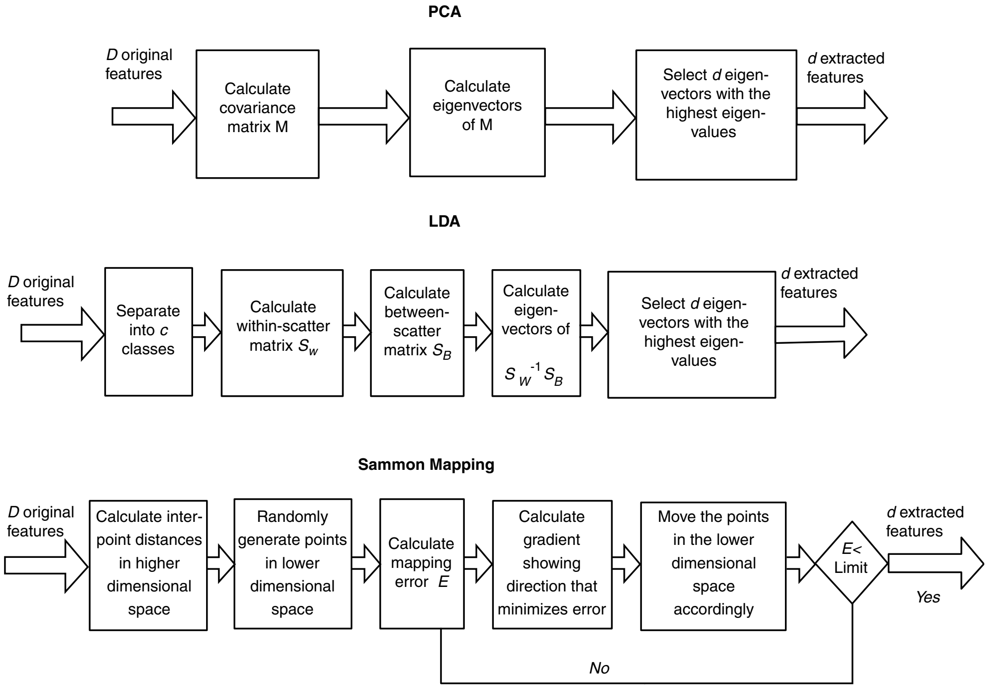

PCA
D original features
Calculate covariance matrix M
Calculate eigenvectors of M
Select *d* eigen-vectors with the highest eigen-values
d extracted features
LDA
D original features
Separate into *c* classes
Calculate within-scatter matrix S_w
Calculate between-scatter matrix S_B
Calculate eigen-vectors of $S_W^{-1} S_B$
Select *d* eigen-vectors with the highest eigen-values
d extracted features
Sammon Mapping
D original features
Calculate inter-point distances in higher dimensional space
Randomly generate points in lower dimensional space
Calculate mapping error *E*
Calculate gradient showing direction that minimizes error
Move the points in the lower dimensional space accordingly
E< Limit
d extracted features
Yes
No

Figure 3. Procedures for PCA, LDA, and Sammon mapping.

orthogonal features. The new features are orthogonal to each other, and there is no covariance (and correlation) between any two features. The variance of the new feature y_j is the eigenvalue λ_j. The original feature vector can be reconstructed as $x = My^T$. The truncation of those eigenvectors, which are associated with the smallest eigenvalues, does not incur a large information loss (approximate loss: $\epsilon = \sum_{j=d+1}^{d'} \lambda_j$). PCA can therefore be used as an information-conserving, correlation-eliminating, and dimensionality reduction feature extraction method.

The process of determining most influential features having maximum eigenvalues is called singular value decomposition (SVD). Another method similar to PCA is called latent semantic indexing (LSI). LSI has been successfully used in information retrieval for clustering documents. Below we give brief descriptions of some applications using PCA and LSI.

Linear Discriminative Analysis (LDA)

Unlike PCA, LDA considers the class information. The only difference between PCA and LDA is the matrix that is considered. LDA uses a within-scatter matrix of all c classes: $S_W = \sum_{i=1}^{c} \sum_{j=1}^{N_i} [(x_j - \mu_i)^T (x_j - \mu_i)]$, and a between-scatter matrix: $S_B = \sum_{i=1}^{c} [(\mu_i - \mu)^T (\mu_i - \mu)]$, where N_i is the number of objects within class i, μ_i is the common mean of class i, and μ is the mixture mean of all classes. Then at most $d = c - 1$ nonzero eigenvectors, associated with the largest eigenvalues of the matrix $S_W^{-1} S_B$, define the $D \times d$ feature extraction matrix M. This transformation maximizes the between-class scatter while minimizing the within-class scatter (i.e., maximize $\frac{det(S_B)}{det(S_W)}$), where $det(.)$ denotes determinant of a square matrix. Such a transformation should retain class separability while reducing the variation due to other sources. The objective of LDA is to perform dimensionality reduction while preserving as much of the class discriminatory information as possible. It seeks to find directions along which the classes are best separated. LDA works well if the data have a multivariate normal distribution.

Sammon Map

Sammon map is an example of a nonlinear feature extraction method, whereas the above methods are all linear [7]. It is mainly used for two-dimensional (2-D) visualization of high-dimensional data. The nonlinear mapping can be performed for any dimensionality $d < D$. Sammon's algorithm uses the gradient descent technique to minimize an error function in order to map from D to d dimensions. The error function is given as

$$E = \frac{1}{\sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \delta_{jk}} \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \frac{\delta_{jk} - d_{jk}^2}{\delta_{jk}}$$

where δ_{jk} is the distance between two points in the extracted d-dimensional space and d_{jk} is the distance between two points in the original D-dimensional space. The mapping attempts to fit N points in the lower space, such that their inter-point distances approximate the corresponding distances in the higher space. In Ref. 16, the linear feature extraction method PCA is compared with nonlinear methods such as Sammon's mapping, multidimensional scaling (17), and self-organizing mapping (18) using texture data. Results clearly show that classification performance improves, particularly for small values of d, by using nonlinear methods compared with that of linear methods such as PCA. The reason being, performing nonlinear feature extraction provides a better characterization of the data in a smaller number of features.

FEATURE SELECTION

Feature selection can be defined as follows: Given a set of features $S = \{v_1, v_2, \ldots, v_D\}$, find a subset S' of S with $|S'| = d$ such that $J(S') \geq J(T)$ for all $T \subset S$, $|T| = d$, where J is the evaluation function. Here d is usually specified by the user.

A feature selection algorithm requires the following ingredients: a generation or search strategy, an evaluation method, a stopping criterion, and/or a validation method (19–22). See Fig. 4 for a block diagram showing relationships between these components. The search or generation strategy decides the way in which combinations of features are tested for a certain goodness. As exhaustive search is usually prohibitive, alternative strategies must be employed. The evaluation or selection function assesses the goodness of a set of features and provides a ranking possibility for the selection process. The stopping criterion is of less importance. Usually a predefined number of features to be selected decides the stopping of search procedures. Validation is not part of the selection process, but it is nonetheless carried out to check the validity of the

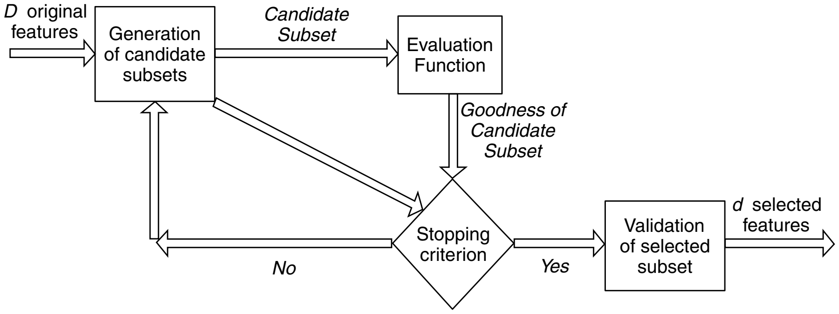


Figure 4. Feature selection process.

selected features. In the following we briefly discuss the most important search strategies and evaluation criteria.

Generation/Search Procedure

With a total of D features, there are 2^D candidate feature subsets to be searched. This is number huge even for moderate D. In the literature, there are different approaches for solving this problem, namely, complete, heuristic, and random.

Complete Search. Schlimmer (23) argues that just because the search must be complete does not mean that it must be exhaustive. Different heuristic functions are used to reduce the search space without jeopardizing the chances of finding the optimal subset. The branch-and-bound method (24) is one such method that guarantees optimality if the features obey monotonocity. Unfortunately many often-used evaluation criteria such as classifier error rate are not monotonic.

Heuristic Search. In each iteration of this method, all remaining features yet to be selected are considered for selection. This is called sequential forward selection. When the search is backward, it is called sequential backward selection. There are also combinations of these two approaches such as beam search (25).

Random Search. It searches randomly and usually stops after a maximum number of iterations. This has advantages over the heuristic method in that, unlike the heuristic methods, it is less likely to be trapped in local optima (26).

Evaluation Functions

An evaluation function or selection criterion $J()$ aims at finding the best set of features in the reduced dimensionality d from the set of all features. Hence the best set S' maximizes the criterion function over all other possible combinations of d features. Some important evaluation functions are briefly described here.

Distance Measure. It is also known as a separability, divergence, or discrimination measure. For a two-class problem, a feature f_i is preferred to another feature f_j if f_i induces a greater difference between the two-class conditional probabilities than f_j (27).

Information Measure. These measures typically determine the information gain from a feature that is the difference between the prior uncertainty and the expected posterior uncertainty using the feature. The feature giving higher information gain is selected (28).

Dependence Measures. It quantifies the ability of a feature to predict the value of the class variable. An example is the correlation coefficient. If feature f_i has a higher correlation with the class variable than feature f_j, then feature f_i is preferred (29).

Consistency Measures. These measures prefer a consistent hypothesis definable over as few features as possible. A feature set is consistent if for the same set of values for the feature set the class variable does not change (26).

Classifier Error Rate Measure. The above four types of criteria are typically known as *filter* type, whereas the classifier error rate is known as *wrapper* type. The classifier that will be used after feature selection is also used to select the features. The feature set giving the minimum classifier error rate is selected. More details on wrapper methods are given in Ref. 30.

Feature Selection for Unsupervised Learning

The above discussion is mostly for supervised learning where class information is available. Lately, feature selection has been attempted for unsupervised learning, and among different unsupervised learning, it has been mostly applied to clustering and visualization.

In the last several years, several methods for feature selection for clustering are proposed, most of which are wrapper in approach. Here a clustering algorithm is used to evaluate the candidate feature subsets. Wrapper methods can be categorized based on whether they select features for the whole data (global type) or for each cluster separately (local type). The global type assumes a subset of features to be more important than others for the whole data, whereas the local type assumes each cluster to have a subset of important features. In the case of global type, a feature selection method is run over the whole data, whereas for local type, first clustering is done over the data using all features and then important features are selected for each cluster separately using a feature selection method.

Selecting a set of features for unsupervised learning such as clustering is arguably more difficult than selecting for supervised learning such as classification because of the absence of any class information in the former. This is also the reason for extensive research being conducted for feature selection for classification compared with that for clustering. The difficult part is to evaluate the candidate subsets and compare against each other to select the optimal subset of features. First of all, quantifying the quality of clustering is far less straightforward and less accurate than classification. On top of it, one must compare the quality of clustering across varying dimensionality to select the optimal subset of features. So, one requires evaluation methods that are invariant to varying dimensionality. Examples of such methods used in various research work are trace measure (31,32), visualization (32,33), ranking of features and user selects several of the most important features (31,32,34–37), Bayesian statistical estimation framework (38), and entropy (31,39). These methods are of the global type. Examples of local methods are Manhattan distance (40) and dense regions (41).

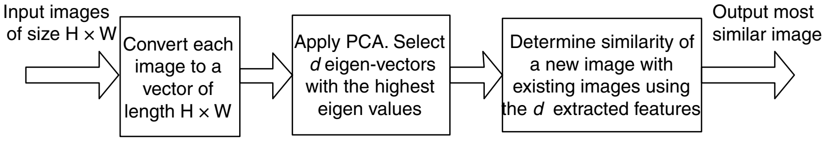

Input images of size H × W
Convert each image to a vector of length H × W
Apply PCA. Select d eigen-vectors with the highest eigen values
Determine similarity of a new image with existing images using the d extracted features
Output most similar image

Figure 5. Procedure for face recognition.

DIMENSIONALITY REDUCTION IN PRACTICE

In this section we discuss some applications of feature extraction and selection methods.

Uses of Feature Extraction Methods

Uses of PCA in Regression Analysis. PCA can be used in regression analysis in several ways (9). If the independent variables are highly correlated, then they can be transformed to principal components (PCs) and the PCs can be used as the independent variables. If we do not want to transform the independent variables, then the PCs can be used indirectly to improve the precision of the regression parameter estimates associated with the independent variables. PCA can also be used as a diagnostic tool to detect multicolinearities among the independent variables. Multicolinearity means that one or more independent variables are essentially linear combinations of other independent variables.

Using PCs to detect Outlying and Influential Observations. A major advantage of PCA is that if the first two PCs account for a substantial portion of the total variation, then we can approximate the distribution of the observations in the variable space by plotting the PCs (9). This 2-D representation of the D-dimensional observations can be used in several ways. The plot can be examined for outlying observations or for influential observations, or it can be used to see whether the observations can be visually clustered. Outlying observations are observations that lie at a considerable distance from the bulk of the observations or do not conform to the general pattern the observations exhibit. Outlying observations are called influential observations if their deletion from a particular analysis leads to different results.

Use of PCs in Cluster Analysis. If the first two or three PCs account for a substantial proportion of the total variation, then we can also use the plots to visually identify clusters (9). A *cluster* is a group of observations that are “closer” to each other than they are to observations in other clusters or groups. Many clustering algorithms are used to cluster data (42). No significant advantage exists in transforming the original observations to principal components before the clustering because the same information is contained in the original and in the transformed data. That is, for any distance function, the distances among examples computed from principal components are equal to the corresponding distances computed from the original variables using an equivalent but different distance function. The only advantage of employing PCs in cluster analysis is to be able to plot the components and visually search for clusters of observations. PCs can also be used to verify the clusters determined on the basis of another clustering algorithm. We can see whether the defined clusters are homogeneous, distinct, and aesthetically appealing to the eye. Clustering algorithms will define clusters even if none exist, i.e., even if the observations are evenly spread throughout the variable space. For this reason, a plot of the data on the first two PCs can be informative if they account for a large portion of the total variance.

Application to Computer Vision. PCA is used in computer vision to find patterns and to compress the images (43).

PCA for Finding Patterns. An example of its application to face recognition is as follows (Fig. 5). Say we have 20 images. Each image is H pixels high by W pixels wide. For each image we can create an image vector of $H \times W$ dimensions. We can then put all images together in one big image-matrix as follows:

$$\begin{pmatrix} ImageVec1 \\ ImageVec2 \\ . \\ . \\ ImageVec20 \end{pmatrix}$$

which gives us a starting point for the PCA analysis. Once PCA is performed, we have original data in terms of the eigenvectors found from the covariance matrix. Why is this useful? Say we want to do facial recognition, and so our original images were of peoples faces. Then, the problem is, given a new image, whose face from the original set is it? The way this is done in computer vision is to measure the difference between the new image and the original images, not along the original axes, but along the new axes derived from the PCA analysis. It turns out that these new axes work much better for recognizing faces, because the PCA analysis has extracted these new axes *based on their ability to capture the variability among the images*. In a way, the PCA analysis can identify the statistical patterns in the data. As all vectors have $H \times W$ dimensions, we will get $H \times W$ eigenvectors. In practice, one can leave out most of the less significant eigenvectors and the recognition still performs well.

PCA for Image Compression. Using PCA for image compression is also known as the Hotelling or Karhunen–Loeve (KL) transform. If there are 20 images, each with $H \times W$

pixels, one can form $H \times W$ vectors, each with 20 dimensions. Each vector consists of all intensity values from the same pixel from each picture. Notice that this factor is different from the previous example. By performing PCA on this, one gets 20 eigenvectors because each vector is 20-D. To compress the data, one then chooses to transform the data only using, say five eigenvectors. This gives a final dataset with only five dimensions, which has saved three quarters of the space. However, when the original data are reproduced, the images have lost some information. This compression technique is said to be lossy because the decompressed image is not exactly the same as the original.

Applications of PCA to Microarray Gene Expression Data. Each data point produced by a DNA microarray hybridization experiment represents the ratio of expression levels of a particular gene under two different experimental conditions. The result, from an experiment with n genes on a single chip, is a series of n expression-level ratios. Typically, the numerator of each ratio is the expression level of the gene in the varying condition of interest, whereas the denominator is the expression level of the gene in some reference condition. The data from a series of m such experiments may be represented as a gene expression matrix, in which each of the n rows consists of an m-element expression vector for a single gene. The expression measurement is positive if the gene is induced (turned up) with respect to the reference state and negative if it is repressed (turned down).

A PCA analysis of DNA microarray data can consider the genes as variables or the experiments as variables or both. When genes are variables, the analysis creates a set of "principal gene components" that indicate the features of genes that best explain the experimental responses they produce. When experiments are the variables, the analysis creates a set of "principal experiment components" that indicate the features of the experimental conditions that best explain the gene behaviors they elicit. When both experiments and genes are analyzed together, there is a combination of these affects. In Ref. 44, the authors considered the experiments as variables. They applied PCA to the publicly released yeast sporulation dataset (45). They found that most variance ($> 90\%$) in the sporulation dataset is contained in the first two principal components, which allows most information to be visualized in two dimensions.

Application of SVD to Document Indexing. Regular keyword searches approach a document collection with a kind of accountant mentality: A document contains a given word or it does not, with no middle ground. We create a result set by looking through each document in turn for certain keywords and phrases, tossing aside any document that does not contain them, and ordering the rest based on some ranking system.

LSI adds an important step to the document indexing process (46). In addition to recording which keywords a document contains, the method examines the document collection as a whole, to see which other documents contain some of those same words. LSI considers documents that have many words in common to be semantically close, and ones with few words in common to be semantically distant. This simple method correlates surprisingly well with how a human being, looking at content, might classify a document collection. When one searches an LSI-indexed database, the search engine looks at similarity values it has calculated for every content word and returns the documents that it thinks best fit the query. Because two documents may be semantically very close even if they do not share a particular keyword, LSI does not require an exact match to return useful results. Where a plain keyword search will fail if there is no exact match, LSI will often return relevant documents that do not contain the keyword at all. For example, searching for "Saddam Hussain" can return documents on Iraq which has no mention of "Saddam Hussain" in it.

The first step in doing LSI is culling all those extraneous words from a document, leaving only *content words* likely to have semantic meaning. Using this list of content words and documents, we can now generate a term-document matrix. This is a very large grid, with documents listed along the horizontal axis, and content words along the vertical axis. For each content word in our list, we go across the appropriate row and put an "X" in the column for any document where that word appears. If the word does not appear, we leave that column blank. The key step in LSI is decomposing this matrix using SVD. LSI works by projecting this large, multidimensional space down into a smaller number of dimensions. A typical term space might have tens of thousands of dimensions and be projected down into fewer than 150. In this reduction, information is lost, and content words are superimposed on one another. What we are losing is noise from our original term-document matrix, revealing similarities that were latent in the document collection. Similar things become more similar, whereas dissimilar things remain distinct. This reductive mapping is what gives LSI its seemingly intelligent behavior of being able to correlate semantically related terms. We are really exploiting a property of natural language, namely that words with similar meaning tend to occur together.

Uses of Feature Selection Methods

Image Retrieval. Feature selection is applied in Ref. 47 to content-based image retrieval. Recent years have seen a rapid increase of the size and amount of image collections from both civilian and military equipment. However, we cannot access or make use of the information unless it is organized so as to allow efficient browsing, searching, and retrieval. Content-based image retrieval (48) is proposed to efficiently handle large-scale image collections. Instead of being manually annotated by text-based keywords, images would be indexed by their own visual contents (features), such as color, texture, and shape. One of the biggest problems to make content-based image retrieval truly scalable to large-sized image collections is still the curse of dimensionality (5). As suggested in Ref. 48, the dimensionality of the feature space is normally of the order of 10^2. Dimensionality reduction is a promising approach to solve this problem. The image retrieval system proposed in Ref. 47 performs feature selection, and these features are then used to index images for efficient retrieval.

Customer Relationship Management (CRM). A case of feature selection is presented in Ref. 49 for customer relationship management. In this context, each customer means a big revenue and the loss of one will likely trigger a significant segment to defect; it is imperative to have a team of highly experienced experts monitor each customer's intention and movement based on massively collected data. A set of key indicators, proven useful in predicting potential defectors, is used by the CRM team. The problem is that it is difficult to find new indicators describing the dynamically changing business environment among many possible features. The machine-recorded data are simply too enormous for any human expert to browse and obtain any insight from. Feature selection is employed to search for possible new indicators. They are later presented to experts for scrutiny. This approach considerably improves the team's efficiency in finding new changing indicators.

Intrusion Detection. As network-based computer systems play increasingly vital roles in modern society, they have become the targets of our enemies and criminals. The security of a computer system is compromised when an intrusion takes place. Intrusion detection is often used as one way to protect computer systems. In Ref. 50, Lee et al. proposed a systematic data mining framework for analyzing audit data and constructing intrusion detection models. Under this framework, a large amount of audit data is first analyzed using data mining algorithms to obtain the frequent activity patterns. These patterns are then used to guide the selection of system features as well as for the construction of additional temporal and statistical features for another phase of automated learning. Classifiers based on these selected features are then inductively learned using the appropriately formatted audit data. These classifiers can be used as intrusion detection models because they can classify whether an observed system activity is "legitimate" or "intrusive." Feature selection plays an important role in building such classification models for intrusion detection.

Genomic Analysis. Structural and functional data from analysis of the human genome has increased many fold in recent years, presenting enormous opportunities and challenges for data mining. In particular, gene expression microarray is a rapidly maturing technology that provides the opportunity to assay the expression levels of thousands or tens of thousands of genes in a single experiment. These assays provide the input to a wide variety of data mining tasks, including classification and clustering. However, the number of instances in these experiments is often severely limited. In Ref. 51, for example, Xing et al. used a case involving only 38 training data points in a 7130-dimensional space to exemplify the above situation, which is becoming increasingly common in molecular biology applications. In this extreme case of very few observations on a large number of features, Xing et al. investigated the possible use of feature selection on a microarray classification problem. All classifiers tested in the experiments performed significantly better in the reduced feature space than in the full feature space.

Text Categorization. Text categorization is the problem of automatically assigning predefined categories to free text documents (52,53). This problem is of great practical importance given the massive volume of online text available through the World Wide Web, e-mails, and digital libraries. A major characteristic, or difficulty of text categorization problems, is the high dimensionality of the feature space. This dimensionality is prohibitively high for many mining algorithms. Therefore, it is highly desirable to reduce the original feature space without sacrificing categorization accuracy. In Ref. 54, different feature selection methods are evaluated and compared with reduced high-dimensional space in text categorization problems. It is reported that the methods under evaluation can effectively remove 50–90% of the terms while maintaining the categorization accuracy.

CONCLUSIONS AND FUTURE DIRECTIONS

As computers become increasingly powerful, many applications can produce massive data of high dimensionality. Dimensionality reduction is an efficient way of dealing data with high dimensionality. The purpose is to reduce the data so that computational load decreases and patterns of better quality can be extracted by pattern recognition and data mining algorithms. In this article, we described the concepts of feature extraction and feature selection and briefly introduced some representative methods. We then presented in brief some cases of dimensionality reduction to illustrate its application to many problem domains. The need for dimensionality reduction techniques presents new challenges, and novel methods are expected to be developed.

Some research directions in dimensionality reduction are as follows:

1. Feature extraction.
   a. Study how to sparsify the resulting matrix after transformation so that the connection between the extracted features and original features can become evident.
   b. Investigate how to make use of both labeled and unlabeld data in feature extraction.
2. Feature selection.
   a. Unsupervised feature selection remains a challenging task.
   b. Study efficient methods to detect interacting features (may still be slower than sequential forward heuristic methods but faster than exhaustive methods). and
3. Investigate how to maximally take advantage of both feature extraction and selection methods.

FURTHER READING

M. Dash and H. Liu, Feature selection for classification, *Int. J. Intell. Data Analysis*, **1**(3): 131–156, 1997.

H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. Boston, MA: Kluwer Academic, 1998.

BIBLIOGRAPHY

1. A. Y. Ng, Preventing overfitting of crossvalidation data, in *Proc. of Fourteenth International Conference on Machine Learning*, 1997, pp. 245–253.
2. R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton: Princeton University Press, 1961.
3. J. H. Friedman, On Bias, Variance, 0/1 - Loss, and the Curse-of-Dimensionality. *Technical Report*. Stanford, CA: Stanford University, 1996.
4. G.J. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition*. New York: John Wiley and Sons, 1992.
5. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York: Springer, 2001.
6. U. M. Fayyad and R. Uthurusamy, Evolving data mining into solutions for insights, *Commun. Assoc. Comput. Mach.*, **45**(8): 28–31, 2002.
7. J. W. Sammon, A non-linear mapping for data structure analysis, *IEEE Trans. Comput.*, **C-18**(5): 401–409, 1969.
8. J. Kittler, *Feature Selection and Extraction*. Orlando, FL: Academic Press, 1986, pp. 59–83.
9. G. H. Dunteman, *Principal Components Analysis*. Thousand Oaks, CA: Sage, 1989.
10. G. H. John, R. Kohavi, and K. Pfleger, Irrelevant feature and the subset selection problem, in W. W. Cohen and H. Hirsh (eds.), *Machine Learning: Proc. of the Eleventh International Conference*. New Brunswick, NJ: Rutgers University, 1994, pp. 121–129.
11. D. Foley and J. W. Sammon, An optimal set of discriminant vectors, *IEEE Trans. Comput.*, **24**(5): 281–289, 1975.
12. S. T. Roweis and L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, *Science*, **290**: 2323–2326, 2000.
13. J. B. Tenenbaum, V. deSilva, and J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, *Science*, **290**: 2319–2323, 2000.
14. I. T. Joliffe, *Principal Component Analysis*. New York: Springer-Verlag, 1986.
15. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*. Cambridge, MA: Cambridge University Press, 1988.
16. S De Backer, A. Naud, and P. Scheunders, Non-linear dimensionality reduction techniques for unsupervised feature extraction, *Pattern Recogn. Lett.*, **19**: 711–720, 1998.
17. J. B. Kruskal and M. Wish, *Multidimensional Scaling*. Beverly Hills, CA: Sage, 1978.
18. T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen, Som pak: The self-organizing map program package, 1996.
19. M. Dash and H. Liu, Feature selection for classification, *Int. J. Intell. Data Analysis*, **1**(3): 131–156, 1997.
20. H. Liu and H. Motoda, eds, *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Boston, MA: Kluwer Academic, 1998.
21. H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. Boston, MA: Kluwer Academic, 1998.
22. L. Yu and H. Liu, Efficient feature selection via analysis of relevance and redundancy, *J. Machine Learning Res.*, **5**: 1205–1224, 2004.
23. J. C. Schlimmer, Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning, *Proc. of the Tenth International Conference on Machine Learning*, 1993, pp. 284–290.
24. P. M. Narendra and K. Fukunaga, A branch and bound algorithm for feature subset selecting, *IEEE Trans. Comput.*, **C-26**(9): 917–922, 1977.
25. J. Doak, An Evaluation of Feature Selection Methods and Their Application to Computer Security. *Technical Report* Davis, CA: University of California, Department of Computer Science, 1992.
26. H. Liu and R. Setiono, A probabilistic approach to feature selection—a filter solution, in L. Saitta, (ed.), *Proceedings of International Conference on Machine Learning (ICML-96), July 3–6, 1996*. San Francisco, CA: Morgan Kaufmann, 1996, pp. 319–327.
27. I. Kononenko, Estimating attributes: Analysis and extension of RELIEF, in *Proc. of European Conference on Machine Learning (ECML)*, 1994. pp. 171–182.
28. J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.
29. A. N. Mucciardi and E. E. Gose, A comparison of seven techniques for choosing subsets of pattern recognition, *IEEE Trans. Comput.*, **C-20**: 1023–1031, 1971.
30. A. L. Blum and P. Langley, Selection of relevant features and examples in machine learning, *Artif. Intell.*, **97**: 245–271, 1997.
31. M. Dash and H. Liu, Feature selection for clustering, *Proc. of Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2000.
32. J. G. Dy and C. E. Brodley, Visualization and interactive feature selection for unsupervised data, *Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2000, pp. 360–364.
33. J. G. Dy and C. E. Brodley, Feature selection for unsupervised learning, *J. Machine Learning Res.*, **5**: 845–889, 2004.
34. M. Devaney and A. Ram, Efficient feature selection in conceptual clustering, *Proc. of the International Conference on Machine Learning (ICML)*, 1997, pp. 92–97.
35. D. H. Fisher, Knowledge acquisition via incremental conceptual clustering, *Machine Learning*, **2**: 139–172, 1987.
36. L. Talavera, Feature selection as a preprocessing step for hierarchical clustering, in *Proc. of International Conference on Machine Learning (ICML)*, 1999.
37. L. Talavera, Feature selection and incremental learning of probabilistic concept hierarchies, in *Proc. of International Conference on Machine Learning (ICML)*, 2000.
38. S. Vaithyanathan and B. Dom, Model selection in unsupervised learning with applications to document clustering, in *Proc. of the International Conference on Machine Learning (ICML)*, 1999, pp. 433–443.
39. M. Dash, K. Choi, P. Scheuermann, and H. Liu, Feature selection for clustering—A filter solution, *Proc. of IEEE International Conference on Data Mining (ICDM)*, 2002.
40. C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park, Fast algorithms for projected clustering. *Proc. of ACM SIGMOD Conference on Management of Data*, 1999, pp. 61–72.
41. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, *Proc. of ACM SIGMOD Conference on Management of Data*, 1998.
42. A. K. Jain, M. N. Murty, and P. J. Flynn, Data clustering: A review, *ACM Comput. Surveys*, **31**(3): 264–323, 1999.
43. R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York: Wiley Interscience, 2001.
44. S. Raychaudhuri, J. M. Stuart, and R. B. Altman, Principal components analysis to summarize microarray experiments:

Application to sporulation time series, *Proc. of Pacific Symposium on Biocomputing*, 2000, pp. 455–466.

45. S. Chu, J. DeRisi, M. Eisen, J. Mulholland, D. Botstein, P. O. Brown, and I. Herskowitz, The transcriptional program of sporulation in budding yeast, *Science*, **282**: 699–705, 1998.
46. C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, Latent semantic indexing: A probabilistic analysis, *Proc. of the ACM Conference on Principles of Database Systems (PODS)*, 1998.
47. D. L. Swets and J. J. Weng, Efficient content-based image retrieval using automatic feature selection, in *Proc. IEEE International Symposium on Computer Vision*, 1995, pp. 85–90.
48. Y. Rui, T. S. Huang, and S. Chang, Image retrieval: Current techniques, promising directions and open issues, *J. Visual Commun. Image Represent.*, **10**(4): 39–62, 1999.
49. K. S. Ng and H. Liu, Customer retention via data mining, *AI Rev.*, **14**(6): 569–590, 2000.
50. W. Lee, S. J. Stolfo, and K. W. Mok, Adaptive intrusion detection: A data mining approach, *AI Rev.*, **14**(6): 533–567, 2000.
51. E. Xing, M. Jordan, and R. Karp, Feature selection for high-dimensional genomic microarray data, in *Proc. of the Eighteenth International Conference on Machine Learning*, 2001.
52. E. Leopold and J. Kindermann, Text categorization with support vector machines. How to represent texts in input space? *Machine Learning*, **46**: 423–444, 2002.
53. K. Nigam, A. K. Mccallum, S. Thrun, and T. Mitchell, Text classification from labeled and unlabeled documents using EM, *Machine Learning*, **39**: 103–134, 2000.
54. Y Yang and J. O. Pederson, A comparative study on feature selection in text categorization, in *Proc. of Fourteenth International Conference on Machine Learning*, 1997, pp. 412–420.

Manoranjan Dash
Nanyang Technological
University
Singapore
Huan Liu
Arizona State University
Tempe, Arizona

EVOLUTIONARY LEARNING

INTRODUCTION

Over the last three decades, evolutionary computation has become popular in various applications such as optimization and machine learning. Examples include scheduling problems, routing problems, game playing, and robot navigation. Here, several different ways are described in which evolutionary algorithms can be applied to machine learning tasks, which are related to the disciplines of engineering and computer science.

Four main types of evolutionary algorithms (EAs) exist, although in recent years a more unified approach has been attempted (1):

- Genetic algorithms
- Genetic programming (described in the section entitled "Genetic Programming")
- Evolutionary programming
- Evolution strategies

The above algorithms are based on the principles of a Darwinian natural selection system, which includes the following characteristics:

- A population of individuals, i.e. candidate solutions to the underlying problem.
- Genetic operators (e.g., crossover, mutation) that modify the population of solutions dynamically when applied to them.
- A fitness function that is used to measure how good or bad an individual is. Fitter solutions have a higher probability to survive and to contribute to children who inherit some characteristics. Populations evolve and their average fitness (goodness of solutions or effectiveness) is improved over time.

Genetic algorithms (GAs) adopt a string representation of individuals. Frequently, this takes the form of binary strings. Populations of such strings evolve by applying mutation (random modification of a gene within the string) and crossover (exchange the successive genes of two parent individuals, to maximize the fitness of two created offsprings) (2). GAs are general-purpose algorithms because, in theory at least, they can be applied to any domain.

Evolutionary programming (EP) employs mutation only as the means to derive a new N-membered population from an existing population of N members (3). Individuals (possibly mutated) are selected probabilistically to contribute to the next generation. Traditionally, EP was used to evolve finite state machines, but more recently it has been applied to other structures and domains.

An evolution strategy (ES) concentrates on numerical optimization problems. Individuals are encoded as real number vectors. Initially, $(1 + \lambda)$-ES models were used. One parent produces λ children using mutation, and the fittest of the $(1 + \lambda)$ parent-offspring individuals is selected to be the parent in the next generation. ES uses an adaptive mutation operator. According to this technique, each gene i is mutated using a Gaussian (normal) distributed perturbation $G(0, \sigma_i)$ with a mean of zero and a standard deviation of σ_i that evolves over time. Every individual that requires N genes by the problem definition is encoded with a total of $2N$ genes. The additional N genes represent the variances σ_i of the original genes required by the problem. ES can also be used with a $(\mu + \lambda)$ model, in which μ parents are selected to produce λ children and the fittest μ individuals from the $(\mu + \lambda)$ parents/children are selected in a deterministic fashion to participate in the new generation. More recent ES approaches use recombination (crossover) operators as well.

More details about the various aspects of EAs can be found in this encyclopedia.

The following sections describe some common approaches for the application of EAs to machine learning problems.

EVOLVING NEURAL NETWORKS

EAs can be used to train neural networks in the following ways:

- Adapt the weights of a network.
- Find the right topology of a network, in terms of how many hidden units per layer, the number of hidden layers, connectivity of the net (which nodes connect to which others).
- Derive a learning algorithm.

Artificial neural networks are biologically inspired structures that can learn from experience and generalize from trained examples to unseen ones. Artificial neural networks have been applied in many diverse fields.

One of the most popular neural network architectures is the multilayer perceptron (see also the encyclopedia's articles on Perceptrons and Feedforward Neural Nets). Such a network is shown in Fig. 1. A number of nodes (neurons) are arranged in layers, which include one input layer and one output layer with one or more hidden layers in between. Nodes are connected via links called weights. The output o_j of each node j in the network is given by Equation (1), or a similar "sigmoidal" function, where its input *net* is given by (2):

$$f(\text{net}) = \frac{1}{1 + e^{-\text{net}}} \tag{1}$$

$$\text{net}_j = \sum_i w_{ij} o_i \tag{2}$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

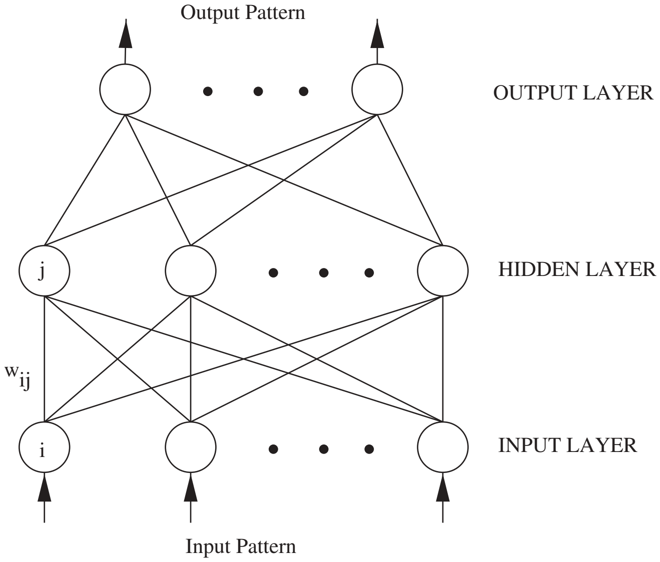


Figure 1. An example of a multilayer feedforward neural network.

The above network is a example of supervised learning, in which several training data exist in the form of inputs and corresponding correct outputs. The multilayer perceptron should learn (i.e., find appropriate values for the weights) the input–output behavior described by the training data, and also should generalize well in unseen data (i.e., data not inside the training set).

A learning algorithm is responsible to adapt the learning parameters (weights) of a neural network. One of the most widely used learning algorithms for the multilayer perceptron is the backpropagation algorithm. In its basic form, backpropagation is based on gradient descent optimization, although more advanced optimization algorithms such as conjugate gradient can be applied as well (4). The aim of backpropagation is to minimize the training error, which is defined as:

$$E = \frac{1}{2}\sum_{p}\sum_{k}(t_{pk} - o_{pk})^2 \tag{3}$$

where k, p belong to the sets of output nodes and training patterns, respectively. The values o_{pk} and t_{pk} are the actual network outputs and the training set target outputs, respectively, of the node k in the output layer for the training pattern p.

Gradient descent algorithms suffer from the local minima problem, which is something that can be avoided if EAs are used to learn the network weights instead.

A typical way to evolve the weights of a neural network is to use GA individuals. These individuals encode all weights of the network in the form of a string that contains real numbers. Thus, a chromosome $a_1a_2\ldots a_n$ represents a network that has n weights, for which the first weight has the value a_1 the second weight has the value a_2, and the n-th weight is equal to a_n.

The initial population of the GA could initialize weights randomly with values from a selected range, for example, between −1 and 1. Mutation can be performed by selecting a hidden or an output node and adding random values (e.g., in the range −1 to 1) to its incoming weights (5). The crossover operator can be defined as the probabilistic selection of two parents who produce a single offspring by copying the incoming weights of a randomly selected non input parent node to the corresponding node of the child. This operation is repeated for all hidden and output nodes of the offspring.

For example, in Fig. 2, the offspring inherits the incoming weights for nodes 4 and 5 from *Parent 2*, whereas the incoming weights for node 3 are coming from *Parent 1*.

The network that will be used in an application is chosen as the network that performs best (based on the training and the generalization error) in the last generation of the GA.

The topology of a feedforward neural network (number of hidden nodes or layers and the connectivity) can also be determined using a GA. For example, a matrix such as the one shown in Fig. 3 describes the connectivity of a neural network (6). One (1) specifies that a connection is present between two nodes and zero (0) stands for no connection. Notice that the matrix is not symmetric because the network is feedforward and no recurrent connections exist, for example, node 1 propagates a signal (value) to node 3, but node 3 does not send a signal to node 1.

To apply a GA for connectivity evolution, the connectivity matrix must be translated to a linear chromosome. This translation can be done by placing all the rows of

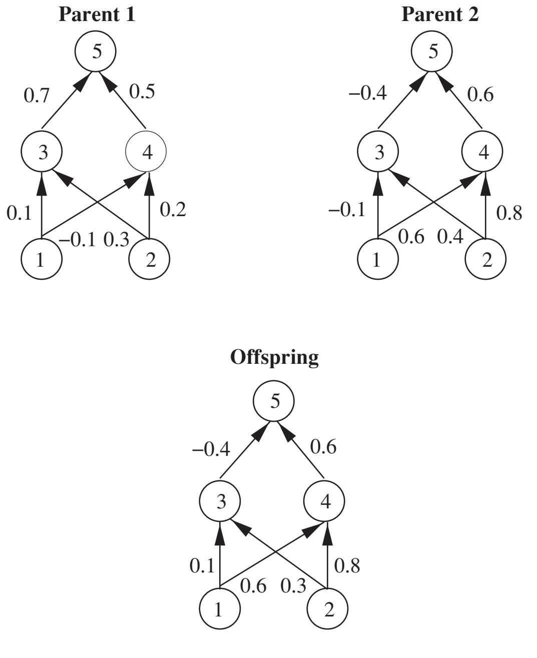


Figure 2. The crossover operator for the evolution of the weights of a multilayer perceptron. For each hidden and output node of the offspring, its incoming weights are coming from a randomly selected parent (all the weights of a single node are inherited from the same parent).

$$\begin{array}{r|ccccc} \textbf{from node:} & \textbf{1} & \textbf{2} & \textbf{3} & \textbf{4} & \textbf{5} \\ \textbf{to node: 1} & 0 & 0 & 0 & 0 & 0 \\ \textbf{2} & 0 & 0 & 0 & 0 & 0 \\ \textbf{3} & 1 & 1 & 0 & 0 & 0 \\ \textbf{4} & 1 & 1 & 0 & 0 & 0 \\ \textbf{5} & 0 & 0 & 1 & 1 & 0 \end{array}$$

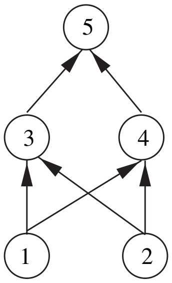


Figure 3. The connectivity of the network on the right is described using the matrix on the left. The matrix can be encoded as the chromosome 0000000000110001100000110.

the matrix next to each other within a string. Thus, the matrix in Fig. 3 corresponds to the chromosome 0000000000110001100000110.

The initial population of the GA is created using randomly selected chromosomes that correspond to networks with different connectivity. Then, the backpropagation algorithm is applied to each net, and the fitness of each trained network is measured according to the error in the training set given by Equation (3) and the error of a test set. The next generation of the GA is evolved by applying mutation and crossover genetic operations. The mutation can be defined as the flipping of bits in an individual. The crossover operator selects a row index k and exchanges the rows between 1 and k in two parents to create two children. The neural network that performs best in the last generation is designated as the best overall solution.

Similar approaches can determine the number of hidden nodes and layers in a network. In such cases, the GA chromosome could be a binary string, that when decoded gives the number of hidden nodes per layer.

Finally, GAs can be used to evolve the learning algorithm, which is the rule that determines the updates of weights after the presentation of training data to the net. One simple approach (see Ref. 6 for a detailed description) is to try to evolve a learning rule whose inputs are the activation function a_i of node i the training signal t_j (correct output of node j), the current output o_j of node j, and the current weight w_{ij} between nodes i, j. The output of the learning rule is the weight update, which, in this simple case, consists of the weighted sum of all the inputs and the pairwise products of the inputs:

$$\begin{aligned} \Delta w_{ij} = \eta(&k_1 w_{ij} + k_2 a_1 + k_3 o_j + k_4 t_j + k_5 w_{ij} a_i \\ &+ k_6 w_{ij} o_j + k_7 w_{ij} t_j + k_8 a_i o_j + k_9 a_i t_j \\ &+ k_{10} o_j t_j) \end{aligned} \tag{4}$$

where η (the learning rate) and the coefficients $k_m, m = 1, 2, \ldots 10$ are evolved using a GA.

Note that in all the approaches described in this section, the chosen network is the network that performs best in the last generation (or across all generations). However, this does not mean that the generalization ability of such a network is the best possible. A different approach is described in Ref. 7, where all (or the top n) individuals (networks) of the last population (generation) are used to determine the final output. For example, in a classification task where each network predicts the class that the input (s) belongs to, all the networks of the last population are considered to be voting toward a class, and the majority of votes is treated as the final prediction. Another possibility for combining the outputs of the individuals in the final generation is to form a weighted sum of their outputs.

A detailed review of various techniques to evolve artificial neural networks can be found in Ref. 8.

REINFORCEMENT LEARNING

In many problems, supervised learning is not possible because no explicit teacher exists [i.e., no pairs of data exist to describe the input(s) and the corresponding correct output(s)] For example, a robot that navigates in an environment is trying to achieve certain goals such as locating targets and avoiding obstacles. The problem can be complicated more by assuming that the environment is unknown or that changes over time.

Although in these situations no information exists about the perfect action that the robot (agent) should take while being in a specific state, the agent can estimate its own performance, as it receives reward or penalty feedback signals by the environment from time to time. The agent should learn to take appropriate actions based on the feedback. Rewards and penalties are referred to geherally as rewards, because a penalty can be seen as a negative reward.

This type of problem is known as *reinforcement learning*. The reinforcement learning problem is the learning of behavior by trial and error without an explicit teacher (9) (see Fig. 4). Some reinforcement learning architectures not only use trial and error for the learning agent, but also use a learned model of the environment (world). Note that some people refer to the learning agent as the *animat*.

Reinforcement learning can be seen commonly as a control task, because at every step the agent must take an appropriate control action so as to maximize the received rewards in the long term (10). Rewards in reinforcement learning are numeric scalar values, and although, this

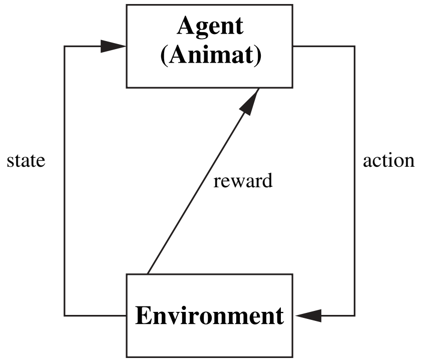


Figure 4. The reinforcement learning problem. The goal is to maximize the total reward in the long term.

seems limiting, a large number of applications exist for which the goals include numeric scalar rewards. For example (9):

- Foraging: rewards are positive for finding food, and negative for energetic motion.
- Classic control problems, such as pole-balancing.
- Recycling robot: rewards are positive for dropping soda cans in the recycling bin, negative for colliding with objects, and are more negative for bumping hard into objects.
- Game playing, such as chess.

When considering the above example applications, it becomes apparent that in many cases the rewards are not continuous; a reward might not be received at every single step, but only from time to time, or in the very end. In games such as chess, the only reward (win or lose) is received in the very end of the match, and the agent does not receive "explicit" indicative signals of how well or bad it functions during the match. In this case, the agent's goal is to learn a policy $\pi: S \rightarrow A$ that maps states to actions. The optimal policy π^* is defined as the policy that achieves the maximum cumulative reward.

To solve the reinforcement learning problem, algorithms operate in either the value or the policy space. Thus, reinforcement learning algorithms search the value or the policy space. Examples of such algorithms include value iteration, policy iteration, Q-learning, and $TD(\lambda)$ (11).

Value function methods attempt to approximate the value function $V^{\pi^*}(s)$, which gives the expected cumulative reward for the optimal policy from any state s. The value function for a policy π (not necessarily the optimal policy) is defined as:

$$V^{\pi}(s_t) = \sum_{i=0}^{h} r_{t+i} \tag{5}$$

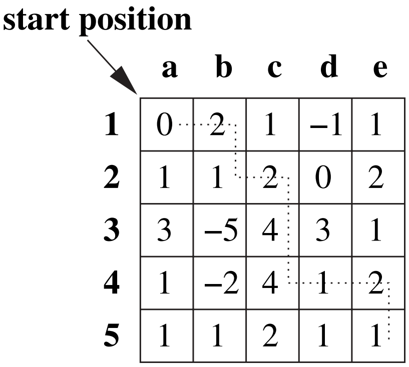
where r_i is the reward received at time step t and assumes that h is a finite value that describes the total number of steps [if $h = \infty$, a discount rate should be included in the above formula as well (11)]. Then the optimal policy is calculated as:

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \quad \forall s \tag{6}$$

Policy-space methods search directly for the optimal policy. EAs can be used to search for the optimal policy in the policy space. An individual in the population represents a policy explicitly.

To illustrate how an EA can be applied to reinforcement learning problems, consider the following task (12):

1. An agent in the grid shown in Fig. 5 moves from state to state by selecting either of the two actions: move right (R) or move down (D).



	a	b	c	d	e
1	0	2	1	−1	1
2	1	1	2	0	2
3	3	−5	4	3	1
4	1	−2	4	1	2
5	1	1	2	1	1

Figure 5. A simple grid-world reinforcement learning problem. The dotted line indicates one of the optimum paths.

2. The initial position (state) of the agent is a1 and it receives the reward indicated when visiting each state.
3. The agent keeps moving until it moves off the grid world.
4. The goal is to learn a policy that returns the highest cumulative reward. For example, a policy that moves the agent in the sequences R, D, R, D, D, R, R, D is an optimal policy (not a unique one in this case) because it achieves the optimal score of 17.

A direct way to apply a GA to the above problem would be to choose a chromosome representation that contains $n = 25$ genes that correspond to each state of the grid. A gene contains a value that corresponds to the action that the chromosome (policy) should take from that state. For example, the chromosome shown in Fig. 6 represents a policy that takes the actions R at state a1, D at state a2, R at state a3, and so on. The initial population of the GA can be created randomly and the standard mutation and crossover genetic operators can be applied to the described representation of a policy to evolve several generations according to a defined fitness.

The fitness of an individual in this problem could be defined naturally as the total reward received when the sequence of actions described by the individual (policy) is applied until the agent moves off the grid. In more complex problems, in which a generalization capability of the agent

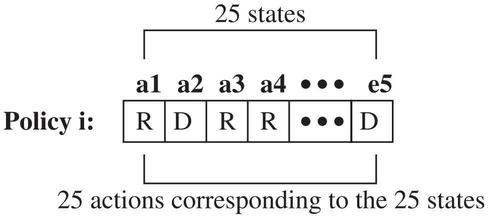


Figure 6. Genetic algorithm policy representation for the problem of Fig. 5.

would be desired (i.e., the agent should be able to receive the maximum reward even when starting from different positions, or placed in a new grid), the fitness should include a number of different test cases, which measure the total efficiency of the individual.

For problems where the state space is very large, the above representation is be feasible. In these situations, the chromosome representation could be modified so each gene represents a "region" of similar states and the gene value corresponds to a common action associated with these states. An extension of that would be to have a condition–action rule in each gene, so that the condition expresses a predicate that matches a set of states (12). The set of states that trigger an action must be defined according to a "similarity" function of states.

COEVOLUTION

Traditional EAs involve individuals in a population that try to adapt in a fixed environment (or a number of static environments). The performance of an individual is measured using an absolute fitness function, and the evolutionary process derives highly fit solutions based on the defined fitness function after a number of generations.

However, many problems exist for which a fitness function is difficult or impossible to define, for example, the task of evolving a game player such as in chess. One could attempt to define the fitness function as the number of wins/losses when the computerized solution plays against a human player. This approach is difficult because a human player might not be available, or, in the case where one is the evolved individuals will learn to play well only with the available human opponent(s), and they will not be able to play well (generalize) against other opponents (unless the human players improve through time) (1).

A second case in which a fitness function cannot be defined is when different individuals perform different subtasks and several individuals must cooperate to solve the whole problem.

A third case involves situations in which it is infeasible to evaluate an individual because of an extremely large number of test cases (1). For example, an individual that represents some software cannot be evaluated against all possible inputs.

The above problem of being unable to define (easily or not at all) a fitness function can be addressed by considering the *coevolution* process found in nature. Natural predators and preys evolve simultaneously, and they adapt to the characteristics of each other. Preys (e.g., organisms) evolve their characteristics to defend against predators (e.g., parasites). Then, predators evolve to circumvent the new defenses of the preys (organisms) and so on.

EAs apply coevolution using populations that compete with each other. For example, let us consider the case of two populations. Initially, both populations are likely to be unfit. Then, individuals of the first population try to adapt to the environment that consists of the second population. To do so, all the individuals of the first population are evaluated against all the individuals of the second population, and a new generation of the first population is created. Then, all the individuals of the second population are evaluated against those of the first one, and a new generation for the second population evolves.

According to this process, for two competing populations *A, B*, the efficiency of an individual in *A* is assessed with a relative fitness function that is defined as its performance when measured against the individuals of the *B*, and vice versa. The fitness is relative, not absolute, because the individuals of the other population continue to change.

In the case of chess, two populations of computerized chess players could be coevolved. Because the fitness of both populations increases, the coevolutionary system will continue to evolve players with higher performance over time.

Another example of coevolution are robots that are able to navigate in different environments (therefore generalize well in unknown environments). One of the populations could consist of the actual robots and the second population could consist of the different environments. Both the robots and the environments evolve over time. To make sure that robots do not "forget" how to behave in older population environments, the fitness of the robot should also take into account its past performance. A limited version of this approach (only one environment with different starting positions) is demonstrated in Ref. 13.

As a special case, some coevolutionary systems could involve a single population. Individuals compete with other individuals within the same population.

CLASSIFIER SYSTEMS

Often, EAs are used in machine learning tasks in the form of *classifier systems*. Broadly speaking, a classifier system is a system whose interaction with the environment creates a chain of "internal" events (classifiers) of the form

$$\textit{if}\ \text{condition}\ \textit{then}\ \text{activate} \tag{7}$$

that triggers a chain of external events implicitly.

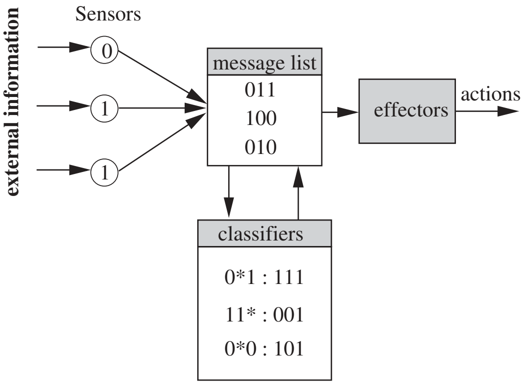


Figure 7. A simple classifier system.

Figure 7 shows a typical simple classifier system. A set of sensors detects information flowing into the system from the external environment. This information depends on the previous outputs of the classifier system. The sensors are able to decode the input to one or more messages, which are posted to the message list.

The messages in the message list can activate the classifiers that have the from of a string rule. If a message matches the first part of a classifier (i.e., the part before the colon : in Fig. 7), the classifier is activated and posts a new message in the message list. For example, given the classifiers in Fig. 7, if a message 011 arrives in the message list, it will activate the classifier 0*1: 111, which will post the new message 111 in the message list. Certain messages can trigger an effector in the effector system (Fig. 7), which will cause an external action to be directed toward the enviroment.

Note that at any instant, more than one classifier can be activated (as a message can be matched by any number of them). In this case, the classifiers compete against each other to designate a winner. The algorithm behind this is known as the *bucket brigade* algorithm (14).

From time to time, new rules (classifiers) must be derived (learned) and this is done by means of a GA. More details about classifier systems can be found in Ref. 2.

GENETIC PROGRAMMING

Genetic programming (GP) (15) aims at the automatic discovery of computer programs. GP is based on a more powerful representation than GAs because every individual in the population is a computer program. The characteristics of computer programs (such as conditionals, loops, assignments, variables, functions) are present in each individual in the population.

GP is an effective evolutionary tool in machine learning applications because it creates computer programs automatically that solve tasks encoded by the fitness function.

The representation of a GP computer program is done through the use of trees. For example, in Fig. 8, the tree shown represents the single program statement

$$\text{if}\, x > y \quad \text{then} \quad a := 10 \tag{8}$$

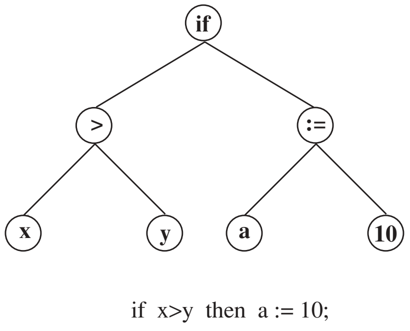


Figure 8. In genetic programming, the representation of a computer program is a tree.

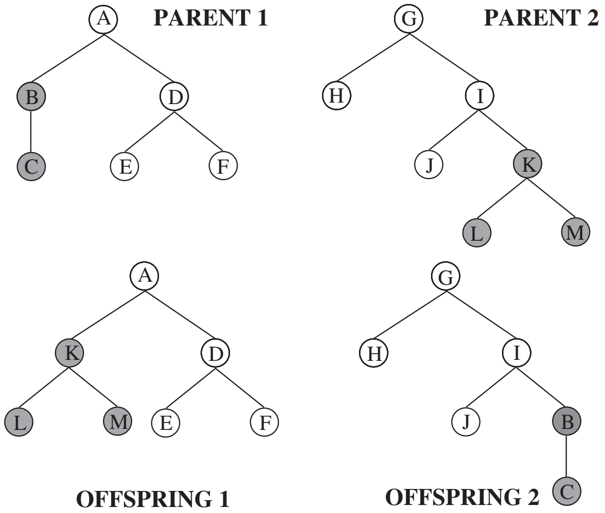


Figure 9. The crossover operator in genetic programming.

Only when the evaluation of the first subtree (i.e., $x > y$) is true is the second subtree of the root node executed (i.e., $a := 10$). This tree representation is equivalent to the parse trees that compilers construct internally to represent a given program.

The application of GP follows the same steps as GAs. The genetic operators used commonly are reproduction, crossover, and mutation.

Reproduction selects an individual from the current generation based on its fitness and copies it to the next generation. Crossover is performed by choosing two points randomly in the probabilistically selected parents and swapping the parent subtrees that are defined by these points (Fig. 9). Note that, unlike GAs, the GP crossover contributes more to different shapes and sizes of the trees (programs) in the population. Thus, sufficient diversity exists in the population and the phenomenon of premature convergence (2) is unlikely to occur. Mutation is performed by choosing a node randomly in the chosen individual and by replacing the subtree that starts at this node with a randomly generated tree.

A tree in the population is composed of functions and terminals chosen from the function set F and the terminal set T, respectively. The function set may include arithmetic or Boolean operations, conditional operators (such as *if-then-else*), iteration functions (e.g., *while*), operators that create statements similar to those found in programming languages, and any functions that are associated closely with the problem domain. The terminal set consists of state variables and inputs and possibly constants (numbers, boolean constants or the random generator R that returns a random value in a prespecified range).

The function and terminal sets should satisfy the *closure* and *sufficiency* properties. The closure property requires that any function in the function set accepts as an argument any terminal from the terminal set and any value returned by any other function. Thus, each function should be closed for any combination of arguments that it may receive. For example, the normal division arithmetic operator always returns a number, unless it is dividing by zero. To be consistent with the closure property, the division function must be redefined to return an “acceptable” value when the denominator is zero. The sufficiency property requires that the sets of functions and terminals that are provided to GP can express a solution to the problem under consideration. A solution cannot be expected to be found if the functions or terminals provided are “incomplete”.

An important issue in the application of GP is the creation of the initial population. Usually, creation is done randomly, after setting a maximum initial depth of each individual (tree). The depth is defined as the length of the longest nonbacktracking path from the root to a leaf. The practical application of GP indicates that the initial population generative method of *ramped half-and-hcdf* gives satisfactory results as it creates enough diversity in the initial population. According to this method, an equal number of individuals for each depth (up to the maximum) is created. Half of the individuals are full trees (trees for which the length of every nonbacktraddng path between the root and any leaf is equal to the prespecified maximum depth) and half are of variable shape (15).

To obtain a solution to a problem, one might need to run GP multiple times, (with different initial conditions, i.e., initial population), because the starting point in a search can lead to different results. This procedure is also appropriate for the application of other EAs.

GP has been applied successfully in a variety of “lear ing” tasks, which include control, game playing, and electrical circuit design.

HYBRID SYSTEMS

EAs can be combined with other techniques to form hybrid systems that can be applied to machine learning tasks. It is common to create hybrid systems of evolutionary techniques and other “soft-computing” approaches such as neural networks and fuzzy logic. Such hybrid systems have discrete operational modules, one of which is using EAs while the other module(s) are based on different nonevolutionary approaches (e.g., neural networks).

Consider the general control problem in which an agent (e.g., a robot) must make decisions and take actions while operating in a completely unknown environment. Assuming that the environment does not change frequently (or it changes slowly enough), the neurogenetic architecture which is an example of a hybrid solution, could be applied to this problem (10).

The neurogenetic architecture consists of two separate steps:

1. Learn a model of the unknown environment using a neural network. This task could be done offline (the agent is not using the model for decisions until the model’s training is complete) or online (depending on the complexity of the environment and the learning rate of the neural network algorithm applied).
2. The model learned is used by a GA, to help the agent in making optimum decisions according to its targets.

At each time step, the agent is using the model of the world to predict the next state of the system given specific candidate input actions. The agent does not actually attempt these actions, but it considers the possible state that it will encounter if it takes several different actions. A GA is used as an optimizer, which ”tells“ the agent which of the candidate actions is optimum. It is only then that the agent applies the optimum action to the real world, and the next step of the agent must be considered.

The application of such an approach can use a single step model prediction (”myopic“ optimization) or multistep prediction of the state of the model at some point in the future. As seen in the section entitled, “Reinforcement Learning”, in the general rainforcement learning problem, the long term behavior should be optimized. However, in certain dynamic systems, even the ”myopic“ approach can result in long-term optimization.

BIBLIOGRAPHY

1. K. A. De Jong, *Evolutionary Computation: A Unified Approach*, Cambridge MA: MIT Press, 2006.
2. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison Wesley, 1989.
3. L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, New York: John Wiley, 1966.
4. J. E. Dennis, and R. B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Englewood Cliffs, NJ: Prentice Hall, 1983.
5. D. J. Montana, and L. D. Davies, Training feedforward networks using genetic algorithms, in *Proceedings of the International Joint Conference on Artificial Intelligence*, San Mateo, CA: Morgan Kaufmann, 1989.
6. M. Mitchell, *An Introduction to Genetic Algorithms*, Cambridge MA: MIT Press, l996.
7. X. Yao, Y. Liu, and P. Darwen, How to make best use of evolutionary learning, in R. Stocker, H. Jenilek, and B. Durnota, (eds.), *Complex Systems From Local Interactions to Global Phenomena*, Amsterdam: IOS Press, 1996.
8. X. Yao, Evolving artificial neural networks. *Proceedings of the IEEE*, **87**(9): 1423–1447, l999.
9. R. Sutton, Reinforcement learning architectures for animats, *Proceedings of the International Workshop on the Simulation of Adaptive Behavior: From Animals to Animats*, Cambridge, MA: MIT Press, 1991, pp. 288–296.
10. D. C. Dracopoulos, *Evolutionary Learning Algorithms for Neural Adaptive Control*, Berlin: Springer Verlag, 1997.
11. R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: MIT Press/Bradford Books, 1998.
12. D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, Evolutionary algorithms for reinforcement learning, *Journal of Artificial Intelligence Research*, **11**: 199–229, 1999.
13. K. Sakamoto, and Q. Zhao, Co-evolving robot controllers that generalize well, *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 2005, pp. 2311–2316.
14. J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor: The University of Michigan Press, 1975.

15. J. R. Koza, *Genetic Programming,* Cambridge, MA: MIT Press, 1982.

FURTHER READING

D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, Piscataway, NJ: IEEE Press, 1995.

Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Berlin: Springer Verlag, 1996.

Dimitris C. Dracopoulos
University of Westminster
London, United Kingdom

EXPERT SYSTEMS

INTRODUCTION

In the early 1970s, substantial interest existed in studying decisions by experts that did not use statistical or other mathematical tools and in determining whether and how such decisions could be modeled in a computer. In particular, researchers were interested in investigating conceptual and symbolic methods appropriate for modeling physician and other expert decision making [e.g., Shortliffe (1)]. From this environment, the notion of an expert system evolved.

The concept of expert systems is almost magical: Simply capture human expertise and put it into a computer program. Rather than worry about a person, a computer program that includes all relevant and appropriate knowledge could be developed and shipped around the world. For example, Rose (2) reported that Southern California Edison (SCE) had an expert whose troubleshooting had helped keep a dam safe. However, SCE was afraid their expert would retire or quit, and they worried that he might "get hit by a bus." As a result, SCE planned on using an expert system to try to "clone" one of their engineers, in a computer program that captured his expertise.

With such hype, it is probably not surprising that, unfortunately, expert systems never lived up to their hype. Bobrow et al. (3) noted that the term "expert" may have created unrealistic expectations about what a computer program could do. Unfortunately, as noted by Business Week (4) "... grandiose promises of problem solving 'expert in a box' proved illusory." However, the term "expert" also generated "commercial" hopes for a discipline that had been academically based [e.g., Shortliffe (1)].

As a result, that same *Business Week* article also noted that expert systems had proliferated rapidly throughout finance in business applications, which was being used for a range of activities such as market analysis to credit evaluation. From the early to mid 1970s to the mid 1980s, expert systems application base seemed almost universal. Since then, expert systems have been applied to just about every conceivable discipline, ranging from chemistry to medicine to business.

The term expert system apparently began to be replaced by the term "knowledge-based system" in the mid 1980s to mid 1990s [e.g., Hayes-Roth (5) and Davis (6)]. The shift began to remove the need for labeling a system with "expert," and reduce the hype, but still would require that the system be "knowledge based." This name shift put less direct pressure on developers to build systems that were equivalent to experts, but it also was sign of a commercial and research shift away from expert systems and an evolution to other forms of problem solving approaches.

Purpose and Scope

The purpose of this article is to review key concepts in expert systems across the lifecycle of expert system development. As a result, we will analyze the choice of the application area for system development, gathering knowledge through so-called knowledge acquisition, choosing a knowledge representation, building in explanation, and verifying and validating the system.

Although it would be easy to focus only on the technical issues, a key finding in the expert systems literature was noted as businesses actually began to implement expert systems [e.g., Barker and O'Connor (7)]: "... To successfully develop and provide ongoing support for expert systems and to integrate them into the fabric of one's business, ... one must attend to the needs of the business and to human resource and organizational issues as well as to technical issues." One of the first developers of expert systems, E. Feigenbaum, was quoted as saying [Lyons (8)], "I'm not interested in theoretical concepts. I like to see my work used in the real world." Accordingly, we will not only consider technical issues, but also some nontechnical organizational and people issues, along with the applications.

Expert systems were a key part of pioneering artificial intelligence efforts to model human behavior. Expert systems have led to substantial additional and emerging research. Accordingly, this article briefly investigates some additional and emerging issues.

Finally, in an article of this type it is inevitable that some key works and important researchers are omitted. As space is limited, some topics that might be addressed are not. The author apologizes in advance for any such omissions.

Outline of this Article

This article proceeds in the following manner. This first section has provided an introduction and statement of purpose and scope. The second section investigates expert systems and human reasoning, while the third section analyzes the structural nature of an "expert system." The fourth section provides some definitions of an expert system. The following two sections analyze some characteristics of expert system applications and investigate some expert system applications. Then, the following five sections trace expert systems through the lifecycle of choosing an application that is likely to work, knowledge acquisition, knowledge representation, explanation, and verification and validation of the system. The final three sections investigate, respectively, expert system strengths and limitations, extensions, and emerging issues, followed by a brief conclusion.

EXPERT SYSTEMS AND HUMAN REASONING

Initially, computer scientists were interested in capturing nonquantitative decision-making models in a computer, and they used expert systems to generate those models

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

[e.g., Shortliffe (1)]. What were some basic assumptions about human reasoning that drove expert systems? Perhaps the initial primary assumptions were:

- Experts know more than nonexperts.
- People use information and knowledge based on their past experience.
- People use heuristics.
- People use specific, a priori rules to solve problems.
- People use focused knowledge to solve problems.

Experts Know More Than Nonexperts

Expert systems assume that experts know more or at least something different than nonexperts in a field. Accordingly, expert systems assume that experts are differentiated from nonexperts by what knowledge they have. As a result, capturing that expert knowledge can potentially change the knowledge of nonexperts.

People use Past Experience

People use their past experience (actions, education, etc.) as the basis of how to solve problems. As a result, system developers interested in building systems to solve problems can consult with people to try to capture that past experience, and they use it to solve problems where that experience could be used.

People use Heuristics

Heuristics are so-called "rules of thumb." Often, past experience is captured and summarized in heuristics. Rather than optimize every decision, people [e.g., Simon (9)] use heuristics that they have found from past experience to drive them toward good, feasible solutions. To solve complex problems, expert systems assume that it is possible to capture those heuristics in a computer program and assemble them for reuse.

People use Rules

Much everyday and business problem solving seems based on rules. For example, when choosing a wine for dinner, simple rules such as "if the dinner includes a red meat then the wine should be red," help guide dinners to the choice of a wine. People use rules to solve problems. As noted by Clancey (10), rules were recognized as a simple and uniform approach to capture heuristic information. Heuristics and other knowledge are captured and kept in a rule-based form. If people use rules, then computer programs could use those same rules to solve problems.

Problem Solving Requires Focused Knowledge

Expert systems researchers [e.g., Feigenbaum (8)] note that one view of human intelligence is that it requires knowledge about *particular* problems and how to solve those particular problems. Accordingly, one approach to mimicking human intelligence is to generate systems that solve only particular problems.

STRUCTURAL NATURE OF EXPERT SYSTEMS

Because it was assumed that human problem solvers used rules and knowledge could be captured as rules, rule bases and their processing were the critical component of expert systems. Accordingly, the structure of a classic expert system was designed to meet those needs. Ultimately, expert systems were composed of five components: data/database, user interface, user, knowledge base/rule base, and an inference engine to facilitate analysis of that knowledge base.

Data

The data used by the system could include computer-generated data, data gathered from a database, and data gathered from the user. For example, computer-generated data might derive from an analysis of financial statement data as part of a program to analyze the financial position of a company. Additional data might be gathered selectively straight from an integrated database. Furthermore, the user might be required to generate some assessment or provide some required data. Typically, initial expert systems required that the user provide key inputs to the system.

User Interface

Because the user typically interacted with the system and provided it with data, the user interface was critical. However, an expert system user interface could take many forms. Typically, the system would display a question, and the user would select one or more answers from a list, as in a multiple choice test. Then, the system would go to another question and ultimately provide a recommended solution. In some cases, the user would need to analyze a picture or a movie clip to answer the questions.

User

In any case, in a classic expert system, the user is a key component to the system, because it is the user who ultimately provides environmental assessments, generates inputs for the system, and as a result, disambiguates the questions provided by the system to gather data from the user. Research has found that different user groups (e.g., novice or expert) given the same interrogation by the system would provide different answers. As a result, generating that user interface and building the system for a particular type of user are critical.

Knowledge Base/Rule Base

The knowledge base typically consisted of a static set of "if . . . then . . ." rules that was used to solve the problem. Rules periodically could be added or removed. However, the knowledge needed for solving the particular problem could be summarized and isolated. In addition, the very knowledge that was used to solve an inquiry also could be used to help explain why a particular decision was made. (Some researchers include explanation facility as its own component.) Accordingly, gathering, explaining, and verifying and validating that knowledge is the focus of the rest of this discussion.

Inference Engines

Inference engines facilitate use of the rule base. Given the necessary information as to the existing conditions provided by the user, inference engines allow processing of a set of rules to arrive at a conclusion by reasoning through the rule base. For example with the system "if a then b," and "if b then c" would allow us to "reason" that a led to b and then to c. As rule-based systems became the norm, the inference engine saved each developer from doing the same thing, and allowed developers to focus on generation of the knowledge base. Developers became referred to as knowledge engineers. Ultimately, data was gathered, combined, and processed with the appropriate knowledge to infer the matching solution.

Expert System Software

Because the expert system components were distinct, software could be designed to allow developers to focus on problem solution rather than building the components themselves. As a result, a wide range of so-called "expert system shells" were generated, for example [e.g., Richter (11)], EMYCIN [from MYCIN, Buchanan and Shortliffe (12)], ART (Automated Reasoning Tool by Inference Corporation), M.4 [http://www.teknowledge.com/, (13)] or Exsys (http://www.exsys.com/).

WHAT IS AN EXPERT SYSTEM?

The term expert system has been applied broadly to several systems apparently for many different reasons. At various points in time, the term "expert system" has implied a type of knowledge representation, a system to perform a particular task, the level of performance of the system.

Rule-Based Knowledge Representation

As noted above, people seemed to use rules to reason to conclusions, and experts were recognized as supplying rules that would be used to guide others through task solution. As a result, most so-called "expert systems" probably were "rule-based systems." Researchers had observed that this type of reasoning apparently was used by people to solve problems. So-called experts seemed to reason this way, so the systems were "expert."

Activity/Task of the System

Another rationale for labeling a system an "expert system," was because the system performed a specific task that human experts did. Experts seem to structured reasoning approaches that could be modeled to help solve various problems (e.g., choosing a wine to go with dinner).

Level of Performance of the System

One perspective was that a system was an "expert system" if it performed a task at the level of a "human expert." For example, Buchanan and Feigenbaum (14) argue that the DENDRAL system functioned at the same level as a human expert.

System Dependence

However, although the system was expert, it was generally still dependent on people for environmental assessments of conditions and corresponding data input. Expert systems were dependent on the user for a range of activities and thus dependent on the user. For example, as noted in Hart et al. (15, p. 590) "... facts recorded in databases often require interpretation." As a result, most self proclaimed expert systems typically provided an interactive consultation that meant the system was still dependent on people.

Definitions

Accordingly, over the years, the term expert systems has been defined several ways, including the following:

- A program that uses available information, heuristics, and inference to suggest solutions to problems in a particular discipline. (answers.com)
- "The term expert systems refer to computer programs that apply substantial knowledge of specific areas of expertise to the problem solving process." [Bobrow et al. (3, p. 880)]
- "... the term expert system originally implied a computer-based consultation system using AI techniques to emulate the decision-making behavior of an expert in a specialized, knowledge-intensive field." [Shortliffe (1, p. 831)]

As a result, we will call a system an expert system when it has the following characteristics:

- a rule-based approach is used to model decision making knowledge, and those rules may include some kind of factor, to capture uncertainty
- interacts with a user from whom it gathers environmental assessments, through an interactive consultation (not always present)
- designed to help facilitate solution of a particular task, typically narrow in scope
- generally performs at the level of an informed analyst

CHARACTERISTICS OF EXPERT SYSTEM APPLICATIONS

Because expert systems related to the ability of a computer program to mimic an expert, expert systems were necessarily about applications and about comparing human experts and systems. Often, the initial goal of expert systems at some level was to show that the system could perform at the same level as a person. But as they put these systems in environments with people, we began to realize several key factors. First, typically, for a rule-base to solve the problem, the problem will need to be structurable. Second, systems may support or replace humans. Third, one of the key reasons that a system might replace a human is the amount of available time to solve the problem, not just knowledge.

Structured versus Unstructured Tasks

Expert systems and their rule-based approaches rely on being able to structure a problem in a formal manner. Rules provided a unifying and simple formalism that could be used to structure a task. Thus, although the problem may not have had sufficient data to be analyzed statistically, or could not be optimized, information still facilitated structuring the problem and knowledge about the problem in a formal manner.

Support versus Replace

Expert systems were often recognized as a vehicle to replace human experts [e.g., Rose (2)]. Many systems apparently were designed initially to replace people. However, in many decision-making situations, the focus was on providing a decision maker with support. For example, as noted by Kneale (16) in a discussion of an accounting system ExperTAX, the expert system is not designed to replace accountants, but instead to enhance and support advice for people.

Available Time

Another important issue in the support versus replace question was how much time was available to make the decision. If a problem needed to be solved in real time, then perhaps support was out the question, particularly if many decisions must be made. Furthermore, even if the system was to support an expert, perhaps it could provide insights and knowledge so the expert did not need to search for information elsewhere.

APPLICATIONS

Because of its focus on modeling and mimicking expertise, ultimately, the field of expert systems has been application oriented. Many applications of expert systems exist in a wide range of areas, including chemical applications, medical diagnosis, mineral exploration, computer configuration, financial applications and taxation applications. Applications have played an important role in expert system technology development. As expert system technologies and approaches were applied to help solve real world problems, new theoretical developments were generated, some of which are discussed below.

Chemical Applications

Some early applications of expert systems took place in this arena [e.g., Buchanan and Feigenbaum (14)]. DENDRAL and Meta-DENDRAL are programs that assist chemists with interpreting data. The DENDRAL programs use a substantial amount of knowledge about mass spectrometry to help with the inference as to what a compound may be. The output from the program is a detailed list with as much detail as the program can provide. Ultimately, Buchanan and Feigenbaum (14) argued that the program had a level of performance equal to a human expert.

Medical Diagnosis Expert Systems

Medicine was one of the first applications of expert systems. By 1984, Clancey and Shortliffe (17) presented a collection of papers that covered the first decade of applications in this domain. Shortliffe (1) briefly summarized some contributions of medical expert systems to medicine. MYCIN was a success at diagnosing infectious diseases. Present illness program (PIP) generated hypotheses about disease in patients with renal disease. INTERNIST-1 was a system designed to assist diagnosis of general internal medicine problems. Since that time, substantial research has occurred in medical expert systems. One of the critical developments associated with medical expert system was using uncertainty on rules [e.g., (18)], which is discussed additionally below.

Geology Advisor

In geology, an expert system was developed to assist in the analysis of drilling site soil samples for oil exploration. PROSPECTOR I and II [(McCammon (19)] were built with over 2,000 rules capturing information about the geologic setting and kinds of rocks and minerals, to help geologists find hidden mineral deposits. PROSPECTOR I [Duda et al. (20) and Hart et al. (15)] was developed along with an alternative representation of uncertainty on rules that garnered substantial attention and is discussed further below.

Computer Configuration

Configuration was one of the first major industrial applications of expert systems. Perhaps the best known configuration expert system was XCON, also known as R1 [e.g., Barker and O'Connor (7)]. XCON was touted as the first expert system in daily production use in an industry setting. At one point in time, XCON was only one of many expert systems in use in at the former computer manufacturer "Digital" to configure hardware and software. As an expert system, XCON was used to validate the customer orders for technical correctness (configurability) and to guide order assembly. Barker and O'Connor (7) also describe many other expert systems that were in use at Digital Equipment Corporation (DEC) during the same time as XCON, including

- XSEL, which was used interactively to assist in the choice of saleable parts for a customer order
- XFL, which was used to diagram a computer room floor layout for the configuration under consideration
- XNET, which was used to design local area networks to select appropriate components

Not surprisingly, these industrial applications had very large knowledge bases. For example, as of September 1988, XCON had over 10,000 rules; XSEL had over 3500 rules; XFL had over 1800 rules; and XNET, a prototype, had roughly 1700 rules.

Taxation Applications at the IRS

Beckman (21) reviewed and summarized the taxation applications expert systems literature, and he provided a focus on applications at the Internal Revenue Service (IRS). Throughout the IRS's involvement in artificial intelligence starting in 1983, the IRS focused on the ability of the technology to help solve real-world problems. As reported by Beckman (21), many expert system projects were developed and tested, including the following. A "tax return issue identification" expert system was designed to help identify individual tax returns with "good audit potential." A "reasonable cause determination" expert system was developed because it was found that the error rate by people was too high. As a result, the system was designed to improve the consistency and quality of so-called "reasonable cause determinations." An "automated under-reporter" expert system that was designed to help tax examiners assess whether individual taxpayers properly reported income.

Auditing and Accounting

The fields of auditing and accounting have generated a substantial literature of applications. The notion behind the development of many such systems was inviting: Auditors and accountants used rules to solve many problems they faced. Expert systems were used to model judgment decisions made by the participants. Brown et al. (22) provide a recent survey of the field.

CHOOSING AN APPLICATION

Two basic perspectives exist on choosing an application to build an expert system. Prerau (23), Bobrow et al. (3), and others have analyzed what characteristics in the domain were important in the selection of a problem around which to build an expert system. Their perspective was one of how well the needs of the domain met the needs of the technology: choose the right problem so that the expert system technology can blossom. Alternatively, Myers et al. (24) and others have viewed it from the business perspective, stressing the need for making sure that the system was in an area that was consistent with the way the company was going to run their business: Make sure the expert system application meets the objectives of the company developing it. In any case, many issues were suggested as conditions that needed to be considered when the domain and expert system application were aligned, including the following issues.

Art and Science

Hart et al. (15, p. 590) note that "Mineral exploration is perhaps as much an art as science, and the state of this art does not admit the construction of models as rigorous and complete, as, say, those of Newtonian mechanics." If a scientific model exists, then a rule-based approach is not needed; the scientific model can be used.

Expertise Issues

Because expert systems are dependent on human experts as a source of their knowledge, experts must work on the project. For example, Prerau (23) notes the importance of having access to an expert from which expertise can be gathered, and the expert must have sufficient time to spend on the project development. Other concerns such as willingness to work on the project also must be considered.

Benefit

In addition, the task should be one that provides enough returns to make it worthwhile. There is no sense in building a system if the value to the builders does not exceed the costs.

Testability

Because the system is to be categorized as an expert system, the system must perform appropriately. This task requires that the results are testable.

KNOWLEDGE ACQUISITION

Early expert systems research was not so much concerned with knowledge acquisition or any other issues, per se. Instead the concern was mostly about the ability of the system to mimic human experts. However, over time as more systems demonstrated the feasibility of capturing expertise, greater attention was paid to knowledge acquisition.

In general, expert system expertise was solicited initially in a team environment, in which programmers and the expert worked hand-in-hand to generate the system. Faculty from multiple disciplines were often coauthors on research describing the resulting systems. However, as the base of applications broadened, it became apparent that interviews with experts, which were designed to try and elicit the appropriate knowledge, was generally the most frequently used approach. Prerau (25) notes the importance of getting step-by-step detail, and that using some form of "quasi-English if-then rules" to document the findings. However, many other creative approaches exist for gathering knowledge from experts, including the following applications.

ExperTAX

One particularly innovative approach was used by Coopers and Lybrand in the development of their expert system "ExperTAX" [e.g., Shpilberg et al. (26) and Kneale (16)] The goal of the project was to try and understand how partners in a professional services firm analyzed tax planning problems. Ultimately, to gather the knowledge necessary to solve a particular tax problem, they had a team of three expert partners behind a curtain. On the other side of the curtain was a beginner, with many documents. While videotaping the process, the partners guided the beginner toward a solution. The camera captured what questions were asked, what documents were needed, and what information was used. Ultimately, each partner spent a total of over 50 hours working on the system.

Problems with Gathering Knowledge from Experts

Various problems have been reported associated with gathering knowledge from experts. First, knowledge is power. As a result, unfortunately, experts do not always have incentives to cooperate. For example, one consultant noted in Orlikowski (27, p. 246) as to why expert consultants at one company were not interested in participating in knowledge acquisition,

> "Power in this firm is your client base and technical ability . . . It is definitely a function of consulting firms. Now if you put all of this in a . . . database, you will lose power. There will be nothing that's privy to you, so you will lose power. It's important that I am selling something that no one else has. When I hear people talk about the importance of sharing expertise in the firm, I say, 'Reality is a nice construct.' "

As a result, it has been suggested that experts may withhold secrets [e.g., (2)].

Second, as noted in Rose (2), experts often do not consciously understand what they do. As a result, any attempt to interview them will not result in the quality or quantity of knowledge that is necessary for a system to work. In an example discussed in Rose (2), SCE had their programmers study dam safety and construction engineering before the knowledge acquisition. Then the programmers met one-on-one with the expert in a windowless conference room. Their first meeting lasted seven hours. They captured all interaction using a tape recorder. Unfortunately, the attempts to build the system ran into difficulties. Early versions of the program indicated problems. Virtually every scenario ended with the recommendation to pack the problem wet area with gravel and keep it under observation. They narrowed the focus to a single dam in an effort to generate sufficient detail and insights. However, even after months of work, the knowledge base had only 20 different rules.

KNOWLEDGE REPRESENTATION

Several forms of knowledge representation exist in artificial intelligence. However, expert systems typically refer to so-called rule-based systems. However, some extensions to deterministic rules have been developed to account for uncertainty and ambiguity.

"If . . .then . . ." Rules

"If . . . then . . ." rules are the primary type of knowledge used in classic expert systems. As noted above those rules are used to capture heuristic reasoning that experts apparently often employ. However, over time researchers began to develop and integrate alternative forms of knowledge representation, such as frame-based or case-based reasoning, into their systems. Systems that included multiple types of knowledge sometimes were referred to as hybrid systems or labeled after a particular type of knowledge representation (e.g., case-based).

Uncertain Knowledge

Unfortunately, not all statements of knowledge are with complete certainty. One approach to capturing uncertainty of knowledge was to use some form of probability on each of the rules. As expert systems were developed, many different approaches were generated, which often depended on the particular application. For example, Buchanan and Shortliffe (12, p. 248) for rules of the sort "if e then h," generated certainty factors (CF) for a medical expert system. MYCIN attributes a "meaning" to different certainty factors [Buchanan and Shortliffe (12, p. 91)]. The larger the weight, the greater the belief in the specific rule. If $CF = 1.0$, then the hypothesis is "known to be correct." If $CF = -1.0$ then that means that the hypothesis ". . . has been effectively disproven." "When $CF = 0$ then there is either no evidence regarding the hypothesis or the supporting evidence is equally balanced by evidence suggesting that the hypothesis is not true."

Duda et al. (20) and Hart et al. (15) developed a different approach for Prospector, which is an expert system designed to aid geological exploration. They used the specification of "if E then H (to degree S,N)." S and N are numeric values that represent the strength of association between E and H. S is called a sufficiency factor, because a large S means that a high probability for E is sufficient to produce a high probability of H; N is called a necessity factor, because a small value of N means that a high probability for E is necessary to produce a high probability of H, where $S = P(E \mid H)/P(E \mid H')$ and $N = P(E' \mid H)/P(E' \mid H')$. S and N are likelihood ratios. This approach was extended to include the reliability of the evidence [e.g., (28)].

In addition to probability-based approaches, additional approaches emerged and found their way into expert systems. For example, fuzzy sets [Zadeh (29)] and Dempster-Shafer belief functions [Shafer (30)] were used to provide alternative approaches.

Interaction of Knowledge Acquisition and Representation

Unfortunately, it does not seem that knowledge acquisition and representation are independent of each other. For example, recent research (31) illustrates that the two are tightly intertwined. An empirical analysis of logically equivalent but different knowledge representations can result in different knowledge being gathered. That is, soliciting knowledge in one knowledge representation can generate knowledge perceived as different than a logically equivalent one. As a result, if the developer wants "if . . . then . . ." rules, then they should use those rules as the form of knowledge in the acquisition process and throughout system development.

EXPLANATION

Researchers developed techniques so that given complex rule bases or other structured forms of knowledge representation systems could analyze the knowledge to find a solution. However, a human user of the system might look at the systems and not understand "why" that particular solution was chosen. As a result, it became important for systems to provide an explanation as to why they chose a particular solution.

Importance of Explanation Facilities

Arnold et al. (32) did an empirical analysis of the use of an explanation facility. They found that novice and expert users employed the explanation capabilities differently. In addition, they found that users were more likely to follow a recommendation if an explanation was given. As a result, explanation is an important strand of expert system research, which includes the following approaches.

Trace Through the Rules

Perhaps the first approach toward generating a system that could provide an explanation for the choice was to generate a trace of the rules. The trace was simply a listing of which rules were executed in generating the solution. Much research on explanation leveraged knowledge and context from the specific application area. Although primitive, this approach still provided more insight into why a decision was made, as compared with probability or optimization approaches.

Model-Based Reasoning

In general, explanation is facilitated by the existence of a model that can be used to illustrate why a question is being asked or why a conclusion was drawn. One model-based domain that has gathered a lot of attention is the financial model of a company that depends on several accounting relationships. This financial model has been investigated by many researchers as a basis of explaining decisions [e.g., (33)].

Dialog-Based Systems

Quilici (34) had an interesting approach to explanation, suggesting that in the long-run expert systems must participate in dialogs with their users. Quilici suggested that providing a trace was not likely to be enough, but instead the system needed to know when and how to convince a user. This task would require that the system understand why its advice was not being accepted.

Explanation as to what Decisions were Made in Building the Program

Swartout (35) argued that as part of explanation, a system needs to explain what its developers did and why. Accordingly, he built XPLAIN to provide the user with insights about decisions made during creation of the program to get insight into the knowledge and facilitate explanation.

VERIFICATION AND VALIDATION

As noted above, one factor that makes a system an expert system, is the level of performance of a system. As a result, perhaps more than any other type of system, verification and validation that some system functions at a particular level of expertise is important in establishing the basic nature of the system. Accordingly, an important set of issues is ensuring that the system developed works appropriately and that the knowledge contained in the system is correct. Assuring those conditions is done using verification and validation.

Verification

Verification is more concerned with the syntactical issues. As noted by O’Keefe et al. (36), verification refers to building the system right. Verification refers to making sure that the technology has been implemented correctly. Accordingly, verification is concerned that the structural nature of the “if. . . then . . .” rules is appropriate. For example, verification is concerned that no loops existed in the rule base (“if a then b” and “if b then a”) or that no rules conflict (e.g., “if a then b,” “if a then c”). Preece and Shinghal (37) examine these structural issues in greater detail. Verification also is concerned that any weights on rules have been performed correctly. For example, O’Leary (38) provides many approaches to help determine whether expert system weights on the rules have been put together appropriately or whether any anomalies should be investigated.

Validation

Validation is concerned more with the semantic issues. As noted by O’Keefe et al. (36) validation refers to building the right system. O’Leary (39) lays out some critical issues regarding validation of expert systems and ties his approach to a structure based on research methods. O’Leary (39) suggests that some of the key functions of validation, all consistent with the nature of expert systems, are

- ascertaining what the system knows, does not know or knows incorrectly
- ascertaining the level of decision making expertise of the system
- analyzes the reliability of the system.

Although O’Leary (39) is concerned with the theory and basic guidelines, he provides (40) several practical methods for expert system validation.

EXPERT SYSTEM STRENGTHS AND LIMITATIONS

Unfortunately, the mere term “expert” has put much pressure that the system performs at an appropriate level. This label is both a strength and a weakness. This section lists some other strengths and limitations of expert systems.

Strengths

Expert systems have provided the ability to solve real problems using the manipulation of syntactic and semantic information, rather than quantified information, providing a major change in the view as to what computer could do. In particular, if the problem being posed to the system is one for which rule based knowledge is effective, then the system is likely to provide a recommended solution.

Furthermore, expert systems can be integrated with other computer-based capabilities. As a result, they can do substantial “pre-analysis” of the data. For example, in the case of financial systems, financial ratios can be computed and analyzed, saving much time and effort.

Limitations

However, some limitations are associated with expert systems. One of the biggest “complaints” against expert systems has been the extent to which they are limited in scope and that the systems do not know their limitations. Classic expert systems have rules that focus only on the problems that it is designed to solve, which results in their limited scope. Generally expert systems do not know when a problem being posed by the user is outside of scope of the system.

Furthermore, as noted by *Business Week* (4), from a practical perspective, expert systems "... require complex and subtle interactions between machines and humans, each teaching and learning from other." Rather than being static, systems and people need to learn and change to accommodate each other.

In addition, Clancey (10) was an early investigator who noted that people, other than the authors of the rules, may have difficulty modifying the rule set. Clancey (10) also had concerns with the basic rule formalism for capturing knowledge. For example, Clancey noted "... the view that expert knowledge can be encoded as a uniform ... set of if/then associations is found to be wanting."

Getting and keeping up-to-date knowledge is another potential limitation. For example, in the area of U.S. taxation, the tax rules change every year. Some rules are new and some rules are no longer valid. Such rule-base changes are not unusual in any setting where technology is involved that must change often more than once a year. For example, imagine developing a system to help someone choose the right mobile phone.

Finally, a primary limitation to expert systems is illustrated by comment from Mike Ditka, a hall of fame American Football player. On a radio interview on Los Angeles Area radio (September 18, 2007), while talking about evaluating football players, he noted "... the intangibles are more important than the tangibles." Viewed from the perspective of expert systems, this quote suggests that although we can capture (tangible) knowledge, that other (intangible) knowledge is available, but not captured, and in many cases that additional knowledge may be the most important.

EXTENSIONS TO EXPERT SYSTEMS AND EMERGING RESEARCH ISSUES

The basic model of the expert system presented to this point is one where knowledge is gathered from a single expert and that knowledge is categorized as "if ... then ..." rules, as a basis for mapping expertise into a computer program. However, some extensions have occurred to that basic model, including the following ideas.

Multiple Experts or Knowledge Bases

Ng and Abramson (41) discussed a medical system named "Pathfinder" that was designed around multiple experts. Rather than having the system designers try to merge knowledge gathered from multiple experts into a single knowledge base, the design concept was to allow the system to put together knowledge from the multiple experts when it needed it.

Knowledge from Data

Gathering knowledge from experts ultimately became known as a "bottle neck." In some cases data was available, so rather than capturing what people said they did, an analysis of the data found what they actually did. Some researchers began to try to get knowledge from data, rather than going through classic interview processes. Ultimately, the focus on generating knowledge from data ended up creating the notion and field of knowledge discovery.

Neural nets also provided a vehicle to capture knowledge about data. Ultimately, neural nets have been used to create rules that are used in expert systems and expert systems have been built to try to explain rules generated from neural networks.

Alternative Forms of Knowledge Representation

As researchers studied reasoning and built systems they found that rules apparently were not the only way that people thought, or the ways that the researchers could represent knowledge. For example, one line of reasoning suggested that people used cases or examples on which to base their reasoning. As another example, researchers built frame-based reasoning systems. Frames allow researchers to capture patterns, that allow heuristic matching, for example, as was done with GRUNDY [Rich (42)]. As a result, case-based reasoning and other forms of knowledge representation helped push researchers to forms of knowledge representation beyond rules in an attempt to match the way that people use knowledge [Hayes (43)].

Alternative Problem Solving Approaches

Knowledge representation not only changed, but also other types of problem solving approaches were used. For example, as noted by Shortliffe (1, p. 831), "The term (expert systems) has subsequently been broadened as the field has been popularized, so that an expert system's roots in artificial intelligence research can no longer be presumed ... any decision support system (is) an expert system if it is designed to give expert level problem specific advice ..."

Expertise

Because expert systems were intent on capturing human expertise in a computer program, a need existed to better understand expertise and what it meant to be an expert. As a result, since the introduction of expert systems, substantial additional research is needed in the concept of expertise, not just how expertise can be mapped into a computer program.

Uncertainty Representation

Generating expert systems for different domains ended up facilitating the development of several approaches for representing uncertainty. However, additional research has focused on moving toward Bayes' Nets and influence diagrams [e.g., Pearl (44)] and moving away from the MYCIN certainty factors and the Prospector likelihood ratios.

The Internet and Connecting Systems

Generally, the expert system wave came before the Internet. As a result, the focus was on systems for a specific computer and not networked computers. As a result, limited research was available on networks of expert systems. However, since the advent of the Internet, expert system concepts were extended to knowledge servers (e.g., Reference 45) and multiple intelligent agents. In addition, technologies such as extensible markup language (XML) are now used to capture information containing rules and data and to communicate it around the world (e.g., xpertrule.com).

Ontologies

Furthermore, developers found that as expert systems grew or were connected and integrated with other systems that more formal variable definition was necessary. Large variable sets needed to be controlled and managed carefully, particularly in multilingual environments. As a result, extending those expert system capabilities led to some work on ontologies.

Embedded Intelligence versus Stand-Alone Systems

Increasingly, rather than highly visible stand alone applications, rule-based intelligence was built into other production applications. Because the systems were not stand alone expert systems, users did not even "see" the embedded expertise: People don't go check on what the expert system has to say—programs now are just more intelligent. For example, fixing spelling errors and grammar errors in Microsoft Word requires a certain amount of intelligence.

Business Rules

As another form of evolution, businesses are interested in so-called "business rules." As might be anticipated, business rules assume that businesses use rules in their interaction with other businesses. Rather than wait for people to make decisions, business rules capture those decision making capabilities. Business rules have virtually all of the same concerns as we saw in expert system rules in terms of knowledge acquisition, knowledge representation, verification and validation, and so on.

CONCLUSION

Expert systems have provided an important starting point for understanding and mimicking human expertise. However, they were only a start. Expert systems focused on heuristic decision making and rules, which are generally manifested in "if-then" rules, possibly employing weights on the rules to capture uncertainty or ambiguity. Expert system provided the foundations on which many other developments have been made.

BIBLIOGRAPHY

1. E. Shortliffe, Medical expert systems, knowledge tools for physicians, *West. J. Med.*, **145**(6): 830–839, 1986.
2. F. Rose, An 'electronic' clone of a skilled engineer is very hard to create, *Wall Street J.*, August **12**: 1988.
3. D. Bobrow, S. Mittal, and M. Stefik, Expert systems: perils and promise, *Commun. ACM*, **29**(9): 880–894, 1986.
4. Business Week, The new rocket science, *Business Week*, November 1992.
5. F. Hayes-Roth, The knowledge-based expert system, *IEEE Comp.*, 11–28, 1984.
6. R. Davis, Knowledge-based systems, *Science*, **231**: 4741, 1986.
7. V. Barker and D. O'Connor, Expert systems for configuration at digital: XCON and Beyond, *Commun. ACM*, March 1989, **32**(3): 98–318, 1989.
8. D. Lyons, Artificial intelligence gets real, *Forbes*, November 30: 1998.
9. H. A. Simon, *Administrative Behavior*, 2nd ed., New York: The Free Press, 1965.
10. W. J. Clancey, *The Epistemology of a Rule-based Expert System*, Stanford CS-81-896, 1981.
11. M. Richter, *AI Tools and Techniques*, Norwood, NJ: Ablex Publishing, 1989.
12. B. Buchanan and E. Shortliffe, *Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project*, Reading, MA: Addison-Wesley, 1984.
13. Cimflex Teknowledge, *M4 User's Guide*, Palo Alto, CA: Cimflex Teknowledge, 1991.
14. B. Buchanan and E. Feigenbaum, *Dentral and Meta-Dentral: Their Applications Dimension*, Heuristic Programming Project Memo, 78–1, February 1978.
15. P. Hart, R. Duda, and M. Einaudi, PROSPECTOR-A computer-based consultation system for mineral exploration, *Mathemat. Geol.*, **10**(5): 1978.
16. D. Kneale, How Coopers & Lybrand put expertise into its computers, *Wall Street J.*, November **14**: 1986.
17. W. J. Clancey and E. H. Shortliffe (eds.), *Readings in Medical Artificial Intelligence: The First Decade*, Reading, MA: Addison-Wesley, 1984
18. P. Szolovits, Uncertainty and decisions in medical informatics, *Methods Informat. Med.*, **34**: 111–134, 1995.
19. R. McCammon, Prospector II: towards a knowledge base for mineral deposits, *Mathemat. Geol.*, **26**(8): 917–937, 1994.
20. R. Duda, J. Gaschnig, and P. Hart, Model design in the prospector consultant system for mineral exploration, in D. Mitchie (ed.), *Expert Systems for the Micro Electronic Age*, Edinburgh: Edinburgh University Press, 1979, pp. 153–167.
21. T. J. Beckman, AI in the IRS, *Proc. of the AI Systems in Government Conference*, 1989.
22. C. Brown, A. A. Baldwin, and A. Sangster, Accounting and auditing, in V. Liebowitz (ed.), *The Handbook of Applied Expert Systems*, Boca Raton, FL: CRC Press, pp. 27-1–27-12, 1998.
23. D. Prerau, Selection of an appropriate domain for an expert system, *AI Mag.*, **6**(2): 26–30, 1985.
24. M. Meyer, A. Detore, S. Siegel, and K. Curley, The strategic use of expert systems for risk management in the insurance industry, *Proc. of the 1990 ACM conf. on Trends and Directions in Expert Systems*, 1990.
25. D. Prerau, Knowledge acquisition in the development of a large expert system, *AI Mag.*, **8**(2): 43–51, 1987.
26. D. Shpilberg, L. Graham, and H. Schatz, Expertax: an expert system for corporate tax accrual and planning, *Expert Systems*, **3**(3): 1986.
27. W. Orlikowski, Learning from notes, *Informat. Soc.*, **9**: 237–250, 1993.
28. D. O'Leary, On the representation and impact of reliability of expert system weights, *Internat. J. Man-Machine Stud.*, **29**: 637–646, 1988.
29. L. Zadeh, Fuzzy sets, *Informat. Control*, **8**: 338–353, 1965.

30. G. Shafer, *A Mathematical Theory of Evidence*, Princeton, Princeton University Press, NJ: 1976.

31. D. O'Leary, Knowledge representation of rules, *Intelli. Syst. Account. Fin. Manage.*, **15**(1-2): 73–84, 2007.

32. V. Arnold, N. Clark, P. Collier, S. Leech, and S. Sutton, The differential use and effect of knowledge based system explanations in novice and expert judgment decisions, *MIS Quarte.*, **30**: 79–97, 2006.

33. W. Hamscher, Explaining financial results, *Internat. J. Intell. Syst. Account., Fin. Managem.*, **3**: 1–20, 1994.

34. A. Quilici, Recognizing and revising unconvincing explanations, *Internat. J. Intell. Sys. Account., Fin. Managem.*, **3**: 21–34, 1994.

35. W. Swartout, XPLAIN: a system for creating and explaining expert consulting programs, *Artifi. Intelli.*, **40**: 353–385, 1989.

36. R. O'Keefe, O. Balci, and E. Smith, Validating expert system performance, *IEEE Expert*, **2**(4): 81–90, 1987.

37. A. Preece and R. Shinghal, Foundation and application of knowledge base verification, *Internat. J. Intell. Sys.*, **9**: 683–702, 1994.

38. D. O'Leary, Verification of uncertain knowledge-based systems, *Managem. Sci.*, **42**: 1663–1675, 1996.

39. D. O'Leary, Validation of expert systems, *Decision Sci.*, **18**(3): 468–486, 1987.

40. D. O'Leary, Methods of validating expert systems, *Interfaces* **18**(6): 72–79, 1988.

41. K. Ng and B. Abramson, Probabilistic multi-knowledge base systems, *J. Appl. Intell.*, **4**(2): 219–236, 1994.

42. E. Rich, User modeling via stereotypes, *Cogni. Sci.* **3**: 329–354, 1989.

43. P. Hayes, The logic of frames, In D. Metzing (ed.), *Frame Conceptions and Text Understanding*, New York: de Gruyter, 1979, pp. 45–61.

44. J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, San Mateo, CA: Morgan Kaufman, 1989.

45. N. Abernethy, J. Wu, M. Hewitt, and R. Altman, Sophia: a flexible web-based knowledge server, *IEEE Intell. Syst.*, **14**: 79–85, 1999.

FURTHER READING

B. Buchanan and E. Shortliffe, *Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project*, Reading, MA: Addison-Wesley, 1984.

F. Hayes-Roth, D. Waterman, and D. Lenat, *Building Expert Systems*, Reading MA: Addison-Wesley, 1983.

J. Liebowitz, *The Handbook of Applied Expert Systems*, Boca Raton, FL: CRC Press, 1997.

S. -H. Liao, Expert system methodologies and applications-a decade review from 1995 to 2004, *Expert Sys. Applicat.*, **28**(1): 93–103, 2005.

Daniel E. O' Leary
University of Southern California
Los Angeles, California

F

FUZZY MODELING FUNDAMENTALS

This article introduces the basic concepts, notation, and basic operations for fuzzy sets that are needed in fuzzy modeling. Because research on fuzzy set theory has been underway for over 30 years now, it is practically impossible to cover all aspects of current developments in this area. Therefore, the main goal of this article is to provide an introduction to and a summary of the basic concepts and operations that are relevant to the study of fuzzy sets. We introduce in this article the definition of linguistic variables and linguistic values and explain how to use them in fuzzy rules, which are an efficient tool for quantitative modeling of words or sentences in a natural or artificial language. By interpreting fuzzy rules as fuzzy relations, we describe different schemes of fuzzy reasoning, in which inference procedures based on the concept of the compositional rule of inference are used to derive conclusions from a set of fuzzy rules and known facts. Fuzzy rules and fuzzy reasoning are the basic components of fuzzy inference systems, which are the most important modeling tool, based on fuzzy set theory.

The "fuzzy inference system" is a popular computing framework based on the concepts of fuzzy set theory, fuzzy if-then rules, and fuzzy reasoning (1). It has found successful applications in a wide variety of fields, such as automatic control, data classification, decision analysis, expert systems, time series prediction, robotics, and pattern recognition (2). Because of its multidisciplinary nature, the fuzzy inference system is known by numerous other names, such as "fuzzy expert system" (3), "fuzzy model" (4), "fuzzy associative memory" (5), and simply "fuzzy system."

The basic structure of a fuzzy inference system consists of three conceptual components: a "rule base," which contains a selection of fuzzy rules; a "data base" (or "dictionary"), which defines the membership functions used in the fuzzy rules; and a "reasoning mechanism," which performs the inference procedure on the rules and given facts to derive a reasonable output or conclusion. In general, we can say that a fuzzy inference system implements a nonlinear mapping from its input space to output space. This mapping is accomplished by several fuzzy if-then rules, each of which describes the local behavior of the mapping. In particular, the antecedent of a rule defines a fuzzy region in the input space, whereas the consequent specifies the output in the fuzzy region.

In what follows, we shall first introduce the basic concepts of fuzzy sets and fuzzy reasoning. Then, we will introduce and compare the three types of fuzzy inference systems that have been employed in various applications. Finally, we will address briefly the features and problems of fuzzy modeling, which is concerned with the construction of fuzzy inference systems for modeling a given target system. In this article, we will assume that all fuzzy sets, fuzzy rules, and operations are of type-1 category, unless otherwise specified.

FUZZY SET THEORY

Let X be a space of objects and x be a generic element of X. A classic set A, $A \subseteq X$, is defined by a collection of elements or objects $x \in X$, such that each x can either belong or not belong to the set A. By defining a "characteristic function" for each element $x \in X$, we can represent a classic set A by a set of order pairs (x,0) or (x,1), which indicates $x \notin A$ or $x \in A$, respectively.

Unlike the aforementioned conventional set, a fuzzy set (6) expresses the degree to which an element belong to a set. Hence, the characteristic function of a fuzzy set is allowed to have values between 0 and 1, which denotes the degree of membership of an element in a given set.

Definition 1. Fuzzy sets and membership functions. If X is a collection of objects denoted generically by x, then a "fuzzy set" A in X is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\} \tag{1}$$

where $\mu_A(x)$ is called "membership function" (MF) for the fuzzy set A. The MF maps each element of X to a membership grade (or membership value) between 0 and 1.

Obviously, the definition of a fuzzy set is a simple extension of the definition of a classic set in which the characteristic function is permitted to have any values between 0 and 1. If the values of the membership function $\mu_A(x)$ is restricted to either 0 or 1, then A is reduced to a classic set and $\mu_A(x)$ is the characteristic function of A. This function can be observed with the following example.

Example 1. Fuzzy set with a discrete universe of discourse X. Let X = {Tijuana, Acapulco, Cancun} be the set of cities one may choose to organize a conference in. The fuzzy set A = "desirable city to organize a conference in" may be described as follows:

$$A = \{(\text{Tijuana}, 0.5), (\text{Acapulco}, 0.7), (\text{Cancun}, 0.9)\}$$

In this case, the universe of discourse X is discrete—in this example, three cities in Mexico. Of course, the membership grades listed above are quite subjective; anyone can come up with three different values according to his or her preference.

Corresponding to the ordinary set operations of union, intersection and complement, fuzzy sets have similar operations, which were initially defined in Zadeh's seminal paper (6). Before introducing these three fuzzy set operations, first we shall define the notion of containment, which plays a central role in both ordinary and fuzzy sets. This definition of containment is a natural extension of the case for ordinary sets.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Definition 2. Containment. The fuzzy set A is "contained" in fuzzy set B (or, equivalently, A is a "subset" of B) if and only if $\mu_A(x) \leq \mu_B(x)$ for all x. Mathematically,

$$A \subseteq B \Leftrightarrow \mu_A(x) \leq \mu_B(x) \tag{2}$$

Definition 3. Union. The "union" of two fuzzy sets A and B is a fuzzy set C, written as C = A∪B or C = A OR B, whose MF is related to those of A and B by

$$\mu_C(x) = \max(\mu_A(x),\ \mu_B(x)) = \mu_A(x) \vee \mu_B(x) \tag{3}$$

Definition 4. Intersection. The "intersection" of two fuzzy sets A and B is a fuzzy set C, written as C = A∩B or C = A AND B, whose MF is related to those of A and B by

$$\mu_C(x) = \min(\mu_A(x),\ \mu_B(x)) = \mu_A(x) \wedge \mu_B(x) \tag{4}$$

Definition 5. Complement or Negation. The "complement" of a fuzzy set A, denoted by A (⌉A, NOT A), is

$$\mu_A(x) = 1 - \mu_A(x) \tag{5}$$

As mentioned earlier, a fuzzy set is completely characterized by its MF. Because most fuzzy sets in use have a universe of discourse X consisting of the real line R, it would be impractical to list all the pairs defining a membership function. A more convenient and concise way to define a MF is to express it as a mathematical formula. First we define several classes of parameterized MFs of one dimension.

Definition 6. Triangular MFs. A "triangular MF" is specified by three parameters {a, b, c} as follows:

$$y = \text{triangle}(x; a, b, c) = \begin{cases} 0, & x \leq a \\ (x-a)/(b-a), & a \leq x \leq b \\ (c-x)/(c-b), & b \leq x \leq c \\ 0, & c \leq x \end{cases} \tag{6}$$

The parameters {a,b,c} (with $a < b < c$) determine the x coordinates of the three corners of the underlying triangular MF. Figure 1 (b) illustrates a triangular MF defined by triangle(x; 10, 20, 40).

Definition 7. Trapezoidal MFs. A "trapezoidal MF" is specified by four parameters {a, b, c, d} as follows:

$$\text{traezoid}(x; a, b, c, d) = \begin{cases} 0, & x \leq a \\ (x-a)/(b-a), & a \leq x \leq b \\ 1, & b \leq x \leq c \\ (d-x)/(d-c), & c \leq x \leq d \\ 0, & d \leq x \end{cases} \tag{7}$$

The parameters {a, b, c, d} (with $a < b \leq c < d$) determine the x coordinates of the four corners of the underlying trapezoidal MF. Figure 1 (b) illustrates a trapezoidal MF defined by trapezoid(x; 10, 20, 40, 75).

Because of their simple formulas and computational efficiency, both triangular MFs and trapezoidal MFs have been used extensively, especially in real-time implementations. However, because the MFs are composed of straight line segments, they are not smooth at the corner points specified by the parameters. In the following, we introduce other types of MFs defined by smooth and nonlinear functions.

Definition 8. Gaussian MFs. A "Gaussian MF" is specified by two parameters {c, σ}:

$$\begin{array}{lcl} & & -1\dfrac{(x-c)^2}{\sigma} \\ \text{gaussian}(x; c, \sigma) & = & e^2 \end{array} \tag{8}$$

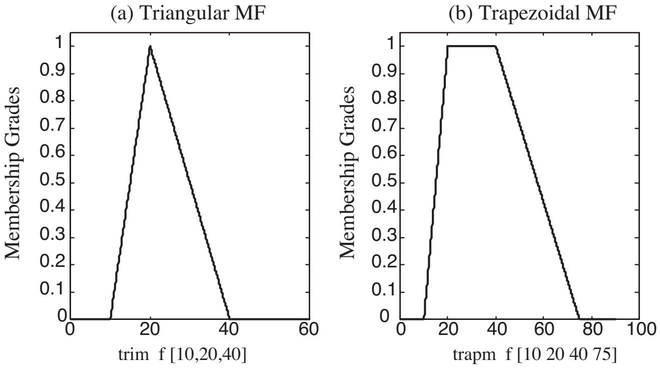
A "Gaussian" MF is determined completely by c and σ; c represents the MFs center and σ determines the MFs width. Figure 2 (a) plots a Gaussian MF defined by gaussian (x; 50, 20).

Definition 9. Generalized bell MFs. A "generalized bell MF" is specified by three parameters {a, b, c}:

$$\text{bell}(x; a,\ b,\ c) = \frac{1}{1 + |(x-c)/a|^{2b}} \tag{9}$$

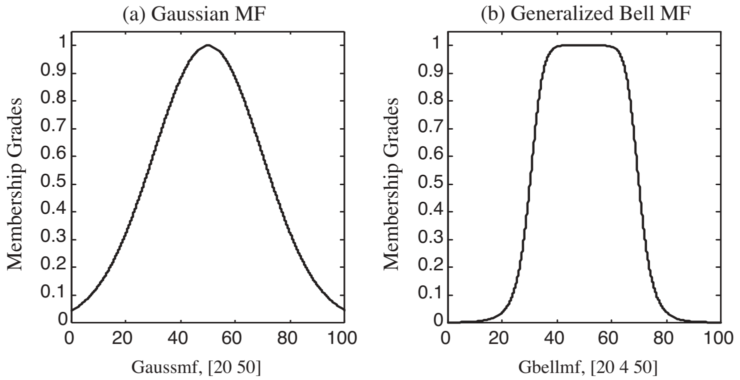
where the parameter b is usually positive. We can note that this MF is a direct generalization of the Cauchy distribution used in probability theory, so it is also referred to as the "Cauchy MF." Figure 2 (b) illustrates a generalized bell MF defined by bell (x; 20, 4, 50).

Although the Gaussian MFs and bell MFs achieve smoothness, they cannot specify asymmetric MFs, which are important in certain applications. Next we define the sigmoidal MF, which is either open left or right.



(a) Triangular MF (b) Trapezoidal MF

Figure 1. Examples of two types of parameterized MFs.



(a) Gaussian MF (b) Generalized Bell MF

Figure 2. Examples of two classes of parameterized continuous MFs.

.

Definition 10. Sigmoidal MFs. A "Sigmoidal MF" is defined by the following equation:

$$\mathrm{sig}(x;\ a,\ c) = \frac{1}{1+\exp[-a(x-c)]} \tag{10}$$

where a controls the slope at the crossover point $x = c$.

Depending on the sign of the parameter "a," a sigmoidal MF is inherently open right or left and thus is appropriate for representing concepts such as "very large" or "very negative." Figure 3 shows two sigmoidal functions $y_1 = \mathrm{sig}(x;\ 1,\ -5)$ and $y_2 = \mathrm{sig}(x;\ -2,\ 5)$.

FUZZY RULES AND FUZZY REASONING

In this section, we introduce the concepts of the extension principle and fuzzy relations, which extend the notions of fuzzy sets introduced previously. Then we give the definition of linguistic variables and linguistic values and show how to use them in fuzzy rules. By interpreting fuzzy rules as fuzzy relations, we describe different schemes of fuzzy reasoning. Fuzzy rules and fuzzy reasoning are the backbone of fuzzy inference systems, which are the most important modeling tool based on fuzzy set theory.

Fuzzy Relations

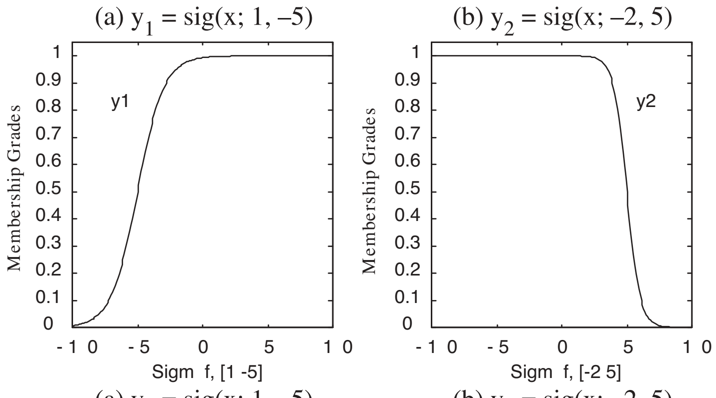
The "extension principle" is a basic concept of fuzzy set theory that provides a general procedure for extending crisp domains of mathematical expressions to fuzzy domains. This procedure generalizes a common one-to-one mapping of a function f to a mapping between fuzzy sets. More specifically, lets assume that f is a function from X to Y and A is a fuzzy set on X defined as

$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \cdots + \mu_A(x_n)/x_n$$

Then the extension principle states that the image of fuzzy set A under the mapping f can be expressed as a fuzzy set B,

$$B = f(A) = \mu_A(x_1)/y_1 + \mu_A(x_2)/y_2 + \cdots + \mu_A(x_n)/y_n$$

where $y_i = f(x_i)$, $i = 1, \ldots, n$. In other words, the fuzzy set B can be defined through the values of f in $x_1, x_2, \ldots, x_n$. If



(a) $y_1 = \mathrm{sig}(x;\ 1, -5)$ (b) $y_2 = \mathrm{sig}(x;\ -2, 5)$

Figure 3. Two sigmoidal functions y_1 and y_2.

f is a many-to-one mapping, then $x_1, x_2 \in X, x_1 \neq x_2$ exists, such that $f(x_1) = f(x_2) = y^*, y^* \in Y$. In this case, the membership grade of B at $y = y^*$ is the maximum of the membership grades of A at $x = x_1$ and $x = x_2$, because $f(x) = y^*$ may result from $x = x_1$ or $x = x_2$. More generally speaking, we have

$$\begin{aligned} \mu_B(y) &= \max \mu_A(x) \\ x &= f^1(y) \end{aligned}$$

A simple example of this concept is shown below.

Example 2. Application of the extension principle to fuzzy sets. Lets suppose we have the following fuzzy set with discrete universe

$$A = 0.2/-2 + 0.5/-1 + 0.7/0 + 0.9/1 + 0.4/2$$

and lets suppose that we have the following mapping

$$y = x^2 + 1$$

After applying the extension principle, we have the following result

$$\begin{aligned} B &= 0.2/5 + 0.5/2 + 0.7/1 + 0.9/2 + 0.4/5 \\ B &= 0.7/1 + (0.2 \vee 0.4)/5 + (0.5 \vee 0.9)/2 \\ B &= 0.7/1 + 0.4/5 + 0.9/2 \end{aligned}$$

where $\vee$ represents "max."

Binary fuzzy relations are fuzzy sets in X × Y which map each element in X × Y to a membership grade between 0 and 1. In particular, unary fuzzy relations are fuzzy sets with one-dimensional MFs; binary fuzzy relations are fuzzy sets with two-dimensional MFs, and so on. Here we will restrict our attention to binary fuzzy relations. A generalization to *n*-ary fuzzy relations is not so difficult.

Definition 11. Binary fuzzy relation. Let X and Y be two universes of discourse. Then

$$\mathcal{R} = \{((x, y), \mu_{\mathcal{R}}(x, y))|(x, y) \in X \times Y \tag{11}$$

is a binary fuzzy relation in X × Y.

Example 3. Binary fuzzy relations. Let X = {1, 2, 3} and Y = {1, 2, 3, 4, 5} and $\mathcal{R}$ = "y is slightly greater than x." The MF of the fuzzy relation $\mathcal{R}$ can be defined (subjectively) as

$$\mu_{\mathcal{R}}(x, y) = \begin{cases} (y - x)/(y + x), & \text{if} \quad y > x \\ 0, & \text{if} \quad y \leq x \end{cases} \tag{12}$$

This fuzzy relation $\mathcal{R}$ can be expressed as a relation matrix in the following form:

$$\mathcal{R} = \begin{pmatrix} 0 & 0.333 & 0.500 & 0.600 & 0.666 \\ 0 & 0 & 0.200 & 0.333 & 0.428 \\ 0 & 0 & 0 & 0.142 & 0.250 \end{pmatrix}$$

where the element at row *i* and column *j* is equal to the membership grade between the *i*th element of X and *j*th element of Y.

Other common examples of binary fuzzy relations are the following:

- x is similar to y (x and y are objects)
- x depends on y (x and y are events)
- If x is big, then y is small (x is an observed reading and y is the corresponding action)

The last example, "If x is A, then y is B," is used repeatedly in fuzzy systems. We will explore fuzzy relations of this type in the following section.

Fuzzy relations in different product spaces can be combined through a composition operation. Different composition operations have been proposed for fuzzy relations; the best known is the max-min composition proposed by Zadeh in 1965 (6).

Definition 12. Max-min composition. Let $\Re_1$ and $\Re_2$ be two fuzzy relations defined on X × Y and Y × Z, respectively. The "max-min composition" of $\Re_1$ and $\Re_2$ is a fuzzy set defined by

$$\begin{aligned} \Re_1 \circ \Re_2 = \{[(x, z), \max, \min(\mu_{\Re_1}(x, y), \\ \mu_{\Re_2}(y, z))]|x \in X,\ y \in Y,\ z \in Z\}y \end{aligned} \tag{13}$$

When $\Re_1$ and $\Re_2$ are expressed as relation matrices, the calculation of the composition $\Re_1 \circ \Re_2$ is almost the same as matrix multiplication, except that × and + are replaced by the "min" and "max" operations, respectively. For this reason, the max-min composition is called the "max-min product."

Fuzzy Rules

As was pointed out by Zadeh in his work on this area (7), conventional techniques for system analysis are intrinsically unsuited for dealing with humanistic systems, whose behavior is strongly influenced by human judgment, perception, and emotions. This finding is a manifestation of what might be called the "principle of incompatibility": "As the complexity of a system increases, our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which precision and significance become almost mutually exclusive characteristics" (7). It was because of this belief that Zadeh proposed the concept of linguistic variables (8,9) as an alternative approach to modeling human thinking.

Definition 13. Linguistic variables. A "Linguistic variable" is characterized by a quintuple (x, T(x), X, G, M) in which x is the name of the variable; T(x) is the "term set" of x-that is, the set of its "linguistic values" or "linguistic terms"; X is the universe of discourse, G is a "syntactic rule" which generates the terms in T(x); and M is a "semantic rule" which associates with each linguistic value A its meaning M(A), where M(A) denotes a fuzzy set in X.

Definition 14. Concentration and dilation of linguistic values. Let A be a linguistic value characterized by a fuzzy set membership function $\mu_A(.)$. Then, A^k is interpreted as a modified version of the original linguistic value expressed as

$$A^k = \int_X [\mu_A(x)]^k/x \tag{14}$$

In particular, the operation of "concentration" is defined as

$$\mathrm{CON}(A) = A^2 \tag{15}$$

whereas that of "dilation" is expressed by

$$\mathrm{DIL}(A) = A^{0.5} \tag{16}$$

Conventionally, we take CON(A) and DIL(A) to be the results of applying the hedges "very" and "more or less," respectively, to the linguistic term A. However, other consistent definitions for these linguistic hedges are possible and well justified for various applications.

Following the definitions given before, we can interpret the negation operator NOT and the connectives AND and OR as

$$\begin{aligned} \mathrm{NOT}(A) &= \rceil A = \int_X [1 - \mu_A(x)]/x \\ A\ \mathrm{AND}\ B &= A \cap B = \int_X [\mu_A(x) \wedge \mu_B(x)]/x \\ A\ \mathrm{OR}\ B &= A \cup B = \int_X [\mu_A(x) \vee \mu_B(x)]/x \end{aligned} \tag{17}$$

respectively, where A and B are two linguistic values whose meanings are defined by $\mu_A(.)$ and $\mu_B(.)$.

Definition 15. Fuzzy If-Then Rules. A "fuzzy if-then rule" (also known as "fuzzy rule," "fuzzy implication," or "fuzzy conditional statement") assumes the form

$$\textbf{if}\ x\ \text{is}\ A\ \textbf{then}\ y\ \text{is}\ B \tag{18}$$

where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y, respectively. Often "x is A" is called "antecedent" or "premise," while "y is B" is called the "consequence" or "conclusion."

Examples of fuzzy if-then rules are widespread in our daily linguistic expressions, such as the following:

- If pressure is high, then volume is small.
- If the road is slippery, then driving is dangerous.
- If the speed is high, then apply the brake a little.

Before we can employ fuzzy if-then rules to model and analyze a system, first we have to formalize what is meant by the expression "if x is A then y is B," which is sometimes abbreviated as A → B. In essence, the expression describes a relation between two variables x and y; this suggests that a fuzzy if-then rule is defined as a binary fuzzy relation R on the product space X ^{x} Y. Generally speaking, there are two ways to interpret the fuzzy rule A → B. If we interpret A → B as A "coupled with" B then

$$R = A \to B = A \times B = \int_{x\ x\ y} \mu_A(x) * \mu_B(y)/(x, y)$$

where * is an operator for intersection (10). On the other hand, if A → B is interpreted as A "entails" B, then it can be written as one of two different formulas:

- Material implication:

$$R = A \to B = \rceil A \cup B \tag{19}$$

- Propositional Calculus:

$$R = A \to B = \rceil A \cup (A \cap B) \tag{20}$$

Although these two formulas are different in appearance, they both reduce to the familiar identity $A \to B \equiv \rceil A \cup B$ when A and B are propositions in the sense of two-valued logic.

Fuzzy reasoning, also known as approximate reasoning, is an inference procedure that derives conclusions from a set of fuzzy if-then rules and known facts. The basic rule of inference in traditional two-valued logic is "modus ponens," according to which we can infer the truth of a proposition B from the truth of A and the implication A → B. This concept is illustrated as follows:

premise 1 (fact):	x is A,
premise 2 (rule):	if x is A then y is B,
consequence (conclusion):	y is B.

However, in much of human reasoning, modus ponens is employed in an approximate manner. This concept is written as:

premise 1 (fact):	x is A′
premise 2 (rule):	if x is A then y is B,
consequence (conclusion):	y is B′

where A′ is close to A and B′ is close to B. When A, B, A′ and B′ are fuzzy sets of appropriate universes, the foregoing inference procedure is called "approximate reasoning" or "fuzzy reasoning"; it is also called "generalized modus ponens" (GMP), because it has modus ponens as a special case.

Definition 16. Fuzzy reasoning. Let A, A′, and B be fuzzy sets of X, X, and Y, respectively. Assume that the fuzzy implication A → B is expressed as a fuzzy relation R on X ^{x} Y. Then the fuzzy set B induced by "x is A′" and the fuzzy rule "if x is A then y is B" is defined by

$$\begin{aligned} \mu_{B'}(y) &= \max_X \min[\mu_{A'}(x), \mu_R(x, y)] \\ &= V_X[\mu_{A'}(x) \wedge \mu_R(x, y)] \end{aligned} \tag{21}$$

Now we can use the inference procedure of fuzzy reasoning to derive conclusions provided that the fuzzy implication A → B is defined as an appropriate binary fuzzy relation.

Single Rule with Single Antecedent. This rule is the simplest case, and the formula is available in Equation (21). Another simplification of the equation yields

$$\begin{aligned}\mu_{B'}(y) &= [\vee_X(\mu_{A'}(x) \wedge \mu_A(x))] \wedge \mu_B(y)\\ &= \omega \wedge \mu_B(y)\end{aligned}$$

In other words, first we find the degree of match ω as the maximum of $\mu_{A'}(x) \wedge \mu_A(x)$, then the MF of the resulting B′ is equal to the MF of B clipped by ω. Intuitively, ω represents a measure of degree of belief for the antecedent part of a rule; this measure gets propagated by the if-then rules and the resulting degree of belief or MF for the consequent part should be no greater than ω.

Multiple Rules with Multiple Antecedents. The process of fuzzy reasoning or approximate reasoning for the general case can be divided into four steps:

1. Degrees of compatibility: Compare the known facts with the antecedents of fuzzy rules to find the degrees of compatibility with respect to each antecedent MF.
2. Firing strength: Combine degrees of compatibility with respect to antecedent MFs in a rule using fuzzy AND or OR operators to form a firing strength that indicates the degree to which the antecedent part of the rule is satisfied.
3. Qualified (induced) consequent MFs: Apply the firing strength to the consequent MF of a rule to generate a qualified consequent MF.
4. Overall output MF: Aggregate all the qualified consequent MFs to obtain an overall output MF.

FUZZY INFERENCE SYSTEMS

In this section, we describe the three types of fuzzy inference systems that have been widely used in the applications. The differences between these three fuzzy inference systems lie in the consequents of their fuzzy rules, and thus their aggregation and defuzzification procedures differ accordingly.

The “Mamdani fuzzy inference system” (10) was proposed as the first attempt to control a steam engine and boiler combination by a set of linguistic control rules obtained from experienced human operators. Figure 4 is an illustration of how a two-rule Mamdani fuzzy inference system derives the overall output z when subjected to two numeric inputs x and y.

In Mamdani’s application, two fuzzy inference systems were used as two controllers to generate the heat input to the boiler and throttle opening of the engine cylinder, respectively, to regulate the steam pressure in the boiler and the speed of the engine. Because the engine and boiler take only numeric values as inputs, a defuzzifier was used to convert a fuzzy set to a numeric value.

Defuzzification

Defuzzification refers to the way a numeric value is extracted from a fuzzy set as a representative value. In general, five methods exist for defuzzifying a fuzzy set A of a universe of discourse Z, as shown in Fig. 5 (Here the fuzzy set A is usually represented by an aggregated output MF, such as C′ in Fig. 4). A brief explanation of each defuzzification strategy follows.

- Centroid of area z_{COA}:

$$z_{COA} = \frac{\int_z \mu_A(z) z dz}{\int_z \mu_A(z) dz} \tag{22}$$

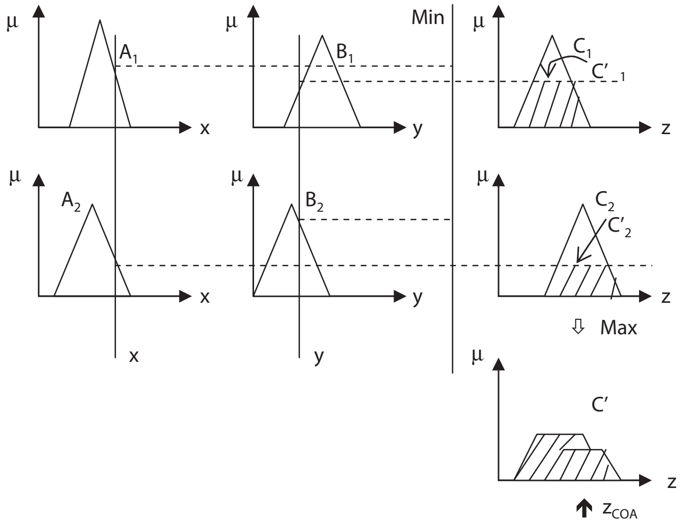


Figure 4. The Mamdani fuzzy inference system using the min and max operators.

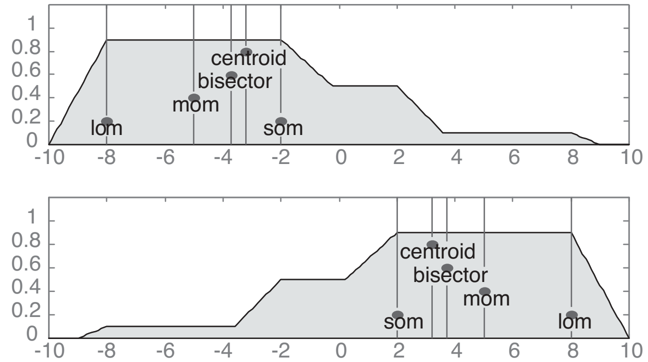


Figure 5. Various defuzzification methods for obtaining a numeric ouput.

where $\mu_A(z)$ is the aggregated output MF. This example is the most widely adopted defuzzification strategy, which is reminiscent of the calculation of expected values of probability distributions.

- Bisector of area z_{BOA}: z_{BOA} satisfies

$$\int_{\alpha}^{Z_{BOA}} \mu_A(z)dz = \int_{z^{BOA}}^{\beta} \mu_A(z)dz \tag{23}$$

where $\alpha = \min\{z|z \in Z\}$ and $\beta = \max\{z|z \in Z\}$.

- Mean of maximum z_{MOM}: z_{MOM} is the average of the maximizing z at which the MF reach a maximum μ^*. Mathematically,

$$z_{MOM} = \frac{\int_{z'} zdz}{\int_{z'} dz} \tag{24}$$

where $z\prime = \{z|\mu_A(z) = \mu*\}$. In particular, if $\mu_A(z)$ has a single maximum at $z = z^*$, then $z_{MOM} = z^*$. Moreover, if $\mu_A(z)$ reaches its maximum whenever $z \in [z_{left}, z_{right}]$ then $z_{MOM} = (z_{left} + z_{right})/2$.

- Smallest of maximum z_{SOM}: z_{SOM} is the minimum (in terms of magnitude) of the maximizing z.
- Largest of maximum z_{LOM}: z_{LOM} is the maximum (in terms of magnitude) of the maximizing z. Because of their obvious bias, z_{SOM} and z_{LOM} are not used as often as the other three defuzzification methods.

The calculation needed to carry out any of these five defuzzification operations is time consuming unless special hardware support is available. Furthermore, these defuzzification operations are not easily subject to rigorous mathematical analysis, so most studies are based on experimental results. This result leads to the propositions of other types of fuzzy inference systems that do not need defuzzification at all; two systems will be described in the following. Other more flexible defuzzification methods can be found in several more recent papers (11,12).

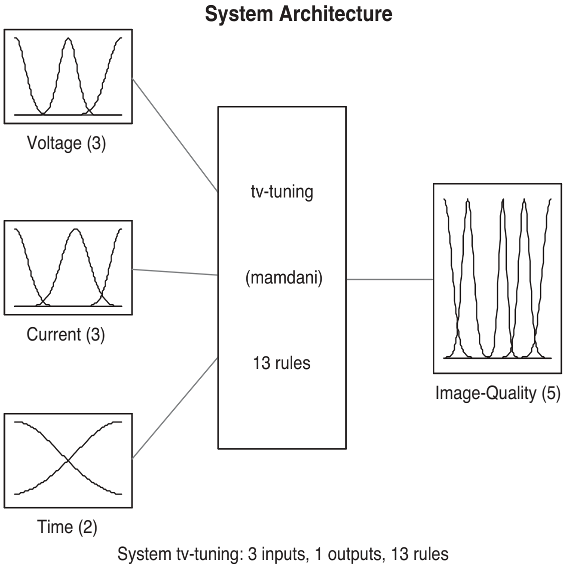
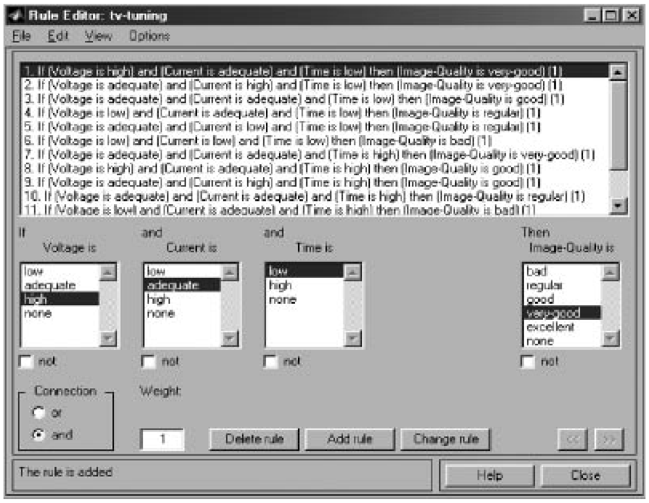


Figure 6. Architecture of the fuzzy system for quality evalution.

We will give a simple example to illustrate the use of the Mamdani fuzzy inference system. We will consider the case of determining the quality of a image produce by a Television as a result of controlling the electrical tuning process based on the input variables: voltage, current, and time (13). Automating the electrical tuning process during the manufacturing of televisions results in increased productivity and reduction of production costs, as well as increasing the quality of the imaging system of the television. The fuzzy model will consist of a set of rules relating these variables, which represent expert knowledge in the electrical tuning process of televisions. In Fig. 6 we show the architecture of the fuzzy system relating the input variables (voltage, current, and time) with the output variable (quality of the image), which was implemented by using the MATLAB Fuzzy Logic Toolbox. We show in Fig. 7 the fuzzy rule base, which was implemented by using the “rule



1. If (Voltage is high) and (Current is adequate) and (Time is low) then (Image-Quality is very-good) (1)
2. If (Voltage is adequate) and (Current is high) and (Time is low) then (Image-Quality is very-good) (1)
3. If (Voltage is adequate) and (Current is adequate) and (Time is low) then (Image-Quality is good) (1)
4. If (Voltage is low) and (Current is adequate) and (Time is low) then (Image-Quality is regular) (1)
5. If (Voltage is adequate) and (Current is low) and (Time is low) then (Image-Quality is regular) (1)
6. If (Voltage is low) and (Current is low) and (Time is low) then (Image-Quality is bad) (1)
7. If (Voltage is adequate) and (Current is adequate) and (Time is high) then (Image-Quality is very-good) (1)
8. If (Voltage is high) and (Current is adequate) and (Time is high) then (Image-Quality is good) (1)
9. If (Voltage is adequate) and (Current is high) and (Time is high) then (Image-Quality is good) (1)
10. If (Voltage is adequate) and (Current is adequate) and (Time is high) then (Image-Quality is regular) (1)
11. If (Voltage is low) and (Current is adequate) and (Time is high) then (Image-Quality is bad) (1)

If Voltage is: high
and Current is: adequate
and Time is: low
Then Image-Quality is: very-good
Connection: ○ or ● and
Weight: 1
The rule is added

Figure 7. Fuzzy rule base for quality evaluation.

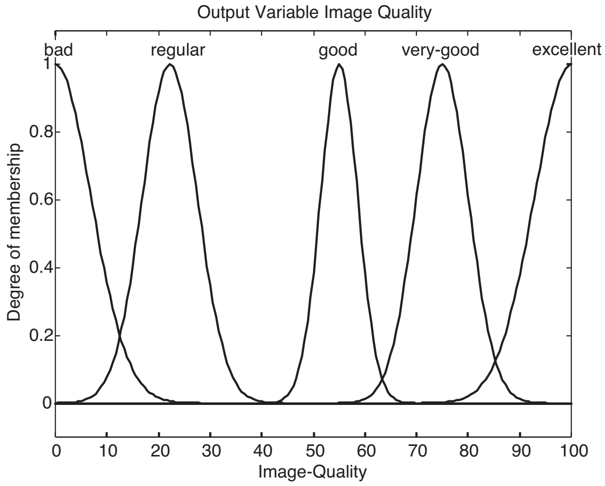
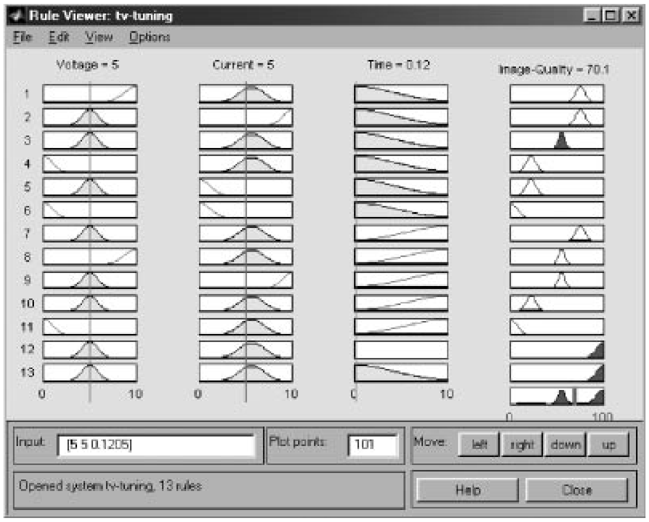


Figure 8. Gaussian membership functions for the output linguistic variable.



Input: [5 5 0.1205]

Plot points: 101

Opened system tv-tuning, 13 rules

Figure 10. Use of the fuzzy rule base with specific values.

editor" of the same toolbox. In Fig. 8 we can appreciate the membership functions for the image-quality variable. We show in Fig. 9 the membership functions for the voltage variable. We also show in Fig. 10 the use of the "rule viewer" of MATLAB to calculate specific values. Finally, in Fig. 11 we show the nonlinear surface for the Mamdani model.

Sugeno Fuzzy Models

The "Sugeno fuzzy model" (also known as the "TSK fuzzy model") was proposed by Takagi, Sugeno, and Kang in an effort to develop a systematic approach to generating fuzzy rules from a given input-output data set (4,14). A typical fuzzy rule in a Sugeno fuzzy model has the form:

$$\textbf{if } \mathrm{x \text{ is } A} \textbf{ and } \mathrm{y \text{ is } B} \textbf{ then } \mathrm{z} = \mathrm{f(x, y)}$$

where A and B are fuzzy sets in the antecedent, whereas z = f(x,y) is a traditional function in the consequent. Usually f(x,y) is a polynomial in the input variables x and y, but it can be any function as long as it can appropriately describe the output of the model within the fuzzy region specified by the antecedent of the rule. When f(x,y) is a first-order polynomial, the resulting fuzzy inference system is called a "first-order Sugeno fuzzy model." When f is constant, we then have a "zero-order Sugeno fuzzy model," which can be viewed either as a special case of the Mamdani inference system, in which each rule's consequent is specified by a fuzzy singleton; or a special case of the Tsukamoto fuzzy model (to be introduced next), in which each rule's consequent is specified by a MF of a step function center at the constant.

Figure 12 shows the fuzzy reasoning procedure for a first-order Sugeno model. Because each rule has a numeric output, the overall output is obtained via "weighted average," thus avoiding the time-consuming process of defuzzification required in a Mamdani model. In practice, the weighted average operator is sometimes replaced with the "weighted sum" operator (that is, $w_1z_1 + w_2z_2$ in Fig. 12) to reduce computation further specially, in the training of a fuzzy inference system. However, this simplification could lead to the loss of MF linguistic meanings unless the sum of firing strengths (that is, Σw_i) is close to unity.

Unlike the Mamdani fuzzy model, the Sugeno fuzzy model cannot follow the compositional rule of inference strictly in its fuzzy reasoning mechanism. This result poses some difficulties when the inputs to a Sugeno fuzzy model are fuzzy. Specifically, we can still employ the matching of fuzzy sets to find the firing strength of each rule. However, the resulting overall output via either weighted average or weighted sum is always crisp; this finding is counterintuitive because a fuzzy model should propagate the fuzziness from inputs to outputs in an appropriate manner. Without the use of the time-consuming defuzzification procedure,

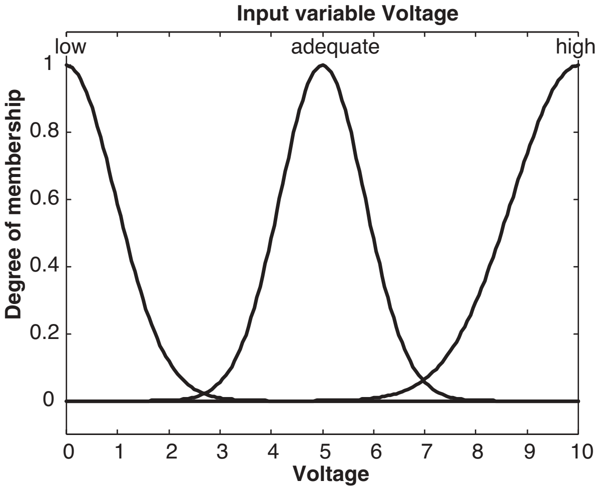


Figure 9. Gaussian membership functions for the voltage linguistic variable.

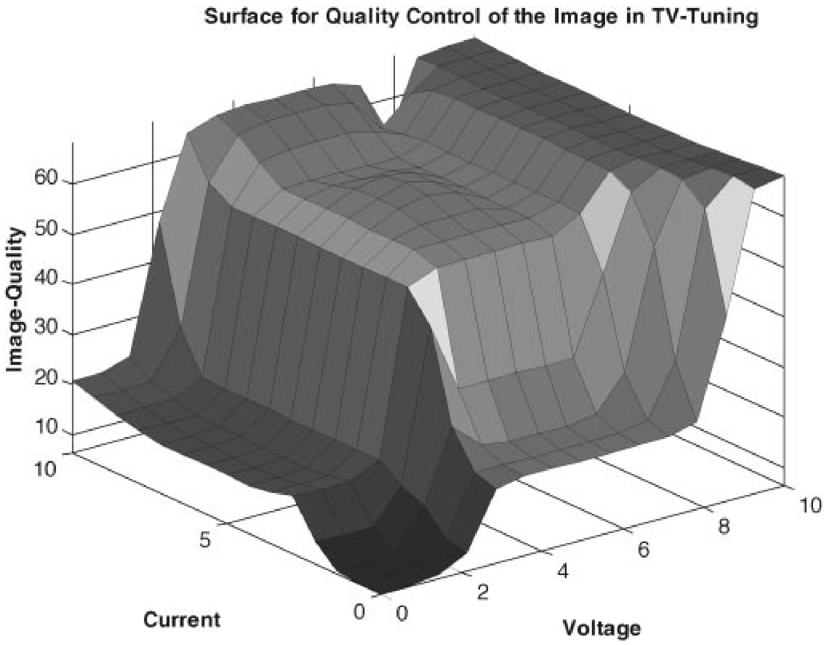


Figure 11. Nonlinear surface of the Mamdani fuzzy model.

the Sugeno fuzzy model is by far the most popular candidate for sample-data-based modeling.

We will give a simple example to illustrate the use of the Sugeno fuzzy inference system. We will consider again the television example (i.e., determining the quality of the images produced by the television depending on the voltage and current of the electrical tuning process). In Fig. 13 we show the architecture of the Sugeno model for this example. We show in Fig. 14 the fuzzy rule base of the Sugeno model. We also show in Fig. 15 the membership functions for the current input variable. In Fig. 16 we show the nonlinear surface of the Sugeno model.

Finally, we show in Fig. 17 the use of the "rule viewer" of the Fuzzy Logic Toolbox of MATLAB. The rule viewer is used when we want to evaluate the output of a fuzzy system using specific values for the input variables. In Fig. 17, for example, we give a voltage of 5 volts, a current intensity of 5 Amperes, and a time of production of 5 seconds, and obtain as a result a quality of 92.2%, which is excellent. Of course, this example only illustratives the potential use of fuzzy logic in this type of application.

Tsukamoto Fuzzy Models. In the "Tsukamoto fuzzy models" (15), the consequent of each fuzzy if-then rule is represented by a fuzzy set with a monotonical MF, as shown in

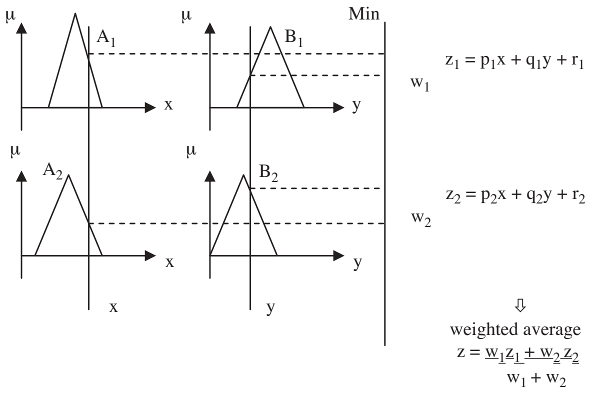

μ
A_1
x
μ
B_1
y
Min
$z_1 = p_1x + q_1y + r_1$
w_1
μ
A_2
x
μ
B_2
y
$z_2 = p_2x + q_2y + r_2$
w_2
x
y
⇩
weighted average
$z = \frac{w_1z_1 + w_2z_2}{w_1 + w_2}$

Figure 12. The Sugeno fuzzy model.

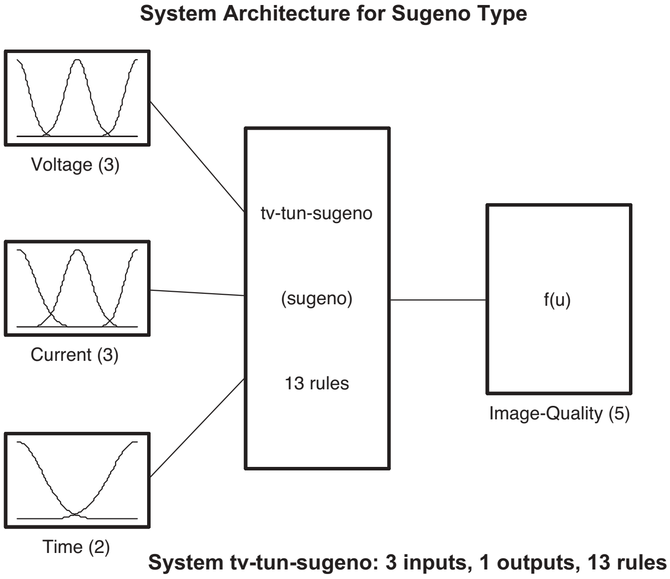
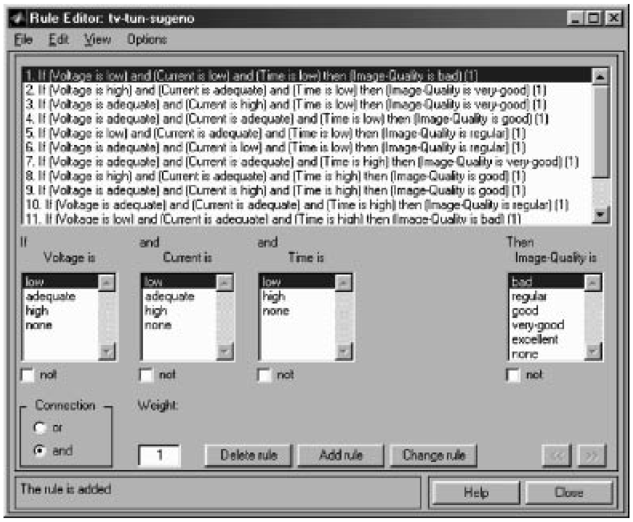

System Architecture for Sugeno Type
Voltage (3)
Current (3)
Time (2)
tv-tun-sugeno
(sugeno)
13 rules
f(u)
Image-Quality (5)
System tv-tun-sugeno: 3 inputs, 1 outputs, 13 rules

Figure 13. Architecture of the Sugeno fuzzy model for quality evaluation.



1. If (Voltage is low) and (Current is low) and (Time is low) then (Image-Quality is bad) (1)
2. If (Voltage is high) and (Current is adequate) and (Time is low) then (Image-Quality is very-good) (1)
3. If (Voltage is adequate) and (Current is high) and (Time is low) then (Image-Quality is very-good) (1)
4. If (Voltage is adequate) and (Current is adequate) and (Time is low) then (Image-Quality is good) (1)
5. If (Voltage is low) and (Current is adequate) and (Time is low) then (Image-Quality is regular) (1)
6. If (Voltage is adequate) and (Current is low) and (Time is low) then (Image-Quality is regular) (1)
7. If (Voltage is adequate) and (Current is adequate) and (Time is high) then (Image-Quality is very-good) (1)
8. If (Voltage is high) and (Current is adequate) and (Time is high) then (Image-Quality is good) (1)
9. If (Voltage is adequate) and (Current is high) and (Time is high) then (Image-Quality is good) (1)
10. If (Voltage is adequate) and (Current is adequate) and (Time is high) then (Image-Quality is regular) (1)
11. If (Voltage is low) and (Current is adequate) and (Time is high) then (Image-Quality is bad) (1)

If Voltage is	and Current is	and Time is	Then Image-Quality is
low	low	low	bad
adequate	adequate	high	regular
high	high	none	good
none	none		very-good
			excellent
			none
□ not	□ not	□ not	□ not

Connection: ○ or ● and

Weight: 1

The rule is added

Figure 14. Fuzzy rule base for quality evaluation using the "rule editor."

Fig. 18. As a result, the inferred output of each rule is defined as a numeric value induced by the rule firing strength. The overall output is taken as the weighted average of each rule's output. Figure 18 illustrates the reasoning procedure for a two-input–two-rule system.

Because each rule infers a numeric output, the Tsukamoto fuzzy model aggregates each rule's output by the method of weighted average and thus avoids the time-consuming process of defuzzification. However, the Tsukamoto fuzzy model is not used often because it is not as transparent as either the Mamdani or Sugeno fuzzy models. Because the reasoning method of the Tsukamoto fuzzy model does not follow strictly the compositional rule of inference, the output is always crisp even when the inputs are fuzzy.

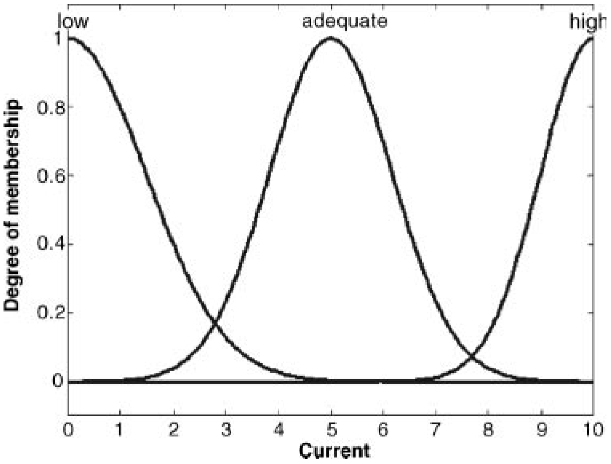


Figure 15. Membership functions for the current linguistic variable.

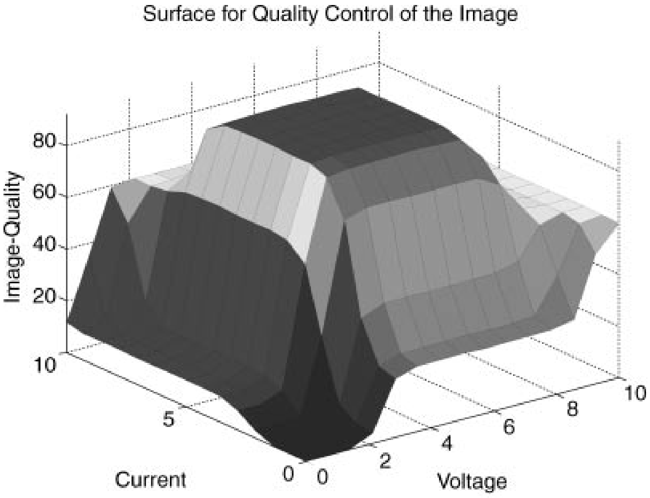
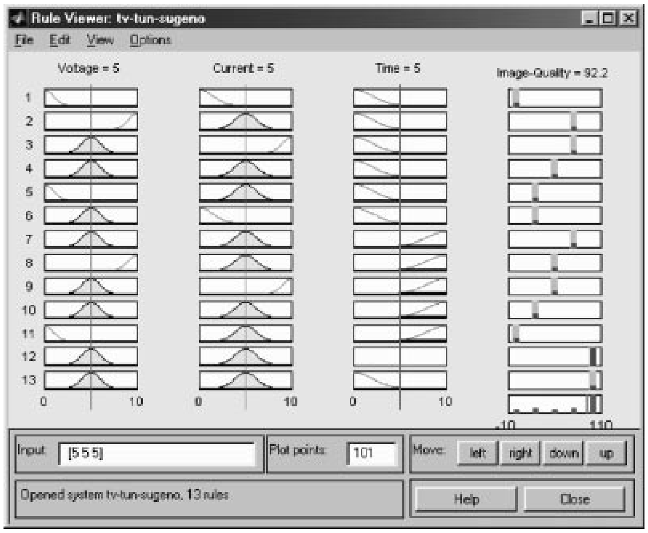


Figure 16. Nonlinear surface for the Sugeno fuzzy model for quality evaluation.

Certain common issues surround all the three fuzzy inference systems introduced previously, such as how to partition an input space and how to construct a fuzzy inference system for a particular application. We will examine these issues in more detail in the following Section.

Input Space Partitioning

Now it should be clear that the main idea of fuzzy inference systems resembles that of "divide and conquer"— the antecedent of a fuzzy rule defines a local fuzzy region, whereas the consequent describes the behavior within the region via various constituents. The consequent constituent can be a



Input: [5 5 5]

Plot points: 101

Opened system tv-tun-sugeno, 13 rules

Figure 17. Application of the rule viewer of MATLAB with specific values.

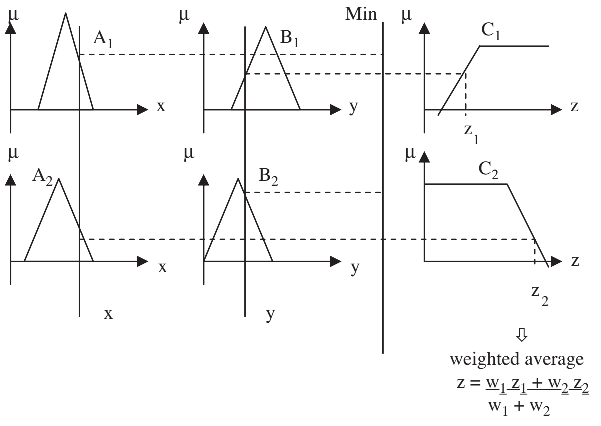


Figure 18. The Tsukamoto fuzzy model.

consequent MF (Mamdani and Tsukamoto fuzzy models), a constant value (zero-order Sugeno model), a linear equation (first-order Sugeno model), or a nonlinear equation (higher-order Sugeno models). Different consequent constituents result in different fuzzy inference systems, but their antecedents are always the same. Therefore, the following discussion of methods of partitioning input spaces to form the antecedents of fuzzy rules is applicable to all three types of fuzzy inference systems.

- Grid partition: This partition method is often chosen in designing a fuzzy controller, which usually involves only several state variables as the inputs to the controller. This partition strategy needs only a small number of MFs for each input. However, it encounters problems when we have many inputs. For instance, a fuzzy model with 12 inputs and 2 MFs on each input would result in $2^{12} = 4096$ fuzzy if-then rules, which is prohibitively large. This problem, which is usually referred to as the "curse of dimensionality," can be alleviated by other partition strategies.
- Tree partition: In this method, each region can be uniquely specified along a corresponding decision tree. The tree partition relieves the problem of an exponential increase in the number of rules. However, more MFs for each input are needed to define these fuzzy regions, and these MFs do not usually bear clear linguistic meanings. In other words, orthogonality holds roughly in X × Y, but not in either X or Y alone.
- Scatter partition: By covering a subset of the whole input space that characterizes a region of possible occurrence of the input vectors, the scatter partition can also limit the number of rules to a reasonable amount. However, the scatter partition is usually dictated by desired input-output data pairs and thus, in general, orthogonality does not hold in X, Y, or X × Y. This result makes it hard to estimate the overall mapping directly from the consequent of each rule's output.

FUZZY MODELING

In general, we design a fuzzy inference system based on the past known behavior of a target system. The fuzzy system is then expected to reproduce the behavior of the target system. For example, if the target system is a human operator in charge of a electrochemical reaction process, then the fuzzy inference system becomes a fuzzy logic controller that can regulate and control the process (Castillo and Melin, 2001) (16). Another example could the human recognition using fuzzy logic (17).

Let us now consider how we might construct a fuzzy inference system for a specific application. Generally speaking, the standard method for constructing a fuzzy inference system, which is a process usually called "fuzzy modeling," has the following features:

- The rule structure of a fuzzy inference system makes it easy to incorporate human expertise about the target system directly into the modeling process. Namely, fuzzy modeling takes advantage of "domain knowledge" that might not be employed easily or directly in other modeling approaches.
- When the input–output data of a target system is available, conventional system identification techniques can be used for fuzzy modeling. In other words, the use of "numerical data" also plays an important role in "fuzzy modeling," just as in other mathematical modeling methods.

Conceptually, fuzzy modeling can be pursued in two stages, which are not totally disjoint. The first stage is the identification of the "surface structure," which includes the following tasks:

1. Select relevant input and output variables.
2. Choose a specific type of fuzzy inference system.
3. Determine the number of linguistic terms associated with each input and output variables.
4. Design a collection of fuzzy if-then rules.

Note that to accomplish the preceding tasks, we rely on our own knowledge (common sense, simple physical laws, and so on) of the target system, information provided by human experts who are familiar with the target system, or simply trial and error.

After the first stage of fuzzy modeling, we obtain a rule base that can more or less describe the behavior of the target system by means of linguistic terms. The meaning of these linguistic terms is determined in the second stage, the identification of "deep structure," which determines the MFs of each linguistic term (and the coefficients of each rule's output in the case that a Sugeno model is used). Specifically, the identification of deep structure includes the following tasks:

1. Choose an appropriate family of parameterized MFs.
2. Interview human experts familiar with the target systems to determine the parameters of the MFs used in the rule base.

3. Refine the parameters of the MFs using regression and optimization techniques.

Tasks 1 and 2 assume the availability of human experts, while task 3 assumes the availability of a desired input–output data set. When a fuzzy inference system is used as a controller for a given plant, then the objective in task 3 should be changed to that of searching for parameters that will generate the best performance of the plant.

SUMMARY

In this article, we have presented the main ideas underlying fuzzy logic and we have only started to point out the many possible applications of this powerful computational theory. We have discussed in some detail fuzzy set theory, fuzzy reasoning and fuzzy inference systems. At the end, we also gave some remarks about fuzzy modeling. In the following chapters, we will show how fuzzy logic techniques (in some cases, in conjunction with other methodologies) can be applied to solve real world complex problems.

BIBLIOGRAPHY

1. J.-S. R. Jang, C.-T. Sun, and E. Mizutani , *Neurofuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Englewood Cliffs, NJ: Prentice-Hall, 1997.
2. M. Jamshidi, Large-Scale Systems: Modelling, Control and Fuzzy Logic, Englewood Cliffs, NJ: Prentice-Hall, 1997.
3. A. Kandel , Fuzzy Expert Systems, Boca Raton FL: CRC Press Inc., 1992.
4. M. Sugeno, and G. T. Kang, Structure identification of fuzzy model, *J. Fuzzy Sets Sys.*, **28**: 15–33, 1988.
5. B. Kosko, *Fuzzy Engineering*, Englewood Cliffs, NJ: Prentice-Hall, 1997.
6. L. A. Zadeh, Fuzzy sets, *J. Information and Control*, **8**: 338–353, 1965.
7. L. A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Trans. Systems, Man and Cybernetics*, **3**: 28–44, 1973.
8. L. A. Zadeh, Similarity relations and fuzzy ordering, *J. Informat. Sci.*, **3**: 177–206, 1971a.
9. L. A. Zadeh, Quantitative fuzzy semantics, *J. Informat. Sci.*, **3**: 159–176, 1971b.
10. E. H. Mamdani, and S. Assilian, An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller, *Internat J. Man-Mach. Studies*, **7**: 1–13, 1975.
11. R. R. Yager, and D. P. Filev, SLIDE: A Simple Adaptive Defuzzification Method, *IEEE Trans. Fuzzy Sys.*, **1**: 69–78, 1993.
12. T. A. Runkler, and M. Glesner, Defuzzification and ranking in the context of membership value semantics, rule modality, and measurement theory, *Proc. of European Congress on Fuzzy and Intelligent Technologies*, 1994.
13. O. Castillo, and P. Melin, *Soft Computing and Fractal Theory for Intelligent Manufacturing*, New York: Springer-Verlag, 2003.
14. T. Takagi, and M. Sugeno, Fuzzy Identification of systems and its applications to modeling and control, *IEEE Trans. on Systems, Man and Cybernet.* **15**: 116–132, 1985.
15. Y. Tsukamoto, An approach to fuzzy reasoning method, in M. M.Gupta, R. K. Ragade, and R. R. Yager, (eds.), *Advanced in Fuzzy Set Theory and Applications*, Amsterdam, the Nehterlands: North-Holland, 1979, pp. 137–149.
16. O. Castillo, and P. Melin, *Soft Computing for Control of Non-Linear Dynamical Systems*, New York: Springer-Verlag, 2001.
17. P. Melin, and O. Castillo, *Hybrid Intelligent Systems for Pattern Recognition*, New York: Springer-Verlag, 2005.

FURTHER READING

L. A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning–1, *J. Informat. Sci.*, **8**: 199–249, 1975.

Patricia Melin
Oscar Castillo
Tijuana Institute of Technology
Tijuana, Mexico

GENETIC ALGORITHMS

FOUNDATIONS OF GENETIC ALGORITHM

The original form of genetic algorithms (GAs) was described by Goldberg (1). GAs are stochastic search techniques based on the mechanism of natural selection and natural genetics. The central theme of research on GA is to keep a balance between exploitation and exploration in its search to the optimal solution for survival in many different environments. Features for self-repair, self-guidance, and reproduction are the rules in biologic systems, whereas they barely exist in the most sophisticated artificial systems. GA has been theoretically and empirically proved to provide a robust search in complex search spaces. Many research papers and dissertations have established the validity of GA approach in function optimization problems and application problems (2–4).

GAs, differing from conventional search techniques, start with an initial set of random solutions called *population*. Each individual in the population is called a *chromosome*, representing a solution to the problem at hand. A chromosome is a string of symbols, usually but not necessarily, a binary bit string. The chromosomes *evolve* through successive iterations, called *generations*. During each generation, the chromosomes are *evaluated*, using some measures of *fitness*. To create the next generation, new chromosomes, called *offspring*, are generated by either merging two chromosomes from the current generation using a *crossover* operator and/or modifying a chromosome using a *mutation* operator. A new generation is formed by selecting some parents, according to the fitness values, and offspring, and rejecting others so as to keep the *population size* constant. Fitter chromosomes have higher probabilities of being selected. After several generations, the algorithms converge to the best chromosome, which hopefully represents the optimum or suboptimal solution to the problem. In general, GAs have five basic components, as summarized by Michalewicz (5):

1. A genetic representation of potential solutions to the problem.
2. A way to create a population (an initial set of potential solutions).
3. An evaluation function rating solutions in terms of their fitness.
4. Genetic operators that alter the genetic composition of offspring (crossover, mutation, selection, etc.).
5. Parameter values that genetic algorithms use (population size, probabilities of applying genetic operators, etc.).

Figure 1 shows a general structure of GA. Let $P(t)$ and $C(t)$ be parents and offspring in the current generation t, and the general implementation structure of GA is described as follows:

Implementation of Genetic Algorithm

For implementing GA, several GA components should be considered. First, *genetic representation* of a solution is decided. Second, the *fitness evaluation* of the solution using the objective functions subjected to constraints is used. Last, *genetic operators* such as crossover operator, mutation operator, and selection methods are applied to the population of GA. These implementation processes are repeated until the predefined generation number is satisfied or the optimal solution is reached. The detailed implementation logics and procedures of GA are suggested in the following subsections.

Genetic Representation. How to encode a solution of the problem into a chromosome is a key issue for GAs. The issue has been investigated from many aspects, such as mapping characters from genotype space to phenotype space when individuals are decoded into solutions and the metamorphosis properties when individuals are manipulated by genetic operators.

Classification of Encodings. In Holland's work, encoding is carried out by using binary strings (6). The binary encoding for function optimization problems is known to have severe drawbacks because of the existence of Hamming cliffs; i.e., pairs of encodings have a large Hamming distance while belonging to points of minimal distance in phenotype space. For example, the pair of 01111111111 and 10000000000 belongs to neighboring points in phenotype space (points of minimal Euclidean distance) but has maximum Hamming distance in genotype space. To cross the Hamming cliff, all bits have to be changed at once. The probability that crossover and mutation will occur to cross it can be very small. In this sense, the binary code does not preserve locality of points in the phenotype space.

For many problems from the computer science and engineering world, it is nearly impossible to represent their solutions with the binary encoding. During the last ten years, various encoding methods have been created for particular problems to have an effective implementation of GAs. According to the symbols used as the alleles of a gene, the encoding methods can be classified as binary encoding, real number encoding, integer/literal permutation encoding, and a general data structure encoding; According to the structure of encodings, the encoding methods also can be classified into the following two types: one-dimensional encoding and multidimensional encoding; According to what kinds of contents are encoded into the encodings, the encoding methods can also be divided as solution only and solution + parameters.

Properties of Encodings. When a new encoding method is given, usually it is necessary to examine whether we can

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

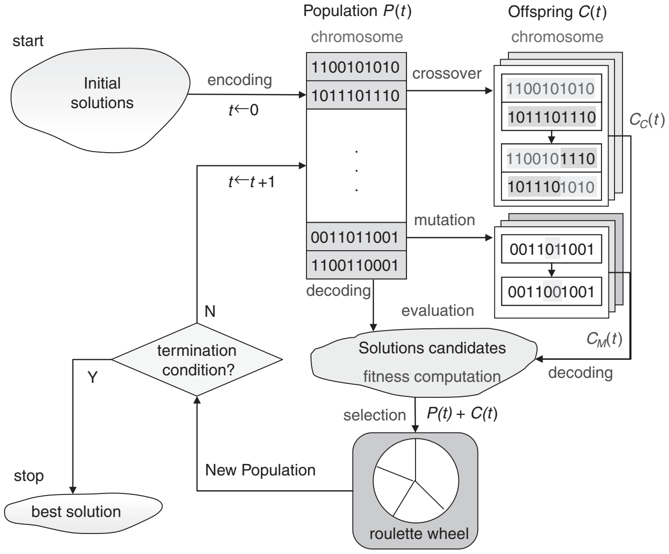


Figure 1. The general structure of genetic algorithms.

build an effective genetic search with the encoding. Several principles have been proposed to evaluate an encoding (7,8):

1. Space: Chromosomes should not require extravagant amounts of memory.
2. Time: The time complexities of evaluating, recombining, and mutating chromosomes should be small.
3. Feasibility: All chromosomes, particularly those generated by simple crossover (i.e., one-cut point crossover) and mutation, should represent feasible solutions.
4. Uniqueness: The mapping from chromosomes to solutions (decoding) may belong to one of the following three cases: 1-to-1 mapping, n-to-1 mapping, and 1-to-n mapping. The 1-to-1 mapping is the best one among three cases, and 1-to-n mapping is the most undesired one.
5. Heritability: Offspring of simple crossover (i.e., one-cut point crossover) should represent solutions that combine substructures of their parental solutions.
6. Locality: A mutated chromosome should usually represent a solution similar to that of its parent.

Initialization. In general, two ways exist to generate the initial population, heuristic initialization and random initialization, by using an encoding procedure satisfying system constraints and/or a boundary condition. Although the mean fitness of the heuristic initialization is already high so that it may help GAs to find solutions faster. Unfortunately, in most large-scale problems, for example, network design problems, it may just explore a small part of the solution space, and it is difficult to find global optimal solutions because of the lack of diversity in the population.

Fitness Evaluation. Fitness evaluation is to check the solution value of the objective function subjected to constraints by using a decoding procedure. In general, the objective function provides the mechanism evaluating each individual. However, its range of values varies from problem to problem. To maintain uniformity over various problem domains, we may use the fitness function to normalize the objective function to a range of 0 to 1. The normalized value of the objective function is the fitness of the individual, and the selection mechanism uses it to evaluate the individuals of the population.

When GAs are used to search, the population undergoes evolution with fitness and forms a new population. At that time, in each generation, relatively good solutions are reproduced and relatively bad solutions are killed so that the offspring composed of the good solutions are reproduced. To distinguish between the solutions, an evaluation function (also called fitness function) plays an important role in the environment, and scaling mechanisms are also necessary to be applied in objective function for being fitness function.

Genetic Operators. When GAs are used, both the search direction to optimal solution and the search speed should be considered as an important factor, in order to keep a balance between exploration and exploitation in search space. In general, the exploitation of the accumulated information resulting from a GA search is done by the selection mechanism, whereas the exploration to new regions of the search space is accounted for by genetic operators.

The genetic operators mimic the process of heredity of genes to create new offspring at each generation. The operators are used to alter the genetic composition of individuals during representation. In essence, the operators perform a random search and cannot guarantee to yield an improved offspring. Three common genetic operators exist: crossover, mutation, and selection.

Crossover. Crossover is the main genetic operator. It operates on two chromosomes at a time and generates offspring by combining both chromosomes' features. A simple way to achieve crossover would be to choose a random cut-point and to generate the offspring by combining the segment of one parent to the left of the cut-point with the segment of the other parent to the right of the cut-point (e.g., one-cut point, two-cut point, multi-cut-point, or uniform crossover). This method works well with the bit string representation. The performance of GAs depends to a great extend on the performance of the crossover operator used (e.g., partial-mapped crossover, order crossover, or position-based crossover) (2).

Mutation. Mutation is a background operator that produces spontaneous random changes in various chromosomes. A simple way to achieve mutation would be to alter one or more genes. In GAs, mutation serves the crucial role of either 1) replacing the genes lost from the population during the selection process so that they can be tried in a new context or 2) providing the genes that were not present in the initial population. Many different mutation operators are available for different genetic representations. Such as replacement mutation works well with the bit string representation, the uniform mutation, boundary mutation, dynamic mutation, and so on work well with the real number representation; several mutation operators work well for integer and string representations (e.g., inversion mutation, insertion mutation, displacement mutation, and swap mutation).

Selection. A selection (reproduction) operator is intended to improve the average quality of the population by giving the high-quality chromosomes a better chance to get copied into the next generation. Selection provides the driving force in a GA. With too much force, a genetic search will terminate prematurely, whereas with too little force, evolutionary progress will be slower than necessary. Typically, a lower selection pressure is indicated at the start of the genetic search in favor of a wide exploration of the search space, whereas a higher selection pressure is recommended at the end in order to narrow the search space. The selection directs the genetic search toward promising regions in the search space. During the past two decades, many selection methods have been proposed, examined, and compared. One of the common proportional selections is the so-called *Roulette wheel selection*, and other selection types like *Tournament selection, Elitist selection, (μ,λ) selection*, and $(\mu + \lambda)$ *selection* are deterministic procedures that select the best chromosomes from parents and offspring.

Major Advantages of Genetic Algorithm

GA has received considerable attention regarding their potential as a novel optimization technique. Three major advantages exist when applying GA to optimization problems:

Adaptability. GA does not have much mathematical requirements about the optimization problems. Because of the evolutionary nature, GA will search for solutions without regard to the specific inner workings of the problem. GA can handle any kind of objective functions and any kind of constraints, i.e., linear or nonlinear, defined on discrete, continuous, or mixed search spaces.

Robustness. The use of evolution operators makes GA very effective in performing global search (in probability), whereas most of conventional heuristics usually perform local search. It has been proved by many studies that GA is more efficient and more robust in locating an optimal solution and reducing a computational effort than other conventional heuristics.

Flexibility. GA provides us with a great flexibility to hybridize with domain-dependent heuristics to make an efficient implementation for a specific problem.

ADAPTATION OF GENETIC ALGORITHMS

Since GAs are inspired from the idea of evolution, it is natural to expect that the adaptation is used not only for finding solutions to a given problem but also for tuning GAs to the particular problem. During the past few years, many adaptation techniques have been suggested and tested to obtain an effective implementation of GAs to real-world problems. In general, two kinds of adaptations exist: 1) adaptation to problems and 2) adaptation to evolutionary processes.

The difference between these two adaptations is that the first one advocates modifying some components of GAs, such as representation, crossover, mutation, and selection to choose an appropriate form of the algorithm to meet the nature of a given problem. The second one suggests a way to tune the parameters of the changing configurations of GAs while solving the problem. According to Herrera and Lozano, the later type of adoption can be divided even more into the following classes (9): adaptive parameter settings, adaptive genetic operators, adaptive selection, adaptive representation, and adaptive fitness function.

Among these classes, the parameter adaptation has been studied extensively in the past ten years because the strategy parameters such as mutation probability, crossover probability, and population size are key factors in the determination of the exploitation versus exploration tradeoff.

Structure Adaptation

The structure adaptation technique aims at adapting the GA's structure or problem's structure to obtain an effective implementation of GAs to the problems. GAs were first created as a kind of generic and weak method featuring binary encoding and binary genetic operators. This approach requires a modification of an original problem into an appropriate form suitable for GAs. The approach includes a mapping between potential solutions and binary representation, taking care of decoding or repair procedures. For complex problems, such an approach usually fails to provide successful applications.

To overcome such problems, various nonstandard implementations of GAs have been created for particular problems. This approach leaves the problem unchanged and adapts GAs by modifying a chromosome representation of a potential solution and by applying appropriate genetic operators.

But in general, it is not a good choice to use the whole original solution of a given problem as the chromosome because many real problems are too complex to have a suitable implementation of GAs with the whole solution representation. Generally, the encoding methods can be either direct or indirect. In the direct encoding method, the whole solution for a given problem is used as a chromosome. For a complex problem, however, such a method will make almost all conventional genetic operators unusable because many offspring will be infeasible or illegal. On the contrary, in the indirect encoding method, just the necessary part of a solution is used as a chromosome. Solutions then can be generated by a decoder. A decoder is a problem-specific and determining procedure to generate a solution according to the permutation and/or the combination of the items produced by GAs. With this method, the GAs will focus their search solely on the interesting part of solution space.

A third approach is to adapt both GAs and the given problem. A common feature of combinatorial optimization problems is to find a permutation and/or a combination of some items associated with side constraints. If the permutation and/or combination can be determined, a solution then can be derived with a problem-specific procedure. With this third approach, GAs are used to evolve an appropriate permutation and/or combination of some items under consideration, and a heuristic method is subsequently used to construct a solution according to the permutation and combination.

Parameter Adaptation

The behaviors of GAs are characterized by the balance between exploitation and exploration in the search space. The balance is affected strongly by the strategy parameters such as population size, maximum generation, crossover probability, and mutation probability. How to choose a value to each parameter and how to find the values efficiently are very important and promising areas of research of GAs. A recent survey on adaptation techniques is given by Herrera and Lozano (9) and by Hinterding, et al. (10).

Usually, fixed parameters are used in most applications of GAs. The values for the parameters are determined with a set-and-test approach. Because a GA is an intrinsically dynamic and adaptive process, the use of constant parameters is thus in contrast to the general evolutionary spirit. Therefore, it is a natural idea to try to modify the values of strategy parameters during the run of the algorithm. It is possible to do this in various ways: 1) by using some rule; 2) by taking feedback information from the current state of search; or 3) by employing some self-adaptive mechanism. Gen and Cheng surveyed various adaptive methods using fuzzy logic controlled (FLC) (3). Subbu, et al. suggested a fuzzy logic controlled GA (FLC-GA), and the FLC-GA uses a fuzzy knowledge-base developed (11). This scheme can adaptively adjust the rates of crossover and mutation operators. Song, et al. used two FLCs (12): one for the crossover rate and the other for the mutation rate. These parameters are considered as the input variables of GAs and are also taken as the output variables of the FLC. Yun and Gen proposed an extended FLC-GA method based on the basic concept of Song, et al.'s method (13). A detailed survey is introduced in Ref. 14.

MULTIOBJECTIVE GENETIC ALGORITHM

Optimization deals with the problems of seeking solutions over a set of possible choices to optimize certain criteria. If only one criterion can be taken into consideration, it becomes a single objective optimization problem, which have been studied extensively for the past 50 years. If more than one criterion must be treated simultaneously, we have multiple objective optimization problems (15,16). Multiple objective problems develop in the design, modeling, and planning of many complex real systems in the areas of industrial production, urban transportation, capital budgeting, forest management, reservoir management, layout and landscaping of new cities, energy distribution, and so on. It is easy to find that almost every important real-world decision problem involves multiple and conflicting objectives that need to be tackled while respecting various constraints, leading to overwhelming problem complexity. The multiple objective optimization problems have been receiving growing interest from researchers with various background since early 1960 (17). Several scholars have made significant contributions to the problem. Among them, Pareto is perhaps one of the most recognized pioneers in the field (18). Recently, GAs have received considerable attention as a novel approach to multiobjective optimization problems, resulting in a fresh body of research and applications known as evolutionary multiobjective optimization (EMO).

Basic Concepts of Multiobjective Optimizations

A single objective optimization problem is usually given in the following form:

$$\max \quad z = f(x) \tag{1}$$

$$\text{s.t.} \quad g_i(x) \le 0, \qquad i = 1, 2, \ldots, m \tag{2}$$

where $x \in R^n$ is a vector of n decision variables, $f(x)$ is the objective function, and $g_i(x)$ are inequality constraint m functions, which form the area of feasible solutions. We usually denote the feasible area in decision space with the set S as follows:

$$S = \{x \in R^n | g_i(x) \le 0, \qquad i = 1, 2, \ldots, m, x \ge 0\} \tag{3}$$

Without loss of generality, a multiple objective optimization problem can be formally represented as follows:

$$\max \quad \{z_1 = f_1(x), z_2 = f_2(x), \cdots, z_q = f_q(x)\} \tag{4}$$

$$\text{s.t.} \quad g_i(x) \le 0, \qquad i = 1, 2, \ldots, m \tag{5}$$

Sometimes, we graph the multiple objective problem in both decision space and criterion space. S is used to denote the feasible region in the *decision space*, and Z is used to denote the feasible region in the *criterion space*.

$$\begin{aligned} Z = \{z \in R^q | z_1 = f_1(x), z_2 = f_2(x), \cdots, z_q \\ = f_q(x), \quad x \in S\} \end{aligned} \tag{6}$$

where $x \in S$ is a vector of values of q objective functions. In the other words, Z is the set of images of all points in S. Although S is confined to the nonnegative region of R^n, Z is not confined necessarily to the nonnegative region of R^q.

Nondominated Solutions

In principle, multiple objective optimization problems are very different from single objective optimization problems. For the single objective case, one attempts to obtain the best solution, which is absolutely superior to all other alternatives. In the case of multiple objectives, there does not exist necessarily such a solution that is the best with respect to all objectives because of incommensurability and conflict among objectives. Therefore, a set of solutions usually exists for the multiple objective cases that cannot be simply compared with each other. Such kind of solutions are called *nondominated solutions* or *Pareto optimal solutions*, for which no improvement in any objective function is possible without sacrificing on at least one of the other objective functions. For a given nondominated point in the criterion space Z, its image point in the decision space S is called *efficient* or *noninferior*. A point in S is efficient if and only if its image in Z is nondominated.

Definition 1. For a given point $z_0 \in Z$, it is nondominated if and only if another point $z \in Z$ does not exist such that for the maximization case,

$$z_k > z_k^0, \qquad \text{for some } k \in \{1, 2, \ldots, q\} \tag{7}$$

$$z_l > z_l^0, \qquad \text{for all } l \neq k \tag{8}$$

where z_0 is a dominated point in the criterion space Z with q objective functions.

Definition 2. For a given point $x_0 \in S$, it is efficient if and only if another point $x \in S$ does not exist such that for the maximization case,

$$f_k(x) > f_k(x_0), \qquad \text{for some } k \in \{1, 2, \ldots, q\} \tag{9}$$

$$f_1(x) \geq f_1(x_0), \qquad \text{for all } l \neq k \tag{10}$$

where x_0 is an inefficient in the decision space S with q objective functions.

Features of Genetic Search. The inherent characteristics of GAs demonstrate why genetic search is possibly well suited to the multiple objective optimization problems. The basic feature of GAs is the multiple directional and global search by maintaining a population of potential solutions from generation to generation. The *population-to-population* approach is hopeful to explore all Pareto solutions.

GAs do not have much mathematical requirements about the problems and can handle any kind of objective functions and constraints. Because of their evolutionary nature, the GAs can search for solutions without regard to the specific inner workings of the problem. Therefore, it is more hope for solving much complex problems than the conventional methods.

Because GAs, as a kind of meta-heuristics, provide us a great flexibility to hybridize with conventional methods into their main framework, we can take both advantages of the GAs and the conventional methods to make much more efficient implementations for the problems. The ingrowing research on applying GAs to the multiple objective optimization problems present a formidable theoretical and practical challenge to the mathematical community (3).

Fitness Assignment Mechanism

GAs are essentially a kind of meta-strategy methods. When applying the GAs to solve a given problem, it is necessary to refine on each major component of GAs, such as *encoding methods*, *recombination operators*, *fitness assignment*, *selection operators*, *constraints handling*, and so on, in order to obtain a best solution to the given problem. Because the multiobjective optimization problems are the natural extensions of constrained and combinatorial optimization problems, so many useful methods based on GAs developed during the past two decades. One of special issues in the multiobjective optimization problems is fitness assignment mechanism. Since the 1980s, several fitness assignment mechanisms have been proposed and applied in multiobjective optimization problems (3,19). Although most fitness assignment mechanisms are just a different approach and suitable to different cases of multiobjective optimization problems, to understanding the development of multiobjective GAs, we classify algorithms according to proposed years of different approaches:

Type 1: Vector Evaluation Approach. Vector evaluated genetic algorithm (veGA) (20) is the first notable work to solve multiobjective problems in which it uses a vector fitness measure to create the next generation (20). The selection step in each generation becomes a loop. Each time through the loop the appropriate fraction of the next generation, or subpopulation, is selected on the basis of each objective. The entire population is shuffled thoroughly to apply crossover and mutation operators, which is performed to achieve the mating of individuals of different subpopulations.

Type 2: Pareto Ranking + Diversity. ***Multiobjective Genetic Algorithm (21).*** Fonseca and Fleming proposed a multiobjective genetic algorithm (moGA) in which the rank of a certain individual corresponds to the number of individuals in the current population by which it is dominated. Based on this scheme, all the nondominated individuals are assigned rank 1, whereas dominated ones are penalized according to the population density of the corresponding region of the tradeoff surface.

Nondominated Sorting Genetic Algorithm (22). Srinivas and Deb also developed a Pareto ranking-based fitness assignment and called it the nondominated sorting genetic algorithm (nsGA). In each method, the nondominated solutions constituting a nondominated front are assigned the same dummy fitness value. These solutions are shared with their dummy fitness values (phenotypic sharing on the decision vectors) and are ignored in the other classification process. Finally, the dummy fitness is set to a value less than the smallest shared fitness value in the current nondominated front. Then the next front is extracted. This procedure is repeated until all individuals in the population are classified.

Type 3: Weighted Sum + Elitist Preserve. ***Random-Weight Genetic Algorithm (23).*** Ishibuchi and Murata proposed a weighted-sum based fitness assignment method, called a random-weight genetic algorithm (rwGA), to obtain a variable search direction toward the Pareto frontier. The weighted-sum approach can be viewed as an extension of methods used in the multiobjective optimizations to GAs. It assigns weights to each objective function and combines the weighted objectives into a single objective function. In rwGA, each objective $f_k(x)$ is assigned a weight $w_k = r_k/\sum_{j=1}^{q} r_j$, where r_j is a non-negative random number between [0, 1] with q objective functions. And the scalar fitness value is calculated by summing up the weighted objective value $w_k{\cdot}f_k(x)$. To search for multiple solutions in parallel, the weights are not fixed and can move uniformly the sample area toward the whole frontier.

Strength Pareto Evolutionary Algorithm II(24). Zitzler and Thiele proposed a strength Pareto evolutionary algorithm (spEA) (25) and an extended version spEA II (24) that combines several features of previous (moGAs) in a unique manner. The fitness assignment procedure is a two-stage process. First, the individuals in the external nondominated set P' are ranked. Each solution $i \in P$, is assigned a real value $s_i \in [0, 1)$, called *strength*; s_i is proportional to the number of population members $j \in P$ for which $i > j$. Let n denote the number of individuals in P that are covered by i, and assume N is the size of P. Then s_i is defined as $s_i = n/(N+1)$. The fitness f_i of objective i is equal to its strength: $f_i = s_i$. Afterward, the individuals in the population P are evaluated. The fitness of an individual $j \in P$ is calculated by summing the strengths of all external nondominated solutions $i \in P$, that cover j. The fitness is $f_j = 1 + \sum_{i \in (i>j)} s_i$, where $f_j \in [1, N)$.

Adaptive-Weight Genetic Algorithm (3). Gen and Cheng proposed another weight sum-based fitness assignment method, called the adaptive-weight genetic algorithm (awGA), which uses some useful information from the current population to readjust weights to obtain a search pressure toward the Pareto frontier. When considering the maximization problem with q objectives, we define two extreme points: the maximum extreme point $z^+ = \{z_1^{\max}, z_2^{\max}, \ldots, z_q^{\max}\}$ and the minimum extreme point $z^- = \{z_1^{\min}, z_2^{\min}, \ldots, z_q^{\min}\}$ in each generation. Each objective k is assigned a weight $w_k = 1/(z_k^{\max} - z_k^{\min})$. And the scalar fitness value is calculated by $\sum_{k=1}^{q}(f_k(x) - z_k^{\min})/(z_k^{\max} - z_k^{\min})$.

As show in Fig. 2, the hyperplane divides the criteria space Z into two half spaces: One half space contains the positive ideal point, denoted as Z^+, and the other half space contains the negative ideal point, denoted as Z^-. All examined Pareto solutions lie in the space Z^+, and all points lie in the Z^+ have larger fitness values than the points in the space Z^-. As the maximum extreme point approximates to the positive ideal point along with the evolutionary progress, the hyperplane will gradually approach to the positive ideal point. Therefore, awGA can readjust its weights according to the current population to obtain a search pressure toward to the positive ideal point.

Nondominated Sorting Genetic Algorithm II (26). Deb et al. suggested a nondominated sorting-based approach, called a nondominated sorting genetic algorithm II (nsGA II) (19,27), which alleviates the three difficulties: computational complexity, nonelitism approach, and the need for specifying a sharing parameter. The nsGA II was advanced from its origin, nsGA. In nsGA II, a nondominated sorting approach is used for each individual to create Pareto rank, and a crowding distance assignment method is applied to implement density estimation. In a fitness assignment between two individuals, nsGA II prefers the point with a lower rank value, or the point located in a region with fewer numbers of points if both points belong to the same front. Therefore, by combining a fast nondominated sorting approach, an elitism scheme and a parameterless sharing method with the original nsGA, nsGA II claims to produce a better spread of solutions in some testing problems.

Interactive Adaptive-Weight Genetic Algorithm (28). Lin and Gen proposed an interactive adaptive-weight genetic algorithm (i-awGA), which is an improved adaptive-weight fitness assignment approach with the consideration of the disadvantages of weighted-sum approach and Pareto ranking-based approach. They combined a penalty term to the fitness value for all of dominated solutions. First, calculate the adaptive weight $w_i = 1/(z_i^{\max} - z_i^{\min})$ for each objective $i = 1, 2, \ldots, q$ by using awGA. Afterward, calculate the penalty term $p(v_k) = 0$, if v_k is a nondominated solution in the nondominated set P. Otherwise $p(v_{k'}) = 1$ for a dominated solution $v_{k'}$. Last, calculate the fitness value of each chromosome by combining the method as follows and they adopted roulette wheel selection as supplementary

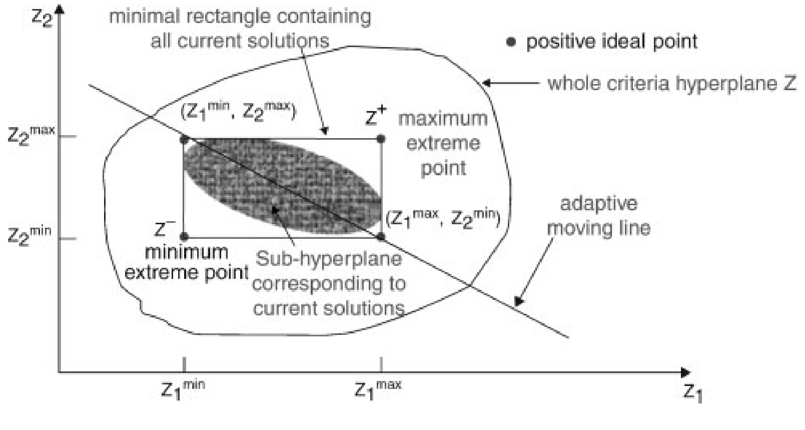

Z_2
minimal rectangle containing all current solutions
● positive ideal point
whole criteria hyperplane Z
(Z_1^{min}, Z_2^{max})
Z^+ maximum extreme point
Z_2^{max}
Z_2^{min}
adaptive moving line
(Z_1^{max}, Z_2^{min})
Z^- minimum extreme point
Sub-hyperplane corresponding to current solutions
Z_1^{min}
Z_1^{max}
Z_1

Figure 2. Adaptive-weights and adaptive hyperplane.

to the i-awGA.

$$\text{eval}(v_k) = \sum_{i=1}^{q} w_i(z_i^k - z_i^{\min}) + p(v_k), \qquad \forall\, k \in popSize \quad (11)$$

Performance Measures

Let S_j be a solution set ($j = 1, 2, \ldots, J$). To evaluate the efficiency of the different fitness assignment approaches, we have to define explicitly measures evaluating closeness of S_j from a known set of the Pareto-optimal set S^*. For example, the following common three measures are considered that are used already in different moGA studies. They provide a good estimate of convergence if a reference set for S^* (i.e., the Pareto optimal solution set or a near-Pareto optimal solution set) is chosen.

Number of Obtained Solutions $|S_j|$. Evaluate each solution set depend on the number of obtained solutions.

Ratio of Nondominated Solutions $R_{NDS}(S_j)$. This measure simply counts the number of solutions that are members of the Pareto optimal set S^*. The $R_{NDS}(S_j)$ measure can be written as follows:

$$R_{NDS}(S_j) = \frac{|S_j - \{x \in S_j | \exists r \in S^* : r < x\}|}{|S_j|} \qquad (12)$$

where $r < x$ means that the solution x is dominated by the solution r. The $R_{NDS}(S_j) = 1$ means all solutions are members of the Pareto-optimal set S^*, and $R_{NDS}(S_j) = 0$ means no solution is a member of the S^*. It is an important measure that although the number of obtained solutions $|S_j|$ is large, if that the ratio of nondominated solutions $R_{NDS}(S_j)$ is 0, it may be the worst result. The difficulty with the above measures is that although a member of S_j is Pareto-optimal, if that solution does not exist in S^*, it may not be counted in $R_{NDS}(S_j)$ as a non-Pareto-optimal solution. Thus, it is essential that a large set for S^* is necessary in the above equations.

Average Distance $\mathbf{D1_R}(S_j)$. Instead of finding whether a solution of S_j belongs to the set S^*, this measure finds an average distance of the solutions of S_j from S^*, as follows:

$$\text{D1}_\text{R}(S_j) = \frac{1}{|S^*|} \sum_{r \in s^*} \min\{d_{rx} | x \in S_j\} \qquad (13)$$

where d_{rx} is the distance between a current solution x and a reference solution r in the two-dimensional normalized objective space. f_i means the objective function for each objective $i = 1, 2, \ldots, q$.

$$d_{rx} = \sqrt{\sum_{i=1}^{q} (f_i(r) - f_i(x))^2} \qquad (14)$$

The smaller the value of $\text{D1}_\text{R}(S_j)$ is, the better the solution set S_j is. This measure explicitly computes a measure of the closeness of a solution set S_j from the set S^*.

Reference Set S^*. For making a large number of solutions in the reference set S^*, the first step calculates the solution sets with special GA parameter settings and a much longer computation time by each approach that is used in comparison experiments, and the second step combine these solution sets to calculate the reference set S^*. In the future, a combination of small but reasonable GA parameter settings for comparison experiments will be conducted, and thus ensure the effectiveness of the reference set S^*.

OVERALL PROCEDURE OF GENETIC ALGORITHM

The $P(t)$ and $C(t)$ are parents and offspring, respectively, in current generation t; the implementation structure of GA with combining the adaptive method and the multiobjective fitness assignment method is described as follows:

```
procedure: adaptive multiobjective GA
input: problem data, GA parameters
output: Pareto optimal solutions $E$
begin
    $t \leftarrow 0$; // $t$: generation number
    initialize $P(t)$ by encoding routine;        // $P(t)$: population of individuals
    calculate objectives $z_i(P)$, $i=1,\ldots,q$ by decoding routine;
    create Pareto $E(P)$;
    evaluate $eval(P)$ by fitness assignment routine;
    while (not terminating condition) do
        create $C(t)$ from $P(t)$ by crossover routine;        // $C(t)$: offspring
        create $C(t)$ from $P(t)$ by mutation routine;
        calculate objectives $z_i(C)$, $i=1,\ldots,q$ by decoding routine;
        update Pareto $E(P, C)$;
        evaluate $eval(P, C)$ by fitness assignment routine;
        auto-tuning GA parameters by parameter adaptive routine;
        select $P(t+1)$ from $P(t)$ and $C(t)$ by selection routine;
        $t \leftarrow t + 1$;
    end
    output Pareto optimal solutions $E(P, C)$
end
```

GA-Based Applications

GAs are powerful and broadly applicable stochastic search and optimization techniques. The major reason is that the advantages (adaptability, robustness, and flexibility) of GAs are very useful for applying the GAs to many complex problems that are very difficult to solve by conventional techniques. However, most problems of computer science and engineering are optimization problems subject to complex constraints. Simple GAs usually do not produce successful applications for these thorny optimization problems. Therefore, a method to tailor GAs to meet the nature of these problems is one of the major focuses of research into computer science and engineering-oriented genetic algorithms.

Combinatorial Optimizations

Combinatorial optimization studies problems that are characterized by a finite number of feasible solutions. An important and widespread area of applications concerns the efficient use of scarce resources to increase productivity. Typical problems include knapsack, set-covering, bin-packing, quadratic assignment, minimum spanning tree, machine scheduling, sequencing and balancing, cellular manufacturing design, vehicle routing, facility location and layout, traveling salesman problem, and so on (2).

Minimum Spanning Tree Models. The minimum spanning tree (MST) problem is one of the best-known network optimization problems used for designing backbone networks. Let G be a weighted, connected, undirected graph with node set V and edges E. A spanning tree on G is a maximal, acyclic subgraph of G; that is, it connects all of G's nodes and contains no cycles. A spanning tree's cost is the sum of the costs of its edges; a spanning tree with the smallest possible cost is a MST on G. Recently, researchers have described GAs for several kinds of constrained MST-related problems: capacitated MST (29), degree-constrained MST (30), stochastic MST (31), quadratic MST (32), probabilistic MST (33), multicriteria MST (34), and leaf-constrained MST (35) etc.

Genetic Representations. Lin and Gen (36) summarized the several kinds of classification of encoding methods as: (1) Characteristic vectors-based encoding [binary-based encoding (37,38), random key-based encoding (39,40)], (2) edge-based encoding (41,42), and (3) node-based Encoding [Prüfer number-based encoding (43), predecessor-based encoding (44)]. Furthermore, Lin and Gen propose a node-based encoding method, PrimPred-based encoding, that adopted Prim's algorithm in chromosome generating procedure (36).

Knapsack Models. Suppose that we want to fill up a knapsack by selecting some objects among various objects (generally called items). n different items are available, and each item j has a weight of w_j and a profit of p_j. The knapsack can hold a weight of at most W. The problem is to find an optimal subset of items so as to maximize the total profits subject to the knapsack's weight capacity. The profits, weights, and capacity are positive integers. The other extended knapsack models (multiple-choice knapsack model, multiconstraints knapsack model, etc.) are introduced in Ref. 45.

Binary-Based Encoding. A binary string is a natural representation of knapsack problem, where one means the inclusion and zero means the exclusion of one item from the knapsack. For example, a solution for the ten-item problem can be represented as $v_k = [0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0]$. It means that items 2, 4, and 9 are selected to be filled in the knapsack.

Order-Based Encoding. For a ten item problem, the kth chromosome $v_k = [2, 4, 9, 1, 3, 5, 7, 8, 6, 10]$ is an order of item for it to be filled into the knapsack. The same result (items 2, 4, and 9 are selected) can be decoded.

Set-Covering Model. The problem is a classic question in computer science and complexity theory. As input you are given several sets. They may have some elements in common. The problem is to select a minimum number of these sets so that the sets you have picked contain all the elements that are contained in any of the sets in the input. The sets can be formulated as an m-row/n-column zero–one matrix, and the objective is cover rows of the matrix by a subset of columns at minimal cost. Considering a vector $\boldsymbol{x}$ such that $x_j = 1$ if column j (with a cost $c_j > 0$) is in the solution and $x_j = 0$ otherwise ($j = 1, 2, \ldots, n$). The objective is to cover $A = [a_{ij}]$, $i = 1,2,\ldots,m, j = 1,2,\ldots,n$, a zero–one matrix by a subset of columns at minimal cost.

Column-Based Encoding. Column-based encoding is an n-bit binary string that is used for an n column problem. A value of 1 for the ith bit implies that column i is in the solution. For a six-column problem, the vector $x_1, x_{3,}$ and x_5 are selected by chromosome $v_k = [1\ 0\ 1\ 0\ 1\ 0]$.

Row-Based Encoding. The length of chromosome is equal to the number of rows for a given problem. The location of each gene corresponds to a row, and the encoded value of each gene is a column that covers that row.

Bin-Packing Model. The bin-packing problem consists of placing n objects into several bins (at most n bins). Each object has a weight ($w_i > 0$) and each bin has a limited bin capacity ($c_i > 0$). The objective is to find a best assignment of objects to bins such that the total weight of the objects in each bin does not exceed its capacity and the number of bins used is minimized.

Bin-Based Encoding. The position of a gene is used to represent an object, and the value of the gene is used to represent a bin in which the corresponding object is put it. For instance, the chromosome $v_k = [1\ 4\ 2\ 3\ 5\ 2]$ would encode a solution where the first object is in bin 1, the second is in bin 4, the third is in bin 2, the fourth is in bin 3, the fifth is in bin 5, and the sixth is in bin 2.

Object-Based Encoding. Encode the permutations of objects and then apply a decoder to retrieve the corresponding solution.

Group-Based Encoding. Chromosomes are item oriented, instead of being group oriented.

Traveling Salesman Model. The traveling salesman problem (TSP) is one of the most widely studied combinatorial optimization problems. Its statement is deceptively simple: A salesman seeks the shortest tour through n cities.

Permutation-Based Encoding. This direct representation is perhaps the most natural representation of a TSP, where cities are listed in the order in which they are visited. For example, a tour of a nine-city TSP: $3-2-5-4-7-1-6-9-8$ is represented simply as follows: v_k = [3 2 5 4 7 1 6 9 8]. This representation is also called as path-based encoding or order-based encoding.

Random Key-Based Encoding. This indirect representation encodes a solution with random numbers from (0,1). These values are used as sort keys to decode the solution. For example, a chromosome to a nine-city problem may be $v_k = [0.23\ 0.82\ 0.45\ 0.74\ 0.87\ 0.11\ 0.56\ 0.69\ 0.78]$, where position i in the list represents city i, and the random number in position i determines the visiting order of city i in a TSP tour. We sort the random keys in ascending order to get the following tour: $6-1-3-7-8-4-9-2-5$.

Genetic Operators. For permutation encoding, such as partial-mapped crossover, order crossover, or position-based crossover and inversion mutation, insertion mutation, or displacement mutation can be adopted. The detail description is introduced in Ref. 2.

Network Design Optimization

Network models are a fundamental issue in many disciplines, including applied mathematics, computer science, engineering, management, and operations research. Furthermore, because any system or structure may be considered abstractly as a set of elements, certain pairs of which are related in a special way, it has a representation as a network. Networks provide a useful way to modeling real-world problems, which are used extensively in many different types of systems, such as communications, mechanical, electronic, manufacturing, and logistics (46).

Shortest Path Model. The shortest path problem (SPP) is the heart of network optimization problems. Let $G = (N, A)$ be a directed network, which consists of a finite set of nodes $N = \{1, 2, \ldots, n\}$ and a set of directed arcs $A = \{(i,j), (k, l), \ldots, (s, t)\}$ connecting m pairs of nodes in N. Arc (i, j) is said to be incident with nodes i and j, and it is directed from node i to node j. Suppose that each arc (i, j) has been assigned to a nonnegative value c_{ij}, the cost of (i, j). The SPP is to find the minimum cost z from a specified source node 1 to another specified sink node n.

Variable-Length Encoding. Munemoto et al. proposed a variable-length encoding to construct the shortest path (47). Its element represents nodes included in a path between a designated pair of source and destination nodes. Ahn and Ramakrishna developed this variable-length encoding. A new crossover operator exchanges partial chromosomes (partial-routes), and the mutation introduces new partial chromosomes (partial-routes) (48).

Fixed-Length Encoding. Inagaki et al. proposed a *fixed (deterministic) length chromosome* (49). The chromosomes in the algorithm are sequences of integers, and each gene represents a node ID that is selected randomly from the set of nodes connected with the node corresponding to its locus number. All the chromosomes have the same (fixed) length. In the crossover phase, one gene (from two-parent chromosomes) is selected at the locus of the starting node ID and put in the same locus of an offspring. One gene is then selected randomly at the locus of the previously chosen gene's number. This process is continued until the destination node is reached.

Priority-Based Encoding. Gen et al. proposed a priority-based encoding method (50). As all know, a gene in a chromosome is characterized by two factors: locus, i.e., the position of gene located within the structure of chromosome, and allele, i.e., the value the gene takes. In this encoding method, the position of a gene is used to represent node ID and its value represents the priority of the node among the candidates to construct a path. A path can be determined uniquely by the encoding.

Random Key-Based Encoding. Gen and Lin proposed an extended version of priority-based encoding (51) in a real number string, i.e., random key-based encoding. It not only can be decoded a path by same decoding procedure with priority-based encoding, but also most crossover and mutation operators can be adopted, because the chromosome is represented by real number code.

Maximum Flow Model. In a capacitated network, the maximum flow problem (MXF) is to send as much flow as possible between two special nodes, a source node s and a sink node t, without exceeding the capacity of any arc. The MXF model and the shortest path model are complementary. The two problems differ because they capture different aspects: the shortest path problem model arc costs but not arc capacities and maximum flow problem model capacities but not costs. Taken together, the shortest path problem and the maximum flow problem combine all the basic ingredients of network models. As such, they have become the nuclei of network optimization.

Priority-Based Encoding. One specific difficulty of the MXF is the solution presented by various numbers of paths. Until now, for presenting a solution of MXF with various paths, the general idea of chromosome design is to add several shortest paths-based encoding to one chromosome. The length of these representations is variable depending on various paths, and most offspring is infeasible after crossover and mutation operations. Gen and Lin adopt the priority-based encoding that is an effective representation to present a solution with various paths (52). For decoding process, after a path is calculated by a given priority-based chromosome, we update the flow capacity for each arc on the network. Then we can obtain another new path by the same chromosome depending on the new network structure. By repeating this way, we can obtain a solution with various numbers of paths for MXF problem.

Bicriteria MXF/MCF Model. The MXF finds a solution that sends the maximum flow from a source node s to a sink node t. The minimum cost flow problem (MCF) determines a least cost shipment of a commodity through a network to satisfy demands at certain nodes from available supplies at other nodes. The bicriteria MXF/MCF model is an extended version considering the flow costs, flow capacities, and multiobjective optimization problems. This model provides a useful way for modeling real-world problems. For example, in a communication network, we want to find a set of links that consider the connecting cost (or delay) and the high throughput (or reliability) for increasing the network performance; in a manufacturing system, the two criteria under consideration are minimizing manufacturing cost and maximizing quality.

Priority-Based Encoding. Gen and Lin proposed an extended priority-based encoding for this bicriteria MXF/MCF model (28,53). They proposed a new crossover operator, called weight mapping crossover (WMX) and adopt insertion mutation, immigration operator, and interactive adaptive-weight fitness assignment to accelerate the evolutionary process.

Advanced Planning and Scheduling

The planning and scheduling of manufacturing systems always require resource capacity constraints, disjunctive constraints, and precedence constraints, because of the tight due dates, multiple customer-specific orders, and flexible process strategies. In this subsection, some hot topics in advanced planning and scheduling (APS) are introduced. These models mainly support the integrated, constraint-based, and planning of the manufacturing system to reduce lead times, lower inventories, increase throughput, and so on.

Job-Shop Scheduling Model. In the job-shop scheduling problem (JSP), we are given a set of jobs and a set of machines. Each machine can handle at most one job at a time. Each job consists of a chain of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. The objective is to find a schedule; that is, an allocation of the operations to time intervals on the machines that has a minimum duration required to complete all jobs (54).

Genetic Representations. Gen and Cheng gave the nine different representations for JSP in Ref. 2: operation-based encoding, job-based encoding, preference-list-based encoding, job-pair-relation-based encoding, priority-based encoding, disjunctive-graph-based encoding, completion-time-based encoding, machine-based encoding, and random key-based encoding. These representations can be classified into two basic encoding approaches: direct approach and indirect approach.

Flexible Job-Shop Scheduling Model. Flexible job shop is a generalization of the job shop and the parallel machine environment (55), which provides a closer approximation to a wide range of real manufacturing systems. In particular, a set of parallel machines exists with possibly different efficiency. The flexible job shop scheduling problem (fJSP) is to assign each operation to an available machine and to sequence the operations assigned on each machine to minimize the makespan, that is, the time required to complete all jobs.

Parallel Machine-Based Encoding. The chromosome is a list of machines placed in parallel. For each machine, we associate operations to execute. Each operation is coded by three elements: operation k, job i, and starting time t_{ikj}^{S} of operation o_{ik} on the machine j.

Parallel Job-Based Encoding. The chromosome is represented by a list of jobs. Information of each job is shown in the corresponding row where each case is constituted of two terms: machine j, which executes the operation and corresponding starting time ${t_{ikj}}^{\mathrm{S}}$.

Operations Machine-Based Encoding. Kacem et al. proposed an operations machine-based approach (56), which is based on a traditional representation called schemata theorem representation; it was first introduced in GAs by Holland (5).

Multistage Operation-Based Encoding. Gen and Zhang proposed a multistage operation-based encoding for fJSP (57,58). In the encoding process, all operations are defined as a multistage network denoting each operation as one stage. At each stage, available machines of each operation are defined as states. The length of chromosome is the number of operations. An integer number in the kth gene represents a machine number to which the operation k is assigned.

Resource-Constrained Project Scheduling Model. The objective of the resource-constrained project scheduling problem (rcPSP) is to schedule the activities such that precedence and resource constraints are obeyed and the makespan of the project is to be minimized. The resource constraints refer to limited renewable resources such as manpower, material, and machines that are necessary for carrying out the project activities (59).

Priority-Based Encoding. Gen and Cheng adopted priority-based encoding for this rcPSP (2). To improve the effectiveness of priority-based GA approach for large-scale rcPSP problems and extended resource-constrained multiple project scheduling problem, Kim et al. combined priority dispatching rules in priority-based encoding process (60,61).

Recently, some researchers have studied various algorithms for solving an rcPSP problem on a large scale. Their works have dealt with a variety of situations in which one or both of these types of constraints are relaxed, or at least simplified. And comparisons of eight different priority dispatching rules (minimum job slack, resource scheduling method, minimum late finish time, greatest resource demand, greatest resource utilization, shortest imminent operation, most jobs possible, and select jobs randomly) for an rc-PSP problem have been reported in previous studies

Assembly Line Balancing Model. Assembly line balancing problems (ALB) consist of distributing work required to assemble a product in mass or series production on an assembly line among a set of work stations. Several constraints and different objectives may be considered. The simple assembly line balancing problem consists of assigning tasks to workstations such that precedence relations between tasks and zoning or other constraints are met. The objective is to make the work content at each station most balanced. Two versions of the problem exist. The Type I simple assembly line balancing (sALB-I) problem, as described by Scholl, consists in finding an assignment of tasks to workstations such that the required number of workstations is minimized given a cycle time, i.e., the maximum work time of any workstation. The Type II simple assembly line balancing (sALB-II) problem consists in allocating tasks to a given number of workstations to minimize the cycle time.

Genetic Representations. GAs have been applied to solve various assembly line balancing problems (62–64). The genetic representations can be summarized as follows: 1) *Standard encoding* (65), the chromosome is defined as a vector containing the indexes of the stations to which the tasks are assigned; 2) *Order-based encoding* (66), the chromosomes are defined as a task sequence in feasible order; 3) *Priority-based encoding* (3), the chromosome represents the solution in an indirect manner: coding priority values of tasks and coding a sequence of priority rules and corresponding construction schemes; and 4) *Group encoding* (67), the encoding of each solution consists of two parts: The task part is identical to the standard encoding, and the group part contains a gene for each station.

Recently, Gao et al. proposed an innovative GA hybridized with local search for a robotic-based ALB problem (68). Based on different neighborhood structures, five local search procedures are developed to enhance the searchability of GA. The coordination between the local search procedures are well considered to escape from local optima and to reduce computation time.

Advanced Planning and Scheduling Model. The advanced planning and scheduling (APS) model includes a range of capabilities from finite capacity planning at the plant floor level through constraint-based planning to the latest applications of advanced logic for supply chain planning and collaboration (69). The objective of APS problem is usually to determine an optimal schedule with operation sequences for all the orders (jobs). That is, the problem we are treating can be defined as follows: A set of K orders are to be processed on N machines with alternative operations sequences and alternative machines for operations in the environment of the multiplant chain; we want to find an operations sequence for each job and a schedule in which jobs pass between machines and a schedule in which operations on the same jobs are processed such that it satisfies the precedence constraints and it is optimal with respect to the makespan minimization.

Moon–Kim–Gen's Approach. Several related works by Moon et al. (70) and Moon and Seo (71) have reported a GA approach especially for solving such kinds of APS problems. However, to derive a feasible complete schedule, they only considered the optimal operation sequence, but they selected the resources in terms of minimum processing time. That means, for machines assignment, they consider that the minimum processing time assignment is the optimal choosing strategy for the solution. However the transition time between plants and setup time between operations are ignored.

Multistage Operation-Based Encoding. Zhang and Gen proposed a multistage operation-based encoding for solving the APS problem (72,73). This encoding method considers both operation sequence and machine selection so that it is easy to present a solution of the problem. The chromosome presentation of multistage operation-based encoding consists of two parts: 1) priority-based encoding for operation sequence and 2) machine permutation encoding for machine selection.

AGV Dispatching Model in Manufacturing System. Automated guided vehicle (AGV) is a mobile robot used highly in industrial applications to move materials from point to point. AGV help to reduce costs of manufacturing and increase efficiency in a manufacturing system. For example, a flexible manufacturing system (FMS) is composed of various cells, also called working stations (or machine), each with a specific operation such as milling, washing, or assembly. Each cell is connected to the guide path network by a pickup/delivery (P/D) point where pallets are transferred from/to the AGVs. The objectives of AGV system are as follows: Minimize time required to complete all jobs (i.e., makespan), minimize vehicle travel times (empty or/and loaded), evenly distribute workload over AGVs, minimize total costs of movement, and minimize the time of labor that is handled after its due time (i.e., tardiness), minimize expected waiting times of loads, minimize the number of AGVs, and so on.

Priority-Based Encoding. Lin et al. adopted priority-based encoding for solving the AGV dispatching problem in FMS (74). In this encoding method, the position of a gene is used to represent task (AGV's transport) ID and its value is used to represent the priority of the task for constructing a sequence among candidates. A feasible sequence can be determined uniquely from this encoding by considering a task precedence constraint. After a generated task sequence, separate tasks occur to several groups for assigning different AGVs.

Logistics Network Optimization

It is said that logistics is the "last frontier for cost reduction" and the "third profit source" of enterprises, by Peter Deruke, an American management specialist (75). The interest in developing effective logistics system design models and efficient optimization methods has been stimulated by high costs of logistics and is potentially capable of securing considerable savings.

Transportation Models. The transportation problem (TP) was proposed originally by Hitchcock in 1941 (76). Since then the research on the problem has received a great deal of attention, and various variants of the basic transportation problem have been investigated. According to what kind of objective is used, the problem can be characterized as follows: 1) linear problem or nonlinear problem and 2) single objective problem or multiple objective problem. According to what kind of constraints is under consideration, the problem can be classified even more into 1) planar problem or solid problem and 2) balanced problem or unbalanced problem.

Matrix-Based Encoding. A matrix is perhaps the most natural representation of a solution for a transportation problem. The allocation matrix of a transportation problem can be written as follows:

$$X_k = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \qquad (15)$$

where X_k denotes the kth chromosome (solution) and the element of it, x_{ij} is the corresponding decision variable (5).

Prüfer Number-Based Encoding. The Prüfer number-based encoding incorporating this data structure of TP was proposed by Gen and Li (77). This GA uses the Prüfer number encoding based on a spanning tree, which is capable of representing all possible trees. Using the Prüfer number representation, the memory only requires $m + n - 2$ for a chromosome implementation. A transportation model has separable sets of nodes for plants and customers. From this point, Gen and Li designed a criterion for checking the feasibility of the chromosome.

Location Allocation Models. As an extension of TP, a location–allocation decision is a very important factor in logistics network design problems. It can be classified as 1) location problems, involve determining the location of one or more new DCs in one or more of several potential sites; 2) allocation problems, assume that the number and location of DCs are known as a priori and attempt to determine how each customer is to be served; and 3) location–allocation problems, involve determining not only how much each customer is to receive from each DC but also the number of DCs along with their locations and capacities.

Genetic Representation. In continuous location problems, a binary representation may result in locating two DCs that are very close to each other. Taniguchi et al. used a real number representation (78) where a chromosome consists of $m(x,y)$ pairs representing the sites of DCs to be located and p is the number of DCs. For instance, this is represented as $v = [(x_1,y_1),(x_2,y_2),\ldots,(x_i,y_i),\ldots,(x_m,y_m)]$, where the coordinate (x_i,y_i) denotes the location of the ith DC, $i = 1,\ldots, m$.

Multistage Logistics Network Models. For some real-world applications of logistics, it is often that the transportation problem is extended to satisfy several other additional constraints or is performed in several stages. A two-stage transportation problem (tsTP) model is proposed by Gen et al. (79), which minimize the total logistic system cost, including the opening cost of DCs and the shipping cost from plants to DCs, and from DCs to customers under the capacity constraints of plants and DCs. And some extended models of multistage logistics network are introduced in Refs. 80 and 81.

Priority-Based Encoding. Gen et al. proposed a new encoding method based on priority-based encoding (79). For each stage of transportation, a chromosome consists of priorities of sources and depots to obtain a transportation tree and its length is equal to total number of sources (m) and depots (n); i.e., $m + n$. The transportation tree corresponding with a given chromosome is generated by sequential arc appending between sources and depots. At each step, only one arc is added to the tree selecting a source (depot) with the highest priority and connecting it to a depot (source) considering minimum cost.

Flexible Logistics Network Models. Recently, with the constant demand to reduce transportation costs and improve customer service quality, design and optimization of logistics face more challenging issues, which is the improvement of the flexibility of logistics system; i.e., we want to change the traditional structure of the logistics: Plant-DC-Retailer-Customer, the use of direct shipment (Plant-Customer) as much as possible, or direct delivery (Plant-Retailer or DC-Customer). Lin et al. formulate a flexible multistage logistics network model by considering the direct shipment and direct delivery of logistics and inventory (82).

Genetic Representation. Lin et al. hybridized the priority-based encoding with random number-based encoding by dividing a chromosome into two segments (82). The first segment is encoded by using a priority-based encoding method that can escape the repairing mechanisms in the searching process of GA. The second segment of a chromosome consists of two parts: The first part with K loci containing the guide information about how to assign retailers in the network, and the other with length L including that information of customers. Each locus is assigned an integer in the range from 0 to 2.

Advanced Applications

Communication Network Models. The use of communication networks has increased significantly in the last decade because of the dramatic growth in the use of Internet for business and personal use. As the society trans-forms itself into an information society the network becomes the primary source for information creation, storage, distribution, and retrieval. The design and development of a reliable network to support the primary resource of an information society becomes a very critical activity. The reliability and service quality requirements of communication networks and the large investments in communication infrastructure have made it critical to design optimized networks that meet performance parameters. These factors have encouraged researchers to develop new models and methodologies for network design. GA and other evolutionary algorithms have been applied successfully to large and complex optimization problems in communication networks over the past decade, covering a variety of problem areas. Kampstra et al. gave an extensive literature survey, listing over 350 references on the use of GA and other evolutionary algorithms for solving communication network design problems (83).

Real-Time Tasks Scheduling Models. Real-time tasks can be classified into many kinds. Some real-time tasks are invoked repetitively. For example, one may wish to monitor the speed, altitude, and attitude of an aircraft every 100 ms. This sensor information will be used by periodic tasks that control the surfaces of the aircraft to maintain stability and other desired characteristics. In contrast, many other tasks are aperiodic, which occur only occasionally. Aperiodic tasks with a bounded interarrival time are called sporadic tasks. Critical (or hard real-time) tasks are those whose timely execution is critical. If the deadline is missed, catastrophes occur. Noncritical (or soft real-time) tasks are, as the name implies, not critical to the application. Gen and Yoo detailed survey GA-based approaches for various real-time tasks scheduling problems (84), such as a continuous soft real-time task scheduling problem on multiprocessor systems, real-time task scheduling in homogeneous multiprocessor systems, and real-time task scheduling in heterogeneous multiprocessor systems.

Reliability Optimization Models. In the broadest sense, *reliability* is a *measure of performance* of systems. As systems have grown more complex, the consequences of their unreliable behavior have become severe in terms of cost, effort, lives, and so on and the interest in assessing system reliability and the need for improving the reliability of products and systems have become very important. Reliability optimization problems concentrate on optimal allocation of redundancy components and optimal selection of alternative designs to meet a system requirement. Gen and Yun reported a survey GA-based approach for various reliability optimization problems (85), such as reliability optimization of redundant system, reliability optimization with alternative design, reliability optimization with time-dependent reliability, reliability optimization with interval coefficients, bicriteria reliability optimization, and reliability optimization with fuzzy goals.

BIBLIOGRAPHY

1. D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison-Wesley, 1989.
2. M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, New York: John Wiley & Sons, 1997.
3. M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, New York: John Wiley & Sons, 2000.
4. M. Gen, Genetic algorithms and their applications, *Springer Handbook of Engineering Statistics*, H. Pham (ed.), New York: Springer-Verlag, 2006, pp. 749–773.
5. Z. Michalewicz, *Genetic Algorithm + Data Structures = Evolu-Evolution Programs*, New York: Springer-Verlag, 1994.
6. J. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press, 1975.
7. I. Rechenberg, *Optimieriung technischer Systeme nach Prinzipien der biologischen Evolution*, Stuttgart: Frommann-Holzboog, 1973.
8. H. Schwefel, *Evolution and Optimum Seeking*, New York: John Wiley & Sons, 1995.
9. F. Herrera and M. Lozano, Adaptation of genetic algorithm parameters based on fuzzy logic controllers, in F. Herrera and J. Verdegay, (eds.), *Genetic Algorithms and Soft Computing*, Physica-Verlag, 1996, pp. 95–125.
10. R. Hinterding, Z. Michalewicz, and A. Eiben, Adaptation in evolutionary computation: a survey, *Proc. of IEEE Inter. Conf. on Evolutionary Computation,* Piscataway, NJ, 1997, pp. 65–69.
11. R. Subbu, A. Sanderson, and P. Bonissone, Fuzzy logic controlled genetic algorithms versus tuned genetic algorithms: an agile manufacturing application, *Proc. of the 1999 IEEE Inter. Symp. on Intelligent Control (ISIC)*, 1998, pp. 434–440.
12. Y. H. Song, G. S. Wang, P. T. Wang, and A. T. Johns, Environmental/economic dispatch using fuzzy logic controlled genetic algorithms, *IEEE Proc. on Generation, Transmission and Distribution*, **144**(4): 377–382, 1997.
13. Y. Yun and M. Gen, Performance analysis of adaptive genetic algorithms with fuzzy logic and heuristics, *Fuzzy Optimiz. Decision Making*, **2**(2): 161–175, 2003.
14. Y. Yun, Study on adaptive hybrid genetic algorithm and its applications to engineering design problems, PhD dissertation, Tokyo, Japan: Waseda University, 2005.
15. K. Dev, *Optimization for Engineering Design: Algorithms and Examples*, New Delhi: Prentice-Hall, 1995.
16. R. E. Steuer, *Multiple Criteria Optimization: Theory, Computation, and Application*, New York: John Wiley & Sons, 1986.
17. C. Hwang and K. Yoon, *Multiple Attribute Decision Making: Methods and Applications*, Berlin: Springer-Verlag, 1981.
18. V. Pareto, Manuale di Economica Polittica, Societa Editrice Libraia, Milan, Italy, 1906; translated into English by A. S. Schwier, as *Manual of Political Economy*, New York: Macmillan, 1971.
19. K. Deb, Genetic algorithms in multimodal function optimization, M.S. dissertation, Tuscaloosa: University of Alabama, 1989.
20. J. D. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms, *Proc. 1st Inter. Conf. on GAs*, 1985. pp. 93–100.
21. C. Fonseca and P. Fleming, An overview of evolutionary algorithms in multiobjective optimization, *Evolutionary Computation*, **3**(1): 1–16, 1995.
22. N. Srinivas and K. Deb, Multiobjective function optimization using nondominated sorting genetic algorithms, *Evolutionary Computation*, **3**: 221–248, 1995.
23. H. Ishibuchi and T. Murata, A multiobjective genetic local search algorithm and its application to flowshop scheduling, *IEEE Trans. on Systems., Man, & Cyber.*, **28**(3): 392–403, 1998.

24. E. Zitzler and L. Thiele, SPEA2: improving the strength pareto evolutionary algorithm, *Technical Report 103*, Computer Engineering and Communication Networks Lab (TIK), 2001.

25. E. Zitzler and L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, *IEEE Trans. on Evolutionary Computation*, **3**(4): 257–271, 1999.

26. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Trans. Evolutionary Computation*, **6**(2): 182–197, 2002.

27. K. Deb, *Multiobjective Optimization Using Evolutionary Algorithms*. Chichester, UK: Wiley, 2001.

28. L. Lin and M. Gen, Bicriteria network design problem using interactive adaptive-weight GA and priority-based encoding method, *IEEE Trans. Evolut. Computat.* In press.

29. A. Kershenbaum, Computing capacitated minimal spanning trees efficiently, *Networks*, **4**: 299–310, 1974.

30. S. Narula and C. Ho, Degree-constrained minimum spanning tree, *Computers Operat. Research*, **7**: 239–249, 1980.

31. H. Ishii, H. Shiode, and T. Nishida, Stochastic spanning tree problem, *Discrete Applied Mathematics*, **3**: 263–273, 1981.

32. W. Xu, Quadratic minimum spanning tree problems and related topics, Ph. D. dissertation, College Park: University of Maryland, 1984.

33. D. Bertismas, The probabilistic minimum spanning tree problem, *Networks* **20**: 245–275, 1990.

34. G. Zhou and M. Gen, Genetic algorithm approach on multicriteria minimum spanning tree problem, *European J. Operat. Res.*, **114**: 141–151, 1999.

35. L. M. Fernandes and L. Gouveia, Minimal spanning trees with a constraint on the number of leaves, *European J. Operat. Res.*, **104**: 250–261, 1998.

36. L. Lin and M. Gen, Node-based genetic algorithm for communication spanning tree problem, *IEICE Trans. Communications*, **E89-B**(4): 1091–1098, 2006.

37. L. Davis, D. Orvosh, A. Cox, and Y. Qiu, A genetic algorithm for survivable network design, *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, pp. 408–415.

38. P. Piggott and F. Suraweera, Encoding graphs for genetic algorithms: an investigation using the minimum spanning tree problem, in *Progress in Evolutionary Computation, vol. 956*, X. Yao, (ed.), New York: Springer, 1995, pp. 305–314.

39. B. Schindler, F. Rothlauf, and H. Pesch, Evolution strategies, network random keys, and the one-max tree problem, *Proc. Applic. of Evol. Computing on EvoWorkshops*, 2002, pp. 143–152.

40. F. Rothlauf., J. Gerstacker, and A. Heinzl, On the optimal communication spanning tree problem, *IlliGAL Technical Report*, Univ. of Illinois, 2003.

41. J. Knowles and D. Corne, A new evolutionary approach to the degree-constrained minimum spanning tree problem, *IEEE Trans. Evolutionary Comput.*, **4**(2): 125–134, 2000.

42. G. Raidl and B. Julstrom, Edge sets: an effective evolutionary coding of spanning trees, *IEEE Trans. Evolut. Comput.*, **7**(3): 225–239, 2003.

43. G. Zhou and M. Gen, Approach to degree-constrained minimum spanning tree problem using genetic algorithm, *Engineering Design Automation*, **3**(2): 157–165, 1997.

44. H. Chou, G. Premkumar, and C. Chu, Genetic algorithms for communications network design – an empirical study of the factors that influence performance, *IEEE Trans. Evolut. Comput.*, **5**(3): 236–249, 2001.

45. S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, Chichester: John Wiley & Sons, 1990.

46. M. Gen, R. Cheng and L. Lin, *Network Model and Optimization: Multiobjective Genetic Algorithm Approach*, Springer, 2008.

47. M. Munetomo, Y. Takai, and Y. Sato, An adaptive network routing algorithm employing path genetic operators, *Proc. 7th Int. Conf. on Genetic Algorithms*, 1997, pp. 643–649.

48. C. W. Ahn and R. S. Ramakrishna, A genetic algorithm for shortest path routing problem and the sizing of populations, *IEEE Trans. Evolut. Computat.*, **6**(6): 566–579, 2000.

49. J. Inagaki, M. Haseyama, and H. Kitajima, A genetic algorithm for determining multiple routes and its applications, *Proc. IEEE Inter. Symp. on Circuits and Systems*, 1999, pp. 137–140.

50. M. Gen, R. Cheng, and D. Wang, Genetic algorithms for solving shortest path problems, *Proc. IEEE Inter. Conf. on Evolutionary Computation*, 1997, pp. 401–406.

51. M. Gen and L. Lin, A new approach for shortest path routing problem by random key-based GA, *Proc. of Genetic and Evolutionary Computation Conference*, 2006, pp. 1411–1412.

52. M. Gen, L. Lin, and R. Cheng, Bicriteria network optimization problem using priority-based genetic algorithm, *IEEE Trans. Electron., Informat. Systems*, **124**(10): 1972–1978, 2004.

53. M. Gen and L. Lin, Multi-objective hybrid genetic algorithm for bicriteria network design problem, *Complexity Internat.*, **11**: 73–83, 2005.

54. C. Cheng, V. Vempati, and N. Aljaber, An application of genetic algorithms for flow shop problems, *Euro. J. Operat. Res.*, **80**: 389–396, 1995.

55. M. Pinedo, *Scheduling Theory, Algorithms and Systems*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

56. I. Kacem, S. Hammadi, and P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible jobshop scheduling problems, *IEEE Trans. Systems, Man Cybernet.*, Part C, **32**(1): 408–419, 2002.

57. H. Zhang and M. Gen, Multistage-based genetic algorithm for flexible job-shop scheduling problem, *J. Complexity Internat.*, **11**: 223–232, 2005.

58. M. Gen and H. Zhang, Effective designing chromosome for optimizing advanced planning and scheduling, *Intelligent Engineering Systems Through Artificial Neural Networks,* vol. 16, C. H. Dali et al. ASME Press, 2006, pp. 61–66.

59. M. Gen, K. W. Kim, and G. Yamazaki, Project scheduling using hybrid genetic algorithm with fuzzy logic controller in SCM Environment, *J. Tsinghua Sci. Technol.*, **8**(1):19–29, 2003.

60. K. W. Kim, M. Gen, and G. Yamazaki, Hybrid genetic algorithm with fuzzy logic for resource-constrained project scheduling, *Applied Soft Comp.*, **2**(3): 174–188, 2003.

61. K. W. Kim, Y. S. Yun, J. M. Yoon, M. Gen, and G. Yamazaki, Hybrid genetic algorithm with adaptive abilities for resource-constrained multiple project scheduling, *Computers In Industry*, **56**(2): 143–160, 2005.

62. Y. Tsujimura, M. Gen, and E. Kubota, Solving fuzzy assembly-line balancing problem with genetic algorithms, *Computers & Industrial Engineering*, **29**(1/4): 543–547, 1995.

63. M. Gen, Y. Tsujimura, and Y. Li, Fuzzy assembly line balancing using genetic algorithms, *Comput. Industrial Engineer.*, **31**(3/4): 631–634, 1996.

64. J. Rubinovitz and G. Levitin, Genetic algorithm for line balancing, *Internat. J. Production Econ.*, **41**: 343–354, 1995.

65. E. J. Anderson and M. C. Ferris, Genetic algorithms for combinatorial optimization: the assembly line balancing problem, *ORSA J. Computing*, **6**: 161–173, 1994.

66. Y. Y. Leu, L. A. Matheson, and L. P. Rees, Assembly line balancing using genetic algorithms with heuristic-generated initial populations and multiple evaluation criteria, *Decision Sciences*, **25**: 581–606, 1994.

67. E. Falkenauer, A hybrid grouping algorithm for bin packing, *J. Heuristics*, **2**: 5–30, 1996.

68. J. Gao, G. Chen, L. Sun, and M. Gen, An efficient approach for type II robotic assembly line balancing problems, *Comp. Industr. Engineer.*, In press.

69. D. Turbide, Advanced planning and scheduling (APS) systems, *Midrange ERP Magazine*, **1**: 1998.

70. C. Moon, J. S. Kim, and M. Gen, Advanced planning and scheduling based on precedence and resource constraints for e-plant chains, *Internat. J. Product. Res.*, **42**(15): 2941–2955, 2004.

71. C. Moon and Y. Seo, Evolutionary algorithm for advanced process planning and scheduling in a multi-plant, *Comp. Indust. Engineer.*, **48**(2): 311–325, 2005.

72. H. Zhang, M. Gen, and Y. Seo, An effective coding approach for multiobjective integrated resource selection and operation sequences problem, *J. Intelli. Manufact.*, **17**(4): 385–397, 2006.

73. H. Zhang, Study on evolutionary scheduling problems in integrated manufacturing system, Ph.D. dissertation, Tokyo, Japan: Waseda University, 2006.

74. L. Lin, S. W. Shinn, M. Gen, and H. Hwang, Network model and effective evolutionary approach for AGV dispatching in manufacturing system, *J. Intelli. Manufact.*, **17**(4): 465–477, 2006.

75. J. Guo, Third-party logistics - key to rail freight development in China, *Japan Railway Transp. Rev.*, **29**: 32–37, 2001.

76. F. Hitchcock, The distribution of a product from several sources to numerous locations, *J. of Math. Physics*, **20**: 224–230, 1941.

77. M. Gen and Y. Z. Li, Solving multi-objective transportation problem by spanning tree-base genetic algorithm, *Adaptive Comput. Design Manufactu.*, 98–108, 1998.

78. J. Taniguchi, X. Wang, M. Gen and T. Yokota, Hybrid genetic algorithm with fuzzy logic controller for obstacle location-allocation problem, *IEEE Trans. Electron., Informat. Syst.*, **124**(10): 2027–2033, 2004.

79. M. Gen, F. Altiparamk, and L. Lin, A genetic algorithm for two-stage transportation problem using priority-based encoding, *OR Spectrum*, **28**(3): 337–354, 2006.

80. F. Altiparmak, M. Gen, L. Lin, and T. Paksoy, A genetic algorithm approach for multi-objective optimization of supply chain networks, *Comp. Industr. Engineer.*, **51**(1): 197–216, 2006.

81. F. Altiparmak, M. Gen, L. Lin, and I. Karaoglan, A steady-state genetic algorithm for multi-product supply chain network design, *Computers Industr. Engineer.*, In press.

82. L. Lin, M. Gen, and X. Wang, Integrated multistage logistics network design by using hybrid evolutionary algorithm, *Comput. Indust. Engineer.*, In press.

83. P. Kampstra, R. D. Mei, and A. E. Eiben, Evolutionary computing in telecommunication network design: a survey, 2006. Available: http://www.math.vu.nl/~mei/articles/2006/kampstra/art.pdf.

84. M. Gen and M. Yoo, Real time tasks scheduling using hybrid genetic algorithm, in *Computational Intelligence in Multimedia Processing*, Ella-Aboul Hassanien (ed.), Berlin: Springer Verlag, 2007.

85. M. Gen and Y. S. Yun, Soft computing approach for reliability optimization: state-of-the-art survey, *Reliabil. Engineer. Syst. Safety*, **91**(9): 1008–1026, 2006.

FURTHER READING

L. Fogel, A. Owens, and M. Walsh, *Artificial Intelligence through Simulated Evolution*, New York: John Wiley & Sons, 1966.

J. R. Koza, *Genetic Programming*, Cambridge, MA: MIT Press, 1992.

J. R. Koza, *Genetic Programming II*, Cambridge, MA: MIT Press, 1994.

S. Kobayashi, Foundations of genetic algorithms and its applications, *Communications of ORSJ*, **45**: 256–261, 1993.

B. Sendhoff, M. Kreuts, and W. Seelen, A condition for the genotype phenotype mapping: casualty, *Proc. 7th Inter. Conf. on GAs, San Francisco, CA*, 1997, pp. 354–361.

L. Davis, ed., *Handbook of Genetic Algorithms*, New York: Van Nostrand Reinhold, 1991.

B. Julstrom, What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm, *Proc. 6th Inter. Conf. on Genetic Algorithms,* San Francisco, CA, 1995, pp. 81–87.

J. Horn, N. Nafpliotis, and D. Goldberg, A niched pareto genetic algorithm for multiobjective optimization, *Proc. 1st IEEE Conf. on Evolutionary Computation*, 1994, pp. 82–87.

T. Murata, H. Ishibuchi, and H. Tanaka, Multiobjective genetic algorithm and its application to flowshop scheduling, *Comput. Industr. Engineering*, **30**(4): 957–968, 1996.

D. Goldberg and J. Richardson, Genetic algorithms with sharing for multimodal function optimization, *Proc. 2nd Inter. Conf. on Genetic Algorithms*, 1987, pp. 41–49.

C. Fonseca and P. Fleming, Genetic algorithms for multiobjective optimization: formulation, discussion and generalization, *Proc. 5th Inter. Conf. on Genetic Algorithms*, 1993, pp. 416–423.

N. Srinivas and K. Deb, Multiobjective function optimization using nondominated sorting genetic algorithms, *Evolutionary Computation*, **3**: 221–248, 1995.

M. Gen, K. W. Kim, and G. Yamazaki, Project scheduling using hybrid genetic algorithm with fuzzy logic controller in SCM Environment, *J. Tsinghua Sci. Technol.*, **8**(1): 19–29, 2003.

MITSUO GEN
LIN LIN
Waseda University
Kitakyushu, Japan

G

GRANULAR COMPUTING

INTRODUCTION

Granular computing (GrC) is a term coined in 1997 as the name of an emerging and fast-growing research area in computer science and related fields (1,2). In its short history of 10 years, we have already witnessed a rapid development and extensive results (1,3–15).

A granule, the basic notion of granular computing, may be interpreted as one of the numerous small particles forming a larger unit. Collectively, they provide a representation of the unit with respect to a particular level of granularity. The central idea of the new paradigm of granular computing is the conceptualization and problem solving at different levels of granularity (12). On the one hand, one focuses on the suitable level of relevant conceptualizations without considering irrelevant lower level details. On the other hand, one changes granularity at different stages of problem solving.

The ideas of granular computing (i.e., problem solving under different granularity) have been explored in many fields, such as artificial intelligence, interval analysis, quantization, rough set theory, Dempster–Shafer theory of belief functions, divide and conquer, cluster analysis, machine learning, programming, databases, and many others (13,14). Although the subject matters and detailed formulations are different, the philosophy and the fundamental principles remain the same. The main objectives of granular computing are therefore to extract the commonality from a diversity of fields and to study systematically and formally such domain-independent principles (15,16). In particular, three perspectives of granular computing have been identified and studied (16). From the philosophical perspective, granular computing is a way of structured thinking (17). From the methodological perspective, granular computing is a general method of structured problem solving (15,16,18). From the computational perspective, granular computing is a new paradigm of structured information processing (3,4).

Granular computing is used as an umbrella term to cover theories, methodologies, techniques, and tools that make use of granules in problem solving (19). However, it should not be viewed as a simple collection of isolated, independent, or loosely connected pieces, nor as a simple restatement of existing results. One needs to re-examine, re-evaluate, reformulate, summarize, synthesize, combine, and extend results from existing studies in a unified framework. The introduction of granular computing provides such a broader context in which one can examine the inherent connections between concrete models and extract the abstract ideas and fundamental principles. Granular computing aims at a wider holistic view of problem solving, in contrast to narrow and fragmented views.

Bohm and Peat argued that science must go beyond a fragmented view of nature (20). Their argument is applicable to the study of granular computing. Therefore, we introduce and formulate granular computing as a way of thinking and a general method of problem solving that embraces a variety of concrete theories and methods.

EXEMPLAR MODELS OF GRANULAR COMPUTING

Historically speaking, the explicit consideration of granular computing is from the studies of the theories of fuzzy sets and rough sets (1,3,21–23). The concept of granular computing is developed based on the notion of information granulation first discussed by Zadeh in 1979 (24). Unfortunately, not much attention has been paid to information granulation until the publication of a seminal paper in 1997 (14). The attention to granular computing is also generated, to a large extent, by studies of rough set theory (25,26). It is through the study of this concrete model that one gains appreciation for the potential usefulness of granular computing in general (1,23).

Granular Computing in Fuzzy Set Theory

In his 1997 paper, Zadeh discussed a general framework of granular computing within the fuzzy set theory (14). Granules are constructed and defined based on the concept of generalized constraints. Relationships between granules are represented in terms of fuzzy graphs or fuzzy if-then rules. The associated computation method is known as computing with words (27).

Let X be a variable taking values in a universe U. A generalized constraint on the values of X can be expressed as $X\ isr\ R$, where R is a constraining relation, isr is a variable copula, and r is a discrete variable whose value defines the way in which R constrains X. Examples of constraints are equality, possibilistic, probabilistic, fuzzy, and veristic constraints. For example, an equality constraint, $r = e$, is given by $X\ ise\ R$, which means $X = R$. A possibilistic constraint, $r = blank$, is given by $X\,is\,R$, where R is a possibility distribution of X. With the introduction of generalized constraints, a granule is defined by a fuzzy set:

$$G = \{X | X\, isr\, R\} \tag{1}$$

Depending on the types of constraints, various classes of granules can be obtained. From simple granules, one may obtain Cartesian granules by considering combinations of constraints (14).

One may label granules by natural language words, which establishes a basis for computing with words. As one of the core components of fuzzy logic, computing with words deals with fuzzy if-then rules of the form:

$$if\, X\, isr_1\, A\, then\, Y\, isr_2\, B \tag{2}$$

where r_1 and r_2 may represent different types of constraints, although the same type is commonly used. A set

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

of fuzzy if-then rules can be interpreted in terms of a fuzzy graph. Inference can be carried out using fuzzy if-then rules or fuzzy graphs (14,27).

More results on granular computing using fuzzy sets can be found in, for example, references (1–3,6,8,,9,11,21,28).

Granular Computing in Rough Set Theory

Rough set theory is another generalization of classic sets based on the notion of indiscernibility (25,26). Granulation is a consequence of indiscernibility of objects. The loss of information through granulation implies that some subsets of the universe can only be approximately described.

Let $E \subseteq U \times U$ denote an equivalence relation on the universe U. The pair $apr = (U, E)$ is called an approximation space. The equivalence relation E partitions the set U into disjoint subsets known as the quotient set U/E. Each equivalence class may be viewed as a granule consisting of indistinguishable elements. It is also referred to as an equivalence granule. A particular semantic interpretation of equivalence relations is provided based on the notion of information tables. Two objects are equivalent if they have exactly the same value with respect to a set of attributes. Thus, an equivalence granule is characterized by the equality constraint (29).

An arbitrary set $X \subseteq U$ may not necessarily be a union of some equivalence classes, which implies that one may not be able to describe X precisely using the equivalence classes of E. In this case, one may characterize X by a pair of lower and upper approximations:

$$\begin{aligned} \underline{apr}(X) &= \bigcup_{[x]_E \subseteq X} [x]_E, \\ \overline{apr}(X) &= \bigcup_{[x]_E \cap X \neq \varnothing} [x]_E \end{aligned} \tag{3}$$

where $[x]_E = \{y | xEy\}$ is the equivalence class containing x. The lower approximation $\underline{apr}(X)$ is the union of all the equivalence granules that are subsets of X. The upper approximation $\overline{apr}(X)$ is the union of all the equivalence granules that have a nonempty intersection with X.

Based on the approximations of sets, one may perform data analysis and data mining tasks in information tables, such as attribute reduction, dependency analysis, and learning of decision rules (23). Many proposals have been made regarding granular computing within rough set theory. More results can be found in Refs. 1,5,13,23,29–33.

Granular Computing in a Wider Context

Granular computing using the theories of fuzzy and rough sets is restricted to a set-theoretic setting, where a granule is a crisp or fuzzy subset of a universe. Another set-theoretic model of granular computing is neighborhood systems, in which an element is associated with a family of neighborhoods (21,22,30,34,35). Although rough set theory considers partitions consisting of nonoverlapping granules, neighborhood systems can deal with both nonoverlapping and overlapping granules.

There is a need to study granular computing in broader contexts by moving beyond the set-theoretic setting (13,15–17). One may treat a granule as an abstract notion to be concretized in a particular domain. A granule is a small particle of a whole unit. Different sized granules lead to different levels of details, which, in turn, enables us to represent the whole using multiple levels, multiple resolutions, or hierarchies. Granular computing is, therefore, viewed as a way of thinking using multilevel granularity.

Depending on particular problems, one may consider granulated theories, granulated maps, granulated solutions, granulated plans, and so on. By considering granular computing in a wider context, one can extract the basic principles from a diversity of fields, including concept formation, clustering, abstraction, machine learning, data mining, programming, theorem proving, and many more (13,15,16). It is within this wider context that one can appreciate the power, effectiveness, flexibility, and general applicability of granular computing as a way of thinking and as a general method of problem solving.

GRANULAR COMPUTING AS A WAY OF THINKING

Problem solving in general is an extremely complex process and involves many different techniques. It might be difficult to give a universally effective method or to design a set of precise instructions for problem solving. Nevertheless, the basic processes and the systematic ways of thinking are common elements of problem solving, regardless of any particular problem to be solved. From the philosophical and conceptual points of view, granular computing concerns a way of thinking that underlies human problem solving. It is based on our perception of the world in multiple levels of granularity and our ability to solve a problem with differing granularity.

Granular Computing Models Human Problem Solving

Zadeh identified three basic and closely related concepts that underlie human cognition, namely, granulation, organization, and causation (14). Granulation decomposes the whole unit into parts (granules), organization integrates parts into the whole unit, and causation involves association of causes and effects. Yager and Filev argued that humans have developed a granular view of the world and objects that we perceive, measure, conceptualize, and reason are granular (28). It is evident that granulation plays a central role in human perception and problem solving.

Granular computing, therefore, reflects naturally the ways in which humans granulate information and reason with it. In fact, models of granular computing are the formalization of ideas and principles of human problem solving. The effectiveness, flexibility, and adaptivity of human problem solving suggest that such a formulation is rational and may lead to useful problem-solving theories and tools.

The basic ideas and principles of granular computing have been investigated under different names such as abstraction and granularity in artificial intelligence. Hobbs proposed a theory of granularity (12). The theory is motivated by the fact that humans view the world under various grain sizes and abstract only those things relevant to the present interests (12). Human intelligence and flexibility, to a large degree, depend on the ability to conceptualize the world at different granularity and to switch granularity. With the theory of granularity, we can map the complexities of the real world around us into simpler theories that are computationally tractable to reason in. Based on similar motivations, Giunchigalia and Walsh proposed a theory of abstraction (36). Abstraction can be thought of as the process that allows us to consider relevant materials and to forget irrelevant details that would get in the way of what we are trying to do. The theory of abstraction may be viewed as a model of granular computing that subsumes many existing studies.

Granular Computing is Motivated by Practical Needs

Human problem-solving skills may be considered as the result of a long time adaptation to the environments. The practical reasons that motivate human adaptation also motivate the study of granular computing.

In many situations, when a problem involves incomplete, uncertain, or vague information, it may be difficult to differentiate distinct elements and one is forced to consider granules (23,25,26). Both theories of fuzzy and rough sets can be interpreted based on the similarity of objects. They are motivated by the practical needs to describe physically existing and ill-defined granules.

In some situations, although detailed information may be available, it may be sufficient to use granules to have an efficient and practical solution. In fact, very precise solutions may not be required at all for many practical problems. It may also happen that the acquisition of precise information is too costly, and coarse-grained information reduces cost (14). These observations suggest a basic guiding principle of fuzzy logic: "*Exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution cost, and better rapport with reality*" (14). This principle offers a more practical philosophy for real-world problem solving. Instead of searching for the optimal solution, one may search for good approximate solutions. One only needs to examine the problem at a finer granulation level with more detailed information when there is a need or benefit for doing so (19).

Through granulation and abstraction, irrelevant details are filtered out, which may enable us to observe high-level structures and organizations that may not be easily seen otherwise. Granulation leads to a high-level organization, which provides a remedy for our inability to grasp every detailed aspect of a large problem. For example, the granulation of ideas in a scientific paper produces an organization in terms of title, keywords, abstract, section headings, and subsection headings, which greatly improves the readability as well as our understanding of the paper (37).

The necessity of information granulation, as well as the simplicity and efficiency derived from information granulation, may account for the popularity of granular computing.

Granular Computing is Consistent with the Organization of Knowledge

Every concept is understood as a unit of thought consisting of two parts, the intension and the extension of the concept (38–40). The intension of a concept consists of all properties or attributes that are valid for all those objects to which the concept applies. The extension of a concept is the set of objects or entities that are instances of the concept. All objects in the extension have the same properties that characterize the concept. In other words, the intension of a concept is an abstract description of common features or properties shared by elements in the extension, whereas the extension consists of concrete examples of the concept. A concept is thus described jointly by its intension and extension. This formulation enables us to study concepts in a logic setting in terms of intensions and also in a set-theoretic setting in terms of extensions. The descriptions of granules characterize concepts from the intension point of view, whereas granules themselves characterize concepts from the extension point of view. Through the connections between extensions of concepts, one may establish relationships between concepts (41,42).

In characterizing human knowledge, one needs to consider two topics, namely, context and hierarchy (43). Knowledge is contextual and hierarchical. A context in which concepts are formed provides meaningful interpretation of the concepts. Knowledge is organized in a tower or a partial ordering. The base-level, or first-level, concepts are the most fundamental concepts, and higher level concepts depend on lower level concepts. To some extent, granulation and inherent hierarchical granulation structures reflect naturally the way in which human knowledge is organized. The construction, interpretation, and description of granules and granulations are of fundamental importance in the understanding, representation, organization, and synthesis of data, information, and knowledge.

GRANULAR COMPUTING AS A GENERAL METHOD OF PROBLEM SOLVING

The underlying ideas of granular computing have been used either explicitly or implicitly for solving a wide diversity of problems. To illustrate its effectiveness and flexibility as a general method of problem solving, a few examples are discussed.

One-Dimensional Granulation

In problem solving, it is common to represent a certain physical property based on a quantitative measure. A quantitative measure may be viewed as a homomorphism from a set of objects to the set of real numbers (44). The set of real numbers is a linear order under the relation $\leq$. Granular computing based on the granulation of a linear order is therefore useful in many applications.

Suppose $(L, \preceq)$ is linearly ordered set with a linear order $\preceq$. A useful granulation of L is given by considering intervals of L. Given two elements $a, b \in L$, a closed interval of L is defined by:

$$[a,b] = \{x \in L | a \preceq x \preceq b\} \tag{4}$$

which is a subset of L. One can lift operations on L to operations on intervals of L based on the concept of power algebras (19,45). Let $'$ be a unary operation and $\circ$ a binary operation on L. The lifted operations on intervals of L, also denoted by $'$ and $\circ$, are defined by:

$$\begin{aligned} [a,b]' &= \{x' | x \in [a,b]\}, \\ [a,b] \circ [c,d] &= \{x \circ y | x \in [a,b], y \in [c,d]\} \end{aligned} \tag{5}$$

for intervals $[a,b]$ and $[c,d]$. In general, the lifted operations on intervals may not be closed. That is, they may not produce intervals of L. Based on the relation $\preceq$, we can define four relations on intervals:

$$\begin{aligned} &[a,b]_* \preceq_* [c,d] \Leftrightarrow \forall x \in [a,b] \, \forall y \in [c,d] x \preceq y, \\ &[a,b]_* \preceq^* [c,d] \Leftrightarrow \forall x \in [a,b] \, \exists y \in [c,d] x \preceq y, \\ &[a,b]^* \preceq_* [c,d] \Leftrightarrow \exists x \in [a,b] \, \forall y \in [c,d] x \preceq y, \\ &[a,b]^* \preceq^* [c,d] \Leftrightarrow \exists x \in [a,b] \, \exists y \in [c,d] x \preceq y \end{aligned} \tag{6}$$

for intervals $[a,b]$ and $[c,d]$. Additional relations can be defined on intervals by considering $[a,b]$ and $[c,d]$ as two subsets of L. For example, we say that the two intervals overlap if $[a,b] \cap [c,d] \neq \varnothing$ and disjoin otherwise. Similarly, $[a,b]$ is a subinterval of $[c,d]$ if $[a,b] \subseteq [c,d]$.

Concrete models of granular computing based on 1-D granulation are interval analysis (46), temporal granulation, and reasoning (47–51).

1-D granulations can be easily extended into partially ordered sets, such as Boolean algebras and lattices. Interval set algebra is an example of such an extension (52).

High-Dimensional Granulation

1-D granulations can be extended to 2-D granulations by considering a pair of linear orders $(L_1, \preceq_1)$ and $(L_2, \preceq_1)$. In this case, we have the Cartesian product $L_1 \times L_2$. Corresponding to an interval, we define a granule as a rectangle in the 2-D space. If the 2-D space is an Euclidean space, we can define a distance function between two points. A granule can also be a circle with a particular radius.

Spatial granulation and reasoning is an example of 2-D granulation (48,53). Another example of 2-D granulation is the hierarchical coding and progressive transmission of images (55).

The idea of extending a 1-D granulation to a 2-D granulation can be applied to study high-dimensional granulations.

Top-Down Programming

The top-down programming is an effective technique to deal with the complex problem of programming, which is based on the notions of structured programming and stepwise refinement (55). The principles and characteristics of the top-down design and stepwise refinement, as discussed by Ledgard, et al. (55), provide a convincing demonstration that granular computing is a general method of problem solving.

According to Ledgard et al. (55), the top-down programming approach has the following characteristics:

Design in Levels. A level consists of a set of modules. At higher levels, only a brief description of a module is given. The details of a module are to be refined, divided into smaller modules, and developed in lower levels.

Initial Language Independence. The initial levels focus on expressions that are relevant to the problem solution, without explicit reference to machine- and language-dependent features.

Postponement of Details to Lower Levels. The higher levels concern critical and broad issues and the structure of the problem solution. The details such as the choice of specific algorithms and data structures are postponed to lower levels.

Formalization of Each Level. Before proceeding to a lower level, one needs to obtain a formal and precise description of the current level, which ensures a full understanding regarding the structure of the current sketched solution.

Verification of Each Level. The sketched solution at each level must be verified so that errors pertinent to the current level will be detected.

Successive Refinements. Top-down programming is a successive refinement process. Starting from the top level, each level is redefined, formalized, and verified until one obtains a complete program.

In terms of granular computing, program modules correspond to granules, and levels of the top-down programming correspond to different levels of granularity. One can immediately see that those characteristics also hold for granular computing in general.

Top-down programming offers a general top-down problem-solving method, which may be considered as the core of granular computing. The hierarchical organization of human knowledge makes the top-down approach an effective way of problem solving.

By observing the systematic way of top-down programming, some authors suggest that the similar approach can be used in developing, teaching, and communicating mathematical proofs (56,57). Leron proposed a structured method for presenting mathematical proofs (57). The main objective is to increase the comprehensibility of mathematical presentations and at the same time retain their rigor. The traditional linear fashion presents a proof step-by-step from hypotheses to conclusion. In contrast, the structured method arranges the proof in levels and proceeds in a top-down manner. Like the top-down, stepwise refinement programming approach, a level consists of short autonomous modules, each embodying one major idea of the proof to be further concretized in the subsequent levels. The top level is a very general description of the main line of the proof. The second level elaborates on the generalities of the top level by supplying proofs of unsubstantiated statements, details of general descriptions, and so on. For some more complicated tasks, the second level only gives a brief description and the details are postponed to the lower levels. The process continues by supplying more details of the higher levels until a complete proof is reached.

TWO BASIC ISSUES OF GRANULAR COMPUTING

The two related basic issues of granular computing are granulation and computing with granules (13,19). The former deals with the formation, representation, and interpretation of granules, whereas the latter deals with the use of granules in problem solving. They can be studied from the semantic and algorithmic aspects, respectively (4,19).

Semantic Studies versus Algorithmic Studies

The interpretation of granules focuses on the semantic side of granule constructions. It addresses the question of *why* two objects are put into the same granule. Typically, elements in a granule are drawn together by indistinguishability, similarity, proximity, or functionality (14). Furthermore, information granulation depends on the available knowledge. In the construction of granules, it is necessary to study criteria for deciding whether two elements should be put into the same granule based on available information. In other words, one must provide necessary semantic interpretations for notions such as indistinguishability, similarity, and proximity. It is also necessary to study granulation structures derivable from various granulations of the universe (13). The formation and representation of granules deal with algorithmic issues of granule construction. They address the problem of *how* to put two objects into the same granule. Algorithms need to be developed for constructing granules efficiently.

Computation with granules can be similarly studied from both the semantic and algorithmic aspects. On the one hand, one needs to interpret various relationships between granules such as closeness, dependency, and association, and to define and interpret operations on granules. On the other hand, one needs to design methodologies and tools for computing with granules such as approximation, reasoning, and inference.

Both the semantic and algorithmic aspects of granular computing are important. However, many existing methods of granular computing do not pay enough attention to the semantic aspect. It is equally, if not more, important to investigate semantic issues involved in granular computing. The results may provide not only interpretations and justifications for a particular granular computing model, but also guidelines that prevent possible misuses of the model. The results from algorithmic study may lead to efficient and effective granular computing methods and tools.

Granulation

The notion of granulation can be studied in many different contexts. A family of granules collectively is referred to as a granulation of a problem. The granulation of a problem, particularly the semantics of granulation, is domain- and application-dependent. Nevertheless, one can still identify some domain-independent issues (13).

Granulation Criteria. A granulation criterion deals with the semantic interpretation of granules and addresses the question of *why* two objects are put into the same granule.

Granulation Structures. It is necessary to study granulation structures derivable from various granulations of the universe. Two structures can be observed: the structure of individual granules and structure of a granulation. Multilevel granulations produce a natural hierarchical structure (53,58–60).

Granulation Methods. A granulation method addresses the problem of *how* to put two objects into the same granule. The construction process can be modeled as either top-down or bottom-up. A top-down process divides large granules, whereas a bottom-up process combines smaller granules. Both processes lead naturally to a hierarchical organization of granules and granulations (33,58).

Representation/Description of Granules. Another semantics-related issue is the interpretation of the results of a granulation method. Once constructed, it is necessary to describe, name, and label granules using certain languages.

Quantitative Characteristics of Granules and Granulations. One can associate quantitative measures to granules and granulations to capture their features.

These issues can be understood by examining a concrete example of granulation known as the cluster analysis (61), which can be done by simply changing granulation into clustering and granules into clusters. *Clustering structures* may be hierarchical or nonhierarchical, exclusive or overlapping. Typically, a similarity or distance function is used to define the relationships between objects. *Clustering criteria* may be defined based on the similarity or distance function and the required cluster structures. For example, one would expect strong similarities between objects in the same cluster and weak similarities between objects in different clusters. Many *clustering methods* have been proposed and studied, including the families of hierarchical agglomerative, hierarchical divisive, iterative partitioning, density search, factor analytic, clumping, and graph theoretic methods (62). Cluster analysis can be used as an exploratory tool to interpret data and find regularities from data (61). This process requires the active participation of experts to interpret the results of clustering methods and judge their significance. A good *representation* of clusters and their *quantitative characterizations* may make the task of exploration much easier.

Computing and Reasoning with Granules

A granulated view summarizes available information and knowledge about a problem. As a basic task of granular computing, one can examine and explore further relationships between granules at a lower level and relationships between granulations at a higher level (13).

Mappings Between Different Level of Granulations. In a granulation hierarchy, the connections between different levels of granulations can be described by mappings. Giunchglia and Walsh considered an abstraction as a mapping between a pair of formal systems in the development of a theory of abstraction (36). A mapping links different representations of the same problem at different levels of detail. One can classify and study different types of granulations by focusing on the properties of the mappings (36).

Granularity Conversion. A basic task of granular computing is to change views with respect to different levels of granularity. As we move from one level of detail to another, we need to convert the representation of a problem accordingly (36,49). A move to a more detailed view may reveal information that otherwise cannot be seen, whereas a move to a simpler view can improve the high-level understanding by omitting irrelevant details of the problem (12,14,36,49,60).

Property Preservation. Granulation allows different representations of the same problem in different levels of detail. It is naturally expected that the same problem must be consistently represented (49). A granulation and its related computing methods are meaningful only if they preserve certain desired properties (36,60).

Operators. The relationship between granules at different levels and conversion of granularity can be precisely defined by operators (49,59). They serve as the basic building blocks of granular computing. There are at least two types of operators that can be defined. One type deals with the shift from a fine granularity to a coarse granularity. A characteristic of such an operator is that it will discard certain details, which makes distinct objects no longer differentiable. Depending on the context, many interpretations and definitions are available, such as abstraction, simplification, generalization, coarsening, zooming-out, and so on (12,36,51,59,60,63,64). The other type deals with the change from a coarse granularity to a fine granularity. A characteristic of such an operator is that it will provide more details so that a group of objects can be further classified. They can be defined and interpreted differently, such as articulation, specification, expanding, refining, zooming-in, and so on (2,36,51,59,60,63,64). Other types of operators may also be defined. For example, with the granulation, one may not be able to exactly characterize an arbitrary subset of a fine-grained universe in a coarse-grained universe, which leads to the introduction of approximation operators in rough set theory (26,52).

Granular computing methods describe our ability to switch granularity in problem solving. Detailed and domain-specific methods can be developed by elaborating these issues with explicit reference to an application.

CONCLUSION

Granular computing is introduced from two perspectives: as a way of thinking and as a general method of problem solving. The former perspective concerns the philosophical investigation and conceptual formulation. The results suggest that a general method for problem solving can be described based on granular computing.

The introduction of granular computing provides a unified and general framework to integrate a number of fragmentary studies that either explicitly or implicitly adopt similar or the same ideas and principles. The fields that have strong influences on granular computing are the theories of fuzzy and rough sets, cognitive science, and artificial intelligence.

The subject of granular computing can be studied by using its own principles, namely, formulation and investigation at different levels of granularity. We focus on a high-level examination of granular computing, although some details are discussed. The significance of granular computing lies in its basic principles that are common to problem solving.

BIBLIOGRAPHY

1. T. Y. Lin, Y. Y. Yao, and L. A. Zadeh (eds.), *Rough Sets, Granular Computing and Data Mining*, Heidelberg: Physica-Verlag, 2002.
2. L. A. Zadeh, Some reflections on soft computing, granular computing and their roles in the conception, design and utilization of information/intelligent systems, *Soft Computing*, **2**: 23–25, 1998.
3. A. Bargiela, and W. Pedrycz, *Granular Computing: An Introduction*, Boston, MA: Kluwer Academic Publishers, 2002.
4. A. Bargiela, and W. Pedrycz, The roots of granular computing, *Proc. 2006 IEEE International Conference on Granular Computing*, Atlanta, 2006, pp. 806–809.
5. X. H. Hu, Q. Liu, A. Skowron, T. Y. Lin, R. R. Yager, and B. Zhang, (eds.), *Proc. 2005 IEEE International Conference on Granular Computing*, Beijing, 2005.
6. M. Inuiguchi, S. Hirano, and S. Tsumoto, (eds.), *Rough Set Theory and Granular Computing*, Berlin: Springer, 2003.
7. Journal of Nanchang Institute of Technology, special issue of *The Proceedings of the International Forum on Theory of GrC from Rough Set Perspective*, 2006.
8. W. Pedrycz, (ed.), *Granular Computing: An Emerging Paradigm*, Berlin: Springer-Verlag, 2001.
9. G. Wang, Q. Liu, Y. Y. Yao, and A. Skowron, (eds.), *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, LNAI 2639, Berlin: Springer, 2003.
10. Y. Q. Zhang, and T. Y. Lin, (eds.), *Proc. of the 2006 IEEE International Conference on Granular Computing*, Atlanta, 2006.
11. N. Zhong, A. Skowron, and S. Ohsuga (eds.), *New Directions in Rough Sets, Data Mining, and Granular-Soft Computing*, LNAI 1574, Berlin: Springer, 1999.
12. J. R. Hobbs, Granularity, *Proc. Ninth Internation Joint Conference on Artificial Intelligence*, Los Angeles, 1985, pp. 432–435.
13. Y. Y. Yao, A partition model of granular computing, *LNCS Trans. Rough Sets*, **1**: 232–253, 2004.
14. L. A. Zadeh, Towards a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic, *Fuzzy Sets and Systems*, **19**: 111–127, 1997.
15. Y. Y. Yao, Perspectives of granular computing, *Proc. 2005 IEEE International Conference on Granular Computing*, Vol. 1, Beijing, 2005, pp. 85–90.
16. Y. Y. Yao, Three perspectives of granular computing, *J. Nanchang Instit. Technol.*, **25**: 16–21, 2006.
17. Y. Y. Yao, Granular computing, *Comp. Sci. (Ji Suan Ji Ke Xue)*, **31**: 1–5, 2004.
18. Y. Y. Yao, The art of Granular computing, *Rough Sets and Intelligent System Paradigms*, LNAI, Berlin: Springer, 2007, 101–112.
19. Y. Y. Yao, Granular computing: basic issues and possible solutions, *Proc. 5th Joint Conference on Information Sciences*, Atlantic City. NJ, 2000, pp. 186–189.
20. D. Bohm, and F. D. Peat, *Science, Order, and Creativity*, 2nd ed., London: Routledge, 2000.

21. T. Y. Lin, From rough sets and neighborhood systems to information granulation and computing in words, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1997, pp. 1602–1606.

22. T. Y. Lin, and C. J. Liau, Granular computing and rough sets, in O. Maimon, and L. Rokach (eds.), *The Data Mining and Knowledge Discovery Handbook*, Berlin: Springer, 2005, pp. 535–561.

23. Z. Pawlak, Granularity of knowledge, indiscernibility and rough sets, *Proc. 98 IEEE International Conference on Fuzzy Systems*, Anchorage, AK, 1998, pp. 106–110.

24. L. A. Zadeh, Fuzzy sets and information granularity, in N. Gupta, R. Ragade, and R. Yager, (eds.), *Advances in Fuzzy Set Theory and Applications*, Amsterdam: North-Holland, 1979, pp. 3–18.

25. Z. Pawlak, Rough sets, *Int. J. Comp. Inform. Sci.*, **11**: 341–356, 1982.

26. Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data*, Boston, MA: Kluwer Academic Publishers, 1991.

27. L. A. Zadeh, Fuzzy logic = computing with words, *IEEE Transactions on Fuzzy Systems*, **4**: 103–111, 1996.

28. R. R. Yager, and D. Filev, Operations for granular computing: mixing words with numbers, *Proc. 1998 IEEE International Conference on Fuzzy Systems*, Anchorage, AK, 1998, pp. 123–128.

29. Y. Y. Yao, and N. Zhong, Granular computing using information tables, in T. Y. Lin, Y. Y. Yao, and L. A. Zadeh, (eds.), *Data Mining, Rough Sets and Granular Computing* , Heidelberg: Physica-Verlag, 2002, pp. 102–124.

30. T. Y. Lin, Granular computing on binary relations I: data mining and neighborhood systems, II: rough set representations and belief functions, in A. Skowron, and L. Polkowski, (eds.), *Rough Sets in Knowledge Discovery 1*, Heidelberg: Physica-Verlag, 1998, pp. 107–140.

31. L. Polkowski, and A. Skowron, Towards adaptive calculus of granules, *Proc. 1998 IEEE International Conference on Fuzzy Systems*, Anchorage, AK, 1998, pp. 111–116.

32. A. Skowron, and J. Stepaniuk, Information granules: Towards foundations of granular computing, *Int. J. Intell. Sys.*, **16**: 57–85, 2001.

33. Y. Y. Yao, Information granulation and rough set approximation, *Int. J. Intell. Sys.*, **16**: 87–104, 2001.

34. T. Y. Lin, Granular computing: structures, representations, applications and future directions, *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, LNAI 2639, Berlin: Springer, 2003, 16–24.

35. Y. Y. Yao, Granular computing using neighborhood systems, in R. Roy, T. Furuhashi, and P. K. Chawdhry, (eds.), *Advances in Soft Computing: Engineering Design and Manufacturing*, London: Springer-Verlag, 1999, pp. 539–553.

36. F. Giunchglia, and T. Walsh, A theory of abstraction, *Artif. Intell.*, **56**: 323–390, 1992.

37. Y. Y. Yao, Granular computing for the design of information retrieval support systems, in W. Wu, H. Xiong, and S. Shekhar, (eds.), *Information Retrieval and Clustering*, Dordrecht, The Netherlands: Kluwer Academic Publishers, 2004, pp. 299–329.

38. J. F. Sowa, *Conceptual Structures, Information Processing in Mind and Machine*, Reading, MA: Addison-Wesley, 1984.

39. I. van Mechelen, J. Hampton, R. S. Michalski, and P. Theuns, (eds.), *Categories and Concepts, Theoretical Views and Inductive Data Analysis*, New York: Academic Press, 1993.

40. R. Wille, Concept lattices and conceptual knowledge systems, *Comput. Mathemat. Appl.*, **23**: 493–515, 1992.

41. Y. Y. Yao, Modeling data mining with granular computing, *Proc. 25th Annual International Computer Software and Applications Conference*, Chicago, 2001, pp. 638–643.

42. Y. Y. Yao, A step towards the foundations of data mining, in B. V. Dasarathy, (ed.), *Data Mining and Knowledge Discovery: Theory, Tools, and Technology V*, The International Society for Optical Engineering, 2003, pp. 254–263.

43. L. Peikoff, *Objectivism: the Philosophy of Ayn Rand*, New York: Dutton, 1991.

44. F. S. Roberts, *Measurement Theory*, Reading, MA: Addison-Wesley, 1979.

45. C. Brink, Power structures, *Algebra Universalis*, **30**: 177–216, 1993.

46. R. E. Moore, *Interval Analysis*, Englewood NJ: Prentice-Hall, Cliffs, 1966.

47. J. F. Allen, Maintaining knowledge about temporal intervals, *Comm. ACM,* **26**: 832–843, 1983.

48. C. Bettini, and A. Montanari, (eds.), *Spatial and Temporal Granularity: Papers from the AAAI Workshop*, Technical Report WS-00–08, Menlo Park, CA: The AAAI Press, 2000.

49. L. Zhang, and B. Zhang, The quotient space theory of problem solving, *Fundamenta Informatcae*, **59**: 287–298, 2004.

50. J. Euzenat, Granularity in relational formalisms - with application to time and space representation, *Computat. Intell.*, **17**: 703–737, 2001.

51. K. Hornsby, Temporal zooming, *Trans. GIS*, **5**: 255–272, 2001.

52. Y. Y. Yao, Two views of the theory of rough sets in finite universes, *Int. J. Approximat. Reas.*, **15**: 291–317, 1996.

53. J. G. Stell, and M. F. Worboys, Stratified map spaces: a formal basis for multi-resolution spatial databases, *Proc. 8th International Symposium on Spatial Data Handling*, Vancouver, 1998, pp. 180–189.

54. A. Lippman, and W. Butera, Coding image sequences for interactive retrieval, *Comm. ACM*, **32**: 852–860, 1989.

55. H. F. Ledgard, J. F. Gueras, and P. A. Nagin, *PASCAL with Style: Programming Proverbs*, Rechelle Park, NJ: Hayden Book Company, Inc., 1979.

56. M. Friske, Teaching proofs: A lesson from software engineering, *Amer. Mathemat. Monthly*, **92**: 142–144, 1995.

57. U. Leron, Structuring mathematical proofs, *Amer. Mathemat. Monthly*, **90**: 174–185, 1983.

58. N. Jardine, and R. Sibson, *Mathematical Taxonomy*, New York: Wiley, 1971.

59. G. McCalla, J. Greer, J. Barrie, and P. Pospisil, Granularity hierarchies, *Comp. Mathemat. Appl.*, **23**: 363–375, 1992.

60. B. Zhang, and L. Zhang, *Theory and Applications of Problem Solving*, Amsterdam: North-Holland, 1992.

61. M. R. Anderberg, *Cluster Analysis for Applications*, New York: Academic Press, 1973.

62. M. S. Aldenderfer and R. K. Blashfield, *Cluster Analysis*, London: Sage Publications, The International Professional Publishers, 1984.

63. G. Shafer, *A Mathematical Theory of Evidence*, Princeton, NJ: Princeton University Press, 1976.

64. Y. Y. Yao, C.-J. Liau, and N. Zhong, Granular computing based on rough sets, quotient space theory, and belief functions, *Foundations of Intelligent Systems*, LNAI 2871, Berlin: Springer, 2003, 152–159.

YIYU YAO
University of Regina
Regina, Saskatchewan,
Canada
NING ZHONG
Maebashi Institute of
Technology
Maebashi-City, Japan

HOPFIELD NEURAL NETWORKS

The development of artificial neural networks has been motivated by the desire to find improved methods of solving problems that are difficult for traditional computing software or hardware. The success of early neural networks led to the claim that they could solve virtually any type of problem. Although this claim was quickly shown to be overly optimistic, research continued during the 1970s into the use of neural networks, especially for pattern association problems. The early 1980s marked the beginning of renewed widespread interest in neural networks. A key player in the increased visibility of, and respect for, neural networks is physicist John Hopfield of the California Institute of Technology. Together with David Tank of AT&T, Hopfield developed a group of recurrent networks that are known as Hopfield neural networks (HNNs). The first of these, the discrete Hopfield neural network (DHNN), was designed as content addressable memory (CAM). The continuous Hopfield neural network (CHNN) can also serve as a CAM, but it is most widely used for combinatorial optimization problems.

One reason that Hopfield's work caught the attention of the scientific community, and the public, was the close connection between the models and the successful development of neural network chips by researchers at AT&T and by Carver Mead and his coworkers. Hopfield's emphasis on practical implications made the engineering connection very strong. By making explicit the relationship between the HNN and electrical circuits, Hopfield opened the field of neural networks to an influx of physical theory. Although many concepts incorporated in the HNN had antecedents in earlier neural network research, Hopfield and Tank brought them together with both clear mathematical analysis and strong emphasis on practical applications (1).

ARTIFICIAL NEURAL NETWORKS

An artificial neural network (ANN) approach to problem solving is inspired by certain aspects of biological nervous systems. An ANN is composed of a large number of very simple processing elements (neurons). The neurons are interconnected by weighted pathways. The pattern of connection among the neurons is called the network architecture. At any time, a neuron has a level of activity, which it communicates to other neurons by sending it as a signal over these pathways. As the weights on the pathways contain much of the important information in the network, the information is distributed, rather than localized, as in traditional computers.

Architectures

One of the most basic distinctions between different types of neural networks is based on whether the network architecture allows for feedback among the neurons. A fully interconnected recurrent network is shown in Fig. 1.

Weights

In addition to the design of the ANN architecture, a major consideration in developing a neural network is the determination of the connection weights. For many networks, this is done by means of a training phase, in which known examples of the desired input–output patterns are presented to the network and the weights are adjusted according to a specified training algorithm. This is especially typical of feed-forward networks. In the standard Hopfield networks, the weights are fixed when the network is designed.

Network Operation

To use a neural network, after the weights are set, an input pattern is presented and the output signal of each neuron is adjusted according to the standard process for the specific ANN model. In general, each neuron sends its output signal to the other neurons to which it is connected; the signal is multiplied by the weight on the connection pathway; each neuron sums its incoming signals. Each neuron's output signal is a nonlinear function of its summed input. In a feed-forward network, these computations are performed one layer at a time, starting with the input units, and progressing through the network to the output units. For a recurrent network, such as an HNN, the updating of each neuron's activity level continues until the state of the net (the pattern of activations) converges. The process differs for the discrete and continuous forms of HNN; before discussing the details, we summarize the primary types of applications for which HNNs are used.

APPLICATIONS OF HOPFIELD NEURAL NETWORKS

Memory in biological systems is fundamentally different than in a traditional digital computer, in which information is stored by assigning an address, corresponding to a physical location, where the data are written. On the other hand, your memory of an event is a combination of many sights, sounds, smells, and so on. The idea of associative memory came from psychology rather than from engineering, but during the 1970s, much of the neural network research (especially work by James A. Anderson at Brown University and Teuvo Kohonon at the University of Helsinki) focused on the development of mathematical models of associative (or content addressable) memory. The use of an energy function analysis facilitates the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

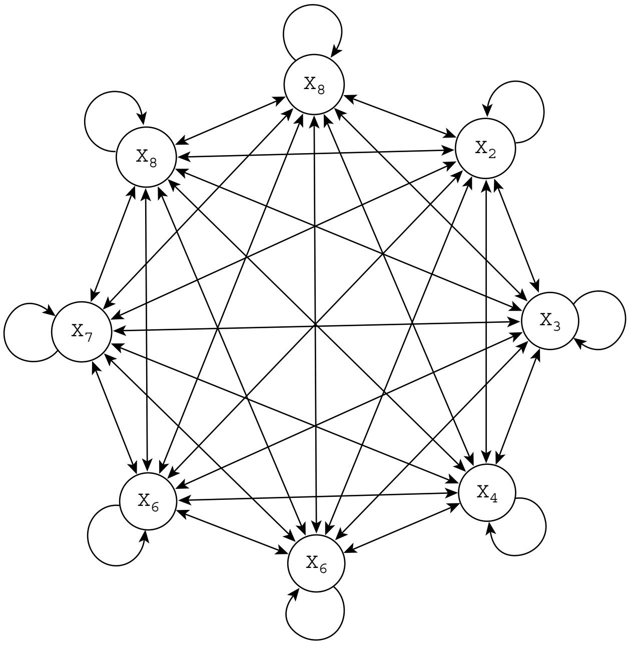


Figure 1. A fully interconnected network allows signals to flow between neurons.

understanding of associative memories that can be constructed as electronic “collective-decision circuits” (2).

The process used by biological systems to solve optimization problems also differs from that used in traditional computing techniques. Although no claim is made that neural network approaches to optimization problems directly model the methods used by biological systems, ANNs do have some potential advantages over traditional techniques for certain types of optimization problems. ANNs can find near-optimal solutions quickly for large problems. They can also handle situations in which some conditions are desirable but not absolutely required. Neural network solutions (and, in particular, HNN) have been investigated for many applications because of their potential for parallel computation and computational advantage when they are implemented with analog very large-scale integration (VLSI) techniques.

Many other forms of recurrent neural networks have also been developed. Networks with specific recurrent structure are used for problems in which the signal varies with time. Neural networks for the study of learning, perception, development, cognition, and motor control also use recurrent structures.

Associative Memory

One important use of an HNN is as an autoassociative memory, which can store (or memorize) a certain number of patterns. When a modified form of one of the stored patterns is presented as input, the HNN can recall the original pattern after a few iterations.

Before the weights of an associative memory neural net are determined, the patterns to be stored must be converted to an appropriate representation for computation. Usually each pattern is represented as a vector with components that are either 0 or 1 (binary form) or ±1 (bipolar form); the bipolar form is often computationally preferable for associative memory applications. The same representation is also used for patterns that are presented to the network for recognition.

Optimization

The second primary area of application for HNNs is combinatorial optimization problems. The use of a continuous HNN for solving optimization problems was first illustrated for the traveling salesman problem (TSP), a well-known but difficult optimization problem (3) and a task assignment problem (2). Since then, HNNs have been applied to optimization problems from many areas, including game theory, computer science, graph theory, molecular biology, VLSI computer-aided design, reliability, and management science. Many examples are included in Ref. 4.

The HNN approach is based on the idea that the network weights and other parameters can be found from an energy function; the network configuration (pattern of neuron activations) that produces a minimum of the energy function corresponds to the desired solution of the optimization problem. The appropriate choice of energy function for a particular problem has been the subject of much research.

DISCRETE HOPFIELD NETWORKS

The iterative autoassociative network developed by Hopfield (5,6) is a fully interconnected neural network, with symmetric weights and no self-connections, i.e., $w_{ij} = w_{ji}$ and $w_{ii} = 0$. In a DHNN, only one unit updates its activation at a time (this update is based on the signals it receives from the other units). The asynchronous updating of the units allows an energy (or Lyapunov) function to be found for the network. The existence of such a function forms the basis for a proof that the net will converge to a stable set of activations.

Operation

The primary considerations in using a DHNN are determining the network weights and updating the activations.

Setting the Weights. The earliest version of the DHNN used binary input vectors; later descriptions are often based on bipolar inputs. The weight matrix to store a pattern, represented as the column vector, $\mathbf{p} = (p_1,\ldots, p_i,\ldots p_n)^T$ is the matrix $\mathbf{P}\,\mathbf{P}^T - \mathbf{I}$. The matrix $\mathbf{pp}^T$ is known as the outer or matrix product of the vectors $\mathbf{p}$ and $\mathbf{p}^T$. Subtracting the identity matrix has the effect of setting the diagonal entries to 0, which is necessary to allow the network to reconstruct one of the stored patterns when a degraded or noisy form of the pattern is presented as input. The weight matrix $\mathbf{W}$ in which several patterns are stored is the sum of the individual matrices generated for each pattern.

Updating the Activations. To use a DHNN to recall a stored pattern, an input stimulus pattern $\mathbf{x}$ is presented to the network (one component to each neuron). Typically, the input is similar to one of the stored memories. Each neuron transmits its signal to all of the other neurons. The signal received by the ith neuron is $\sum_j x_j w_{ji}$; by the symmetry of the weights, this is also the ith row of the product $\mathbf{Wx}$. One neuron, chosen at random, updates its activation. Its activation is 1 if the signal it received was non-negative, i.e., if $\sum_j x_j w_{ji} \geq 0$; the activation is -1 if $\sum_j x_j w_{ji} < 0$. The new pattern is again broadcast to all neurons, and another neuron is chosen to update its activation. The process continues until the network reaches a stable state, a configuration of activations that does not change.

Example. To illustrate the use of a DHNN, consider the following simple example, adapted from Ref. 7. Suppose we wish to store the three bipolar patterns:

$$\begin{aligned} p_1 &= (\ 1 \quad 1 \quad 1 \quad 1 \quad 1)^{\mathrm{T}} \\ p_2 &= (\ 1 \ -1 \ -1 \quad 1 \ -1)^{\mathrm{T}} \\ p_3 &= (-1 \quad 1 \ -1 \ -1 \ -1)^{\mathrm{T}} \end{aligned}$$

The weight matrix to store these three patterns is $\mathbf{W}_1 + \mathbf{W}_2 + \mathbf{W}_3 = \mathbf{W}$:

$$\begin{array}{ccccc} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{array} + \begin{array}{ccccc} 0 & -1 & -1 & 1 & -1 \\ -1 & 0 & 1 & -1 & 1 \\ -1 & 1 & 0 & -1 & 1 \\ 1 & -1 & -1 & 0 & -1 \\ -1 & 1 & 1 & -1 & 0 \end{array}$$

$$+ \begin{array}{ccccc} 0 & -1 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 & -1 \\ 1 & -1 & 0 & 1 & 1 \\ 1 & -1 & 1 & 0 & 1 \\ 1 & -1 & 1 & 1 & 0 \end{array} = \begin{array}{ccccc} 0 & -1 & 1 & 3 & 1 \\ -1 & 0 & 1 & -1 & 1 \\ 1 & 1 & 0 & 1 & 3 \\ 3 & -1 & 1 & 0 & 1 \\ 1 & 1 & 3 & 1 & 0 \end{array}$$

We present as an input (or probe) vector $\mathbf{x} = (1, -1, -1, 1, 1)^T$, which differs from the second stored pattern in only the last component. To update the network, compute $\mathbf{Wx} = (4, -3, 4, 4, -2)^T$. If the third neuron is chosen, its activation will change from -1 to 1, because it received a signal of 4. Using the updated vector of activations, $(1, -1, 1, 1, 1)^T$ gives $\mathbf{Wx} = (6, 0, 4, 6, 4)^T$. If neuron 1, 3, 4, or 5 is chosen, its activity will not change, and eventually neuron 2 will be chosen. As we are using the convention that a neuron's activity is set to 1 if it receives a non-negative signal, neuron 2 will change its activation and the updated vector of activations becomes $(1, 1, 1, 1, 1)^T$, which is the first stored pattern, but not the stored pattern that is most similar to the probe. If the fifth neuron had been chosen for the first update (instead of the third neuron), the network would have reached the second stored pattern immediately.

This example illustrates both the operation of a DHNN for use as an associative memory and some of the issues that must be considered. These include questions concerning the circumstances under which convergence to a stable state is guaranteed, the question as to whether that stable state will be one of the stored patterns (and if so, will it be the closest pattern to the input?), and the relationship between the number of stored memories and the ability of the network to recall the patterns with little or no error.

Issues

The primary issues concerning the use of a DHNN are convergence and storage capacity.

Convergence. For any iterative process, it is important to understand its convergence characteristics. It can be shown that the general DHNN will converge to a stable limit point (pattern of activation of the units) by considering an energy function for the system. An energy function is a function that is bounded below and is a nonincreasing function of the state of the system. For a neural network, the state of the system is the vector of activations of the units. Thus, if an energy function can be found for an iterative neural network, the ANN will converge to a stable set of activations.

The general DHNN allows an external signal y_i to be maintained during processing, so that the total signal received by neuron X_i is $y_i + \sum_j x_j w_{ji}$. The threshold for determining whether a neuron is ON or OFF may be set to any desired constant θ_i; when chosen to update its activation, a unit will set its activation to ON if

$$y_i + \sum_j x_j w_{ji} \geq \theta_i$$

a unit will set its activation to OFF if

$$y_i + \sum_j x_j w_{ji} < \theta_i.$$

An energy function for the general DHNN described here is given by

$$\mathrm{E} = -0.5 \sum_{i \neq j} \sum_j x_i x_j w_{ij} - \sum_i x_i y_i + \sum_i \theta_i x_i \qquad (1)$$

If the activation of the net changes by an amount Δx_i, the energy changes by the corresponding amount

$$\Delta E = -\left[y_i + \sum_{i \neq j} x_j w_{ij} - \theta_i \right] \Delta x_i \qquad (2)$$

To show that $\Delta E \leq 0$, consider the two cases in which the activation of neuron X_i will change.

1. If X_i is ON, it will turn OFF if $y_i + \sum_j x_j w_{ji} < \theta_i$. This gives a negative change for x_i. As the quantity $\left[y_i + \sum_{i \neq j} x_j w_{ij} - \theta_i \right]$ in the expression for ΔE is also negative, we have $\Delta E < 0$.
2. On the other hand, if X_i is OFF, it will turn ON if $y_i + \sum_j x_j w_{ji} > \theta_i$. This gives a positive change for x_i. As $\left[y_i + \sum_{i \neq j} x_j w_{ij} - \theta_i \right]$ is positive in this case, the result is again that $\Delta E < 0$.

Therefore, the energy cannot increase. As the energy is bounded, the net must reach a stable equilibrium where the energy does not change with further iteration. This proof uses the fact that the energy change only depends on the change in activation of one unit, and that the weight matrix is symmetric. Setting the diagonal weights to 0 corresponds to the assumption that biological neurons do not have self-connections. From a computational point of view, zeroing out the diagonal makes it more likely that the network will converge to one of the stored patterns, rather than simply reproducing the input pattern.

Storage Capacity. In addition to knowing under what circumstances a Hopfield network is guaranteed to converge, it is also useful to understand how many patterns may be stored in, and recalled from, such a network. Although more patterns may be stored if the pattern vectors are orthogonal, that structure cannot be assumed in general. Therefore, most results are based on the assumption that the patterns to be stored are random. Hopfield found experimentally that P, the number of binary patterns that can be stored and recalled with reasonable accuracy, is given (approximately) by $P = 0.15\,n$, where n is the number of neurons. For a similar DHNN, using bipolar patterns, it has been found (7) that $P = \frac{n}{2\log_2 n}$.

CONTINUOUS HOPFIELD NETWORK

In contrast to the discrete form, the activations of the neurons in a continuous Hopfield net can take on a continuous range of values (most often between 0 and 1). The network dynamics are specified by differential equations for the change in activations. These differential equations are intimately connected to the underlying energy function for the network.

For a CHNN, we denote the internal activity of a neuron as u_i; its output signal is $v_i = g(u_i)$, where g is a monotonically nondecreasing function of the input signal received by unit U_i. Most commonly g is taken to be the sigmoid function $v = 0.5\ (1 + \tanh(\alpha\ u))$, which has range (0, 1). The parameter α controls the steepness of the sigmoid. The differential equations governing the change in the internal activity of each unit are closely related to the energy function that will be minimized as the network activations evolve. Either the evolution equation or the energy function may be specified and the other relationship derived from it. A standard form for the energy function is

$$E = 0.5 \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\, v_i v_j + \sum_{i=1}^{n} \theta_i\, v_i \tag{3}$$

the corresponding evolution equation is

$$\frac{d}{dt} u_i = -\frac{\partial E}{\partial v_i} = -\sum_{j=1}^{n} w_{ij}\, v_j - \theta_i \tag{4}$$

CHNNs that are used to solve constrained optimization problems have several standard characteristics. Each unit represents a hypothesis; the unit is ON if the hypothesis is true and OFF if the hypothesis is false. The weights are fixed to represent both the constraints of the problem and the function to be optimized. The solution of the problem corresponds to the minimum of the energy function. Each unit's activation evolves so that the energy function decreases.

In the next sections, we illustrate the use of CHNN for constraint satisfaction and constrained optimization, first for a very simple example, and then for the well-known N-queens and TSP problems.

Simple Example

To introduce the use of a CHNN, consider the network shown in Fig. 2, in which it is desired to have exactly one unit ON. The weights must be chosen so that the network dynamics correspond to reducing the energy function.

To have a network that converges to a pattern of activations that solves a specified problem, it is common to design the energy function so that its minimum will be achieved for a pattern of activations that solves the given problem. For this example, the energy function might be formulated as

$$E = \left[1 - \sum_i v_i\right]^2$$

so that its minimum value (0) is achieved when exactly one of the units is ON and the other two units each have activation of zero. Expanding the energy equation

$$E = 1 - 2v_1 - 2v_2 - 2v_3 + v_1^2 + v_1 v_2 + v_1 v_3 + v_2 v_1 + v_2^2 + v_2 v_3 + v_1 v_3 + v_2 v_3 + v_3^2$$

and comparing it with the standard form given in Equation 3, shows that $\theta_i = -2$, and $w_{ij} = 1$. (Note that there is a self-connection on each unit; this does not interfere with the convergence analysis for a CHNN.) The energy function could also be scaled by a positive constant factor, if desired.

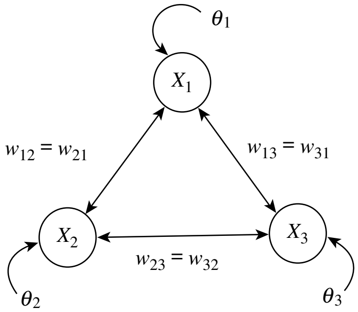


Figure 2. A simple Hopfield network to illustrate the interrelationship between the weights and the energy function.

The differential equations governing the change in the internal activity u_i for each neuron are given by

$$\frac{d}{dt}u_i = -\frac{\partial E}{\partial v_i} = 2[1-(v_1+v_2+v_3)]$$

The *N*-queens Problem

The problem of how to place 8 queens on an 8-by-8 chessboard in mutually nonattacking positions was proposed in 1848, and it has been widely studied since then. It is used as a benchmark for many methods of solving combinatorial optimization problems. In a neural network approach, one neuron is used for each square on the chessboard. The activation of the neuron indicates whether a queen is located on that square. As a queen commands vertically, horizontally, and diagonally, only one queen should be present on any row or column of the board. The arrangement of the neurons for a smaller 5-queens problem is shown in Fig. 3. For simplicity, connection pathways are shown for only one unit. To implement the energy function and evolution equations given below, the units in each row and each column are fully interconnected; similarly the units along each diagonal and each antidiagonal are also fully interconnected (4).

One example of a valid solution to the 5-queens problem is represented by the network configuration in which neurons U_{15}, U_{23}, U_{31}, U_{44}, and U_{52} are ON, and all others are OFF.

The constraints are as follows:

(a) One and only one queen is placed in each row.
(b) One and only one queen is placed in each column.
(c) At most one queen is placed on each diagonal.

An energy function can be constructed for this problem, as follows:

$$\begin{aligned} E = {} & \frac{C_1}{2}\sum_x\sum_i\sum_{j\neq i} V_{xi}V_{xj} + \frac{C_2}{2}\sum_i\sum_x\sum_{y\neq x} V_{xi}V_{yi} \\ & + \frac{C_3}{2}\sum_x\left[\sum_i V_{xi}-1\right]^2 + \frac{C_4}{2}\sum_i\left[\sum_x V_{xi}-1\right]^2 \\ & + \frac{C_5}{2}\sum_x\sum_i\sum_{1\leq x+k;i+k\leq N}(V_{xi}V_{x+k;i+k}) \\ & + \frac{C_6}{2}\sum_x\sum_i\sum_{1\leq x+k;i-k\leq N}(V_{xi}V_{x+k;i-k}) \end{aligned} \tag{5}$$

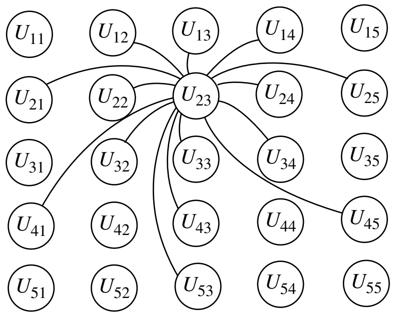


Figure 3. The arrangement of neurons for a 5-queens problem. Connection pathways are shown only for unit U_{23}.

(The inner summation in the last two terms runs over all values of k such that $x + k$ and $i + k$ are both between 1 and N.)

The first constraint is represented by the first and third terms in the energy function; the second constraint is represented by the second and fourth terms in the energy function; and the third constraint is represented by the fifth and sixth terms in the energy function (one term for the diagonal and one for the anti-diagonal).

The corresponding motion equation for unit U_{xi} is

$$\begin{aligned} \frac{dU_{xi}}{dt} = {} & -C_1\sum_{j\neq i}V_{xj} - C_2\sum_{y\neq x}V_{yi} - C_3\left[\sum_i V_{xi}-1\right] \\ & -C_4\left[\sum_x V_{xi}-1\right] - C_5\sum_x\sum_{1\leq x+k;i+k\leq N}V_{x+k;i+k} \\ & -C_6\sum_{1\leq x+k;i-k\leq N}V_{x+k;i-k} \end{aligned} \tag{6}$$

For further discussion of CHNN solutions to this problem, see Refs. 4 and 8.

The Traveling Salesman Problem

The TSP is a well-known example of a class of computationally hard problems for which the amount of time required to find an optimal solution increases exponentially as the problem size increases. In the TSP, every city in a given set of n cities is to be visited once and only once. A tour may begin with any city, and it ends by returning to the initial city. The goal is to find a tour that has the shortest possible length.

With a Hopfield network, the TSP is represented by an n-by-n matrix of neurons in which the rows of the matrix represent cities and the columns represent the position in the tour when the city is visited. For example, if unit U_{24} is ON for the TSP, it indicates that the second city is visited as the fourth stop on the tour. A valid solution is achieved when the network reaches a state of a permutation matrix, i.e., exactly one unit on in each row and each column. The arrangement of the neurons for a five-city TSP is shown in Fig. 4, with connection pathways shown only for unit U_{23}. A widely used energy function for the

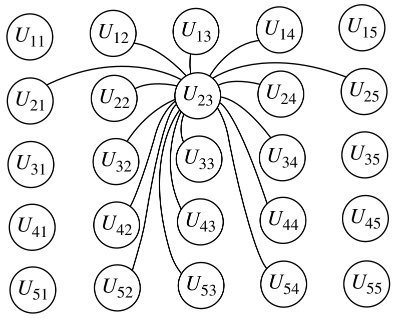


Figure 4. The arrangement of neurons for a five-city traveling salesman problem. Connection pathways are shown only for unit U_{23}.

TSP is

$$E = \frac{C_1}{2}\sum_x\sum_i\sum_{j\neq i}V_{xi}V_{xj} + \frac{C_2}{2}\sum_i\sum_x\sum_{y\neq x}V_{xi}V_{yi} + \frac{C_3}{2}\sum_x\left(\sum_i V_{xi} - 1\right)^2 + \frac{C_4}{2}\sum_i\left(\sum_x V_{xi-1}\right)^2 + \frac{C_5}{2}\sum_x\sum_{y\neq x}\sum_i D_{xy}V_{xi}(V_{y,i+1} + V_{y,i-1}) \qquad (7)$$

The first four terms in the energy function represent the validity constraints: The first term is minimized (zero) if each city is visited at most once. Similarly, the second term is zero if at most one city is visited at each stage in the tour. The third and fourth terms encourage each row and column in the network matrix to have one neuron ON. The fifth term gives the value of the corresponding tour length. This term represents the TSP objective function. It is desired to make its value as small as possible while maintaining the validity of the tour.

To guarantee convergence of the network, the motion dynamics are obtained from the energy function according to the relationship

$$\begin{aligned} du_{xi}/dt &= -\partial E/\partial V_{xi} \\ &= -C_1\sum_{j\neq i}V_{xj} - C_2\sum_{y\neq x}V_{yi} - C_3\left(\sum_j V_{xj} - 1\right) \\ &\quad -C_4\left(\sum_y V_{yi} - 1\right) - C_5\sum_{y\neq x}d_{xy}(V_{y,i+1} + V_{y,i-1}) \end{aligned} \qquad (8)$$

where the internal activation u and the output signal v for any unit are related by the sigmoidal function $v = 0.5\,(1 + \tanh(\alpha\, u))$.

For simulations, each neuron is updated using Euler's first-order difference equation:

$$u_{xi}(t + \Delta t) = u_{xi}(t) + \left(\frac{du_{xi}}{dt}\right)\Delta t$$

The neurons' activations are initialized with random values, and the activations are allowed to evolve according to the governing equations for the network dynamics. The activations are updated iteratively until the network converges; the final configuration of activations gives the network's solution to the TSP.

The choice of network parameters has a significant effect on the quality of solutions obtained. The relative sizes of the coefficients in the energy equation influence the network to either emphasize valid tours (at the expense of tour length) or to seek short tours (which may not be valid). A very steep sigmoid function may force the network to converge quickly (but not necessarily to a good solution), whereas a shallow slope on the sigmoid may result in the final activations not being close to 0 or 1.

Simulation Results. The energy function in the original presentation of a Hopfield network solution of the TSP was given as

$$E = \frac{A}{2}\sum_x\sum_i\sum_{j\neq i}v_{xi}v_{xj} + \frac{B}{2}\sum_i\sum_x\sum_{y\neq x}v_{xi}v_{yi} + \frac{C}{2}\left[N - \sum_x\sum_i v_{xi}\right]^2 + \frac{D}{2}\sum_x\sum_{y\neq x}\sum_i d_{xy}v_{xi}(v_{y,i+1} + v_{y,i-1}) \qquad (9)$$

The third term in this form of the energy function encourages N neurons to be on, but it does not try to influence their location. The original differential equation for the activity of unit U_{xi} was given by

$$\frac{d}{dt}u_{xi} = -\frac{u_{xi}}{\tau} - A\sum_{j\neq i}V_{xj} - B\sum_{y\neq x}v_{yi} + C\left[N - \sum_x\sum_i v_{xi}\right] - D\sum_{y\neq x}d_{xy}(v_{y,i+1} + v_{y,i-1}). \qquad (10)$$

The first term on the right-hand side of this equation is a decay term, which can be motivated by analogy to electrical circuits, but it does not have a corresponding term in the energy equation. The parameter values that Hopfield and Tank used, namely,

$$A = B = 500;\ C = 200, D = 500, N = 15, \alpha = 50, \text{ and } \tau = 1$$

give very little emphasis to the decay term, so the lack of corresponding energy term has relatively little significance. The parameter N must be taken to be larger than the actual number of cities in the problem to counterbalance the continuing inhibitory effect of the distance term; as the minimum of the distance component of the energy function is positive, the corresponding term in Equation (10) acts to try to turn a unit OFF even when there are no constraint violations.

Although Hopfield and Tank (3) reported a very high rate of success in finding valid tours (16/20 trials) with about one half of the trials producing one of the two shortest tours, other researchers have been unable to match these results. The coordinates of the Hopfield and Tank 10-city test problem were generated randomly; the same locations have been used as a benchmark for other neural network solutions. Many variations have been investigated, including alternative energy functions, methods of choosing parameter values, and procedures for setting the initial activations.

Wilson and Pawley (9) provide a detailed statement of the Hopfield–Tank algorithm, together with an analysis of their experiments. Using the Hopfield–Tank parameters, with $\Delta t = 10^{-5}$, they found 15 valid tours in 100 attempts; (45 froze and 40 failed to converge in 1000 epochs).

Wilson and Pawley tried several variations of the Hopfield and Tank algorithm, in attempting to obtain a success rate for valid tours that would approach that achieved by Hopfield and Tank. They experimented with different parameter values, different initial activity configurations, and imposing a large distance penalty for visiting the same city twice, none of which helped much. Fixing the starting city helped on the Hopfield–Tank cities, but not on other randomly generated sets of cities.

One variation that did improve the ability of the net to generate valid tours was a modification of the initialization procedure. The Willshaw initialization is based on the rationale that cities on opposite sides of a square probably should be on opposite sides of tour. The starting activity of each unit is biased to reflect this fact. Cities far from the center of the square received a stronger bias than those near the middle. The formula, in terms of the ith city and jth position, where the coordinates of the ith city are x_i, y_i:

$$\text{bias}(i,j) = \cos\left[\text{atan}\left(\frac{y_i - 0.5}{x_i - 0.5}\right) + \frac{2\pi(j-1)}{n}\right]$$
$$\sqrt{(x_i - 0.5)^2 + (y_i - 0.5)^2}$$

Although special analysis that relies on the geometry of the problem can improve the solution to the actual TSP, it does not generalize easily to other applications.

Issues

Proof of Convergence. For an energy function of the form of Equation (3), the Hopfield net will converge if the activations change according to the differential equation given in Equation (4), as the following simple calculations show. If $v_i = g(u_i)$ is monotonic nondecreasing, then $\dfrac{dv_i}{du_i} \geq 0$. As

$$\frac{dE}{dt} = \sum_i \frac{dv_i}{dt}\frac{\partial E}{\partial v_i} = -\sum_i \frac{dv_i}{dt}\frac{du_i}{dt} = -\sum_i \frac{dv_i}{du_i}\frac{du_i}{dt}\frac{du_i}{dt}$$

the energy is nonincreasing, as required.

In the original presentation of the CHNN (6), the energy function is

$$E = -0.5\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} v_i v_j - \sum_{i=1}^{n} \theta_i v_i + \frac{1}{\tau}\sum_{i=1}^{n}\int_0^{v_i} g_i^{-1}(v)\, dv$$

If the weight matrix is symmetric and the activity of each neuron changes with time according to the differential equation:

$$\frac{d}{dt}u_i = -\frac{u_i}{\tau} + \sum_{j=1}^{n} w_{ij} v_j + \theta_i \qquad (11)$$

the net will converge. The argument is essentially the same as above.

Note that the weights must be symmetric for the equations given here to be valid. This symmetry follows from the fact that connections in a standard Hopfield are bidirectional; i.e., the connection from unit i to unit j, and the connection from unit j to unit i are the same connection. Results for asymmetrical Hopfield networks are discussed below.

Choice of Coefficients. The relative importance assigned to each of the terms in the energy function plays a very important role in determining the quality of the solutions obtained. A variety of experimental investigations into the appropriate coefficients have been reported. Theoretical results have also been obtained; the choice of energy function coefficients is discussed further in the section on recent developments.

Local Minima. One shortcoming of the CHNN, as with any optimization procedure that always moves in the direction of improving the solution, is convergence to a local optima that is not the global optimum. A Boltzmann machine incorporates a simulated annealing process into the updates, so that early in the iterations, each unit has a fairly high probability of not updating its activation in the manner dictated by the equations given for the DHNN. As the iterations progress, the "temperature" of the network is reduced, and at a lower temperature, the units become more closely controlled by the updating equations (10,11).

RECENT DEVELOPMENTS

Hopfield neural networks are being used for applications in many areas. Recent developments of both theoretical and practical importance can be found in journals such as *Neural Information Processing—Letters and Reviews* and *IEEE Transactions on Neural Networks*, or in conference proceedings, either for meetings that focus on neural network applications or for gatherings of researchers in a particular specialty. In the next sections, we consider some directions in which the basic Hopfield neural network model is being generalized. Methods of adapting the weights in HNN, both for CAM and for optimization problems, are being developed. Investigation into HNN with nonsymmetric weights is giving theoretical results for conditions under which such a network are guaranteed to converge. Research also continues into the determination of the storage capacity of the DHNN.

Adaptive Weights

Much of the neural network research has focused on networks in which either the activities of the neurons evolve or the strengths of the synapses (weights) adapt, but not both. However, a complete model of a biological process requires dynamical equations for both to specify the behavior of the system. On the other hand, applications of Hopfield networks to constrained optimization problems repeatedly illustrate the importance and difficulty of determining the proper weights to assure convergence to a good solution. Progress is being made in both of these areas.

Learning Patterns. Dong (12) has developed an energy function for a system in which both activations and weights are adaptive and has applied it to the study of the development of synaptic connection in the visual cortex. His dynamical equations for the activity of the neurons are essentially the same as given in Equation (11). The adaptation of the weights follows a differential form of Hebbian learning, based on the "recent" correlation of the activities of the neurons that are on either end of the weighted pathway; this leads to Hebbian learning with a decay term. The weights remain symmetric throughout the process, so that the convergence analysis follows an energy function approach as described previously.

As a simple example, consider two neurons and the weight w on the connection path between them. Dong's dynamical equations for this illustrative special case are as follows:

$$\begin{aligned} a\frac{du_1}{dt} &= -u_1 + w\,v_1 \\ a\frac{du_2}{dt} &= -u_2 + w\,v_2 \\ v &= f(g\,u) \\ b\frac{ds}{dt} &= -s + v_1 v_2 \\ w &= f(h\,s) \end{aligned}$$

The function f is piecewise linear, with a range between -1 and 1; i.e., $f(x) = -1$ if $x \le -1$, $f(x) = x$ if $-1 < x < 1$, $f(x) = 1$ if $x \ge 1$.

The energy function is

$$E(v_1, v_2, w) = -w\,v_1\,v_2 + \frac{1}{2g}v_1^2 + \frac{1}{2g}v_2^2 + \frac{1}{2h}\,w^2$$

The origin (0, 0, 0) is a stable point, corresponding to unlearned connections and no neuron activity. If the constants g and h are greater than 1, the configurations (1, 1, 1), (-1, -1, 1), (1, -1, -1), and (-1, 1, -1) are stable points. Each of these configurations has the property, which holds in general for stable points, that the weight on the connection is sign($v_i v_j$). The training of the network is conducted by presenting each pattern to be learned as the external input signal for a brief period of time and cycling through the patterns until the weights have converged. The behavior of the system during learning depends on the strength of the external input to the system relative to the size of the weights between neurons. When the input signals dominate, the network can learn several input patterns; for weaker input signals, the network ultimately chooses only one of the patterns to memorize. These ideas provide the basis for a model of the first stage of cortical visual processing in mammals.

Constrained Optimization. The appropriate choice of the weights in a Hopfield net for constrained optimization has been the subject of much experimental work. It is well known that using larger values for the coefficients of the constraint terms helps guide the network toward valid solutions, but it may result in poor quality solutions. On the other hand, increasing the value of the coefficient of the objective term helps to improve the quality of a solution, but it may result in an invalid solution because of a constraint violation.

Recently, Park and Fausett (8) introduced a method for determining the coefficients of the energy function (and thereby the weights) adaptively as the network evolves. As the network evolves in the direction of minimization of the total energy, each term in the energy function competes with the other terms to influence the path to be followed. To find good coefficients for the energy function, the components of the energy are monitored, and the coefficients are adapted, depending on how far each component of the energy function is to its goal (minimum value), until a balanced relationship among the coefficients is reached. Using a steepest ascent procedure with normalization, the coefficients are updated after every epoch of iteration until they reach a state of near equilibrium. Although this approach may seem counter-intuitive at first, it has the desired effect of increasing the coefficients of those terms that are contributing the most to the value of the energy function. It is those terms that most need to be reduced during network iteration. The final coefficient values are used to set the weight connections, and the network is run again to solve the problem.

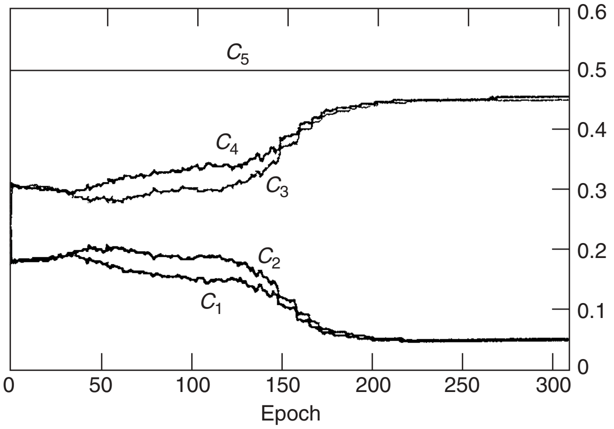


Figure 5. Evolution of coefficients on the constraint terms of the TSP; coefficient C_5 = 0.5.

A sample of the coefficient evolution for the 10-city TSP is illustrated in Fig. 5. In this example, the coefficient of the objective term (representing tour length) in the energy function is fixed as $C_5 = 0.5$; the other coefficients (on the constraint terms) evolve subject to the restriction that $C_1 + C_2 + C_3 + C_4 = 1$. When the network was rerun with the converged coefficients, 94% of the trials resulted in valid tours; the length of the generated tours ranged from 2.69 to 3.84, with a mean length of 2.83. The efficacy of this method is even more striking on larger problems. Although the results vary depending on the choice of the fixed value for the coefficient of the objective term, 20-city and 30-city problems (generated in a manner similar to that used by Hopfield and Tank for the 10-city problem) were successfully solved, with a high rate of valid solutions, for C_5 in the range of 0.2 to 0.5.

Storage Capacity

Another area of active research for Hopfield networks used as CAM is the storage capacity of the network. Many investigations are based on the assumption that the patterns are random (independent, identically distributed uniform random variables). The question is, how many patterns (vectors with components of $+1$ or -1) can be stored and retrieved (from a minor degradation of a stored pattern); a small fraction of errors may be allowed in the retrieved pattern. Hopfield suggested, based on numerical simulations, that P, the number of patterns that can be stored and retrieved, is given by $P = cn$, with $c = 0.15$.

More recently, martingale techniques have been applied (13) to a different joint distribution of the spins (patterns), extending the theoretical results to situations beyond those investigated previously (7). Assuming that the patterns have the same probability distribution, are orthogonal in expectation, and are independent, Francois shows that there are energy barriers (which depend on d, the acceptable fraction of errors in the reconstructed pattern) surrounding each memorized pattern. For almost perfect recall ($d = 1/n$), the storage capacity can be as large as $c = [2(1 + g) \ln n]^{-1}$ with $g > 2$. Other researchers have studied the effect of a noisy environment on the convergence of the network (13).

Stability Results

Investigations into the stability of more general Hopfield-type models have considered asynchronous updates for a continuous-valued, discrete-time Hopfield model (14). In general, stability arguments rely on sophisticated mathematical theory and are not easily summarized in a brief presentation.

One approach to the investigation of asynchronous updates is based on the Takeda–Goodman synchronous model:

$$x(k+1) = TF(x(k)) + (I - B)\,x(k) + u$$

where T is the interconnection matrix of the neural network (usually assumed symmetric), F is a diagonal nonlinear function (usually assumed monotonic, often sigmoidal), and u is a vector of inputs (assumed constant). With a few additional assumptions, the previous stability results have been extended by considering a class of desynchronizations (15).

Asymmetric Weights

The stability of asymmetric Hopfield networks is of practical interest, both for more general models (e.g., connectionist expert systems) and for the implementation of theoretically symmetric networks (because it is almost impossible to preserve the symmetry of the connections exactly in hardware).

Many results for nonsymmetric connections depend on the absolute value of the weights; however, these may be overly restrictive. For example, if $w_{ij} = -\,w_{ji}$ for all i, j, the network is absolutely stable, but results relying on absolute value considerations will not establish the fact. It has also been shown that if the largest eigenvalue of $\mathbf{W} + \mathbf{W}^T$ is less than 2, then the network is absolutely stable. A more convenient corollary of this result is that if

$$\sum_{i;j} (w_{ij} + w_{ji})^2 < 4$$

then the network is absolutely stable (16).

To study computational models based on asymmetric Hopfield-type networks, a classification theory for the energy functions associated with Hopfield networks has been introduced and convergence conditions deduced for several different forms of asymmetric networks. For example, two networks have been developed, using a triangular structure, to solve the maximum independent set of a graph problem. Although this problem can be solved with a standard Hopfield network, the triangular network is a more simple and efficient procedure. See Ref. 17 for details.

SUMMARY AND CONCLUSIONS

Hopfield neural networks comprise a rich and varied realm of the overall field of artificial neural networks. Applications can be found in many areas of engineering. Continuing investigation into the theoretical and practical considerations governing the convergence properties of the networks provides a firm foundation for the use of Hopfield models and their extension to more generalized settings. Work continues on differences in performance that may occur when the networks are implemented with fully parallel (asynchronous) updating of the activations.

BIBLIOGRAPHY

1. J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, Cambridge, MA: MIT Press, 1988.
2. D. W. Tank and J. J. Hopfield, Collective computation in neuronlike circuits, *Scientific American*, **257**: 104–114, 1987.
3. J. J. Hopfield and D. W. Tank, Computing with neural circuits: a model, *Science* **8**: 625–633, 1986.
4. Y. Takefuji, *Neural Network Parallel Computing*, Boston, MA: Kluwer Academic Publishers, 1992.
5. J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. of the Nat. Acad. of Sci.***79**: 2554–2558, 1982.
6. J. J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. of the Nat. Acad. of Sci.* **81**: 3088–3092, 1984.
7. R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, The capacity of the Hopfield associative memory, *IEEE Trans. on Inf. Theory*, IT-33, 461–482, 1987.
8. C. Y. Park and D. W. Fausett, Energy function analysis for improved performance of Hopfield-type neural networks, *Intell. Engineering Sys. Through Artificial Neural Net.*, **5**: 995–1000, 1995.
9. G. V. Wilson and G. S. Pawley, On the stability of the traveling salesman problem algorithm of Hopfield and Tank, *Biological Cybernetics*, **58**: 63–70, 1988.
10. E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, Chichester, England, U.K.: Wiley, 1989.
11. L. V. Fausett, *Fundamentals of Neural Networks*, Englewood Cliffs, NJ: Prentice Hall, 1994.
12. D. Dong, *Dynamic Properties of Neural Networks*, Ph.D. Dissertation, Pasadena, CA: California Institute of Technology, 1991.
13. O. Francois, New rigorous results for the Hopfield's neural network model, *Neural Networks*, **9**: 503–507, 1996.

14. S. Hu, X. Liao, and X. Mao, Stochastic Hopfield neural networks, *J. Physics A: Math. and Gen.*, 2235–2249, 2003.
15. A. Bhaya, E. Kaszkurewics, and V. S. Kozyakin, Existence and stability of a unique equilibrium in continuous-valued discrete-time asynchronous Hopfield neural networks, *IEEE Trans. on NN*, **7**: 620–628, 1996.
16. K. Matsuoka, Stability conditions for nonlinear continuous neural networks with asymmetric connection weights, *Neural Networks*, **5**: 495–500, 1992.
17. Z-B. Xu, G-Q. Hu, and C-P. Kwong, Asymmetric Hopfield-type networks: Theory and applications, *Neural Networks*, **9**: 483–501, 1996.

Laurene V. Fausett
Georgia Southern University
Statesboro, Georgia

INTELLIGENT AGENT

INTRODUCTION

“Agent” has different meanings in different contexts. In many dictionaries, there are about 10 items listed for “agent.” In computer science, an agent is referred to as a program that acts on a user or other programs, which is akin to its original meaning in dictionaries. If an agent can demonstrate some special features like learning, reasoning, and decision making, it may be called an intelligent agent. This article tries to provide an insight into what intelligent agent is for general readers in the context of computer science in general and artificial intelligence in particular.

The concept of agent can be traced back to the 1980s in the computer science and artificial intelligence communities. Agent systems are evolved from distributed artificial intelligence and distributed problem solving as well as parallel artificial intelligence (1). However, there is still no universally accepted definition of the term “agent.” One definition, which is adopted from Ref. 2, is attracting more and more attention. This definition states that, an intelligent agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives. This definition implies that the agent possesses the following minimal characteristics (30):

- Autonomy: Agents operate without the direct intervention of humans or others and have some kinds of control over their internal states.
- Social ability: Agents interact with other agents (and possibly humans) via some kinds of agent communication languages.
- Reactivity: Agents perceive their environment and respond in a timely fashion to changes that occur in it.
- Proactivity: Agents do not simply act in response to their environment; they can exhibit goal-directed behavior by taking the initiative.

Multiagent systems are systems composed of multiple interacting agents. Agents (adaptive or intelligent agents and multiagent systems) constitute one of the most prominent and attractive technologies in computer science at the beginning of this new century. Agent and multiagent system technologies, methods, and theories are currently contributing to many diverse domains. These domains include information retrieval, user interface design, robotics, electronic commerce, computer-mediated collaboration, computer games, education and training, smart environments, ubiquitous computers, and social simulation.

This technology, is not only very promising but also emerging as a new way of thinking, a conceptual paradigm for analyzing problems and for designing systems, for dealing with complexity, distribution, and interactivity, and perhaps a new perspective on computing and intelligence.

Agent-based computing has been a source of technologies to several research areas, both theoretical and applied. These areas include distributed planning and decision making, automated auction mechanisms, and learning mechanisms. Moreover, agent technologies have drawn from, and contributed to, a diverse range of academic disciplines, in the humanities, the sciences, and the social sciences. The fundamental research issues in agent technologies include multiagent planning, agent communication languages, coordination mechanisms, matchmaking architectures and algorithms, information agents and basic ontologies, sophisticated auction mechanism design, negotiation strategies, and learning.

Agent technologies are a natural extension of current component-based approaches and have the potential to greatly impact the lives and work of all of us. Accordingly, this area is one of the most dynamic and exciting in computer science today. Some application domains where agent technologies will play a crucial role, including (4,5) *Ambient Intelligence,* the seamless delivery of ubiquitous computing, continuous communications, and intelligent user interfaces to consumer and industrial devices; *Grid Computing*, where multiagent system approaches will enable efficient use of the resources of high-performance computing infrastructure in science, engineering, medical, and commercial applications; the *Semantic Web*, where agents are needed both to provide services and to make best use of the resources available, often in cooperation with others; and *Self-* and Autonomic Computing*: To configure and maintain complex computer systems, including IT infrastructure, it is highly required that those systems have self-awareness, self-organization, self-configuration, self-management, self-diagnosis, self correction, and self-repair capabilities. These self-* systems can be viewed as autonomous entities and components with interactions, which provides an application domain for research and development of agent technologies. In addition to these areas agent technologies will also be used in other fields like peer-to-peer computing, computational biology and bioinformatics, web services, and so on.

There are many different views about intelligent agents or multiagent systems. Two principal views are as follows: (1) agents as a paradigm for software engineering; and (2) agents as a tool for understanding human societies (6).

Software engineers have derived a progressively better understanding of the characteristics of complexity in software. It is now widely recognized that *interaction* is probably the most important characteristic of complex software. It is believed that, in the future, computation will be understood chiefly as a process of interaction. Just as we can understand many systems as being composed of essentially passive objects, which have a state and upon which we can perform operations, so we can understand many others as being made up of interacting, semiautonomous agents. This recognition has led to the growth of interest in agents as a new paradigm for software engineering.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

There are far too many variables and unknown quantities in human societies. We can do little about it such as predicting very broad but short-term trends, and even then the process is full of many errors. However, multiagent systems do provide an interesting and novel tool for simulating societies, which may help shed some light on various kinds of social processes.

There are a number of similarities between the object- and agent-oriented views of system development. For example, both emphasize the importance of interactions between entities. However, there are also significant differences between agents and objects (6–8).

- Agents embody a stronger notion of autonomy than objects, and in particular, they decide for themselves whether to perform an action on request from other agents.
- Agents are capable of flexible (reactive, proactive, social) behavior, and the standard object model has nothing to say about such types of behavior.
- A multiagent system is inherently multithreaded, in that each agent is assumed to have at least one thread of control.

Expert systems were the most important artificial intelligence technology of the 1980s. An expert system is one that is capable of solving problems or giving advice in some knowledge-rich domain. The main differences between agents and expert systems are as follows (6):

- Classic expert systems are disembodied—they are not coupled to any environment in which they act, but rather act through a user as a "middleman."
- Expert systems are not generally capable of reactive, proactive behavior.
- Expert systems are not generally equipped with social ability, in the sense of cooperation, coordination, and negotiation.

This article presents a tutorial overview of the field of intelligent agents in brief. Related issues are discussed, which include agent architecture, agent communication languages, agent-oriented software engineering, agent development tools, challenge issues in agent technologies, and so on.

ARCHITECTURES OF INTELLIGENT AGENTS

A wide range of system architecture exists for intelligent agents. This section introduces some representative architectures for intelligent agents, which include the BDI architecture (for individual agents) and open agent architecture (for multiagent systems).

BDI Architecture

The architecture for individual agents can be classified into three classes: deliberative, reactive, and hybrid agent architecture.

Deliberative agents assume an explicit symbolic model of the environment and the capability of logical reasoning as a basis for intelligent actions, and so they maintain the tradition of classic artificial intelligence. The modeling of the environment is normally performed in advance and forms the main component of the agent's knowledge base. Deliberative agents have as a second significant property, in addition to their internal symbolic environment model, their capability to make logical decisions. The agent, as part of the decision-making process, uses the knowledge contained in its model to modify its internal state. This internal state is the mental state and is composed of three components: belief, desire, and intention (BDI).

The procedural reasoning system (PRS) (9) was perhaps the first agent architecture to explicitly embody the BDI paradigm and has proved to be the most durable agent architecture developed to date. The PRS is often referred to as a BDI architecture, which is shown in Fig. 1.

Beliefs contain the fundamental views of an agent with regard to its environment. An agent uses them, in particular, to express its expectations of the possible future states. Desires are derived directly from beliefs. They contain an agent's judgments of future situations. Goals represent that a subset of an agent's desires on whose fulfillment it could act. In contrast to its desire, an agent's goal must be realistic and must not conflict with each other. Intentions are a subset of the goals. If an agent decides to follow a specific goal, this goal becomes an intention. Plans combine an agent's intentions into consistent units. There is a close connection between intentions and plans: Intentions constitute the subplans of an agent's overall plan, and, conversely, the set of all plans reflects the agent's intentions. The BDI model comes from research done in the field of artificial intelligence especially in common sense reasoning and planning over the past 20 years and has proven to be the most robust and flexible model for intelligent agent systems.

Several logical theories of BDI systems have been developed. Closely related to this work on BDI architectures is Shoham's proposal for *agent-oriented programming*, which is a multiagent programming model in which agents are explicitly programmed in terms of mentalistic notions such as belief and desire (10).

The best-known reactive agent architecture is the *subsumption architecture* (11). There are two defining characteristics of the subsumption architecture: (1) An agent's

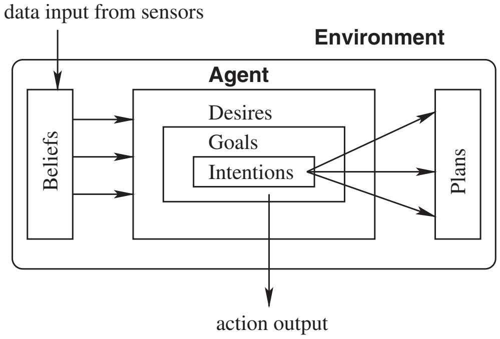


Figure 1. BDI architecture.

decision making is realized through a set of *task-accomplishing behaviors*, and (2) many behaviors can "fire" simultaneously.

For most problems, neither a purely deliberative architecture nor a purely reactive architecture is appropriate. Hybrid architectures are required, which are typically realized as a number of vertical or horizontal software layers. A typical example of hybrid agent architectures is InteRRaP (12).

Open Agent Architecture

Open agent architecture (OAA) developed by the SRI (Stanford Research Institute) is a research framework for constructing multiagent systems (13). This architecture makes it possible for software services to be provided through the cooperative efforts of distributed collections of autonomous agents. Communication and cooperation between agents are brokered by one or more *facilitators*, which are responsible for matching requests, from users and agents, with descriptions of the capabilities of other agents. Thus, it is not generally required that a requester (user or agent) know the identities, locations, or the number of other agents involved in satisfying a request. Facilitators are not viewed as centralized controllers, but rather as *coordinators*, as they draw on knowledge and advice from several different, potentially distributed, sources to guide their delegation choices.

OAA is structured so as to minimize the effort involved in creating new agents and "wrapping" legacy applications, written in various languages and operating on various platforms; to encourage the reuse of existing agents; and to allow for dynamism and flexibility in the makeup of agent communities. Distinct features of OAA as compared with related work include extreme flexibility in using facilitator-based delegation of complex goals, triggers, and data management requests; agent-based provision of multimodal user interfaces; and built-in support for including the user as a privileged member of the agent community.

COMMUNICATION LANGUAGES FOR INTELLIGENT AGENTS

Typically, applications containing multiple agents make use of an agent communication language (ACL); the idea is similar to a human society using a common language such as English. However, it is noted that agent-based applications can be (and have been) developed using traditional third-generation languages like Lisp, C, or Prolog and object-oriented languages such as Java, C++, and Smalltalk. Next, agents working together need to share a certain amount of foundational, "common" knowledge called *ontology* (14), in just the same way that humans do. There are also some current and emerging computing technologies such as client/server model and CORBA that lend themselves to supporting agent-based applications.

The two agent communication languages with broadest uptake are KQML (Knowledge Query and Manipulation Language) (15) and FIPA ACL (16). The most important difference between the two languages is in the collection of performatives they provide.

KQML was developed in the early 1990s as part of the U.S. government's ARPA Knowledge Sharing Effort and is a language and protocol for exchanging information and knowledge, which has been used extensively. KQML was conceived as both a message format and a message-handling protocol to support run-time knowledge sharing among agents. The KQML language can be thought of as consisting of three layers: the *content* layer, the *message* layer, and the *communication* layer. The content layer bears the actual content of message, in the program's own representation language. The communication layer encodes a set of message features that describes the lower level communication parameters, such as the identity of the sender and recipient, and a unique identifier associated with the communication. It is the message layer that is used to encode a message that one application would like to transmit to another. The message layer forms the core of the KQML language and determines the kinds of interactions one can have with a KQML-speaking agent. Each KQML message has a *performative* and a number of *parameters*. Here is an example of a KQML message:

```
(ask-one
   :content (PRICE IBM ?price)
   :receiver stock-server
   :language LPROLOG
   :ontology NYSE-TICKS
)
```

In 1995, the Foundation for Intelligent Physical Agent (FIPA) began its work on developing standards of agent systems. The centerpiece of this initiative was the development of an ACL. This ACL incorporates many aspects of KQML. It defines 20 performatives for defining the intended interpretation of messages, and it does not mandate any specific language for message content. The concrete syntax for FIPA ACL messages closely resembles that of KQML. Here is an example of a FIPA ACL message:

```
(inform
   :sender agent1
   :receiver agent2
   :content (price good2 150)
   :language si
   :ontology hpl-auction
)
```

If two agents are to communicate about certain domain, then it is necessary for them to agree on the *terminology* that they use to describe this domain. For example, in an agent system for financial investment planning, one agent advertises its capability to a middle agent as "pattern watcher in the *stock* market," whereas another agent requests an agent that is a "pattern watcher in the *share* market." In such a situation, problems arise when the middle agent tries to match them. How could the middle agent know the "*stock* market" and the "*share* market" are the same thing? The agents thus need to be able to agree on what terms like *share* or *stock* mean. Thus a specification of a set of terms, an *ontology*, is required. An ontology is a formal definition of a body of knowledge. The most typical type of ontology used in building agents involves a structural component. Essentially a taxonomy of class and subclass relations coupled with definitions of the relationships between these things.

METHODOLOGIES AND DEVELOPMENT TOOLS FOR INTELLIGENT AGENTS

Existing software development techniques (for example, object-oriented analysis and design) are inadequate for analyzing and designing agent systems. There is a fundamental mismatch between the concepts used by other mainstream software engineering paradigms and the agent-oriented perspective. In particular, extant approaches fail to adequately capture an agent's flexible, autonomous problem-solving behavior, the richness of an agent's interactions, and the complexity of an agent system's organizational structure. This section provides an overview of the state of the art in agent-oriented software engineering and a summary of currently available agent development tools.

A *methodology* is a codified set of procedures for some phases of software engineering, such as analysis and design. A system engineering methodology groups the methods and principles used in a particular discipline. A *method* is a systematic way of doing something or, alternatively, the techniques or arrangements of work for a particular subject.

Software Engineering with Agents, Agent-Based Software Engineering, Multiagent Systems Engineering (MaSE), and *Agent-Oriented Software Engineering* (AOSE) are semantically equivalent terms, but *MaSE* refers to a particular methodology and *AOSE* seems to be the most widely used term. In AOSE, there are some preliminary *methodologies* for engineering multiagent systems—these methodologies provide structured but nonmathematical approaches to the analysis and design of agent systems. These can be broadly divided into two groups:

Those that take their inspiration from object-oriented development, and either extending object-oriented methodologies or adapt object-oriented methodologies to the purposes of AOSE. Representatives of this category include the AAII methodology (17) and the Gaia methodology (18,19).

Those that adapt knowledge engineering or other techniques. One representative in this category is the use of *Z* for specifying agent systems (20).

The AAII methodology draws primarily on object-oriented methodologies and enhances them with some agent-based concepts. The methodology is aimed at the construction of a set of models that, when fully elaborated, define an agent system specification. The AAII methodology provides both *internal* and *external* models. The external model presents a system-level view: The main components visible in this model are agents themselves. The external model is thus primarily concerned with agents and the relationships between them. In contrast, the internal model is entirely concerned with the internals of agents: their beliefs, desires, and intentions.

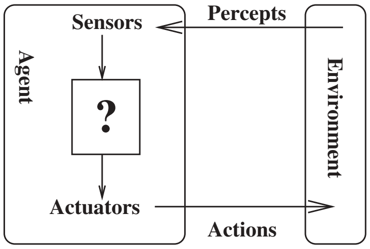


Figure 2. Agents and environments.

The Gaia methodology is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly. In applying Gaia, the analyst moves from abstract to increasingly concrete concepts. Figure 2 shows that agents interact with environments through sensors and actuators, which is adopted from Ref. (21) (page 33). Gaia methodology is well matched with this view of agents. In Gaia, a set of new organizational abstractions that are necessary for designing and constructing systems in complex and open environments are employed. These organizational abstractions, which can be used in the analysis and design phases, include the *environment* in which a multiagent system is situated; the *roles* that have to be played in the agent organization and of their interactions, and the organizational rules and organizational structures. Based on Gaia, the emphasis of the analysis and design is to identify the *environment* that the agent system is situated, the key *roles* in the system and document the various *agent types* that will be used in the system.

In agents in *Z*, a four-tiered hierarchy of the entities that can exist in an agent-based systems is defined. It starts with *entities*, and then *objects* to be entities that have capabilities are defined. *Agents* are then defined to be objects that have goals and are thus in some sense active. Finally, *autonomous agents* are defined to be agents with motivations. The formal definitions of agents and autonomous agents rely on inheriting the properties of lower level components. In the *Z* notation, this is achieved through schema inclusion.

In addition to these representatives, there are also some methodologies for modeling agent systems based on UML notations as well as some formal methods for engineering multiagent systems.

There are dozens of agent construction tools. The tools are categorized as either commercially available products or academic and research projects. The representatives are listed in Table 1.

When building agent systems, certain techniques are required to convert legacy programs into agents. Generally, there are three principal approaches to be taken: implementing a *transducer*, implementing a *wrapper*, and *rewriting* the original programs (22).

Table 1. Agent Construction Tools

Product	Company/ Research Organization	Commercial/ Academic	Language	Description
AgentBuilder	Reticular Systems, Inc.	Commercial	Java	Integrated Agent and Agency Development Environment
Aglets	IBM Japan	Commercial	Java	Mobile Agents
JACK Intelligent Agents	Agent Oriented Software P/Ltd	Commercial	JACK Agent Language	Agent Development Environment
Agent Tcl	Dartmouth University	Academic	Tcl	Mobile Agents
FIPA-OS		Academic	Java	Component-based Toolkit
JADE	TILAB	Academic	Java	Multiagent Framework
JATLite	Stanford University	Academic	Java	Java Packages for Multiagents
Java Agent Framework (JAF)	University of Massachusetts	Academic	Java	Agent Framework
Multi-Agent Modeling Language (MAML)	Central European University	Academic	MAML	Programming Language
Multiagent Systems Tool (MAST)	Technical University of Madrid	Academic	C++	Multiple Heterogeneous Agents
Open Agent Architecture (OAA)	SRI International	Academic	C, Prolog C++, Perl Lisp, Java	Agent Framework
RETSINA	Carnegie-Mellon University	Academic		Communicating Agents
Zeus	British Telecommunications Labs	Academic	Java	Agent Building Environment

TYPICAL APPLICATIONS OF INTELLIGENT AGENTS

Agent technology is rapidly breaking out of universities and research laboratories, and is used to solve real-world problems in a range of industrial and commercial applications. Some of the key systems are outlined below (7).

- YAMS system. YAMS (Yet Another Manufacturing System) applies the well-known Contract Net Protocol to manufacturing control. YAMS adopts a multiagent approach, where each factory and factory component is represented as an agent. Each agent has a collection of plans, representing its capabilities. The contract net protocol allows tasks to be delegated to individual factories, and from individual factories down to flexible manufacturing systems, and then to individual work cells.
- OASIS Air Traffic Control System. OASIS is a sophisticated agent-realized air traffic control system, which is undergoing field trials at Sydney airport in Australia. In this system, agents are used to represent both aircraft and the various air-traffic control systems in operation. The agent metaphor thus provides a useful and natural way of modeling real-world autonomous components. As an aircraft enters Sydney airspace, an agent is allocated for it, and the agent is instantiated with the information and goals corresponding to the real-world aircraft. OASIS is implemented using the typical BDI architecture.
- Maxims. Maxims is an electronic mail filtering agent that "learns to prioritize, delete, forward, sort, and archive mail messages on behalf of a user." It works by "looking over the shoulder" of a user as he or she works with their e-mail reading program and uses every action the user performs as a lesson. Maxims constantly makes internal predictions about what a user will do with a message. If these predictions turn out to be inaccurate, then Maxims keeps them to itself. But when it finds it is having a useful degree of success in its predictions, it starts to make suggestions to the user about what to do.
- The WARREN financial portfolio management system. WARREN is a multiagent system that integrates information finding and filtering in the context of supporting users to manage their financial portfolios. The system consists of agents that coop eratively self-organize to monitor and track stock quotes, financial news, financial analysts reports, and company earnings reports in order to appraise the portfolio owner of the evolving financial picture. The agents not only answer relevant queries but also continuously monitor available information resources for the occurrence of

interesting events and alert the portfolio manager agent or the user.

To date, the main areas in which agent-based applications have been reported include manufacturing, process control, telecommunication systems, air traffic control, traffic and transportation management, information filtering and gathering, electronic commerce, business process management, entertainment, and medical care. In addition to these existing areas, there are a number of emerging application domains for agent technologies and multiagent systems (4,5). Several of these domains are presented here to demonstrate their wide range and diversity. They indicate the potential impact of agent-related technologies on human life and society. More details can be found in Refs. (4) and (5).

- *Ambient Intelligence* (23). Ambient intelligence represents a vision of the future where we shall be surrounded by electronic environments that are sensitive and responsive to people. Ambient intelligence technologies are expected to combine concepts of ubiquitous computing and intelligent systems putting humans in the centre of techno logical developments. Ambient Intelligence emphasizes greater user-friendliness, more efficient services support, user-empowerment, and support for human interac tions. It builds on three recent key technologies: ubiquitous computing, ubiquitous communication, and intelligent user interfaces; yet it offers perhaps the strongest motivation for, and justification of, agent technologies. The consensus is that auton omy, distribution, adaptation, responsiveness, and so on, are the key characterizing features of ambient intelligent artfacts, and in this sense, they share the same char acteristics as agent.
- *Bioinformatics and Computational Biology* (24). One application of multiagent systems in the biological sciences is for simulation modeling of biological systems, in a manner similar to their use for the simulation of socioeconomic and public policy domains. Another area of application in biology is in bioinformatics. The genomic revolution that has spawned microarrays and high throughout technologies has produced vast amounts of complex biological data that require in tegration and multidimensional analysis. Information-gathering agents can help human researchers in finding appropriate research literature or in conducting auto mated or semiautomated testing of data. Data mining agents can be used to do the integration and multidimensional analysis. A potential longer term application of multiagent systems technologies is the use of agents engaged in a reasoned argument to achieve resolution about ambiguous or conflicting experimental evidence, in a manner similar to the way in which human scientists do currently.
- *Grid Computing* (25). Managing access to computing and data resources is a com plex and time-consuming task. As grid and cluster computing matures, deciding which systems to use, where the data reside for a particular application domain, how to migrate the data to the point of computation (or vice versa), and data rates required to maintain a particular application "behavior" become significant. To support these systems it is important to develop brokering approaches based on intelligent techniques—to support service discovery, performance management, and data selection. Intelligent agents provide a useful means to achieve these objectives. An important and emerging area within grid computing is the role of service ontologies—especially domain-specific ontologies, which may be used to capture particular application needs. Using these ontologies, scientists may be able to share and disseminate their data and software more effectively. This has been recognized as being important, and current efforts toward establishing "semantic grids" is a useful first step in this direction. The agent community on the other hand can find grid environments useful testbeds to deploy agents on a large scale. Often, within the multiagent community, agents are restricted to a few 10 s of agents, and often agents undertake identical tasks. To support grid computing, agents can offer different roles, be organized into regional or national dynamic "groups," and be able to migrate between groups to support load balancing. Therefore agents could play an important role in grid computing, and grid computing can offer useful testbeds for investigating agent services. The grid is not only a low-level infrastructure for supporting computation, but it can also facilitate and enable information and knowledge sharing at the higher semantic levels, to support knowledge integration and dissemination.
- *Electronic Business* (20). The continuing growth of electronic business puts high demands on the underlying technology and infrastructure. Decentralization and flexibility of the information and communication systems are of concern. Agent technology—especially the mobility, autonomy, and intelligence of agents—offers a promising approach in these directions. Even though the notion of "intelligent agent" has been stressed over the last years and many questions like security have remain unanswered, electronic commerce poses new challenges and opportunities for agents. To date agents have been used in the first stages of ecommerce, product and merchant discovery, and brokering. The next step will involve moving into real trading—negotiating deals and making purchases. This stage will involve considerable research and development, including generating new products and services such as market-specific agent shells, payment and contracting methods, risk assessment and coverage, quality and performance certification, security, trust, and individual-ization.

In addition to these domains, agent-oriented perspectives are well suited for constructing hybrid intelligent systems (27). Solving complex problems in real-world contexts, such as financial investment planning or mining large data collections, involves many different subtasks, each of which requires different techniques. To deal with such problems, a great diversity of intelligent techniques are available, including traditional techniques like expert systems approaches and soft computing techniques like fuzzy logic, neural networks, or genetic algorithms. These techniques are complementary approaches to intelligent information processing rather than competing ones, and thus, better results in problem solving are achieved when these techniques are combined in hybrid intelligent systems. Multiagent systems are ideally suited to model the manifold interactions among the many different components of hybrid intelligent systems.

TECHNOLOGICAL CHALLENGES

There are a number of broad technological challenges for research and development over the next decade. These are summarized as follows based on the descriptions in Refs. 4 and 5.

- **Increase quality of agent software to industrial standard**. One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems. Clearly, the basic principles of software and knowledge engineering need to be applied to the development and deployment of multiagent systems, but they also need to be augmented to suit the differing demands of this new paradigm. Technology examples include agent- oriented design methodologies, tools and development environments, and seamless integration with current technologies.
- **Provide effective agreed standards to allow open systems development**. In addition to standard languages and interaction protocols, open agent societies will require the ability to collectively evolve languages and protocols specific to the application domain and to the agents involved. Research in this area will draw on linguistics, social anthropology, biology, the philosophy of language, and information theory. Technology examples contain agent communication language, interaction protocols, and multiagent architectures.
- **Provide semantic infrastructure for open agent communities**. To make information agents widely available in real-world applications, a greater understand ing of how agents, databases, and information systems interact is required. This also demands new Web standards that enable structural and semantic description of information. The creation of common ontologies, thesaurus, or knowledge bases play a central role here.
- **Develop reasoning capabilities for agents in open environments**. The next challenge for agent-based computing is to develop appropriate representations of analogous computational concepts to the norms, legislation, authorities, enforcement, and so on, which can underpin the development and deployment of dynamic electronic institutions. The automation of coalition formation can save time and labor and is more effective at finding better coalitions than humans in complex settings. Related issues include negotiation and argumentation and domain-specific models of reasoning.
- **Develop agent ability to understand user requirements**. At the architecture level, future avenues for learning research include developing distributed models of profile management, as well as more general distributed agent learning techniques rather than just single agent learning in multiagent domains. Developing approaches to personalization that can operate in a standards-based, pervasive computing en vironment presents many interesting research challenges, including how to integrate machine learning techniques (for profile adaptation) with structured XML-based profile representations. Another area deserving of greater activity is that of dis tributed profile management—a task for which the agent-based paradigm should be well suited. The impact of the emerging semantic Web on approaches for wrapper induction and text-mining also requires careful study.
- **Develop agent ability to adapt to changes in environment**. Learning tech nology is crucial for open and scalable multiagent systems, but it is still in early development. Many agent research areas have been looking mainly at nonadaptive technology. However, with increasing maturity of these areas, learning techniques will increasingly move toward the center stage in these areas. Examples of areas where learning will receive more attention in the future are communication, negoti ation, planning and coordination, and information and knowledge management.
- **Trust and reputation management**. Collaboration of any kind, especially in situations in which computers act on behalf of users or organizations, will only succeed if there is trust. To ensure this trust requires a variety of factors to be in place. First, a user must have confidence that an agent or a group of agents that represents them within an open system will act effectively on their behalf. Second, agents must be secure and tamperproof and must not reveal information inappropriately. Finally, if users are to trust the outcome of an open agent system, they must have confidence that agents representing other parties or organizations will behave within certain constraints. Mechanisms to do this include reputation mechanisms; the use of norms (social rules) by all members of an open system; and self- enforcing protocols, which ensure that it is not in the interests of any party to break them; and electronic contracts.
- **Virtual organization formation and management**. Virtual organizations have been identified as one of the key contributions of grid computing, but the conditions under which a new virtual organization should be formed, and the procedures for its formation, operation, and dissolution are still not well defined. In current grid applications, virtual organizations are statically defined by the users of the workflows, which mean that they are incapable of handling dynamic situations and reconfiguring themselves in

an automated manner. This automated formation and ongoing management of virtual organizations in open environments thus constitutes a major research challenge, a key objective of which is to ensure that they are both agile (can adapt to changing circumstances) and resilient (can achieve their aims in a dynamic and uncertain environment).

CONCLUSIONS

This article is mainly focused on the technology view of intelligent agents. Actually, intelligent agents are becoming a computing paradigm. In Ref. 21, Russell and Norvig define artificial intelligence as the study of agents that perceive the environments and take actions. Systems consisting of interacting intelligent agents is evolving a main stream software engineering approach for developing applications in complex domains (19). Intelligent agents and multiagent systems as a computing paradigm are well suited to modeling systems with very high complexity. On the other hand, there are also some challenging issues like complex emergent behavior, self-organized criticality, and phase transition, which are related to multiagent systems. In this respect, autonomy oriented computing (AOC) (28) provides a means of modeling and characterizing complex emergent behaviors in multiagent systems.

As Jennings et al. stated in Ref. 7, the field of intelligent agents is a vibrant and rapidly expanding area of research and development. It represents a melting pot of ideas originating from such areas as distributed computing, object-oriented systems, software engineering, artificial intelligence, economics, sociology, and organizational science. The basic conceptual framework of intelligent agents has become common currency in a range of closely related disciplines and offers a natural and powerful means of analyzing, designing, and implementing a diverse range of software solutions.

Agent-based approaches have been a source of technologies to several research areas, both theoretical and practical. These areas include distributed planning and decision making, automated auction mechanisms, communication languages, coordination mechanisms, matchmaking architectures and algorithms, ontologies and information agents, negotiation, and learning mechanisms. Moreover, agent technologies have drawn from, and contributed to, a diverse range of academic disciplines, in the humanities, the natural sciences, and the social sciences. Agents offer a new and often more appropriate route to the development of complex systems, especially in open and dynamic environments.

Wooldridge (6) is a good introductory text for agent and multiagent systems. For a more comprehensive discussion on these topics, refer to Ref. 8. Ferber (29) is an undergraduate textbook, which focused on multiagent aspects rather than on the theory and practice of individual agents.

For a road map of agent and multiagent system research, refer to Refs. 4, 5, and 7. More resources on intelligent agents can be found in the *Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems, Autonomous Agents and Multi-Agent Systems* (journal), and *AgentLink, the European Network of Excellence for Agent Based Computing* (www.agentlink.org).

In a broader sense, objects and components in today's distributed and concurrent systems are starting to approach the view of agents that was focused here. Active objects or actors in object-oriented community are also becoming closer to the view of agents described in this article (30). In the computer network context, agents are also used to refer to a piece of software such as mail user agents, mail transfer agents, and mail delivery agents, even though they are not as smart as what was described here.

BIBLIOGRAPHY

1. A. Bond and L. Gasser (eds.), *Readings in Distributed Artificial Intelligence*, San Mateo, CA: Morgan Kaufmann, 1988.
2. M. Wooldridge and N. R. Jennings, Intelligent agents: Theory and practice, *Knowledge Eng. Rev.*, **10**(2): 115–152, 1995.
3. J. M. Bradshaw (ed.), *Software Agents*, Menlo Park, CA: AAAI Press, 1997, Chapter 1.
4. M. Luck, P. McBurney, and C. Preist, *Agent Technology: Enabling Next Generation Computing—A Roadmap for Agent Based Computing*, Southampton, UK: AgentLink, 2003.
5. M. Luck, P. McBurney, O. Shehory, and S. Willmott, *Agent Technology Roadmap: A Roadmap for Agent Based Computing*, Southampton, UK: AgentLink III, 2005.
6. M. Wooldridge, *An Introduction to Multiagent Systems*, Chichester: John Wiley & Sons, 2002.
7. N. R. Jennings, K. Sycara, and M. Wooldridge, A roadmap of agent research and development, *J. Auton. Agents Multi-Agent Syst.*, **1**: 7–38, 1998.
8. G. Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Cambridge, MA: MIT Press, 1999.
9. M. Georgeff and A. Lansky, Reactive reasoning and planning, *Proc. of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle, WA, 1987, pp. 677–682.
10. Y. Shoham, Agent-oriented programming, *Artif. Intell.*, **60**(1): 51–92, 1993.
11. R. A. Brooks, Intelligence without Representation, *Artif. Intell.*, **47**: 139–159, 1991.
12. J. P. Müller, *The Design of Intelligent Agents: A Layered Approach*, LNCS 1177, Berlin: Springer, 1996.
13. A. Cheyer and D. Martin, The Open Agent Architecture, *J. Auton. Agents Multi-Agent Syst.*, **4**(1,2): 143–148, 2001; M. N. Huhns and M. P. Singh (eds.), *Readings in Agents*, San Francisco, CA: Morgan Kaufmann, 1998.
14. W. Swartout and A. Tate, Ontologies, *IEEE Intel. Syst. Their Applicat.*, **14**(1): 18–19, 1999.
15. T. Finin, Y. Labrou, and J. Mayfield, KQML as an agent communication language, in J. M. Bradshaw (ed.), *Software Agents*, Menlo Park, CA: AAAI Press/ The MIT Press, 1997, pp. 291–316.
16. FIPA, Agent Communication Language. Available: http://www.fipa.org/spec/f8a22.zip.
17. D. Kinny, M. Georgeff, and A. Rao, A Methodology and modeling technique for systems of BDI agents, *Workshop on Modeling Autonomous Agents in a Multi-Agent World*, LNAI 1038, New York: Springer, 1996, pp. 56–71.

18. M. Wooldridge, N. Jennings, and D. Kinny, The Gaia Methodology for agent-oriented analysis and design, *J. Auton. Agents Multi-Agent Syst.*, **3**(3): 285–312, 2000.

19. F. Zambonelli, N. Jennings, and M. Wooldridge, Developing multiagent systems: the gaia methodology, *ACM Trans. Softw. Eng. Methodol.*, **12**(3): 317–370, 2003.

20. M. d'Inverno and M. Luck, *Understanding Agent Systems*, Berlin: Springer, 2001.

21. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach,* 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2003.

22. M. R. Genesereth and S. P. Ketchpel, Software Agents, *Commun. ACM*, **37**(7): 48–53, 1998.

23. T. Basten, M. Geilen, and H. de Groot (eds.), *Ambient Intelligence: Impact on Embedded System Design*, Boston, MA: Kluwer Academic Publishers, 2003.

24. S. Krawetz and D. Womble (eds.), *Introduction to Bioinformatics: A Theoretical and Practical Approach*, Totowa, NJ: Humana Press, 2003.

25. "Agent Based Cluster and Grid Computing" Session, *Proce. of the 3rd International Symposium on Cluster Computing and the Grid*, Tokyo, Japan, IEEE Computer Society Press, 2003.

26. R. Guttman, A. Moukas, and P. Maes, Agents as mediators in electronic commerce, in M. Klusch (ed.), *Intelligent Information Agents*, Berlin: Springer, 1999.

27. Z. Zhang and C. Zhang, *Agent-Based Hybrid Intelligent Systems: An Agent-Based Framework for Complex Problem Solving*, LNAI 2938, Berlin: Springer, 2004.

28. J. Liu, X. Jin, and K. Tsui, *Autonomy Oriented Computing: From Problem Solving to Complex Systems Modeling*, New York: Kluwer Academic Publishers, 2005.

29. J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Harlow, UK: Addison-Wesley, 1999.

30. Z. Guessoum and J.-P. Briot, From active objects to autonomous agents, *IEEE Concurrency*, **7**(3): 68–76, 1999.

FURTHER READING

M. Wooldridge, Agent-based software engineering, *IEE Proc. Softw. Eng.*, **144**(1): 26–37, 1997.

N. R. Jennings, On agent-based software engineering, *Artif. Intell.*, **117**: 277–296, 2000.

S. Russell and P. Norvig, *A Modern Approach to Artificial Intelligence*, 2nd ed., Upper Saddle River, NJ: Prentice-Hall, 2003.

N. R. Jennings and M. J. Wooldridge (eds.), *Agent Technology: Foundations, Applications, and Markets*, Berlin: Springer, 1998.

V. Subrahmanian, P. Bonatti, J. Dix, et al., *Heterogeneous Agent Systems*, Cambridge, MA: MIT Press, 2000.

M. Wooldridge and P. Ciancarini, Agent-oriented software engineering: The state of the art, in P. Ciancarini and M. Wooldridge (eds.), *Agent-Oriented Software Engineering*, LNAI 1957, New York: Springer, 2001. Available online: http://www.csc.liv.ac.uk/~mjw/pubs/.

Chengqi Zhang
University of Technology
Sydney, Australia
Zili Zhang
Deakin University
Geelong, Australia

K

KNOWLEDGE ACQUISITION

Knowledge acquisition is the process by which problem-solving expertise is obtained from some knowledge source, usually a domain expert. This knowledge is then implemented into an expert system program that can provide expert assistance to nonexperts when and where a human expert is not available.

Traditionally knowledge acquisition is accomplished through a series of long and intensive interviews between a knowledge engineer, who is a computer specialist, and a domain expert, who has superior knowledge in the domain of interest. This process is usually referred to as *knowledge elicitation* to distinguish it from the more general knowledge acquisition term.

Experience has shown that knowledge acquisition from experts is the most difficult, time-consuming, and costly part of developing an expert system (1). The difficulty of knowledge acquisition has stimulated research in developing machines that autonomously acquire knowledge without the assistance of humans. Although progress has been made in the area of automated knowledge acquisition, in the foreseeable future, most of the knowledge for practical expert systems will be obtained through the interaction of domain experts and knowledge engineers.

THE KNOWLEDGE ENGINEERING PROCESS

Knowledge acquisition is an activity of a larger process used to develop expert systems, called *knowledge engineering*. The knowledge engineering process consists of several phases, each consisting of several tasks. Although knowledge engineering phases and tasks are usually shown in sequence, in practice they are conducted iteratively. Figure 1 depicts the phases of the knowledge engineering process. The following is a summary of the activities conducted in each phase:

Phase I: Problem assessment. This phase assesses the applicability and feasibility of an expert system solution to a particular problem.

Phase II: Knowledge acquisition. This phase involves the acquisition of knowledge from a domain expert and/or other sources of knowledge. It also involves interpreting, analyzing, and documenting the acquired knowledge.

Phase III: Knowledge representation. This phase involves the selection of a knowledge representation scheme and control strategy. Acquired knowledge is represented using the selected representation.

Phase IV: Knowledge coding. This phase involves coding the knowledge using appropriate expert system development software.

Phase V: Knowledge validation and verification. This phase ensures that the developed system performs at an acceptable level of expertise and that it correctly implements its initial specification.

Phase VI: Maintenance. This is an ongoing phase that corrects system errors and deficiencies. It also updates the system knowledge as the requirements evolve.

An interesting aspect of the iterative nature of the knowledge engineering process is its synergistic effect. Both the system and the development team improve their knowledge about the problem and how best to solve it as the development progresses.

DIFFICULTIES IN KNOWLEDGE ACQUISITION

Experience has shown that knowledge acquisition is a difficult, expensive, and time-consuming process. The major source of difficulty stems from a well-recognized fact in the field of cognitive psychology that eliciting knowledge from humans is an inherently difficult task (2). Humans are usually unaware of their mental processes when solving a problem (3). They may not be able to communicate their knowledge, not because they cannot express it, but because they are unaware of what knowledge they are using in their problem-solving activities (4). Furthermore, humans will provide an explanation of their performance that is different from the way they actually perform their tasks (5).

As most expert system projects rely on elicitation of knowledge between an expert and a knowledge engineer, many of the problems identified by cognitive psychologists are manifested. These problems are as follows:

- Experts may be unaware of knowledge used.
- Experts may be unable to articulate their knowledge.
- Experts may provide irrelevant, incomplete, incorrect, or inconsistent knowledge.

Additional problems that add to the complexity of acquiring knowledge include the following:

- Experts may not be available or may be unwilling to cooperate.
- Lack of well-defined knowledge acquisition methods.
- The complexities of dealing with a large number of participants with different backgrounds, different skills and knowledge sets, and using different terminology.
- The multiplicity of the sources of knowledge required for the system.
- The exponential growth in the complexity and interdependencies of knowledge with the size of the domain.
- The mismatch of the level of abstraction of knowledge between experts and computers.
- Potential interpersonal communication problems between the knowledge engineer and the expert.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

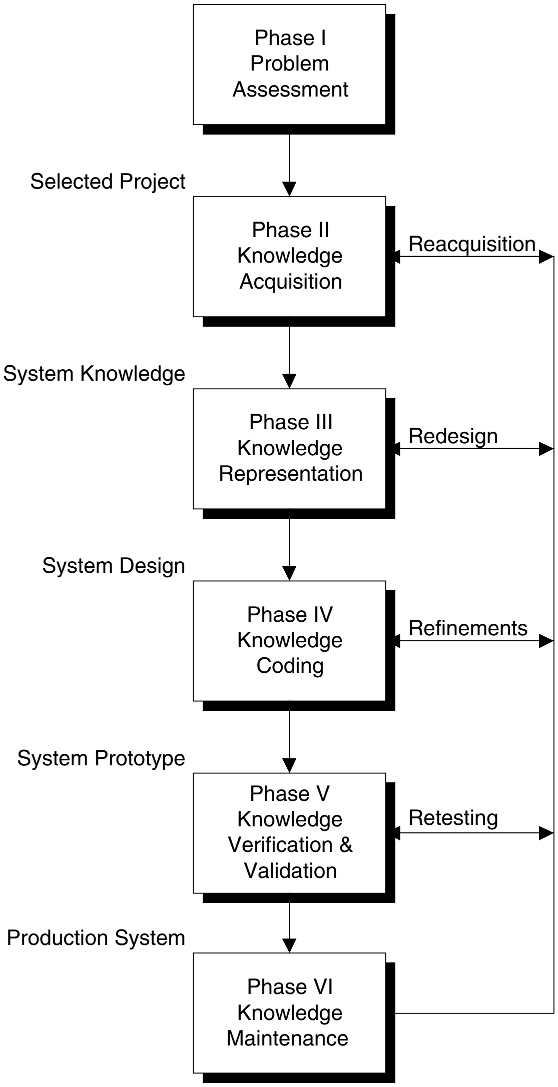


Figure 1. Phases of the knowledge engineering process. Although the phases appear sequential, there is considerable overlap and iteration in their execution.

FUNDAMENTAL CONCEPTS OF KNOWLEDGE

Levels of Knowledge

Knowledge can be broadly classified into two levels: shallow knowledge and deep knowledge.

Shallow knowledge refers to surface-level information that can be used to solve problems in very specific domains. Shallow knowledge is usually empirical and represents knowledge accumulated through the experience of solving past problems. Although shallow knowledge can be easily represented by computers, it is limited in representing and solving problems of a knowledge domain; thus, it is usually insufficient in describing complex situations.

Deep knowledge refers to the fundamental knowledge about a problem represented by its internal structure, fundamental laws, functional relationships, and so on. Deep knowledge can be applied to different tasks and under different situations. Deep knowledge is difficult to represent using computers, as it requires a complete and thorough understanding of the basic elements of knowledge and their complex interactions.

Types of Knowledge

In addition to the above two categories, knowledge can be classified by various types as follows:

Declarative knowledge describes *what* is known about a problem. It is a descriptive representation of knowledge that includes simple statements that are either true or false. The factual statement "The sky is blue" is an example of declarative knowledge. Facts, concepts, and relations are typical examples of declarative knowledge.

Procedural knowledge describes *how* a problem is solved. It provides a step-by-step sequence of instructions on how to solve the problem. For example, "If the temperature falls below 50, turn on the heater." Rules, strategies, and procedures are examples of procedural knowledge.

Heuristic knowledge is a special type of knowledge that describes *rules-of-thumb* used to guide the reasoning process to solve a problem. Heuristic knowledge is acquired through extensive experience. Experts usually compile deep knowledge into simple heuristics to aid in problem solving.

Episodic knowledge is time-stamped knowledge organized as a case or episode. This knowledge can confer the capability to perform protracted tasks or to answer queries about temporal relationships and to use temporal relationships.

Meta-knowledge describes knowledge about knowledge. It is used to select other knowledge and to direct the reasoning on how to best solve a problem.

It is important to identify the type of domain knowledge to be acquired as different types of knowledge are best elicited by different techniques. In many situations, the domain knowledge consists of several types. In these situations, it is usually preferred to employ more than one technique to acquire the knowledge.

Sources of Knowledge

Knowledge may be obtained from a variety of sources. These sources can be divided into two main types: *documented* and *undocumented*. Documented sources include manuals, books, articles, reports, standard procedures, regulations, guidelines, pictures, maps, video, films, and computer databases. Undocumented knowledge largely exists in human minds. Sources of undocumented knowledge include experts, end users, and observed behavior.

PROCESS OF KNOWLEDGE ACQUISITION

The process of knowledge acquisition is a cyclical one. It begins with the collection and recording of knowledge, followed by its interpretation, analysis, and organization.

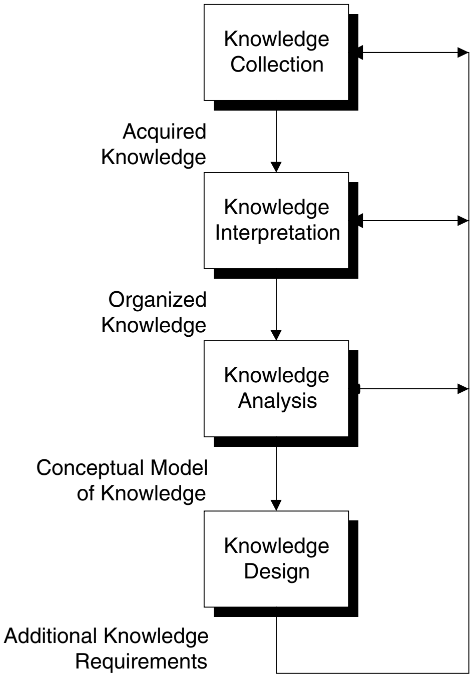


Figure 2. The knowledge acquisition process. The process is cyclic; information obtained from each cycle is used to design new ways to acquire knowledge.

Finally methods are designed for clarifying and collecting additional knowledge based on acquired knowledge. Figure 2 illustrates the knowledge acquisition process.

Knowledge Collection. Knowledge collection is the task of acquiring knowledge from a knowledge source. Usually, this step requires significant interaction between an expert and a knowledge engineer. At the initial stages of knowledge collection, information obtained from the expert represent a broad overview of the domain and the general requirements of the expert system. Later stages of knowledge collection are characterized by their narrow focus, with emphasis on the details of how the expert performs the various tasks. Knowledge acquisition sessions are recorded and transcribed in preparation for interpretation and analysis.

Knowledge Interpretation. This task involves reviewing the collected information and the identification and classification of key pieces of knowledge, such as facts, concepts, objects, rules, problem-solving strategies, and heuristics. In early iterations of the cycle, the knowledge collected will be of a general nature. During later stages, different and deeper problem-solving knowledge will be uncovered.

Knowledge Analysis. This task takes the key pieces of knowledge uncovered during the knowledge interpretation phase and forms theory on the representation of knowledge and problem-solving strategies used. It requires assembling the acquired knowledge into related groups and storing them in the knowledge dictionary. The output of this task is a conceptual model of the domain knowledge that shows the information an expert system will require, the reasoning it will perform, and the sequence of steps it will take in order to accomplish its task. A variety of graphical techniques are typically used to develop the conceptual model. These techniques include flowcharts, cognitive maps, inference networks, decision tables, and decision trees.

Knowledge Design. After the completion of the collection, interpretation, and analysis tasks, some concepts and problem-solving strategies emerge as requiring further investigation and clarification. This task identifies this information and designs an agenda that includes clarifying old issues and discussing new ones with the expert during the following iteration of the acquisition cycle.

Although theoretically the cycle could continue indefinitely, in practice, the process is repeated until the resulting system meets some acceptable performance measures.

PARTICIPANTS IN KNOWLEDGE ACQUISITION

The main participants in knowledge acquisition are the domain expert, the knowledge engineer, and the end user. Each participant plays an important role in knowledge acquisition and must possess certain qualifications to contribute effectively to the knowledge acquisition process.

The Expert

The expert is usually the primary source of knowledge for most expert system projects. The expert's main task is to communicate his/her domain expertise to the knowledge engineer for encoding into an expert system. In addition to possessing extensive knowledge and problem-solving skills in a given domain, an expert should have the following qualifications:

1. Ability to communicate the problem-solving knowledge
2. Willingness and eagerness to participate in the project
3. Ability to work well with others
4. Availability for the duration of the project

The Knowledge Engineer

The main responsibility of a knowledge engineer is to acquire, analyze, interpret, design, and encode the knowledge. Knowledge engineers must have the technical skills for interpreting, analyzing, and coding the collected knowledge. Additionally they should have the following qualifications:

1. Good communications and interpersonal skills
2. Good knowledge elicitation and interviewing skills
3. Good project management skills

The End-user

End users are an important, yet often ignored, additional source of knowledge. They provide a high-level understanding of the problem. They are particularly useful in providing a general perspective and insight early on during the knowledge elicitation process. Some of the qualifications required for end users to support knowledge acquisition include:

1. Availability and willingness to participate in the project
2. Open-minded attitude toward change

METHODS OF KNOWLEDGE ACQUISITION

Knowledge acquisition methods are classified in different ways and appear under different names in different literature. In this article, we follow a classification based on the degree of automation in the acquisition process. The classification divides knowledge acquisition methods into three categories: manual methods, combined manual and automated methods, and automated methods (6). This classification is depicted in Fig. 3.

Manual methods are largely based on some kind of interview between an expert and a knowledge engineer. The knowledge engineer elicits knowledge from the expert during interviewing sessions, refines it with the expert, and then represents it in a knowledge base. The two manual methods commonly used are interviews (structured, unstructured, and questionnaire) and task-based methods (protocol analysis, observation, and case analysis). In some cases, an expert may play the role of a knowledge engineer and self-elicit the knowledge without the help of a knowledge engineer.

Combined manual and automated methods use techniques and tools to support both experts and knowledge engineers in the knowledge acquisition process. Methods intended to support experts provide an environment for constructing the knowledge base with little or no support from a knowledge engineer. Methods intended to support knowledge engineers provide an environment of acquiring and representing knowledge with minimal support from the experts.

Automated methods minimize or even eliminate the roles of both experts and knowledge engineers. They are based on machine learning methods and include learning by induction, neural networks, genetic algorithms, and analogical and case-based reasoning.

It is important to note that the categories of the above classification are not mutually exclusive as some overlap can exist between them.

MANUAL METHODS

Interviews

Interviews are the most common elicitation method used for knowledge acquisition. It involves a two-way dialog between the expert and the knowledge engineer. Information is collected by various means and subsequently transcribed, interpreted, analyzed, and coded. Two types of interviews are used: unstructured and structured. Although many techniques have been proposed for conducting interviews, effective interviewing is still largely an art.

Unstructured Interviews. Unstructured interviews are conducted without prior planning or organization. They

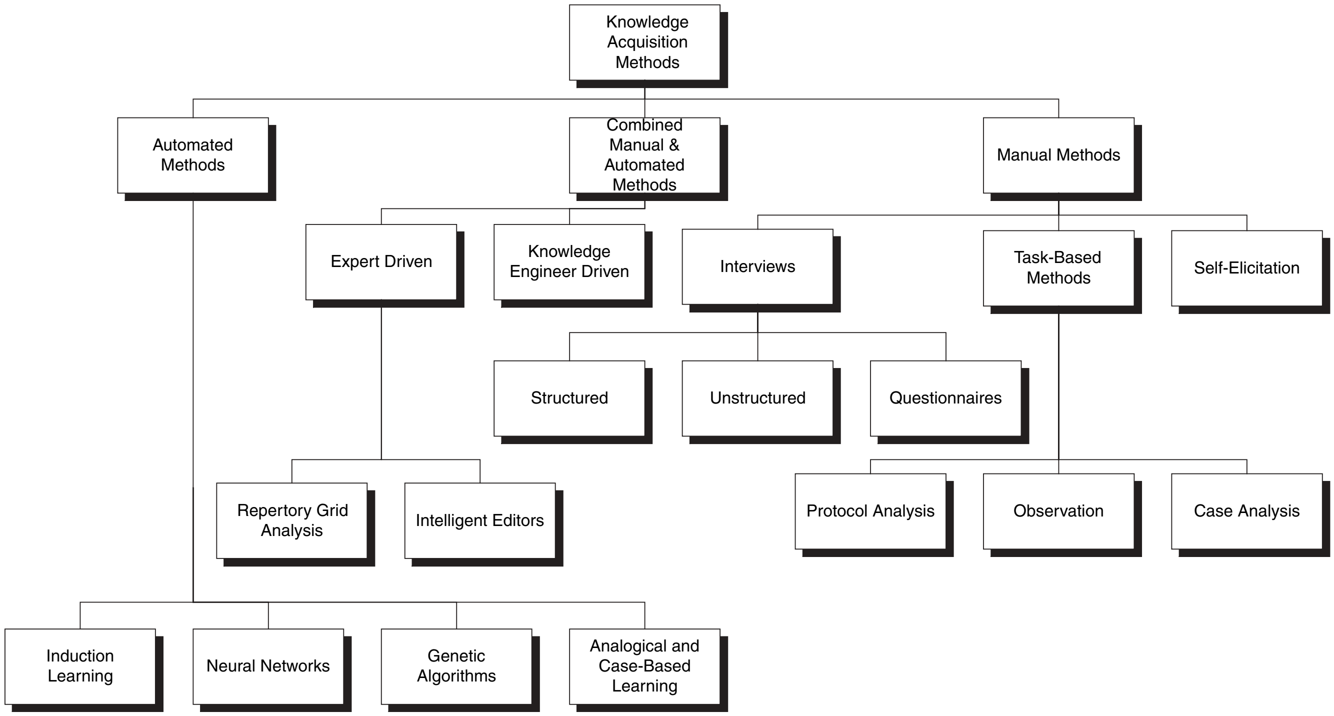


Figure 3. Knowledge acquisition methods. This classification is based on the degree of automation in the acquisition process.

are an informal technique that helps the knowledge engineer gain a general understanding of the problem, its most important attributes, and general problem-solving methods. During unstructured interviews, the knowledge engineer asks some opening questions and lets the expert talk about the problem, its major objects, concepts, and problem-solving strategies. The role of the knowledge engineer is limited to asking clarifying questions or redirecting the interview toward more interesting areas.

Unstructured interviews appear in several variations (6). In the "talkthrough" interview, the expert talks through the steps he follows to solve a specific problem. In the "teachthrough" interview, the expert plays the role of an instructor and explains "what" he does and "why" he does it in order to solve a problem. In the "readthrough" interview, the expert instructs the knowledge engineer on how to read and interpret the documents used for the task.

Unstructured interviews are useful in uncovering the basic structure of the domain, the main attributes of the problem, and the general problem-solving methods used by the expert. They are appropriate during the early stages of knowledge acquisition when the knowledge engineer is exploring the domain.

However, unstructured interviews suffer from several drawbacks (7). First, unstructured interviews lack the organization for the effective transfer of knowledge. Second, due to lack of structure, domain experts find it difficult to express important elements of their knowledge. Third, experts interpret the lack of structure as requiring little or no preparation. Fourth, data collected from an unstructured interview are often unrelated. Fifth, very few knowledge engineers can conduct an effective unstructured interview. Finally, unstructured situations do not facilitate the acquisition of specific information from experts.

Structured Interviews. Structured interviews maintain a focus on one aspect of the problem at a time by eliciting details on that aspect before moving to a different one. This focus is maintained by structuring the interview based on a prior identification of the problem's key issues obtained through earlier unstructured interviews or other sources. The interview structure forces an organized exchange between the expert and the knowledge engineer and reduces the interpretation problems and the distortion caused by the subjectivity of the expert.

Structured interviews require extensive preparation from the part of the knowledge engineer. In addition, conducting and managing the interview properly require attention to several issues. Some of the basic issues relate to items such as setting up the interview, scheduling the session, choosing the interview location, and the conduct of the first interview. Other issues include knowing how to begin and end the interview and how to ask questions in a way that will provide the desired information. Many guidelines exist in the literature on how to conduct effective structured interviews. For example, see the guidelines suggested in McGraw and Harbison-Briggs (7), Prerau (8), and Scott et al. (9).

The main advantage of structured interviews is their focus and the resulting detailed information obtained on a given issue. They are usually easier to manage, and the information collected is easier to analyze and interpret. Structured interviews are particularly useful in identifying the structure of the domain objects and their properties, concept relationships, and general-problem solving strategies.

The main limitation of structured interviews is that concepts unrelated to the interview focus may not be discovered. This limitation will be particularly manifested when the knowledge engineer is not fully aware of the topics' main issues. Additionally, structured interviews provide little insight on procedural knowledge.

Questionnaires. Although questionnaires are not strictly an interviewing method, they are used in knowledge acquisition to complement interviews by asking the expert to clarify already developed topics during advanced stages of knowledge acquisition.

Task-Based Methods

Task-based methods refer to a set of techniques that present the expert with a task and attempt to follow his or her reasoning in solving the problem. Task-based methods can help the knowledge engineer in identifying what information is being used, why it is being used, and how it is being used. The methods that can be grouped under this approach include protocol analysis, observation, and case studies.

Protocol Analysis. In protocol analysis, the expert is asked to perform a real task and to verbalize at the same time his or her thought process while performing the task. Usually a recording is made during this process, using a tape or video recorder, which becomes later a record, or protocol, that traces the behavior of the expert while solving a problem. As with interviews, this recording is transcribed, analyzed, reviewed, and coded by the knowledge engineer.

The main difference between a protocol analysis and an interview is that a protocol analysis is mainly a one-way communication. The knowledge engineer task is limited to selecting a task, preparing the scenario, and presenting it to the expert. During the session, the expert does most of the talking as the knowledge engineer listens and records the process.

The main advantage of protocol analysis is that it provides immediate insight of problem-solving methods, rather than retrospectively after the fact. It is particularly useful for a non-procedural type of knowledge, where the expert applies a great deal of mental and intellectual effort to solve a problem.

However, several cognitive psychologists have argued that asking experts to verbalize their problem-solving knowledge while performing a task creates an unnatural situation that influences task performance (10). In addition, some problems, such as ones that involve perceptual-motor tasks, do not have a natural verbalization. Forcing an expert to "think aloud" in these situations can lead to the collection of misleading and inaccurate information.

Observation. Another useful knowledge acquisition technique is observing the expert in the field while solving a problem. Observation is usually conducted at the place where the expert makes the actual decisions. Experience has shown that the realism of the expert problem-solving approach is greatly influenced by the usual physical environment of the problem.

The main advantage of this approach is that it allows the knowledge engineer to observe the decision making of the expert in a realistic environment. It provides an unbiased and unobtrusive technique for collecting knowledge. It is particularly useful for collecting information on procedural knowledge.

Observations are usually expensive and time consuming. A large amount of information is usually collected from which only a small fraction is useful.

Case Analysis. A case is an actual problem that has been solved together with its solution and the steps taken to solve it. There are two primary ways a case analysis is used for knowledge elicitation: the retrospective and observational case analyses (11). In a *retrospective* case analysis, the expert is asked to review a case and explain in retrospect how it was solved. The expert begins by reviewing the given recommendation and then works backward to identify the problem concepts and knowledge components used to support this recommendation. In an *observational* case analysis, the expert is asked to solve the problem, whereas the knowledge engineer observes the problem-solving approach of the expert.

Several types of cases could be used in conjunction with either the retrospective or observational case analyses. The two common types used by knowledge engineers are the typical case and the unusual case. The *typical* case represents a situation that is well understood and known by the expert. The results of a typical case usually reveal the typical knowledge used by the expert to solve a problem. The *unusual* case represents an unusual or novel situation that requires a deeper level of problem-solving knowledge. Usually typical cases are used initially in the project when a general understanding of the domain and the problem-solving expertise is required. Unusual cases are used later in the project when deeper knowledge is needed to provide greater problem-solving expertise to the system.

A main advantage of the case analysis method is that information is obtained in the context of a realistic situation, thus providing more accurate insight into problem-solving strategies. A case analysis usually reveals more specific problem-solving knowledge than that obtained from interviewing techniques. The retrospective case analysis has the further advantage of not interfering with the problem-solving activity, because retrospection requires the expert to recall from memory the information needed to solve the problem, rather than actually solving the problem.

A major disadvantage of the case analysis method, particularly the retrospective type, is that it may provide incomplete information and few details on the domain under study. Another disadvantage is the expert's bias toward typical situations solved that could produce inconsistent results. Selecting an unusual but solvable case could be challenging and presents yet another difficulty for this approach.

Self-Elicitation. In some cases, the expert may have both the technical interest and the needed training to play the role of a knowledge engineer. In this case the expert may acquire and represent the knowledge directly without the intermediary of a knowledge engineer. This process can be accomplished through self-administered questionnaires or through self-reporting. Self-reporting can take the form of an activity log, knowledge charts, introductory tutorials, or other similar documents that report on the problem-solving activities of the expert.

A main problem with self-elicitation methods is that experts are usually not trained in knowledge engineering methods and techniques. The resulting knowledge tends to have a high degree of bias, ambiguity, new and untested problem-solving strategies, as well as vagueness about the nature of associations among events (11). In addition, experts lose interest rapidly in the process, and consequently, the quality of the acquired knowledge decreases as the reporting progresses. Self-elicitation methods are useful when experts are inaccessible and in the gathering of preliminary knowledge of the domain.

COMBINED MANUAL AND AUTOMATED METHODS

Manual knowledge acquisition methods are usually time consuming, expensive, and even unreliable. Combined manual and automated methods use techniques and tools designed to reduce or eliminate the problems associated with manual methods. They are designed to support both experts and knowledge engineers in the knowledge acquisition process.

Methods to Support the Experts

Repertory Grid Analysis. Repertory grid analysis (RGA) is one of several elicitation techniques that attempt to gain insight into the expert's mental model of the problem domain. It is based on a technique, derived from psychology, called the *classification interview*. When applied to knowledge acquisition, these techniques are usually aided by a computer. RGA is based on Kelly's model of human thinking called Personal Construct Theory (12). According to this theory, people classify and categorize knowledge and perceptions about the world. Based on this classification, they are able to anticipate and act on everyday decisions.

The RGA involves the following steps:

1. Construction of conclusion items. These items are the options that will be recommended by the expert system. For example, an investment portfolio advisor conclusion items might include the following options: 100% investment in savings; a portfolio with 100% stocks (portfolio 1); a portfolio with 60% stocks, 30% bonds, and 10% savings (portfolio 2); and a portfolio with 20% stocks, 40% bonds, and 40% savings (portfolio 3).
2. Construction of traits. These traits are the important attributes that the expert considers in making decisions. For example, in the investment portfolio advisor example, traits might include age, investment

amount, and investment style. Traits are identified by picking three conclusion items and identifying the distinguishing characteristics of each from the two others. Each trait is given values on a bipolar scale (i.e., a pair of opposite values). In the investment portfolio advisor example, the identified traits could have the following values: young/old, small/large, and conservative/aggressive.

3. Rating of conclusion items according to traits. The expert rates each conclusion item on a scale of one to five. Five is given to an item that satisfies the left-hand pole of the trait and one to an item that satisfies the right pole. The answers are recorded in a grid as shown in Table 1.
4. Rule generation. Once the grid is completed, rules are generated that provide decision items given a desired trait importance.

Several knowledge acquisition tools have been developed based on the RGA method. The best known tool of this group is the Expertise Transfer System (ETS) (13). ETS is used to build a knowledge system through several iterative steps: (1) ETS interviews the experts to uncover conclusion items, problem-solving traits, trait structure, trait weights, etc.; (2) information acquired from the expert is built into information bases; (3) information bases are analyzed and built into knowledge bases (rules, frames, or networks); (4) knowledge bases are incrementally refined using test case histories; and (5) knowledge bases are implemented into expert systems. Other representative tools in this category include KRITON (15), and AQUINAS(16).

Intelligent Editors. An intelligent editor allows the domain expert to capture the knowledge directly without the intermediary of a knowledge engineer. The expert conducts a dialog with the editor using a natural language interface that includes a domain-specific vocabulary. Through the intelligent editor, the expert can manipulate the rules of the expert system without knowing the internal structure of these rules.

The editor assists the expert in building, testing, and refining a knowledge base by retrieving rules related to a specific topic and by reviewing and modifying the rules if necessary. The editor also provides an explanation facility. The expert can query the system for conclusions given a set of inputs. If the expert is unhappy with the results, he can have the editor show all the rules used to arrive at that conclusion.

Table 1. A Repertory Grid for an Investment Portfolio Advisor

Attribute	Age	Investment Amount	Investment Style
Trait	Young(5)	Small(1)	Conservative(1)
Opposite	Old(1)	Large(5)	Agressive(5)
Savings	2	1	1
Portfolio 1	4	4	5
Portfolio 2	3	3	3
Portfolio 3	2	2	2

Some editors have the ability to suggest reasonable alternatives and to prompt the expert for clarifications when required. Other editors have the ability to perform syntax and semantic checks on the newly entered knowledge and detect inconsistencies when they occur.

A classic example of intelligent editors is a program called TEIRESIAS that was developed to assist experts in the creation and revision of rules for a specific expert system while working with the EMYCIN shell (1).

Methods to Support the Knowledge Engineer

Several types of tools have been developed to support knowledge acquisition. They include knowledge-base editors, explanation facilities, and semantic checkers.

Knowledge-base editors facilitate the task of capturing the knowledge and entering it into the knowledge base. They provide syntax and semantic checks to minimize errors and ensure validity and consistency. Several types of editors exist. Rule editors simplify the task of defining, modifying, and testing production rules. Graphical editors support the development of structured graphic objects used in developing the knowledge base (17).

Explanation facilities support the knowledge engineer in acquiring and debugging the knowledge base by tracing the steps followed in the reasoning process of the expert to arrive at a conclusion.

Semantic checkers support the construction of, and changes to, knowledge bases. They ensure no errors or inconsistencies exist in the knowledge.

AUTOMATED METHODS

Automated methods refer to the autonomous acquisition of knowledge through the use of *machine learning* approaches. The objective of using machine learning is to reduce the cost and time associated with manual methods, minimize or eliminate the use of experts and knowledge engineers, and improve the quality of acquired knowledge. In this section we will discuss five of these approaches. They include inductive learning, neural networks, genetic algorithms, and case-based reasoning and analogical reasoning.

Inductive Learning

Inductive learning is the process of acquiring generalized knowledge from example cases. This type of learning is accomplished through the process of reasoning from a set of facts to conclude general principles or rules.

Rule induction is a special type of inductive learning in which rules are generated by a computer program from example cases. A rule-induction system is given an example set that contains the problem knowledge together with its outcome. The example set can be obtained from the domain expert or from a database that contains historical records. The rule-induction-system uses an induction algorithm to create rules that match the results given with the example set. The generated rules can then be used to evaluate new cases where the outcome is not known.

Consider the simple example set of Table 2 that is used in approving or disapproving loans for applicants. Applica-

Table 2. Example Dataset from a Loan Application Database Used for Rule Induction

Name	Annual Income	Assets	Age	Loan Decision
Applicant A	High	None	Young	Yes
Applicant B	Medium	Medium	Middle	Yes
Applicant C	Low	High	Young	Yes
Applicant D	Low	None	Young	No

tion for a loan includes information about the applicant's income, assets, and age. These are the decision factors used to approve or disapprove a loan. The data in this table show several example cases, each with its final decision. From this simple example case, a rule-induction system may infer the following rules:

1. If income is high, approve the loan.
2. If income is low and assets are high, approve the loan.
3. If income is medium, assets are medium, and age is middle or higher, approve the loan.

The heart of any induction systems is the induction algorithm, which is used to induce rules from examples. Induction algorithms vary from traditional statistical methods to neural computing models.

A classic and widely used algorithm for inductive learning is ID3 (18). The ID3 algorithm first converts the knowledge matrix into a decision tree. Irrelevant decision factors are eliminated, and relevant factors are organized efficiently.

Rule induction offers many advantages. First, it allows knowledge to be acquired directly from example cases, thus avoiding the problems associated with acquiring knowledge from an expert through a knowledge engineer. Second, induction systems can discover new knowledge from the set of examples that may be unknown to the expert. Third, induction can uncover critical decision factors and eliminate irrelevant ones. In addition, an induction system can uncover contradictory results in the example set and report them to the expert.

Induction systems, however, suffer from several disadvantages. They do not select the decision factors of a problem. An expert is still needed to select the important factors for making a decision. They can generate rules that are difficult to understand. They are only useful for rule-based, classification problems. They may require a very large set of examples to generate useful rules. In some cases, the examples must be sanitized to remove exception cases. Additionally, the computing power required to perform the induction grows exponentially with the number of decision factors.

Neural Networks

Neural networks are a relatively new approach to building intelligent systems. The neural network approach is based on constructing computers with architectures and processing capabilities that attempt to mimic the architecture and processing of the human brain. A neural network is a large network of simple processing elements (PEs) that process information dynamically in response to external inputs. The processing elements are a simplified representation of brain neurons. The basic structure of a neural network consists of three layers: input, intermediate (called the *hidden layer)*, and output. Figure 4 depicts a simple three-layer network.

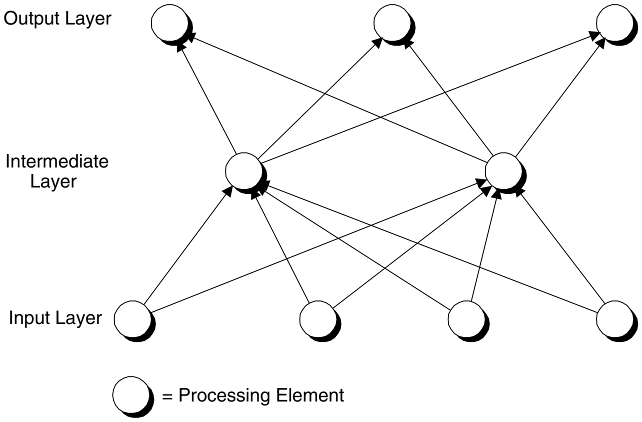


Figure 4. A three-layer neural network architecture. The layers of the network are the input, intermediate (hidden), and output layers.

Each processing element receives inputs, processes the inputs, and generates a single output. Each input corresponds to a decision factor. For example, for a loan approval application, the decision factors may be the income level, assets, or age. The output of the network is the solution to the problem. In the loan approval application, a solution may be simply a "yes" or "no." A neural network, however, uses numerical values only to represent inputs and outputs.

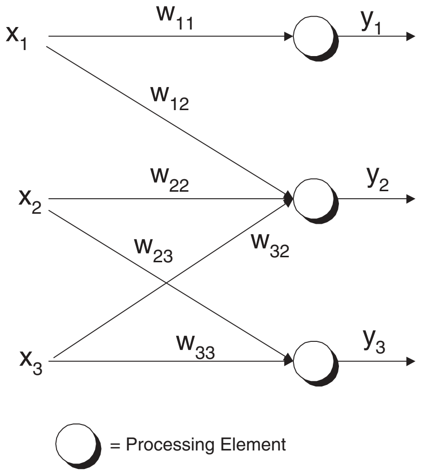
Each input x_i is assigned a *weight* w_i that describes the relative strength of the input. Weights serve to increase or decrease the effects of the corresponding x_i input value. A summation function multiplies each input value x_i by its weight w_i and sums them together for a weighted sum y. As Figure 5 illustrates, for j processing elements, the formula for n input is

$$y_j = \sum_j w_{ij} x_i$$

Based on the value of the summation function, a processing element may or may not produce an output. For example, if the sum is larger than a *threshold value* T, the processing element produces an output y. This value may then be input to other nodes for a final response from the network. If the total input is less than T, no output is produced. In more sophisticated models, the output will depend on a more complex activation function.

Learning in a Neural Network. The knowledge in a neural network is distributed in the form of internode connections and weighted links. These weights must be learned in some way. The learning process can occur in one of two ways: *supervised* and *unsupervised learning*.

In supervised learning, the neural network is repeatedly presented with a set of inputs and a desired output



$$y_1 = x_1w_{11}$$
$$y_2 = x_1w_{12} + x_2w_{22} + x_3w_{32}$$
$$y_3 = x_2w_{23} + x_3w_{33}$$

Figure 5. Summation function for several neurons.

response. The weights are then adjusted until the difference between the actual and the desired response is zero. In one variation of this approach, the difference between the actual output and the desired output is used to calculate new adjusted weights. In another variation, the system simply acknowledges for each input set whether the output is correct. The network adjusts weights in an attempt to achieve correct results. One of the simpler supervised learning algorithms uses the following formula to adjust the weights w_i:

$$w_i(\text{new}) = w_i(\text{old}) + \alpha \times d \times \frac{x_i}{|x_i|^2}$$

where α is a parameter that determines the rate of learning, and d is the difference between actual and desired outputs.

In unsupervised learning, the training set consists of input stimuli only. No desired output response is available to guide the system. The system must find the weights w_{ij} without the knowledge of a desired output response.

Neural networks can automatically acquire knowledge from historical data. In that respect they are similar to rule induction. They do not, however, need an initial set of decision factors or complete and unambiguous sets of data. Neural networks are particularly useful in identifying patterns and relationships that may be subsequently developed into rules for expert systems. Neural networks could also be used to supplement rules derived by other techniques.

Genetic Algorithms

Genetic algorithms refer to a variety of problem-solving techniques that are based on models of natural adaptation and evolution. They are designed the way populations adapt to and evolve in their environments. Members that adapt well are selected for mating and reproduction. The descendants of these members inherit genetic traits from both parents. Members of this second generation that also adapt well are selected for mating and reproduction, and the evolutionary cycle continues. After several generations, members of the resultant population will have adapted optimally or at least very well to the environment.

Genetic algorithms start with a fixed population of data structures that are candidate solutions to specific domain tasks. After requiring these structures to execute the specified tasks several times, the structures are rated for their effectiveness as a domain solution. On the basis of these evaluations, a new generation of data structures is created using specific "genetic operators" such as reproduction, crossover, inversion, and mutation. Poor performing structures are discarded. This process is repeated until the resultant population consists only of the highest performing structures.

Many genetic algorithms use eight-bit strings of binary digits to represent solutions. Genetic algorithms use four primary operations on these strings:

1. *Reproduction* is an operation that produces new generations of improved solutions by selecting parents with higher performance rating.
2. *Crossover* is an operation that randomly selects a bit position in the eight-bit string and concatenates the head of one parent with the tail of the second parent to produce a child. Consider two parents designated *xxxxxxxx* and *yyyyyyyy*, respectively. Suppose the second bit position has been selected as the crossover point (i.e., *xx*: *xxxxxx* and *yy*: *yyyyyy*). After the crossover operation is performed, two children are generated, namely *xxyyyyyy* and *yyxxxxxx*.
3. *Inversion* is a unary operation that is applied to a single string. It selects a bit position at random, and then concatenates the tail of the string to the head of the same string. For example, if the second position was selected for the following string ($x_1x_2 : x_3x_4x_5x_6x_7x_8$), the inverted string would be $x_3x_4x_5x_6x_7x_8x_1x_2$.
4. *Mutation* is an operation ensures that the selection process does not get caught in a local minimum. It selects any bit position in a string at random and changes it.

The power of genetic algorithms lies in that they provide a set of efficient, domain-independent search heuristics for a wide range of applications. With experience, the ability of a genetic algorithm to learn increases, enabling it to accumulate good solutions and reject inferior ones.

Analogical Reasoning and Case-Based Reasoning

Analogical reasoning is the process of adapting solutions used to solve previous problems in solving new problems. It is a very common human reasoning process in which new concepts are learned through previous experience with similar concepts. A past experience is used as a framework for solving the new analogous experience. Analogical learning consists of the following five steps:

1. Recognizing that a new problem or situation is similar to a previously encountered problem or situation.
2. Retrieving cases that solved problems similar to the current problem using the similarity of the new problem to the previous problem as an index for searching the case database.
3. Adapting solutions to retrieved cases to conform with the current problem.
4. Testing the new solutions.
5. Assigning indexes to the new problem and storing it with its solution

Unlike induction learning, which requires a large number of examples to train the system, analogical learning can be accomplished using a single example or case that closely matches the new problem at hand.

KNOWLEDGE ANALYSIS

After knowledge is collected, it must be interpreted and analyzed. First a transcript of the knowledge acquisition session is produced. This transcript is then reviewed and analyzed to identify key pieces of knowledge and their relationships. A variety of graphical techniques are used to provide a perspective of the collected knowledge and its organization (14).

Knowledge Transcription

After the knowledge collection phase, an exact and complete transcript of the knowledge acquisition session is usually made. This transcript is used as a basis for interpreting and analyzing the collected knowledge. Transcription can also be partial. In case of a partial transcription, notes taken during knowledge acquisition session can be used to guide the selection of what should be transcribed.

Each transcript is indexed appropriately with such information as the project title, session date and time, session location, attendees, and the topic of the session. A paragraph index number is assigned to cross-reference the source of knowledge extracted from the transcript with the knowledge documentation. This cross-referencing facilitates the effort of locating the source of knowledge if additional information is needed.

Knowledge Interpretation

Knowledge interpretation begins by reviewing the transcript and identifying the key pieces of knowledge or "chunks" (19). Usually declarative knowledge is easy to identify. Procedural knowledge is harder to recognize, as it can be scattered across the transcript, making it harder to relate. In addition to identifying key pieces of knowledge, an important goal of reviewing the transcript is to identify any issues that need further clarification by the expert.

Several techniques can be used in knowledge interpretation. These include (1) using handwritten notes taken during the knowledge acquisition session in knowledge identification, (2) highlighting of key information in the transcript using word processing software features or a pen, and (3) labeling each piece of knowledge with the type of knowledge it represents.

Knowledge Analysis and Organization

After identifying the different types of knowledge, they need to be analyzed and classified. This effort includes the following steps:

1. Recording each identified piece of knowledge with other related pieces in the *knowledge dictionary*. A knowledge dictionary is a repository that maintains, in alphabetical order, a description of each type of knowledge, for example, objects, rules, problem-solving strategies, and heuristics.
2. Organizing, classifying, and relating the pieces of knowledge collected with similar knowledge stored in the knowledge dictionary. This is a complex iterative step that requires the involvement of the expert to confirm and help refine the structure of knowledge developed.
3. Reviewing the collected knowledge to identify those areas that need further clarification.

Graphical techniques that show how the different pieces of knowledge are related are particularly useful. The next section overviews some of the knowledge representation methods that support both the knowledge engineer and the expert in analyzing knowledge.

KNOWLEDGE REPRESENTATION

Knowledge acquired from experts and other sources must be organized in such a way that it can be implemented and accessed whenever needed to provide problem-solving expertise. Knowledge representation methods can be classified into two broad types: those that support the analysis of the acquired knowledge and the development of a conceptual model of the expert system, and those that support the implementation formalism of the development environment.

The first type of representation, called *intermediate representation*, allows knowledge engineers to focus on organizing, analyzing, and understanding the acquired knowledge without concerning themselves with the representation formalisms of the implementation environment. The intermediate representation is continually refined and updated through additional knowledge acquisition until the knowledge engineers are satisfied they have a sufficiently complete model to guide the implementation design. Intermediate representation methods are usually pictorial and include flowcharts, graphs, semantic networks, scripts, fact tables, decision tables, and decision trees.

The second type of representation, called the *implementation representation*, is used to create an implementation design for the chosen development environment. The conceptual model is mapped directly into the representation model of the development environment without the need to understand the function that the knowledge should serve. Implementation representation often used includes frames or production rules.

Each representation method emphasizes certain aspects of the knowledge represented. The choice of a representation method will depend on how well the representation schemes support the structure of the problem. We consider in this article eight of the most common knowledge representation techniques:

- Logic.
- Production rules.
- Frames.
- Semantic networks.
- Objects–attribute–value triplets.
- Scripts.
- Decision tables.
- Decision trees.

Logic

Logic is the oldest form of knowledge representation. It uses symbols to represent knowledge. Operators are applied to these symbols to produce logical reasoning. Logic is a formal well-grounded approach to knowledge representation and inferencing. There are several types of logic representation techniques. The two approaches used in artificial intelligence and expert system development are *propositional logic* and *predicate calculus*.

Propositional Logic. A proposition is a statement that is either true or false. Symbols, such as letters, are used to represent different propositions. For example, consider propositions A and B used to derive conclusion C:

A	=	Employees work only on weekdays
B	=	Today is Saturday
C	=	Employees are not working today

Propositional logic provides logical operators such as AND, OR, NOT, IMPLIES, and EQUIVALENCE that allows reasoning using various rule structures. Table 3 lists the propositional logic operators and their common symbols.

The AND operator combines two propositions and returns true if both propositions are true. The OR operator combines two propositions and returns true if either one or both propositions are true. The NOT operator is a unary operator that returns false if proposition A is true; otherwise it returns true if proposition A is false. The EQUIVALENCE operator returns true when both propositions have the same truth assignment. The IMPLIES operator indicates that if proposition A is true, then proposition B is also true.

Table 3. Logical Operators and Their Symbols

Operator	Symbol
AND	$\wedge$, & , $\cap$
OR	$\vee$, $\cup$, +
NOT	$\neg$, ~
IMPLIES	$\supset$, $\rightarrow$
EQUIVALANCE	$\equiv$

Table 4. Truth Table for IMPLIES Operator

A	B	A → B
T	T	T
T	F	F
F	T	T
F	F	T

A truth table is used to show all possible combinations of an operator. Table 4 shows the truth table for the IMPLIES operator.

Since propositional logic deals only with the truth of complete statements, its ability to represent real-world knowledge is limited.

Predicate Calculus. Predicate calculus is an extension of propositional logic that provides finer presentation of knowledge. It permits breaking down a statement into the objects about which something is being asserted and the assertion itself. For example, in the statement *color* (*sky*, *blue*), the objects sky and blue are associated through a color relationship.

Predicate calculus allows the use of variables and functions of variables in a statement. It also uses the same operators used in propositional logic in addition to two other symbols, the universal quantifier $\forall$ and the existential quantifier $\exists$, that can be used to define the range or scope of variables in an expression. Inferencing capability in predicate calculus is accomplished through the use of these operators.

As predicate calculus permits breaking statements down into component parts, it allows for a more powerful representation model that is more applicable to practical problems.

Production Rules

Production rules are a popular knowledge representation scheme used for the development of expert systems. Knowledge in production rules is presented as condition-action pairs: IF a *condition* (also called antecedent or premise) is satisfied, THEN an action (or consequence or conclusion) occurs. For example:

IF the sky is clear
THEN it is not going to rain

A rule can have multiple conditions joined with AND operators, OR operators, or a combination of both. The conclusion can contain a single statement or several statements joined with an AND. A certainty factor, usually a value between −1 and 1, can also be associated with a rule to capture the confidence of the expert with the results of the rule (20).

Production rules represent the system's knowledge base. Each rule represents an independent portion of knowledge that can be developed and modified independently of other rules. An inference mechanism uses these rules along with information contained in the working memory to make recommendations. When the IF portion of a rule is satisfied, the rule fires and the statements in the THEN part of the rule are added to the working memory. These statements can trigger other rules to fire. This process continues until the system reaches a conclusion.

Production rules offer many advantages. They have simple syntax, are easy to understand, and are highly modular, and their results are easily inferred and explained. Production rules are, however, not suitable for representing many types of knowledge, particularly descriptive knowledge. They could also be difficult to search, control, and maintain for large complex systems.

Semantic Networks

Semantic networks are graphical depictions of a domain's important objects and their relationships. It consists of nodes and arcs that connect the nodes. The nodes represent the objects and their properties. Objects can represent tangible or intangible items such as concepts or events. The arcs represent the relationships between the objects. Some of the most common arc types are the IS-A and HAS-A type. The IS-A relationship type is used to show class membership; that is, an object belongs to a larger class of objects. The HAS-A relationship type indicates the characteristics of an object.

Figure 6 shows a simple example of a semantic network. In this example, the "Pyramid" node is connected to a property node, indicating that "a pyramid has faces." It is also connected to the "Structure" node via an IS-A link, indicating that "a pyramid is a structure." The "Structure" node is connected to a "Material" node via a MADE OF link, and the "Stone," "Wood," and "Steel" nodes are connected to the "Material" node via an IS-A link.

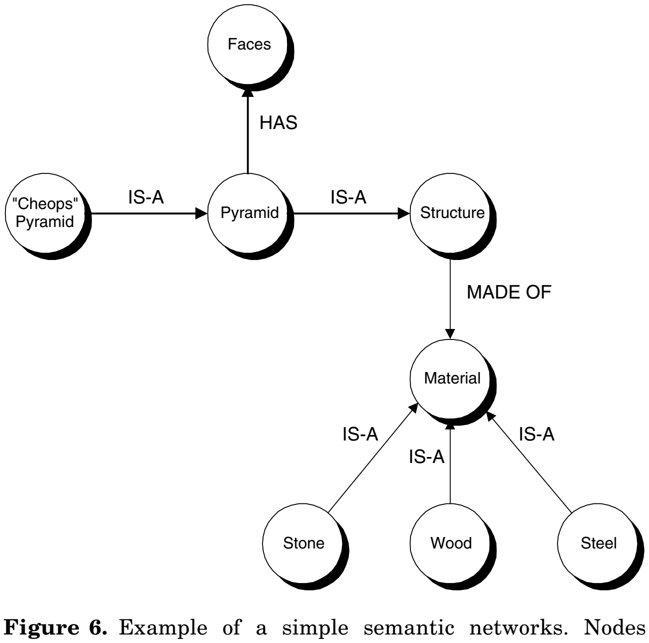


Figure 6. Example of a simple semantic networks. Nodes represent objects, and links represent the relationship between the objects.

A very useful characteristic of semantic networks is the concept of *inheritance*. Inheritance is the mechanism by which nodes connected to other nodes through an IS-A relationship inherit the characteristics of these nodes. A main advantage of inheritance is that it simplifies adding new knowledge to the network. When a new node is added, it inherits a wealth of information throughout the network via the IS-A links. Similarly, when a general node is added (e.g., the "Structure" node), other nodes inherit its properties.

Semantic networks have many advantages as a knowledge representation scheme. They are easy to understand and provide flexibility and economy of effort in adding new objects and relationships. They provide a storage and processing mechanism similar to that of humans, and the inheritance mechanism provides an efficient way of inferencing. Semantic networks also have several limitations. Exceptions offer potential difficulty to the mechanism of inheritance, and because semantic networks do not represent sequence and time, procedural knowledge is difficult to represent.

Frames

A frame is a data structure that includes both declarative and procedural knowledge about a particular object. In that respect, frames are similar to objects used in object-oriented programming. A frame consists of a collection of *slots* that may be of any size and type. Slots have a name and any number of subslots called *facets*. Each facet has a name and any number of values. Figure 7 depicts a simple frame for "Cheops" pyramid.

Facets contain information such as attribute value pairs, default values, conditions for filling a slot, pointers to other related frames, functions, and procedures that are activated under different conditions. The conditions that can activate a procedure are specified in the IF-CHANGED and IF-NEEDED facets. An IF-CHANGED facet contains a procedural attachment, called a *demon*. This procedure is invoked when a value of a slot is changed. An IF-NEEDED facet is used when no slot value is given. It specifies a procedure that is invoked to compute a value for the slot.

For example, the "Cheops" pyramid frame of Figure 7 has attribute value slots (A-KIND-OF, MATERIAL, BASE-LENGTH, HEIGHT), slots that take default values (NO.-OF-FACES and NO.-OF-SATTELITES), and slots

```
("Cheops" Pyramid
              (A-KIND-OF(VALUE pyramid))
              (MATERIAL(VALUE limestone granite))
              (BASE-LENGTH(VALUE 233m))
              (HEIGHT(VALUE 146m))
              (NO.-OF-FACES(DEFAULT fget))
              (ANGLE(VALUE if-needed))
              (BASE-AREA(VALUE if-needed))
              (VOLUME(VALUE if-needed))
              (NO.-OF-SATTELITES(DEFAULT fget))
```

Figure 7. Example of a frame for "Cheops" pyramid. This frame illustrates different types of slots.

with attached IF-NEEDED procedures (ANGLE, BASE-AREA, VOLUME). The value fget in the default values slots is a function call that retrieves a default value from another frame such as the general pyramid frame for which "Cheops" is a KIND-OF. When activated, the fget function recursively looks for default values for the slot from ancestor frames until one is found.

Frames are usually connected together to form a hierarchical structure. This hierarchical arrangement of frames allows inheritance. Each frame inherits the characteristics and behavior of all related frames at higher levels of the hierarchy. For example, the "Cheops" pyramid frame is linked to a general pyramid frame that contains information common to all pyramids. In this case, the "Cheops" pyramid frame inherits all the descriptive and procedural information of the pyramid frame.

Inferencing in frames is based on the premise that previous experiences with objects and events create certain expectations about newly encountered objects and events. First, knowledge about an object or situation is stored in long-term memory as a frame. Then, when a similar object or situation is encountered, an appropriate frame is retrieved from memory and used for reasoning about the new situation.

Frames have many advantages. They are a powerful mechanism for representing knowledge, because both declarative and procedural information are captured. In addition, slots for new attributes and procedures are easy to set up. Frames have, however, a complicated reasoning. As a result, the implementation of their inferencing mechanism is difficult.

Objects–Attribute–Value Triplets

An object, attribute, and value triplet, also known as the *O–A–V triplet*, is another way of representing knowledge. *Objects* can represent physical or abstract items. *Attributes* are properties of the objects, and *values* are specific values that an attribute has at a given time. An attribute can have single or multiple values. These values can be static or dynamic. Figure 8 illustrates a simple O–A–V triplet.

O–A–V triplets can be considered as a variation of either the semantic networks or the frames. They are useful in depicting a relationship between objects, such as inheritance, part-of, and causal relationships.

Scripts

Scripts are frame-like structures used to represent stereotypical situations such as eating in a restaurant, shopping in a supermarket, or visiting a doctor. Similar to a script for a play, the script structure is described in terms of roles, entry conditions, props, tracks, and scenes. *Roles* refer to the people involved in the script. *Entry conditions* describe the conditions that must be satisfied before the events described in the script can occur. *Props* are the items used in the events of the script. *Track* refers to variations that might occur in a particular script. Finally, *scenes* are the sequence of events that take place for the script situation. Figure 9 depicts a typical script. It is adapted from the well-known restaurant example used to show how knowledge is represented in scripts.

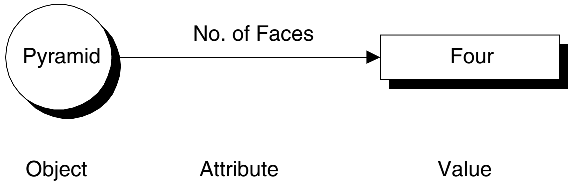


Figure 8. Example of a simple O–A–V triplet.

Script Name	:	Restaurant
Track	:	Fast-food restaurant
Roles	:	Customer
		Server
Props	:	Counter
		Tray
		Food
		Money
		Napkins
		Salt/Pepper/Catsup/Straws
Entry Conditions	:	Customer is hungry
		Customer has money
Scene 1	:	Customer parks car
		Customer enters restaurant
		Customer waits in line at counter
		Customer reads the menu on the wall and makes a decision about what to order
Scene 2	:	Customer gives order to server
		Server fills order by putting food on tray
		Customer pays server
Scene 3	:	Customer gets napkins, straws, salt, etc.
		Customer takes tray to an unoccupied table
		Customer eats food quickly
Scene 4	:	Customer cleans up table
		Customer discards trash
		Customer leaves restaurant
		Customer drives away
Results	:	Customer is no longer hungry
		Customer has less money

Figure 9. Example of a restaurant script.

Similar to frames, reasoning with scripts begins with the creation of a partially filled script that describes the current situation. A known script with similar properties is retrieved from memory using the script name, preconditions, or any other keywords as index values for the search. The slots of the current situation script are then filled with inherited and default values from the retrieved scripts.

Scripts offer many of the advantages of frames, particularly the expressive power. However, similar to frames, they and their inference mechanisms are difficult to implement.

Decision Tables

A decision table is a two-dimensional table that enumerates all possible combinations of attribute values and the conclusions that can be made for each combination of these values. An example of a decision table is shown in Fig. 10. This example gives an expert's recommendations for

Attributes								
Age	Y[1]	Y	Y	Y	O[2]	O	O	O
Investment Amount	S[3]	S	L[4]	L	S	S	L	L
Investment Style	C[5]	A[6]	C	A	C	A	C	A
Conclusions								
Savings	X	X			X	X		
Portfolio 1				X				
Portfolio 2			X					X
Portfolio 3							X	

[1]Y=Young [3]S=Small [5]C=Conservative
[2]O=Old [4]L=Large [6]A=Aggressive

Figure 10. Example of a decision table for an investment portfolio advisor.

investment decisions based on age, amount of investment, and investment style.

Decision tables are suitable for a small number of decision attributes, each with a small number of possible values. If the number of attributes or possible values is large, the decision table becomes quite complex. Decision tables are suitable as an intermediate representation for documenting and analyzing knowledge. It is not possible to make inferences directly from the tables, except through rule induction.

Decision Trees

Decision trees are a graphical representation of a problem domain search space. A decision tree is composed of nodes and branches. Initial and intermediate nodes represent decision attributes, and leaf nodes represent conclusions. A path from the root node to a leaf node corresponds to a decision path that might be encountered in the problem domain. Figure 11 shows the decision tree version of the problem presented as a decision table in Fig. 10.

Decision trees are useful not only to show the problem-solving steps, but also the order in which input data are requested and the reasoning steps the expert system should take in order to reach a conclusion. Decision trees are more natural for experts to understand and use than formal methods such as rules of frames. They are particularly useful to represent the knowledge of identification systems (diagnostics, troubleshooting, classification, etc.).

VALIDATION AND VERIFICATION OF KNOWLEDGE

An important activity of knowledge acquisition is the testing and evaluation of the quality and correctness of the acquired knowledge and its implementation. This activity can be separated into two components: validation and verification (21).

Validation refers to determining whether the "right" system was built, i.e., whether the system does what it was meant to do at an acceptable level of accuracy. Validating the knowledge involves confirming the acquired knowledge is sufficient to perform the task at a sufficient level of expertise.

Verification refers to determining whether the system was built "right," i.e., whether the system correctly implements its specifications. Verifying a system means that the program accurately implements the acquired knowledge as acquired and documented.

Validation and verification of knowledge are highly interrelated. Errors in the knowledge implementation are often discovered during validation when the acquired knowledge is checked to see whether it performs the desired task at a sufficient level of expertise.

Validation and Verification as Part of Knowledge Acquisition

Since expert systems are developed iteratively, they inherently include repeated validation and verification testing as part of the development process. Each time a version of the

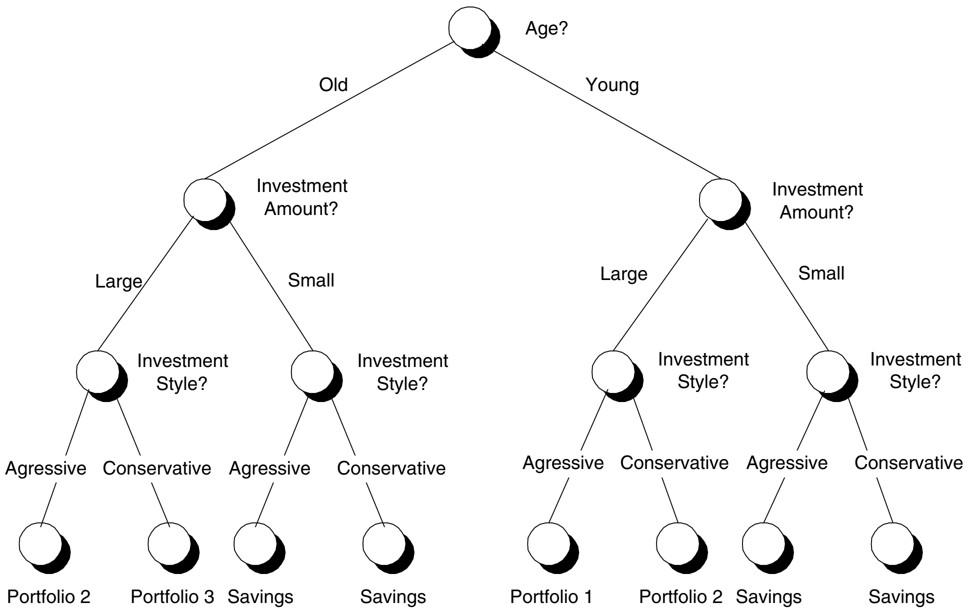


Figure 11. Example of a decision tree for the investment portfolio advisor of Fig. 10.

expert system program is run to test the knowledge, the correctness of the program is checked as well. Thus, in addition to finding deficiencies in the acquired knowledge, the knowledge acquisition cycle detects and corrects programming errors. Validation and verification during knowledge acquisition can occur before implementation has begun using manual simulation or after initial implementation by testing the evolving prototype.

Validation Using Manual Simulation. Early in the expert system development project and before implementation has begun, knowledge acquisition follows a basic development cycle: (1) eliciting knowledge; (2) interpreting, analyzing, and organizing acquired knowledge; and (3) testing knowledge. In this approach, a test case is analyzed by the expert and manually using hand simulation of the acquired knowledge. The results of the expert's analysis are compared with those of the hand simulation. If the results differ, the appropriate area of knowledge is revised and corrected. This process is repeated until no discrepancies occur between the expert's analysis and the results of the simulation of the acquired knowledge.

Validation Using Evolving Prototype. When enough knowledge is acquired to allow a prototype implementation, the knowledge acquisition process follows a modified cycle consisting of the following steps: (1) eliciting knowledge; (2) interpreting, analyzing, and organizing acquired knowledge; (3) implementing knowledge; and (4) testing knowledge. During the testing phase, a test case is presented to the expert and run using the evolving prototype. The results of the expert's analysis are compared against the results of the prototype. If the results differ, the portion of the knowledge that produced the discrepancy is identified and is manually simulated to see whether it agrees with the expert's analysis. If manual simulation produces results that agree with the expert, then an implementation error is likely the source of the discrepancy. If manual simulation does not agree with the expert's analysis, acquired knowledge is revised, modified, or expanded until it comes into agreement with the expert analysis. This process is repeated throughout the knowledge acquisition phase.

Validation testing during expert system development could be conducted by the domain expert or by a group of consulting experts. Using multiple experts has the advantage of removing potential biases of single experts, and will generally reveal and correct more errors in the expert system's knowledge and implementation. It also provides the nontechnical benefit of adding credibility to the validation effort.

On the other hand, multiple experts might disagree and provide contradicting opinions. In that case, one of several approaches can be used to integrate the expert's opinions (22). These techniques include selecting the majority decision; blending different lines of reasoning through consensus methods, such as Delphi; applying analytical models used in multiple-criteria decision making; selecting a specific line of reasoning based on the situation; and using blackboard systems that maximize the independence among knowledge sources by appropriately dividing the problem domain.

Validation of the Developed Expert System

In some domains, the correctness of the expert system recommendation can be trivially determined without the need for comparison against the human expert's judgment. In other domains, the correctness of the results needs to be confirmed by experts who will generally agree on the quality of the system's recommendations.

Validation of the developed system is accomplished by comparing the developed system's operational results against the judgment of the expert. A variation of this approach is to run a number of test cases on the developed system and compare the system's recommendations against the results obtained by the human experts.

If feasible, it is highly recommended to evaluate the performance of the expert system in the field under actual operating conditions. This approach provides the most realistic validation of the system in addition to, if tests are successful, convincing potential users of the value of the system.

As in the case of validating an expert system during development, validating a developed expert system can be accomplished using a single expert or multiple experts. These are usually the same experts that performed validation testing during the expert system development.

Verification of the Expert System Program

Verification ensures that the program accurately implements the acquired knowledge. The knowledge acquisition process by its nature uncovers errors not only in the knowledge but in the implementation as well. Implementation errors are often identified during validation when the knowledge of the system is checked for correctness.

In addition to ensuring that the coded knowledge reflects the documented knowledge accurately, verification requires checking the expert system program for internal errors in the knowledge base and the control logic that provides the inferencing mechanism. For example, a rule-based system should not have redundant, conflicting, inconsistent, superfluous, subsumed, or circular rules. In frame-based systems, there should not be any slot with illegal values, inheritance conflicts that are unresolved, circular inheritance paths, and so on. Most rule-and frame-based systems provide capabilities for checking many of these potential problems. Other testing methods should be employed for potential problems not checked automatically. Software systems with better testing and error detection capability enhance the verification phase of the system. Verifying the control logic that performs inferencing can be minimized if the project is using a standard, commercial off-the-shelf tool.

BIBLIOGRAPHY

1. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat (eds.), *Building Expert Systems*, Reading, MA: Addison-Wesley, 1983.
2. R. E. Nisbett and T. D. Wilson, Telling more than we can know: Verbal reports on mental processes, *Psychological Review*, **84**: 231–259, 1977.

3. N. Dixon, *Preconscious Processing*, Chichester: John Wiley & Sons, 1981.
4. H. M. Collins, *Changing Order: Replication and Induction in Scientific Practice*, London: Sage, 1985.
5. L. Bainbridge, Asking questions and accessing knowledge, *Future Computing Systems*, **1**: 143–149, 1986.
6. E. Turban, J. E. Aronson, and T-P. Liang, *Decision Support Systems and Intelligent Systems*, Upper Saddle River, NJ: Prentice Hall, 2005.
7. K. L. McGraw and K. Harbison-Briggs, *Knowledge Acquisition: Principals and Guidelines*, Englewood Cliffs, NJ: Prentice-Hall, 1989.
8. D. S. Prerau, *Developing and Managing Expert Systems: Proven Techniques for Business and Industry*, Reading, MA: Addison-Wesley, 1990.
9. A. C. Scott, J. E. Clayton, and E. L. Gibson, *A Practical Guide to Knowledge Acquisition*, Reading, MA: Addison-Wesley, 1991.
10. J. Evans, The knowledge elicitation problem: A psychological perspective, *Behavior and Information Technology*, **7** (2): 111–130, 1988.
11. J. Durkin, *Expert Systems: Design and Development*, New York: McMillan, 1994.
12. D. D. Wolfgram, *Expert System*, New York: John Wiley & Sons, 1987.
13. G. A. Kelly, *The Psychology of Personal Constructs*, New York: Norton, 1955.
14. J. H. Boose, *Expertise Transfer for Expert Systems Design*, New York: Elsevier, 1986.
15. A. J. Diederich, A. I. Ruhmann, and A. M. May, Kriton: A knowledge acquisition tool for expert systems, *International Journal of Man-Machine Studies*, **26** (1): 29–40, 1987.
16. J. H. Boose and J. M. Bradshaw, Expertise transfer and complex problems: Using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems, *International Journal of Man-Machine Studies*, **26** (1): 3–28, 1987.
17. M. Freiling, J. Alexander, S. Messick, S. Rehfuss, and S. Shulman, Starting a knowledge engineering project: a step-by-step approach, *The AI Magazine*, 150–164, 1985.
18. P. R. Cohen and E. A. Feigenbaum, *The Handbook of Artificial Intelligence*, vol. 3, Reading, MA: Addison-Wesley, 1982.
19. J. Bell and R. J. Hardiman, The third role—the naturalistic knowledge engineer, in D. Diaper (ed.), *Knowledge Elicitation: Principles, Techniques and Applications*, New York: John Wiley & Sons, 1989.
20. E. Shortliffe and B. G. Buchanan, A model of inexact reasoning in medicine, *mathematical biosciences*, **23**: 351–375, 1968.
21. R. M. O'Keefe, O. Balci, and E. P. Smith, Validating expert system performance, *IEEE Expert*, **2** (4): 81–90, 1987.
22. S. M. Alexander and G. W. Evans, The integration of multiple experts: a review of methodologies, in E. Turban and P. Watkins (eds.), *Applied Expert System*, Amsterdam: North Holland, 1988.

MAGDI N. KAMEL
Naval Postgraduate School
Monterey, California.

KNOWLEDGE-BASED COMPUTATION

INTRODUCTION

The field of artificial intelligence (AI) studies the computational requirements for performing tasks such as perception, reasoning, and learning (1). Knowledge appears to play a key role in achieving high-level performance on many human tasks, as people draw on background knowledge and pick strategies to apply that knowledge. The *knowledge-based computation* approach to AI studies how to develop intelligent systems that exploit explicitly represented knowledge, and it sees knowledge as the crucial determiner of system performance.

AI theories depend on specifying processes, domain content, and representations for that content. Research in knowledge-based computation addresses questions such as how knowledge should be represented in computational systems for particular tasks, which knowledge must be captured for a particular task and task domain, how knowledge should be organized and accessed, and how knowledge can be applied to the task itself. Applications of knowledge-based computation have been fielded in a wide range of areas, demonstrating the practical value of knowledge-based approaches.

The phrase "knowledge-based systems" is often used to describe rule-based expert systems, which replicate expert performance with the "narrow but deep" knowledge required for high-level performance in focused domains. These systems provided early and visible successes, and they continue to have great impact. However, knowledge-based computation also includes additional knowledge-based methods, such as model-based reasoning, which exploits models of structure and behavior, and case-based reasoning, which exploits stored records of specific problem-solving episodes. The tasks amenable to knowledge-based techniques can go beyond problem solving to include areas such as story understanding, planning, diagnosis, explanation, and learning.

Knowledge-based methods contrast with conventional programming, in which knowledge about the task domain is often implicit in the design of the program and its specific mechanisms, rather than reflected in an explicit form accessible to system manipulation. They also contrast with AI approaches in which knowledge is not explicitly represented, such as neural network models that capture knowledge in a distributed form. Using explicit representations can facilitate the addition of specific new pieces of knowledge, can aid in explaining system behavior, and can facilitate examination of the systems' knowledge state. Examination of the system's knowledge state can be useful for outside observers, increasing their confidence in the system by enabling them to understand or confirm the system's decisions. It may also be useful for the system to examine its own knowledge and to perform metareasoning—reasoning about its own reasoning process—to guide or refine its own internal processes.

This article begins by summarizing some central hypotheses proposed as theoretical foundations for knowledge-based computing. Next, because a crucial question for knowledge-based computation is how knowledge should be represented, it discusses principles for knowledge representation and illustrates some ways those principles are realized. It then describes a sampling of major currents of knowledge-based computing, highlighting their issues, strengths, and challenges.

FUNDAMENTAL PRINCIPLES

Groundwork for knowledge-based computation was laid by research in the symbolic computation paradigm articulated by Newell and Simon in the early days of AI. Their Physical Symbol Systems hypothesis proposed that "a physical symbol system has the necessary and sufficient means for general intelligent action" ((2): 116). They describe physical symbol systems as machines existing within a larger world of objects, which "[produce] through time an evolving collection of symbol structures," whose symbols relate to objects that they *designate:* Given the symbol, the system can either affect the object, or can behave in ways that depend on the object. Physical symbol systems can *interpret* expressions, executing the process an expression designates.

Given the Physical Symbol Systems hypothesis, a key question is how such systems accomplish intelligent action. Newell and Simon's Heuristic Search Hypothesis proposes that they do so by search: "generating and progressively modifying structures until [they produce] a solution structure" ((2): 120).

As the field of AI addressed new tasks, it became clear that general reasoning methods have wide applicability, but that their success is crucially bound to the specific knowledge that they apply, which gave rise to the view that "knowledge is power," articulated by Lenat and Feigenbaum (3). On this view, a small set of general-purpose reasoning methods is sufficient for achieving high-level performance in a wide range of domains—provided the reasoning system has the right knowledge. The *knowledge principle* states:

> A system exhibits intelligent understanding and action at a high level of competence primarily because of the *specific* knowledge that it can bring to bear: the concepts, facts, representations, methods, models, metaphors, and heuristics about its domain of endeavor (3).

Lenat and Feigenbaum proposed that a first important part of problem solving is formulating a representation of the problem. As knowledge is then added to the system, it first reaches the Competence Threshold, and then—through the addition of more rarely used knowledge—the Total Expert Threshold, at which point the system's knowledge is sufficient to handle rare problems.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

KNOWLEDGE REPRESENTATION

Principles for Knowledge Representation Schemes

For programs to manipulate and exploit knowledge, their knowledge must be represented in a suitable form within the computer. Consequently, the development of knowledge-based systems is inextricably tied to the development of the knowledge representations they will use. Davis et al. (4) propose five roles for knowledge representations:

1. *A surrogate:* Given a representation scheme, a system may explore effects of actions by manipulating the representations rather than the objects themselves.
2. *A set of ontological commitments:* The representation scheme determines which concepts and relationships can exist for the system, determining which features of the system's domain will be preserved, abstracted, or ignored.
3. *A fragmentary theory of intelligent reasoning:* The representation scheme is associated to a theory of reasoning with that scheme.
4. *A medium for efficient computation:* The representation scheme must support the types of reasoning required for its intended uses.
5. *A medium of human expression:* The scheme must support expression of the desired information and must support human encoding and understanding of represented knowledge.

Selection of the right knowledge representation scheme can play a crucial role in the success of knowledge-based systems and the types of questions that they can address. For example, qualitative models use coarse-grained representations to enable commonsense reasoning (5).

Logic and Knowledge Representation

The field of logic studies formal languages, truth conditions, and rules for deriving conclusions (see *Formal Logic*). The use of logic to represent knowledge long predates AI, and with the advent of AI, logic was applied to AI knowledge representation to provide a formal structure for knowledge and reasoning, addressing the questions of what form a representation should take and what inferences are sanctioned.

The language of first-order logic includes predicates denoting propositions, logic operators such as $\wedge$ (and), $\neg$ (not), and $\Rightarrow$ (implies) (note that $\vee$ (or) can be derived from $\wedge$ and $\neg$), functions, variables, and quantifiers $\exists$ (there exists) and $\forall$ (for all) to form expressions. Values for the variables are selected from an agreed on universe of discourse. For example, an assertion that all animals covered with hair are mammals could be expressed as:

$$\forall x((animal(x) \wedge has\,Hair(x)) \Rightarrow mammal(x))$$

given the predicates *mammal, animal*, and *hasHair*.

Inference rules license the formation of conclusions. For example, *modus ponens* asserts that given two propositions P and Q, and given the rule $(P \Rightarrow Q)$ (which represents "P implies Q"), then if P is known to be true, Q is true as well. In many real-world situations, general rules have exceptions; in a medical domain, symptom *X* might suggest disease *Y*—unless symptom *Z* has been observed as well. This process motivates research on how to enable default reasoning in a logical framework.

A long-standing current of knowledge representation research focuses on the problem of formalizing common sense knowledge. This challenge was highlighted in 1959 by McCarthy (6), and has been addressed in efforts such as Hayes' ontology for liquids (7). For a fuller discussion of the logic-based perspective on knowledge representation and reasoning, see Brachman and Levesque (8). Formalization of common sense knowledge is examined by Davis (9) and plays a key role in the Cyc project (10), which aims to accumulate a knowledge base spanning human consensus knowledge.

Semantic Networks

Semantic networks (11) represent knowledge as a network of labeled nodes and arcs, facilitating graphical visualization of knowledge for human inspection as well as providing an indexing structure for retrieval of particular types of knowledge for automated reasoning. Semantic networks can capture classification knowledge, with nodes representing categories and arcs representing subcategory relations in a hierarchical structure from the most generic categories to specific instances. An illustration is a zoological taxonomy (e.g., a mammal is an animal is a living thing). Specific graphical notations may be used to represent aspects of first-order logic (12). A simple example of a semantic network fragment, involving statements about tigers, mammals, and carnivores, is shown in Fig. 1.

Semantic networks may also be used for computation. For example, networks may pass messages in the form of tokens or markers from node to node, for tasks such as identifying relationships or managing expectations during parsing or language understanding, (13). Sowa's conceptual graphs (14) illustrate the use of semantic networks to model the semantics of natural language. A conceptual graph represents a proposition as a set of labeled nodes and unlabeled arcs. In the graphical form, rectangles represent concepts and ovals represent conceptual relations; extensions support the use of boxes for nesting conceptual graphs, to encode complex natural language propositions to represent statements about propositions. For example, "Jim believes that tigers are mammals" includes both the proposition that tigers are mammals and the proposition that Jim believes the former proposition.

Conceptual Dependency Theory: Primitives of Meaning

Developing a knowledge representation scheme requires selecting the basic units of meaning. Schank's Conceptual Dependency (CD) Theory (15) provides a concrete example of how the requirements for a knowledge representation are reflected in the primitives chosen to represent actions

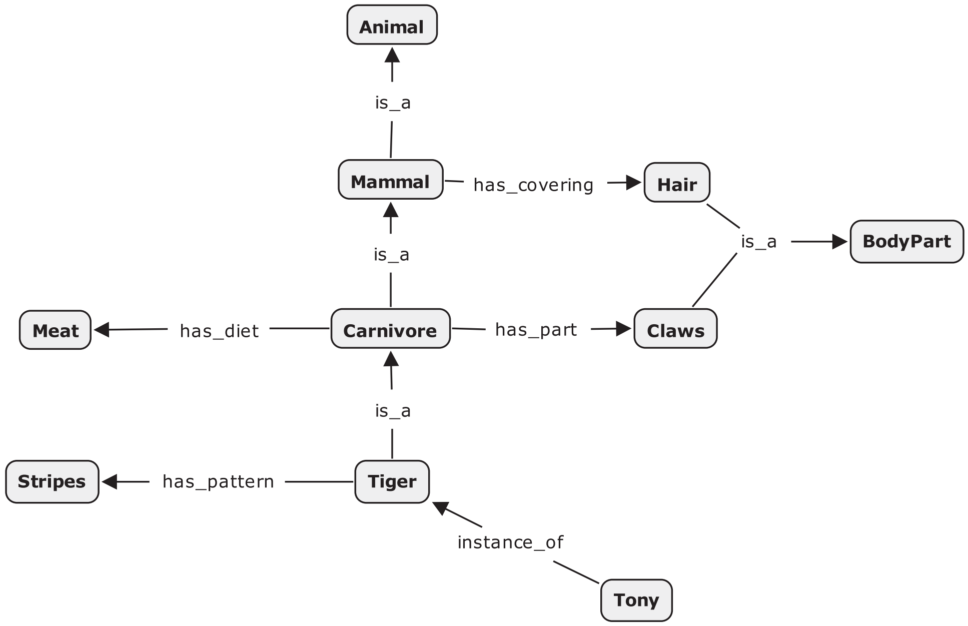


Figure 1. A sample semantic network about mammals.

for a particular domain. CD theory is both a theory of the requirements for designing sets of primitives and an example of the application of the proposed theory to develop a specific set of primitives.

CD theory aims to represent everyday actions to support the task of story understanding—establishing the coherence of stories by filling in their causal connections. CD theory distills everyday actions into a small set of 12 "primitive acts," listed in Table 1. PTRANS stands for Physical TRANSfer (of location), and underlies verbs such as "to go," "to walk," "to run," "to drive," "to fly," and so on. PROPEL describes the act of causing a force to be applied to an object in a specified direction, and underlies verbs such as "push," "throw," and "shoot." ATRANS describes transfer of possession, underlying verbs such as "buy," "sell," and "lend." MTRANS describes mental transfers (of information), underlying verbs such as "read," "listen," and "hear;" MBUILD describes formation of conclusions. ATTEND describes focusing a sense organ on a stimulus (as in "listen" or "look"); SPEAK describes production of sounds; INGEST, taking something into the body (as in "eat," "breathe," and "inject"); EXPEL, expelling from the body; MOVE, moving a body part; and GRASP, grasping an object. The names of the primitives are selected both to suggest their meaning to humans and to avoid ambiguities of natural language. A final primitive, DO, is used to represent unspecified actions.

The CD acts can depend on other acts in two ways, by causality (one causes another, enables another, motivates another, and so on) or by instrumentality. For example, walking somewhere can be represented as a PTRANS to that location, with the instrumental action of MOVEing the feet. Taking in textual information by reading visually is represented by MTRANS with an instrumental ATTEND of the eyes, whereas the representation for reading braille includes an ATTEND of the hand.

Each primitive is associated with a structure of "slots" to fill to describe an act, providing expectations. CD allowed a highly limited set of slots, including the ACTOR of the act, the OBJECT of the act, the direction (FROM and TO) and the INSTRUMENTAL ACTION by which the act was performed. Each primitive could be associated with inferences (e.g., that at the end of a PTRANS, the OBJECT of the PTRANS was at the location specified by the TO slot).

The structure of CD provides expectations that were used to guide conceptual parsing of natural language text, representation, and inferencing within a line of story understanding systems (16). The small number of primitives facilitated connecting events by inference chaining, by limiting the set of inference procedures needed.

Frames, Frame Systems, and Scripts

Knowledge representation schemes may also collect information into larger units. Minsky's theory *of frames*, based on the premise that units of knowledge should be large and structured, proposed the use of large-scale structures for representing stereotyped situations, such as being in a living room or at a child's birthday party. Frames can be seen as networks of nodes and connections, linking together associated information such as expectations, responses to expectation failures, and viewpoints. The slots of frames are associated with default information, recommending standard inferences and enabling a frame system to draw defeasible conclusions about information that may not be confirmable deductively. For example, birthday parties often include the presentation of gifts, but the assumption that gifts were given may not hold in a particular case. Numerous frame systems have been developed to support knowledge storage, retrieval, and inference, providing general tools that may be used to manage the knowledge for task-specific systems.

Table 1. The Conceptual Dependency Theory Primitive Actions

PTRANS	MTRANS	SPEAK	MOVE
PROPEL	MBUILD	INGEST	GRASP
ATRANS	ATTEND	EXPEL	DO

Schank and Abelson's Script theory (17) illustrates the use of large-scale knowledge structures for story understanding. During story understanding, many inferences could be generated in principle; which ones are actually appropriate is context-dependent. For example, it is reasonable to infer that a person who wants food will ask for it, when at a restaurant—but not when that person is looking into a refrigerator at home. Scripts facilitate context-specific inferences by packaging the standard events that occur in particular contexts, such as restaurant dining.

Script theory was based on a theory of stereotyped knowledge structures, which were hypothesized to be built up by people through repeated experience. Scripts capture expectations, inferences, and knowledge that apply to common situations. For example, the restaurant script includes the following standard events, with standard roles of *restaurant*, *diner, food*, and *waiter*, which are filled in for each specific episode:

Diner enters the *restaurant*.
Diner sits at a table.
Diner orders *food* from *waiter*.
Waiter brings *food*.
Diner eats *food*.
Diner pays for *food*.
Diner leaves the *restaurant*.

Script-based expectations can aid disambiguation during story understanding. For example, when the restaurant script is active and "Mary asked for a hamburger" is processed, "Mary" refers to the *diner* and "hamburger" provides the role-filler for *food*. When "She paid for it" is encountered later, the script provides the expectation that "she" is Mary and "it" is the hamburger. In addition, scripts are useful for guiding summarization of text. Routine events, provided by the script, can be assumed; only the role-fillers provide new information and need to be reported in a summary.

Later research on Memory Organization Packages (MOPs) (18) developed hierarchical episodic memory models with shared structure, enabling cross-contextual learning by permitting a component to be refined in one context and reapplied in another. For example, something learned about payment in the context of a restaurant (e.g., that some restaurants refuse to accept credit cards to pay small amounts), would also be available in other MOPs which share the PAY scene, such as MOPs for grocery stores or service stations.

RULE-BASED REASONING

Rule-based systems use knowledge encoded in the form of rules to draw conclusions from chains of rule applications. Rule-based reasoning has been studied both as a cognitive model and as an AI method, and it was central to the explosive growth of industrial expert systems applications in the 1970s and 1980s.

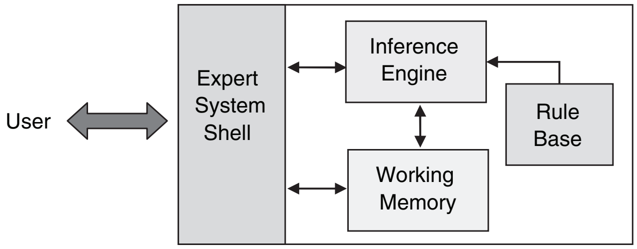


Figure 2. Production system architecture.

Production Systems

Production systems contain three components, an inference engine, a rule base, and a working memory, as illustrated in Fig. 2. Prior to problem solving, the working memory is initialized with a set of facts, which are updated during processing by deleting existing facts and adding new conclusions, or by adding information provided externally (e.g., by sensors or from querying the user).

Production systems apply their knowledge through the execution of production rules. Each production rule has two parts, a conditional part and an action part (see *Production Rules)*. At each processing step, the inference engine identifies rules whose conditional parts match the facts in working memory, to execute or "fire" the rules by performing their associated actions. This approach to flexible control differs from the prespecification of a solution path, and from the mixture of knowledge and control, commonly found in programs written in traditional programming languages.

An early focus of research in production systems was to model human problem-solving processes, as explored by Newell and Simon in the 1970s (19). Production systems continue to be studied as cognitive models. Soar, a cognitive architecture that represents knowledge in the form of productions (20), has been used in successive versions by a large body of researchers since the 1980s. As the goal of the Soar project is to support all capabilities required for a general intelligent agent, the Soar project also investigates capabilities such as interruptibility and the integration of learning and problem solving.

Rule-Based Expert Systems

A classic example of a rule-based expert system is MYCIN, a system for medical diagnosis (21). MYCIN is a goal-driven abduction system that aims to capture physicians' expert knowledge and to model reasoning with missing or incomplete information.

MYCIN's task domain is the diagnosis of infectious blood diseases including meningitis and bacteremia, which require rapid treatment. Although laboratory tests could be used to identify the organism causing blood disease, when MYCIN was developed, complete testing took 24–48 hours or more, too long for timely treatment. The time-critical nature of the task led doctors to acquire substantial expertise in the domain, making it especially interesting for studying diagnostic reasoning.

To capture expert knowledge, MYCIN uses production rules associated with certainty factors. Each rule represents a conclusion that an expert would draw given some evidence, and the certainty factor associated with each rule captures how strongly the evidence supports the rule's conclusion. For example, one MYCIN production rule states that "IF the infection is primary-bacteremia, and the site of the culture is one of the sterile sites, and the suspected portal of entry is the gastrointestinal tract, THEN there is suggestive evidence (0.7) that infection is bacterioid." The rules were acquired in interviews with doctors, conducted by knowledge engineers who elicited the rules and encoded them in a machine-readable form. MYCIN's initial rule base contained 450 production rules. In an evaluation of the system, MYCIN's performance on randomly selected case histories of meningitis was measured against that of members of Stanford Medical School, with MYCIN's performance comparable with—and in some cases better than—the humans' performance (22).

To enable MYCIN's inference engine to apply to other tasks, it was made into a stand-alone system, EMYCIN (for "empty MYCIN"). More generally, rule-based system *shells* became widely available, enabling developers of rule-based systems to focus only on domain knowledge. The success of such systems in many domains provided support for the knowledge principle.

Control Strategies for Rule Execution

Rule-based systems may guide rule execution either with a data-driven *forward chaining* strategy or with a goal-driven *backward chaining* strategy.

Forward Chaining. In a forward chaining system, the inference engine checks the conditional part of a rule against the facts stored in working memory to fire rules whose antecedents are matched. If multiple rules match, additional strategies determine which one to fire. For example, rules can be associated with salience values specifying their priorities compared with a default, and the most salient rule triggered. Other strategies include favoring rules matching recently generated facts (to help to focus on a single line of reasoning), favoring rules that have fired less recently (to help avoid loops), or favoring more specific rules (to exploit knowledge relevant to the specific situation), or using special-purpose reasoning based on metarules. Execution may stop when predefined conditions are met (e.g., when the system generates a desired conclusion), or may continue indefinitely, (e.g., if the production system is generating actions to control a robot's behavior).

Table 2 illustrates production rules and forward chaining with a sample rule base written in the syntax of JESS (Java Expert System Shell) (23). The sample rule base identifies mammals based on their characteristics of having body hair or producing milk. Given the initial facts shown in (B), the inference engine matches both the rules in the rule base in (A) but executes the rule *Hair→Mammal* before *Milk→Mammal* due to the higher salience value of the first rule. The second rule is consistent with the first rule and when both are executed, the modified working memory shown in (C) contains an additional fact about the animal being a mammal.

Table 2. Sample JESS Rule Base for Classifying Mammals, Initial Working Memory, and Working Memory After Rule Application

(A) Rule Base

```
(defrule Hair->Mammal
   (declare (salience 100))
   (attribute (type hasHair) (value “yes”)) ⇒
   (assert (animal (type mammal) (value “yes”))))


(defrule Milk->Mammal
   (declare (salience 50))
   (attribute (type producesMilk) (value “yes”)) ⇒
   (assert (animal (type mammal) (value “yes”))))


(defrule Ruminant+Mammal->Ungulate
   (attribute (type isRuminant) (value “yes”))
   (animal (type mammal) (value “yes”)) ⇒
   (assert (animal (type ungulate) (value “yes”))))
```

(B) Initial Working Memory

```
(attribute (type hasHair) (value “yes”))
   (attribute (type producesMilk) (value “yes”))
```

(C) Modified Working Memory

```
(attribute (type hasHair) (value “yes”))
   (attribute (type producesMilk) (value “yes”))
   (animal (type mammal) (value “yes”))
```

Rule-based system shells require efficient strategies to determine which rules to fire to handle large-scale rule sets. The Rete algorithm (24), used in a modified version by JESS, provides a method for determining which rules to fire without having to check every rule against working memory in each step. Rete builds a network of nodes in which each node, except for leaf and root nodes, corresponds to a pattern in the conditional statement of the rule. Paths from the root to a leaf node correspond to the conditional statement of a rule. Nodes store the facts that satisfy their pattern. As new facts are asserted or modified, the nodes in the network are annotated with the facts. As the algorithm annotates nodes with facts, it checks whether rules linked to the annotated nodes need to be fired due to changes in the conditional nodes. A rule can be fired if the nodes in its conditional part are satisfied. Figure 3 illustrates the network for the rule base of part *A* of Table 2, with the node values resulting from processing the facts of part *B*.

Backward Chaining. Forward chaining systems start from known information and generate possible conclusions. In backward chaining systems, such as MYCIN, chaining is focused by a hypothesized goal condition, which the system

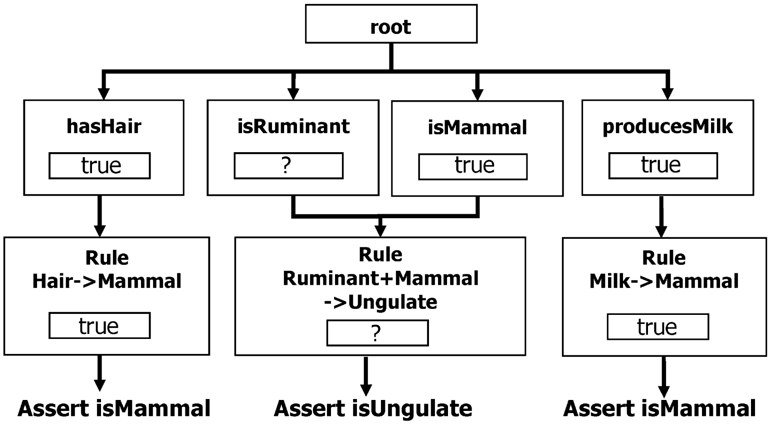


Figure 3. The Rete node network for three sample rules.

attempts to establish by pursuing a chain of rules chosen to confirm that hypothesis. For the MYCIN task, the goal is a possible diagnosis: a candidate infecting organism. As described previously, the system identifies rules and inquires about facts in support of the selected goal.

In backward chaining, the system starts from the goal, selects rules from which the goal could be concluded, and determines whether the facts in working memory are sufficient to trigger the selected rules. If not, it takes the antecedents of the selected rules as new goals to establish, forming a chain of rules and, if necessary, eventually asking the user for information that it cannot establish. Different strategies may control the order in which alternatives are pursued.

As an example of the chaining process, if the goal is to determine whether a particular animal is a mammal, the first rule in Table 2 suggests checking whether the animal has hair. In some contexts (such as medical systems, for which test results may be available), the system may also ask users questions to verify or reject the selected hypothesis. At any given time, the system may be considering multiple chains backward from a hypothesized goal, each of which containing a sequence of rules that, when triggered, leads to the selected goal. If the systems cannot construct any chain that fires, the hypothesized goal is rejected.

For example, in the rule base in Table 3, to determine whether an animal under consideration is a tiger, the system first attempts to establish whether the animal has black stripes, a tawny color, and is a mammal, and a carnivore. If this information is not available internally, it may ask the observer to provide the necessary information to trigger the Tiger rule. If any of the required information is missing (e.g., the user knows that the animal has the right color, is a mammal, and has black stripes, but not whether it is a carnivore), either of the rules to determine whether an animal is a carnivore may be tried to find the missing information.

When a rule-based system asks questions, how those questions are managed is important to user acceptance of the system. Consistent with the general practices of human physicians, MYCIN first asks general background questions about the patient before focusing on specific questions related to the hypothesized goal in the diagnosis of the disease and, as facts are needed, tries to gather related information at the same time to increase coherence of the interaction and improve user confidence.

Managing Uncertainty and Vagueness

In practice, it may be difficult to assign appropriate certainty factors. One approach to address this problem is to use machine learning to refine certainty factors. For example, the RAPTURE system (25) maps the rules in a rule base to a neural network architecture and then uses a modified version of the backpropagation learning algorithm (see *Artificial Neural Networks*) to revise the certainty factors associated with the rules. This system has been successfully applied to revise the MYCIN rule base. An alternative approach to handling uncertainty is to use methods based on probability, as described in the probabilistic graphical models section.

Table 3. JESS Rule Base for Classifying a Tiger

```
defrule Mammal+Carnivore+Tawny+Stripes->Tiger
  (animal (type mammal) (value “yes”))
  (attribute (type carnivore) (value “yes”))
  (attribute (type tawnyColor) (value “yes”))
  (attribute (type blackStripes) (value “yes”))⇒
  (assert (animal (type tiger) (value “yes”))))


(defrule Mammal+EatsMeat->Carnivore
(animal (type mammal) (value “yes”))
  (attribute (type eatsMeat) (value “yes”))⇒
  (assert (animal (type carnivore) (value yes”))))


(defrule Mammal+PointedTeeth+
Claws+ForwardEyes->Carnivore
  (animal (type mammal) (value “yes”))
  (attribute (type pointedTeeth) (value “yes”))
  (attribute (type claws) (value “yes”))
  (attribute (type fowardPointingEyes) (value “yes”))⇒
  (assert (attribute (type carnivore) (value “yes”)))
```

In some domains, such as control and pattern recognition, production systems commonly use fuzzy logic (26) to deal with imprecise measurements. In fuzzy rules, the conditional and action part are described with fuzzy sets, functions that specify a membership value between 0 and 1 for each element of the set. For example, a fuzzy rule whose antecedent checked whether a patient has a high temperature might use a fuzzy set to describe a range of temperature values, and their corresponding membership values would determine the extent to which a particular temperature value is considered "high" (see *Fuzzy Logic and Theory*). The consequent of the rule could generate a fuzzy set that is converted to a crisp value through a process called "defuzzification." Fuzzy rules enable vague knowledge to be expressed in a language that is natural to the experts.

Applications and Applications Issues

Rule-based expert systems have been applied to an extensive range of problem-solving tasks such as diagnosis, interpretation, planning, scheduling, and system configuration. Specific examples include expert systems for configuring computer systems and to perform audit risk analysis, to identify debit card fraud. Rule-based systems are also embedded within other systems to support decision making (e.g., to enforce software agent policies in open network environments) (see *Expert Systems*).

One of the social and economic motivations for expert systems is to make expertise, which is normally expensive and available in limited supply, more widely available. For example, medical expert systems could increase the accessibility of medical knowledge. However, larger issues can impede the transition from research system to technology. A case in point is MYCIN, which for ethical and legal reasons was never deployed, despite its successful evaluations. For knowledge-based systems, both the inference engine and the knowledge must be verified (see *Knowledge Verification*). In addition, as described later in this article, knowledge acquisition may be a difficult problem.

BLACKBOARD SYSTEMS

Blackboard systems coordinate independent processes for cooperative problem solving. Blackboard systems reflect the metaphor of experts working around a shared blackboard accessible to all, and each one able to consult, add, or remove the entries. In blackboard systems, the shared *blackboard* is hierarchically organized, representing information at different levels of abstraction. A set of independent programs, called *knowledge sources*, each reflect different capabilities or perspectives on the overall task, and individually monitor and update the blackboard during processing, incrementally taking advantage of results generated by the ongoing processing of other knowledge sources. For example, the HEARSAY-II system uses a blackboard system architecture to process continuous speech input, with knowledge sources performing functions such as extracting acoustic parameters, classifying acoustic segments, recognizing words, parsing phrases, and making predictions (27). Whenever the blackboard changes (e.g., with an addition), each knowledge source determines whether the change is relevant to its knowledge. Those knowledge sources that are relevant become eligible to be run by a scheduler, which controls knowledge source execution.

PROBABILISTIC GRAPHICAL MODELS

In a purely deductive reasoning framework, each rule describes a set of antecedents from which a conclusion must necessarily follow. However, deterministic rules may not be sufficient to characterize the connections in a domain. For example, rules for drawing medical conclusions must reflect the possibility of false positives. In complex real-world domains, rules cannot exhaustively include all potentially relevant factors, resulting on uncertainty in whether the conclusions of a rule will hold for any given instance. For effective reasoning, it is desirable to summarize the level of uncertainty and take it into account when drawing conclusions, which may be done by applying approaches such as probability theory (see *Probability and Statistics)* or alternative methods such as Dempster–Schafer theory, which distinguishes between belief and plausibility (28).

Probabilistic graphical models apply probability theory to knowledge-based systems, using a graph structure to represent information about dependence and independence. For example, Bayesian Networks are directed acyclic graphs in which nodes represent random variables and links represent the variables' dependence relationships. By implicitly encoding independence assumptions in the network structure, Bayesian Networks provide a concise representation.

For Bayesian Networks, knowledge capture involves developing the domain model encoded in the network, first qualitatively, reflecting relevance through the choice of connections, and then quantitatively, in terms of conditional probabilities. The inference problem becomes the problem of how to compute each node's belief, given current evidence; approximation techniques have been developed to enable rapid computations. See Charniak (29) for an overview and Pearl (30) for an extensive discussion. Dynamic Probabilistic Networks can be used to model dynamic systems, and stochastic simulation algorithms can be used to rapidly approximate the results of these networks. See *Bayesian Belief Networks* and *Hidden Markov Models* for related information.

MODEL-BASED REASONING

Device models provide another useful form of knowledge for diagnosis and troubleshooting. Model-based reasoning (MBR) exploits models of the structure and behavior of devices—characterizations of internal function, rather than simply descriptions of associations between observed antecedents and conclusions—to support a process of prediction and observation (31). The model-based troubleshooting process starts from observations of a device, such as measurements of inputs and outputs, a model of device structure, such as components and their connections, and descriptions of each component's behavior. To diagnose component failures, the approach generates hypotheses about the components causing the problem and tests hypotheses against device behavior to find suitable candidates. It then discriminates between the hypotheses consistent with the symptoms, by methods such as probing, selecting new points to measure within the device. Additional knowledge may be brought to bear in this process, for example, to select the probes expected to be most informative based both on their discriminating power and on known failure probabilities.

CASE-BASED REASONING

Case-based reasoning (CBR) focuses on reasoning from specific experiences, rather than from general rules or models. A case-based reasoner addresses new situations by retrieving prior cases and adapting their lessons to fit new circumstances. Case-based reasoning can be seen as combining a number of processes (32):

$$\text{Case-based reasoning} = \text{retrieval} + \text{analogy} + \text{adaptation} + \text{learning}$$

Early investigations of CBR were inspired by studies of human cognition, such as on the role and organization of episodic memory in understanding and the role of cases in human reasoning (see Ref. (33) for a survey of case-based reasoning viewed as a cognitive model). As humans solve problems, they may be reminded of similar problems in the past, suggesting starting points for new problems or warning of possible pitfalls to avoid. For example, a doctor responding to an adverse reaction to medication might be reminded of a specific similar emergency and how it was addressed, providing useful guidance that would not be contained in general rules (e.g., that adrenaline kits in a particular emergency room are kept on the top shelf). Motivations for applying CBR include that CBR can facilitate knowledge capture from experts, by enabling direct storage of their "war stories," rather than generation of rules, and systems may be fielded with a small set of "seed cases" to be augmented by the system's own experiences. Likewise, CBR systems can reuse reasoning effort, can adapt and apply prior solutions even when the reasons for their successes are poorly understood, and can justify their answers in terms of real examples rather than generalizations, which users may find harder to accept (32,34).

A fundamental difference between rule-based and case-based reasoning is that rule-based systems model problem solving as a process of *generate and test*, whereas case-based systems rely on *retrieve and adapt* (35). Reuse may be possible even when the underlying causal factors are unknown (e.g., when adapting an externally provided solution). For example, a novice cook may be able to adapt a vanilla cake recipe to chocolate, by adding cocoa power, without understanding why the basic recipe produces its results.

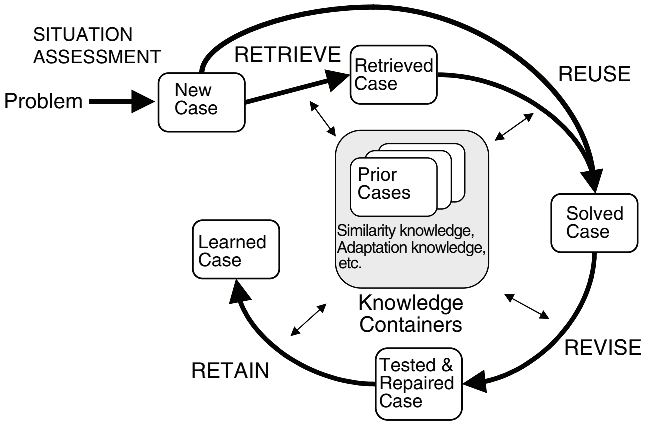


Figure 4. The case-based reasoning cycle (adapted with changes from Ref. 36).

The CBR Cycle and Knowledge Sources

A case-based reasoning system's processing can be seen as a cycle, beginning with retrieval of a case to address a new problem, and ending with a new case placed in memory for future use, as illustrated in Fig. 4. Given a problem description, situation assessment generates a problem description used as an index to retrieve a relevant prior case. The system attempts to reuse the solution of the prior case, revising it as needed for differences. The case is then retained in memory for future use, possibly after filtering for the expected benefit of retaining it.

As suggested by Fig. 4, CBR systems rely on multiple knowledge sources, the CBR "knowledge containers" (37), which include the cases themselves, knowledge implicit in the choices of representational vocabulary, similarity criteria, indexing knowledge to guide retrieval, and case adaptation information. These knowledge containers overlap, in the sense that, for example, an extensive case library may cover enough problems that the availability of cases compensates for limited adaptation knowledge; rich adaptation knowledge may enable successful performance with few cases. The ability to select where to place system knowledge facilitates the development of CBR systems by enabling system developers to provide knowledge in whichever container is most practical for a given task.

Methods and Issues in CBR

CBR systems often perform "nearest-neighbor" retrieval, selecting cases that minimize the distance between the current problem and the retrieved problem, based on a distance function considering the distance for each attribute. If problems are described by vectors of numeric or symbolic feature values, with the new problem situation $Q = q_1, q_2, \ldots, q_n$ and a previously solved problem $P = p_1, p_2, \ldots, p_n$, and $W = w_1, w_2, \ldots, w_n$ is a vector of non-negative weights reflecting feature importance, then the distance function is often defined by:

$$distance(Q,P) = (\Sigma_i w_i \times \text{difference}(q_i, p_i)^2)^{\frac{1}{2}}$$

for a given difference function. One simple approach is to define difference $(q_i, p_i) = |q_i - p_i|$ for numerical features, and 1 for identical symbolic feature values, 0 otherwise. However, more complex distance functions may be chosen to make difference values more directly comparable and to reflect other aspects of the task domain.

For classification or regression tasks, the risk of error due to noise may be mitigated by retrieving the top k cases and combining their solutions (e.g., by taking their majority vote to assign a categorical value, or by averaging their solution values for a numerical one).

For large case bases, cases may be organized into discrimination trees for retrieval efficiency. In case-based problem solving, cases may be indexed by the goals and constraints they satisfy, with indices varying from abstract, domain-independent features to highly concrete features. For example, the CHEF planning system, which plans in the domain of cooking, uses features ranging from "a plan step side-effect disabled a required condition for a concurrent step" to "the dish uses chicken" (38)). Retrieval based on concrete indices helps to retrieve cases with specific matches, facilitating reapplication; indexing based on abstract indices aids in cross-contextual retrievals, enabling cases to be applied to novel situations. Indexing vocabularies have been developed for a number of domains.

Case adaptation knowledge is often rule-based, but other knowledge-based methods, such as model-based reasoning have also been applied. As it may be difficult to generate needed knowledge in poorly understood domains—where CBR may be a method of choice—case adaptation remains a central challenge to CBR. Some research has addressed this fact by applying case-based methods to the adaptation process. Extensive index reformulation or case adaptation may be necessary to apply a prior lesson to a novel situation in a different domain, potentially resulting in a creative reasoning process (e.g., Refs. 39 and 40).

Case-based reasoning is widely used in help desk applications, which guide case selection through a conversational process with the user, focusing on retrieval support rather than case adaptation (41). The FormTool system, developed for plastics color matching, illustrates a high-impact application that includes case adaptation (42). As the number of long-term applications of CBR increased, how to control case-base growth while maintaining system competence was recognized as an important area (43), and interest grew in how to maintain CBR systems' knowledge (see Ref. 44 for an analysis and survey of CBR system maintenance). An extensive discussion of research on core CBR issues and methods is available in Ref. 45.

EXPLANATION IN KNOWLEDGE-BASED SYSTEMS

The explicit representation of knowledge makes reasoning processes amenable to explanation. This explanation process may be aimed for the system's own internal use or for the benefit of an external user.

Internal Explanation

Understanding programs that form connections by inference chaining can be seen as performing a basic form of explanation, generating causal connections to account for why events in a story are coherent. Script-based understanding systems also use their knowledge to explain, but by fitting new information into existing knowledge structures: an event (e.g., handing money to someone) is explained if it is expected by an active script (e.g., as the payment in the restaurant script).

Script-based models explain routine events, but cannot explain novel events. However, the attempt to apply scripts can still be useful for focusing the explanation effort, because the parts of a story that are useful to explain are the expectation failures or anomalies. For example, what to explain about a death would be quite different if the anomalous aspect were that it was premature (in which case the focus might be the cause) or if it were the advanced age of the deceased (in which case the focus might be explaining the secrets of the longevity of the deceased). The SWALE system (39) illustrates multiple roles of knowledge in internal explanation with a case-based approach drawing on knowledge sources including MOPs, explanation cases, an indexing vocabulary for anomalies, and a collection of explanation requirements called explanation purposes.

SWALE's explanation cases, which are adapted to explain new events, can be seen as providing flexible schemas to handle novel events. An alternative approach for generating new schemas is to perform explanation-based learning, in which a domain theory is used to explain the relevance of particular features, enabling correct generalizations from a single example (46).

Explanation for an External User

Expert systems research recognized early on the importance of explaining system behavior to users to increase their acceptance and confidence in system decisions (e.g., Ref. 21). A basic approach to the explanation process in a chaining system is to display the rule chain leading to a system decision. However, this detailed trace may be difficult for users to follow, and it may not include all information end users need. To address this problem, reconstructive explanation (47) treats the explanation process itself as a problem-solving task, reorganizing and augmenting explanations as needed.

One of the benefits of case-based reasoning is that the cases used to generate solutions can also provide compelling support to users, as shown by Ref. 48. However, the use of cases is not a panacea; additional reasoning may be needed to select the most effective cases to use for explanation, and additional explanation may be required to account for why the presented case was selected or how it was adapted (see Ref. 49 for a sampling of this work).

KNOWLEDGE ACQUISITION ISSUES

A classic problem for knowledge-based systems is how to secure the needed knowledge. Experts often have difficulties expressing their knowledge in a rule-based form, requiring the rule capture process to be mediated by knowledge engineers, who interview experts, represent, and refine the needed knowledge in a labor-intensive process, resulting in the “knowledge acquisition bottleneck” (50) (see *knowledge acquisition*).

Despite methodologies developed to facilitate knowledge capture, the process remains laborious. For example, a project that captured the knowledge in a chemistry textbook resulted in impressive system capabilities (at the level expected for college-level advanced placement examinations), but at an estimated knowledge acquisition cost of $10,000 per page (51). Research is under way on methods for supporting knowledge acquisition from humans, aiding them in identifying and resolving differences in their conceptualizations of a domain, and knowledge engineering, as well as on automatically extracting knowledge from sources such as the World Wide Web. User-centered knowledge acquisition methodologies may help to eliminate the need for knowledge engineers as mediators between the system and domain expert, enabling domain experts to enter their knowledge directly into a knowledge base. Machine learning methods may be useful for rule generation and refinement (see *Machine Learning)*.

Expert systems are normally designed with narrow and deep knowledge of a specific domain, which can cause brittleness, as systems are incapable of gracefully handling situations outside their narrow domain. The Cyc project (10) aims to address this and other problems by encoding an immense knowledge base of carefully crafted representations of common sense knowledge. Another endeavor, Open Mind Commonsense, takes a different tack, capturing informal knowledge entered by volunteers. Although this knowledge is less suited to machine reasoning, it has been used to develop advisory applications in domains for which it can provide a payoff and for which failure is noncritical (52).

THE SEMANTIC WEB

The World Wide Web now contains vast amounts of information on a large variety of topics. Although machines provide, display, and even produce the content of web pages dynamically, the information on the pages is designed primarily for human use. Despite promising work on automatically extracting such information (53), the problem remains challenging.

One cause of the difficulty of machine processing for web pages is that most web pages lack metadata tags describing their content. The Semantic Web (54) is a vision of a future World Wide Web that provides information to make Web content meaningful for machines as well as people. Semantic Web applications exploit web pages tagged with metadata defining the meaning of their content to enable knowledge-based methods to improve existing services such as Web search (e.g., by improving retrieval quality and enabling search systems to return answers, rather than pages). They also enable new services, such as software agents that use the metadata to find, compare, and respond to information from many sources, for tasks such as automating situation awareness aids or facilitating supply chain integration. A sampling of such applications is provided by Ref. 55.

The Semantic Web defines rules, concepts, and statements to annotate web pages with labels that define the semantics of the information on the pages allowing machines to make inferences about its content. The vocabulary used to define such labels is based on *ontologies*, which capture a “specification of a shared conceptualization of a domain” (56).

New languages, building on Web technologies such as XML (eXtensible Markup Language) and RDF (Resource Description Format) have emerged for the Semantic Web and have been adopted by the World Wide-Web Consortium (W3C) as standardized Semantic Web languages. A prominent example is the Web Ontology Language (OWL)(57), used for building ontologies that can be published on the Web and used to annotate Web content or make inferences about a subject. OWL builds on previous work on ontology languages including the DARPA Markup Language (DAML) and DAML+OIL (DAML Ontology Inference Layer). It covers and extends the language constructs and representational features offered by these languages to provide a language that can meet the requirements for representing knowledge in the Semantic Web. Both DAML and DAML-OIL also build on RDF and RDF Schema (RDFS).

RDF is an assertion language intended to express propositions about resources on the Web. In general, a resource is considered anything that can be assigned a Uniform Resource Identifier (URI). RDF expresses propositions as triples: a subject, a predicate, and an object. Each element can be a resource with a unique URI, whereas an object can also be a literal, which is a typed or untyped text string. Resources may be divided into groups called classes and members of a class are known as instances of the class. RDF Schema is a semantic extension of RDF that provides basic constructs for defining classes and properties and relationships between classes such as a class being a subclass of another class. DAML and DAML+OIL support constructs to define more complex relationships such as cardinality restrictions on properties.

OWL is a recent extension of existing ontology languages, providing a rich set of language features to enable efficient representation of ontologies. The language is specifically engineered for the Web, supporting features that make it easy to publish ontologies or to reuse existing ontologies on the Web, to annotate Web content, or to make inferences. OWL has three different sublanguages with a different level of expressiveness to serve the needs of the Semantic Web community. OWL Lite supports defining classification hierarchies and simple constraints. OWL DL (DL refers to description logic) is more expressive than OWL Lite, adding additional language constructs from OWL while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL Full is the complete OWL language providing maximum expressiveness without computational guarantees. Table 4 illustrates a fragment from an OWL ontology on food. The ontology includes a base class ConsumableThing, a class NonConsumableThing that is complementary to ConsumableThing, a class EdibleThing that is a ConsumableThing,

Table 4. Excerpt From an Ontology on Food, [Adapted from Ref. 58].

```
<owl:Class rdf:ID="ConsumableThing"/>


<owl:Class rdf:ID="NonConsumableThing">
<owl:complementOf rdf:resource="ConsumableThing"/>
</owl:Class>


<owl:Classrdf:ID="EdibleThing">
<rdfs:subClassOf rdf:resource="ConsumableThing"/>
</owl:Class>


<owl:Classrdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="ConsumableThing"/>
  <owl:disjointWith rdf:resource="EdibleThing"/>
  </owl:Class>
```

and a class PotableLiquid that is also a ConsumableThing but different from EdibleThing.

The promise of the Semantic Web and the creation of standard Semantic Web languages have prompted considerable interest in developing inference systems and tools to build and merge ontologies and to mark up web pages to provide meaning to the content. For example, Protégé (59) is an open-source framework, developed at Stanford University, for building editing tools that support the creation, visualization, and manipulation of ontologies in various representation formats; Protégé-OWL is a version for editing OWL ontologies. Other frameworks such as KAON (Karlsruhe ontology management infrastructure) (60) assist users in the creation, storage, and management of ontologies and support scalable and efficient reasoning with ontologies.

The Semantic Web of the future is likely to provide a set of standardized ontologies, enabling users to rapidly build their own ontologies by using existing and agreed on definitions of concepts. However, when new ontologies need to be constructed or existing ontologies modified, it will be crucial to aid users in finding the right parts of those ontologies. Consequently, research has focused on building tools to support retrieval and reuse of ontologies. For example, Swoogle (61) is a search engine for Semantic Web documents (online documents written in a Semantic Web language) that facilitates search within the documents based on keywords, focusing on matching class and property definitions in the Semantic Web documents. Hendler proposes that the Semantic Web is a step toward the large-scale knowledge source needed to realize the full potential of the knowledge principle (62).

HYBRID SYSTEMS

Each knowledge-based computation paradigm provides particular strengths. Consequently, many knowledge-based systems take a hybrid approach, combining multiple strategies. To provide a few illustrations combining case-based reasoning with other approaches, rule-based reasoning has been combined with CBR in domains such as legal reasoning, with cases and rules being jointly applied for legal arguments, and for generating pronunciations, with cases handling exceptions. Model-based reasoning has been combined with CBR for predicting forage consumption for rangeland management, with CBR generating an initial solution from a stored case, for refinement based on a model. These and other integrations are surveyed in Ref. 63. Hybrid methods may also be used internally, for one method to support the internal processing of another, for example, with constraint-based reasoning used to support case adaptation in CBR.

Hybrid approaches may also extend to combinations of knowledge-based and nonknowledge-based methods. For example, neural networks may be combined with knowledge-based approaches for each to make some classes of decisions, or symbolic knowledge may be inserted into neural networks to enable its refinement using neural network methods, for later extraction as refined symbolic knowledge (64). Such methods may simplify the development of intelligent systems and facilitate the application of knowledge-based systems technologies.

CONCLUSION

Knowledge-based computation spans many AI approaches. Each exploits explicit knowledge, but the forms of knowledge and mechanisms to manipulate them vary widely, for example, from formal to informal, from associations to models, from rules to cases, and from deductive to abductive inference. Knowledge-based computation has already been applied spanning a wide range of task areas, and the requirements of hard real-world problems have prompted calls to increase the emphasis on knowledge in areas such as AI planning, for which the use of knowledge can have considerable effect (65).

Despite these successes, a continuing challenge since the early days of knowledge-based computation has been how to obtain the needed knowledge. New knowledge capture methods, large-scale knowledge sources, and the advent of the Semantic Web promise to have significant impact on both research and applications in the next generation of knowledge-based computation systems.

ACKNOWLEDGMENT

We would like to thank George Luger for very helpful comments on a draft of this article.

BIBLIOGRAPHY

1. D. Leake, Artificial intelligence, in D. Considine, G. Considine, (eds.), *Van Nostrand's Scientific Encyclopedia*. New York Wiley, 2002 pp. 239–245.

2. A. Newell, and H. Simon, Computer science as empirical inquiry: Symbols and search. *Commun. ACM* **19**:113–126, 1976.

3. D. B. Lenat, and E. A. Feigenbaum On the thresholds of knowledge, in Proceedings of the Tenth International Joint

Conference on Artificial Intelligence, San Francisco, CA: Morgan Kaufmann, 1987.

4. R. Davis, H. Shrobe, and P. Szolovits, What is a knowledge representation?, *AI Magazine*, **14**(1): 17–33, 1993.
5. B. Bredeweg, and P. Struss, Current topics in qualitative reasoning, *AI Magazine*, **24**(4): 13–16, 2003.
6. J. McCarthy, Programs with common sense, in *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*. London: Her Majesty's Stationary Office, 1959, pp. 75–91.
7. P. Hayes, Naive physics 1: Ontology for liquids, in J. Hobbs, R. Moore, (eds.), *Formal Theories of the Commonsense world*. Norwood, NJ: Ablex, 1985, pp. 71–107.
8. R. Brachman, and H. Levesque, *Knowledge Representation and Reasoning*. San Francisco, CA: Morgan Kaufmann, 2004.
9. E. Davis, *Representations of Commonsense Knowledge*. San Mateo, CA: Morgan Kaufmann, 1990.
10. D. Lenat, and R. Guha, *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Reading, MA: Addison-Wesley, 1990.
11. M. Quillian, Semantic memory, in M. Minsky, (ed.), *Semantic Information Processing*. Cambridge, MA: MIT Press, 1968.
12. S. Shapiro, A net structure for semantic information storage, deduction, and retrieval, in *Proceedings of the Second International Joint Conference on Artificial Intelligence*, IJCAI, London, 1971, pp. 512–523.
13. E. Charniak, Passing markers: A theory of contextual influence in language comprehension, *Cognitive Sc.***7**: 171–190, 1985.
14. J. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA: Brooks Cole Publishing Co., 1999.
15. R. Schank, Conceptual dependency: A theory of natural language understanding, *Cog. Psych.* **3**(4): 552–631, 1972.
16. R. Schank, and C. Riesbeck, *Inside Computer Understanding: Five Programs with Miniatures*. Hillsdale NJ: Lawrence Erlbaum, 1981.
17. R. Schank, and R. Abelson, *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Lawrence Erlbaum, 1977.
18. R. Schank, *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge, England: Cambridge University Press, 1982.
19. A. Newell, and H. Simon, *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.
20. A. Newell, *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press, 1990.
21. B. Buchanan, and E. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley, 1984.
22. B. Buchanan, and E. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley, 1984.
23. E. Friedman-Hill, *Jess in Action, Java Rule-based Systems*. Greenwich, CT: Manning Publications, 2003.
24. C. L. Forgy, Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intell.***19**: 17–37, 1982.
25. J. J. Mahooney, and R. J. Mooney, Combining connectionist and symbolic learning to refine certainty-factor rule bases, in *Proceedings of the Eleventh International Conference of Machine Learning*, Los Altos, CA: Morgan Kaufmann, 1993, pp. 173–180.
26. L. A. Zadeh, The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy sets Sys.*, **11**(3): 199–227, 1983.
27. L. Erman, F. Hayes-Roth, V. Lesser, and D. Reddy, The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Comp. Surv. (CSUR)*, **12**(2): 213–251, 1980.
28. G. Shafer, The Dempster-Shafer theory, in S. C. Shapiro (ed.), *Encyclopedia of Artificial Intelligence*, 2nd ed. New York: Wiley, 1992, pp. 330–331.
29. E. Charniak, Bayesian networks without tears. *AI Mag.*, **12**(4): 50–63, 1991.
30. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San mateo, CA: Morgan Kaufmann, 1988.
31. R. Davis, Model-based reasoning: Troubleshooting, in H. Shrobe (ed.), *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*. Palo Alto, CA: Morgan Kaufmann, 1988.
32. D. Leake, CBR in context: The present and future, in D. Leake (ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. Menlo Park, CA: AAAI Press, 1996, pp. 3–30. Available: http://www.cs.indiana.edu/~leake/papers/a-96-01.html.
33. D. Leake, Cognition as case-based reasoning, in W. Bechtel, G. Graham (eds.), *A Companion to Cognitive Science*. Oxford: Blackwell, 1998, pp. 465–476.
34. J. Kolodner, *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann, 1993.
35. C. Riesbeck, What next? The future of CBR in postmodern AI, in D. Leake (ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. Menlo Park, CA: AAAI Press, 1996, pp. 371–388.
36. A. Aamodt, and E. Plaza, Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Comm.*, **7**(1): 39–52, 1994. Available: http://www.iiia.csic.es/People/enric/AICom.pdf.
37. M. Richter, Introduction, in M. Lenz, B. Bartsch-Sporl, H. D. Burkhard, S. Wess (eds.), *CBR Technology: From Foundations to Applications*. Berlin: Springer, 1998, 1–15.
38. K. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task*. San Diego, Academic Press, 1989.
39. R. Schank, and D. Leake, Creativity and learning in a case-based explainer. *Artif. Intell.* **40**(1–3): 353–385, 1989. Also in J. Carbonell, (ed.), *Machine Learning: Paradigms and Methods*, Cambridge, MA: MIT Press, 1990.
40. L. Wills, and J. Kolodner, Towards more creative case-based design systems. in D. Leake (ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. Menlo Park, CA: AAAI Press, 1996, pp. 81–92.
41. D. Aha, L. Breslow, H. Munoz-Avila, Conversational case-based reasoning. *Appl. Intell.***14**: 9–32, 2001.
42. W. Cheetham, Tenth anniversary of the plastics color formulation tool. *AI Magazine*, **26**(3): 51–62, 2005.
43. B. Smyth, and M. Keane, Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems, in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, San Mateo, CA: Morgan Kaufmann, 1995, 377–382.
44. D. Wilson, and D. Leake, Maintaining case-based reasoners: Dimensions and directions. *Computat. Intell.***17**(2): 196–213, 2001.

45. R. Mantaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. Maher, M. Cox, K. Forbus, M. Keane, A. Aamodt, and I. Watson, Retrieval, reuse, revise, and retention in CBR. *Knowledge Based Sys.*, 2006, In press.

46. G. DeJong, and R. Mooney, Explanation-based learning: An alternative view. *Mach. Learning*, **1**(1): 145–176, 1986.

47. M. R. Wick, W. B. Thompson, Reconstructive expert system explanation. *Artif. Intell.* **54**: 33–70, 1992.

48. P. Cunningham, D. Doyle, and J. Loughrey, An evaluation of the usefulness of case-based explanation, in *Case-Based Reasoning Research and Development: Proceedings of the Fifth International Conference on Case-Based Reasoning, ICCBR-03*, Berlin: Springer-Verlag, 2003, pp. 122–130.

49. D. Leake, D. McSherry, Explanation in Case-Based Reasoning. *Artificial Intelligence Review*, **24**(2): 2005.

50. F. Hayes-Roth, D. Waterman, and D. E. Lenat, *Building Expert Systems*. Reading, MA: Addison-Wesley, 1983.

51. N. Friedland, P. Allen, G. Matthews, M. Witbrock, D. Baxter, J. Curtis, B. Shepard, P. Mi-raglia, J. Angele, S. Staab, E. Moench, H. Oppermann, D. Wenke, D. Israel, V. Chaudhri, B. Porter, K. Barker, J. Fan, S. Chaw, P. Yeh, D. Tecuci, and P. Clark, Project Halo: Towards a digital Aristotle. *AI Magazine*, **25**(4): 29–48, 2004.

52. H. Lieberman, H. Liu, P. Singh, and B. Barry, Beating common sense into interactive applications, *AI Magazine*, **25**(4): 63–76, 2004.

53. E. Etzioni, *Proceedings of the aaai 2007 spring symposium on machine reading*. Technical report, AAAI, 2007.

54. T. Berners-Lee, J. Hendler, and O. Lassila, The semantic web, *Scienti. Amer.*, **284**(5): 34–43, 2001.

55. Y. Gil, E. Motta, V. Benjamins, M. Musen, (eds.), *The Semantic Web - ISWC 2005*. Berlin: Springer Verlag, 2005.

56. T. R. Gruber, A translation approach to portable ontologies, *Knowledge Acquis.*, **5**(2): 199–220, 1993.

57. D. McGuinness, and F. Harmelen, Owl web ontology language overview, W3C recommendation. World Wide Web Consortium, 2004.

58. M. K. Smith, C. Welty, D. McGuinness, Owl web ontology language guide, W3C recommendation. World Wide Web Consortium, 2004.

59. N. Noy, M. Sintek, S. Decker, M. Crubzy, R. Fergerson, and M. Musen, Creating semantic web contents with Protégé-2000. *IEEE Intell. Sys.*, **48**(2): 60–71, 2001.

60. E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, N. Stojanovic, R. Studer, G. Stumme, Y. Sure, J. Tane, R. Volz, and V. Zacharias, Kaon - towards a large scale semantic web. in K. Bauknecht, A. M. Tjoa, G. Quirchmayr, (eds.), E-Commerce and Web Technologies, *Third International Conference, EC-Web 2002*, Aix-en-Provence, France, 2002, pp. 304–313.

61. L. Ding, R. Pan, T. Finin, A. Joshi, Y. Peng, and P. Kolari, Finding and ranking knowledge on the semantic web, in *Proceedings of the 4th International Semantic Web Conference*, Springer Verlag, 2005, pp. 156–170.

62. J. Hendler, Knowledge is power: A view from the semantic web, *AI Magazine* **26**(4): 76–84, 2005.

63. C. Marling, M. Sqalli, E. Rissland, H. Munoz-Avila, and D. Aha, Case-based reasoning integrations, *AI Magazine*, **23**(1): 69–86, 2002.

64. J. Shavlik, A framework for combining symbolic and neural learning, *Machine Learning*, **14**(3): 321–331, 1994.

65. D. Wilkins, and M. desJardins, A call for knowledge-based planning, *AI Magazine*, **22**(1): 99–115, 2001.

David Leake
Thomas Reichherzer
Indiana University
Bloomington, Indiana

K

KNOWLEDGE MANAGEMENT APPLICATION

Information and communication technologies (ICT) that support the handling of knowledge in organizations have been discussed for a long time. From the 1950s to the 1980s, systems that apply artificial intelligence (AI) technologies had a powerful impact on the conceptualization of knowledge, not only in the discipline computer science, but also in fields such as management science, organization science, or psychology. However, many business organizations that attempt to implement these technologies were frustrated by their comparably high complexity and the difficulties of applying them to business challenges. Thus, AI technologies survived only in specific application fields. In the 1990s, after a period of high attention to the increase of efficiency, organizations were faced with the transformation of the economy into a knowledge economy and its challenges to increase significantly the speed of innovation and to improve the way organizations handle (distributed) knowledge. For those countries that do not have (anymore) the possibility to exploit some form of natural resources, it is knowledge that creates wealth. Organizations strive to increase productivity of *knowledge work*. Knowledge work is creative work, it solves ill-structured problems in complex domains with high variety and many exceptions, and thus it requires a high level of skills and expertise from employees. The challenge is to design, implement, and maintain an organizational and ICT environment conducive for this type of work. The importance of knowledge work can be underlined by a recent study that showed that most jobs created in the United States between 1990 and 2000 in fact are jobs in the knowledge work sector (1).

Concepts of knowledge management (KM) have been suggested to meet this challenge, starting with the highly innovative work by authors such as in Refs. (2–5), just to name a few. Many authors from a variety of disciplines created, applied, and reflected several approaches, concepts, methods, tools, and strategies for KM. These innovations have led to several terms that are used differently, approaches that are incommensurable, and a lack of applicability in a business context. More recently, however, several instruments have emerged as state-of-the-art of KM practice. Examples are competence management, community management, or semantic content management.

Backed by tremendous interest in KM in the academic field and in business practice, vendors of information and communication systems as well as researchers in the field of computer science and management information systems showed prototypes, tools, and systems to support KM called *knowledge management systems* (KMSs). The term KMS has been a strong metaphor for the development of a new breed of ICT systems. In this view, KMSs combine, integrate, and extend several heterogeneous ICT. Examples are AI technologies, communication systems, content and document management systems, group support systems, Intranet technologies, learning environments, search engines, visualization technologies, and workflow management systems. Given the complexity of these technologies, it seems obvious that the development of KMS is a complex undertaking. Recently, many vendors have insisted that their products have "knowledge management technology inside." More recently, however, it seems that many technologies provided by the avant-garde systems have been woven into the enterprise infrastructure implemented in many organizations. Whereas enterprise resource planning systems target the informational representation of business transactions, enterprise knowledge infrastructures create an ICT environment for knowledge work throughout the organization.

The aim of this article is to give an overview of the manifold recent developments in the field of KM in general and with respect to KMS in particular. To achieve this goal, first KM approaches are analyzed systematically as the conceptual basis of ICT applications built to foster the implementation of KM initiatives in businesses and in organizations. The next sections review the ICT roots of KMS, define the term, and obtain a set of characteristics that differentiates KMS from its roots. The article then outlines an ideal architecture before it discusses classes of KMS and summarizes some empirical findings on the state-of-practice. The last section gives an outlook on future trends and concludes the article.

KNOWLEDGE MANAGEMENT

The field of KM has drawn insights, ideas, theories, metaphors, and approaches from diverse disciplines. The roots of the term can be traced back to the late 1960s and early 1970s in the Anglo-American literature. However, it took almost another 20 years until the term appeared again in the mid-1980s in the context as it is still used today (e.g., Refs. 4 and 5). The underlying concepts have been around for some time. Many fields and disciplines exist that deal with the handling of knowledge, intelligence, innovation, change, learning, or memory in organizations. Various approaches have played a role in the development of the theories of organizational learning, organizational memory, and, ultimately, of KM. These theories can be divided into several categories: a psychologic and sociologic line of development (e.g., organizational psychology and sociology), the sociology of knowledge with concepts such as social networks and foundations for approaches in organizational learning, a business line of development (e.g., human resource management, organization science, strategic management) with the knowledge-based view of business strategy and intellectual asset management, and an ICT line of development (e.g., systems theory AI, or management information systems).

In addition to this interdisciplinary perspective on KM, another popular conceptualization compares it with data management and information (resource) management. The

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

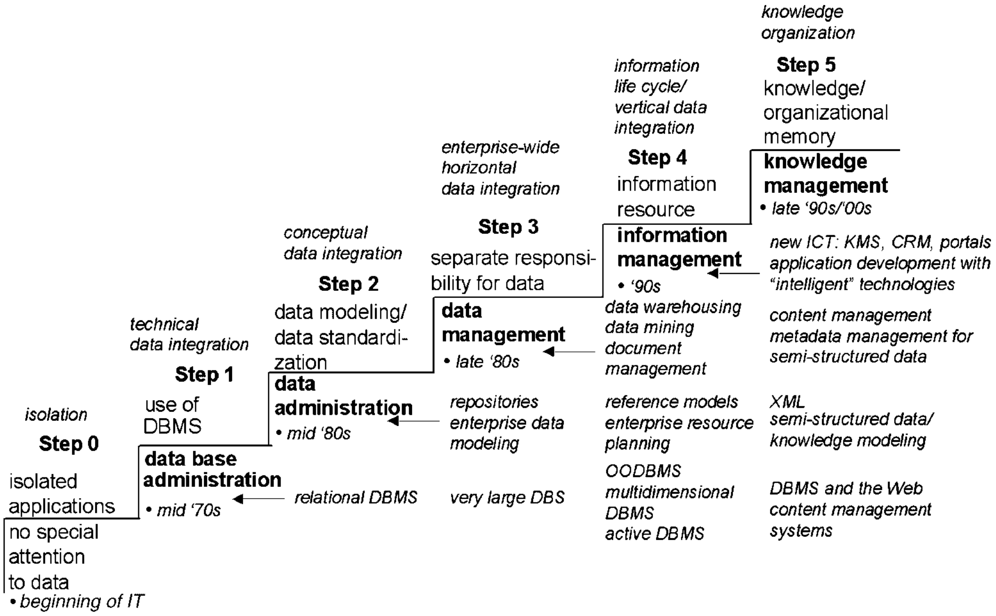

isolation
Step 0
isolated applications
no special attention to data
• beginning of IT
technical data integration
Step 1
use of DBMS
data base administration
• mid '70s
relational DBMS
conceptual data integration
Step 2
data modeling/ data standardization
data administration
• mid '80s
repositories enterprise data modeling
very large DBS
enterprise-wide horizontal data integration
Step 3
separate responsibility for data
data management
• late '80s
data warehousing data mining document management
reference models enterprise resource planning
OODBMS multidimensional DBMS active DBMS
information life cycle/ vertical data integration
Step 4
information resource
information management
• '90s
new ICT: KMS, CRM, portals application development with "intelligent" technologies
content management metadata management for semi-structured data
XML semi-structured data/ knowledge modeling
DBMS and the Web content management systems
knowledge organization
Step 5
knowledge/ organizational memory
knowledge management
• late '90s/'00s

Figure 1. Historical development of information processing with focus on data (Based on Ref. 6.)

perspective on KM in these approaches can be characterized as primarily technology oriented. Many authors who went to the trouble of making a clear distinction between data, information, and knowledge within the IS discipline seem to agree on some form of hierarchical relationship. Each higher level is based on or extends the preceding level. This conceptualization is used to postulate different demands for management (i.e. goals, approach, organizational roles, methods, instruments) and different resulting systems (e.g., database systems, data warehouses, information and communication systems, and knowledge management systems) on each of these levels. After a period with no special attention to data, in the 1970s and the beginning of the 1980s, the focus was on data management (see Fig. 1). First, the main goal was to technically integrate previously isolated data storage units. Step two consists of semantic or conceptual data integration, data modeling, and data handling. Step three creates a separate organizational responsibility for data management composed of both technical and conceptual tasks. On step four, information was understood as a production factor that had to be managed like other production factors (e.g., capital, labor). Thus, the scope of information management was much broader compared with data management. The most important aspects were (1) the extension from the management of syntactic and semantic to pragmatic aspects of information, (2) the understanding of information as an instrument for preparing decisions and actions, (3) information logistics, (4) the contingency approach to information, i.e., the different interpretation of data in different situations, and (5) the perspective-based approach to information, i.e., different groups of users might interpret the same data differently.

Whereas organizations have realized substantial benefits from data and information management, knowledge has proven to be difficult to manage. Knowledge work and knowledge-intensive business processes have been difficult to reengineer (7). An organization's ability to learn or to handle knowledge assets, processes, or services have been considered new key success factors. It has required new organizational design alternatives, implemented with the help of KM instruments, and also new information and communication systems to support the smooth flow of knowledge, which consequently has been called KMS. Already existing tasks on lower steps have been once again extended. With the advent of advanced database and network technologies, as well as with the availability of sophisticated AI technologies for purposes such as text mining, user profiling, behavior analysis, pattern analysis, and semantic text analysis, KM extended the focus of information management to handle new information and communication technologies as well as to enrich application development with intelligent technologies.

Knowledge management is defined as (1) the management function responsible for regular (2) selection, implementation and evaluation of knowledge strategies (3) that aim at creating an environment to support work with knowledge (4) internal and external to the organization (5) to improve organizational performance. The implementation of knowledge strategies comproes all (6) knowledge management instruments (7) suitable to improve the organization-wide level of competencies, education, and ability to learn.

In item (1), the term "management" is used here in a functional sense (managerial functions approach) to describe the processes and functions (such as planning, organization, leadership, and control) in organizations as opposed to the institutional sense (managerial roles approach) that describes the persons or groups that are responsible for management tasks and roles. In item (2), the systematic interventions into an organization's knowledge base have to be tied to business strategy. Knowledge strategies guide the implementation of a KM initiative and tie it to business strategy. According to traditional strategic management, a strategic gap is the difference between what an organization should do to compete and what it is doing currently. Strategies try to close this gap by aligning what an organization can do considering its strengths and weaknesses with what it must do to act on opportunities and threats. A knowledge strategy addresses knowledge gaps—differences between what an organization must know to execute its strategy and what it actually knows (8). In item (3), KM creates an organizational and technological infrastructure to improve knowledge work. In item (4), knowledge processes are not restricted to the organization's boundaries, but involve cooperation with partners, suppliers, and customers. Examples for knowledge processes are submission of knowledge elements to a knowledge base, discovery of applicable knowledge, acquisition of knowledge external to an organization, or moderating a community of practice, interest, or purpose. In item (5), KM aims primarily to improve organizational effectiveness. However, creating, maintaining, or distributing intellectual capital results in a higher valuation of an organization.

In item (6), depending on the perspective on KM, objects of the implementation of knowledge strategies can be objectified knowledge resources, (documented knowledge, people, organizational or social structures, knowledge-related information, and communication technologies). These resources are targeted by KM instruments such as collections of organizational, human resources, and ICT measures that are aligned, clearly defined, and can be deployed purposefully to achieve knowledge-related goals. In addition, ICT measures are independent of a particular knowledge domain (examples include expert advice, personal knowledge routines, idea and proposal management, competence management, technology-enhanced learning, good/best practice management, case debriefings, lessons learned, or semantic content management). In item (7), KM is not exclusively about individual learning. Collective learning is of differing types [single loop, double loop, deutero learning (9)], occurs on various levels of the organization (e.g., work group, project, community or network, organization, network of organizations, business ecosystem), and occurs in various phases (e.g., identification or creation, diffusion, integration, application, or feedback). KM aims to improve the organizational competence base as well as the ability to learn. No areas focus explicitly on the contents [i.e., the actual subjects, topics or knowledge area(s)] around which a KM initiative builds a supportive environment. The reason for this is that the definition of KM should support all kinds of knowledge areas.

Two groups of KM approaches exist: human-and technology-oriented. Basically, these approaches reflect their origin, either in a human/process-oriented organizational learning, organization science background, or in a technological/structural MIS or computer science/AI background. They even have found their way into KM strategy, where a technical codification strategy is distinguished from a human-oriented personalization strategy (10). It is agreed that more holistic KM conceptualizations exist that encompass both directions. These approaches can be distinguished with respect to the main focus area of KM they concentrate on (12). KM measures and tools are bundled as KM instruments to provide specific KM services for one of four KM focus areas identified by Wiig (11), as follows: people, intellectual capital, enterprise effectiveness, and information technology/management. The approaches also differ in their conceptualizations of knowledge. Table 1 shows perspective, focus area, and definitions of knowledge as well as characterization of strategy, organizational design, KM instruments and systems that together make up a KM initiative bound tightly to one central KM approach.

A technology-oriented codification strategy focuses on externalized knowledge that is separable from people and can be documented, retained, and reused in other application areas or organizational units. Corresponding KM instruments focus heavily on information and communication technologies with an economic model that develops reusable knowledge assets and standardization of procedures even in knowledge-intensive areas. A human-oriented personalization strategy focuses on knowledge that is inseparable from people and aims to create an environment in which people can work together efficiently. Corresponding KM instruments develop networks of people so that tacit knowledge can be shared. As the focus is on people, a pivotal goal is to reduce time-to-proficiency when a knowledge worker takes on a new role. A process-oriented, on-demand strategy bridges the gap between these two roles by designing services systematically for knowledge-intensive business processes and for knowledge processes. Complex knowledge management services are composed of basic services offered by heterogeneous systems such as document, content, workflow management, communication, collaboration, and personal information management systems. Viewing KM instruments from a service perspective bound to processes eases the integration into a general business framework.

An implementation of ICT to support a strategically relevant KM initiative must not only select a KM approach, strategy, organizational design, and combination of KM tools and systems, but also must integrate KM instruments with the supporting technology. KM instruments and systems are discussed in the following sections.

ROOTS OF KNOWLEDGE MANAGEMENT SYSTEMS

A review of the literature on ICT to support KM reveals several common terms, such as knowledge warehouse, KM software, suite, (support) system, technology, as well as learning management platform, portal, suite, system, or organizational memory (information) system (12,13). In addition to these terms that suggest a comprehensive platform in support of KM, many authors provide more or less extensive lists of individual tools or technologies that can be used to support KM initiatives as a whole or for certain processes, life cycle phases, or tasks thereof (14–17). The latter can be described as roots of KMS that are combined and integrated to build KMS.

Figure 2 uses the metaphor of a magnetic field produced by a coil to show the technological roots and the influences that impact the design and the implementation of KMS.

Table 1. Comparison of approaches to knowledge management

Dimensions	Technology-oriented	Human-oriented	Process-oriented
(1) Approach			
Perspective	engineering, congnitive	cultivation, community	business, customer-orientation, socio-technical
Focus area	IT: maximize capture, transformation, storage, retrieval and development of knowledge	people: maximize effectiveness of people-centric learning organization	intellectual asset & enterprise effectiveness: maximize building and value reallocation of knowledge assets, maximize operational effectiveness
Knowledge	documented, separable from people	exclusively in the heads of people	asset, skill, competence, embedded in social networks and (knowledge) processes
(2) Strategy			
Knowledge Strategy	codification; reuse documented knowledge	personalization; foster handling of knowledge of persons/in groups	on-demand; situation-oriented design of knowledge processes for business processes
Goals	improve documentation and retention of knowledge, acquire external knowledge, turn implicit into explicit knowledge	improve communication, train newly recruited, improve knowledge sharing, improve personnel development	improve visibility of knowledge, improve access to and use of tacit and explicit knowledge, improve innovation, change culture
(3) Organization			
Roles	author, knowledge (base) administrator, knowledge broker	expert, mentor, network chair, community manager, moderator	knowledge partner and stakeholder, boundary spanner, coordinator for KM, subject matter specialist, owner and manager of knowledge processes
Tasks	storing, semantic release and distribution, refinement, deletion/archiving of knowledge,acquisition of external knowledge	establish, foster, and moderate communities; document competences and expertise; organize knowledge sharing events	identify knowledge stances; design knowledge maps, profiles, portals and processes; personalize organizational knowledge base; implement learning paths
Culture	technocratic	socio-cultural	socio-technical, management
(4) KM instruments and systems			
Instruments	semantic document and content management, instruments for discovery, publication, collaboration, learning and adaptation	competence, idea, and proposal management; personal knowledge routines; expert advice; communities; knowledge networks, self-managed ad-hoc learning	management of patents and licenses, KM scorecards, case debriefings, lessons learned, good/best practices, knowledge process reengineering, technology-enhanced learning
Contents	knowledge about organization, processes, products; internal studies, patents, online journals	employee yellow pages, skills directories, ideas, proposals, knowledge about business partners	cases, lessons learned, good/best practices, learning objects, profiles, valuations, comments, feedback to knowledge elements
Architecture	integrative KMS	interactive KMS	KMS bridging the gap
Type	infrastructure for documented knowledge	infrastructure for communication, management of competences	process-oriented system offering composed services for knowledge and business processes
Functions	publication, classification, formalizing, organization, search, presentation, visualization of knowledge	asynchronous and synchronous communication, collaboration and cooperation, community support	profiling, personalization, contextualization, recommendation, technology-enhanced learning, navigation from knowledge elements to people and processes
Tools/systems	semantic document and contentmanagement system, Wiki, knowledge portal	skill management system, computer-mediated communication, social software	process warehouse, integrated case-based reasoning, lessons learned, learning object and good/best practice repository

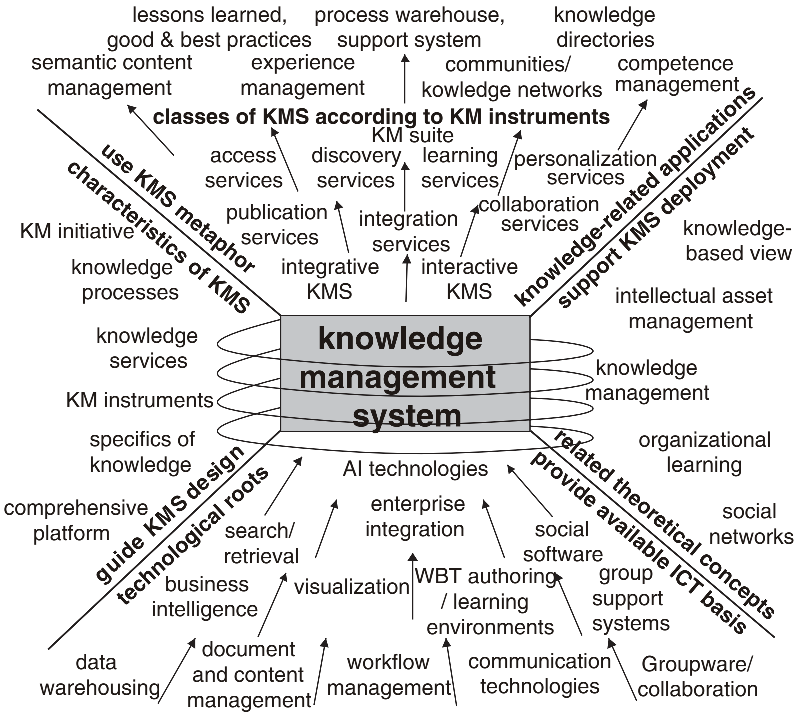


Figure 2. Technologic roots and influences of knowledge management systems.

The term KMS plays the role of the coil, the magnetic center. Theoretical approaches that support the deployment of KMS are shown to the right of the magnetic center (see section on "Knowledge management"). The main characteristics of KMS stress the differences to their ICT predecessors (see section "Toward a definition of KMS") and are shown on the left side. Together, both influences provide the energy to integrate, (re-)interpret, (re-)arrange, and (re-) combine ICT technologies that are the roots of KMS into a set of KMS-specific services (see section on "Architecture") that in turn are integrated into application systems, tools, and platforms with a clear focus on the support of KM concepts and instruments. Finally, KM instruments represent the classes of KMS in a narrow sense (see section on "Classification").

In the following sections, the most important ICT will be reviewed that forms the technologic roots of KMS. Comprehensive KMS combine and integrate the functionality of several of these predecessors.

Data Warehousing

A data warehouse is a subject-oriented, integrated, nonvolatile, time-variant collection of data in support of management decision processes (18). It is assumed implicitly that a data warehouse is separated physically from operational systems. External databases are the sources from where data are loaded regularly into the data warehouse. Data are organized by how users refer to them. Inconsistencies are removed and data are cleaned to remove errors and misinterpretations, are converted (e.g., concerning measures, currencies, and sometimes summarized and denormalized) before they are integrated into the data warehouse. Data in the data warehouse usually are optimized for analysis with business intelligence tools (e.g., star and snowflake data model, multidimensional databases).

Document and Content Management

The term *document management* denotes the automated control of electronic documents, both individual and compound documents, throughout their entire lifecycle within an organization (i.e., creation, storage, organization, transmission, retrieval, manipulation, update, and eventual disposition of documents). Document management systems provide functions to support all tasks related to the management of electronic documents, such as to capture, structure, distribute, retrieve, output, access, edit, and archive documents over their entire lifecycle. Web content management systems are applied to handle efficiently all electronic resources required to design, run, and maintain a website and to support all tasks related to authoring, acquiring, reviewing, transforming, storing, publishing, and delivering contents in Web formats. They are used to manage the entire web publishing process; to offer mechanisms for releasing new contents; to support HTML generation with the help of templates, standard input, and output screens; and to separate content and layout that provides a standardized look and feel of the web pages. As a consequence, participants who are not familiar with HTML can publish web content that fits into an organization's corporate (web) identity.

Workflow Management

A workflow is the operative, technologic counterpart of a business process and consists of activities related to one another that are triggered by external events and are carried out by persons using resources such as documents, application software, and data. A workflow management system “defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications” (19). Most workflow management systems primarily support well-structured organizational processes. More recently, some systems also support flexible workflows and so-called ad-hoc workflows. An ad-hoc workflow is a sequence of tasks that cannot be standardized, but they must be designed spontaneously by participants. Workflow functionality can be used in knowledge management to support processes such as the publication or the distribution of knowledge elements. Several KMS contain flexible functions for workflow management, such as Open Text Livelink.

Communication Technologies

Communication systems are electronic systems that support asynchronous and synchronous communication between individuals, such as point-to-point communication systems, collectives, and multipoint communication systems. Examples of synchronous communication systems include teleconferencing systems such as text conferencing (chat), instant messaging, audio, and video conferencing systems. Examples of asynchronous communication systems include email, listserver, and newsgroups.

Groupware/Collaboration

Groupware is a category of software to support workgroups and teams. Usually, groupware is classified according to a matrix of group interaction with the two dimensions time and place: same time versus different time as well as same place versus different place. Groupware tools can be classified even more into (1) communication systems such as email, audio/video systems, and chat systems; (2) information sharing systems such as message boards, tele-consultation systems, co-browser; (3) cooperation systems such as co-authoring, shared CAD, whiteboard, word processor, spreadsheet, and group decision support systems; (4) coordination systems such as group calendar, shared planning, notification systems; and (5) social encounter systems such as media spaces and virtual reality. A Groupware platform provides general support for collecting, organizing, and sharing information within (distributed) collectives of people, such as work groups and project teams over corporate networks as well as the Internet. Examples for Groupware platforms are Lotus Notes, Microsoft Exchange, and BSCW (which is available freely over the Internet). Groove, which was developed by Groove Networks (now Microsoft) is a recent example for a Groupware platform that uses the peer-to-peer metaphor instead of the client-server paradigm.

Business Intelligence

Business intelligence denotes the analytic process that transforms fragmented, organizational, and competitive data into goal-oriented “knowledge” about competencies, positions, actions, and goals of internal and external actors and processes. The analytic process requires an integrated data basis usually provided by a data warehouse. Examples of technologies that support this process are decision support system technologies; multidimensional analysis; online analytical processing; data mining, text mining, and web mining technologies; balanced scorecard; business simulation techniques; and also artificial intelligence technologies, such as case-based reasoning and issue management.

Visualization

Visualization is used in a multitude of tools and systems. Most visualization systems are based on graph theory. In addition to two-dimensional graphs that represent elements and relationships, several tools also provide three-dimensional visualization techniques. Examples are tools for data, function, organization, process or object-oriented modeling, or tools that provide mapping techniques that have a long tradition in psychology, sociology, and pedagogy, such as mind mapping.

Web-Based Training (WBT) Authoring and Learning Environments

Learning environments are application systems that offer specified learning content to the learner in an interactive way, and thus they support the teaching and/or learning process. Computer-based training has its historical roots in programmed instruction or learning in the late 1950s, which was based on the concept of operant conditioning developed by Skinner. Psychologic and pedagogic, as well as technologic advancements have led to a wide variety of systems and learning environments that reflect the diversity of learning. Examples are drill and practice systems, (intelligent) tutoring systems, active assistance systems, micro-worlds, simulation systems, experimental game systems, hypertext-/hypermedia learning systems, as well as WBT, multimedia learning environments, teleteaching, distance learning, tele-tutoring, and computer-supported collaborative learning. Recently, these diverse concepts have found their way into integrated learning (content) management systems that overlap with KMS.

Group Support Systems (GSSs)

GSSs, also called group decision support systems, are interactive systems that combine communication, computer, and decision technologies to support the formulation and the solution of unstructured problems in group meetings. GSSs integrate technologies to support communication in groups, structure of processes by which groups interact (e.g., agenda setting, facilitation), and information processing (e.g., aggregating, evaluating, or structuring information). They can be classified according to the level of support in (1) removing communication barriers, (2) decision modeling and group decision techniques, and (3) expert advice to select and to arrange rules in a meeting that lead to machine-induced group communication patterns (20).

Search

A search engine is a program that can be used to find resources (e.g., documents or images) either in an organization's Intranet or in the WWW. Search engines apply programs that trace permanently the Web or an Intranet for new web pages, so-called spiders or robots. A newfound web page is scanned for keywords that are stored together with the URL of the web page in the search engine's database. At the time when a user submits a search term to the search engine, only this database is searched and intelligent algorithms are applied to retrieve those web pages that fit most to what the user has searched for. So-called meta- or multi-search engines forward search strings including boolean operators to various search services, collect and filter the results for redundancies, and present them accordingly. Both search engines and meta-search engines can be distinguished even more with respect to the search domain that they support, such as organization-internal and/or organization-external systems.

Enterprise Integration

This bundle of technologies aims to provide the basis for interactions between a variety of data and document sources and between application components and systems. Integration in information processing can be classified according to the object into data, function, and program integration. Enterprise integration is an integration infrastructure, sometimes called middleware, that covers these classes and provides a basis for fully automated, organization-wide integration, or even integration between organizations. Typical standard technologies for data integration are based on XML, XML Schema, and XSLT that offer a metalanguage for annotation, description of the structure, and transformation of semi-structured data. Because of the importance of users as participants accessing KMS, integration of user data sometimes is discussed separately, called identity management. The semantic web stack, particularly RDF, RDF Schema, and OWL, provide the basis for semantic integration as required in KMS. With respect to function integration, the web service stack offers a standard way for describing and discovering public interfaces of software systems. With respect to process integration, many initiatives exist for standardizing XML-based languages to describe workflows that in turn invoke Web services, such as the Business Process Execution Language (21).

Social Software

Social software is a recent concept, a subset of computer-mediated communication that covers software and is used to create and to maintain social networks or virtual communities. Typically, this category of software allows easy-to-use mechanisms to create and to maintain online profiles (social identity), build relationships and reputation (social capital), stay aware of a network's activities (social presence), comment on and recommend to others (social feedback), interact with others (social interaction), organize physical meetings (social planning), and share content (social spaces) on the Internet. Social software focuses on supporting individuals who enter networks or communities voluntarily and therefore supports informal gatherings rather than formal organizational groupings in teams or workgroups, which typically are focused by Groupware, project management, and collaboration software. Because of this informal, self-directed nature of joining networks, it could be described as employing a peer-to-peer, bottom-up metaphor rather than a server-based, top-down metaphor (22). It has the potential of building larger and more effective networks. Examples for software that can be used with this goal in mind are easy-to-use content management systems such as text, audio and video Blogs, Wikis, fora, real-time communication (e.g., instant messaging or chat), and software platforms for rich interactions between its members that build on the friend-of-a-friend metaphor, such as the FOAF project, MSN Groups, Tribe.Net, Meetup.com or, with a business connotation, LinkedIn or Xing. Currently, many organizations adopt these technologies and attempt to profit from them. Social software seems to be particularly promising to fill in the gap of the less supported personalization and collaboration portion of organizational KMS. However, it remains to be seen whether and how the additional challenges in business or organizational settings, particularly with respect to power distribution, incentive systems, data privacy and concerns about knowledge risks, can be overcome.

AI Technologies

Many specific technologies are discussed as supporting KM. Most technologies have their roots in the field of AI. Results from AI research play a crucial role in the development of KMS and provide intelligent functions for KM. Examples for AI-based tools for KM are as follows:

- *Experience and know-how data base systems* are ordered collections of application solutions, such as specialized data base systems that store experiences, lessons learned, best practices, as well as technical solutions. Experience data bases rely technologically on conventional information retrieval and document management technology, augmented with business process models and ontologies about the application domain as well as additional meta data categories for describing knowledge documents. The term experience data base aims more at management, organizational, and technical experiences, such as customer relations, business processes, projects, whereas the term know-how database aims more at technical problems and solutions.
- *Case-based reasoning systems* provide an approach to solve problems with the help of known solutions for similar problems that has its roots in AI research. The approach is composed of four steps: (1) retrieve cases from the system's case base which are similar to the problem presented by the user, (2) reuse solved cases, (3) revise the selected case and confirm the solution, and (4) retain the learned case if it is an interesting extension of the case base.
- *Recommender systems* extend systems that support information retrieval and give recommendations

based on techniques such as test of context correspondence, frequency analysis, and agent technologies. Some authors also use the term *collaborative filtering* to denote the social process of recommending. The systems collect and aggregate recommendations of a multitude of people and make good matches between the recommenders and those who seek recommendations. To accomplish this task, recommender systems have to model the users' characteristics, interests, and/or behavior. This action is called user modeling, profiling, or personalization. Profiles are a requirement for the application of many intelligent technologies, especially intelligent software agents. Systems that use content-based filtering recommend items similar to those a given user has liked in the past.

- *Intelligent software agents* are autonomous units of software that execute actions for a user. Intelligent software agents use their intelligence to perform parts of their tasks autonomously and to interact with their environment in a useful manner. Thus, software agents differ from more traditional software programs with respect to their autonomy, ability to communicate and cooperate, mobility, reactive and proactive behavior, reasoning, and adaptive behavior; some agents even might show human characteristics. Roots of agent technology can be traced back to (1) approaches of distributed AI where agents deconstruct tasks into sub-tasks, distribute them, and combine their results and (2) developments in the area of networks and communication systems. Intelligent or semi-intelligent agents can be classified according to their main area of application into information, cooperation, and transaction agents and are applied in a multitude of settings. Prominent examples for agents can be found in electronic market processes. In KM, agents can be used to scan emails, newsgroups, and chats; to group and update automatically user-specific messages and information items in the Internet (newswatchers); to analyze and classify documents; to search, integrate, evaluate, and visualize information from a multitude of sources; to handle information subscriptions intelligently; to identify and network experts; to visualize knowledge networks; and to recommend participants, experts, communities; and documents.

TOWARD A DEFINITION OF KNOWLEDGE MANAGEMENT SYSTEMS

During the last couple of years, the term KMS has gained acceptance in the literature and on the market, but the term is often used ambiguously for specific KM tools, for KM platforms, or for a combination of tools that are applied with KM in mind. Investigations about the notion of KMS often remain on the abstract level of what a KMS is used for, such as "a class of information systems applied to managing organizational knowledge" (13). Consequently, the term KMS is used here, since numerous similar conceptualizations exist that complement the functionality and the architectures of KMS. The following list summarizes the most important characteristics of KMS as found in the literature (see Fig. 2, left-hand side).

Specifics of Knowledge

KMSs are applied to manage knowledge that is described as "personalized information [...] related to facts, procedures, concepts, interpretations, ideas, observations, and judgments" (13). From the perspective of KMSs, knowledge is information that is organized meaningfully, accumulated, and embedded in a context of creation and application. KMSs leverage primarily codified knowledge and also aid communication or inference used to interpret situations and to generate activities, behavior, and solutions. KMS help to assimilate contextualized information, provide access to sources of knowledge and, with the help of shared context, increase the breadth of knowledge sharing between persons rather than storing knowledge itself (13). The internal context of knowledge describes the circumstances of its creation. The external context relates to retrieval and application of knowledge. Contextualization is a key characteristic of KMS that provides a semantic link between explicit, codified knowledge and the persons that hold or seek knowledge in certain subject areas. Therefore, users play the roles of active, involved participants in the knowledge network fostered by KMSs.

KM Initiative

The primary goal of KMS is to bring knowledge from the past to bear on present activities, which results in increased levels of organizational effectiveness (12). Thus, KMSs are the technologic part of a KM initiative that also comprises person-oriented and organizational instruments targeted at improving productivity of knowledge work (23). KM initiatives can be classified into technology-, human-, and process-oriented initiatives (see section on "Knowledge management"). The type of initiative determines the type of KMS for its support. In effect, KMSs aid knowledge work by supporting employees' awareness, networking, exploration, and exploitation of knowledge assets.

Knowledge Processes

KMSs are developed to support and to enhance knowledge-intensive tasks, processes, or projects of knowledge creation, organization, storage, retrieval, transfer, refinement and packaging, (re-)use, revision and feedback (also called the knowledge life cycle) to support knowledge work ultimately (2). In this view, KMSs provide a seamless pipeline for the flow of knowledge through a refinement process (24). Although the focus used to be on explicit knowledge with most KMSs being built on some form of content or document management system, with the advent of sophisticated collaboration technologies and so-called social software, KMSs increasingly target both explicit and implicit knowledge by helping to network people and to share and refine implicit knowledge.

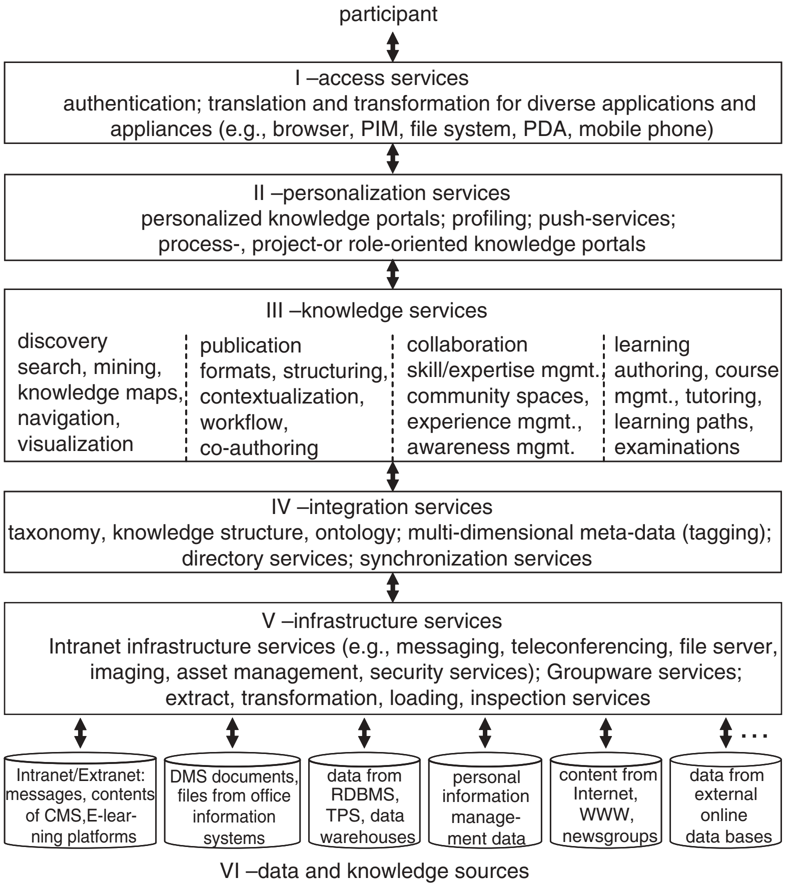

participant
I –access services
authentication; translation and transformation for diverse applications and appliances (e.g., browser, PIM, file system, PDA, mobile phone)
II –personalization services
personalized knowledge portals; profiling; push-services; process-, project-or role-oriented knowledge portals
III –knowledge services
discovery search, mining, knowledge maps, navigation, visualization
publication formats, structuring, contextualization, workflow, co-authoring
collaboration skill/expertise mgmt. community spaces, experience mgmt., awareness mgmt.
learning authoring, course mgmt., tutoring, learning paths, examinations
IV –integration services
taxonomy, knowledge structure, ontology; multi-dimensional meta-data (tagging); directory services; synchronization services
V –infrastructure services
Intranet infrastructure services (e.g., messaging, teleconferencing, file server, imaging, asset management, security services); Groupware services; extract, transformation, loading, inspection services
...
Intranet/Extranet: messages, contents of CMS,E-learning platforms
DMS documents, files from office information systems
data from RDBMS, TPS, data warehouses
personal information management data
content from Internet, WWW, newsgroups
data from external online data bases
VI –data and knowledge sources

Figure 3. Architecture of knowledge management system.

Comprehensive Platform

Whereas the foci on individual initiatives, processes, and participants can be seen as a goal-, application-, and user-centric approach, an IT-centric approach provides a base system to capture and to distribute knowledge (25). This platform is then used throughout the organization. In this case, a KMS is not an application system targeted at a single KM initiative, but it is a platform that can be used either as-is to support knowledge processes or as the integrating base system and repository on which KM application systems are built. In this case, *comprehensive* indicates that the platform offers functionality for user administration, messaging, conferencing, and sharing of documented knowledge, such as publication, search, retrieval, and presentation.

Knowledge Services

KMSs are ICT platforms on which several integrated services are built. The processes that have to be supported give a first indication of the types of services that are needed. Examples are rather basic services (collaboration, workflow management, document and content management, visualization, search and retrieval) or more advanced services (personalization, text analysis, clustering, and categorization) to increase the relevance of retrieved and pushed information, advanced graphical techniques for navigation, awareness services, shared workspaces, (distributed) learning services, as well as integration of and reasoning about various (document) sources on the basis of a shared ontology (15).

KM Instruments

KMSs are applied in many application areas and support KM instruments specifically, such as capture, creation, and sharing of good or best practices; implementation of experience management systems; creation of corporate knowledge directories, taxonomies, or ontologies; competency management; collaborative filtering and handling of interests used to connect people; creation and fostering of communities or knowledge networks; and facilitation of knowledge process reengineering (13,26,27). Thus, KMSs offer a targeted combination and integration of knowledge services that together foster one or more KM instrument(s).

Consequently, a KMS is defined as a comprehensive ICT platform for collaboration and knowledge sharing with advanced knowledge services built on top that are contextualized, integrated on the basis of a shared ontology, and personalized for participants networked in communities. KMSs foster the implementation of KM instruments to support knowledge processes targeted at increasing organizational effectiveness.

Actual implementations of ICT systems certainly fulfill the characteristics of an ideal KMS only to a certain degree. Thus, a continuum between traditional IS and advanced KMSs might be imagined with minimal requirements that provide some orientation (28).

ARCHITECTURE

Many KMS solutions that are implemented in organizations and offered on the market are client/server solutions. Figure 3 shows an ideal layered architecture for KMS that represents an amalgamation of theory-driven, market-oriented, and several vendor-specific architectures (24,29) such as Open Text Livelink, http://www.opentext.com/. A thorough analysis of these architectures and the process of amalgamation can be found in Ref. 23. The ideal architecture is oriented toward the metaphor of a central KM server that integrates all knowledge shared in an organization and offers a variety of services to the participant or to upward layers.

Data and knowledge sources include organization-internal and organization-external sources as well as sources of structured and semi-structured information and knowledge.

Infrastructure services provide basic functionality for synchronous and asynchronous communication, sharing of data and documents, as well as management of electronic assets. Extract, transformation, and loading tools provide access to data and knowledge sources. Inspection services (viewer) are required for heterogeneous data and for document formats.

Integration services help to organize and link knowledge elements meaningfully from a variety of sources by means of an ontology. They are used to analyze the semantics of the organizational knowledge base and to manage meta data about knowledge elements and users. Synchronization services export and (re-)integrate a portion of the knowledge workspace for work offline.

Knowledge services provide intelligent functions for discovery, such as search, retrieval, and presentation of knowledge elements and experts; for publication, such as structuring, contextualization, and release of knowledge elements; for collaboration, such as the joint creation, sharing, and application of knowledge; and for learning, such as authoring tools and tools for managing courses, tutoring, learning paths, and examinations; as well as for reflecting on learning and knowledge processes established in the organization.

Personalization services provide a more effective access to the large amounts of knowledge elements. Subject matter specialists or managers of knowledge processes can organize a portion of the contents and services for specific roles or can develop role-oriented push services. The services can be personalized with the help of (automated) interest profiles, personal category nets, and personalizable portals.

Access services transform contents and communication to and from KMS to fit heterogeneous applications and appliances. KMS have to be protected against eavesdropping and unauthorized use by tools for authentication and for authorization.

Summing up, many functions exist to provide knowledge-related services that have been combined into a KMS architecture. However, this architecture can be seen as ideal in the sense that almost all actual tools and systems offered on the market or implemented in organizations only offer a certain portion of these services. The next section organizes the abundant number of tools and systems that are discussed as being helpful for KM.

CLASSIFICATION

The field is still immature in the sense that no classes of systems exist that the literature has agreed on. Several proposals for classifications of systems exist which lack mostly completeness and also exclusiveness in the sense that one system fits into one and only one category. A comprehensive overview of classifications of technologies, tools, and systems that support KM can be found in Ref. 23. The classifications in the literature fall into two categories. Market-oriented classifications try to cover either technologies, tools, and systems that support KM (wide view) potentially or they cover the functionality of KMSs (narrow view). Theoretical classifications are based on existing models that describe types of knowledge (abstract view) or KM processes or tasks, respectively (concrete view) that could be supported potentially by ICT in general or KMSs in particular. Taken together, KM tools and systems fall into one of the following four categories:

- *Technological roots* This group is composed of more traditional ICT that can be used to support KM initiatives. The most important roots have been described above.
- *Platforms* Corporate Intranet infrastructures, enterprise document and content management systems, or Groupware platforms can be designed “with KM in mind.” These platforms turn infrastructures into comprehensive, integrated KMS solutions. A modern, integrated Intranet platform can be considered as a KM platform in the sense of a kind of “starter solution” for knowledge sharing. This KM platform comprises at least the levels Intranet infrastructure including extract, transformation, and loading, as well as access and security in the KMS architecture presented above. Integrated KMS solutions combine a large set of technologies for knowledge sharing into a common platform.
- *Specialized tools* Some KM tools have roots in the AI field and perform specific functions necessary for KM.

Others are necessary to integrate several of these functions or several of the more traditional ICT. These tools are heterogeneous with each tool targeting a specific challenge within individual steps of knowledge processes or along the knowledge lifecycle.

- *KMS in a narrow sense* These systems provide functionality that goes well beyond the functions in roots, platforms, and specialized tools in that they represent the ICT part of a KM instrument. In an empirical study, large organizations have been surveyed for their KMS implementations (23). Based on the findings of this empirical study together with the findings reported in the literature (13), KMSs (in a narrow sense) can be classified according to the KM instruments they support.

In the next sections, KMSs in a narrow sense are described in detail for exemplary KM instruments (30).

Knowledge Directories

Different types of knowledge maps are suggested for all categories of KM instruments. A central goal is to create corporate knowledge directories that visualize existing knowledge in organizations and support a more efficient access to and handling of knowledge. The main objects of mapping are experts, project teams, networks, white papers or articles, patents, lessons learned, meeting protocols, or generally document stores. Knowledge source maps visualize the location of knowledge, either people (sometimes also called knowledge carrier maps, or information systems) and their relation to knowledge domains or topics. Knowledge asset maps visualize the amount and the complexity of knowledge that a person or a system holds. Knowledge development and application maps are combinations of process models and knowledge carrier maps. Knowledge development maps visualize processes or learning paths that can or must be performed by individuals or teams to acquire certain skills. Knowledge application maps describe what process steps have to be performed in what situation at what step in a business process, such as who should be contacted for a second opinion. Knowledge structure maps show the types of relationships between knowledge domains or topics. The formal definition of knowledge structures results in ontologies and is an important instrument for the integration of diverse knowledge sources.

Competence Management

Competencies held by individuals in organizations are analyzed, visualized, evaluated, improved, and applied systematically. Competence management composes expertise locators, yellow and blue pages, as well as skill management systems, which are also called people-finder systems. Skill management makes skill profiles accessible, and it defines learning paths for employees that have to be updated together with skill profiles. A central skill ontology has to be defined that provides context for all existing, required, and wanted skills in the organization. Training measures have to be offered. Skill management systems often not only contain information about skills, their holders, and their skill levels, but also contain information about job positions, projects, and training measures in which employees learned, used, and improved their skills. Yellow and blue pages are directories of organization-internal and -external experts, respectively. Profiles of the experts together with contact details are listed according to several knowledge domains for which they might be approached. Information about employees' skill levels and degrees of expertise can be used to connect people; to staff projects; to find training and education measures for training into, on, along, near, off, and out of the job (31); to filter; and to personalize KMS contents and functions.

Experience Management

These systems ease documentation, sharing, and application of personal experiences in organizations and have to be integrated into the daily work practices of employees to be accepted. Several approaches exist that support capturing of experiences, such as information mapping, learning histories, or micro-articles. The systematic management of personal experiences enables a company to solve recurring problems more effectively. However, some barriers exist that prevent the documentation of experiences or the reuse of already documented experiences. Foremost, time required for documenting experiences is a critical factor because it imposes additional efforts on employees. Simultaneously, sufficient context of the experience has to be provided. ICT solutions help to ease the effort of documenting experiences, to detect context automatically, and to apply rights management to the knowledge assets.

Communities/Knowledge Networks

Community management targets the creation and the fostering of communities or knowledge networks. Communities differ from knowledge networks with respect to who initiated their foundation. Communities are founded by like-minded people (bottom-up) and can at most be fostered by the organization. Knowledge networks are established and legitimated by management (top-down). However, organizational and ICT measures to foster communities are the same as those used to support knowledge networks. Communities per definition cannot be controlled or induced externally. But organizations can provide employees with time and space to share thoughts; to establish IT tools, such as social software, community builder, and home spaces that support exchange of thoughts; to create new roles like community managers that help to keep discussions going; and to look for important topics that should gain management attention.

Process Warehouses and Support Systems

Knowledge process reengineering (KPR) aims to redesign business processes from a knowledge perspective. The term references the field of business process reengineering (BPR) that aims at fundamental (process innovation) or evolutionary (process redesign) changes of business processes in organizations to increase organizational effectiveness. In addition to traditional BPR instruments, knowledge-intensive business processes are improved partially by KPR. The focus is on designing knowledge processes that connect business processes, defining cooperation scenarios, improving communication patterns between employees, and "soft" skills, or on designing an organizational culture supportive of knowledge sharing (2). Business processes are modeled with the help of modeling techniques. The models are stored in model bases. The model base can be expanded so that it handles not only knowledge about the process, but also knowledge created and applied in the process. This process is termed process warehouse, and it can be used as a foundation for systematic KPR. Examples for contents in process warehouses are exceptional cases, case-based experiences, reasons for decisions, checklists, hints, frequently asked questions and answers, potential cooperation partners, or suggestions for improvements.

Lessons Learned, Good and Best Practices

Lessons learned are the essence of experiences made jointly and documented systematically by members of the organization in projects or learning experiments. In a process of self-reflection, for example at the end of a project milestone, also called after-action review or project debriefing, project members review jointly and document critical experiences made in this project. ICT supports coding, linking, storing, and sharing lessons learned. Templates support structured documentation of experiences and help the team to include important context information. Lessons learned target project experiences and their reasons but ideally make no statement about how processes should be adapted considering these experiences. The sharing of good or best practices is an approach to capture, create, and share experiences in a process-oriented form as procedures or workflows that have proven to be valuable or effective within one organizational unit and may be applied in other organizational units. As managers might argue about what exactly is "best" practice, several organizations use different levels of best practice, such as a good (unproven) idea, good practice, local best practice, company best practice, industry best practice. Permanent best practice teams provide guidelines and support identification, transfer, implementation, evaluation, and improvement of practices.

Semantic Content Management

"Semantic" is used here to indicate that content is embedded in context, and it is well described with the help of meta data that assigns meaning and structure to the content. These descriptions are machine-interpretable and can be used for inferencing. Semantic content management extends document management and enterprise content management. Certainly, the instrument is related tightly to an IT solution, but rules must exist that guide definition and use of semantics, monitor external knowledge sources for interesting content that should be integrated, develop an appropriate content structure, as well as publish semantically enriched documents in the system. Semantic content management also allows for "smart" searching and collaborative filtering, and it can be integrated with competence management to handle interests used to connect people with the help of the joint analysis of semantic content and skills.

KMS can also be distinguished according to the main organizational level on which they focus. The list contains a wider set of KM-related tools and systems, as KMSs in a narrow sense span the three levels, which are described as follows:

1. *Enterprise KMS* Includes enterprise-wide broadcasting systems, knowledge repositories, enterprise knowledge portals, directory services, meta-search systems, knowledge push systems with information subscriptions, community support, knowledge visualization systems, knowledge work process support, learning management systems, social network services and intelligent agents that support organizational information processing.
2. *Group and community KMS* Includes community builders and workspaces; Wikis; theme, project, or community Blogs; ad-hoc workflow management systems; multipoint communication systems, such as listserver, newsgroups, group video conferencing, and collaboration systems; and intelligent agents that support information processing in groups.
3. *Personal KMS* Includes personal search systems, such as desktop search, user profiling, search filters, knowledge discovery and mapping; point-to-point communication systems, such as email, point-to-point video conferencing, or instant messaging; personal Blogs; and intelligent agents that support personal knowledge management for knowledge search, sharing, integration, or visualization.

KMSs available on the market fall into at least one of these categories. A classification of KMS can only be considered as preliminary because of the considerable dynamics of the market for KMS. At this stage, the analysis of KMS is a challenge.

STATE-OF-PRACTICE

In the following list, the state of practice of KMS is summarized in the form of theses that describe activities that concern KMSs in German-speaking countries as investigated in an empirical study (23):

1. Almost all large organizations have an Intranet and/or Groupware platform in place that offers a solid foundation for KMSs. These platforms, together with a multitude of extensions and add-on tools, provide good, basic KM functionality, which includes the easy sharing of documents and access to company information.
2. Large organizations have already implemented KM-specific functions. Many implemented functions are

not used intensively, in some cases because of technical problems, but mostly because they require substantial organizational changes and significant administrative effort.

3. Most organizations rely on organization-specific developments and combinations of tools and systems rather than on standard KMS solutions. The market for KMS solutions is confusing and dynamic, and integration with existing systems is often difficult. Organizations might also fear the loss of strategic advantages if they exchange their home-grown KMS solutions for standard software.
4. Explicit, documented knowledge is emphasized strongly. This finding is not surprising because in many cases, large amounts of documents have existed already in electronic form and an improved handling of documents and the redesign of corresponding business processes can improve quickly organizational effectiveness. A trend toward collaboration and learning functions exists, because technical requirements for media-rich electronic communication can now be met at reasonable costs.
5. Comprehensive KMS are highly complex ICT systems because of (1) the technical complexity of advanced knowledge services and of large volumes of data, documents, messages, links, as well as contextualization and personalization data, (2) the organizational complexity of a solution that affects business and knowledge processes throughout the organization; and (3) the human complexity because of the substantial change in habits, roles, and responsibilities that is required as KMS have to be integrated into daily practices of knowledge work.
6. In many organizations, a multitude of partial systems are developed without a common framework to integrate them. Some organizations also build enterprise knowledge portals that at least integrate access to ICT systems relevant for the KM initiative. Only recently, comprehensive and integrated KMS offer functionality integrated within one system.

CONCLUSION

KM is a lively and dynamic field that composes technology-, human-, and process-oriented approaches. A KM initiative is one characteristic that distinguishes KMSs from more traditional systems that represent the roots of KMS. The most important services offered by KMS in the sense of comprehensive platforms can be systematized with the help of an architecture. In a narrow sense, KMSs can be classified with the help of the KM instruments they support. The field of KM has changed considerably during the last 20 years. Particularly, since KM has come back after some years of declining interest after the dot-com bubble burst, several new or extended facets have developed. Some are described briefly in the following list as potential trends for the future of KM:

Business. Whereas during the initial development of the field the focus was mostly on the knowledge side of KM, now the field concentrates increasingly on management with increasing activities to measure the outcome of KM, business models, and the integration of knowledge processes into the business process landscape of organizations. KM instruments, particularly those that are supported by information and communication technologies, are reframed and recombined as services.

Collaboration. Generally, a shift in perspective of KMS vendors has occurred, as well as organizations that apply those systems from a focus on documents that contain knowledge to relationships between resources and people, a combination and integration of functions for handling internal and external context, locating experts, competency management, and so on. Advanced services that support collaboration in teams and communities, link knowledge providers and seekers as well as e-learning functionality, have been integrated into many KMSs. To be successful, such KMSs should also consider diversity of knowledge workers, which is another megatrend that will likely have its impact on KM.

Mobility. KMSs are still developed with the knowledge worker at the desktop in mind. However, mobile devices will have sophisticated knowledge tools for the knowledge worker no matter where he or she might be. Also, substantial research and development activities exist surrounding the "Internet of things," i.e., the ubiquitous availability of computing power embedded into artifacts.

Knowledge Ecosystem. The success of easy-to-use content management systems and social software on the Internet increases the proportion of active contributors of all Internet users greatly. Because of network effects, the user bases show tremendous growth rates, and phenomena develop that are sometimes described as collective intelligence. Breaking the boundaries of the organization and having customers, suppliers, or the entire business ecosystem that surround an organization contribute to its knowledge base might be a viable option for many businesses and organizations. Thus, KMS would be extended from an organizational knowledge base, an organizational memory, toward a knowledge ecosystem that is enhanced by more than just the organization's members.

Safety. Most KM instruments aim to increase transparency, sharing, and reusing of knowledge. However, an important new strand in the field aims at prioritizing knowledge assets and balancing chances and risks of easing access to valuable and competitively superior knowledge. A systematic management of knowledge risks identifies, assesses, governs, and evaluates knowledge risks with respect to knowledge-intensive business processes.

These trends might continue as many organizations strive to profit from the promised benefits of comprehensive ICT platforms for increasing productivity of knowledge work and, consequently, organizational effectiveness, and for fostering an organizational environment conducive to attract and to retain creative knowledge workers.

BIBLIOGRAPHY

1. E. N. Wolff, The growth of information workers, *Communicat. ACM*, **48**(10): 37–42, 2005.
2. T. H Davenport, S. L Jarvenpaa and M. C. Beers, Improving knowledge work processes, *Sloan Management Review*, **37**(4): 53–65, 1996.
3. I. Nonaka, The knowledge-creating company, *Harvard Bus. Rev.*, **69**(11–12): 96–104, 1991.
4. K.-E. Sveiby and T. Lloyd, *Managing Knowhow*, London, 1987.
5. K. M. Wiig, Management of Knowledge: Perspectives of a New Opportunity, in Bernold, T. (Ed.): *User Interfaces: Gateway or Bottleneck?,* Proceedings of the Technology Assessment and Management Conference of the Gottlieb Duttweiler Institute Rüschlikon/Zurich (CH), 20–21 October 1986, Amsterdam 1988, 101–116.
6. E. Ortner, Informations management. Wie es entstand, was es ist und wohin es sich entwickelt, *Informatik-Spektrum*, **14**: 315–327, 1991.
7. T. H. Davenport, Business process reengineering: where it's been, where it's going, in V. Grover, W. J. Kettinger (eds.), *Business Process Change: Reengineering Concepts, Methods and Technologies*, Harrisburg, (PA), 1995, pp. 1–13.
8. M. H. Zack, Developing a knowledge strategy, *California Manage. Rev.*, **41**(3): 125–145, 1999.
9. C. Argyris, D. Schön, *Organizational Learning: A Theory of Action Perspective*, Reading, MA: Addison-Wesley, 1978.
10. M. T. Hansen, N. Nohria and T. Tierney, What's your strategy for managing knowledge?*Harvard Bus. Rev.*, **77**(3–4): 106–116, 1999.
11. K. M. Wiig, What future knowledge management users may expect, *J. Knowledge Managem.*, **3**(2): 155–165, 1999.
12. E. Stein and V. Zwass, Actualizing organizational memory with information systems, *Informat. Sys. Res.*, **6**(2): 85–117, 1995.
13. M. Alavi, D. E. Leidner, Review: knowledge management and knowledge management systems: conceptual foundations and research issues, *MIS Quarterly*, **25**(1): 107–136, 2001.
14. D. Binney, The knowledge management spectrum - Understanding the KM landscape, *J. Knowledge Manage.*, **5**(1): 33–42, 2001.
15. U. M. Borghoff and R. Pareschi, eds., *Information Technology for Knowledge Management*, Berlin: Springer, 1998.
16. P. Meso and R. Smith, A resource-based view of organizational knowledge management systems, *J. Knowledge Managem.*, **4**(3): 224–234, 2000.
17. R. L. Ruggles, The state of the notion: knowledge management in practice, *California Manage. Rev.*, **40**(3): 80–89, 1998.
18. W. H. Inmon, *Building the Data Warehouse*, New York, 1992.
19. WfMC – Workflow Management Coalition, ed. *Terminology & Glossary*, Document no. WFMC-TC-1011, Issue 3.0, Hampshire (UK)1999. Available: http://www.wfmc.org.
20. G. DeSanctis and R. B. Gallupe, A foundation for the study of group decision support systems, *Management Sci.*, **33**(5): 589–609, 1987.
21. G. Alonso, F. Casati, H. Kuno, V. Machiraju, *Web Services. Concepts, Architectures and Applications*, Berlin: Springer, 2004
22. S. Boyd, Are You Ready for Social Software?*Darwin Online Magazine*, May 2003. Available: http://www.darwinmag.com/read/050103/social.html.
23. R. Maier, *Knowledge Management Systems: Information And Communication Technologies for Knowledge Management*, 3rd ed. Berlin: Springer, 2007.
24. M. H Zack, Managing codified knowledge, *Sloan Management Rev.*, **40**(4): 45–58, 1999.
25. M. Jennex and L. Olfman, Organizational memory, in C. W Holsapple (ed.), *Handbook on Knowledge Management*, vol. 1. Berlin: Springer, 2003, pp. 207–234.
26. R. McDermott, Why information technology inspired but cannot deliver knowledge management, *California Managem. Rev.*, **41**(4): 103–117, 1999.
27. E. Tsui, Tracking the role and evolution of commercial knowledge management software, in C. W. Holsapple, ed., *Handbook on Knowledge Management*. vol. 2. Berlin, 2003, pp. 5–27.
28. R. Maier and T. Hädrich, Centralized versus peer-to-peer knowledge management systems, *Knowledge Proc. Managem. — T J. Corpor. Transform.*, **13**(1): 47–61, 2006.
29. W. Applehans, A. Globe, G. Laugero, *Managing Knowledge. A Practical Web-Based Approach*, Reading, MA: Addison-Wesley, 1999.
30. R. Maier, T. Hädrich and R. Peinl, *Enterprise Knowledge Infrastructures*, Berlin: Springer, 2005.
31. C. Scholz, *Personal management*, 5th ed., Munich: Vahlen, 2000.

FURTHER READING

T. H. Davenport, G. J. B. Probst, eds., *Knowledge Management Case Book*, 2nd ed. Erlangen: Publicis, Wiley, 2002.

C. W. Holsapple, ed., *Handbook on Knowledge Management*, Berlin: Springer, 2003.

R. Maier, *Knowledge Management Systems: Information And Communication Technologies for Knowledge Management*, 3rd ed. Berlin: Springer, 2007.

R. Maier, T. Hädrich,and R. Peinl, *Enterprise Knowledge Infrastructures*, Berlin: Springer, 2005.

RONALD K. MAIER
University of Innsbruck
Innsbruck, Austria

MACHINE LEARNING

DEFINITION AND STATE

Learning, as defined in Webster's dictionary, is the act to gain knowledge or skill or a behavioral tendency by study, instruction, or experience. As such, learning has found its place in diverse fields ranging from philosophy to psychology and pedagogy. This definition of learning is broad and mostly refers to human or animal learning. Machine learning, on the other hand, is more specific about its domain of coverage and has been defined by several authors in the fields of computer science and engineering. For example,

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.(1)
>
> ... the system can acquire new knowledge from external sources, or the system can modify itself to exploit its current knowledge more effectively.(2)
>
> The goal of machine learning is to build computer systems that can adapt and learn from their experience.(3)
>
> Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both.(4)

Thus, machine learning is about programming a computer to improve a performance measure through experience in performing certain tasks. For this purpose, a computer program is used that adapts to its environment by gaining knowledge through experience. In this setting, we also assume that the adaptability is positive, namely a performance measure exists to assess the fitness of the program in the environment and this measure should improve as more examples are given to the program to learn. The requirement of an improving performance measure differentiates machine learning from learning in psychology where behavioral change, not necessarily improving some performance measure, can be considered a form of learning.

Machine learning has a history of about 50 years. From the early day of game-playing computers to the modern practice of data mining, machine learning has developed into a multidisciplinary field in academia. Currently, advanced degrees (master and Ph.D. level) are being offered in this field (Carnegie Mellon University), a central repository of benchmark data has been established (5), and annual conferences and meetings in this area are conducted continually for academic researchers and industrial practitioners. Also, several academic journals are devoted to the communication of ideas and practices in machine learning.

THE LEARNING PROCESS

To achieve the goal of machine learning, we need to consider several things in the process of machine learning.

Reasoning

Learning can sometimes be achieved by simply memorizing patterns. A human learns by memorizing several key points in a field. For a computer program, memory is implemented by using efficient data structures and good database systems. For most cases, memory cannot store all patterns in a learning process; thus, proper reasoning is needed. Two types of reasoning exist: deductive and inductive. A deductive reasoning works with rules and facts to deduce more facts, and an inductive reasoning distills useful hypotheses from a bundle of data by using data summarization. Most current machine learning algorithms involve inductive reasoning. Different memory structures and reasoning procedures may result in different combinations of knowledge representations, features, and algorithms.

Model of Knowledge

The objective of machine learning is to produce a model of knowledge. Such a model can be expressed in two forms: a black-box form and a white-box form. The former focuses on the utility of relationships between inputs and outputs regardless of how such relationships are organized and expressed; and the latter emphasizes the representation of relationships between inputs and outputs in a readable and understandable manner. Selecting suitable representation of models for a task often requires domain knowledge in the area where machine learning is applied to improve a performance measure. Machine learning has been used to perform tasks of classification (e.g., written character recognition), prediction (e.g., credit risk prediction), and problem solving (e.g., robot control). Different tasks require different means to represent the models of knowledge efficiently. For example, robot control may be implemented easily if the knowledge model is expressed in terms of rules, and written character recognition may achieve a higher performance level when the model is represented with neural nets. Knowledge representation depends on not only the nature of a task, but also on current technologies in algorithms design. In addition to the rules and neural nets representation, other popular forms of knowledge representation include frames, maps, and decision trees.

Feature Representation

If knowledge refers to insights to be learned from experience, then features stand for the ways to represent experience. Just like knowledge, features depend heavily on the task and experience. What kinds of features are to be collected? Do we and can we collect them at our will? How are they represented, and how do we avoid noise in the collected features? Many ways exist to represent features.

For example, tuples from a relational database and predicates from logical programming are popular forms for feature representation. Several techniques in statistics can be used to detect outliers and hence improve the quality of collected data. Research is are being conducted to deal with the issues related to features.

Learning Algorithms

As machine learning is about gaining knowledge from experience, a learning algorithm that generates a satisfactory model of knowledge plays an important role. This role is usually accomplished by searching through the space of all possible models. Frequently, this space of models is so large that an exhaustive search is impossible; therefore, a heuristic search processes must be considered. Some algorithms may simply use a greedy search procedure (e.g., inductive decision tree) or a mathematical optimization by taking advantage of the nature of a problem (e.g., a quadratic optimization procedure in kernel-based algorithms). Besides search procedures, the presentation order of experience may also affect the final result significantly. Learning algorithms can be categorized from different perspectives and will be discussed later.

General Procedure of Machine Learning

Starting with the task of a machine learning job, we need to decide what kinds of memory and reasoning are needed to fulfill the task. Based on this analysis, we can choose the best combination of knowledge representation, features, and algorithms to achieve higher performance measure when more experiences are available. A generic processing procedure of machine learning can be illustrated like the one shown in Fig. 1.

In this process, three phases exist, i.e., the training phase, the testing phase, and the validation phase. First, data describing the domain problems to be processed are collected and represented in proper formats. Depending on the types of domain problems, data can be labeled or unlabeled by users. According to some selection criteria, these data are divided into three, usually disjointed, subsets, called training data, testing data, and validation data, respectively. In the training phase, training data are fed into learning algorithms to produce models of knowledge for the domain problem. The correctness of the produced models is evaluated against the testing data with respect to specific error functions. At this moment, parameters of a model are tuned accordingly for improving the quality of learning. Such a produce-and-modify process iterates for several times until the quality criteria are satisfactory. Finally, one or more models may be obtained. Then, the validation phase is invoked to test the validity of the models learned, and the best model will survive for applications. As the model is learned based on subsets of the domain problem, it can be considered a partial model of the underlying domain problem. If the validation results are not satisfactory, the learning process goes on again by selecting new training data or by modifying the learning algorithm. Otherwise, the learning algorithm is effective and the model is accepted. Afterward, the final model learned from this process is taken for practical applications.

Note that, depending on the design concepts of the learning algorithms and application problems, some learning systems invoke all three phases and some do not. The learning process can be performed in a batch style or in an incremental style. In a batch mode, all training data are fed into the learning algorithm and the testing data are used for verifying after the model is produced and are modified according to all training data. Conversely, an incremental learning process produces a model for application based on some training data and updates the model when new training patterns are encountered. Additionally, the selection of data for training, testing, and validation is critical and should be considered seriously to learn an effective and complete model for the final applications.

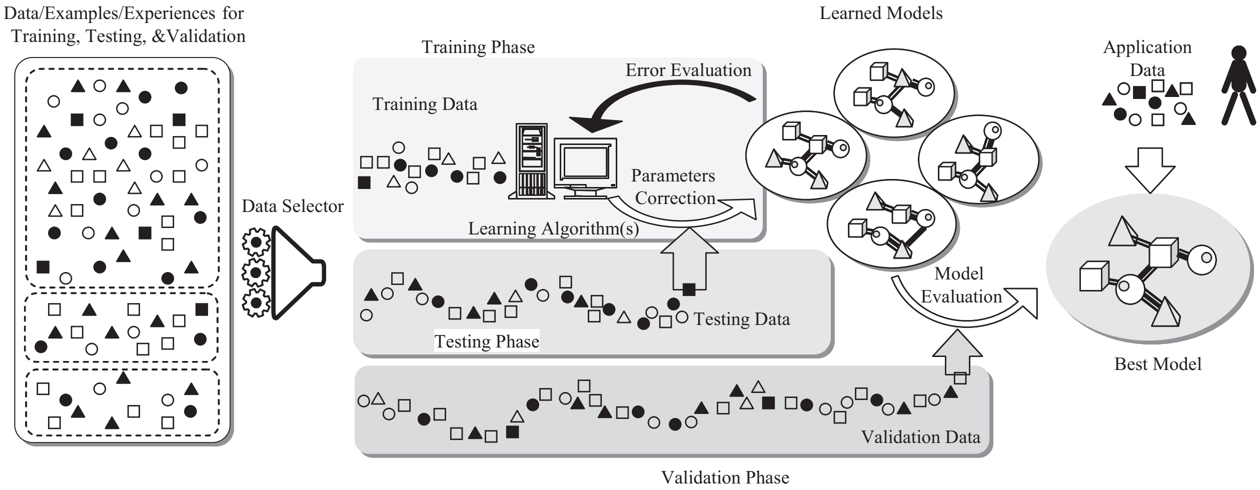


Figure 1. A generic flow of machine learning.

LEARNING METHODS

Many studies in machine learning are devoted to algorithm designs. Computing methods in machine learning can be categorized from different perspectives. For example, they can be classified into supervised, unsupervised, and reinforcement leaning depending on the existence of a *teacher* in the learning environment. Based on the feature representations, some symbolic and numeric algorithms take advantage of different representations of features. There are lazy and rigor algorithms. A typical example of a lazy learner is the nearest neighbor learning algorithm that does not have a separate training phase. Most algorithms belong to the category of rigor algorithms, in which a model must be learned first before it can be used. Also, algorithm independent procedures are used to combine different learners. These procedures include bagging and boosting meta-learners. Below, we discuss the most commonly used algorithms in this field.

Supervised Learning

Supervised learning is a machine learning technique that handles training data labeled with the desired output to guide the progression of learning. The output can be a discrete value (called classification) or a continuous value (called regression). In the case of classification learning, all output classes have a well-defined meaning to users of this type of learning. For example, the output can be a benign or malignant tumor in a medical examination. The input variables should be important factors that affect the output. The goal of supervised learning is to learn a model that predicts an output given a new input case. The model can be assessed based on different performance measures such as accuracy, precision, and recall rates. The representative algorithms of supervised learning include artificial neural networks (ANNs), support vector machines (SVMs) (6), and inductive decision tree (ID3).

Pattern recognition is the most common application of supervised learning techniques, which has been applied widely to image or speech recognition. For example, in digital, handwritten character recognition, digitalized handwritten characters are labeled by their real characters and are collected as patterns of training data. Learning algorithms of classification such as ANN and SVM are employed to produce classification models for the given training data. A performance measurement can be based on the accuracy of successful identification of characters. Until the classification model is rebuilt, application data of new handwritten characters are classified by the learned model. The most similar class label with the lowest error is the recognized character for the new input data.

Unsupervised Learning

Unsupervised learning is different from supervised learning in that no teacher is involved in the learning process. In other words, no desired output exists for reference. The goal of the learner is to attempt to find the regularities or patterns in the input data. Two classic examples of unsupervised learning are clustering and dimensionality reduction. Clustering partitions a set of data points into clusters of data so that intracluster homogeneity is high and intercluster homogeneity is low. For many applications, clustering outputs can be used to extract key concepts from the data set, and the extracted concepts become meaningful labels used in a supervised learning. The judgment of a clustering result can be very subjective in some applications. Lately, objective clustering validation indices have been developed to handle certain types of input data. The self-organizing map (SOM) and expectation-maximization (EM) are two well-known unsupervised learning algorithms.

Unsupervised learning of customer segmentation is useful for business applications. Transactional and demographic data of customers are used to find clusters of customers with similar consumption behavior. Special promotions can be designed for each cluster of customers to boost the sales of products or services. Learning algorithms such as support vector clustering (SVC) and SOM can be used for this purpose. Similar ideas have been used in text mining to find meaningful concepts from texts expressed in natural languages or Web mining to find browsing patterns of users.

Reinforcement Learning

Reinforcement learning (RL) is an approach to machine intelligence that combines the fields of dynamic programming and supervised learning to yield powerful machine learning systems. In RL, the learner (a decision-making agent) is simply given a goal to achieve and then it learns how to achieve that goal by performing enough trial-and-error interactions with its environment. Three fundamental parts camprise a RL model: (1) a discrete set of environment states, (2) a discrete set of agent actions, and (3) a set of scalar reinforcement signals. On the other hand, two main strategies are used for solving RL problems. The first strategy is to search in the space of behaviors to find one that performs well in the environment. The second strategy is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world (7). The Q-learning method and temporal difference approach are two popular and widely used reinforcement learning algorithms.

Game playing, such as chess or poker, is one of the promising application areas of reinforcement learning. A game-playing system normally faces a huge search space. Traditional approaches tackle this NP-complete problem by using parallel or distributed algorithms with problem partitioning. Using reinforcement learning-based methods, tactics or strategies for winning the games are learned from interactions with human players or other programs. Patterns learned in this case are game-playing sequences that win the games.

Genetic Programming

Genetic programming (GP) is an automated method to generate a working computer program for solving hard optimization problems (8). GP provides a general paradigm to solve problems ranging from formula seeking (symbolic regression) to automated circuit designs. Like its sibling algorithm, the genetic algorithm (GA), GP is based on Darwinian evolutionary theory to search heuristically in a large solution space to find a near-optimal solution. Unlike GA, GP can use more flxible data structures to represent a chromosome. A typical chromosome in GP is a computer program like $v_1 + \mathrm{sin}(\mathrm{iif}(v_2 > 0, 1, 0))$ that usually is represented as a parsing tree. Leaf nodes in the tree include problem-dependent variables and random constants, which together are called the terminal set of a GP. On the other hand, internal nodes formed by problem-dependent operators constitute the function set of a GP.

Like other evolutionary algorithms, the fitness to the problem of each parsing tree must be assessed, and this usually is done with the setup of a test environment. Based on fitness values, evolutionary operators (selection, crossover, and mutation) are applied repeatedly to a population of parsing trees until a preset stopping criterion has been met. The final solution is then extracted from the last generation of individuals.

Instance-Based Learning

The most basic instance-based method is the k-nearest neighbor (k-*NN*) algorithm. The procedure of k-NN runs as follows: Given a new instance, the distance between the instance and all samples in the training set is calculated. The distance used in practically all nearest-neighbor classifiers is the Euclidean distance. With the distance calculated, the samples are ranked according to the distance. Then the k samples that are nearest to the new instance are used to assign a classification label or a regression value to the case. As no separate phase of model learning exists, k-*NN* is considered a lazy type of learning algorithm. Two issues related to this algorithm are choosing the number of neighbors (k) and the quick determination of nearest neighbors. The former issue usually is solved by a trial-and-error type of approach, whereas the latter issue is solved by using well-designed data structures.

Bagging

Bootstrap aggregating (or bagging) is a meta-algorithm that can be used to improve the performance of classification and regression models (9). Bagging has its roots in statistics, in which bootstrap technology is used to sample training data with a replacement. Initially, a base supervised learning algorithm is called B such as a decision tree or a neural net, and a training set T of size n. We generate m data sets $T_1, \ldots, T_m$, each with the size n by sampling with a replacement from T. That is, for each data set T_j, a training example is sampled from T with uniform distribution and is returned to the population for next sampling. This procedure is repeated until n examples have been collected for T_j. Then, the algorithm B is applied to T_j to get a prediction model B_j. If the prediction output is a class label, a simple majority vote from outputs of B_j's is used to determine the final class label. On the other hand, if the output is a regression value, then the average of outputs from B_j's is taken to be the output for the bagging predictor.

Bagging has the advantage of increased prediction accuracy and reduced instability of base models. By aggregating different versions of a base model, sensitivity with respect to training data in certain supervised learning algorithms can be reduced. For example, it is known that the output from a decision tree is very sensitive to the training set. If a bagging predictor is constructed using decision tree as the base learner, accuracy can be improved substantially.

Boosting

Boosting is another meta-algorithm that ensembles base learners to perform supervised learning (10). Similar to bagging, boosting trains different versions of a base learner by using different samplings from the original training set. Unlike bagging, each sampling of the training set in boosting does have its own probability distribution for choosing the data. Although training of a bagging predictor is parallel in nature because bootstrap samplings of training sets can be conducted simultaneously, boosting is sequential because the distribution function for sampling data is determined in a sequential order.

The most popular boosting algorithm is called AdaBoost (Adaptive Boosting). Freund and Schapire (10) have indicated that AdaBoost can boost a weak learning algorithm—one that performs slightly better than random guessing—into a strong learning algorithm, which can generate a predictor with arbitrarily low error rate provided that sufficient data are available.

OTHER CONSIDERATIONS

Connections with Other Fields

As a multidisciplinary field, machine learning is related closely to and has gained many great ideas from other scientific fields. These fields include computer science, artificial intelligence, cognitive science, and statistics, among others. The discussion below is based on the historical development of machine learning and the close relationships among the fields.

- *Computer Science*: Because machine learning is about an adaptive computer program, it benefits from advancements and shares many problems in computer science. An algorithmic breakthrough in computer science may advance machine learning to the next level of achievement. Likewise, tractability and complexity theory in computer science constrains the development of machine learning as well. However, differences still exist between these two fields. As computer science emphasizes correct programming, machine learning requires writing programs that can learn.
- *Artificial Intelligence*: Originated as a subfield of artificial intelligence (AI), machine learning has a close tie to AI. History shows that AI has many other subfields such as planning, pattern recognition, natural language processing, and expert systems that may or may not be related closely to machine learning. Traditionally, AI offers more domain-specific solutions via machines, whereas machine learning provides more general-purpose algorithms that can be used in many different settings. Heuristic search algorithms in AI have advanced the capability of machine learning in finding appropriate models, and development and per-

fection of machine learning algorithms have made AI more practical in daily applications.

- *Cognitive Science*: Machine learning has been applied successfully to areas that are hard to explain but can be performed easily by human beings. These areas of applications frequently involve human cognition or perception. For example, it is easy for a person to recognize written characters or spoken words, but it is hard to explain the cognition process and code a program directly to perform such tasks. Today, optical character recognition and speech recognition software based on machine learning algorithms that adapt to different writing styles or speaking accents has provided the best performance in these areas of applications. Machine learning may gain additional improvement by using discovery in cognitive science, and cognitive science may understand human or animal cognition process better by using results from machine learning. For example, reinforcement learning can be used to explain the dopaminergic neuron activity in a brain (11).
- *Statistics*: Much of machine learning work uses inductive reasoning to draw a reasonably good model from a set of data. Statistics has long been known for its ability to summarize results from experimental data. Thus, machine learning and statistics share many principles of data summarization. Learning algorithms such as classification and regression tree (CART) and clustering have their roots in statistics. On the other hand, machine learning has helped statistics to advance its capability in handling a large amount of data by using computers.

Caveat

When we employ machine learning to solve scientific or engineering problems, limitations and constraints associated with it should be well considered. For example, the output attribute predicated by the ID3 must be discrete valued, and the attributes tested in the decision nodes of a tree must also be discrete valued (1). Some delicate phenomena worthy of additional consideration are listed in the following.

- *Concept drift*: A difficult problem with machine learning in many real-world applications is that the target concepts are not stable but may shift from time to time. For example, today's companies collect a large amount of data like sales figures and customers' preferences to find patterns of customer behavior and to predict future sales. As the customer behavior tends to change over time, the model built on old data is inconsistent with the new data and must be updated accordingly. This problem generally is known as concept drift (12).
- *Overfitting*: Given a model space $\boldsymbol{H}$, a model $h \in \boldsymbol{H}$ is said to overfit the training data if some alternative model $h' \in \boldsymbol{H}$, exists such that h has a smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances (1). Empirical error-(i.e., error based on the training data)-based learning algorithms such as decision trees or neural networks often have this type of unwanted behaviors and must be treated carefully. Structural error-based learning algorithms such as support vector machines consider structural complexity in addition to the empirical error when judging the goodness of a model, and they usually can reduce the overfitting effect.
- *Occam's razor principle*: Occam's razor is a logical principle attributed to the Fourteenth century logician and Franciscan friar, William of Occam (or Ockham). The principle states that "Entities should not be multiplied unnecessarily." In other words, the principle states that simpler explanations are more plausible and any unnecessary complexity should be shaved off (4). Sometimes, this principle is also called the "principle of parsimony" or "principle of economy." This principle can be used to help avoid the overfitting phenomenon.
- *The minimum description length principle* recommends choosing the model that minimizes the description length of the model plus the description length of the data given the model. Bayes's theorem and basic results from information theory can be used to provide a rationale for this principle (1).
- *Wolpert and Macready*(13) proposed "no free lunch theorems." They show that all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions. In particular, if algorithm $\boldsymbol{A}$ outperforms algorithm $\boldsymbol{B}$ on some cost functions, then loosely speaking, as many other functions where $\boldsymbol{B}$ outperforms $\boldsymbol{A}$ must exist exactly.
- *The ugly duckling* theorem demonstrates that no such thing as a class of similar objects exists in the world, insofar as all predicates (of the same dimension) have the same importance (14). In the absence of assumptions on features, any two patterns are "equally similar" regardless of the patterns involved. The implication for machine learning is that no canonical set of features exists for any given classification task. Good feature representations of patterns must be problem dependent.
- *The famous Garbage-In–Garbage-Out rule* can be found in many practices of machine learning. Because machine learning is a compound process of selecting data, generating and adjusting models of knowledge, testing the validity of the model, and so on, each stage in the process may produce defective outputs to the next stage. This result could be from imperfect data collected or the limitations of learning algorithms adopted. Users need to pay attentions to each stage when working with machine learning to gain the best performance.

Future Outlook

The techniques of machine learning have been applied successfully to various areas of applications. However, with the growth of versatile data and extensive demands, research on machine learning also becomes a never-ending story. Below are several issues worthy of additional study.

- *Problem representation and data selection*: Feature collection and representation can play a central role in a machine learning job. For the past few decades, efforts on machine learning have focused mainly on effective and efficient learning algorithms and have assumed that data for training or testing are well prepared. However, with the growth of the Internet, data to be processed have become versatile and voluminous, probably containing lots of noises, redundancies, or errors and being represented unstructuredly. Also, data are not static any more; they may be observed in a snap shot and be described by features that are changing with time. These features make the preparation of data for learning algorithms more difficult in practical applications. Methods of proper representation of the underlying problems, selection of the most significant data subset for complete learning, being tolerable or robust of noises, or errors for scalable applications are issues that need to be addressed.
- *More adaptive and robust learning algorithms*: Most current learning algorithms are static in model learning and utilization. Retraining usually is necessary when a lot of new data not learned in the training phase are encountered. Future learning algorithms are expected to be robust and adaptive no matter how dynamically the data are fed and must even be able to change the learning structures and strategies used in the algorithm when necessary. Some of today's learning algorithms like GP can only fulfill these requirements partially.
- *Hybrid learning algorithms*: Each learning algorithm has its pros and cons. Hybridizing two or more heterogeneous learning algorithms that can produce complementary results is a trend in practical applications. Two strategies are used most often. The first one is to make the learning process an n-tier process in which each stage employs one learning algorithm. The second one is to integrate different learning algorithms in a task, for example, GA-based adjustment of ANN structures.
- *Semisupervised learning*: During the last few years, semisupervised learning (SSL) has received increasing attention in the machine learning research community. The basic idea behind it is to learn not only from the labeled training data but also to exploit the structural information in additionally available unlabeled data; that is, SSL combines labeled and unlabeled data during training to improve performance. SSL has been applied successfully to both classification and clustering problems.
- *Measurement criteria*: The success of learning algorithms is measured by performance measurements. Most measurement criteria are defined objectively, and their applicability and usefulness are case dependent. Defining measurement criteria to reflect truly insights of learning algorithms and models of knowledge for the applications is a fundamental yet important issue.
- *Non-monotonic issues*: As mentioned most learning algorithms assume that input data are well prepared and that training data are true. It is possible that data originally considered true are proved to be inadequate by new evidences. When models need to be revised because of the arrival of new data, non-monotonic reasoning and non-monotonic truth maintenance that are important and hard issues in AI must be handled properly.

BIBLIOGRAPHY

1. T. M. Mitchell, *Machine Learning*, New York: McGraw Hill, 1997.
2. J. W. Shavlik and T. G. Dietterich, *Readings in Machine Learning*, San Francisco, CA: Morgan Kaufmann, 1990.
3. T. Dietterich, Machine learning, in R. A. Wilson and F. C. Keil (Eds.), *The MIT Encyclopedia of the Cognitive Sciences*, Cambridge, MA: MIT Press, 2001.
4. E. Alpaydin, *Introduction to Machine Learning*, Cambridge, MA: MIT Press, 2004.
5. A. Asuncion and D. J. Newman, UCI Machine Learning Repository, Irvine, CA: University of California, Department of Information and Computer Science, 2007. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html.
6. V. Vapnik, *The Nature of Statistical Learning Theory*, New York: Springer, 1995.
7. L. P. Kaelbling and M. L. Littman, Reinforcement learning: a survey, *J. Artific. Intelli. Res.*, **4**: 237–285, 1996.
8. J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza., *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Boston, MA: Kluwer Academic Publishers, 2003.
9. L. Breiman, Bagging predictors, *Mach. Learning*, **24**(2): 123–140, 1996.
10. Y. Freund and R. E. Schapire, A short introduction to boosting, *J. Japanese Soc. Artif. Intelli.*, **14**(5): 771–780, 1999.
11. S. Nieuwenhuis, C. B. Holroyd, N. Mol, and M. G. Coles, Reinforcement-related brain potentials from medial frontal cortex: origins and functional significance, *Neurosci Biobehav. Rev.*, **28**(4): 441–448, 2004.
12. G. Widmer and M. Kubat, Learning in the presence of concept drift and hidden contexts, *Mach. Learning*, **23**(1): 69–101, 1996.
13. D. H. Wolpert and W. G. Macready, No free lunch theorems for search, Technical Report SFI-TR-95-02-010, Santa Fe, NM: Santa Fe Institute, 1995.
14. S. Watanable, *Knowing and Guessing: A Quantitative Study of Inference and Information*, New York: Wiley, 1969.

CHIH-CHIN LAI
National University of Tainan
Tainan, Taiwan

SHING-HWANG DOONG
Shu-Te University
Kaohsiung, Taiwan

CHIH-HUNG WU
National University of
Kaohsiung
Kaohsiung, Taiwan

NEURAL CONTROLLERS

INTRODUCTION

Modern control systems have steadily evolved into complex devices that are characterized by their increased nonlinearity, flexibility, intelligence, and enhanced capability to handle uncertainty. Conventional control techniques such as robust control and adaptive control, which largely rely on a well-formulated model of the plant to be controlled, fail to address these issues in situations where a precise analytical model of the plant may be difficult to obtain due to nonlinearity, uncertainty, or complexity. These and similar issues have led to a desire for development of better control techniques that are intelligent, adaptive, self-learning, and capable of handling highly uncertain, nonlinear, and complex systems whose dynamics may be time-delayed, ill-defined, or simply unavailable. Moreover, the solutions of such issues have also contributed to the development of new concepts such as autonomous control with a need for sensor fusion, decision making, planning, and learning. The objective for meeting new challenges in the field of control has led to a reappraisal of existing conventional control techniques. Neural networks (1–3), because of their ability to model nonlinearity and uncertainty and their nondependence on mathematical model of plant, and their learning ability, have been a natural choice as controllers and system identifiers for the practitioners in the controls community. Figure 1 shows a typical scheme in which neural network controller and neural network plant model can be used. In this figure, two neural networks are illustrated. One neural network produces control action to drive the plant, and the other one is the model of the plant. The figure also shows the feedback scheme, which is used to train the networks online, making them adaptive to changes in plant behavior.

A neural network (NN) (4–6) is an information-processing paradigm inspired by the manner in which the heavily interconnected, parallel structure of the human brain processes information. The human brain is the most complex computing device in nature. An NN is a computer model that attempts to match the functionality of the brain in a very fundamental manner. The key feature of the NN paradigm is the novel structure, which is composed of a large number of highly interconnected processing elements (called neurons) that are coupled together with weighted connections (analogous to synapses). The information, which is represented as a bundle of signals, is passed between neurons via connection links that get multiplied by connection weights and get transformed with the help of activation functions at each neuron. In other words, NNs are collections of mathematical models (representing mathematical operations in layered fashion) that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. Similar to biological systems, learning in neural networks typically occurs via training or exposure to an accurate set of input/output data where the training algorithm iteratively adjusts the connection weights. These connection weights store the knowledge necessary to solve specific problems.

NNs have been implemented in several architecture. A typical multilayer feed-forward NN is shown in Fig. 2. The figure shows the inputs and the two layers (one hidden layer and one output layer) of neurons. In general, several hidden layers may exist in an NN. Each input node is connected to all neurons (nodes) in the next layer (hidden layer), and each node in the hidden layer is connected to all neurons in the output layer. Each of these connections has a weight associated with it, and a bias is associated with each node. Each node also applies an activation function to the sum total of its input. Another kind of network is a recurrent network, which contains feedback elements from the output, with time delay. This type of network is very effective for recognizing not only spatial patterns but also temporal patterns. The Hopfield network (7,8) and the Elman network (9) are examples of this kind of network. Another type of network, called the radial basis function network (RBFN) (10,11), was introduced as an alternative to the feed-forward network for approximating continuous functions. The RBFN has a feed-forward structure consisting of a single hidden layer of locally tuned units that are fully interconnected to an output layer. All hidden units simultaneously receive the n-dimensional real-valued input vector x. The hidden unit outputs are not calculated using the weighted sum/sigmoidal activation mechanism. Rather, each hidden unit output is obtained by calculating the “closeness” of the input x to an n-dimensional parameter vector associated with the hidden unit.

Currently, NNs are being applied to several complex real-world problems. They are very efficient and robust at recognizing and classifying patterns, and they possess the ability to make effective decisions in the presence of imprecise input data. They offer ideal solutions to a range of problems such as speech, character, and signal recognition, as well as functional approximation and prediction, and system modeling where the dynamical processes are not well formulated or are extremely complex. The primary advantage of NNs lies in their robustness against uncertainty in the input data and in their capability of learning. They are often effective for solving complex problems that do not have an analytical solution or for which an analytical solution is too difficult to be found.

NEURAL CONTROLLER AND LEARNING

A control system is said to have learning capabilities if the system acquires any information pertaining to the environment, which is unknown in the system’s internal model of the environment and uses that information for updating the model for future estimation, classification, identification, or control such that overall performance of the system

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

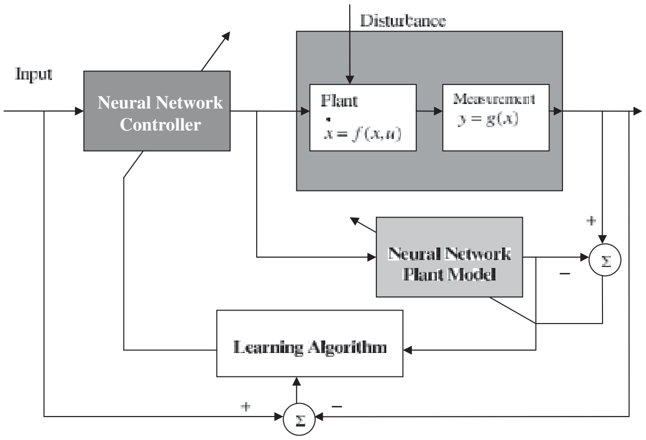


Figure 1. A typical neural network control scheme.

is improved. If this process of learning is carried out while the control system is in operation, the system is said to possess adaptive learning capabilities. An adaptive learning control system has the ability to use information it gained in the past to improve its performance in the future. For the control system to be completely autonomous, it should be able to handle any uncertainty that might arise in the plant and in the environment. In addition, it should be able to operate in a large range of operating conditions, and it should be able to adjust itself to any change in dynamics of the plant. In conventional control techniques, learning occurs by tuning the parameters of the controller (e.g., by tuning proportional gain (K_p), integral gain (K_i), and derivative gain (K_d) of the PID controller by means of the Zeigler–Nichols method). In the NN, learning occurs by adjusting the weights of the connection between neurons.

The field of neural networks had its beginning in the 1940s, with the pioneering work of McCulloch and Pitts (12) on the model of elementary computational neuron. The next major development occurred with the introduction of the *Hebbian Learning Rule* (13) in 1949, which reads as follows: "*When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knob (or enlarges them if they already exist) in contact with the soma of the second cell*." In other words, during a learning process in the brain, repeated activation by one neuron (cell) to another increases the conductance of synapse between the two neurons. This learning concept was used by Rosenblatt (14) to develop the first artificial neuron (or *perceptron*, as he called it) with the capability to learn. Widrow and Hoff (15) in 1960, developed a model of a neuron called ADALINE (ADAptive LInear NEuron), which could learn quickly and accurately, based on a powerful learning rule called the *Widrow–Hoff Learning Rule*. This learning rule first introduced the concept of supervised learning using a "teacher" that guides the learning process based on a least-mean-square (LMS) algorithm. In 1962, Rosenblatt (16) introduced the *perceptron learning rule* and proved that a perceptron could be trained to learn whatever it represents. However, in 1969, the research in the field of NNs received a major setback when Minsky and Papert, in their book *Perceptrons*(17), proved that single-layer networks, which were in use then, were limited in their abilities to process data and were theoretically incapable of solving many problems, including the *Exclusive-OR* logical function. These findings and similar ones led to a stagnation in research on NNs for almost two decades until 1986, when Rumelhart et al., (18) introduced an error backpropagation algorithm to train multilayer NNs and overcame the limitations of the single-layer networks. This development triggered off a renewed interest in NNs, and since then, there has been a flurry of research activities in this field leading to a well-developed science that has found applications in several areas, such as the controls, space, manufacturing, medical sciences, and process industries.

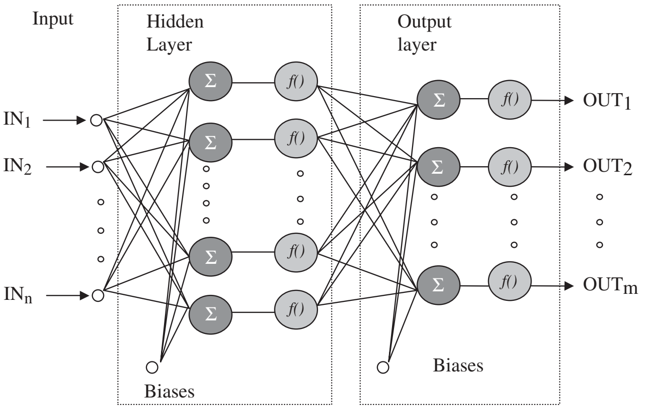


Figure 2. A multilayer feed-forward neural network.

Learning is the most important part of NNs, and intense research has been carried out in devising efficient learning algorithms (19). The three most popular learning algorithms are as follows: the supervised, the unsupervised, and the reinforced learning methods. The supervised learning attempts to reduce the error between the actual and the desired outputs and needs a training dataset comprised of inputs and desired outputs. The objective to minimize the error, along with the presented input–output data, act as a teacher. Backpropagation, based on the gradient descent method, is one of the most popular methods that falls under this category. Unsupervised learning does not involve a teacher guiding the learning process. The data presented to learn the network in this case do not include the input–output pairs. Instead, the parameters of the network (the connection weights and biases) are adjusted based on clustering techniques such as the self-organizing Kohonen's map (20) or competitive learning techniques (21) upon presentation of input patterns. Reinforcement learning is similar to the reward and punishment method in psychological learning. The reinforcement learning technique (22) is based on maximizing the reward or reinforcement signal as a consequence of action taken by the controller. Reward or reinforcement signal is obtained by means of a utility function that calculates a sum of future rewards, which represents correctness of action in some form. This kind of learning technique has received increased interest because of its adaptive feature and the autonomous manner in which the system learns while interacting with the environment.

NEURAL NETWORK APPROXIMATION AND IDENTIFICATION: A CONTROLS PERSPECTIVE

Three-layered NNs (i.e., one input layer, one output layer, and one hidden layer), with the hidden layer having sufficient nodes and a sigmoid transfer function, and a linear transfer function in the input and output layer are considered to be universal approximators (23,24). They can approximate functions to an arbitrary accuracy. This ability of NNs to approximate large classes of nonlinear functions accurately makes them a popular candidate for use in obtaining a dynamic model of a nonlinear plant. The process of obtaining a model of the plant that can approximate the plant response (output) under stimuli (input) is called identification and is an important part of control. The NN, being an excellent function approximator, can identify complex nonlinear governing equations of plant dynamics. By making the weights of the connections and biases of neurons adjustable, the NN can be extremely adaptive to changes in system and environment parameters. Such NN models can aid in the development of an efficient controller. Figure 3 shows the scheme demonstrating how an NN can be made to learn (via supervised learning) to emulate an unknown function or plant dynamics.

The nonlinear dynamics of a plant can be described by the following pair of difference equations expressed in discrete time:

$$\begin{aligned} x(k+1) &= f(x(k), u(k)) \\ y(k) &= h(x(k)) \end{aligned} \tag{1}$$

where *x(k)* is the state of the system at time step k, *y(k)* is the output vector that can be measured, *u(k)* is the applied input, and f and h are nonlinear operational functions. Some issues that arise in neural modeling of dynamical systems (such as uniformity of the approximation and whether the model would be able to capture the underlying (continuous time) differential equation) have been dealt with by Zbikowski and Dzielinski (25). The usual analytical way to model and control a plant governed by Equation (1) has been to perform linearization about an operating point to obtain equations of the form:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{2}$$

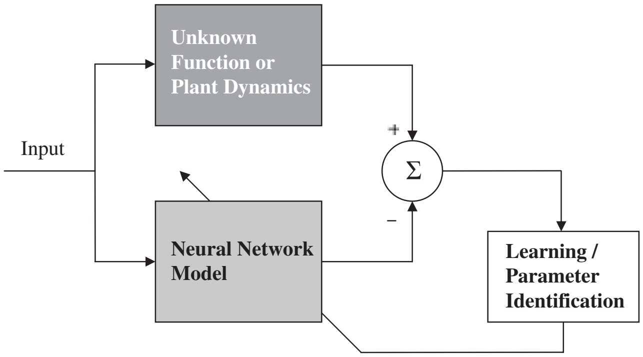

Unknown Function or Plant Dynamics
Input
+
Σ
−
Neural Network Model
Learning / Parameter Identification

Figure 3. Neural network for identification/function approximation.

The system represented by the above set of equations is controllable and observable in the neighborhood of specified operating point. Under some assumptions (26), it can be shown via an implicit function algorithm and inverse mapping theorem (27) that the nonlinear system can be controlled using nonlinear controllers in the neighborhood of the operating point. Extensive computer simulations have further shown that the range within which the system can be controlled is substantially larger than the neighborhood within which the linear model is valid.

The real problem of identification arises when the functions f and h are unknown. In that case, it becomes necessary to estimate those functions from input–output data. Two neural networks N_1 and N_2, with sufficient number of nodes, are used for approximating those functions. If the state variables *x(k)* is accessible, then both networks N_1 and N_2 can be easily trained because input and output data are available. However, if state variables are not accessible, then the error between the plant output and the NN output $(y(k) - \hat{y}(k))$ can be used to train the network via static and dynamic backpropagation (28). The past values of input and output can also be fed as input to the NN via tapped delay line (TDL) to assist in better identification of the plant. These NN models can be used further in arriving at the linearized feedback control of the plant.

The system identification model, which is obtained from the NN, can also be used to predict the plant outputs and controlled variables over a specified time horizon. This information can be extremely useful to optimize the controller to minimize the tracking error. The multistep predictions of output variables (29) via NNs can be made by either using multiple network outputs or recursively using a single network output.

The use of feed-forward networks, recurrent networks, and RBFNs for approximation and identification tasks has been studied extensively, and it has been shown that these networks can approximate arbitrary functions from one finite-dimensional space to another with desired accuracy. It is also true that the same properties are shared by polynomials, trigonometric series, splines, and orthogonal functions. Several research studies have revealed that NNs are more robust to uncertainties and noise, more fault-tolerant, easily implementable on hardware, and enjoy numerous other practical advantages over conventional methods.

NEURAL NETWORKS AS CONTROLLERS

The universal approximation capabilities of multilayer neural networks have made them a very popular choice for identifying nonlinear processes and implementing nonlinear controllers. Several ways (30) exist in which NNs can be made to implement a controller. For example, an NN can be made to mimic a human expert. In this approach, the expert human knowledge, intuition, and experience about the system and appropriate control actions for an ill-defined system are captured in the form of NN mapping (31). The resultant NN emulates the expert and may even perform better than the expert because of its property of ignoring the mistake outliers while being trained. Here, the discretized perception of information and decision of control action by the human operator expert is replaced by continuous interpolations of the network. Similarly, an NN can emulate a well-designed nonlinear controller. This emulation is particularly helpful when a given controller requires a large amount of computational or tuning effort for each process condition. The NN, in this case, approximates the nonlinear controller and the control action eliminates the extra computations and tuning needs. Several applications have been reported in literature where NNs have been used to emulate the controller. Typical examples include multimodel optimal control (32), time optimal control (33), and one-step inverse control (34). In another approach, the NN is trained based on open-loop input/output data to identify an inverse model of the plant. The controller then uses the desired output values to calculate the appropriate control action (input) and is also implementable in the feedback model (35). However, this method faces a problem when no unique inverse model of the plant exists or when the mapping is inadequate for the dynamic plant.

An extensive research effort has been undertaken in the field of controls based on NNs (36–39), and several different architectures (40), using NNs for identification and control of nonlinear systems, have been proposed. This section will provide a brief overview of some popular architectures.

Internal Model Control

A schematic of the structure of the neural internal model control (NIMC) is shown in Fig. 4. Two NNs are included in the control architecture. The first NN, which acts as controller, is generally trained to represent the inverse model of the plant. The second network is the NN model of the plant. The error between the output of the NN plant model and the output of the plant is used as a feedback signal. The NN plant model and the NN controller can be trained offline, and then it can be made to improve over time by letting it learn online. Internal model control (IMC) using conventional methods has been used in the process industry for a long time. As the control strategy is based on a model of the plant, its performance is greatly dependent on the accuracy of the plant model. IMC based on conventional methods tends to perform poorly in the presence of large modeling uncertainty or external disturbances. However, the use of NNs in IMC with online training of the networks has shown significant improvement (41) in system performance.

Model Predictive Control

The model predictive control (MPC) scheme minimizes the future output deviation from the set point, while taking into account the control action needed to achieve the prescribed objective. Neural MPC, based on the receding horizon control (RHC) technique (42), uses an NN plant model to estimate/predict future plant responses to potential control signals over a specified time horizon. An algorithm then computes control actions that optimizes the performance or minimizes the objective function:

$$J(N_1,N_2,N_u) = \sum_{j=N_1}^{N_2}\{y_r(t+j) - y_m(t+j)\}^2 + \rho\sum_{j=1}^{N_u}\{u(t+j-1) - u'(t+j-2)\}^2 \tag{3}$$

where N_1 and N_2 define the minimum and maximum output prediction horizons and N_u is the control horizon. The variable u' is the tentative control signal, y_r is the desired response, and y_m is the network model response. The parameter ρ determines the contribution that the sum of the squares of the control increments has on the performance index. The predictive controllers penalize the excessive control action by providing nonzero ρ. The prediction horizons specify the range of future predicted outputs to be considered, and the control horizon specifies the number of control moves required to reach the desired goal. An MPC architecture based on NN is shown in Fig. 5. As shown in the figure, the NN plant model provides predictive estimates of the states of the plant to the controller, which computes the control action based on the optimization of the objective function given by Equation (3). Predictive control based on conventional methods has proven to be very successful for linear systems. However, for nonlinear systems, the unavailability of accurate models limits the use of MPC. The NNs, with their ability to model nonlinearities in the system accurately, eliminate this problem, and their use in MPC has resulted in robust controllers in a variety of practical situations where the nature of the nonlinearity of the system is not known. However, controllers based on the MPC technique are computationally intensive because of the technique's requirement of carrying out the multistep optimization process. An NN can be made to learn the

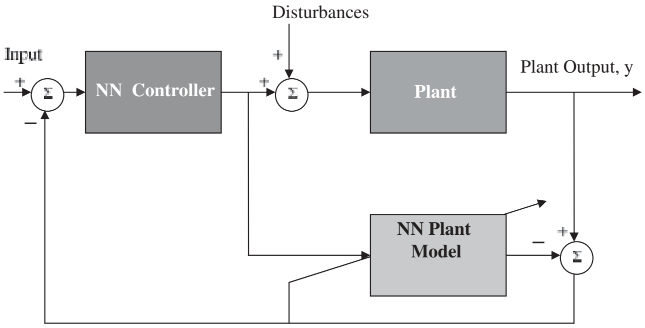


Figure 4. Neural internal model control.

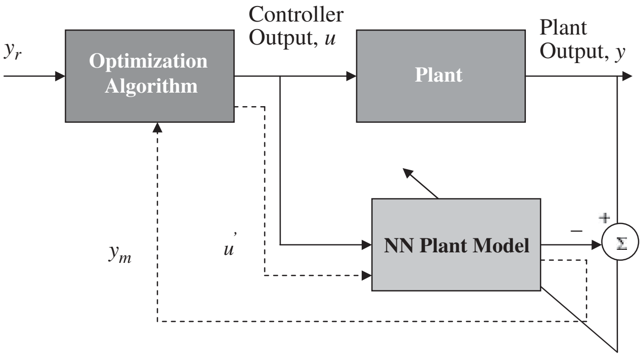


Figure 5. Neural model predictive control.

optimization process and can replace the optimization function after the network has been trained satisfactorily.

Feedback Linearization Control

Feedback linearization control (FLC) aims to transform nonlinear system dynamics into linear dynamics by eliminating nonlinearities. The nonlinear autoregressive moving average (NARMA) model (43) under certain conditions, provides an exact input–output representation of a nonlinear system. One popular NARMA model for identifying nonlinear systems is the NARMA-L2 (44) model. This model can be represented by the following equation:

$$\hat{y}(k+d) = f[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)] + g[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)]u(k+1) \tag{4}$$

where $d \geq 2$. This equation relates the past n plant outputs and past m plant inputs (and one-step future input) to estimate the d-step future output. This estimate of future output can be made to track the desired reference output $y_r(k+d)$. The controller can be defined as

$$u(k+1) = \frac{y_r(k+d) - f[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)]}{g[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)]} \tag{5}$$

which can be realized for $d \geq 2$. Hence, the control scheme begins first with the identification of the plant with the help of an NN. The past plant inputs and past plant outputs can be fed to the NN model via TDLs, and the network can be trained offline and then made to be adaptive to all online data. Figure 6 shows the architecture of neural adaptive feedback linearization control based on the NARMA-L2 model. As shown in the figure, the controller consists of two NN models of the nonlinear functions f and g. The controller receives the past inputs via the TDLs, and implements the logic of Equation (5). The NN models of the functions f and g receive feedback from the plant and get trained online.

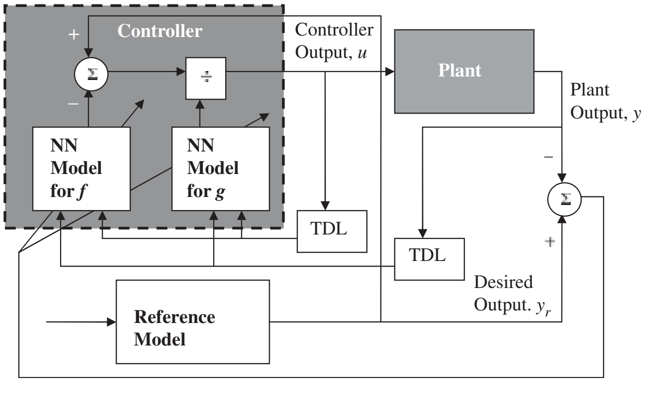


Figure 6. Neural feedback linearization control.

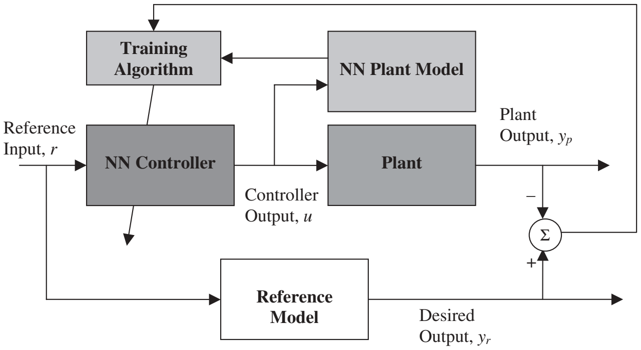


Figure 7. Neural model reference control.

Neural Model Reference Control

The neural model reference control (45) (NMRC), as shown in Fig. 7, uses two NNs: a controller network and a network for plant model. The controller is trained to make the plant respond to minimize the error between the plant output and the output from reference model, which represents the desired closed-loop dynamics of the plant. This goal is achieved by adjusting the parameters of the controller via the training mechanism that minimizes the error between the reference model and the system. The plant model of the network can be obtained offline or adjusted online (if the parameters of the plant are expected to vary) based on input/output data. This model network is then used to predict the controller changes on plant output, which facilitates the adaptive training of the controller.

NEURO-FUZZY CONTROLLERS

Fuzzy set theory (46,47) was specifically designed to mathematically represent uncertainty and vagueness and to provide formalized tools for dealing with the imprecision that is intrinsic to many real-world problems. Designing a fuzzy inference system requires describing human knowledge/experience linguistically. The inference system captures these traits in the form of fuzzy sets, fuzzy logic operation, and fuzzy rules. The ability of fuzzy logic to deal with uncertainty and noise, and its simple understandable linguistic structure, has motivated researchers to use it for controlling complex systems for which precise analytical models may not be available. However, the design of a fuzzy logic controller suffers from certain problems regarding the selection of membership function characteristics (e.g., type and number of membership functions and their shape and range and choosing appropriate fuzzy rules. Developing a rule base is the most time-consuming part of designing a fuzzy logic controller. Thus, a need exists for developing efficient methods to tune membership functions, i.e., to obtain their optimal shapes, range, and number.

Both NN and fuzzy logic (48–52) are model-free estimators and share the common ability to deal with uncertainties and noise. Both of them encode the information in a parallel and distributed architecture in a numerical framework. Hence, it is possible to convert a fuzzy logic architecture to an NN and vice versa. This capability makes it possible to combine the advantages of both NNs and fuzzy logic. A network obtained in this manner could use excellent training algorithms that NNs have at their disposal to obtain the parameters that would not have been possible in a fuzzy logic architecture alone. Moreover, the network obtained this way would have the transparency of a rule-based fuzzy system, because this network would have fuzzy logic capabilities to interpret its actions in terms of linguistic variables. This fusion of two powerful control mechanisms has led to the research and development of neuro-fuzzy controllers (where fuzzy controllers are formulated using the learning capabilities of NNs) and fuzzy neural systems (where fuzzy techniques are applied to speed up the learning or fuzzification of NN is carried out to process fuzzy inputs).

Several algorithms have been developed that address the problem of learning fuzzy rules and tuning the membership functions in an NN architecture. The adaptive-network-based fuzzy inference system (ANFIS) developed by Jang (53) is one of the pioneering works in this field. ANFIS is a fuzzy inference system developed within the framework of an adaptive network (which is a superset of all kinds of feed-forward NNs with supervised learning capabilities). The learning rule proposed for this method is basically a hybrid of the gradient descent method and the least-squares technique, which are implementable both offline (batch learning) and online (pattern learning). This approach, based on the gradient descent method, implements a Sugeno-like fuzzy system, which uses differentiable functions. Subsequent to the development of the ANFIS approach, several methods were proposed for learning rules and for obtaining an optimal set of rules. For example, Mascioli et al. (54) proposed to merge the min–max and ANFIS models to obtain a neuro-fuzzy network and to determine the optimal set of fuzzy rules. Lin and Lee (55) proposed an NN based fuzzy logic control system (NN-FLCS), which learns the structure and parameter of the network to develop fuzzy logic rules and finds the optimal input–output membership functions. A few other neuro-fuzzy approaches developed in recent years include GenFIS (56), NEFGEN (57), FDIMLP (58), and NEFCON (59).

NEURAL CONTROL APPLICATIONS

Since their development, controllers based on an NN have found application in several fields (60), including robotics and automated manufacturing, machining (61), uncertain systems (62,63), aerospace, communication systems, consumer appliances, electric power systems, process engineering, micro-electromechanical systems (MEMS), and power electronics and motion control. NNs have played an important role in machine intelligence by integrating sensors, actuators, software, and computers to make machines capable of acquiring information efficiently and by responding safely and rapidly to both deterministic and unexpected events. Intelligent machines (64) play a potentially important role in a variety of areas, such as manufacturing, the service industry, space explorations, defense, and nonmilitary operations. NNs, along with other soft computing algorithms such as fuzzy logic and genetic algorithm, have been used widely in several intelligent machines and have been a popular choice for machine learning. Typical examples of intelligent machines that have been developed using the NNs include consumer appliances such as washing machines (65), expert systems for medical diagnosis (66,67), and intelligent system (68) for agriculture in Japan.

NNs with their unique ability for pattern recognition in data, classification, and nonlinear functional mapping have been successfully used in areas of knowledge management and data mining with applications in economics, finance, accounting, and marketing. NNs have proved to be theoretically sound alternatives to traditional statistical approaches, and many applications (69–71) in business management and finance have been reported. NN-based modeling and prediction tools have been used in forecasting (72,73), credit card fraud detection (74), credit evaluation (75), bankruptcy prediction (76), state revenue forecasting (77), and prediction of regularities in foreign exchange rates (78). In the areas of marketing and business management, NNs have been used in mining useful information and generating knowledge (79) from a large pool of data that can suitably support marketing decisions and customer relationship management. The following description is of two specific fields with case studies where neural controllers have been successfully used.

Robotics

The field of robotics has provided tough challenges for the controls community because of inherent system nonlinearity, presence of backlash and nonlinear friction, existence of uncertainty from neglected dynamics, and difficulty to obtain precise analytical models of the system. In view of these reasons, NNs have been a favorite choice for identification and control (80,81) of robotic systems. Since the late 1980s, in the field of robotics, NN controllers have been found useful in applications such as manipulator control (82), path planning (83), contact control and grasping (84), multiple robot coordination (85,86), and mobile robot autonomous navigation (87). A case study showing the use of NNs in robot manipulator position control (88) is described as follow.

The dynamics of a rigid n-link robot manipulator can be expressed in Lagrange's form:

$$\tau = M(q)\ddot{q} + V_m(q,\dot{q})\dot{q} + G(q) + F(\dot{q}) + \tau_d \tag{6}$$

where $q(t) \in R^n$ is the vector containing joint angles, $M(q)$ is the inertia matrix, $V_m(q,\dot{q})$ is the matrix containing Coriolis and centrifugal forces, $G(q)$ is the vector containing terms from gravity, $F(\dot{q})$ is the friction, τ_d is the bounded unknown disturbances, and $\tau(t)$ is the vector of input joint torques. The robot is required to follow a desired trajectory $q_d(t) \in R^n$. The tracking error is given by

$$e(t) = q_d(t) - q(t) \tag{7}$$

and the filtered tracking error is defined by

$$r = \dot{e} + \Lambda e \tag{8}$$

where Λ is a positive semi-definite design parameter matrix. Using Equations (6), (7), and (8), the robot dynamics can be written as

$$M\dot{r} = -V_m r + f + \tau_d \tag{9}$$

where *f* is a nonlinear function containing robot parameters (such as link masses, lengths, and inertia), joint friction coefficients, and payload information, and is given by

$$f(x) = M(q)(\ddot{q}_d + \Lambda\dot{e}) + V_m(q,\dot{q})(\dot{q}_d + \Lambda e) + G(q) + F(\dot{q}) \tag{10}$$

The vector x is defined as $x = \left[e^T e^{\dot{T}} q_d^T q_d^{\dot{T}} q_d^{\ddot{T}}\right]$. The control law is given by

$$\tau = \hat{f} + K_v r - v \tag{11}$$

where $\hat{f}$ is the estimate of function $f(x)$ and K_v is the gain matrix. The signal $v(t)$ is the robustifying signal responsible for compensation of unaccounted disturbances. Figure 8 shows the controller structure, which makes use of the NN in the inner loop to approximate function $\hat{f}$. Several ways (89) exist to tune NN parameters and signal $v(t)$, and control gain K_v. The NN controller, designed using this approach, guarantees the tracking error to be bounded by

$$|r| \le \frac{\varepsilon_{NN} + b_d + kC}{K_{v_{\min}}} \tag{12}$$

where ε_{NN} is the NN functional reconstruction error bound, b_d is the robot disturbance term bound, and C represents other constant terms. The denominator $K_{v_{\min}}$ is the smallest PD gain. By choosing a larger PD gain, it is possible to limit the tracking error to an arbitrarily small value. As the NN weights and biases are bounded, it can be shown that the control input τ is also bounded. Moreover, the tuning algorithms guarantee that the closed-loop control system has a strict passivity property that makes it robust to unmodeled uncertainties and disturbances.

Industrial Applications

NNs have been used extensively in industrial environments. In particular NNs have found applications in the aerospace industry (90), automated manufacturing (91), electric power systems (92), steel industry (93), and process industry (94).

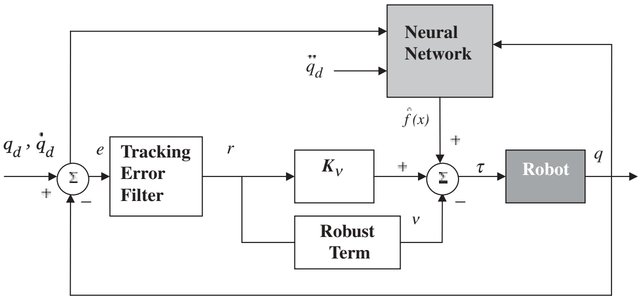


Figure 8. Neural control of a robotic manipulator.

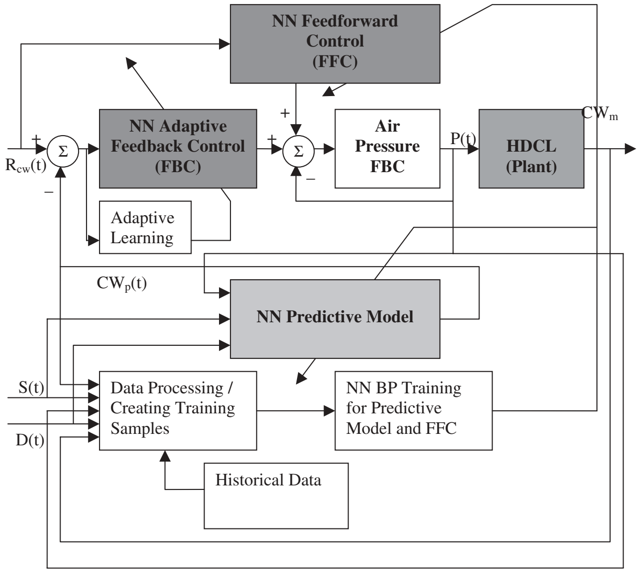

NN Feedforward Control (FFC)
NN Adaptive Feedback Control (FBC)
Air Pressure FBC
HDCL (Plant)
$R_{cw}(t)$
P(t)
CW_m
Adaptive Learning
$CW_p(t)$
NN Predictive Model
S(t)
D(t)
Data Processing / Creating Training Samples
NN BP Training for Predictive Model and FFC
Historical Data
+
–
+
+
–
Σ
Σ

Figure 9. Schematic representation of the HDCL control system.

Lu and Markward (95) have developed an NN-based control system for coating weight control for a hot dip coating line (HDCL), which has been successfully implemented at the Burns Harbor Division of Betlehem Steel Co., Chesterton, IN. The major goals of the control system are to minimize the error between the desired and the actual coating weight, and to minimize the coating weight transition time that determines the sheet transitional footage. The control system, as shown in Fig. 9, consists of two multilayered feed-forward NNs and a neural adaptive controller. These NNs perform coating weight real-time prediction, feed-forward control (FFC), and adaptive feedback control (FBC). Traditionally, these strategies are performed by relevant analytical algorithms and identification models based on the first principles. The NN prediction model takes air pressure P, line speed S, and air knife gap D as inputs and outputs the coating weight CW. The dynamics of the coating process can be represented by the following equation in the Z-domain:

$$CW(z) = G_p F_p(z)P(z) + G_s F_s(z)S(z) + G_d F_d(z)D(z) + \varepsilon(z) \tag{13}$$

where G_p, G_s, and G_d are process gains from P, S, and D, respectively. F_p, F_s, and F_d are Z-transfer functions from inputs P, S, and D to the coating weight, respectively, and ε is the process noise. The measurement equation representing the average coating weight can be written as

$$CW_M(t) = CW(t-\tau) + \omega(t) \tag{14}$$

where τ is the time delay from the pot coating weight to the gauge sensor and ω is the measurement noise.

The neural prediction model is first trained offline via supervised backpropagation, and then the weights are updated in an online training environment. The neural network FFC is an open-loop control scheme that takes S, D, and the desired coating weight R_{cw} to output the FFC component of the air pressure. The training is performed to minimize the desired and predicted coating weights. This network, which is similar to the prediction model, is trained both offline and then online. The NN adaptive FBC, is a self-tuning controller with online learning capabilities that makes use of the adaptive backpropagation learning algorithm.

The NN-based control system designed above has been tested and successfully implemented at the Burns Harbors steel plant. A comparative study of the neural controller with the control system based on the traditional regression model used earlier shows that the NN-based control system (i) provided better prediction results, (ii) had better servo-tracking and robust behavior, (iii) indicated reduced error between target coating weight and actual coating weight (over 69% improvement in average mean), and (iv) showed substantial improvement in "coating weight transitional footage" (9.9% average mean improvement), which indicates how fast the coating weight can reach its new target. The control system could provide a quick response to reach a new target and was robust enough to compensate for the disturbance of line speed.

FUTURE TRENDS

For neural network controllers to be acceptable as a preferred control strategy in industries, the area of neurocontrol needs to establish a sound theoretical foundation. Development of NN-based controllers typically focuses more on algorithms and less on stability issues. Moreover, the black-box nature of the NN leads to industrial apprehensions by control system manufacturers regarding robustness and functioning of the controller. NN-based control, which is a recently developed field, has to compete with fully established and extensively researched conventional control techniques such as PI, PID, optimal, and robust control. Hence, for NNs to gain a wider application in industry, research addressing the significant issues of stability and rigorous analysis needs to be pursued.

NNs provide a perfect platform where numerous learning algorithms can be implemented very easily. This learning capability of NNs can be used to develop goal-driven fully autonomous systems that learn from scratch (as well as from previous examples, experiences, and human knowledge whenever available) with interaction with the environment. Moreover, research needs to be done extensively to fuse other methodologies of intelligent control such as fuzzy logic, expert system, and genetic algorithm to obtain a hybrid system that has the advantages of all component strategies and weaknesses of none. Such hybrid systems would derive their strength from the learning capabilities of biological nerve cells; capabilities of human reasoning, intuition, and experience; and capacity of biological evolutionary mechanisms.

A truly intelligent system needs to acquire all the information about the environment in which it operates. Sensors, which are used in maintaining and updating an intelligent machine's internal description of an external world, enable a system to interact with its environment by providing diverse, redundant, complementary, and timely information. Multiple sensor fusion, which is a systematic method to combine data from a variety of sensory sources, is increasingly becoming an area of active research. NNs, with their learning ability and capacity to approximate functions and recognize patterns, can play an important role in sensor fusion.

SUMMARY

NN-based controllers have recently gained much popularity because of their ability to handle ill-defined, uncertain, and nonlinear problems; their dexterity to learn and adapt to changing situations; and their potential to approximate any function to an arbitrary degree of accuracy. Much research has been carried out in the last two decades on using NNs for controlling complex nonlinear systems, and several architectures have been proposed to use the unique capabilities of NNs to identify and control such systems. Subsequently, neural controllers have found applications in several fields such as the aerospace, communication systems, automated manufacturing, robotics, medical diagnosis systems, electric power systems, and process industries. Research is underway to combine the abilities of the NN with other soft computing methodologies such as fuzzy logic and evolutionary computing to obtain a hybrid intelligent system that derives its strength from human brain-like learning abilities of the NN, intuitive and reasoning capacity of fuzzy logic, and power of biological evolution demonstrated by genetic algorithms. Such hybrid systems would process multiple sensory information to learn and adapt to independently survive and achieve their objectives in an unknown and unstructured environment.

BIBLIOGRAPHY

1. D. E. Rumelhart and J. L. McClelland, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, Cambridge, MA: MIT Press, 1988.
2. S. S. Haykin, *Neural Networks: A Comprehensive Foundation*, Englewood Cliffs, NJ: Prentice Hall Press, 1998.
3. J. E. Dayhoff, *Neural Network Architectures: An Introduction*, New York: Van Nostrand Reinhold, 1990.
4. N. K. Bose and P. Liang, *Neural Network Fundamentals with Graphs, Algorithms and Applications*, New York: McGraw Hill, Inc., 1996.
5. B. Muller, J. Reinhardt, and M. Strickland, *Neural Networks – An Introduction*, Heidelberg: Springer Verlag, 1995.
6. D. Garg, S. Ananthraman, and S. Prabhu, Neural network applications, in John G. Webster, (ed.), *Wiley Encyclopedia of Electrical and Electronic Engineering*, Vol. 41, New York: John Wiley, 1999, pp. 255–265.
7. J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci. USA*, **79**, 2444–2558, 1982.
8. J. Li, A. N. Michel, and W. Porod, Analysis and synthesis of a class of neural networks: Linear systems operating on a closed

hypercube, *IEEE Trans. Circuits and Sys.* **36**(11): 1405–1422, 1989.

9. J. L. Elman, Finding structure in time, *Cognitive Sci.* **14**: 179–211, 1990.
10. D. S. Broomhead and D. Lowe, Multivariable functional interpolation and adaptive networks, *Complex Systems*, **2**: 321–355, 1988.
11. S. Chen, C. F. N. Cowan, and P. M. Grant, Orthogonal least squares learning algorithm for radial basis function networks, *IEEE Trans. on Neural Networks*, **2**(2): 302–309, 1991.
12. W. S. McCulloch and W. H. Pitts, A logical calculus of the ideas imminent in nervous activity, *Bull. Math. Biophy.*, **5**: 115–133, 1943.
13. D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, New York: John Wiley and Sons, 1949.
14. F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychol. Rev.*, **65**: 386–408, 1958.
15. B. Widrow and M. E. Hoff, Adaptive switching circuits, *IREWESCON Convention Record*, Institute of Radio Engineers, New York, **4**: 96–104, 1960.
16. F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Washington, D.C.: Spartan Books, 1962.
17. M. L. Minsky and S. A. Papert, *Perceptrons*, Cambridge MA: MIT Press, 1969.
18. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representations by error propagation, in David E. Rumelhart, James L. McClelland, and *The PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge, MA: MIT Press, 1986. pp. 318–362.
19. B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Englewood Cliffs, NJ: Prentice Hall, 1992.
20. T. Kohonen, *Self Organization and Associative Memory*, 3rd ed., New York: Springer-Verlag, 1989.
21. R. Hecht-Nielsen, *Neurocomputing*, Reading, MA: Addison-Wesley, 1990.
22. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: MIT Press, 1998.
23. K. Hornik, M. Stinchombe, and H. White, Multilayer feedforward network are universal approximators, *Neural Networks*, **2**: 359–366, 1989.
24. B. Irie, and S. Miyake, Capabilities of three–layered perceptrons, *Proc. of IEEE International Conference on Neural Networks*, 1988, pp. 641–648.
25. R. Zbikowski and A. Dzielinski, Neural approximation: A control perspective, in K. J. Hunt, G. R. Irwin, and K. Warwick (eds.), *in Neural Network Engineering in Dynamic Control Systems – Advances in Industrial Control*, London: Springer-Verlag, 1995.
26. K. S. Narendra and S. Mukhopadhyay, Neural networks in control systems, *IEEE Proc. Conference on Decision and Control*, **1**: 1–6, 1992.
27. E. D. Sontag, *Mathematical Control Theory*, New York: Springer-Verlag, 1990.
28. K. S. Narendra, and K. Parthasarathy, Identification and control of dynamical systems using neural networks, *IEEE Trans. Neural Networks*, **1**(1): 4–27, 1990.
29. J. Saint-Donat, N. Bhat, and T. J. McAvoy, Neural net based model prediction control, *Int. J. Control*, **54**(6): 1453–1468, 1991.
30. M. Agarwal, A systematic classification of neural-network-based control, *IEEE Control Systems Magazine*, **17**(2): 75–93, 1997.
31. Y. M. Enab, Intelligent controller design for the ship steering problem, *IEE Proc. Control Theory Appl.*, **143**(1): 17–24, 1996.
32. M. A. Al-Akhras, G. M. Aly, and R. J. Green, Neural network learning approach of intelligent multimodel controller, *IEE Proc. Control Theory Appl.*, **143**(4): 395–400, 1996.
33. C. J. Goh, N. J. Edwards, and A. Y. Zomaya, Feedback control of minimum-time optimal control problems using neural networks, *Optim. Control Appl. Methods*, **14**: 1–16, 1993.
34. J. E. Steck, K. Rokhsaz, and S.-P. Shue, Linear and neural network feedback for flight control decoupling, *IEEE Control Systems Magazine*, **16**(4): 22–30, 1996.
35. M. Ishida and J. Zhan, A policy- and experience-driven neural network and its application to nonlinear process control, *Proc. European Control Conf.*, pp. 471–474, 1993.
36. D. H. Nguyen, and B. Widrow, Neural networks for self-learning control systems, *IEEE Control Systems Magazine*, **10**(3): 18–23, 1990.
37. P. J. Werbos, An overview of neural networks for control, *IEEE Control Systems Magazine*, **11**(1): 40–41, 1991.
38. B. Bavarian, Introduction to neural networks for intelligent control, *IEEE Control Systems Magazine*, **8**(2): 3–7, 1988.
39. A. Delgado, C. Kambhampati, and K. Warwick, Dynamic recurrent neural network for system identification and control, *IEEE Proc. Control Theory and Applications*, **142**(4): 307–314, 1995.
40. M. T. Hagan and H. B. Demuth, Neural networks for control, *Proc. American Control Conference*, **3**: 1642–1656, 1999.
41. Q. A. Li, A. N. Poo, C. M. Lim, and M. H. Ang, Jr., Neuro-based adaptive internal model control for robot manipulators, *Proc. IEEE Intern. Conf. Neural Networks*, pp. 2353–2359, 1995.
42. D. Soloway and P. J. Haley, Neural generalized predictive control, *Proc. IEEE International Symposium on Intelligent Control*, pp. 277–281, 1996.
43. K. S. Narendra, Neural networks for control theory and practice, *Proc. IEEE*, **84**(10): 1385–1406, 1996.
44. K. S. Narendra and S. Mukhopadhyay, Adaptive control using neural networks and approximate models, *IEEE Transactions on Neural Networks*, **8**(3): 475–485, 1997.
45. S. Kuntanapreeda, R. W. Gundersen, and R. R. Fullmer, Neural network model reference control of nonlinear systems, *Interna. Joint Conf. on Neural Networks*, **2**: 94–99, 1992.
46. L. A. Zadeh, Fuzzy sets, *Information and Control*, **8**: 338–353, 1965.
47. R. R. Yager and L. A. Zadeh (eds.), *An Introduction to Fuzzy Logic Applications in Intelligent Systems*, Dordpecht, The Netherlands: Kluwer Academic Publishers, 1991.
48. R. R. Yager and L. A. Zadeh, *Fuzzy Sets, Neural Networks, and Soft Computing*, New York: Van Nostrand Reinhold, 1994.
49. M. Kumar and D. P. Garg, Intelligent learning of fuzzy logic controllers via neural network and genetic algorithm, Paper Number UL_029, *Proc. Japan USA Symposium on Flexible Automation*, Denver, CO, 2004.
50. M. Kumar, and D. Garg, Neural network based intelligent learning and optimization of fuzzy logic controller parameters, Paper Number IMECE 2004–59589, *Proc. of the ASME*

International Mechanical Engineering Congress and Exposition, Anaheim, CA, November 14–19, 2004.

51. M. Kumar, and D. Garg, Neuro-fuzzy controller applied to multiple robot cooperative control, *Industrial Robot: An International Journal*, **32**(3): 234–239, 2005
52. S. Prabhu and D. Garg, Fuzzy logic based reinforcement learning of admittance control for automated robotic manufacturing, *Internat. J. Engineer. Applicat. Artificial Intell.*, **11**: 7–23, 1998.
53. J.-S. R. Jang, ANFIS: Adaptive-network-based fuzzy inference system, *IEEE Trans. Syst. Man and Cybernetics*, **23**(3): 665–685, 1993.
54. F. M. Mascioli, G. M. Varazi, and G. Martinelli, Constructive algorithm for neuro-fuzzy networks, *Proc. Sixth IEEE International Conference on Fuzzy Systems*, **1**: 459–464, 1997.
55. C. T. Lin and C. S. G. Lee, Neural-network-based fuzzy logic control and decision system, *IEEE Trans. Computers*, **40**(12): 1320–1336, 1991.
56. A. Jana, P. H. Yang, D. M. Auslander, and R. N. Dave, Real time neuro-fuzzy control of a nonlinear dynamic system, *Biennial Conference of the North American Fuzzy Information Processing Society*, June 1996, pp. 210–214.
57. A. Rahmoun and S. Berrani, A genetic-based neuro-fuzzy generator: NEFGEN, *Proc. ACS/IEEE International Conference on Computer Systems and Applications*, 2001, pp. 18–23.
58. G. Bologna, FDIMLP: A new neuro-fuzzy model, *Proc. International Joint Conference on Neural Networks*, **2**: 1328–1333, 2001.
59. A. Nürnberger, D. Nauck, and R. Kruse, Neuro-fuzzy control based on the NEFCON-model: Recent developments, *Soft Computing 2*, **4**: 168–182, 1999.
60. Y. Dote, and S. J. Ovaska, Industrial applications of soft computing: a review, *Proc. IEEE*, **89**(9): 1243–1265, 2001.
61. R. E. Haber and J. R. Alique, Nonlinear internal model control using neural networks: an application for machining processes, *Neural Comp. Applicat.*, **13**(1): 47–55, 2004.
62. C. Y. Lee and J. J. Lee, Adaptive control for uncertain nonlinear systems based on multiple neural networks, *IEEE Trans. Syst. Man and Cybernetics, Part B*, **34**(1): 325–333, 2004.
63. S. S. Ge and C. Wang, Adaptive neural control of uncertain MIMO nonlinear systems, *IEEE Transactions on Neural Networks*, **15**(3): 674–692, 2004.
64. C. W. deSilva, *Intelligent Machines: Myths and Realities*, Boca Raton, FL: CRC Press, 2000.
65. T. Nitta, Applications of neural networks to home appliances, *Proc. IEEE Int. Joint Conf. Neural Networks*, 1993, pp. 1056–1060.
66. P. Meesad, and G. G. Yen, Combined numerical and linguistic knowledge representation and its application to medical diagnosis, *IEEE Trans. Systems, Man and Cybernetics, Part A*, **33**(2): 206–222, 2003.
67. F. Schnorrenberg, N. Tsapatsoulis, C. S. Pattichis, C. N. Schizas, S. Kollias, M. Vassiliou, A. Adamou, and K. Kyriacou, Improved detection of breast cancer nuclei using modular neural networks, *IEEE Engineer. Med. Biol. Mag.*, **19**(1): 48–63, 2000.
68. Y. Hashimoto, H. Murase, T. Morimoto, and T. Torii, Intelligent systems for agriculture in Japan, *IEEE Control Systems Mag.*, **21**(5): 71–85, 2001.
69. B. Widrow, D. E. Rumelhart, and M. A. Lehr, Neural networks: Applications in industry, business and science, *Communicat. ACM*, **37**(3): 93–105, 1994.
70. A. Vellido, P. I. G. Lisboa, and J. Vaughan, Neural networks in business: A survey of applications (1992–1998), *Expert Syst. Applicat.*, **17**: 51–70, 1999.
71. A.-P. N. Refenes, A. N. Burgess, and Y. Bentz, Neural networks in financial engineering: A study in methodology, *IEEE Trans. Neural Networks*, **8**(6): 1222–1267, 1997.
72. G. Zhang, B. E. Patuwo, and M. Y. Hu, Forecasting with artificial neural networks: The state of the art, *Internat. J. Forecasting*, **14**: 35–62, 1998.
73. M. Adya and F. Collopy, How effective are neural networks at forecasting and prediction? A review and evaluation, *J. Forecasting*, **17**(5–6): 481–495, 1998.
74. J. R. Dorronsoro, F. Ginel, C. Sánchez, and C. S. Cruz, Neural fraud detection in credit card operations, *IEEE Trans. Neural Networks*, **8**(4): 827–834, 1997.
75. L. C. Thomas, A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers, *Internat. J. Forecasting*, **16**: 149–172, 2000.
76. R. L. Wilson and R. Sharda, Bankruptcy prediction using neural networks, *Decision Support Sys.*, **11**(5): 545–557, 1994.
77. J. V. Hansen and R. D. Nelson, Neural networks and traditional time series methods: a synergistic combination in state economic forecasts, *IEEE Trans. Neural Networks*, **8**(4): 863–873, 1997.
78. H. White and J. Racine, Statistical inference, the bootstrap, and neural-network modeling with application to foreign exchange rates, *IEEE Trans. Neural Networks*, **12**(4): 657–673, 2001.
79. M. J. Shaw, C. Subramaniam, G. W. Tan, and M. E. Welge, Knowledge management and data mining for marketing, *Decision Support Systems*, **31**: 127–137, 2001.
80. S. Prabhu and D. Garg, Artificial neural network based robot control: An overview, *J. Intelli. Robotic Syst.*, **15**(4): 333–365, 1996.
81. S. Ananthraman and D. Garg, Training backpropagation and CMAC neural networks for control of a SCARA robot, *Engineering Applicat. Artificial Intelli.*, **6**(2): 105–115, 1993.
82. Y. H. Kim, F. L. Lewis, and D. M. Dawson, Intelligent optimal control of robotic manipulators using neural networks, *Automatica*, **36**(9): 1355–1364, 2000.
83. S. X. Yang and M. Q.-H. Meng, Real-time collision-free motion planning of a mobile robot using a neural dynamics-based approach, *IEEE Trans. Neural Networks*, **14**(6): 1541–1552, 2003.
84. K. Kiguchi, K. Watanabe, K. Izumi, and T. Fukuda, A human-like grasping force planner for object manipulation by robot manipulators, *Cybernetics and Syst.*, **34**(8): 645–662, 2003.
85. S. Ananthraman and D. Garg, Neurocontrol of cooperative dual robot manipulators, in *Intelligent Control Systems, ASME Special Publication*, No. DSC- **48**: 57–65, 1993.
86. S. S. Ge, L. Huang, and T. H. Lee, Model-based and neural-network-based adaptive control of two robotic arms manipulating an object with relative motion, *Intern. J. Systems Sci.*, **32**(1): 9–23, 2001.
87. A. Howard and H. Seraji, An intelligent terrain-based navigation system for planetary rovers, *IEEE Robotics and Automation Mag.*, **8**(4): 9–17, 2002.
88. F. L. Lewis, Neural network control of robot manipulators, *IEEE Intell. Sys.*, **11**(3): 64–75, 1996.
89. F. L. Lewis, A. Yesildirek, and K. Liu, Multilayer neural-net robot controller with guaranteed tracking performance, *IEEE Trans. Neural Networks*, **7**(2): 1–12, 1996.

90. Z. Hu and S. N. Balakrishnan, Online identification and control of aerospace vehicles using recurrent networks, *Proc. IEEE Conference on Control Applications*, 1999, pp. 160–165.

91. M.-C. Chen and T. Yang, Design of manufacturing systems by a hybrid approach with neural network metamodelling and stochastic local search, *Internat. J. Production Res.*, **40**(1): 71–92, 2002.

92. R. A. Kramer, B. Hoffner, and R. A. Shoureshi, Feedforward neural fuzzy control of electrical power systems containing highly varying loads, *Proc. American Control Conference*, **4**: 2677–2682, 2002.

93. G. Bloch, F. Sirou, V. Eustache, and P. Fatrez, Neural intelligent control for a steel plant, *IEEE Trans. Neural Networks*, **8**(4): 910–918, 1997.

94. M. H. R. FazlurRahman, R. Devanathan, and K. Zhu, Neural network approach for linearizing control of nonlinear process plants, *IEEE Trans. Industrial Electronics*, **47**(2): 470–477, 2000.

95. Yong-Zai Lu and S. W. Markward, Development and application of an integrated neural system for an HDCL, *IEEE Trans. Neural Networks*, **8**(6): 1328–1337, 1997.

DEVENDRA P. GARG
MANISH KUMAR
Duke University
Durham, North Carolina

N

NEURAL NETWORK ARCHITECTURES

INTRODUCTION

From the architectural and computational standpoints, a neural network consists of a collection of simple, nonlinear processing components (neurons) that are combined together via a collection of adjustable numeric connections. The development of a neural network is realized through learning. Selecting an appropriate network's structure (followed by an efficient learning method) becomes essential to its overall performance. In this article, we offer a comprehensive discussion on the architectures of neural networks. We discuss a variety of neurons and topologies encountered in the area and link the properties of the networks and their functionality with the architecture of the constructs. The article follows a bottom-up direction and reflects on historical developments of neurocomputing. We start with the commonly encountered model of the neuron (being regarded as a weighted sum of inputs followed by some nonlinear transformation) and present the main topologies of the networks. Next, we elaborate on more functionally advanced neurons and show how their functionality is exploited in the resulting structures of the networks and their increased heterogeneity (which is reflective of the use of a variety of neurons in their architectures). Throughout the article, a standard notation will be adhered to. Vectors will be denoted by boldface letters. The distance is denoted by $||\cdot||$.

COMPUTATIONAL MODEL OF NEURONS

A commonly encountered model of an artificial neuron (neuron, for short) (1–3) is realized in a form of an n-input single-output nonlinear mapping, see Fig. 1, described as follows:

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right) \tag{1}$$

where $x_1, x_2, \ldots, x_n$ are the inputs of the neuron, whereas w_1, $w_2, \ldots, w_n$ are the associated with the corresponding inputs connections (weights). The nonlinear nondecreasing mapping "f" brings an important component of nonlinear processing to the functionality of the neuron. Positive values of the weights correspond to excitatory synapses of the neuron, whereas negative weights model inhibitory synapses. The adjustable character of the connections makes the neuron (and the entire neural network) highly elastic and facilitates all parametric learning faculties (which in essence are concerned with the adjustments of the values of the connections). Commonly, the neuron is equipped with a bias term w_0 that comes with the constant input equal to 1. Taking it into consideration, Equation (1) is modified and reads as

$$y = f\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

Geometrically, the bias means that the hyperplane

$$\sum_{i=1}^{n} w_i x_i$$

is translated with respect to the origin. With the acceptance of the vector notation, $\mathbf{x} = [x_1\ x_2\ \ldots\ x_n\ 1]^T$ and $\mathbf{w} = [w_1\ w_2\ \ldots\ w_n\ w_0]^T$, the output of the neuron is expressed as $y = f(\mathbf{w}^T\mathbf{x})$

The nonlinear activation function (f) may be realized in different ways. Some commonly encountered examples include the following:

1. Threshold function $f(u) = \begin{cases} 1 \text{ if } u \geq 0 \\ 0 \text{ if } u < 0 \end{cases}$
2. Piecewise linear function $f(u) = \begin{cases} 1 \text{ if } u \geq 1/2 \\ u \text{ if} -1/2 < u < 1/2 \\ 0 \text{ if } u \leq -1/2 \end{cases}$
3. Sigmoid function $f(u) = \dfrac{1}{1 + \exp(-\lambda u)}$ with $\lambda(>0)$ being a positive slope of the function
4. Hyperbolic tangent function $f(u) = \tanh(u)$

In addition to the neuron governed by Equation (1), in the literature we also encounter so-called product neurons where the overall aggregation of the inputs is realized by taking the product of the weighted inputs,

$$y = \prod_{i=1}^{n} x_i^{w_i} \tag{2}$$

As before, w_i serves as a connection of the neuron whose values can be adjusted.

ARCHITECTURES OF NEURAL NETWORKS

Neurons are simple processing units. The processing capabilities of neural networks stem from the use of many neurons being organized in a certain topology (structure). Individual neurons are connected and realize a certain flow of computing. The most commonly present architecture of neural networks consists of neurons organized in a series of interconnected layers (3–5). Depending on the flow of processing, in the general taxonomy of the architectures of neural networks we distinguish between the two important categories, that is feedforward networks and recurrent (feedback) networks. In feedforward neural networks, the neurons are arranged in a series of layers, and a flow of processing is linear: It starts from the inputs of the network, and the results of processing are carried through consecutive layers finally showing at the output layer. An example of a single layer network is illustrated in Fig. 2, whereas Fig. 3 shows a three-layer network. In general, we may envision multilayer topologies, say, an L-layer neural network. In all cases, the structures are regular, and the connectivity occurs between the neurons positioned in successive layers.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

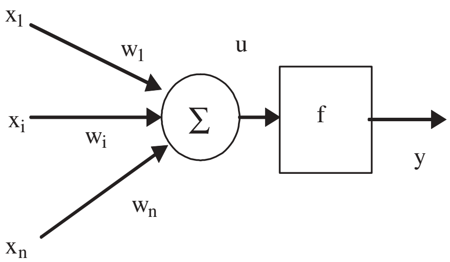


Figure 1. A topology of a neuron; note a two-phase processing of linear aggregation of the its inputs followed by nonlinear mapping (f).

In feedforward networks, input signals (inputs) are processed by the units of the first layer whose outputs becomes inputs for the next layer, and so on, for the rest of the network. Typically, the neurons of each layer have as their inputs the outputs that come from the preceding layer only. Feedforward neural networks produce static nonlinear input-output mappings. The form of the nonlinear mapping realized in this way depends on the number of layers, the size of the layers, and the form of the individual neurons. Intermediate layers between input nodes and output layer are referred to as hidden layers (the name comes from the fact that the neurons present there are not exposed directly to the input signals). We say that the neural network is fully connected if every node in each layer is connected to every other node in the consecutive forward layer. If this does not occur, then we refer to this network as partially connected. Some clarifying note is worth making here. In the terminology used in the literature, the input layer that distributes the inputs to the neurons located at the next layer (however does not realize any computing) is either included in the count of the number of layers or not. In this sense, if we count the layer that distributes the inputs, the network in Fig. 2 consists of 2 layers; likewise following this naming the network in Fig. 3 comes with 3 layers.

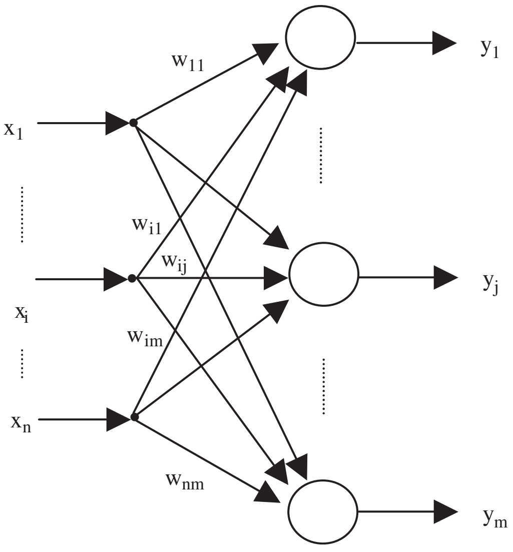


Figure 2. Single-layer feedforward neural network.

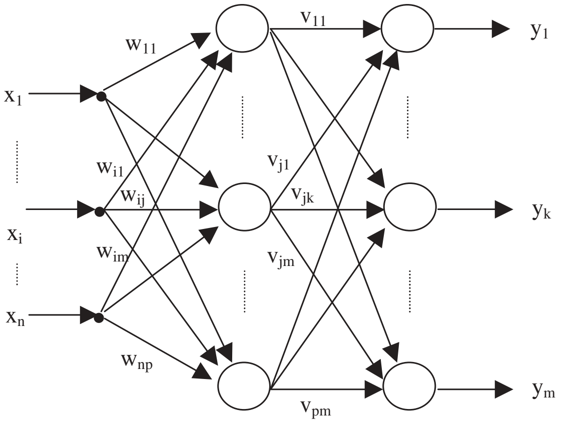


Figure 3. Two-layer feedforward neural network.

Recurrent neural networks (6–8) distinguish themselves from feedforward networks by admitting feedback loops, see Fig. 4. These networks may or may not have hidden layers. Feedback can be of *local* nature if only self-feedback loops exist, namely, if the outputs of neurons are fed back to its own input. Feedback loops of a *global* nature form if the loop engages different neurons located either in the same layer or in different layers. Recurrent neural networks can exhibit full or partial feedback, which depends on how the feedback loops have been structured.

In contrast to feedforward neural networks, neural networks with feedback are of interest when modeling a nonlinear dynamic input-output behavior. The feedback component is crucial to the realization of the dynamic nature of the phenomena.

NEURAL NETWORKS AS UNIVERSAL APPROXIMATORS

Neural networks implicitly encode in its structure a function that maps inputs on outputs. The character of functions that can be represented in this manner depends on the structure of the network. Currently, no definite result indicates which types of networks describe corresponding

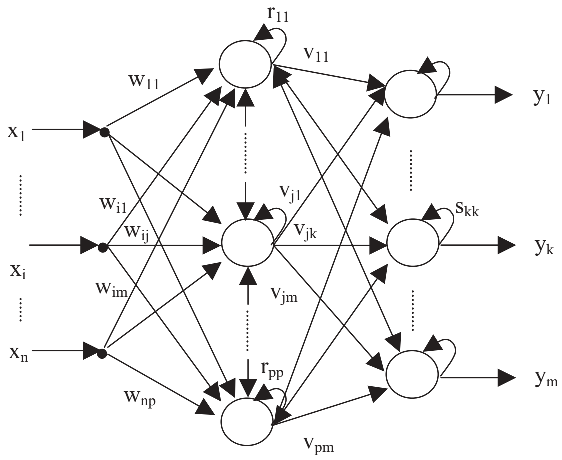


Figure 4. Recurrent neural network.

classes of functions. However, some general findings have been cited. The representation capabilities of neural networks are expressed in the form of a so-called theorem of universal approximation. This theorem states that a feedforward network with a single hidden layer (where the neurons in this layer are equipped with sigmoid type of transfer function) and an output layer composed of linear neurons (viz. with linear transfer functions) is a universal approximator (9–16). In other words, a neural network of such topology can approximate any given bounded continuous function $\mathbf{R}^n \rightarrow \mathbf{R}$ to any arbitrarily *small* approximation error.

In more detail, one formulation of the universal approximation capabilities of neural networks can be articulated as follows.

Given any continuous function "f" defined in the m-dimensional unit hypercube and any positive $\varepsilon > 0$, an integer "n" and real constants a_i, b_i, and w_{ij} exist such that the function

$$F(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \varphi\left(\sum_{j=1}^{m} w_{ij} x_j + b_i\right) \tag{3}$$

realizes an ε-approximation of "f" meaning that

$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon \tag{4}$$

for any $\mathbf{x}$ in this hypercube. φ is a nonconstant, bounded, and monotonically increasing function (we note that the sigmoid function satisfies such requirements).

The theorem of universal approximation of neural networks has to be put in a certain context. It is definitely an important and fundamental finding because it assures us about the representation capabilities of neural networks (viz. if the function satisfies the continuity assumption, we are confident that a neural network approximates it to any desired accuracy). This finding comes in the form of a typical existence theorem because it does not offer any constructive clue on how such a neural network could be constructed. For an interesting discussion of linkages between theoretical results and resulting algorithms refer to the literature (16).

So far, we have presented the most generic version of the neuron and discussed the underlying architectures of the network whose capabilities emerge through a collective processing realized by the neurons. Several generalizations of neurons augment their processing capabilities.

FUNCTIONAL NEURONS

Functional neurons are generic processing units in which we encounter some functional relationships $f_i(x_i, \mathbf{a}_i)$ instead of numeric values of the connections as schematically portrayed in Fig. 5.

The functional links f_i are equipped with some adjustable parameters $\mathbf{a}_i$ whose values can be modified (learned) during the learning process. Note that if we consider $f_i(x_i, \mathbf{a}_i)$ as constants, say a_i, then this construct collapses to the generic neurons discussed so far.

Polynomial neurons constitute an interesting class of functional neurons where f_i is a certain polynomial of the input variable. Typically, the order of this polynomial is kept low, say polynomials of the second order, that is, $f_i(x_i, \mathbf{a}_i) = a_{i0} + a_{i1}x_i + a_{i2}x_i^2$ This type of polynomial neurons yields polynomial neural networks, cf. Refs. 17–19. One can think of a few first harmonic terms that could be included in the functional link, say the terms such as $\sin(\omega x_{i1})$, $\sin(\omega x_{i2})$,... and so on. It is worth noting that the processing capabilities of this type of neuron are higher than the generic neurons, and these neurons exhibit nonlinear characteristics because of the processing realized at the level of each connection.

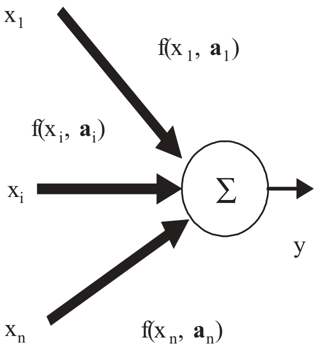


Figure 5. A schematic illustration of functional neuron.

LOGIC NEURONS

Neural networks are highly distributed topologies. The individual neurons do not exhibit any underlying interpretation. The category of fuzzy neurons (or fuzzy logic neurons) addresses these burning issues of a lack of transparency of neural networks. Our objective is to build a network with the aid of conceptually simple and logically appealing nodes (neurons) that complete generic *and* and *or* logic operations. By equipping the neurons with a set of connections, we furnish them with the badly required adaptive properties; the values of the connections could be adjusted easily by implementing some standard learning schemes available in neurocomputing.

Main Categories of Fuzzy Neurons

The logic aspect of transparent neurocomputing requires that the processing elements be endowed with the clearly delineated logic structure. We discuss several types of aggregative and referential neurons. Each neuron comes with a clearly defined semantics of its underlying logic expression and is equipped with significant parametric flexibility necessary to facilitate substantial learning abilities.

Aggregative Neurons. Formally, these neurons realize a logic mapping from $[0,1]^n$ to $[0,1]$. Two main classes of the processing units exist in this category (20–25).

The OR neuron realizes an *and* logic aggregation of inputs $\mathbf{x} = [x_1\, x_2 \ldots x_n]$ with the corresponding connections (weights) $\mathbf{w} = [w_1\ w_2 \ldots\ w_n]$ and then summarizes the partial results in an *or*-wise manner (hence the name of the neuron). In virtue of the logic flavor of processing, the values of the inputs **x** and connections **w** are confined to the unit hypercube $[0,1]^n$. The concise notation captures the logic character of computing, that is $y = \mathrm{OR}(\mathbf{x}; \mathbf{w})$, whereas the realization of the logic operations yields the expression as follows: (commonly referring to it as an s-t combination or s-t aggregation)

$$y = \underset{i=1}{\overset{n}{\mathbf{S}}}(x_i \, t w_i) \tag{5}$$

Bearing in mind the interpretation of the logic connectives (t-norms, t, and t-conorms, s), the OR neuron realizes the following logic expression being viewed as an underlying logic description of the processing of the input signals:

$$(x_1 \, \textit{and} \, w_1) \, \textit{or} \, (x_2 \, \textit{and} \, w_2) \textit{or} \ldots \textit{or} (x_n \, \textit{and} \, w_n) \tag{6}$$

Apparently, the inputs are logically "weighted" by the values of the connections before producing the final result. In other words, we can treat "y" as a truth value of the above statement where the truth values of the inputs are affected by the corresponding weights. Noticeably, lower values of w_i discount the impact of the corresponding inputs; higher values of the connections (especially those being positioned close to 1) do not affect the original truth values of the inputs that result in the logic formula. In limit, if all connections w_i, $i = 1, 2, \ldots, n$ are set to 1, then the neuron produces a plain *or*-combination of the inputs, $y = x_1$ *or* x_2 *or* ... *or* x_n. The values of the connections set to zero eliminate the corresponding inputs. Computationally, the OR neuron exhibits nonlinear characteristics (that is inherently implied by the use of the t- and t-conorms that are evidently nonlinear mappings). The plots of the characteristics of the OR neuron shown in Fig. 6 shows this effect (note that the characteristics are affected by the use of some triangular norms). The connections of the neuron contribute to its adaptive character; the changes in their values form the crux of the parametric learning.

The neurons in the AND neuron category, which are denoted by $y = \mathrm{AND}(\mathbf{x}; \mathbf{w})$ with **x** and **w** being defined as in case of the OR neuron, are governed by the expression

$$y = \underset{i=1}{\overset{n}{\mathrm{T}}}(x_i s w_i) \tag{7}$$

Here, the *or* and *and* connectives are used in a reversed order: First the inputs are combined with the use of the t-conorm, and the partial results produced in this way are aggregated *and*-wise. Higher values of the connections reduce impact of the corresponding inputs. In limit, $w_i = 1$ eliminates the relevance of x_i. With all w_i set to 0, the output of the AND neuron is just an *and* aggregation of the inputs

$$y = x_1 \, \textit{and} \, x_2 \, \textit{and} \ldots \textit{and} \, x_n \tag{8}$$

The characteristics of the AND neuron are shown in Fig. 7; note the influence of the connections and the specific realization of the triangular norms on the mapping completed by the neuron.

Let us conclude that the neurons are highly nonlinear processing units whose nonlinear mapping depends on the specific realizations of the logic connectives. They also come with potential plasticity whose usage becomes critical when learning the networks including such neurons.

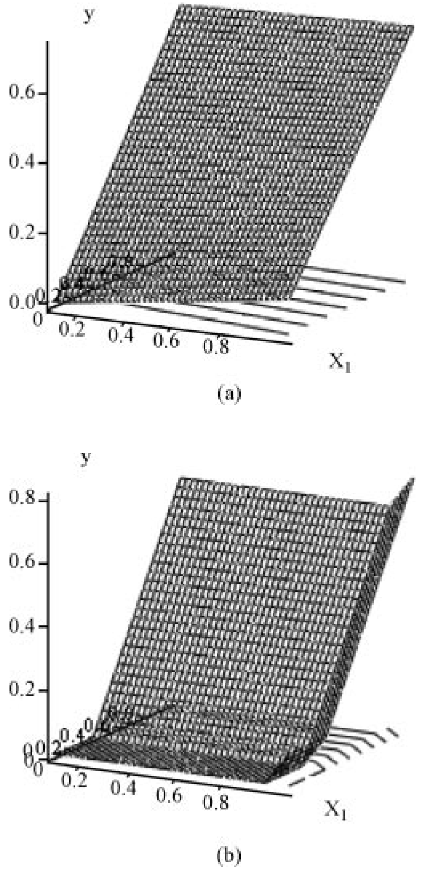


Figure 6. Characteristics of the OR neuron for selected pairs of t- and t-conorms. In all cases, the corresponding connections are set to 0.1 and 0.7 with intent to visualize their effect on the input-output characteristics of the neuron: (a) product and probabilistic sum, (b) Lukasiewicz *and* and *or* connectives.

The architecture of the logic neurons could be augmented by discussing a way in which bias could be introduced and how we can handle the inhibitory nature of some inputs.

Incorporation of the Bias Term (Bias) in the Fuzzy Logic Neurons. In analogy to the standard constructs of a generic neuron as presented above, we could also consider a bias term, denoted by $w_0 \in [0, 1]$, which enters the processing formula of the fuzzy neuron in the following manner:

for the OR neuron

$$y = \underset{i=1}{\overset{n}{\mathbf{S}}}(x_i t w_i) s w_0 \tag{9}$$

for the AND neuron

$$y = \underset{i=1}{\overset{n}{\mathbf{T}}}(x_i s w_i) t w_0 \tag{10}$$

We can offer some useful interpretation of the bias by treating it as some nonzero initial truth value associated

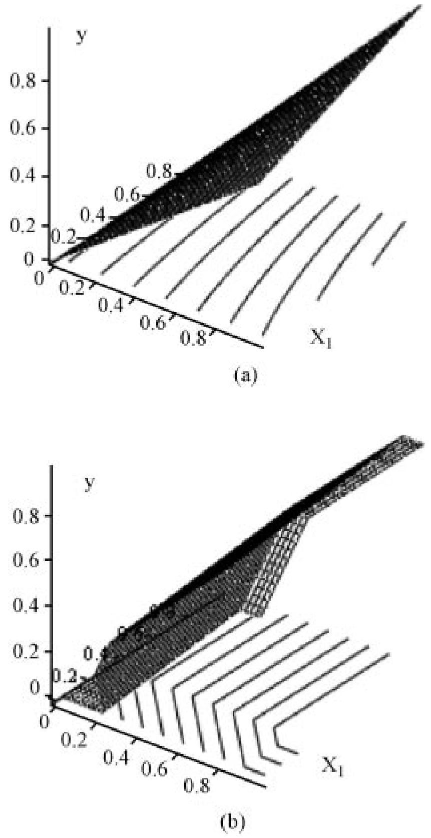


Figure 7. Characteristics of AND neurons for selected pairs of t- and t-conorms. In all cases, the connections are set to 0.l and 0.7 with intent to visualize their effect on the characteristics of the neuron: (a) product and probabilistic sum, (b) Lukasiewicz logic connectives.

with the logic expression of the neuron. For the OR neuron, it means that the output does not reach values lower than the assumed threshold. For the AND neuron equipped with some bias, we conclude that its output cannot exceed the value assumed by the bias. The question whether the bias is essential in the construct of the logic neurons cannot be fully answered in advance. Instead, we may include it into the structure of the neuron and carry out learning. Once its value has been obtained, the relevance of the bias can be established, considering the specific value it has been produced during the learning. It may be that the optimized value of the bias is close to zero for the OR neuron or close to one in the case of the AND neuron, which indicates that it could be eliminated without exhibiting any substantial impact on the performance of the neuron.

Dealing with Inhibitory Character of Input Information. Because of the monotonicity of the t-norms and t-conorms, the computing realized by the neurons exhibits an excitatory character. This finding means that higher values of the inputs (x_i) contribute to the increase in the values of the output of the neuron. The inhibitory nature of computing realized by "standard" neurons by using negative values of the connections or the inputs is not available here as the truth values (membership grades) in fuzzy sets are confined to the unit interval. The inhibitory nature of processing can be accomplished by considering the complement of the original input, that is $1\text{-}x_i$. Hence, when the values of x_i increase, the associated values of the complement decrease; subsequently in this configuration, we could effectively treat such an input as having an inhibitory nature.

Referential (reference) Neurons. The essence of referential computing deals with processing logic predicates. The two-argument (or generally multivariable) predicates such as *similar*, *included in*, and *dominates*(21), are essential components of any logic description of a system. In general, the truth value of the predicate is a degree of satisfaction of the expression P(x, a) where "a" is a certain reference value (reference point). Depending on the meaning of the predicate (P), the expression P(x, a) reads as "x is similar to a," "x is included in a," "x dominates a," and so on. In case of many variables, the compound predicate comes in the form $P(x_1, x_2, \ldots, x_n, a_1, a_2, \ldots, a_n)$ or more concisely $P(\mathbf{x}; \mathbf{a})$ where **x** and **a** are vectors in the *n*-dimensional unit hypercube. We envision the following realization of $P(\mathbf{x}; \mathbf{a})$

$$P(\mathbf{x}; \mathbf{a}) = P(x_1, a_1)\, \mathit{and}\, P(x_2, a_2)\, \mathit{and}\, \ldots \mathit{and}\, P(x_n, a_n) \quad (11)$$

which means that the satisfaction of the multivariable predicate relies on the satisfaction realized for each variable separately. As the variables could come with different level of relevance as to the overall satisfaction of the predicates, we represent this effect by some weights (connections) $w_1, w_2, \ldots, w_n$, so that Equation (11) can be expressed in the following form

$$\begin{aligned} P(\mathbf{x}; \mathbf{a}, \mathbf{w}) = {} & [P(x_1, a_1) \mathit{or}\, w_1]\, \mathit{and}\, [P(x_2, a_2)\ \mathit{or}\, w_2]\, \mathit{and} \ \ldots \\ & \mathit{and}\, [P(x_n, a_n)\, \mathit{or}\, w_n] \end{aligned} \quad (12)$$

Taking another look at the above expression and using a notation $z_i = P(x_i, a_i)$, it corresponds to a certain AND neuron $y = \text{AND}(\mathbf{z}; \mathbf{w})$ with the vector of inputs **z** being the result of the referential computations done for the logic predicate. Then, the general notation to be used reads as $\text{REF}(\mathbf{x}; \mathbf{w}, \mathbf{a})$. In the notation below, we explicitly articulate the role of the connections

$$y = \mathop{\mathbf{T}}_{i=1}^{n} (\text{REF}(x_i, a_i) s w_i) \quad (13)$$

In essence, as visualized in Fig. 8, we may conclude that the reference neuron is a realized as a two-stage construct where first we determine the truth values of the predicate (with **a** being treated as a reference point) and then treat these results as the inputs to the AND neuron.

So far, we have used the general term of predicate-based computing not confining ourselves to any specific nature of the predicate itself. Among several available possibilities of such predicates, we discuss the three of them, which tend to occupy an important place in logic processing. Those examples are inclusion, dominance, and match (similarity) predicates. As the names stipulate, the predicates return truth values of satisfaction of the relationship of inclusion, dominance, and similarity of a certain argument "x" with respect to the given reference "a." The essence of all these

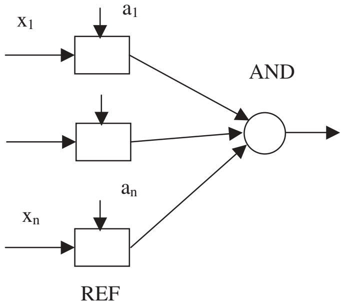


Figure 8. A schematic view of computing realized by a reference neuron and involving two processing phases (referential computing and aggregation).

calculations is in the determination of the given truth values, which is done in the carefully developed logic framework so that the operations retain their semantics and interpretability. What makes our discussion coherent is the fact that the proposed operations originate from triangular norms. The inclusion operation, denoted by ⊂, as discussed earlier, is modeled by an implication ⇒ that is induced by a certain left continuous t-norm (26)

$$a \Rightarrow b = \sup\{c \in [0,1] | atc \leq b\}, a, b \in [0,1] \tag{14}$$

For instance, for the product the inclusion takes on the form $a \Rightarrow b = \min(1, b/a)$. The intuitive form of this predicate is self-evident: The statement "x is included in a" and modeled as INCL(x, a) = x ⇒ a comes with the truth value equal to 1 if x is less or equal to a (which in other words means that x is included in a) and produces lower truth values once x starts exceeding the truth values of "a." Higher values of "x" (those above the values of the reference point "a") start generating lower truth values of the predicate. The dominance predicate acts in a dual manner when compared with the predicate of inclusion. It returns 1 once "x" dominates "a" (so that its values exceeds "a") and values below 1 for x lower than the given threshold. The formal model can be realized as DOM(x, a) = a ⇒ x. With regard to the reference neuron, the notation is equivalent to the one being used in the previous case, that is DOM(**x**; **w**, **a**) with the same meaning of **a** and **w**.

The similarity (match) operation is an aggregate of these two, SIM(x,a) = INCL(x,a) t DOM(x,a), which is appealing from the intuitive standpoint: We say that x is similar to a if x is included in a *and* x dominates a. Noticeably, if x = a the predicate returns 1; if x moves apart from "a," then the truth value of the predicate becomes reduced. The resulting similarity neuron is denoted by SIM(**x**; **w**, **a**) and reads as follows:

$$y = \underset{i=1}{\overset{n}{\mathbf{T}}}(\mathrm{SIM}(x_i, a_i) s w_i) \tag{15}$$

It is worth noting that by moving the reference point to the origin or the **1**-vertex of the unit hypercube (with all its coordinates being set up to 1), the referential neuron starts to resemble the aggregative neuron. In particular, we have

- for $\mathbf{a} = \mathbf{1} = [1\,1\,1\ldots 1]$ the inclusion neuron reduces to the AND neuron
- for $\mathbf{a} = \mathbf{0} = [0\,0\,0\ldots 0]$ the dominance neuron reduces to the AND neuron

One can draw a loose analogy between some types of the referential neurons and the two categories of processing units encountered in neurocomputing. The analogy is based on the *local* versus *global* character of processing realized therein. Perceptrons come with the global character of processing. Radial basis functions realize a local character of processing as focused on receptive fields. In the same vein, the inclusion and dominance neurons are after the global nature of processing, whereas the similarity neuron carries more confined and local processing.

An interesting taxonomy of referential neurons is observed. We distinguish between two main categories of these processing elements. The first group is formed by homogeneous neurons viz. those in which encounter the same type of logic predicate (P) used in the underlying processing. Refer also to Equation (11). As shown above, the underlying logic expression reads as follows:

$$y = (P(x_1, r_1), w_1) \text{ and} (P(x_2, r_2), w_2) \text{ and} \ldots \text{and} (P(x_n, r_n), w_n) \tag{16}$$

where r_i and w_i are the point of reference and the corresponding weight associated with the *i*th variable. Thus, all inputs are processed making use of the same predicate, which could be inclusion, dominance, tolerance, similarity, and so on.

In the second group of referential neurons, we admit a higher level of diversity by allowing different predicates P_i associated with the individual inputs. The general logic expression comes in the following format:

$$y = (P_1(x_1, r_1), w_1) \text{ and} (P_2(x_2, r_2), w_2) \text{ and} \ldots \text{and} (P_n(x_n, r_n), w_n) \tag{17}$$

Referential computing is reflective of processing logic constraints conveyed by logic predicates when dealing with existing domain knowledge and encapsulating it in the structural format of the network or its part.

For instance, we can represent existing constraints for the variables $x_1, x_2, \ldots, x_n$, which are spelled out as

$$x_1 \text{ should not exceed } g_1 \text{ and } x_2 \text{ should not exceed } g_2 \text{ and} \ldots$$
$$x_n \text{ should not exceed } g_n \tag{18}$$

In the form of a single inclusion neuron, Fig. 9(a) demonstrates where the weights are used to calibrate the relevance (importance) of the corresponding constraint. Note that both x_i and g_i standing in the above compound predicate assume values in the unit interval. One might capture a variety of constraints

$$x_1 \text{ should not exceed } g_1 \text{ and } x_2 \text{ should be similar to } g_2 \text{ and} \ldots$$
$$x_n \text{ should dominate } g_n \tag{19}$$

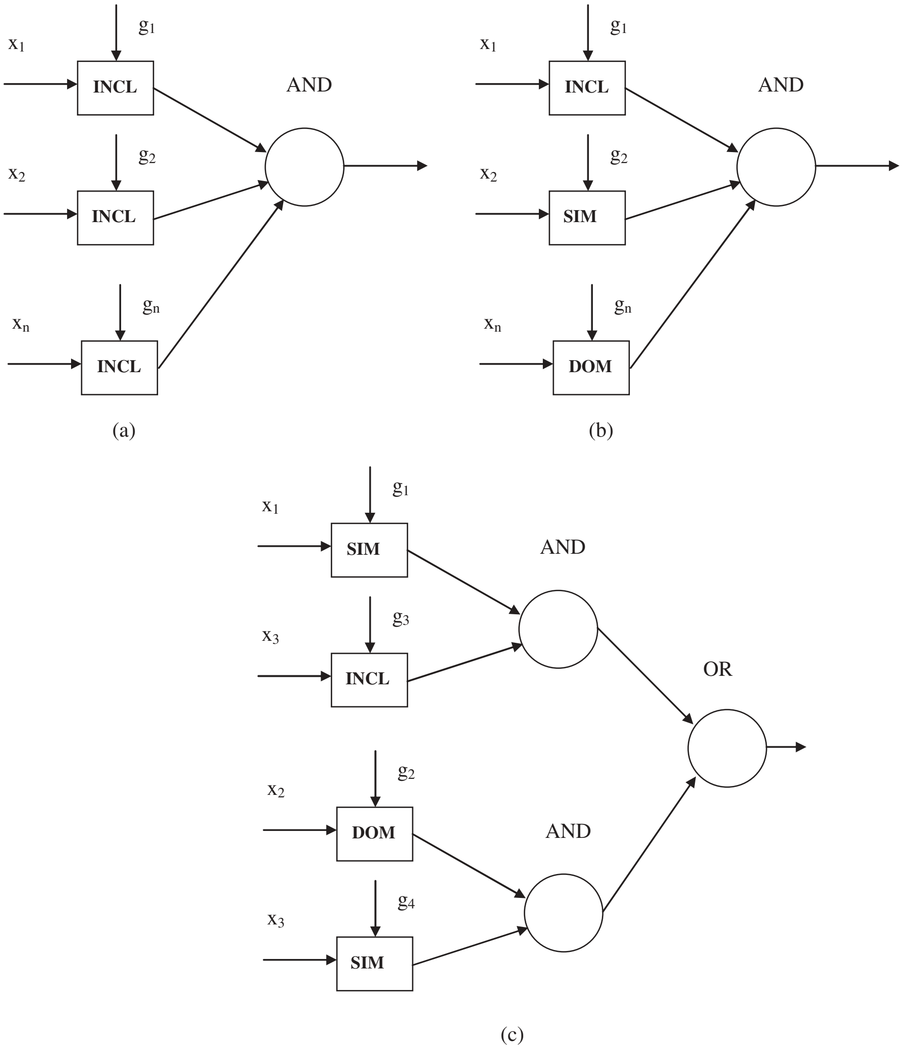


Figure 9. Examples of logic expressions that involve predicates and their realization in the form of logic neurons and architectures of logic-oriented networks.

By considering various predicates; see Fig. 9(b). Interestingly, one could consider a variety of logic constraints of different dimensionality, that is, each referential neuron might have different number of inputs depending on the nature of locally identified constraints. For instance, given three inputs x_1, x_2, and x_3, we have

$$\begin{array}{l}x_1 \text{ should be similar to } g_1 \text{ and } x_3 \text{ should not exceed } g_3 \\ \text{or} \\ x_2 \text{ should dominate } g_2 \text{ and } x_3 \text{ should be similar to } g_4\end{array} \tag{20}$$

This logic description translates into the logic network illustrated in Fig. 9(c).

One could stress that in several architectures of neuro-fuzzy systems, the capabilities of neural networks are combined with the technology of fuzzy sets, cf. Refs. 26 and 27.

GRANULAR NEURONS

A certain interesting generalization of the generic topology of the neuron comes with a realization of its connections as some information granules. Instead of a single numeric value of the connection discussed so far, we admit that it can be represented as some information granule, in particular some interval or a fuzzy set. As the name suggests, by the granular neuron we mean a neuron with granular connection. More precisely, we consider the transformation of many numeric inputs $u_1, u_2, \ldots, u_c$ (confined to the unit interval) of the form

$$Y = N(u_1, u_2, \ldots, u_c, W_1, W_2, \ldots, W_c) = \sum_{\oplus}(W_i \otimes u_i) \tag{21}$$

with W_1, W_2, ... W_c denoting granular weights (connections), see Fig. 10. The symbols of generalized (granular) addition and multiplication (that is $\oplus$, $\otimes$) are used here to emphasize a granular character of the arguments being used in this aggregation. When dealing with interval-valued connections, $W_i = [w_{i-}, w_{i+}]$, the operations of their multiplication by some positive real input u_i produce the results in the form of the following interval

$$W_i \otimes u_i = [w_{i-}u_i, w_{i+}u_i] \tag{22}$$

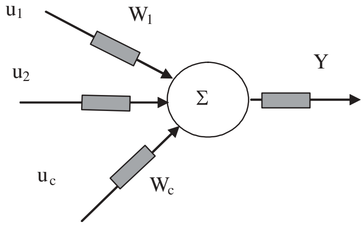


Figure 10. Computational model of a granular neuron; note a granular character of the connections and the resulting output Y.

When adding such intervals being produced at the level of each input of the neuron, we arrive at the expression

$$Y = \left[\sum_{i=1}^{n} w_{i-}u_i, \sum_{i=1}^{n} w_{i+}u_i\right] \tag{23}$$

For the connections represented as fuzzy sets, the result of their multiplication by a positive scalar u_i is realized through the use of the extension principle

$$(W_i \otimes u_i)(y) = \sup_{w:y=wu_i}[W_i(w)] = W_i(y/u_i) \tag{24}$$

Next, the extension principle is used to complete additions of fuzzy numbers, which are the partial results of this processing. Denote by Z_i the fuzzy number $Z_i = W_i \otimes u_i$. We obtain

$$Y = Z_1 \oplus Z_2 \oplus \ldots \oplus Z_n \tag{25}$$

that is

$$Y(y) = \sup\{\min(Z_1(y_1), Z_2(y_2), \ldots, Z_n(y_n))\}$$

$$\text{s.t. } y = y_1 + y_2 + \ldots + y_n \tag{26}$$

Depending on a specific realization, these connections can be formalized as intervals, fuzzy sets, shadowed sets, rough sets, and so on. One could note that despite potential diversity of the formalisms of granular connections, the output is always a granular construct, Fig. 10.

The granular neuron exhibits several interesting properties that generalize the characteristics of (numeric) neurons. Adding a nonlinearity component (g) to the linear aggregation does not change the essence of computing; in case of monotonically increasing relationship (g(Y)), we end up with a transformation of the original output interval or fuzzy set (in this case we have to follow the calculations using the well-known extension principle).

RADIAL BASIS FUNCTION NEURAL NETWORKS

Radial basis function (RBF) neural networks (NNs), are examples of feedforward neural structures that combine a weighted collection of receptive fields defined in the input space for function approximation and classification. The RBFs known as receptive fields are constructs that help the network to focus on individual regions of the multidimensional input space. In the sequel, the activation levels of the RBFs implied by some input x are aggregated by a single linear neuron located in the output layer of the network. The general structure of the network is visualized in Fig. 11.

From the functional standpoint, the activation levels of the receptive fields (RBFs), $z_j = R(\mathbf{x})$ are combined in a linear form as

$$y = \sum_{i=1}^{c} w_j z_j$$

where w_j is the *j*th connection of the neuron. RBFs can assume different forms. Commonly, they are treated as Gaussian receptive fields where $R(\mathbf{x}) = \exp(-||\mathbf{x}-\mathbf{v}_j||^2/\sigma_j^2)$ where $\mathbf{v}_j$ denotes a center of the field, and σ_j describes a spread of the *j*th RBF. As it becomes visible from all these examples of the RBFs, they come with a great deal of flexibility. In particular, they are distributed in the input space by selecting their modal values (centers of the receptive fields) and choosing the spread values. This substantial level of flexibility is of interest when learning the network. Instead of defining a certain class of the receptive fields, there is an alternative way of their formation that directly links to experimental data and exploits the techniques of fuzzy clustering. Let us recall that in fuzzy clustering, data are organized in groups in such a way that we allow data to share nonzero membership grades between several clusters. The commonly used method of Fuzzy C-Means (26) leads to the development of the prototypes of the clusters. The prototypes serve as modal values of the receptive fields, whereas the membership functions are described in the following form:

$$R_i(\mathbf{x}) = \frac{1}{\sum_{j=1}^{c}\left(\frac{||\mathbf{x}-\mathbf{v}_i||}{||\mathbf{x}-\mathbf{v}_j||}\right)^{2/(m-1)}} \tag{27}$$

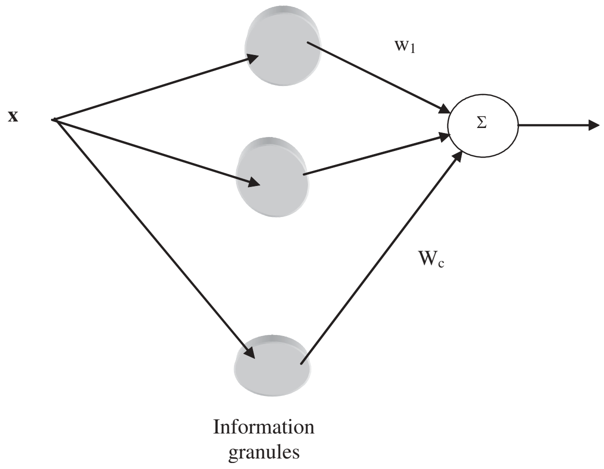


Figure 11. A generic topology of RBF neural network.

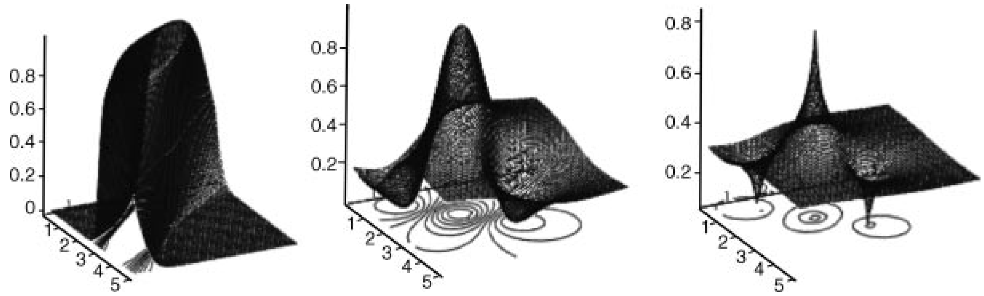


Figure 12. Examples of receptive fields formed by the FCM algorithm: (a) m = 1.2, (b) m = 2.0, (c) m = 3.0.

Where $m > 1$ is a so-called fuzzification coefficient. Interestingly, the location of the fields (as well as their spreads—which do not explicitly show in the above) has been determined on a basis of the available learning data. In this sense, these receptive fields are immediately reflective of the experimental evidence available for training purposes. It is worth stressing that the geometry of the receptive fields obtained in this manner is affected by the values of the fuzzification coefficient. Some illustrative examples are shown in Fig. 12.

The optimal value of "m" can be determined through data clustering. Note that even the neuron in the output layer realizes a linear processing; overall the network is highly nonlinear and the source of nonlinearity is associated with the highly nonlinear character of the RBFs.

We can envision a certain generalization of RBF NNs in which the linear neuron is replaced by a functional neuron or a granular neuron. The topologies of such neural networks fall under the category of neurofuzzy systems that combine some features of neural networks and fuzzy sets.

For the functional neuron, the resulting network can be interpreted as a collection of "if-then" rules

$$\text{- if } x \text{ is } R_i \text{ then } y \text{ is } f_i(\mathbf{x}, \mathbf{a}_i),\ i = 1, 2, \ldots, c \tag{28}$$

where R_i is the *i*th receptive field that serves as some information granule, say some fuzzy set.

In case of the granular neuron, the collection of rules reads as follows

$$\text{- if } x \text{ is } R_i \text{ then } Y \text{ is } W_i,\ \ i = 1, 2, \ldots, c \tag{29}$$

where W_i is the granular connection of the neuron; note that the output is an information granule as well.

The concept of receptive fields is highly appealing. As a matter of fact, in the layer of receptive fields, one could consider using wavelets, local expert networks as discussed in Ref. 28, or kernels (29).

ARCHITECTURES OF LOGIC NETWORKS

The logic neurons (aggregative and referential) can serve as building blocks of more comprehensive and functionally appealing architectures. The diversity of the topologies one can construct with the aid of the proposed neurons is surprisingly high. This architectural multiplicity is important from the application point of view as we can fully reflect the nature of the problem in a flexible manner. It is essential to capture the problem in a logic format and then to set up the logic skeleton also known as the conceptual blueprint—by forming it and finally refining it parametrically through a thorough optimization of the connections. Throughout the entire development process, we are positioned comfortably by monitoring the optimization of the network as well as interpreting its meaning.

The typical logic network that is at the center of logic processing originates from the two-valued logic and comes in the form of the famous Shannon theorem of decomposition of Boolean functions. Let us recall that any Boolean function $\{0,1\}^n \rightarrow \{0,1\}$ can be represented as a logic sum of its corresponding miniterms or a logic product of maxterms. By a minterm of "n" logic variables $x_1, x_2, \ldots, x_n$, we mean a logic product that involves all these variables either in direct or complemented form. Having "n" variables, we end up with 2^n minterms starting from the one that involves all complemented variables and ending up at the logic product with all direct variables. Likewise, by a maxterm we mean a logic sum of all variables or their complements. Now in virtue of the decomposition theorem, we note that the first representation scheme involves a two-layer network, in which the first layer consists of AND gates whose outputs are combined in a single OR gate. The converse topology occurs for the second decomposition mode: A single layer of OR gates is followed by a single AND gate aggregating *or*-wise all partial results.

The proposed network (referred here as a logic processor) generalizes this concept as shown in Fig. 13. The OR-AND mode of the logic processor comes with the two types of aggregative neurons being swapped between the layers. Here, the first (hidden) layer is composed of the OR neuron and is followed by the output realized by means of the AND neuron.

The logic neurons generalize digital gates. The design of the network (viz. any fuzzy function) is realized through learning. If we confine ourselves to {0,1} values, then the network's learning becomes an alternative to a standard digital design, especially a minimization of logic functions. The logic processor translates into a compound logic statement (we skip the connections of the neurons to underline the underlying logic content of the statement)

$$\text{- if } (\text{input}_l\ and \ldots\ and\ \text{input}_j)\ or (\text{input}_d\ and\ \ldots and\ \text{input}_f)$$
$$\text{then}\ output$$

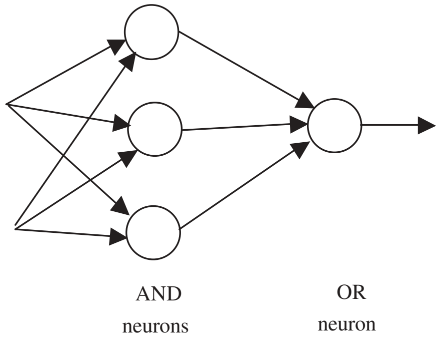
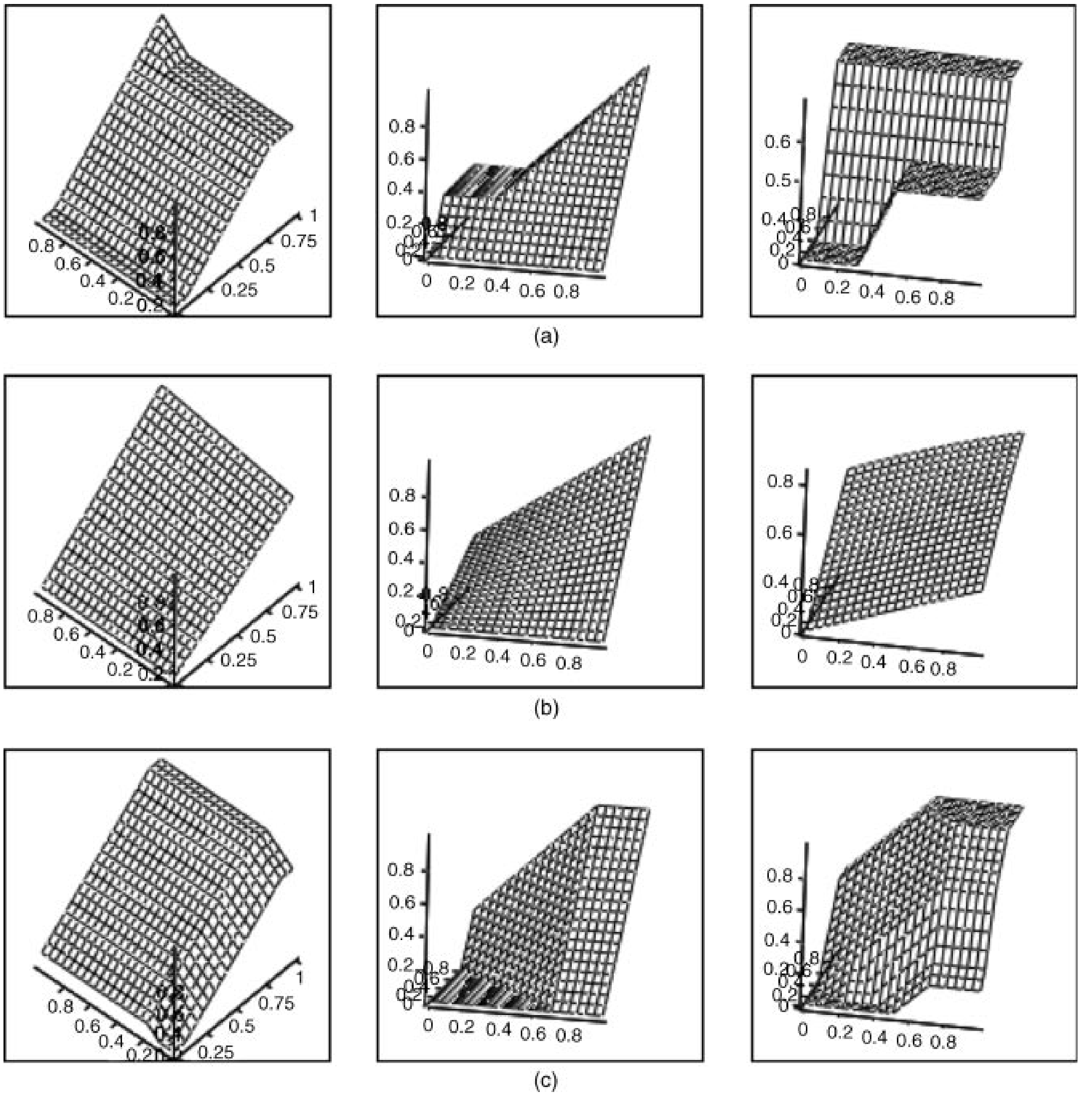


Figure 13. A topology of the logic processor in its AND-OR mode.

The logic processor's topology (and underlying interpretation) is standard. Two LPs can vary in terms of the number of AND neurons as well as their connections, but the format of the resulting logic expression is uniform (as a sum of generalized minterms).

As an illustrative example, let us consider a simple fuzzy neural network in which the hidden layer includes two AND neurons whose outputs are combined through a single OR neuron located in the output layer. The connections of the first AND neuron are equal to 0.3 and 0.7. For the second AND neuron, we have the values of the connections equal to 0.8 and 0.2. The connections of the OR neuron are equal to 0.5 and 0.7, respectively. The input-output characteristics of the network are illustrated in Fig. 14; to demonstrate the flexibility of the architecture, we included several combinations of the connections as well as used alternative realizations of the triangular norms and conorms.

The resulting networks exhibit a significant diversity in terms of the resulting nonlinear dependencies. More importantly, we note that by choosing certain logic connectives (triangular norms) and adjusting the values of the connections, we could substantially affect the behavior (input-output characteristics) of the corresponding network. This plasticity becomes an important feature that plays a paramount role in the overall learning process.



(a)

(b)

(c)

Figure 14. Plots of the characteristics of the fuzzy neural network: output of the two AND neurons and the output of the network (from left to right) for different realization of the logic operators: (a) min and max, (b) product and probabilistic sum, and (c) Lukasiewicz logic operators.

CONCLUDING COMMENTS

We have covered main issues of architectural developments of neural networks and stressed their importance for the resulting quality of the resulting networks. A significant diversity of neurons is available, which come with different levels of computational sophistication and characteristics that could be taken advantage of when dealing with the problem at hand. Although a "standard" neuron is still dominant in neurocomputing, we witness interesting trends of expanding its functionalities, which manifests in granular, functional or logic-oriented neurons.

In the development of the architectures of neural networks and selecting specific neurons, one can take into account some general guidelines:

1. The static or dynamic nature of the problem at hand: This aspect determines whether we should consider networks with feedback or whether a standard feedforward neural network is suitable.
2. The size of the network is expressed in terms of the number of the neurons as well as an layout of the network itself (which links to the number of its layers). Large networks could lead to low values of the approximation error on the training data and could eventually reduce it to values close to zero. The performance of the network might suffer from that when it comes to generalization capabilities of the network (assessed on a basis of the testing data). The learning time also could be excessively long.
3. The choice of the individual neurons. More advanced neurons may lead to a more compact network (as the underlying functionality of the processing elements is more extended). They may, however, require more advanced learning mechanisms whose complexity and a level of sophistication could be high. The choice associates with the available optimization tools.
4. When choosing the architecture of the network, one has to have in mind what learning mechanisms will be considered afterward so that we could take full advantage of the functionality of the existing architecture. In particular, a decision has to be made with regard to supervised-unsupervised learning as well as structural or parametric optimization of the network.
5. If the interpretability of the network is a desired feature, then the use of neurons whose semantics is clear is advisable.

Despite current advances in the learning theory (1,2,6), the applications of neural networks still require careful experimentation and a prudent use of engineering judgment to make their design effective. The development of the networks is an iterative process and requires successive architectural refinements.

BIBLIOGRAPHY

1. M. Anthony and P. L. Bartlet, *Neural Network Learning: Theoretical Foundations*, Cambridge: Cambridge University Press, 1999.
2. S. Ellacott and D. Bose, *Neural Networks: Deterministic Methods of Analysis*, London: Thomson Computer Press, 1996.
3. S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 1998.
4. D. Hush and B. Horne, Progress in supervised neural networks: What's new after Lippmann?, *IEEE Signal Processing Magazine*, **10**(1): 8–39, 1993.
5. R. Morejon and J. Principe, Advanced search algorithms for information-theoretic learning with kernel-based estimators, *IEEE Trans. Neural Networks*, **15**(4): 874–884, 2004.
6. A. Atiya and A. Parlos, New results on recurrent network training: unifying the algorithms and accelerating convergence, *IEEE Trans. Neural Networks*, **11**(3): 697–709, 2000.
7. L. Jin, M. Gupta, and P. Nikiforuk, Approximation capabilities of feedforward and recurrent neural networks, in M. Gupta and N. Sinha (eds.), *Intelligent Control Systems: Theory and Applications*, Piscataway, NJ: IEEE Press, 1996, pp. 234–264.
8. R. Williams and D. Zipser, A learning algorithm for continually running fully recurrent neural networks, *Neural Comput.*, **1**, 270–280, 1989.
9. P. Baldi, Computing with arrays of bell-shaped and sigmoid functions, in R. Lippmann, J. Moody, D. Touretzky (eds.), *Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufmann, 1991, pp. 735–742.
10. P. Courrieu, Function approximation in non-Euclidean spaces, *Neural Networks*, **18**: 91–102, 2005.
11. G. Cybenko, Approximation by superposition of a sigmoidal function, *Math. Control, Signals, and Systems*, **2**: 303–314, 1989.
12. M. Hassoun, *Fundamentals of Artificial Neural Networks*, Cambridge, MA: MIT Press, 1995.
13. K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, **2**: 359–366, 1989.
14. K. Hornik, Some new results on neural network approximation, *Neural Networks*, **6**: 1069–1071, 1993.
15. M. Leshno, Y. Lin, A. Pinkus, and S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, *Neural Networks*, **6**: 861–867, 1993.
16. F. Scarselli and A. Tsoi, Universal approximation using feedforward neural networks: A survey of some existing results and some new results, *Neural Networks*, **11**: 15–37, 1998.
17. E. Gómez-Ramírez, K. Najim, and E. Ikonen, Forecasting time series with a new architecture for polynomial artificial neural network, *Appl. Soft Comput.*, **7**(4): 1209–1216, 2007.
18. S. K. Oh and W. Pedrycz, Multi-layer self-organizing polynomial neural networks and their development with the use of genetic algorithms, *J. Franklin Inst.*, **343**(2): 125–136, 2006.
19. H. S. Park, W. Pedrycz, and S. K. Oh, Evolutionary design of hybrid self-organizing fuzzy polynomial neural networks with the aid of information granulation *Exp. Syst. Applicat.*, **33**(4): 830–846, 2007.
20. W. Pedrycz, Processing in relational structures: Fuzzy relational equations, *Fuzzy Sets Syst.*, **40**: 77–106, 1991.
21. W. Pedrycz, Neurocomputations in relational systems, *IEEE Trans. Pattern Anal. Mach. Intell.*, **13**: 289–297, 1991.
22. W. Pedrycz, Fuzzy neural networks and neurocomputations, *Fuzzy Sets Syst.*, **56**: 1–28, 1993.

23. W. Pedrycz and A. Rocha, Knowledge-based neural networks, *IEEE Trans. Fuzzy Syst.*, **1**: 254–266, 1993.

24. W. Pedrycz, Heterogeneous fuzzy logic networks: fundamentals and development studies, *IEEE Trans. Neural Networks*, **15**: 1466–1481, 2004.

25. W. Pedrycz and M. Reformat, Genetically optimized logic models, *Fuzzy Sets Syst.*, **150**(2): 351–371, 2005.

26. W. Pedrycz and F. Gomide, *An Introduction to Fuzzy Sets: Analysis and Design*, Cambridge, MA: MIT Press, 1998.

27. C. F. Juang and I.-F. Chung, Recurrent fuzzy network design using hybrid evolutionary learning algorithms, *Neurocomputing*, **70**(16–18): 3001–3010, 2007.

28. E. D. Übeyli, Wavelet/mixture of experts network structure for EEG signals classification, *Expert Syst. Applicat.*, **34**(3): 1954–1962, 2008.

29. K. Müller, S. Mika, R. Rätsch, K. Tsuda, and B. Schölkopf, An introduction to kernel-based learning algorithms, *IEEE Trans. Neural Networks*, **12**(2): 181–201, 2001.

Witold Pedrycz
University of Alberta
Edmonton, Alberta,
Canada

PATTERN RECOGNITION

Pattern recognition (PR) concerns the description or classification (recognition) of measurements. PR capability is often a prerequisite for intelligent behavior. PR is not one technique, but rather a broad body of often loosely related knowledge and techniques. PR may be characterized as an *information reduction, information mapping*, or *information labeling* process. Historically, the two major approaches to pattern recognition are statistical (or decision theoretic), hereafter denoted StatPR, and syntactic (or structural), hereafter denoted SyntPR. The technology of artificial neural networks has provided another alternative, neural pattern recognition, hereafter denoted NeurPR. NeurPR is especially well suited for "black box" implementation of PR algorithms. As no single technology is always the optimal solution for a given PR problem, all three are often considered in the quest for a solution.

The structure of a generic PR system is shown in Fig. 1 (1). Notice that it consists of a sensor or set of sensors, a feature extraction mechanism (algorithm), and a classification or description algorithm (depending on the approach). In addition, usually some data that has already been classified or described is assumed available in order to train the system (the so-called "training set").

PATTERNS AND FEATURES

PR, naturally, is based on *patterns*. A pattern can be as basic as a set of measurements or observations, perhaps represented in vector notation. *Features* are any extracted measurement used. Examples of low-level features are signal intensities. Features may be symbolic, numeric, or both. An example of a symbolic feature is color; an example of a numerical feature is weight (measured in pounds). Features may also result from applying a *feature extraction algorithm or operator* to the input data. Additionally, features may be higher-level entities, for example, geometric descriptors of either an image region or a three-dimensional (3-D) object appearing in the image. For example, in *image analysis* applications (2), aspect ratio and Euler number are higher-level geometric features extracted from image regions. Recently, there has been a renewal of interest in employing biometric features based on face, voice, fingerprint, or eye measurements.

Significant computational effort may be required in feature extraction and the extracted features may contain errors or "noise." Features may be represented by continuous, discrete, or discrete-binary variables. *Binary features* may be used to represent the presence or absence of a particular attribute. The inter-related problems of *feature selection* and *feature extraction* must be addressed at the outset of any PR system design.

Statistical PR is explored in depth in numerous books. Good sources include Refs. 1, 3–10.

The Feature Vector and Feature Space

Feature vectors are typically used in StatPR and NeurPR. It is often useful to develop a geometrical viewpoint of features in these cases. Features are arranged in a d-dimensional *feature vector*, denoted $\underline{x}$, which yields a multidimensional *feature space*. If each feature is an unconstrained real number, the feature space is R^d. In other cases, for example, those involving artificial neural networks, it is convenient to restrict feature space to a subspace of R^d. Specifically, if individual neuron outputs and network inputs are restricted to the range [0, 1], for a d-dimensional feature vector, the feature space is a unit volume hypercube in R^d.

Classification of feature vectors may be accomplished by partitioning feature space into regions for each class. Large feature vector dimensionality often occurs unless the data is preprocessed. For example, in *image processing* applications, it is impractical to directly use all the pixel intensities in an image as a feature vector because a 512 × 512-pixel image yields a 262,144 × 1 feature vector.

Feature vectors are somewhat inadequate or at least cumbersome when it is necessary to represent relations between pattern components. Often, classification, recognition, or description of a pattern is desired that is invariant to some (known) pattern changes or deviation from the "ideal" case. These deviations may be because of a variety of causes, including "noise."

In many cases, a set of patterns *from the same class* may exhibit wide variations from a single exemplar of the class. For example, humans are able to recognize (that is, classify) printed or handwritten characters with widely varying font sizes and orientations. Although the exact mechanism that facilitates this capability is unknown, it appears that the matching strongly involves structural analysis of each character.

Feature Vector Overlap. As feature vectors obtained from exemplars of two different classes may overlap in feature space, classification errors occur. An example of this overlap is shown in Fig. 2.

Example of Feature Extraction. Consider the design of a system to identify two types of machine parts. One part, which is denoted a "shim," is typically dark and has no surface intensity variation or "texture." Another part, denoted a "machine bolt," is predominantly bright and has considerable surface intensity variation. For illustration, only texture and brightness are used as features, thus yielding a 2-D *feature space* and *feature vector*. We also assume these features are extracted from suitable measurements. Other possible features, such as shape, weight, and so on, may be used. The problem, as formulated, is challenging because these features are only *typical* of each part type. There exist cases of shims that are bright and textured and bolts that are dark and have little texture, although they are *atypical*, that is, they donot occur often.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

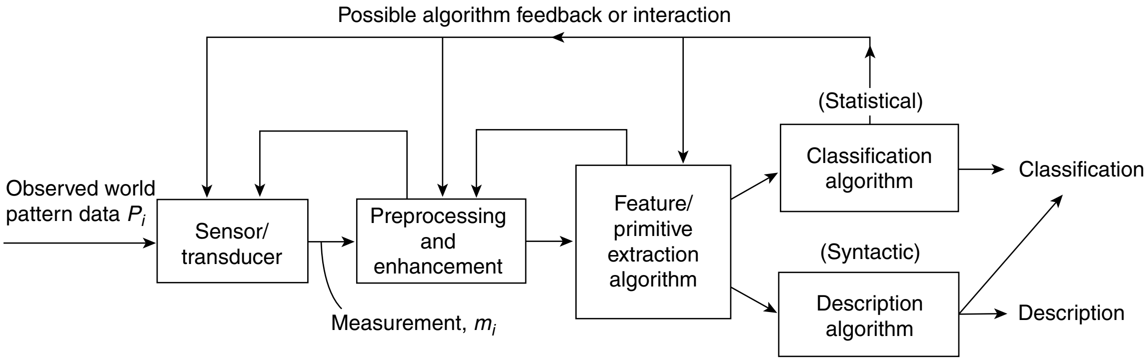

Possible algorithm feedback or interaction
Observed world pattern data P_i
Sensor/ transducer
Measurement, m_i
Preprocessing and enhancement
Feature/ primitive extraction algorithm
(Statistical)
Classification algorithm
Classification
(Syntactic)
Description algorithm
Description

Figure 1. Generic PR system elements (Adapted from Ref. 1).

More importantly, when features overlap, perfect classification is not possible. Therefore, classification error, characterized via the probability *P(error)*, indicates the likelihood an incorrect classification or decision. In this example, element $x_i, i = 1, 2$ is *a feature*, where x_1 is measured or computed brightness and x_2 is measured or computed texture. Furthermore, w_i is a *class*, or a "state of nature," where w_1 is taken to be shim and w_2 is bolt. Feature vector overlap may occur in this example. If the underlying class is w_1 (shims), we expect typical measurements of x_1 and x_2 (brightness and texture, respectively) to be small, whereas if the object under observation is from class w_2 (bolts), we expect the values of x_1 and x_2 to be, on the average, large (or at least larger than those of w_1). Of particular importance is the region where values of the features overlap. In this area, errors in classification are likely. A more general cost or risk measure may be associated with a classification strategy.

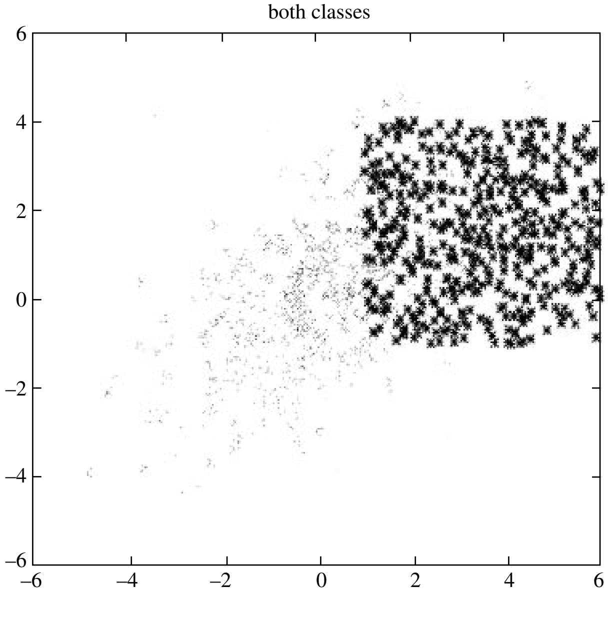


Figure 2. Example of feature vector overlap, leading to classification error.

Pattern Classification

Classification is the assignment of input data into one or more of *c* prespecified classes based on extraction of significant features or attributes and the processing or analysis of these attributes. It is common to resort to probabilistic or grammatical models in classification.

Recognition is the ability to classify. Often, we formulate PR problems with a $c + 1^{st}$ class, corresponding to the "unclassifiable," "donot know," or "cannot decide" class.

Description is an alternative to classification in which a structural description of the input pattern is desired. It is common to resort to linguistic or structural models in description. A *pattern class* is a set of patterns (hopefully sharing some common attributes) known to originate from the same source. The key in many PR applications is to identify suitable attributes (e.g., features) and form a good measure of similarity and an associated matching process.

Preprocessing is the filtering or transforming of the raw input data to aid computational feasibility and feature extraction and minimize noise.

Noise is a concept originating in communications theory. In PR, the concept is generalized to represent a number of nonideal circumstances.

Pattern Matching

Much of StatPR, SyntPR, and NeurPR is based on the concept of *pattern similarity*. For example, if a pattern, *x*, is very similar to other patterns known to belong to class w_1, we would intuitively tend to classify *x* as belonging in w_1. *Quantifying similarity* by developing suitable *similarity measures* is often quite difficult. Universally applicable similarity measures that enable good classification are both desirable and elusive.

Measures of similarity (or dissimilarity) using feature vectors are commonly used. Distance is one measure of vector similarity. The Euclidean distance between vectors $\underline{x}$ and $\underline{y}$ is given by

$$d(\underline{x},\underline{y}) = \|\underline{x}-\underline{y}\| = \sqrt{(\underline{x}-\underline{y})^T(\underline{x}-\underline{y})} = +\sqrt{\sum_{i=1}^{d}(x_i-y_i)^2}$$

A related and more general metric is

$$d_p(\underline{x},\underline{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Commonly, weighted distance measures are used. An example is

$$d_w^2(\underline{x},\underline{y}) = (\underline{x} - \underline{y})^T R(\underline{x} - \underline{y}) = \|\underline{x} - \underline{y}\|_R^2$$

which implements on a *weighted inner product* or weighted R-norm. The matrix R is often required to be positive definite and symmetric. When $\underline{x}$ and $\underline{y}$ are binary, measures such as the Hamming distance are useful.

Training

A set of "typical" patterns, in which typical attributes or the class or structure of each is known, forms a database. This database is called the *training set* and denoted H. In a general sense, the training set provides significant information on how to associate input data with output decisions (i.e., classifications or structural descriptions). Training is often associated (or loosely equated) with learning. The training set is used to enable the system to "learn" relevant information, such as statistical parameters, natural groupings, key features, or underlying structure. In SyntPR, training samples are used to learn or infer grammars.

Supervised and Unsupervised Classification

Training uses representative (and usually labeled) samples of types of patterns to be encountered in the actual application. The training set is denoted H or H_i, in which the subscript denotes a training set for a specific pattern class. In some cases, the training set for class w_i contains examples of patterns in w_i (positive exemplars) as well as examples of patterns not in w_i (negative exemplars).

In this context, *supervised* learning or training assumes a labeled (with respect to pattern class) set, whereas in *unsupervised* learning, the elements of H do not have class labels and the system must determine "natural" partitions of the sample data.

For example, consider the application of PR to *image segmentation* (i.e., the classification of image pixels into groupings that represent some higher entity or information in the images). Unfortunately, it is rare to have either a statistical model to aid in this grouping or a training set. Therefore, so-called *unsupervised learning* techniques are often applied.

Two unsupervised learning approaches that embody more general measures of feature vector similarity and do not require H are known as *hierarchical clustering* and *partitional clustering*. A set of feature vectors is sequentially partitioned (or merged) on the basis of dissimilarity (or similarity). Thus, given only a similarity measure, we either aggregate feature vectors into a single class or sequentially subdivide feature vector partitions. A neural network-based example of unsupervised learning is the Kohonen SOFM.

STATISTICAL PATTERN RECOGNITION (StatPR)

Statistical Analysis

StatPR is used to develop statistically based *decision or classification strategies*, which form *classifiers* and attempts to integrate all available problem information, such as measurements and a priori probabilities. Decision rules may be formulated in several inter-related ways. A measure of expected classification error or "risk" may be formulated, and a decision rule is then developed that minimizes this measure. The Bayesian approach involves converting an a priori class probability $P(w_i)$ into a measurement-conditioned ("a posteriori') probability $P(w_i|\underline{x})$, which leads to a partitioning of R^d, and may be implemented via *discriminant functions*.

Bayes Decision Theory

In the Bayesian approach, the extracted features $\underline{x}$ are modeled as a realization of a (continuous) random vector $\underline{X}$. The case of discrete r.v.s is treated similarly, but with probabilities, as opposed to density functions for characterization of $\underline{x}$. Suppose the class-conditioned probability density functions for feature vector $\underline{x}$ (i.e., $p(\underline{x}|w_i)$ where $i = 1, c$) are available, which may be the result of training or learning. Assume that something is known about the a priori (i.e., before measurement) likelihood of the occurrence of class w_1 or w_2, specifically assume the a priori probabilities $P(w_i), i = 1, c$ are known. For example, in the shim-bolt example above, if we know that on a given day we inspect four times as many shims as bolts, then $P(w_1) = 0.8$ and $P(w_2) = 0.2$. In the absence of this information, an often reasonable assumption is that $P(w_i) = \frac{1}{c}$ (i.e., the a priori probabilities of the states of nature are equal).

Using Bayes Theorem. Bayes theorem is used to enable a solution to the classification problem that uses available feature and training data. The a priori estimate of the probability of a certain class is converted to the a posteriori, or *measurement conditioned*, probability of a state of nature via:

$$p(w_i|\underline{x}) = \frac{[p(\underline{x}|w_i)P(w_i)]}{p(\underline{x})}$$

where

$$p(\underline{x}) = \sum_i p(\underline{x}|w_i)$$

An intuitive classification strategy is that a given realization or sample vector $\underline{x}$ is classified by choosing the state of nature w_i for which $P(w_i|\underline{x})$ is largest. Notice the quantity $p(\underline{x})$ is common to all class-conditional probabilities; therefore, it represents a scaling factor that may be eliminated. Thus, in our shim-bolt example, the decision or classification algorithm is

$$choose \begin{cases} w_1 & \text{if } p(\underline{x}|w_1)P(w_1) > p(\underline{x}|w_2)P(w_2) \\ w_2 & \text{if } p(\underline{x}|w_2)P(w_2) > p(\underline{x}|w_1)P(w_1) \end{cases}$$

Note also that any monotonically nondecreasing function of $P(w_i|\underline{x})$ may be used for this test (see discriminant functions, below). The significance of this approach is that both a priori information ($P(w_i)$) and measurement-related information ($p(\underline{x}|w_i)$) are *combined* in the decision procedure. If $P(w_1) \neq P(w_2)$, for example, this information may be explicitly incorporated in the decision process.

Decision Regions and Discriminant Functions

A *classifier* partitions feature space into class-labeled *decision regions*. In order to use decision regions for a possible and unique class assignment, these regions must cover R^d and be disjoint (nonoverlapping). An exception to the last constraint is the notion of *fuzzy sets*. The border of each decision region is a *decision boundary*. With this viewpoint, classification of feature vector $\underline{x}$ becomes quite simple: We determine the decision region (in R^d) into which $\underline{x}$ falls and assign $\underline{x}$ to this class. Although the classification strategy is straightforward, the determination of decision regions is a challenge. It is sometimes convenient, yet not always necessary (or possible), to visualize decision regions and boundaries. Moreover, computational and geometric aspects of certain decision boundaries (e.g., linear classifiers that generate hyperplanar decision boundaries) are noteworthy.

A number of classifiers are based on *discriminant functions*. In the c-class case, discriminant functions, denoted $g_i(\underline{x}), i = 1, 2, \ldots c$, are used to partition R^d using the *decision rule:* Assign $\underline{x}$ to class w_m (region R_m), where $g_m(\underline{x}) > g_i(\underline{x}) \, \forall i = 1, 2, \ldots c$ and $i \neq m$. The case in which $g_k(\underline{x}) = g_l(\underline{x})$ defines a *decision boundary*.

Linear Separability (1,2). If a linear decision boundary (hyperplanar decision boundary) exists that correctly classifies all the training samples in H for a $c = 2$ class problem, the samples are said to be linearly separable. This hyperplane, denoted H_{ij}, is defined by parameters $\underline{w}$ and w_o in a linear constraint of the form

$$g(\underline{x}) = \underline{w}^T \underline{x} - w_o = 0 \tag{1}$$

$g(\underline{x})$ separates R^d into positive and negative regions R_p and R_n, where

$$g(\underline{x}) = \underline{w}^T \underline{x} - w_o = \begin{cases} > 0 & \text{if} \quad \underline{x} \in R_p \\ 0 & \text{if} \quad \underline{x} \in H_{ij} \\ < 0 & \text{if} \quad \underline{x} \in R_n \end{cases} \tag{2}$$

Problems that are not linearly separable are sometimes referred to as *nonlinearly separable* or *topologically complex*.

From a PR viewpoint, the computational advantages (both in implementation and training) and ease of visualization of linear classifiers account for their popularity. Seminal works include Refs. 11–13.

Using the Bayesian approach, one choice of discriminant function is $g_i(\underline{x}) = P(w_i|\underline{x})$. In the case of equal a priori probabilities and class-conditioned Gaussian density functions, Equation (2) shows that the decision boundaries are hyperplanes.

Training in StatPR. One of the problems not addressed in the previous section is determination of the parameters for the class-conditioned probability density functions. A labeled set of *training samples* (i.e., sets of labeled feature vectors with known class) are often used. This training set is denoted H. In the case of Gaussian pdf models, it is only necessary to estimate $\underline{\mu}_i$ and $\sum_i$ for each class. Large-dimension feature vectors, and consequently density functions, lead to situations in which this approach is impractical. For example, in an *image processing* application, if we use the gray-level measurements directly as features, an image with 100 × 100 pixel spatial resolution yields a 1000 × 1 feature vector, and requires estimation of a 1000 × 1000 covariance matrix. This application is seldom practical.

Nearest Neighbor Classification

An alternative, which is related to the minimum distance classification approach, is the use of a nonparametric technique known as *nearest neighbor classification*. We illustrate the concept of a 1-nearest neighbor classification rule (1-NNR) first. Given a feature vector $\underline{x}$, we determine the vector in H which is closest (in terms of some distance measure) to $\underline{x}$, and denote this vector $\underline{x}\prime$. $\underline{x}$ is classified by assigning it to the class corresponding to $\underline{x}\prime$. A variation is the k-NNR, where the k samples in H that are nearest to $\underline{x}$ are determined, and the class of $\underline{x}$ is based on some measure of the labels of these samples (e.g., a voting scheme may be employed). This approach, although conceptually and computationally straightforward, may be shown to have a greater error rate than the minimum distance classifier. However, the concept of classification based on nearness, or similarity, of features is significant (see Pattern Matching).

General Decision Rules. We formulate a *loss function, cost function*, or *risk function*, denoted λ_{ij}, as the cost or risk of choosing class w_i when class w_j is the true class. For example, in the $c = 2$ (w_1 or w_2) case, there are four values of λ_{ij} (i.e., λ_{11}, λ_{12}, λ_{21}, λ_{22}). λ_{11} and λ_{22} are the costs (or perhaps "rewards" for a correct decision), whereas λ_{12} and λ_{21} are the costs of a classification error. It is desirable to measure or estimate overall classification risk. To measure this risk, the decision rule, cost functions, and the observations $\underline{x}$ are used. A *decision or classification* to choose class w_i is denoted α_i. A *decision rule* is a mapping of the observed feature vector $\underline{x}$ into an α_i through a decision rule $\alpha(\underline{x})$

$$\alpha(\underline{x}) \rightarrow \{\alpha_1, \alpha_2 \ldots \alpha_c\}$$

As

$$P(\alpha_i \cap w_j) = P(\alpha_i|w_j)P(w_j)$$

an overall risk measure for the $c = 2$ case is

$$\begin{aligned} R = {} & \lambda_{11}P(\alpha_1|w_1)P(w_1) + \lambda_{21}P(\alpha_2|w_1)P(w_1) \\ & + \lambda_{12}P(\alpha_1|w_2)P(w_2) + \lambda_{22}P(\alpha_2|w_2)P(w_2) \end{aligned}$$

Of course, the $P(\alpha_i|w_j)$ terms depend on the chosen mapping $\alpha(\underline{x}) \to \alpha_i$, which in turn depends on $\underline{x}$. Thus, a measure of *conditional risk* associated with a $c = 2$ class decision rule is

$$R(\alpha(\underline{x}) \to \alpha_1) = R(\alpha_1|\underline{x}) = \lambda_{11}P(w_1|\underline{x}) + \lambda_{12}P(w_2|\underline{x})$$

for α_1 and

$$R(\alpha(\underline{x}) \to \alpha_2) = R(\alpha_2|\underline{x}) = \lambda_{21}P(w_1|\underline{x}) + \lambda_{22}P(w_2|\underline{x})$$

for α_2. For a c-class decision problem, the expected risk is given by an application of the total probability theorem

$$R(\alpha(\underline{x})) = \int R(\alpha(\underline{x})|\underline{x})p(\underline{x})\underline{d}x$$

Minimizing the conditional risk, $R(\alpha(\underline{x})|\underline{x})$ thus minimizes the expected risk. The lower bound on $R(\alpha(\underline{x}))$ is often referred to as the *Bayes risk*. In order to minimize $R(\alpha(\underline{x}))$ for $c = 2$, because only two choices or classifications (α_1 or α_2) are possible, the decision rule is formulated as

$$R(\alpha_1|\underline{x}) \quad \underset{\alpha_1}{\overset{\alpha_2}{\gtrless}} \quad R(\alpha_2|\underline{x})$$

which may be expanded into

$$\lambda_{11}P(w_1|\underline{x}) + \lambda_{12}P(w_2|\underline{x}) \quad \underset{\alpha_1}{\overset{\alpha_2}{\gtrless}} \quad \lambda_{21}P(w_1|\underline{x}) + \lambda_{22}P(w_2|\underline{x})$$

or

$$(\lambda_{11} - \lambda_{21})p(\underline{x}|w_1)P(w_1) \quad \underset{\alpha_1}{\overset{\alpha_2}{\gtrless}} \quad (\lambda_{22} - \lambda_{12})p(\underline{x}|w_2)P(w_2)$$

When $\lambda_{11} = \lambda_{22} = 0$ (there is no “cost” or “risk” in a correct classification) and $(\lambda_{11} - \lambda_{21}) < 0$, the above may be rewritten as

$$\frac{p(\underline{x}|w_1)}{p(\underline{x}|w_2)} \quad \underset{\alpha_2}{\overset{\alpha_1}{\gtrless}} \quad \frac{(\lambda_{22} - \lambda_{12})P(w_2)}{(\lambda_{11} - \lambda_{21})P(w_1)}$$

This form yields a classifier based on a *likelihood ratio test* (LRT).

For c classes, with the loss function

$$\lambda_{ij} = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases}$$

all errors are *equally costly*. The conditional risk of decision α_i is

$$\begin{aligned} R(\alpha(\underline{x}) \to \alpha_i) &= \sum_{j=1}^{c} \lambda_{ij}P(w_j|\underline{x}) \\ &= \sum_{j \neq i} P(w_j|\underline{x}) = 1 - P(w_i|\underline{x}) \end{aligned}$$

To minimize the conditional risk, the decision rule is therefore to choose the α_i that maximizes $P(w_i|\underline{x})$ (i.e., the w_i for which $P(w_i|\underline{x})$ is largest, which is intuitively appealing. As $P(w_i|\underline{x})$ is the a posteriori probability, this results in the *maximum a posteriori probability* (MAP) classifier, which may be formulated as

$$P(w_i|\underline{x}) \overset{\alpha_i}{>} P(w_j|\underline{x}) \ \ \forall_j \neq i$$

As before, Bayes rule is used to reformulate these tests in terms of class-conditioned density functions and a priori probabilities.

For general formulations of risk (through λ_{ij}), the resulting decision rule is

$$R(\alpha_i|\underline{x}) \overset{\alpha_i}{<} R(\alpha_j|\underline{x}) \ \ \forall_i \neq j$$

Clustering

In some cases, a training set, H, is not available for a PR problem. Instead, an unlabeled set of *typical* features, denoted H_u, is available (see *unsupervised learning*). For each sample, $\underline{x} \in H_u$, the class origin or label is unknown. Desirable attributes of H_u are that the cardinality of H_u is large, all classes are represented in H_u, and subsets of H_u may be formed into natural groupings or “clusters”. Each cluster most likely (or hopefully) corresponds to an underlying pattern class.

Clustering is a popular approach in unsupervised learning (14). Clustering applications in *image analysis*, for example, include Refs. 15 and 16. Iterative algorithms involving cluster splitting and merging in *image analysis* are shown in Ref. 2.

Unsupervised learning approaches attempt to develop a representation for the given sample data, after which a classifier is designed. In this context, clustering may be conceptualized as “how do I build my fences?” Thus, in unsupervised learning, the objective is to *define the classes*. A number of intuitive and practical approaches exist to this problem. For example, a self-consistent procedure is

1. Convert a set of unlabeled samples H_u into a *tentative* training set H_T.
2. Using H_T, apply a supervised training procedure and develop corresponding discriminant functions/decision regions.
3. Use the results of 2 on H_u (i.e., reclassify H_u). If the results are consistent with H_T, stop, otherwise go to 1 and revise H_T.

This approach “clusters” data by observing similarity. There exist neural networks with this feature (see *Self Organizing Feature Maps*). In many PR applications involving unsupervised learning, features naturally fall into natural, easily observed groups. In others, the grouping is unclear and very sensitive to the measure of similarity used. The c-means algorithm [Equation (2)] and it’s derivatives are one of the most popular approaches.

The *c*-means Algorithm.

1. Choose the number of classes, c.
2. Choose class means or exemplars, denoted $\underline{\hat{\mu}}_i, \underline{\hat{\mu}}_2, \cdots \underline{\hat{\mu}}_c$.
3. Classify each of the unlabeled samples $\underline{x}_k$ in H_u.
4. Recompute the estimates for $\underline{\hat{\mu}}_i$ using the results of 3.
5. If the $\underline{\hat{\mu}}_i$ are consistent, stop, otherwise go to step 1, 2, or 3.

Notice the essence of this approach is to achieve a self-consistent partitioning of the data. Choice of initial parameters (c and $\underline{\hat{\mu}}_i(o)$) is a challenging issue, which has spawned an area of study concerning *cluster validity*.

An Example of the c-means Algorithm. Figure 3 shows examples of the c-means algorithm for the $c = 2$ class case on a set of unlabeled data (1). The trajectory of the $\underline{\hat{\mu}}_i$ as a function of iteration is shown.

Iterative and Hierarchical Clustering. Clustering may be achieved through a number of alternative strategies, including iterative and hierarchical approaches. Hierarchical strategies may further be subdivided into agglomerative (merging of clusters) or devisive (splitting of clusters). Hierarchical strategies have the property that not all partitions of the data are considered. However, when the number of samples is large, hierarchical clustering may be inappropriate. In an agglomerative procedure, two samples, once in the same class, remain in the same class throughout subsequent cluster merging, which may lead to resulting data partitions being suboptimal.

Clustering Criterion Functions. Developing appropriate similarity measures $d(\underline{x}_i, \underline{x}_j)$ is paramount in clustering. For a given partition of H_u, denoted P, a measure of the "goodness" of the overall clustering is given by *clustering criterion function*, $J(P)$. If

$$J(P_1) < J(P_2)$$

P_1 is a better partition than P_2. Once a suitable $J(P)$ is defined, the objective is to find P_m such that

$$J(P_m) = \overset{min}{P}\,(J(P))$$

in a computationally efficient manner, which is a problem in discrete optimization. One of the more popular clustering metrics is the *sum of squared error (SSE) criterion*. Given n_i samples in H_i, with sample mean $\underline{m}_i$, where

$$\underline{m}_i = \frac{1}{n_i} \sum_{\underline{x}_j \in H_i} \underline{x}_j$$

the SSE criterion, J_{SSE} is defined as

$$J_{SSE}(P) = \sum_{i=1}^{c} \sum_{\underline{x}_j \in H_i} \|\underline{x} - \underline{m}_i\|^2$$

J_{SSE} thus indicates the total "variance" for a given partition. For example, cluster-swapping approaches are a variant on the c-means iterative algorithm that implements a "good" cluster reorganization strategy, where "good" means

$$J_{SSE}(P_{k+1}) \leq J_{SSE}(P_k)$$

For illustration, our reorganization strategy is restricted to the movement of a single vector $\underline{x}_j$ from H_i to H_j, denoted $H \xrightarrow{\underline{x}_j} H_j$. The revised clusters in P_{k+1} are denoted H_i

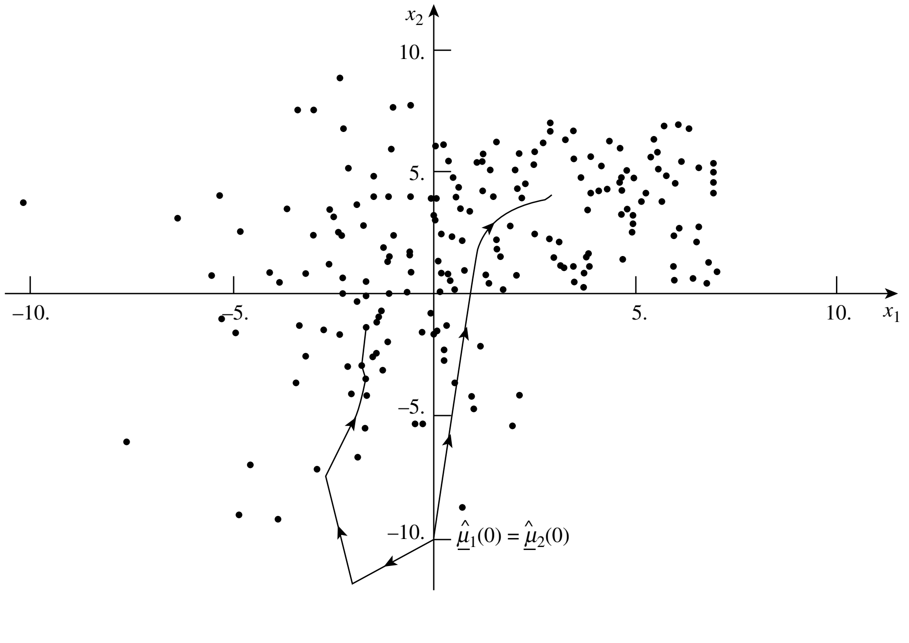


Figure 3. Example of the trajectories of the class means in the c-means algorithm (Adapted from Ref. 1).

and H_j. It is possible to show $H_i \overset{x_k}{\to} H_j$ decreases $J_{SSE}(P_k)$ if

$$\left(\frac{n_j}{n_j+1}\right)\|\underline{x}_j-\underline{m}_j\|^2<\left(\frac{n_i}{n_i-1}\right)\|\underline{x}_j-\underline{m}_i\|^2$$

Hierarchical Clustering. Consider a hierarchical clustering procedure in which clusters are merged so as to produce the smallest increase in the SSE at each step. The *ith* cluster or partition, denoted H_i, contains n_i samples with sample mean $\underline{m}_i$. The smallest increase results from merging the pair of clusters for which the measure M_{ij}, where

$$M_{ij}=\frac{n_i n_j}{n_i+n_j}\|\underline{m}_i-\underline{m}_j\|^2$$

is minimum. Recall

$$J_e=\sum_{j=1}^{c}\sum_{\underline{x}\in H_i}\|\underline{x}-\underline{m}_i\|^2$$

(i.e., J_e measures the total squared error incurred in representing the n samples $\underline{x}_1,\ldots\underline{x}_n$ by c cluster means $\underline{m}_1\ldots\underline{m}_c$).

The *change* in the SSE after merging clusters i and j is

$$\Delta J_e=-\left(\sum_{\underline{x}\in H_i}\|\underline{x}-\underline{m}_i\|^2+\sum_{\underline{x}\in H_j}\|\underline{x}-\underline{m}_j\|^2\right)+\sum_{\underline{x}\in H_i or H_j}\|\underline{x}-\underline{m}_{ij}\|^2$$

where

$$\underline{m}_i=\frac{1}{n_i}\sum_{\underline{x}\in H_i}\underline{x}\quad \underline{m}_j=\frac{1}{n_j}\sum_{\underline{x}\in H_j}\underline{x}$$

and

$$\underline{m}_{ij}=\frac{1}{n_i+n_j}\sum_{\underline{x}\in H_i\, or\, H_j}\underline{x}$$

The objective is to merge clusters so that ΔJ_e is minimum. It is possible to show

$$\Delta J_e=\frac{n_i n_j}{(n_i+n_j)}\|\underline{m}_j-\underline{m}_i\|^2=\frac{n_i n_j}{(n_i+n_j)}\|\underline{m}_i-\underline{m}_j\|^2$$

and therefore use this measure in choosing clusters to merge.

The popularity of clustering has spawned a sizable and varied library of clustering algorithms and software (17), one of the most popular being the *ISODATA algorithm* (10–18).

SYNTACTIC (STRUCTURAL) PATTERN RECOGNITION

Many times the significant information in a pattern is not merely in the presence or absence, or the numerical values, of a set of features. Instead, the interrelationships or interconnections of features yield important *structural* information, which facilitates structural description or classification, which is the basis of syntactic (or structural) PR. Figure 4 shows the general strategy (1).

In using SyntPR approaches, it is necessary to quantify and extract structural information and determine the structural similarity of patterns. One syntactic approach is to relate the structure of patterns with the syntax of a formally defined language in order to capitalize on the vast body of knowledge related to pattern (sentence) generation and analysis (parsing). Syntactic PR approaches are presented in Refs. 1, 19–23. A unified view of StatPR and SyntPR is shown in Ref. 24. An extended example of the use of SyntPR in an *image interpretation* application is shown in Ref. 25.

Typically, SyntPR approaches formulate hierarchical descriptions of complex patterns built up from simpler subpatterns. At the lowest level, *primitive elements* or "building blocks" are extracted from the input data. One distinguishing characteristic of SyntPR involves the choice of primitives. *Primitives must be subpatterns or building blocks, whereas features (in StatPR) are any measurements*.

Syntactic structure quantification is shown using two approaches: formal grammars and relational descriptions (attributed graphs). These tools allow structurally quantitative pattern representation, which facilitate recognition, classification, or description. A class of procedures for *syntactic recognition*, including *parsing* (for formal grammars) and *relational graph matching* (for attributed relational graphs) are then developed. Although it is not mandatory, many SyntPR techniques are based on generation and analysis of complex patterns by a hierarchical decomposition into simpler patterns.

Formal Grammars and Syntactic Recognition by Parsing

The syntax rules of formal grammars may be used to generate patterns (possibly from other patterns) with constrained structural relations. A grammar may therefore serve to model a class-specific pattern-generating source that generates all the patterns with a class-specific

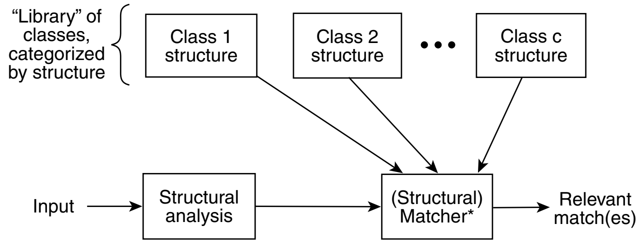


Figure 4. Generic syntactic (or structural) PR system (Adapted from Ref. 1).

structure. Furthermore, it is desirable to have each class-specific grammar derivable from a set of sample patterns (i.e., training must be considered, which raises the issue of *grammatical inference*.

Useful introductions to formal grammars are available in Refs. 26 and 27. References 19, 21–23 are devoted entirely to SyntPR.

Grammars. A *grammar* consists of the following four entities:

1. A set of terminal or primitive symbols (primitives), denoted V_T (or, alternately, $\sum$). In many applications, the choice of the terminal set or primitives is difficult and has a large component of "art" as opposed to "science."
2. A set of *nonterminal symbols, or variables*, which are used as intermediate quantities in the generation of an outcome consisting solely of terminal symbols. This set is denoted as V_N (or, alternately, *N*).
3. A set of *productions, or production rules or rewriting rules* that allow the previous substitutions. It is this set of productions, coupled with the terminal symbols, which principally gives the grammar its "structure." The set of productions is denoted *P*.
4. A starting (or root) symbol, denoted S. $S \in V_N$.

Note that V_T and V_N are disjoint sets (i.e., $V_T \cap V_N = \phi$).

Thus, using the above definitions, we formally denote a grammar G as the four-tuple:

$$G = (V_T, V_N, P, S)$$

Constraining Productions. Given V_T and V_N, the productions *P* may be viewed as constraints on how class-specific patterns may be described. Different types of grammars place restrictions on these mappings. For example, it is reasonable to constrain elements of *P* to the form

$$A \rightarrow B$$

where

$$A \in (V_N \cup V_T)^+ - V_T^+$$

and

$$B \in (V_N \cup V_T)^*$$

Thus, *A* must consist of at least one member of V_N, (i.e., a nonterminal), and *B* is allowed to consist of any arrangement of terminals and nonterminals. This example is a partial characterization of *phrase structure grammar*.

Grammar Application Modes. A grammar may be used in one of two modes: *Generative:* The grammar is used to create a string of terminal symbols using P; a *sentence* in the language of the grammar is thus generated.

Analytic: Given a sentence (possibly in the language of the grammar), together with specification of G, one seeks to determine if the sentence was generated by G and, if so, the structure (usually characterized as the sequence of productions used) of the sentence.

The following formal notation is used. Symbols beginning with a capital letter (e.g., S_1 or S) are elements of V_N. Symbols beginning with a lowercase letter (e.g., a or b) are elements of V_T. n denotes the length of string s, for example,

$$n = |s|$$

Greek letters (e.g., α and β) represent (possibly empty) strings, typically comprised of terminals or nonterminals.

Constraints on the production or rewrite rules, *P*, in string grammar G are explored by considering the "general" production form

$$\alpha_1 \rightarrow \beta_2$$

which means string α_1 "is replaced by" string β_2. In general, α_1 and β_2 may contain terminals or nonterminals.

In a context-free grammar, the production restrictions are

$$\alpha_1 = S_1 \in V_N$$

that is, α_1 *must be a single nonterminal* for every production in *P*, and

$$|S_1| \leq |\beta_2|$$

An alternate characterization of a T_2 grammar is that every production must be of the form

$$S_1 \rightarrow \beta_2$$

where $\beta_2 \in (V_N \cup V_T)^* - \{\epsilon\}$. Note the restriction in the above productions to the replacement of S_1 by string β_2 *independently of the context in which* S_1*appears*.

Context-free grammars can generate a string of terminals or nonterminals in a single production. Moreover, because productions of the form $A \rightarrow \alpha A \beta$ are allowed, context-free grammars are *self-embedding*.

Context-free grammars are important because they are the most descriptively versatile grammars for which effective (and efficient) parsers are available. The production restrictions increase in going from context-sensitive to context-free grammars.

Finite-state or regular grammars are extremely popular. The production restrictions in a finite-state or regular grammar are those of a context-free grammar, plus the additional restriction that *at most one nonterminal symbol is allowed on each side of the production*. for example,

$$\alpha_1 = S_1 \in V_N$$
$$|S_1| \leq |\beta_2|$$

and productions are restricted to

$$A_1 \rightarrow a$$

or

$$A_1 \rightarrow aA_2$$

Finite-state grammars have many well-known characteristics that explain their popularity, including simple graphical representations and known tests for equivalence. Finite-state grammars are useful when analysis (parsing) is to be accomplished with finite-state machines (26).

Other Grammar Types Used for SyntPR. Grammars other than string grammars exist and are usually distinguished by their terminals and nonterminals (as opposed to constraints on P), which are useful in 2-D and higher-dimensional pattern representation applications in that the structure of the productions involving terminals and nonterminals is greater than one dimensional. Higher-dimensional grammars also facilitate relational descriptions. Productions in higher-dimensional grammars are usually more complex, because rewriting rules embody operations more complex than simple 1-D string rewriting. For example, in 2-D cases, standard "attachment points" are defined. Two of the more popular are *tree grammars* and *web grammars*(19). Not surprisingly, there is little correlation between the dimension of the *grammar* used for pattern generation and the dimensionality of the pattern space. For example, a 1-D grammar may be used for 2-D or 3-D patterns.

Example of Grammatical Pattern Description for Chromosome Classification. Figure 5, excerpted from Ref. 28, shows the conversion of a chromosome outline to a string in a formal grammar, where the primitives and productions are given. Using the primitives and productions of grammar, G_M, given in part (a), the string $x =$ *cbbbabbbbdbbbbabbbcbbbabbbbdbbbbabbb* may be produced to describe the sample chromosome outline shown in part (b).

Parsing

Chomsky Normal Form (CNF). A CFG is in Chomsky Normal Form (CNF) if each element of P is in one of the following forms:

$$\begin{aligned} A &\rightarrow BC \ \ where \ A, B, C \in V_N \\ A &\rightarrow a \ \ where \ \in V_N, a \in V_T \end{aligned}$$

The Cocke–Younger–Kasami (CYK) Parsing Algorithm. The CYK algorithm is a parsing approach that will parse string x ina number of steps proportional to $|x|^3$. The CYK algorithm requires the CFG be in Chomsky Normal Form (CNF). With this restriction, the derivation of any string involves a series of binary decisions. First, the CYK table is formed. Given string $x = x_1, x_2 \ldots x_n$, where $x_i \in V_T$, $|x| = n$, and a grammar G, we form a triangular table with entries t_{ij} indexed by i and$j$ where $1 \leq i \leq n$ and $1 \leq j \leq (n - i + 1)$. The origin is at $i = j = 1$, and entry t_{11} is the lower left-hand entry in the table. t_{1n} is the uppermost entry in the table. This structure is shown in Fig. 6.

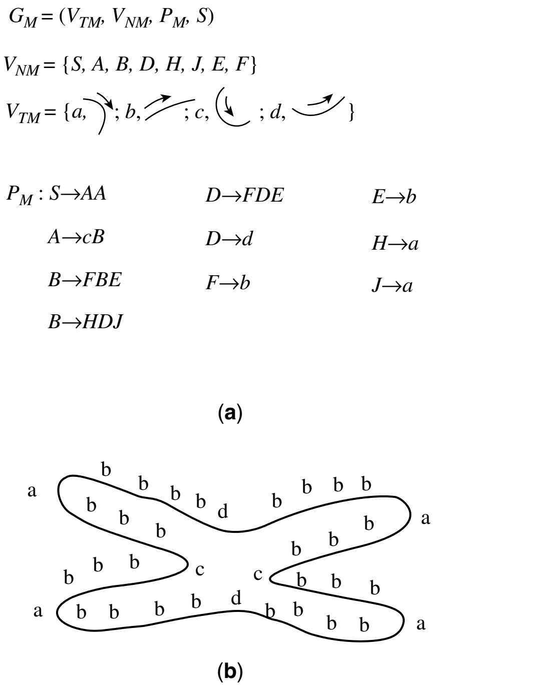


Figure 5. Conversion of a chromosome outline to a string in a formal grammar (excerpted from Ref. 28). (a) Primitives and productions in L(G). (b) Sample chromosome outline yielding string $x =$ *cbbbabbbbdbbbbabbbcbbbabbbbdbbbbabbb*.

To build the CYK table, a few simple rules are used. Starting from location (1,1), *if a substring ofx, beginning withx$_i$, and of lengthjcan be derived from a nonterminal, this nonterminal is placed into cell* (i,j). If cell $(1,n)$ contains S, the table contains a valid derivation of x in $L(G)$. It is convenient to list the x_i, starting with $i = 1$, under the bottom row of the table.

Example: Sample Use of Grammars and the CYK Parsing Algorithm for Recognition

Sample Grammar Productions. Sample grammar productions are shown below. With these constraints, notice there

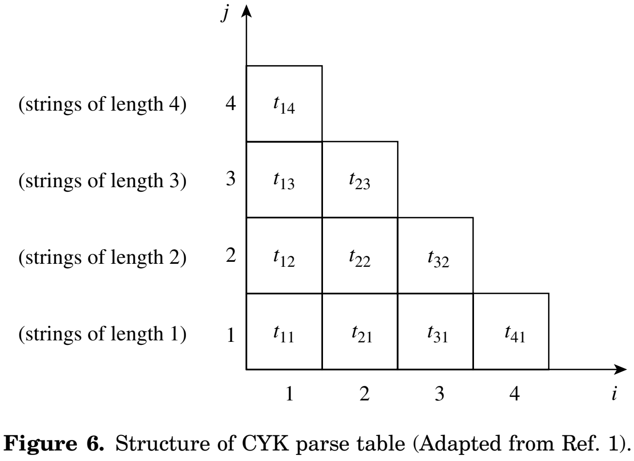

j
(strings of length 4) 4 t_{14}
(strings of length 3) 3 t_{13} t_{23}
(strings of length 2) 2 t_{12} t_{22} t_{32}
(strings of length 1) 1 t_{11} t_{21} t_{31} t_{41}
1 2 3 4 i

Figure 6. Structure of CYK parse table (Adapted from Ref. 1).

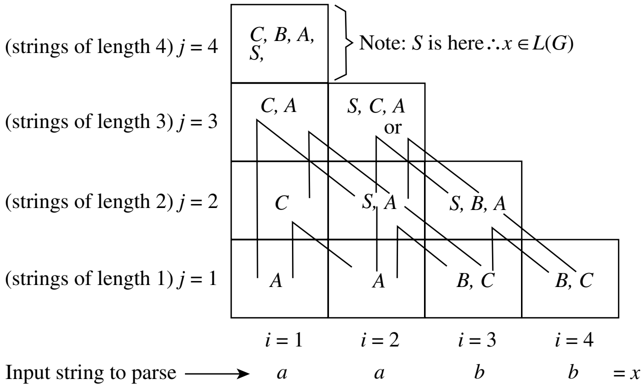

(strings of length 4) j = 4
C, B, A, S,
Note: S is here ∴ x ∈ L(G)
(strings of length 3) j = 3
C, A
S, C, A or
(strings of length 2) j = 2
C
S, A
S, B, A
(strings of length 1) j = 1
A
A
B, C
B, C
i = 1
i = 2
i = 3
i = 4
Input string to parse
a
a
b
b
= x

Figure 7. Construction of a sample parse table for the string $x =$ *aabb* (Adapted from Ref. 1).

are six *forms* for the derivation of the string $x = aabb$.

$$\begin{array}{lcl} S & \rightarrow & AB|BB \\ A & \rightarrow & CC|AB|a \\ B & \rightarrow & BB|CA|b \\ C & \rightarrow & BA|AA|b \end{array}$$

***Parse Table for String* $x = aabb$.** Construction of an example parse table is shown in Fig. 7. Recall cell entry (i,j) corresponds to the possibility of production of a string of length j, starting with symbol x_i. The table is formed from the bottom row ($j = 1$) upward. Entries for cells (1, 1), (2, 1), (3, 1), and (4, 1) are relatively easy to determine because they each correspond with production of a single terminal. For the second ($j = 2$) row of the table, all nonterminals that could yield derivations of substrings of length 2, beginning with $x_i i = 1, 2, 3$, must be considered. For example, cell (1, 2) corresponds with production of two-terminal long-string beginning with "a." Alternately, it is only necessary to consider nonterminals that produce *AA*, as shown in the $j = 1$ row of the table. From Fig. 7, only nonterminal "*C*," in the production $C \rightarrow BA|AA|b$ satisfies this parameter.

Forming the third and fourth ($j = 3$ and $j = 4$, respectively) rows of the table is slightly more complicated. For example, cell (1, 3) corresponds with strings of length 3, beginning with terminal x_1 ("a") in this case, which requires examination of cells (1, 1) and (2, 2), corresponding to producing the desired string with 1 nonterminal followed by 2 nonterminals, (denoted {1 + 2} hereafter) as well as cells (1, 2) and (3, 1) (denoted the {2 + 1} derivation). For the former, it is necessary to consider production of "*AS*," and "*AA*," and nonterminal "*C*" is applicable. For the latter, the production of "*CB*" and "*CC*" is considered, yielding "*A*." Thus, cell (1, 3) contains nonterminals "*C*" and "*A*." Similarly, for cell (2, 3), cells (2, 1) and (3, 2) (the {1 + 2} derivation) as well as cells (2, 2) and (4, 1) (the {2 + 1} derivation) must be considered.

Finally, formation of cell (1, 4) is considered. Possible cell pairings to consider are summarized below:

(1, 1) and (2, 3) $\{1 + 3\} \rightarrow AS, AC, \underline{AA} : C$

(1, 2) and (3, 2) $\{2 + 2\} \rightarrow CS, CB, \underline{CA} : B$

(1, 3) and (4, 1) $\{3 + 1\} \rightarrow CB, \underline{CC}, \underline{AB}, AC : A, S$

Cell pairings that yield a possible nonterminal are shown underlined. Thus, (1, 4) contains nonterminals C, B, A, S. As this cell pairing includes the starting symbol, the parse succeeds and "*aabb*" is a valid string in the language of this grammar. Note that because the grammar is in CNF, it is never necessary to consider more than two-cell pairings (although as we increase j, the number of possible pairings increases).

String Matching. A somewhat simpler approach to classification or recognition of entities using syntactic descriptions is a matching procedure. Consider the c class case. Class-specific grammars $G_1, G_2, \ldots G_c$ are developed. Given an unknown description, x, to classify, it is necessary to determine if $x \in L(G_i)$ for $i = 1, 2, \ldots c$. Suppose the language of each G_i could be generated and stored in a class-specific *library* of patterns. By matching x against *each pattern in each library*, the class membership of x could be determined. String matching metrics yield classification strategies that are a variant of the 1-NNR rule for feature vectors, in which a matching metric using strings instead of vectors is employed.

There are several shortcomings to this procedure. First, often $|L(G_i)| = \infty$, therefore the cataloging or library-based procedure is impossible. Second, even if $L(G_i)$ for each i is denumerable, it usually requires very large libraries. Consequently, the computational effort in matching is excessive. Third, it is an inefficient procedure. Alternatives that employ efficient search algorithms, prescreening of the data, the use of hierarchical matching, and prototypical strings are often preferable. *Note that in SyntPR, the similarity measure(s) used must account for the similarity of primitives as well as similarity of structure.*

Graphical Approaches Using Attributed Relational Graphs

Digraphs and Attributed Relational Graphs (ARGs). *Directed graphs or digraphs* are valuable tools for representing relational information. Here we represent graph G as $G = \{N, R\}$ where N is a set of nodes (or vertices) and R is a subset of $N \times N$, indicating arcs (or edges) in G.

In addition to representing pattern structure, the representation may be extended to include *numerical and perhaps symbolic attributes* of pattern primitives (i.e., relational graph nodes). An extended representation includes features or properties as well as relations with other entities. An *attributed graph*, as defined below, results.

Attributed Graphs. An attributed graph, G_i, is a 3-tuple and is defined as follows:

$$G_i = \{N_i, P_i, R_i\}$$

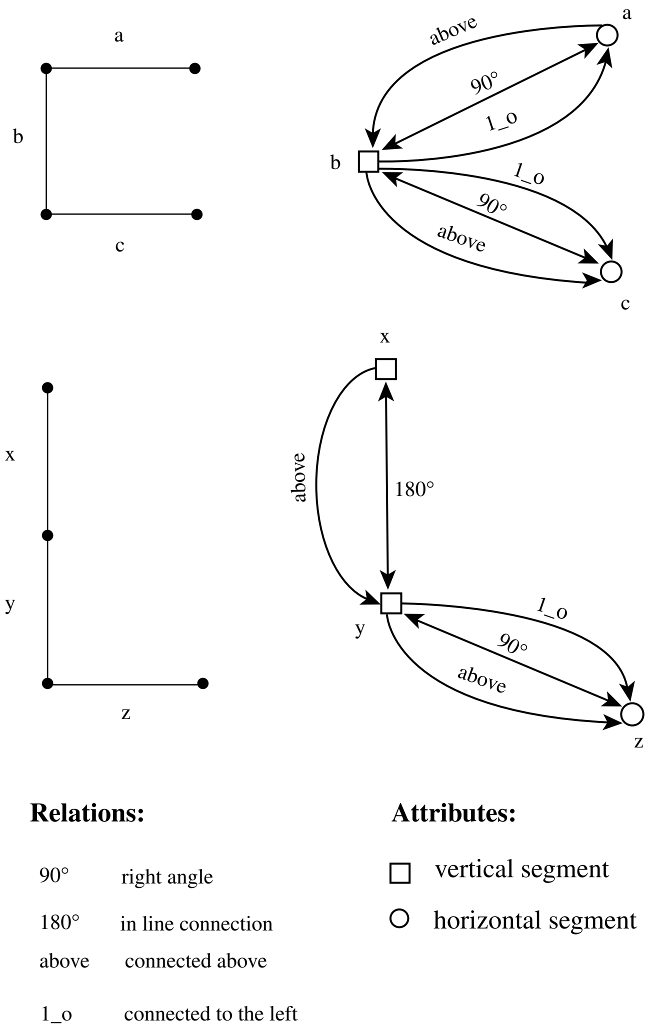


Figure 8. Example of ARGs used to quantify the structure of block characters.

where N_i is a set of nodes, P_i is a set of properties of these nodes, and R_i is a set of relations between nodes. (An alternative viewpoint is that R_i indicates the labeled arcs of G_i, where if an arc exists between nodes a and b, then R_i contains element (a, b).)

ARG Example: Character Recognition. Figure 8 (courtesy of R.D. Ferrell) shows an example of ARGs used to quantify the structure of block characters "*C*" and "*L*." Each line segment of the character is an attributed node in the corresponding graph, with a single attribute indicating either horizontal or vertical spatial orientation. Node relations used indicate whether the segments meet at a 90-or 180-degree angle, as well as connectedness above or to the left.

Comparing ARGs. One way to recognize structure using graphs is to let each pattern (structural) class be represented by a prototypical relational graph. An unknown input pattern is then converted into a structural representation in the form of a representational graph, and this graph is then *compared* with the relational graphs for each class. Notice that "compared" does not necessarily mean matched verbatim.

ARG Matching Measures that Allow Structural Deformations. In order to allow structural deformations, numerous match or "distance" measures have been proposed. These measures include (29, 30)

1. Extraction of *features* from G_1 and G_2, thereby forming feature vectors, $\underset{\sim}{x}_1$ and $\underset{\sim}{x}_2$, respectively, which is followed by the use of StatPR techniques to compare $\underset{\sim}{x}_1$ and $\underset{\sim}{x}_2$. Note the features are *graph features* as opposed to direct pattern features.
2. Using as a matching metric *the minimum number of transformations necessary to transform*G_1 (*the input*) *into*G_2 (*the reference*). Common transformations include: node insertion, node deletion, node splitting, node merging, vertex insertion, and vertex deletion.

Graph Transformation Approaches. Here we consider a set of *comparisons, transformations*, and *associated costs* in deriving a measure $D(G_i, G_j)$. Desirable attributes of $D(G_i, G_j)$ are

1. $D(G_i, G_j) = 0$.
2. $D(G_i, G_j) > 0$ if $i \neq j$.
3. $D(G_i, G_j) = D(G_j, G_i)$.
4. $D(G_i, G_j) \leq D(G_i, G_k) + D(G_k, G_j)$.

Property 4 is referred to as the triangle inequality. Property 3 requires $w_{ni} = w_{nd}$ and $w_{ei} = w_{ed}$, where w_{ni} is the cost of node insertion, w_{nd} is the cost of node deletion, w_{ei} is the cost of edge insertion, and w_{ed} is the cost of edge deletion.

Node Matching Costs and Overall Cost in Matching ARGs. As nodes possess attributes and, therefore, even without considering relational constraints "all nodes are not equal," a similarity measure between node p_i of G_i and node q_j of G_j is required. Denote this cost $f_n(p_i, q_j)$. For candidate match between G_1 and G_2, denoted x, with p nodes, the total cost is

$$c_n(x) = \sum f_n(p_i, q_j)$$

where the summation is over all corresponding node pairs, under node mapping x. For a candidate match configuration (i.e., some pairing of nodes and subsequent transformations), the overall cost for configuration x is

$$D_S(x) = w_{ni}c_{ni} + w_{nd}c_{nd} + w_{bi}c_{bi} + w_{bd}c_{bd} + w_n c_n(x)$$

and the distance measure, D, is defined as

$$D = \overset{min}{x}\{D_s(x)\}$$

NEURAL PATTERN RECOGNITION

Modern digital computers do not emulate the computational paradigm of biological systems. The alternative of *neural computing* emerged from attempts to draw upon knowledge of how biological neural systems store and manipulate information, which leads to a class of *artificial neural systems* termed *neural networks* and involves an amalgamation of research in many diverse fields such as psychology, neuroscience, cognitive science, and systems theory. Neural networks are a relatively new computational paradigm, and it is probably safe to say that the advantages, disadvantages, applications, and relationships to traditional computing are not fully understood. Neural networks are particularly well suited for some pattern association applications.

Fundamental neural network architecture and application information are available in Refs. 2, 31–34. Rosenblatt (35) is generally credited with initial perceptron research. The general feed-forward structure is also an extension of the work of Minsky and Papert (36) and the early work of Nilsson (37) on the transformations enabled by layered machines, as well as the effort of Widrow and Hoff (38) in adaptive systems. A comparison of standard and neural classification approaches is found in Ref. (39).

ANN Components

Basically, three entities characterize an ANN

1. The network topology, or interconnection of neural "units;"
2. The characteristics of individual units or artificial neurons; and
3. The strategy for pattern learning or training.

As in the SyntPR and StatPR approaches to PR, the success of the NeurPR approach is likely to be strongly influenced by the quality of the training data and algorithm. Furthermore, existence of a training set and a training algorithm does not guarantee that a given network will "train" or *generalize* correctly for a specific application.

Key Aspects of Neural Computing

The following are key aspects of neural computing. The overall computational model consists of a *variable interconnection of simple elements, or units*. Modifying patterns of inter-element connectivity as a function of training data is the key learning approach. In other words, the system knowledge, experience, or training is stored in the form of network interconnections.

To be useful, neural systems must be capable of storing information ("trainable"). Neural PR systems are trained with the hope that they will subsequently display correct "generalized" behavior, when presented with new patterns to recognize or classify. That is, the objective is for the network (somehow) in the training process to develop an internal structure that enables it to correctly identify or classify new similar patterns.

Many open questions regarding neural computing and its application to PR problems exist. Furthermore, the mapping of a PR problem into the neural domain (i.e., the design of a problem-specific neural architecture) is a challenge that requires considerable engineering judgment. A fundamental problem is selection of the network parameters, as well as the selection of critical and representable problem features.

Neural Network Structures for PR. Several different "generic" neural network structures are useful for a class of PR problems. Examples are:

The Pattern Associator (PA). This neural implementation is exemplified by feed-forward networks (see Feed-forward Networks). The most commonly used learning (or training) mechanism for FF networks is the backpropagation approach using the *generalized delta rule*.

The Content-Addressable or Associative Memory Model (CAM or AM). This neural network structure is best exemplified by the recurrent network often referred to as the Hopfield model. Typical usage includes recalling stored patterns when presented with incomplete or corrupted initial patterns. (see Hopfield (Recurrent) Networks for PR).

Self-Organizing Networks. These networks exemplify neural implementations of unsupervised learning in the sense that they typically cluster or self-organize input patterns into classes or clusters based on some form of similarity.

Perceptrons

Perceptron and ADALINE Unit Structure. The *Perceptron* is a regular feed-forward network layer with adaptable weights and hardlimiter activation function. Rosenblatt (35) is generally credited with initial perceptron research. The efforts of Widrow and Hoff in adaptive systems, specifically the Adaline and Madeline structures presented in Refs. 40 and 41 are also relevant. For brevity, we will consider them as one generic structure.

The units in the perceptron form a *linear threshold unit, linear* because of the computation of the activation value (inner product) and *threshold* to relate to the type of activation function (hardlimiter). Training of a perceptron is possible with the *perceptron learning rule*. As shown in Fig. 9, the basis for the perceptron/adaline element is a

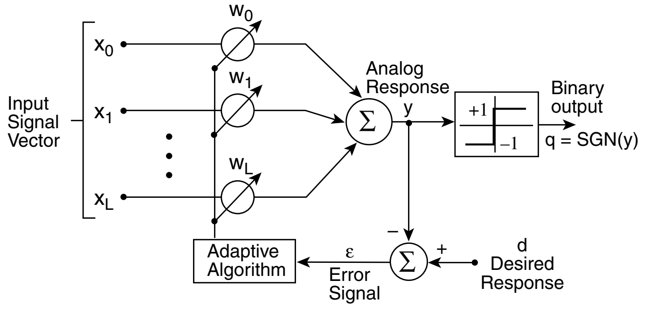


Figure 9. Basic perceptron/adaline element (Adapted from Ref. 41).

single unit whose net activation is computed using

$$net_i = \sum_j w_{ij}x_j = \underline{w}^T\underline{x} \tag{3}$$

The unit output is computed by using a "hard limiter," threshold-type nonlinearity, namely the signum function, for example, for unit i with output o_i

$$o_i = \begin{cases} +1 & if \quad net_i \geq 0 \\ -1 & if \quad net_i < 0 \end{cases} \tag{4}$$

The unit has a *binary output*; however, the formation of net_i (as well as weight adjustments in the training algorithm) is based on the linear portion of the unit (i.e., the mapping obtained before application of the nonlinear activation function).

Combination of Perceptrons or Adaline Units to Achieve More Complex Mappings. Layers of adaline units, often referred to as *multilayer perceptrons* or MLPs may be used to overcome the problems associated with nonlinearly separable mappings. One of the biggest shortcomings of MLPs, however, is the availability of suitable training algorithms. This shortcoming often reduces the applicability of the MLP to small, "hand-worked" solutions. As shown in Fig. 10, combinations of adaline units yield the madaline (modified adaline) or MLP structure, which may be used to form more complex decision regions.

Feed-forward Networks

The *feed-forward (FF) network* is in some sense an extension of the madeline/perceptron structure composed of a hierarchy of processing units, organized in a series of two or more mutually exclusive sets of neurons or layers. The first, or input layer, serves as a holding site for the values applied to the network. The last, or output, layer is the point at which the final state of the network is read. Between these two extremes lie zero or more layers of hidden units; it is here that the real mapping or computing takes place. Links, or weights, connect each unit in one layer to only those in the next higher layer. There is an implied directionality in these connections, in that the output of a unit, scaled by the value of a connecting weight, is fed forward to provide a portion of the activation for the units in the next higher layer. Figure 11 illustrates the typical feed-forward network. The network as shown consists of a layer of d input units (L_i), a layer of c output units (L_o), and a variable number (5 in this example) of internal or "hidden" layers (L_{h_i}) of units. Observe the *feed-forward* structure in which the inputs are directly connected to only units in L_o and the outputs of layer L_k units are only connected to units in layer L_{k+1} or are outputs if $L_k = L_o$.

Training Feed-forward Networks. Once an appropriate network structure is chosen, much of the effort in designing a neural network for PR concerns the design of a reasonable training strategy. Often, for example, while observing a particular training experiment, the designer will notice the weight adjustment strategy "favoring" particular S-R patterns, becoming "painfully" slow (perhaps while stuck in a local minimum), becoming unstable, or oscillating between solutions, which necessitates engineering judgment in considering the following training parameters:

- train by pattern or epoch;
- use of momentum and corresponding weight;
- learning weight/weight changes over time;
- sequential versus random ordering of training vectors;
- whether the training algorithm is "stuck" at a local energy minimum;
- "suitable" unit biases (if applicable); and
- appropriate initial conditions on biases, weights, and so on.

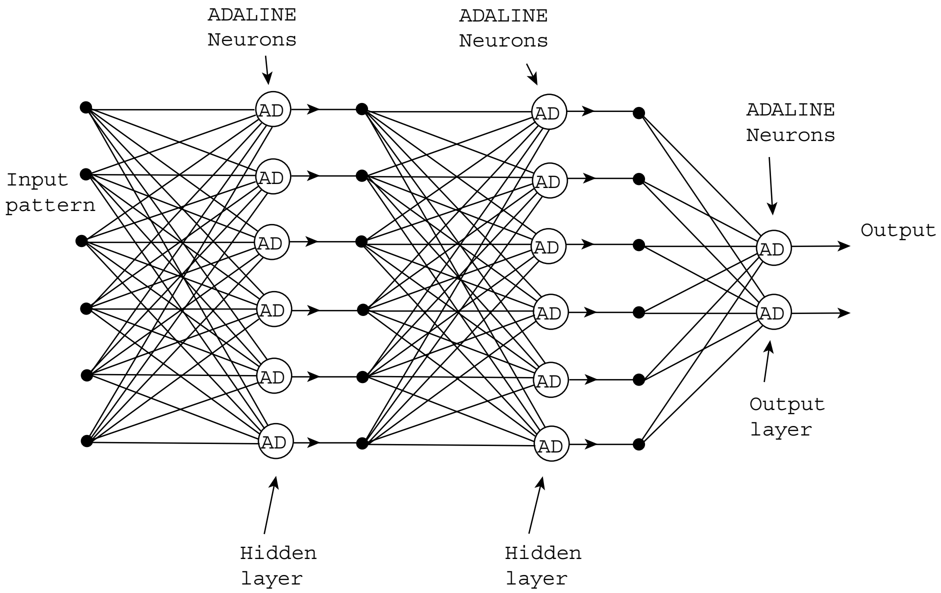


Figure 10. Using combinations of adaline units yield the MLP (madaline) (Adapted from Ref. 41).

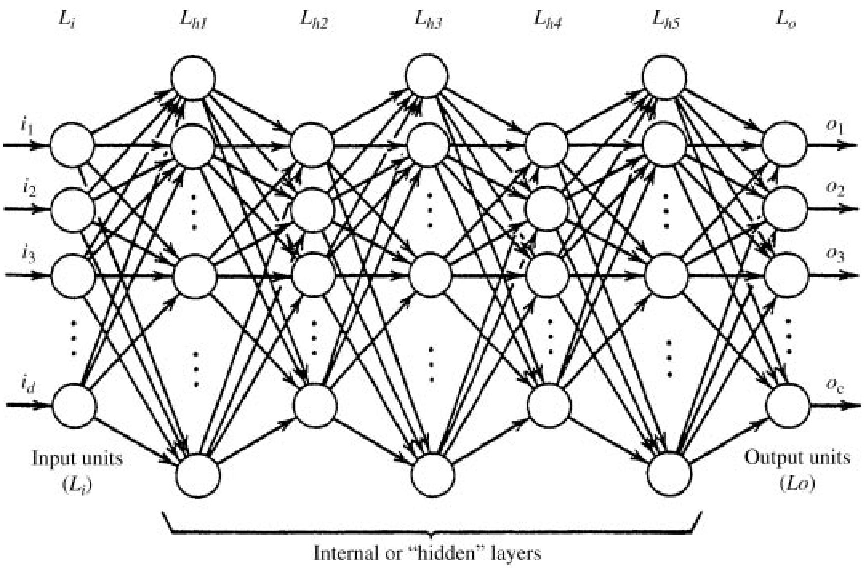


Figure 11. The typical feed-forward network, consisting of layers of simple units. (Adapted from Ref. 1).

Backpropagation - A Multistep Procedure for Training FF Networks. Beginning with an initial (possibly random) weight assignment for a three-layer feed-forward network, proceed as follows:

Step 1: Present input $\underline{x}^p$, form outputs, o_i, of all units in network.
Step 2: Update w_{ji} for output layer.
Step 3: Update w_{ji} for hidden layer(s).
Step 4: Stop if updates are insignificant or error is below a preselected threshold, otherwise proceed to Step 1.

This process leads to an adjustment scheme based on *backpropagation*. A summary of the GDR equations is given below in Table 1.

Hopfield (Recurrent) Networks for Pattern Recognition

Hopfield (42,43) characterized a neural computational paradigm for using a neural net as an autoassociative memory. The following variables are defined:

o_i: the output state of the ith neuron
α_i: the activation threshold of the ith neuron
w_{ij}: the interconnection weight (i.e., the strength of the connection FROM the output of neuron j TO neuron i).

Thus, $\sum_j w_{ij}o_j$ is the total input or activation (net_i) to neuron i. Typically, $w_{ij} \in R$, although other possibilities (e.g., binary interconnections) are possible. With the constraints developed below, for a d-unit network there are $\frac{d(d-1)}{2}$ possibly nonzero and unique weights.

In the Hopfield network, every neuron is allowed to be connected to all other neurons, although the value of w_{ij} varies (it may also be zero to indicate no unit interconnection). To avoid false reinforcement of a neuron state, the constraint $w_{ii} = 0$ is also employed. The w_{ij} values, therefore, play a fundamental role in the structure of the network. In general, a Hopfield network has significant interconnection (i.e., practical networks seldom have sparse W matrices, where $W = [w_{ij}]$).

Table 1. Summary of the GDR Equations for Training Using Backpropagation

(pattern) error measure:	$E^p = \frac{1}{2}\sum_j \left(t_j^p - o_j^p\right)^2$
(pattern) weight correction	$\Delta^p w_{ji} = \epsilon \delta_j^p \tilde{o}_i^p$
(output units)	$\delta_j^p = \left(t_j^p - o_j^p\right) f_j'\left(net_j^p\right)$
(internal units)*	$\delta_j^p = f_j'(net_j^p) \sum_n \delta_n^p w_{nj}$
	*where δ_n^p are from next layer (L^{k+1})
output derivative (assumes sigmoidal characteristic)	$f_j'(net_j^p) = o_j^p(1 - o_j^p)$

Network Dynamics, Unit Firing Characteristic, and State Propagation. A simple form for Hopfield neuron firing characteristics is the nonlinear threshold device

$$o_i = \begin{cases} 1 & \text{if } \sum_{j;\, j \neq i} w_{ij} o_j > \alpha_i \\ 0 & \text{otherwise} \end{cases}$$

Notice the neuron activation characteristic is nonlinear. Commonly, the threshold $\alpha_i = 0$. Viewing the state of a d-neuron Hopfield network at time (or iteration) t_k as an $d \times 1$ vector, $\underline{o}(t_k)$, the state of the system at time t_{k+1} (or iteration $k+1$ in the discrete case) may be described by the nonlinear state transformation

$$W\underline{o}(t_k) \overset{*}{\Rightarrow} \underline{o}(t_{k+1})$$

where the $\overset{*}{\Rightarrow}$ operator indicates the element by element state transition characteristic used to form $\underline{o}(t_{k+1})$. The model may be generalized for each unit to accommodate an additional vector of unit bias inputs.

The network state propagation suggests that the unit transitions are synchronous, that is, each unit, in lockstep fashion with all other units, computes its net activation and subsequent output. Although this is achievable in (serial) simulations, it is not necessary. Also empirical results have shown that it is not even necessary to update all units at each iteration. Surprisingly, network convergence is relatively insensitive to the fraction of units (15–100%) updated at each step.

Hopfield Energy Function and Storage Prescription. For the case of $\alpha_i = 0$, stable (stored) states correspond to minima of the following energy function

$$E = -\left(\frac{1}{2}\right) \sum \sum_{i \neq j} w_{ij} o_i o_j$$

which leads to the rule for determination of w_{ij} and a set of desired stable states $\underline{o}^s, s = 1, 2, \ldots n$, (i.e., the training set (stored states) $H = \{\underline{o}^1, \underline{o}^2, \ldots, \underline{o}^n\}$) as

$$w_{ij} = \sum_{s=1}^{n} (2o_i^s - 1)(2o_j^s - 1) \quad i \neq j$$

(with the previous constraint $w_{ii} = 0$). The convergence of the network to a stable state involves the Hamming distance between the initial state and the desired stable state. Different stable states that are close in Hamming distance are undesirable, because convergence to an incorrect stable state may result. Reference 42 suggests that an n-neuron network allows approximately $0.15n$ stable states; other researchers have proposed more conservative bounds (44).

Hopfield PR Example: Character Recall. Figure 12 shows a Hopfield network used as associative memory for recall of character data. A 10 × 10 pixel array is used to represent the character, yielding 100 pixels. Each pixel value is the state of a single, totally interconnected unit in a Hopfield network. Thus, the network consists of 100 units and approximately 100 × 100 interconnection weights. The network was trained using characters "A,' "C,' "E," and "P.' The top row of Fig. 12 shows initial states for the network; these are distorted patterns corresponding to the training patterns. Succeeding rows show the state evolution of the network. Note that the network converged to elements of H in at most two iterations in this example.

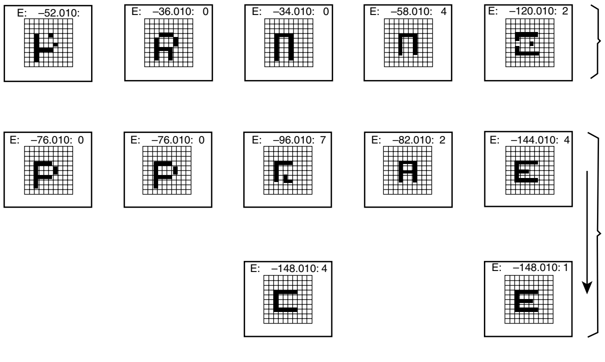


Figure 12. Use of a Hopfield network for character assocaition/ completion/recognition (Adapted from Ref. 2).

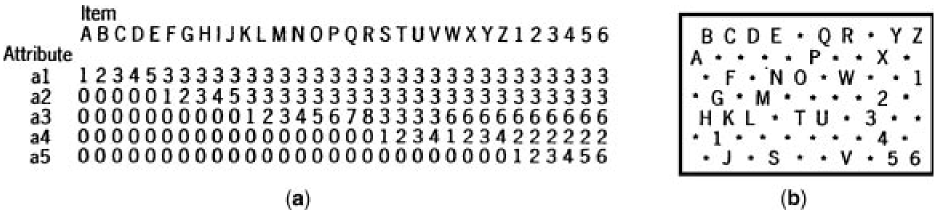
Kohonen Self-Organizing Feature Maps (SOFMs)

Kohonen (45) and Kangas et al. (46) have shown an alternative neural learning structure involving networks that perform dimensionality reduction through conversion of feature space to yield *topologically ordered* similarity graphs or maps or clustering diagrams (with potential statistical interpretations). In addition, a lateral unit interaction function is used to implement a form of local competitive learning.

1-D and 2-D spatial configurations of units are used to form feature or pattern dimensionality reducing maps. For example, a 2-D topology yields a planar map, indexed by a 2-D coordinate system. Of course, 3-D and higher-dimensional maps are possible. Notice each unit, regardless of the topology, receives the input pattern $\underline{x} = (x_1, x_2 \ldots x_d)^T$ in parallel. Considering the topological arrangement of the chosen units, the d-dimensional feature space is mapped into 1-D, 2-D, 3-D, and so on. The coordinate axes used to index the unit topology, however, have no explicit meaning or relation to feature space. They may, however, reflect a similarity relationship between units in the reduced dimensional space, where topological distance is proportional to dissimilarity.

Choosing the dimension of the feature map involves engineering judgment. Some PR applications naturally lead to a certain dimension; for example, a 2-D map may be developed for speech recognition applications, where 2-D unit clusters represent phonemes (47). The dimensions of the chosen topological map may also influence the training time of the network. Once a topological dimension is chosen, the concept of a network neighborhood (or cell or bubble) around each neuron may be introduced. The neighborhood, denoted N_c, is centered at neuron u_c, and the cell or neighborhood size (characterized by its radius in 2-D, for example) may vary with time (typically in the training phase). For example, initially N_c may start as the entire 2-D network, and the radius of N_c shrinks as iteration (described subsequently) proceeds. As a practical matter, the discrete nature of the 2-D net allows the neighborhood of a neuron to be defined in terms of nearest neighbors (e.g., with a square array the four nearest neighbors of u_c are its N, S, E, and W neighbors; the eight nearest neighbors would include the "corners").

Training the SOFM. Each unit, u_i, in the network has the same number of weights as the dimension of the input vector and receives the input pattern $\underline{x} = (x_1, x_2 \ldots x_d)^T$ in parallel. The goal of the self-organizing network, given a large, unlabeled training set, is to have individual neural clusters self-organize to reflect input pattern similarity. Defining a weight vector for neural unit u_i as $\underline{m}_i = (w_{i1}, w_{i2}, \ldots w_{id})^T$, the overall structure may be viewed as



Attribute \ Item	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6
a1	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
a2	0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
a3	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6	6
a4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2	2	2
a5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6

(a)

```
 B C D E • Q R • Y Z
A • • • • P • • X •
 • F • N O • W • • 1
• G • M • • • • 2 •
 H K L • T U • 3 • •
• 1 • • • • • • 4 •
 • J • S • • V • 5 6
```

(b)

Figure 13. Sample results using a 2-D Kohonen SOFM for a 5-D feature case involving uppercase characters (Adapted from 48). Part (a) showns the extraced features for each character. Part (b) shows the resulting map.

an array of matched filters, which competitively adjust unit input weights on the basis of the current weights and goodness of match. A useful viewpoint is that each unit tries to become a matched filter in competition with other units.

Assume the network is initialized with the weights of all units chosen randomly. Thereafter, at each training iteration, denoted k for an input pattern $\underline{x}(k)$, a distance measure $d(\underline{x}, \underline{m}_i)$ between $\underline{x}$ and $\underline{m}_i \, \forall i$ in the network is computed, which may be an inner product measure (correlation), Euclidean distance, or another suitable measure. For simplicity, we proceed using the Euclidean distance. For pattern $\underline{x}(k)$, a *matching phase* is used to define a "winner" unit u_c, with weight vector $\underline{m}_c$, using

$$\|\underline{x}(k) - \underline{m}_c(k)\| = \overset{\min}{i} \{\|\underline{x}(k) - \underline{m}_i(k)\|\}$$

Thus, at iteration k, given $\underline{x}$, c is the index of the best matching unit, which affects all units in the currently defined cell, bubble, or cluster surrounding u_c, $N_c(k)$ through the global network *updating phase* as follows

$$\underline{m}_i(k+1) = \begin{cases} \underline{m}_i(k) + \alpha(k)[\underline{x}(k) - \underline{m}_i(k)] & i \in N_c \\ \underline{m}_i(k) & i \notin N_c \end{cases}$$

The updating strategy bears a strong similarity to the c-means algorithm. $d(\underline{x}, \underline{m}_i)$ is decreased for units inside N_c by moving $\underline{m}_i$ in the direction $(\underline{x} - n_i)$. Therefore, after the adjustment, the weight vectors in N_c are closer to input pattern $\underline{x}$. Weight vectors for units outside N_c are left uncharged. The competitive nature of the algorithm is evident as after the training iteration units outside N_c are *relatively* further from $\underline{x}$. That is, there is an opportunity cost of not being adjusted. Again, α is a possibly iteration-dependent design parameter.

The resulting accuracy of the mapping depends on the choices of N_c, $\alpha(k)$ and the number of iterations. Kohonen cites the use of 10,000–100,000 iterations as typical. Furthermore, $\alpha(k)$ should start with a value close to 1.0 and gradually decrease with k. Similarly, the neighborhood size, $N_c(k)$, deserves careful consideration in algorithm design. Too small a choice of $N_c(0)$ may lead to maps without topological ordering. Therefore, it is reasonable to let $N_c(0)$ be fairly large (Kohonen suggests one half the diameter of the map) shrinking $N_c(k)$ (perhaps linearly) with k to the fine-adjustment phase, where N_c only consists of the nearest neighbors of unit u_c. Of course, a limiting case is where $N_c(k)$ becomes one unit. Additional details of the self-organizing algorithm are summarized in the cited references.

Example: SOFM Application to Unsupervised Learning. - Figure 13(48) shows sample results for a 5-D feature vector case. Uppercase characters are presented as unlabeled training data to a 2-D SOFM. Figure 13(a) shows the unlabeled training set samples H_u; Fig. 13(b) shows the self-organized map resulting from the algorithm. As evidenced by Fig. 13(b), 2-D clustering of the different dimensionality-reduced input patterns occurs. As in other learning examples, vectors were chosen randomly from H_u at each iteration. $\alpha(k)$ decreased linearly with k from $0.5 (= \alpha(o))$ to 0.04 for $k \leq 10,000$. Similarly, for this simulation, the 2-D map was chosen to be of hexagonal structure with 7×10 units. For $k \leq 1000$, the radius of N_c decreased from 6 (almost all of the network) to 1 (u_c and its six nearest neighbors).

Picture Processing- See *Image Processing.*

FURTHER READING

Work on various aspects of PR continues to cross-pollenate journals. Useful sources include: *Pattern Recognition Letters, Pattern Recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Transactions on Systems, Man and Cybernetics, IEEE Transactions on Geoscience and Remote Sensing, IEEE Transactions on Neural Networks, and Image and Vision Computing.*

BIBLIOGRAPHY

1. R. J. Schalkoff, *Pattern Recognition: Statistical, Syntactic and Neural Approaches*. New York: Wiley, 1992.
2. R. J. Schalkoff, *Digital Image Processing*. New York: Wiley, 1989.
3. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
4. P. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
5. K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic Press, 1972.
6. S. T. Bow, *Pattern Recognition*. New York: Marcel Dekker, 1984.
7. S. Watanabe, *Pattern Recognition: Human and Mechanical*. New York: Wiley, 1985.
8. Y. T. Chien, *Interactive Pattern Recognition*. New York: Marcel Dekker, 1978.

9. E. A. Patrick, *Fundamentals of Pattern Recognition*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

10. C. W. Therrien, *Decision Estimation and Classification: An Introduction to Pattern Recognition and Related Topics*. New York: Wiley, 1989.

11. R. A. Fisher, The use of multiple measurements in taxonomic problems, reprinted in *Contributions to Mathematical Statistics*. New York: Wiley, 1950.

12. Y. C. Ho and R. L. Kayshap, An algorithm for linear inequalities and its application, *IEEE Trans. Elec. Comp.*, **EC-14**: 683–688, 1965.

13. K. Fukunaga and D. R. Olsen, Piecewise linear discriminant functions and classification errors for multiclass problems, *IEEE Trans. Inform. Theory*, **IT-16**: 99–100, 1970.

14. A. K. Jain and R. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

15. G. B. Coleman and H. C. Andrews, Image segmentation by clustering, *Proc. IEE*, **67**: 773–785, 1979.

16. J. Bryant, On the clustering of multidimensional pictorial data, *Pattern Recognition*, **11**: 115–125, 1979.

17. R. K. Blashfield, M. S. Aldenderfer, and L. C. Morey, Cluster analysis software, in P. R. Krishniah and L. N. Kanal (eds.), Handbook of Statistics, Vol. 2. Amsterdam, The Netherlands: North Holland, 1982, pp. 245–266.

18. R. C. Dubes and A. K. Jain, Clustering techniques: The user's dilemma, *Pattern Recognition*. **8**: 247–260, 1976.

19. K. S. Fu, *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

20. J. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison Wesley, 1974.

21. R. C. Gonzalez and M. G. Thomason, *Syntactic Pattern Recognition*. Reading, MA: Addison-Wesley, 1978.

22. L. Miclet, *Structural Methods in Pattern Recognition*. New York: Springer-Verlag, 1986.

23. T. Pavlidis, *Structural Pattern Recognition*. New York: Springer-Verlag, 1977.

24. K. S. Fu, A step towards unification of syntactic and statistical pattern recognition, *IEEE Trans. Pattern Anal. Machine Intell.*, **PAMI-8**(3): 398–404, 1986.

25. H. S. Don and K. S. Fu, A syntactic method for image segmentation and object recognition, *Pattern Recognition*, **18**(1): 73–87, 1985.

26. J. E. Hopcroft and J. D. Ullman, *Formal Languages and Their Relation to Automata*. Reading, MA: Addison-Wesley, 1969.

27. R. N. Moll, M. A. Arbib, and A. J. Kfoury (eds.), *An Introduction to Formal Language Theory*. New York: Springer-Verlag, 1988.

28. H. C. Lee and K. S. Fu, A stochastic syntactic analysis procedure and its appplication to pattern classification, *IEEE Trans. Comput.*, **C-21**(7): 660–666, 1972.

29. A. Sanfeliu and K. S. Fu, A distance measure between attributed relational graphs for pattern recognition, *IEEE Trans. SMC*, **SMC-13**(3): 353–362, 1983.

30. L. G. Shapiro and R. M. Haralick, A metric for comparing relational descriptions, *IEEE T-PAMI-*, **7**: 90–94, 1985.

31. R. J. Schalkoff, *Artificial Neural Networks*. New York: Mc-Graw Hill, 1997.

32. J. A. Anderson and E. Rosenfeld (eds.), *Neurocomputing: Foundations of Research*. Cambridge, MA: MIT Press, 1988.

33. D. E. Rummelhart and J. L. McClelland, *Parallel Distributed Processing - Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge, MA: MIT Press, 1986.

34. D. E. Rummelhart and J. L. McClelland, *Parallel Distributed Processing - Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*. Cambridge, MA: MIT Press, 1986.

35. R. Rosenblatt, *Principles of Neurodynamics*. New York: Spartan Books, 1959.

36. M. Minsky and S. Papert, *Perceptrons-An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.

37. N. J. Nilsson, *Learning Machines*. New York: McGraw-Hill, 1965. (Revised as *Mathematical Foundations of Learning Machines*. San Mateo, CA: Morgan-Kaufmann, 1989.)

38. B. Widrow and M. E. Hoff, Adaptive switching circuits, 1960 IRE WESCON Conv. Record, Part 4, Aug. 1960, pp. 96–104 (reprinted in Anderson and Rosenfeld 1988).

39. W. Y. Huang and R. P. Lippmann, Comparison between neural net and conventional classifiers, *Proc. IEEE Int. Conf. Neural Networks*, **IV**: 485–493, 1987.

40. B. Widrow and M. A. Lehr, 30 years of adaptive neural networks: perceptron, mada-line and backpropagation, *Proc. IEEE*, **78**(9): 1415–1442, 1990.

41. B. Widrow and R. G. Winter, Neural nets for adaptive filtering and adaptive pattern recognition, *IEEE Comp.*, **21**: 25–39, 1988.

42. J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci.*, **79**(Biophysics): 2554–2558, 1982.

43. J. J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Natl. Acad. Sci.*, **81**(Biophysics): 3088–3092, 1984.

44. Y. S. Abu-Mostafa and J. M. St. Jacques, Information capacity of the hopfield model, *IEEE Trans. Inform. Theory*, **IT-31**(4): 461–464, 1985.

45. T. Kohonen, *Self-Organization and Associative Memory*. Berlin: Springer-Verlag, 1984.

46. J. A. Kangas, T. Kohonen, and J. T. Laaksonen, Variants of self-organizing maps, *IEEE Trans. Neural Networks*, **1**(1): 93–99, 1990.

47. B. D. Shriver, artificial neural systems, *IEEE Comp.*, **21**: 3, 1988.

48. T. Kohonen, Self-organizing feature maps, tutorial course notes from 1988 Conference on Neural Networks. San Diego, CA, 1988. (Accompanying videotape available from the Institute of Electrical and Electronics Engineers, Inc., 345 E. 47th St., New York 10017.)

Robert J. Schalkoff
Clemson University
Clemson, South Carolina

REASON MAINTENANCE SYSTEMS: TOOLS FOR FOUNDATIONS-BASED BELIEF REVISION

INTRODUCTION

The development of computational systems that can model and solve problems in dynamic nonstructured environments, which are faced with unexpected changes and incomplete knowledge, or in intrinsically distributed scenarios, where information from multiple disparate or even conflicting sources needs to be merged, requires appropriate knowledge representation and reasoning (KRR) systems. The KRR systems that can handle sets of dynamic beliefs are sensitive to communicated and perceived changes in the environment and may drop current beliefs in face of new findings or disregard new data in conflict with existing firm beliefs.

When knowledge is represented by beliefs, some classic reasoning properties no longer hold. In a classic deductive reasoning system, which is also known as a monotonic reasoning system, what is deduced from an initial set of facts still is deduced from an enlarged set of those facts (i.e., the set of logic consequences will hold forever). In a nonmonotonic reasoning system, what is inferred from an initial set of beliefs is a new set of beliefs that may or may not be revised in the future. Although belief revision and nonmonotonic reasoning are different, both deal with beliefs and incomplete knowledge. Belief revision handles the dynamics of belief change (i.e., how an agent changes its current belief state in face of new information) and nonmonotonic reasoning draws conclusions from a set of beliefs.

As a KRR system gathers information to reason about, it has to update its belief space (1). A rational agent, when faced with new data, selects the resulting epistemic state based on the preferences established between its belief states. To establish the preferences between belief states, an agent can rely on symbolic or numeric approaches. In symbolic approaches, the revision is based on the logic content of the propositions—the revision is performed according to the epistemic relevance of the involved propositions; in the numeric approaches, the logic content of formulas is irrelevant—the propositions are ordered according to their certainty (2), possibility (3), credibility (4) or probability (5). The goal of any one of these approaches is to establish the "relative" strength between the represented beliefs and, thus, allow the agent to choose the resulting preferred belief state.

Most belief revision models, symbolic or numeric, follow some rationality principles (6): (*1*) consistency of the information—the agent's resulting belief state must be consistent, (*2*) minimal changes—the changes between consecutive belief states should be minimal, (*3*) preservation of the information—the resulting belief state should preserve as much information as possible from the previous belief state, and (*4*) priority of the most recent information—the resulting belief state should contain the information that triggered the belief revision. Depending on the problem domain features, different principles can be adopted. Whereas in dynamic environments the preference goes to the most recent information (i.e., existing beliefs are dropped in face of new findings), in distributed scenarios, where the merge of data from multiple sources is performed, the preference is based on other principles, such as the reliability of the sources (4), and so on.

Two main approaches to the modeling of epistemic states (7) include the theory of belief revision based on foundations (8) and the theory of belief revision based on coherence (6). In the foundations approach, a belief holds as long as the system finds a justification in its support. In the coherence approach, a belief holds as long as it is coherent with the remaining beliefs of the system. We will only address the first—the theory of belief revision based on foundations—because it is the theory behind the reason maintenance systems (RMS), which are also known as truth maintenance systems (TMS) or belief maintenance systems (BMS).

REASON MAINTENANCE SYSTEMS

The foundations-based theory regards a belief as a temporary knowledge item, which is exclusively supported by valid foundations. This theory distinguishes between core beliefs or foundations—also called base beliefs or hypothesis (9)—that have an independent standing and derived beliefs (1). According to this theory, first, all beliefs without a valid justification are abandoned, and then, the new beliefs (either self-supported or derived beliefs) are added to the system.

RMSs are computational units supported by the foundations-based theory. According to this theory, a belief is always a justified belief: It is either an assumption (a self-supporting core belief) or a derived belief with at least one valid justification. Typically, KRR systems that require consistency maintenance have two main modules: the problem solver and the RMS. The problem-solver module submits assumptions (core beliefs) as well as conclusions (derived beliefs) together with their respective foundations to the RMS; the RMS stores and associates with each belief (core or derived belief) the corresponding set of supporting foundations. The RMS maintains the consistency of the overall reasoning by keeping, for each represented belief, the supporting justifications. The relationship between the problem solver and the RMS is Master/Slave.

Two main categories of TMS are as follows: single- and multiple-context systems. According to Mason (10), these two categories result from the different approaches each type adopts to represent the dependencies between beliefs: In the single context systems, each belief is associated to the beliefs that directly generated it—it is the case of the justification-based TMS (JTMS) (8) or the logic-based TMS (LTMS) (11). In the multiple-context counterparts,

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

each belief is associated with the minimal set of assumptions from which it can be inferred—it is the case of the assumption-based TMS (ATMS) (12) or the multiple belief reasoner (MBR) (9).

Justification-Based TMS

In 1979, Jon Doyle presented the first reason maintenance system—the JTMS (8)—which was designed as an independent module for belief maintenance (13). It works with a single context—the current set of beliefs. The JTMS records *nodes*, which represent the existing beliefs, and *justifications*, which represent the dependencies between the represented beliefs, and, then, labels each node as *in*–believed— or *out*–disbelieved—according to the node's supporting justification. The JTMS can (*1*) add nodes to represent new beliefs, (*2*) add and retract justifications that represent the reasons for holding a belief, and (*3*) identify and mark every contradiction as a *nogood* node. By maintaining the justifications of each belief, the JTMS can establish at all times the current set of believed nodes.

In a JTMS, two sorts of nodes are as follows: ordinary nodes and contradiction nodes. Contradiction nodes are explicitly labeled as such by the user or by the problem-solver module. Ordinary nodes have no special meaning to RMS. In contrast, contradiction nodes have a crucial meaning to the JTMS (14). They should be excluded from any belief context.

Although a node may have multiple justifications, the labeling algorithm selects just one—the supporting justification. As a result, each node *in* has only one set of supporting nodes, which are the nodes that the JTMS uses to establish its belief state.

The foundations set of a node is made of all its antecedents: The direct antecedent nodes, the antecedents of its antecedents, and so forth (the transitive closure of the set of antecedents). This set is also called the node's well-founded argument. A node's set of consequents is obtained through the transitive closure of its consequents. The system builds a dependency network that includes all represented propositions (nodes) and available justifications (directed arcs).

The addition of a new justification may have a profound impact in a JTMS, because it triggers the labeling algorithm: Contradiction nodes may become ordinary nodes and *vice-versa*. The most generic type of justification in a JTMS is the so-called nonmonotonic justification. A nonmonotonic justification of a proposition c is an ordered pair composed of two sets of propositions A_M and A_{NM} such that $<(A_N, A_{NM}) \rightarrow c>$ where c is the consequent, A_M is the monotonic set of antecedents $(a_1, \ldots, a_n)$, or *in*-list of c and A_{NM} is the nonmonotonic set of antecedents $(a_{n+1}, \ldots, a_m)$ or *out*-list of c. Given such a nonmonotonic justification, the JTMS labels proposition c as *in* if all propositions of A_M are *in* and all propositions of A_{NM} are *out*. In other words, if the system believes in the propositions of the first set and has no reasons to believe in the propositions of the second set, then it has found a valid justification to believe in the consequent proposition. A justification may support several types of nodes: (*1*) premises, when both the monotonic set and nonmonotonic set of antecedent propositions are empty; (*2*) deductions, when the nonmonotonic set of antecedent propositions is empty; (*3*) assumptions, when both the monotonic set and nonmonotonic set of antecedent propositions are nonempty. A node has the following structure: <*Node—Id, Proposition, Justification, Label*>, where *Node-Id* is the node identifier, *Proposition* is the problem solver proposition, *Justification* specifies the corresponding *in*-list and *out*-list ($<A_M, A_{NM}>$), *Label* holds the proposition belief state (*{believed, disbelieved / unknown}*).

When the JTMS uses such a type of justification to label a node, the resulting state can either be *in* or *out*. If the node is labeled *in*, then the set of supporting nodes is made of the union of the elements of both A_M and A_{NM}; otherwise, when the node is *out*, the set of supporting nodes is made of one *out* node from A_M or one *in* node from A_{NM}.

Assumption-Based TMS

In 1986, Johan de Kleer (12) proposes a new type of reason maintenance system—the ATMS. For de Kleer, assumptions are self-supported propositions that play the role of foundational nodes, which are believed as long as the system is not confronted with contradictory evidences that play the role of foundational nodes. Each derived proposition has for foundations the set(s) of assumptions from which it was inferred. Each set of foundations/assumptions is called an environment. The ATMS represents all propositions—assumptions, premises, and derived propositions—as nodes and keeps record of the respective sets of foundations, which include the supporting sets of assumptions/hypothesis, in a structure called label. For each proposition in the knowledge base, there is a node in the ATMS, and for each justification the ATMS records a dependency describing how the node was inferred from other nodes. A node has the following structure: <*Proposition, Type, Label, State*>, where *Proposition* is the proposition identification, *Type* specifies the node type (*{assumption node, inferred node, premise node}*), *Label* contains all sets of assumptions from which the proposition was inferred, and *State* represents proposition belief state (*{believed, disbelieved}*).

In an ATMS, justifications are monotonic and define the dependency between a set of antecedent propositions and the consequent proposition. Formally, a justification is of the type: $a_1, \ldots, a_n \rightarrow c$ where $a_1, \ldots, a_n$ is the set of antecedent nodes and c is the consequent node.

The goal of an ATMS is to keep track of all consistent contexts through the (*1*) creation of assumption nodes whenever the problem solver decides to adopt new assumptions/hypothesis; (*2*) creation of ordinary nodes when the problem solver infers new propositions; and (*3*) addition of new justifications to existing nodes when the problem solver finds a new way of inferring the corresponding propositions (15).

On reception of a new node justification, the ATMS triggers the labeling update algorithm. First, it updates the label of the consequent node by adding the new supporting set of assumptions composed of the union of the labels of the antecedent nodes. The new supporting set of assumptions is the minimal set from which, using the provided justification, it is possible to infer the node. Next, recursively, it updates the dependency network by relabeling all nodes that depend on this consequent node. Finally, the

ATMS sets the belief state according to the determined foundations: (*1*) a node is *believed* when it has a non-empty label, that is, the system identified a consistent set of assumptions (foundations) from which it is possible to infer the node; and (*2*) a node is *disbelieved* when its label is empty, that is, the system was unable to find a reason to believe in the proposition.

Logic-Based TMS

The LTMS is a reason-maintenance system proposed by McAllester (11). This reason-maintenance system computes truth values assignments (*true*, *false*, and *unknown*) rather than belief statuses (*believed* or *disbelieved*) to all represented nodes and does not use justifications to establish the dependencies between nodes; instead, it uses logic clauses to specify the constraints between the nodes.

The LTMS approach is similar to JTMS except that in the LTMS: (*1*) the nodes assume no relationships among them except the ones explicitly stated in justifications and (*2*) it is not possible to represent simultaneously a proposition and its negation because it throws a contradiction. If the latter case happens, then the dependency network has to be reconstructed.

Each node in a LTMS has two labels: *Lt* and *Lf*. The label *Lt* can take values *true* or *unknown*, which indicates that the system knows the node is true or that it does not know if the node is true. Similarly, the label *Lf* can take values *true* or *unknown*, which indicates that the system knows the negated node is true or that it does not know whether the negated node is true (16). As a result, there are four possibilities: the node is *true*, the negated node is *true*, the node has an *unknown* truth value or the node is a *contradiction* (both the node and the negated node labels are set to *true*).

Two approaches are used to handle these labels. One is identical to the current context approach applied in the JTMS, where the label values are evaluated according to the current assumption set. The other is similar to the assumption-based approach, where the label on a literal reflects all combinations of assumptions that make it true. Contradiction handling relies in the dependency-directed backtracking algorithm and in the maintenance of a *nogood* database that contains all detected contradictions (when both node and negated node labels are set to true in one assumption set) (16).

BELIEF REVISION SYSTEMS

Belief revision systems (BRSs) are KRR systems that assume, infer, represent, handle, and maintain beliefs. BRSs can adopt a *reason maintenance system* as an external tool to represent, handle, and maintain beliefs or, alternatively, be designed to include the ability "to detect contradictions, identify the culprits and readjust their knowledge bases as to eliminate any contradiction (9)".

Belief Revision Supported by JTMS

The problem solver commands the JTMS operation through the submission of new nodes and the addition or removal of justifications. These operations trigger the JTMS consistency maintenance process—labeling algorithm—which starts with the premise and assumption nodes and uses the justifications to find noncircular arguments in support of every node. These noncircular arguments are called well-founded belief-support justifications.

When the JTMS receives a new justification for a represented node, the updating process depends on the node belief status. If the node is already believed (*in*) and regardless of the validity of the justification, Then few changes occur. However, if the node is disbelieved (*out*) and the new justification is valid, then the labels of this node and all its consequents need to be updated. The JTMS builds a new list that contains the node and all its consequents and initializes/resets their labels to *nil*. Next, using only well-founded nodes, it verifies whether the current node has at least one valid justification. In case of success, it updates the node's label to *in*; otherwise, if no valid justification was found in support of the node, the node is labeled *out*. Then, sequentially, it applies the same procedure to the remaining nodes in the list, to establish their well-founded support: either *in* or *out*. Sometimes, because of existing circularities, some nodes remain unlabeled (*nil*). In these cases, the JTMS applies a relaxation procedure with the intent to label all nodes.

Finally, the JTMS looks for contradictions/inconsistencies. If a contradiction is identified, then the system tries immediately to find the culprit. The blame can be placed on the decision that caused the contradiction or on the last decision, leading to dependency-directed backtracking or chronological backtracking, respectively. The JTMS performs dependency-directed backtracking. When the JTMS finds a believed node in contradiction, it creates a contradiction node and its justification, which contains the identified set of incompatible nodes. Then, it invokes the dependency-directed backtracking algorithm to find and remove the minimal set of assumptions that puts the contradiction node *out*. This procedure consists of (*1*) examining the justification of the contradiction node, (*2*) identifying the underlying set of assumption nodes, (*3*) registering the resulting set of assumptions as inconsistent, and (*4*) changing the label of one of these assumptions. This operation changes the label of the contradiction node to *out*. Finally, it reports back to the problem solver all the changes regarding the beliefs involved in the process.

Belief revision occurs when the responsible node for the contradiction is identified and its label is changed, that is, when an assumption from the set of assumptions that supports the contradiction node is selected for culprit. This behavior is intrinsic to the JTMS.

Belief Revision Supported by ATMS

A belief revision system that relies on an ATMS typically determines and keeps the full set of consequents that are consistent with the represented assumptions/propositions—multiple belief contexts. Because the set of represented propositions is intrinsically dynamic, these systems usually implement a forward-chaining inference mechanism as well as specific knowledge representation (which include bit sets and efficient pattern matching algorithms). The inference rules are ordered according to their respective level of dependency/depth. The deeper the rule is, the greater is its dependency on other rules. A rule with depth D should be executed only after all rules with depth $D-1, \ldots, 1$ have been executed, and rules with equal depth should be executed according to the number of assumptions its antecedent/precondition nodes depend on, starting with those that depend on fewer assumptions.

The problem solver scrolls the depth-ordered rule list and, for each rule, it queries the ATMS about the statuses of the antecedent/precondition nodes. Whenever they are all believed, the problem solver fires the rule and sends the new justification of the consequent/conclusion node to the ATMS. De Kleer (17) calls these instantiated rules consumers because, once fired, they cease to exist and are converted into justifications of the consequent/conclusion node. The ordered set of rules ready to be fired is established through an appropriate scheduling algorithm (17) that intends to find efficiently the most generic context of each node and the most specific version of a contradiction/inconsistency. Simultaneously, the ATMS is constantly checking the consistency between the already represented and the newly received data.

The problem solver has specific knowledge to detect contradictions/inconsistencies. This knowledge is represented as consistency maintenance rules (or inconsistency detection rules). Whenever the problem solver explores a new space/time search point, the consistency maintenance mechanism is automatically triggered to ensure the consistency of all contexts under evaluation. As soon as an inconsistency is detected, it is immediately reported to the ATMS. The ATMS registers the contradictory/inconsistent set of assumptions or *nogood* and removes it from the labels of all represented nodes.

The detection of a contradiction/inconsistency leads to belief revision: (*1*) registration of the contradictory/inconsistent set of assumptions as a nogood, (*2*) removal of the contradictory/inconsistent set of assumptions from the labels of all represented nodes, (*3*) identification of the culprits, and (*4*) removal of the culprits. Whereas the first two stages are intrinsic to the ATMS operation, the identification of the culprit assumptions is external to the ATMS and depends on the existence of specific knowledge on the problem solver side. The culprit assumptions are abandoned, that is, they become disbelieved propositions.

Multiple Belief Reasoner

The MBR of (9) is a BRS system that handles multiple belief contexts simultaneously. It was developed on top of a logic system that keeps track of the dependencies among propositions and knows how to propagate them. The logic system used is the SWM of Shapiro, Wand, and Martins, which is based on relevance logic (18).

The SWM system associates each proposition with one or more supporting triple. Each triple is of the form <*OT, OS, RS*> where *OT* is the *origin tag*, *OS* represents the *origin set*, and *RS* the *restriction set*. The *OT* identifies how the proposition was created (*{hypothesis, derived, extended}*); *OS* holds the set of hypothesis that was actually used to derive the proposition; *RS* contains all known sets that are inconsistent with the *OS*. A proposition derived from multiple inferences will have multiple supporting triples. Only the *RS* field of a triple may change with time to accommodate new identified contradictions; the *OT* and *OS* fields, once created, remain unchanged throughout the system's operation.

The propositions are added to the knowledge base according to the inference rules of SWM together with its supporting triples. MBR uses the notion of context or set of hypotheses. Each context defines a belief space, which holds the context's set of hypotheses as well as the set of all the propositions that are exclusively derived from this context. At each moment, there is one active context, and all knowledge base operations are defined over the belief space of the current context.

When a contradiction is detected, the origin sets of the contradictory propositions are inspected, and their union becomes a known inconsistent set of hypothesis. All propositions with origin sets that contain the recently identified inconsistent set of hypothesis must update their restriction set as to include it. Finally, in the case of MBR, belief revision is based on the preferences established between the represented beliefs, that is, MBR selects from the set of assumptions involved in the contradiction the least-preferred assumptions and blames them for the contradiction.

CONCLUSIONS

Beliefs, which are volatile by nature, require a rational and justified endorsement. The foundations-based belief-revision systems allow the representation of beliefs by storing and maintaining the reasons behind their representation. These reasons form a dependency network between the represented beliefs, ensuring that, whenever a revision occurs, all repercussions will be correctly propagated. According to Doyle (19), these systems allow the representation and modification of beliefs, the storage of the inference relations between beliefs as well as restrict all changes to minimal changes. Reason maintenance systems, which are also known as truth maintenance systems, are external modules specialized in these tasks.

The presented approaches are intended for scenarios where the information is dynamic, distributed, incomplete, or even eventually, incorrect. Their main goal is to maintain, in a rational and justified manner, a dynamic set of beliefs according to perceived observations (belief update), the identified contradictions (consistency maintenance), and the specified belief preferences (belief revision). The reason maintenance/truth maintenance capabilities can be provided either by an external module—JTMS, LTMS, or ATMS—or be internal to the system—MBR.

BIBLIOGRAPHY

1. F. L. Johnson and S. C. Shapiro, Dependency-directed reconsideration belief base optimization for truth maintenance systems, *Proc. of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, Menlo Park, CA, pp. 313–326, 2005.

2. G. Shafer, Belief Functions, *Readings in Uncertain Reasoning*. G. Shafer and J. Pearl, (eds.), San Francisco, CA: Morgan Kaufmann Publishers, 1990.

3. D. Dubois and H. Prade, Belief change and possibility theory, in *Belief Revision*, P. Gärdenfors, (ed.), Cambridge, MA: Cambridge University Press, pp. 142–182, 1992.

4. A. F. Dragoni and P. Giorgini, Belief revision through the belief function formalism in a multi-agent environment, in *Intelligent Agents III—Agent Theories, Vol 1193, Architectures and Languages*. J. P. Müller, M. Wooldridge, and N. R. (eds.), New York: Springer-Verlag, pp. 103–115, 1997.

5. J. Pearl, *Probabalistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

6. C. E. Alchourrón, P. Gärdenfors, and D. Makinson, On the logic of theory change: Partial meet functions for contraction and revision, *J. Symbol. Logic*, **50** (2): 510–530, 1985.

7. G. Harman, *Change in View: Principles of Reasoning*, Cambridge, MA: The MIT Press, 1986.

8. J. Doyle, A truth maintenance system, *Artifi. Intelli.*, **12** (3): 231–272, 1979.

9. J. P. Martins and S. C. Shapiro, A model for belief revision, *Artificial Intelligence*, **35** (1): 25–79, 1988.

10. C. L. Mason, ROO: A distributed toolkit for belief based reasoning agents, *Proc. of Second International Working Conference on Cooperative Knowledge Based Systems 1994*. University of Keele, UK, 1994.

11. D. McAllester, An outlook on truth maintenance, Artificial Intelligence Laboratory, AI Memo 551, Cambridge, MA: Massachusetts Institute of Technology, 1980.

12. J. de Kleer, Problem solving with the ATMS, *Artifi. Intelli.*, **28** (2): 197–224, 1986.

13. D. McAllester, Truth maintenance, *Proc. of the Eighth National Conference on Artificial Intelligence*, Menlo Park, CA, 1990.

14. J. Doyle, The ins and outs of reason maintenance, *Proc. of Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, pp. 349–351, 1983.

15. B. Malheiro and E. Oliveira, Consistency and context management in a multi-agent belief revision testbed, in *Intelligent Agents II—Agent Theories, Architectures and Languages*. M. Wooldridge, J. P. Müller, and M. Tambe, (eds.), New York: Springer-Verlag, 361–375, 1996.

16. M. Stanojevic, S. Vranes and D. Velasevic, Using truth maintenance systems: A tutorial, *IEEE Expert: Intell. Sys. Appl.*, **9** (6): 46–56, 1994.

17. J. de Kleer, An assumption-based truth maintenance system, *Artifi. Intelli.*, **28** (2): 127–162, 1986.

18. S. Shapiro, and M. Wand, *The relevance of relevance*, *Technical Report* N. 46, Computer Science Department, Indiana University, Bloomington, Indiana, 1976.

19. J. Doyle, Reason maintenance and belief revision: Foundations versus coherence theories, in *Belief Revision*, P. Gärdenfors (ed.), Cambridge, MA: Cambridge University Press, 29–51, 1992.

BENEDITA MALHEIRO
Instituto Superior de Engenharia do Porto
Porto, Portugal

WEB INTELLIGENCE (WI)

INTRODUCTION

The study of Web intelligence (WI) was first introduced in several papers and books [see Refs. (1–19)]. Broadly speaking, WI is a new direction for scientific research and development that explores the fundamental roles as well as practical impacts of artificial intelligence (AI) in the context of the Web,[1] such as knowledge representation, planning, knowledge discovery and data mining, intelligent agents, and social network intelligence, as well as advanced information technology (IT), such as wireless networks; ubiquitous devices; social networks; and data/knowledge grids; and the next generation of Web-empowered products, systems, services, and activities.

On one hand, WI applies results from existing disciplines to a totally new domain. On the other hand, WI introduces new problems and challenges to the established disciplines. WI may be considered as an enhancement or an extension of AI and IT (4). The WI technologies revolutionize the way in which information is gathered, stored, processed, presented, shared, and used through electoronization, virtualization, globalization, standardization, personalization, and portals.

The challenges of Internet computing research and development in the next decade will be WI centric, focusing on how we can intelligently make the best use of the widely available Web connectivity. The new WI technologies will be determined precisely by *human needs in a post-industrial era*; namely (2):

- *information* empowerment,
- *knowledge* sharing,
- virtual *social* communities,
- *service* enrichment, and
- practical *wisdom* development.

We observed that one of the most promising paradigm shifts in the Web will be driven by the notion of *wisdom*, and developing the World Wide Wisdom Web (the *Wisdom Web*, or W4) will become a tangible goal for WI research (1,3,7). The new generation of the WWW will enable humans to gain wisdom of living, working, and playing in addition to information search and knowledge queries.

Great potential exits for WI to make useful contributions to e-business (including e-commerce and e-finance), e-science, e-learning, e-government, e-community, and so on. Many specific applications and systems have been proposed and studied. In particular, the e-business activity that involves the end user is undergoing a significant revolution (10). The ability to track users' browsing behavior down to individual mouse clicks has brought the vendor and end customer closer than ever before. It is now possible for a vendor to personalize his product message for individual customers at a massive scale, which is called *targeted marketing* (or direct marketing) (11–13). Web mining and Web usage analysis play an important role in e-business for customer relationship management (CRM) and targeted marketing. Web mining is the use of data mining techniques to discover automatically and extract information from Web documents and services (10,14,15). A challenge is to explore the connection between Web mining and the related agent paradigm, such as Web farming, that is the systematic refining of information resources on the Web for business intelligence (16).

This article investigates various ways to study WI and potential applications. The next section describes what is the Wisdom Web, which includes ten capabilities of the Wisdom Web, and conceptual levels of WI for developing the Wisdom Web. The section after that discusses how to develop various Web-based portals, in particular, intelligent enterprise portals for e-business intelligence, by using WI technologies. Furthermore, based on the discussion, an intelligent Web-based portals-centric schematic diagram of WI-related topics is provided in this section. The section entitled Advanced topics for studying WI describes various ways for studying WI, which include the semantics in the Web and the Web as social networks, as well as proposes new approaches for developing semantic social networks. Based on the above preparation, the section on WI-Based Targeted Marketing shows how to offer advanced features that enable e-business intelligence such as targeted marketing, which is a new business model by an interactive one-to-one communication between marketer and customer, as well as deal with the scalability and complexity of the real world, efficiently and effectively, by using the knowledge grid middleware as a new infrastructure and platform. The final section provides concluding remarks.

THE WORLD WIDE WISDOM WEB (W4)

What is the Wisdom Web?

In the movie *Star Wars: Episode II*, an interesting scene is when Obi Wan Kenobi failed to locate any relevant information about a mysterious planet (where later he discovered the clone manufacturing ground), he turned to his friend for advice. His friend, who apparently knew more than the Jedi's academy knowledge banks combined, gave the following reply: Other people seek *knowledge*, but you my friend know *wisdom*.

The reply in the above scene also provides an answer to the question: What will be the next paradigm shift in the Web and the Internet? The next paradigm shift lies in the notion of *wisdom*. The goal of the new generation WI is to enable users to gain new *wisdom* of living, working,

[1] Here, the term of AI includes classic AI, computational intelligence, and soft computing.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

playing, and learning, in addition to *information* search and *knowledge* queries. Here, the word of *wisdom*, according to the Webster Dictionary (Page: 1658) (17), implies the following meanings (emphasis added):

1. The quality of being wise; knowledge, and *the capacity to make due use of it*; *knowledge of the best ends and the best means*; *discernment and judgment*; discretion; sagacity; skill; dexterity.
2. The results of wise judgments; scientific or practical truth; acquired knowledge; erudition.

In the Web context, the manifestation of wisdom can best be illustrated with a minimalist Wisdom Web example.

When the Web Offers Practical Wisdom

Imagine that you are taking your first trip to the city of Montreal. You would like to find a really nice place to spend your evening. So, you walk into a Cyber Cafe on Sherbrook Street (the only street that you can recognize), and decide to get some *practical wisdom* from a public *Wisdom Web* outlet. You log in with a user name, "Spiderman," and ask:

> What is the best night life in Montreal during this season of the year?

The Wisdom Web *thinks* for about a second or two and then responds:

> Spiderman, the hockey games are on during this season of the year. Would you like to go?

You reply:

> Yes.

Then the Wisdom Web suggests:

> As far as I know, there are still some tickets left and you may purchase some at the Montreal Forum. It is easy to get there by taking Metro to the Atwater station.
>
> Now you decide that this could be an interesting evening for you. . .

One hour later, you arrive at the ticket office by Metro, but surprisingly find that the tickets left are all for the day after tomorrow when you will be traveling in Quebec City.

As you are a bit disappointed, you notice that there is a free Wisdom Web Kiosk right beside the ticket office. Well, that is convenient. So, without too much hesitation, you log on to the Wisdom Web, again as "Spiderman." The Wisdom Web still remembers your conversations an hour ago. As soon as it recognizes that you are "Spiderman," it says to you:

> Hello Spiderman, you were in such a hurry last time that I couldn't have a chance to tell you that all tickets available here are only for the day after tomorrow. They are quite expensive too. . .

Ten Capabilities of the Wisdom Web

To make the above Wisdom Web scenario a reality, the following 10 fundamental capabilities have to be incorporated and standardized (2):

1. *Self-organizing servers.* The Wisdom Web will regulate automatically the functions and cooperations of related websites and application services available. A Wisdom Web server self-nominates automatically to other services its functional *roles* as well as corresponding spatial or temporal *constraints* and operational *settings*.
2. *Specialization.* A Wisdom Web server is an *agent* by itself, which is specialized in performing some roles in a certain service. The *association* of its roles with any service will be measured and updated dynamically, for instance, the association may be forgotten if it is not used for some time.
3. *Growth.* The population of *Wisdom Agents* will change dynamically, as new agents are self-reproduced by their parent agents to become more specialized or as aged agents and are deactivated.
4. *Autocatalysis.* As various roles of wisdom agents are created through specialization and are activated by the *Wisdom Search* requests, their associations with some services and among themselves must be aggregated autocatalytically. In this respect, the autocatalysis of associations is similar to the pheromone laying for positive feedback in an ant colony.
5. *Problem Solver Markup Language (PSML).* PSML is necessary for wisdom agents to specify their roles and settings as well as relationships with any other services.
6. *Semantics.* The Wisdom Web needs to understand what are meant by "Montreal," "season," "year," and "night life," and what is the right judgment of "best," by understanding the granularities of their corresponding subjects and the whereabouts of their ontology definitions.
7. *Metaknowledge.* Besides semantic knowledge extracted and manipulated in the *Wisdom Search*, it is also essential for wisdom agents to incorporate a dynamically created source of metaknowledge that deals with the relationships between concepts and the spatial or temporal constraint knowledge in planning and executing services. It allows agents to self-resolve their conflict of interests.
8. *Planning.* In the above example, the goal is to find a function or an event that may sound attractive to a visitor. The constraint is that they must be happening during this season. Two associated subgoals are involved: To have an access to the recommended function or event, one needs a ticket. Furthermore, to go to get the ticket, one can travel by metro. In the Wisdom Web, ontology alone will *not* be sufficient.
9. *Personalization.* The Wisdom Web remembers the recent encounters and relates different episodes together, according to (1) "Spiderman," (2) time,

and (3) attainability of (sub)goals. In addition, it may identify other goals as well as courses of action for this user as their conversation continues.

10. *A sense of humor.* Although the Wisdom Web does not tell a funny story explicitly, it adds some punch lines to the situation or anxiety that "Spiderman" is presently in when he/she logs on for the second time, which will make "Spiderman" feel absurd.

Levels of WI

To develop a Wisdom Web to benefit from the information infrastructure that the Web has empowered, we have witnessed the fast development as well as applications of many WI techniques and technologies, which cover the following four conceptual levels at least:

1. *Internet-level communication, infrastructure, and security protocols*. The Web is regarded as a *computer-network system*. WI techniques for this level include Web data prefetching systems built upon Web surfing patterns to resolve the issue of Web latency. The intelligence of the Web prefetching comes from an adaptive learning process based on the observation and characterization of user surfing behavior (18,19).
2. *Interface-level multimedia presentation standards*. The Web is regarded as an *interface* for human–Internet interaction. WI techniques for this level are used to develop intelligent Web interfaces in which the capabilities of adaptive cross-language processing, personalized multimedia representation, and multimodal data processing are required.
3. *Knowledge-level information processing and management tools*. The Web is regarded as a *distributed data/knowledge base*. We need to develop semantic markup languages to represent the semantic contents of the Web available in machine-understandable formats for agent-based autonomic computing, such as searching, aggregation, classification, filtering, managing, mining, and discovery on the Web (20).
4. *Application-level ubiquitous computing and social intelligence environments*. The Web is regarded as a *basis for establishing social networks* that contain communities of people (or organizations or other social entities) connected by social relationships, such as friendship, coworking, or information exchange with common interests. They are Web-supported social networks or virtual communities. The study of WI concerns the important issues central to social network intelligence (social intelligence for short) (21). Furthermore, the multimedia contents on the Web are accessible not only from stationary platforms, but also increasingly from mobile platforms (22). Ubiquitous Web access and computing from various wireless devices needs adaptive personalization for which WI techniques are used to construct models of user interests by inferring implicitly from user behavior and actions (23,24).

In particular, the social intelligence approach presents excellent opportunities and challenges for the research and development of WI, as well as a Web-supported social network that needs to be supported by all levels of WI as mentioned above. This approach is based on the observation that the Web is now becoming an integral part of our society, and that scientists should be aware of it and take much care about handling social issues (25). Study in this area must receive as much attention as Web mining, Web agents, ontologies, and related topics.

Wisdom-Oriented Computing

Wisdom-oriented computing is a new computing paradigm aimed at providing not only a medium for seamless information exchange and knowledge sharing (20) but also a type of *man-made resources for sustainable knowledge creation, and scientific and social evolution*(2,3). The Wisdom Web, i.e., the Web that empowers wisdom-oriented computing, will reply on *grid-like service agencies* that self-organize, learn, and evolve their courses of actions to perform service tasks as well as their identities and interrelationships in Web communities. They will cooperate and compete among themselves to optimize their's as well as others, resources and utilities.

Self-organizing learning agents are computational entities that are capable of self-improving their performance in dynamically changing and unpredictable task environments. In Ref. (26), Liu has provided a comprehensive overview of several studies in the field of *autonomy oriented computing*, with in-depth discussions on self-organizing and with adaptive techniques for developing various embodiments of agent based systems, such as autonomous robots, collective vision and motion, autonomous animation, and search and segmentation agents. The core of those techniques is the notion of synthetic or emergent autonomy based on behavioral self-organization.

Developing the Wisdom Web will become a tangible goal for WI researchers and practitioners. The Wisdom Web will enable us to use the global connectivity optimally, as offered by the Web infrastructure, and most importantly, to gain the practical wisdoms of living, working, and playing, in addition to information search and knowledge queries.

To develop the new generation WI systems effectively, we need to define *benchmark* applications, i.e., a new *Turing Test*, that will capture and demonstrate the Wisdom Web capabilities (2).

Take the wisdom-oriented computing benchmark as an example. We can use a service task of compiling and generating a market report on an existing product or *a potential market report on a new product*. To get such service jobs done, an information agent on the Wisdom Web will mine and integrate available Web information, which will in turn be passed onto a market analysis agent. Market analysis will involve the quantitative simulations of customer behavior in a marketplace, instantaneously handled by other service agencies, involving a large number of semantic or computational grid agents (e.g. Ref. 27). Because the number of variables concerned may be in the order of hundreds or thousands, it can easily cost a single system years to generate one predication.

DEVELOPING INTELLIGENT PORTALS BY USING WI TECHNOLOGIES

What is a Portal?

A portal enables a company, an organization, or a community to create a *virtual organization* (or a virtual community) on the Web where key production/information steps are outsourced to partners and customers. In other words, a portal is a single gateway to personalized information needed to enable informed interdisciplinary research, services, and/or business activities. Developing intelligent portals is one of the most sophisticated applications on the Web.

Although specific features of various portals need to be considered, the common requirements of the portals for e-business, e-science, e-government, e-learning, among others, are such that

- they need a unique website (a single gateway) in which all of the contents related to the virtual organization can be accessed although such organization information is geographically distributed in multisite, multi data repositories, and multi-institution, and
- they need to have easy access to expensive remote facilities, computing resources, and share information acquired from different subjects using different techniques and stored in dedicated knowledge-data bases.

Many organizations are implementing a corporate portal first and are then growing this solution into more of an *intelligent* B2B portal. By using a portal to tie in back-end enterprise systems, a company can manage the complex interactions of the virtual enterprise partners through all phases of the value and supply chain.

Here we would like to mention two typical types of enterprises, as examples,

- transnational corporations that have operations, subsidiaries, investments, or branches worldwide, and
- communities with many mid-sized/small-scale companies in a region,

that need such enterprise portals for supporting their e-business and e-commerce activities.

The Virtual Industry Park: An Example of Enterprise Portals

As an example for developing enterprise portals by using WI technologies, here we discuss how to construct an intelligent virtual industry park (VIP) that has been developing in our group. The VIP portal is a website in which all of the contents related to the small/medium-sized companies in Maebashi city, Japan can be accessed.

The construction process can be divided into three phases. We first constructed a basic system including the fundamental functions such as the interface for dynamically registering/updating enterprise information, the database for storing the enterprise information, automatic generation and modification of enterprise homepages, and the domain-specific, keyword-based search engine. When designing the basic system, we also started by analyzing customer performance: each customer what has bought, over time, total volumes, trends, and so on.

Although the basic system can work as a whole one, we now need to know not only past performance on the business front, but also how the customer or prospect enters our VIP portal to target products and to manage promotions and marketing campaigns. To the already demanding requirement to capture transaction data for additional analysis, we now also need to use the Web usage mining techniques to capture the clicks of the mouse that define where the visitor has been on our website. What pages has he or she visited? What is the semantic association between the pages he or she visited? Is the visitor familiar with the Web structure? Or is he or she a new user or a random one? Is the visitor a Web robot or other users? In search for the holy grail of "stickiness," we know that a prime factor is *personalization* for:

- making a dynamic recommendation to a Web user based on the user profile and usage behavior.
- automatic modification of a website's contents and organization.
- combining Web usage data with marketing data to give information about how visitors used a website for marketers.

Hence, we need to extend the basic VIP system by adding more advanced functions such as Web mining, an ontologies-based search engine, as well as automatic e-mail filtering and management.

Finally, a portal for e-business-intelligence can be implemented by adding e-business-related application functions such as targeted marketing and CRM, electronic data interchange, as well as security solution.

An Intelligent Enterprise Portal Centric Schematic Diagram of WI Technologies

From the example stated in the above subsection, we can see that developing an intelligent enterprise portal needs to apply results from existing disciplines of AI and IT to a

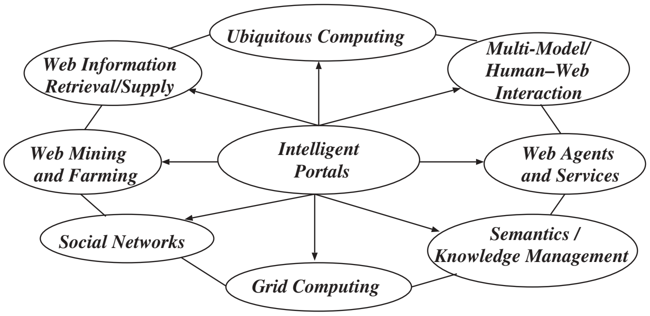


Figure 1. An intelligent enterprise portals centric schematic diagram of WI technologies.

totally new domain. On the other hand, the WI technologies are also expected to introduce new problems and challenges to the established disciplines on the new platform of the Web and the Internet. That is, WI is an enhancement or an extension of AI and IT.

To study advanced WI technologies systematically, and to develop advanced Web-based intelligent enterprise portals and information systems, we provide a schematic diagram of WI technologies from a Web-based, intelligent enterprise portals centric perspective in Fig. 1. In Fig. 1, directed lines denote that the development of intelligent enterprise portals needs to be supported by various WI related techniques, and undirected lines denote that the components of WI techniques are relevant each other.

Web Mining and Farming

The enterprise portal-based e-business activity that involves the end user is undergoing a significant revolution(10). The ability to track users' browsing behavior down to individual mouse clicks has brought the vendor and end customer closer than ever before. It is now possible for a vendor to personalize his product message for individual customers at a massive scale, which is called *targeted marketing* (or *direct marketing*) (11,13). Web mining and Web usage analysis play an important role in e-business for CRM and targeted marketing.

Web mining is the use of data mining techniques to discover and to extract information automatically from large Web data repositories such as Web documents and services (10,12,14,28). Web mining research is at the crossroads of research from several research communities, such as database, information retrieval, artificial intelligence, and especially the subareas of machine learning and natural language processing. Web mining can be divided into four classes of data available on the Web:

- *Web content*: the data that constitutes the Web pages and conveys information to the users, i.e., html, graphical, video, audio files of a Web page.
- *Web structure*: the data that formulates the hyper-link structure of a website and the Web, i.e., various HTML tags used to link one page to another and one website to another website.
- *Web usage*: the data that reflects the usages of Web resources, i.e., entries in Web browser's history and Internet temporary files, proxy server, and Web server logs.
- *Web user profile*: the data that provides demographic information about users of the website, i.e., users' registration data and customers' profile information.

Furthermore, Web content, structure, and usage information, in many cases, are copresent in the same data file. For instance, the file names appeared in the log files and Web structure data contain useful content information. One may safely assume that a file named "WebLogMining.html" must contain information about web log mining. Similarly, the categories of web mining cannot be considered exclusive or isolated from each other. Web content mining sometimes must use Web structure data to classify a web page. In the same way, Web usage mining sometimes has to make use of Web content data and of Web structure information.

A challenge is to explore the connection between Web mining and the related agent paradigm such as Web farming that is the systematic refining of information resources on the Web for business intelligence (16). Web farming extends Web mining into an evolving breed of information analysis in a whole process of Web-based information management including seeding, breeding, gathering, harvesting, refining, and so on.

ADVANCED TOPICS FOR STUDYING WI

With respect to different levels of WI as mentioned in the section entitled "Levels of WI," the Web can be studied in several ways.

Studying the Semantics in the Web

One of the fundamental WI issues is to study the *semantics* in the Web, called the semantic Web, that is, modeling semantics of Web information to

- allow more of the Web content (not just form) to become machine readable and processible.
- allow for recognition of the semantic context in which Web materials are used.
- allow for the reconciliation of terminological differences between diverse user communities.

Thus, information will be machine-processible in ways that support intelligent network services such as information brokers and search agents (20,29).

Main Components of the Semantic Web. The semantic Web is a step toward intelligence of the Web. It is based on languages that make more semantic content of the page available in machine-readable formats for agent-based computing. The main components of semantic Web techniques include:

- a unifying data model such as RDF (Resource Description Framework).
- languages with defined semantics, built on RDF, such as OWL.
- ontologies of standardized terminology to mark up Web resources, used by semantically rich, service-level descriptions (such as OWL-S, the OWL-based Web Service Ontology), and to support tools that assist the generation and processing of semantic markup.

Ontologies and agent technology can play a crucial role in Web intelligence by enabling Web-based knowledge processing, sharing, and reuse between applications. Generally defined as shared formal conceptualizations of particular domains, ontologies provide a common understanding of topics that can be communicated between people and agent-based systems.

An ontology is a formal, explicit specification of a shared conceptualization (30). It provides a vocabulary of terms and relations to model the domain and specifies how you view the target world. An ontology can be very high-level, consisting of concepts that organize the upper parts of a knowledge base, or it can be domain-specific such as a chemical ontology. We here suggest three categories of ontologies: *domain-specific*, *task*, and *universal*.

A domain-specific ontology describes a well-defined technical or business domain.

A task ontology might either be domain-specific, or might be a set of ontologies with respect to several domains (or their reconstruction for that task), in which relations between ontologies are described for meeting the requirement of that task.

A universal ontology describes knowledge at higher levels of generality. It is a more general-purpose ontology (or called a common ontology) that is generated from several domain-specific ontologies. It can serve as a bridge for communication among several domains or tasks.

Roles of Ontologies. Generally speaking, a domain-specific (or task) ontology forms the heart of any knowledge information system for that domain (or task). Ontologies provide a way of capturing a shared understanding of terms that can be used by human and programs to aid in information exchange. Ontologies have been gaining popularity as a method of providing a specification of a controlled vocabulary. Although simple knowledge representation such as Yahoo's taxonomy provides notions of generality and term relations, classic ontologies attempt to capture precise meanings of terms. To specify meanings, an ontology language must be used.

Ontologies will play a major role in supporting information exchange processes in various areas. The roles of ontologies for WI include:

- communication between Web communities.
- agent communication based on semantics.
- knowledge-based Web retrieval.
- understanding Web contents in a semantic way.
- social network and Web community discovery.

More specifically, new requirements for any exchange format on the Web are:

- *Universal expressive power*. A Web-based exchange format must be able to express any form of data.
- *Syntactic interoperability*. Applications must be able to read the data and get a representation that can be exploited.
- *Semantic interoperability*. One important requirement for an exchange format is that data must be understandable. It is about defining mappings between terms within the data, which requires content analysis.

The semantic Web requires interoperability standards that address not only the syntactic form of documents, but also the semantic content. Ontologies serve as metadata schemes for the semantic Web, providing a controlled vocabulary of concepts, each with explicitly defined and machine-processible semantics.

A semantic Web also lets agents use all (meta) data on all Web pages, allowing it to gain knowledge from one site and apply it to logical mappings on other sites for ontology-based Web retrieval and e-business intelligence. For instance, ontologies can be used in e-commerce to enable machine-based communication between buyers and sellers, vertical integration of markets, and description reuse between different marketplaces. Web-search agents use ontologies to find pages with words that are different syntactically but similar semantically.

Although ontology engineering has been studied over the last decade, few (semi) automatic methods for comprehensive ontology construction have been developed. Manual ontology construction remains a tedious, cumbersome task that can easily result in a bottleneck for WI. Learning and construction of domain-specific ontology from Web contents is an important task in both text mining and WI (31–34).

Studying the Web as Social Networks

The study of the Web as a network has resulted in a better understanding of the sociology of Web content creation; it has improved the search engines on the Web dramatically and has created more effective algorithms for community mining and for knowledge management.

We can view the Web as a directed network in which each node is a static web page to another. Thus, the Web can be studied as a *graph* that connects a set of people (or organizations or other social entities) connected by a set of social relationships, such as friendship, coworking or information exchange with common interests (21,35,36).

Social Network Analysis. In his keynote talk at WI '01, Raghavan stated that the main questions about the Web graph include:

- How big is the graph?
- Can we browse from any page to any other?
- Can we exploit the structure of the Web?
- What does the Web graph reveal about social dynamics?
- How to discover and manage the Web communities?

Modern social network theory is built on the work of Stanley Milgram (37). Milgram found so-called the *small-world phenomenon*, that is, typical paths took only six hops to arrive. Ravi Kumar et al. (35) observed there is a strong structural similarity between the Web as a network and social networks. The small-world phenomenon constitutes a basic property of the Web, which is not only interesting, but also useful.

In Ref. 21, it is suggested that the Web graph has several billion nodes (pages of content) and an average degree of about 7. A recurrent observation on the Web graph is the prevalence of power laws: The degree of nodes are distributed according to inverse polynomial distribution (18,19,38–40).

The Web captures automatically a rich interplay between hundreds of millions of people and billions of pages of content. In essence, these interactions embody a *social network* involving people, the pages they create and view, and even the Web pages themselves. These relationships have a bearing on the way in which we create, share, and manage knowledge and information. It is our hope that exploiting these similarities will lead to progress in knowledge management and business intelligence.

The *broader* social network is a self-organizing structure of users, information, and communities of expertise (21,23). Such social networks can play a crucial role in implementing next-generation enterprise portals with functions such as data mining and knowledge management for discovery, analysis, and management of social network knowledge.

The social network is placed at the top of a four-level WI infrastructure as described in the section on "levels of WI" and is supported by functions, provided in all Levels of WI, including security, prefetching, adaptive cross-language processing, personalized multimedia representation, semantic searching, aggregation, classification, filtering, managing, mining, and discovery.

Semantic Social Networks for Intelligent Portals. One of the most sophisticated applications on the Web today is *enterprise information portals* operating with state-of-the-art markup languages to search, retrieve, and repackage data. The enterprise portals are being developed into an even more powerful center based on component-based applications called Web Services (21,23).

WI researchers must study both centralized and distributed information structures. Information on the Web can be either globally distributed throughout the Web within multilayer over the infrastructure of Web protocols, or located locally, centralized on an intelligent portal providing Web services (i.e., the intelligent service provider) that is integrated to its own cluster of specialized intelligent applications. However, each approach has a serious flaw. As pointed out by Alesso and Smith (23), the intelligent portal approach limits uniformity and access, whereas the global semantic Web approach faces combinatory complexity limitations.

A way to solve the above issue is to develop and use the Problem Solver Markup Language (PSML), for collecting globally distributed contents and knowledge from Web-supported, semantic social networks and incorporating them with locally operational knowledge/databases in an enterprise or community for local centralized, adaptable Web intelligent services.

The core of PSML is distributed inference engines that can perform automatic reasoning on the Web by incorporating contents and meta-knowledge autonomically collected and transformed from the semantic Web with locally operational knowledge-data bases. A feasible way as the first step to implement such a PSML is to use existing Prolog-like logic language with agent technologies. In our current experiments, KAUS is used for representation of local information sources and for inference and reasoning. KAUS is a knowledge management system developed in our group that involves data/knowledge bases on the basis of an extended first-order predicate logic and relational data model (41,42). KAUS enables representation of knowledge and data in the first-order logic with data structure in multi-level and can be easily used for inference and reasoning as well as transforming and managing both knowledge and data.

By using this information transformation approach, the dynamic, global information sources on the Web can be combined with the local information sources in an enterprise portal for decision making and e-business intelligence.

Soft Computing for WI

Another challenging problem in WI is how to deal with uncertainty of information on the wired and wireless Web. Adapting existing soft computing solutions, when appropriate for WI applications, must incorporate a robust notion of learning that will scale to the Web, adapt to individual user requirements, and personalize interfaces. Ongoing efforts exist to integrate logic (including nonclassical logic), artificial neural networks, probabilistic and statistical reasoning, fuzzy sets, rough sets, granular computing, genetic algorithm, and other methodologies in the soft computing paradigm, to construct a hybrid approach/system for Web intelligence.

WI-BASED TARGETED MARKETING

An enterprise portal for business intelligence needs the function of WI-based targeted marketing, which is integrated with WI related capabilities such as Web mining, the ontologies-based search engine, personalized recommendation, as well as automatic e-mail filtering and management (8).

Targeted marketing aims at obtaining and maintaining direct relationships between suppliers and buyers within one or more product/market combinations. Targeted marketing becomes more and more popular because of the increased competition and the cost problem.

Furthermore, the scope of targeted marketing can be expanded from considering only how products are distributed, to include enhancing the relationships between an organization and its customers (43) because the strategic importance of long-term relationships with customers. In other words, once customers are acquired, customer retention becomes the target. Retention through customer satisfaction and loyalty can be improved greatly by acquiring and exploiting knowledge about these customers and their needs. Such targeted marketing is called "targeted relationship marketing" or "CRM" (44).

The Market Value Function (MVF) Model

In addition to WI related capabilities, targeted marketing is an important area of applications for data mining and for data warehousing (4,45). Although standard data mining methods may be applied for the purpose of targeted marketing, many specific algorithms need to be developed and applied for direct marketer to make decisions effectively.

Let us consider now a typical problem of targeted marketing. Suppose a health club needs to expand its operation by attracting more members. Assume that each existing member is described by a finite set of attributes. It is natural to examine existing members to identify their common features. Information about the health club may be sent to nonmembers who share the same features of members or similar to members. Other examples include promotion of special types of phone services and marketing of different classes of credit cards. In this case, we explore the relationships (similarities) between people (objects) based on their attribute values. The underlying assumption is that *similar type of people tend to make similar decisions and to choose similar services*. Techniques for mining association rules may not be applicable directly to this type of targeted marketing. One may produce too many or too few rules. The selection of a good set of rules may not be an easy task. Furthermore, the use of the derived rules may produce too many or too few potential new members.

To address this issue, we proposed a new model for targeted marketing by focusing on the issues of knowledge representation and computation of market values (4,12). More specifically, we assume that each object is represented by its values on a finite set of attributes. Also, we assume that market values of objects can be computed using a linear market value function. Thus, we may consider the proposed model to be a *linear* model, which is related to, but is different from, the linear model for information retrieval.

Let U be a finite universe of objects. Elements of U may be customers or products we are interested in market oriented decision making. The universe U is divided into three pair-wise disjoint classes, i.e., $U = P \cup N \cup D$. The sets P, N, and D are called *positive*, *negative*, and *don't know* instances, respectively. Take the earlier health club example, P is the set of current members, N is the set of people who had refused to join the club previously, and D is the set of the rest. The set N may be empty. A targeted marketing problem may be defined as finding elements from D, and possibly from N, that are similar to elements in P, and possibly dissimilar to elements in N. In other words, we want to identify elements from D and N that are more likely to become new members of P. We are interested in finding a market value function so that elements of D can be ranked accordingly.

Information about objects in a finite universe is given by an information table (46,47). The rows of the table correspond to objects of the universe, the columns correspond to attributes, and each cell is the value of an object with respect to an attribute. Formally, an information table is a quadruple:

$$S = (U, At, \{V_a | a \in At\}, \{I_a | a \in At\})$$

where U is a finite nonemptyset of objects, At is a finite nonempty set of attributes, V_a is a nonempty set of values for $a \in At$, $I_a : U \rightarrow V_a$ is an information function for $a \in At$. Each information function I_a is a total function that maps an object of U to exactly one value in V_a. An information table represents all available information and knowledge. Objects are only perceived, observed, or measured by using a finite number of properties (46).

A market value function (MVF) is a real-valued function from the universe to the set of real numbers, $r : U \rightarrow \Re$. In the context of information retrieval, the values of r represent the potential usefulness or relevance of documents with respect to a query. According to the values of r, documents are ranked. For the targeted marketing problem, a market value function ranks objects according to their potential market values. For the health club example, a market value function ranks people according to their likelihood of becoming a member of the health club. The likelihood may be estimated based on its similarity to a typical member of P.

We studied the simplest form of market value functions, i.e., the linear discriminant functions. Let $u_a : V_a \rightarrow \Re$ be a utility function defined on V_a for an attribute $a \in At$. The utility $u_a(\cdot)$ may be positive, negative, or zero. For $v \in V_a$, if $u_a(v) > 0$ and $I_a(x) = v$, i.e., $u_a(I_a(x)) > 0$, then attribute a has a positive contribution to the overall market value of x. If $u_a(I_a(x)) < 0$, then a has a negative contribution. If $u_a(I_a(x)) = 0$, then a has no contribution. The pool of contributions from all attributes is computed by a linear market value function of the following form:

$$r(x) = \sum_{a \in At} w_a u_a(I_a(x)) \tag{1}$$

where w_a is the weight of attribute a. Similarly, the weight w_a may be positive, negative, or zero. Attributes with larger weights (absolute value) are more important, and attributes with weights close to zero are not important. The overall market value of x is a weighted combination of utilities of all attributes. By using a linear market value function, we have implicitly assumed that contributions made by individual attributes are independent. Such an assumption is known as utility independence assumption commonly. Implications of utility independence assumption can be found in literature of multi-criteria decision making (48).

The market value model proposes a linear model to solve the target selection problem of targeted marketing by drawing and extending result from information retrieval (4,12). It is assumed that each object is represented by values of a finite set of attributes. A market value function is a linear combination of utility functions on attribute values, which depends on two parts: *utility function* and *attribute weighting*.

The market value function has some advantages. First, it can rank individuals according to their market value instead of classifying; second, the market value functions is interpretable; and last, the system of the market value function can perform without expertise.

Multi-Aspect Analysis in Multiple Data Sources

Generally speaking, customer data can be obtained from multiple customer touchpoints. In response, multiple data sources that are obtained from multiple customer touchpoints, including the Web, wireless, call centers, and brick-and-mortar store data, need to be integrated into a distributed data warehouse that provides a multi faceted view of their customers, their preferences, interests, and expectations for multi aspect analysis. Hence, a multi strategy and multi agent data mining framework is required (6,49).

One of main reasons for developing a multi agent data mining system is that we cannot expect to develop a single data mining algorithm that can be used to solve all targeted marketing problems because of the complexity of real-world applications. Hence, various data mining agents need to be used cooperatively in the multi step data mining process for performing multi aspect analysis as well as multi level conceptual abstraction and learning.

The other reason for developing a multi agent data mining system is that when performing multi aspect analysis for complex targeted marketing problems, a data mining task needs to be decomposed into subtasks. Thus, these sub tasks can be solved by using one or more data mining agents that are distributed over different computers and multi data repositories on the Internet. The decomposition problem leads us to the problem of distributed cooperative system design.

In the VIP stated in the section on the virtaul Industry Park for instance, mainly three kinds of data sources are considered, namely, customer database, products database, and Web farming database. Furthermore, in addition to the MVF based data mining method (12) mentioned in the section on the MVF model, we have developed various data mining methods, such as the GDT-RS inductive learning system for discovering classification rules (50), the LOI (learning with ordered information) for discovering important features (51,52), as well as the POM (peculiarity oriented mining) for finding peculiarity data/rules (53), to deal with each of such data sources, separately, for various services oriented multi aspect data analysis.

However, when we try to integrate the three kinds of data sources together into the advanced VIP system, we must know how to interact with each of those sources to extract the useful pieces of information, which then have to be combined for building the expected answer to the initial request. Hence, the core question is how to manage, represent, integrate, and use the information coming from huge, distributed, multiple-data sources.

Here, we would like to emphasize that how to manage, analyze, and use the information intelligently from different data sources is a problem that not exists only in the e-business field, but also in e-science, e-learning, e-government, as well as all WI systems and services (54,55). The development of enterprise portals and e-business intelligence is a good example for trying to solve such problem.

Building a Data Mining Grid

To implement an enterprise portal (e.g., the VIP discussed previously) for Web-based targeted marketing and business intelligence, a new infrastructure and platform as the *middleware* is required to deal with large, distributed data sources for multi aspect analysis. One methodology is to create a grid-based, organized society of data mining agents, called a *Data Mining Grid* on the grid computing platform (e.g., the Globus toolkit) (27,55–59). A data mining and must do the following:

- Develop various data mining agents, as mentioned in the section on the MVP model, for various services, oriented multiaspect data analysis;
- Organize the data mining agents into a multi layer grid, such as a data-grid, mining-grid, or knowledge-grid, under the Open Grid Services Architecture that aligns firmly with service-oriented architecture and Web services and understands the user's questions, transforms them to data mining issues, discovers the resources and information about the issues, and obtains a composite answer or solution.
- Use a conceptual model with three level workflows, namely data flow, mining flow, and knowledge flow, with respect to the data grid, the mining grid, and the knowledge grid, respectively, for managing the grid of data mining agents for multi aspect analysis in distributed, multiple-data sources and for organizing the dynamic, status-based business processes.

That is, the data mining grid is made of many smaller components that are called *data mining agents*. Each agent by itself can only do one simple thing. Yet when we join these agents in a *grid*, this implements more complex targeted marketing and business intelligence tasks.

Furthermore, ontologies are also used for description and for integration of multi data source and grid-based data mining agents in data mining process planning (6,7,28), which will provide the following:

- a formal, explicit specification for integrated use of multiple data sources in a semantic way.
- a conceptual representation about the sorts and properties of data/knowledge and data mining agents, as well as relations between data/knowledge and data mining agents.
- a vocabulary of terms and relations to model the domain, and specifying how to view the data sources and how to use data mining agents.
- a common understanding of multiple data sources that can be communicated between grid-based data mining agents.

CONCLUDING REMARKS

WI has been recognized as one of the most important as well as the fastest-growing IT research fields in the era of the World Wide Web, knowledge Web, grid computing, intelligent agent technology, and ubiquitous social computing. WI technologies will continue to produce the new tools and the infrastructure components necessary for creating intelligent enterprise portals that can serve users *wisely*.

To meet the strong demands for participation and the growing interests in WI, the Web Intelligence Consortium (WIC) was formed in spring 2002. The WIC (http://wi-consortium.org/) is an international non-profit organization dedicated to advancing world-wide scientific research and industrial development in the field of WI. It promotes collaborations among world wide WI research centers and organizational members, technology showcases at WI related conferences and workshops, WIC official book and journal publications, WIC newsletters, and WIC official releases of new industrial solutions and standards.

In addition to major WI related conferences/workshops, such as IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, and numerous special issues in international journals/magazines, such as IEEE Computer, a WI-focused scientific journal, *Web Intelligence and Agent Systems: An International Journal* (refer to the WIC homepage), has been providing a standard international forum for disseminating results of advanced research and development in the field of WI.

The interest in WI is growing very fast. We would like to invite everyone, who are interested in the WI related research and development activities, to join the WI community. Your input and participation will determine the future of WI.

ACKNOWLEDGMENTS

We are very grateful to people who have joined or supported the WI community, members of the WIC advisory board, WIC technical committee, and WIC research centers, as well as keynote/invited speakers of WI-IAT conferences, in particular, J. Bradshaw, W. Buntine, N. Cercone, P. Doherty, B. B. Faltings, D. Fensel, E. A. Feigenbaum, I. Foster, G. Gottlob, F. Harmelen, J. Hendler, N. Jennings, W. L. Johnson, C. Kesselman, P. Langley, H. Lieberman, V. Lesser, J. McCarthy, R. Mizhouchi, T. M. Mitchell, T. Nishida, S. Ohsuga, P. Raghavan, Z. W. Ras, P. Schuster, A. Skowron, K. Sycara, B. Wah, M. Wooldridge, X. Wu, P. S. Yu, and L. A. Zadeh. We thank them for their strong support.

REFERENCES

1. J. Liu, N. Zhong, Y. Y. Yao, Z. W. Ras, The wisdom web: new challenges for web intelligence (WI), *J. Intell. Inform. Sys.*, **20**(1): 5–9, 2003.
2. J. Liu, Web intelligence (WI): what makes wisdom web? *Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI-03),* 2003, pp. 1596–1601.
3. J. Liu, New challenges in the world wide wisdom web (W4) research, in N. Zhong, et al. (eds.), *Foundations of Intelligent Systems*, LNAI 2871, Springer, 2003, pp. 1–6.
4. Y. Y. Yao, N. Zhong, J. Liu, and S. Ohsuga, Web intelligence (WI): research challenges and trends in the new information age, in N. Zhong, et al. (eds.), *Web Intelligence: Research and Development*, LNAI 2198, Springer, 2001, pp. 1–17.
5. N. Zhong, J. Liu, Y. Y. Yao, and S. Ohsuga, Web intelligence (WI), *Proc. 24th IEEE Computer Society International Computer Software and Applications Conference (COMPSAC 2000),* Piscataway, NJ: IEEE *Computer Society* Press, 2000, pp. 469–470.
6. N. Zhong, Y. Y. Yao, J. Liu, and S. Ohsuga (eds.), *Web Intelligence: Research and Development*, LNAI 2198, New York: Springer, 2001.
7. N. Zhong, J. Liu, and Y. Y. Yao, In search of the wisdom web,. *IEEE Computer*, **35**(11): 27–31, 2002.
8. N. Zhong, J. Liu, and Y.Y. Yao, (eds.), *Web Intelligence*, New York: Springer, 2003.
9. N. Zhong, J. Liu, and Y. Y. Yao, Envisioning intelligent information technologies from the stand-point of Web intelligence, *Commun. ACM*, **50**(3): 89–94, 2007.
10. J. Srivastava, R. Cooley, M. Deshpande, P. Tan, Web usage mining: discovery and applications of usage patterns from web data, *SIGKDD Explorations, Newsletter of SIGKDD*, **1**: 12–23, 2000.
11. A. R. Simon, S. L. Shaffer, *Data Warehousing and Business Intelligence for e-Commerce*, San Francisco, CA: Morgan Kaufmann, 2001.
12. Y. Y. Yao, N. Zhong, J. Huang, C. Ou, and C. Liu, Using market value functions for targeted marketing data mining, *Interna. J. Pattern Recogn. Artif. Intell.*, **16**(8): 1117–1131, 2002.
13. N. Zhong, J. Liu, and Y. Y. Yao, Web intelligence (WI): a new paradigm for developing the wisdom web and social network intelligence, in N. Zhong, et al. (eds.), *Web Intelligence*, New York: Springer, 2003, pp. 1–16.
14. R. Kosala and H. Blockeel, Web mining research: a survey, *ACM SIGKDD Explor. News.*, **2**: 1–15, 2000.
15. Z. Lu, Y. Y. Yao, N. Zhong, Web log mining, in N. Zhong, et al. (eds.), *Web Intelligence*, New York: Springer, 2003, pp. 172–194.
16. R. D. Hackathorn, *Web Farming for the Data Warehouse*, San Francisco, CA: Morgan Kaufmann, 2000.
17. N. Porter (ed.), *Webster's Revised Unabridged Dictionary*, G&C. Merriam Co, 1913.
18. J. Liu, S. Zhang, Y. Ye, Agent-based characterization of web regularities, in N. Zhong, et al. (eds.), *Web Intelligence*, New York: Springer, 2003, pp. 19–36.
19. J. Liu, S. Zhang, J. Yang, Characterizing web usage regularities with information foraging agents, *IEEE Trans. Know. Data Engin.* **16**(4): 2004.
20. T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, *Scientific Am.* **284**: 34–43, 2001.
21. P. Raghavan, Social networks: from the web to the enterprise, *IEEE Internet Computing*, **6**(1): 91–94, 2002.
22. M. Weiser, The future of ubiquitous computing on campus, *CACM*, **41**(1): 41–42, 1998.
23. H. P. Alesso, C. F. Smith, *The Intelligent Wireless Web*, Reading, MA: Addison-Wesley, 2002.
24. D. Billsus, et al., Adaptive interfaces for ubiquitous web access, *Commun. ACM*, **45**: 34–38, 2002.
25. T. Nishida, Social intelligence design for the web, *IEEE Computer*, **35**(11): 37–41, 2002.
26. J. Liu, *Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization and Adaptive Computation*, Singapore: World Scientific, 2001.
27. F. Berman, From teragrid to knowledge grid, *Commun. ACM*, **44**: 27–28, 2001.
28. N. Zhong, Knowledge discovery and data mining, *The Encyclopedia of Microcomputers*, **27**(suppl. 6): 235–285, 2001.
29. S. Decker, P. Mitra, and S. Melnik, Framework for the semantic web: an RDF tutorial, *IEEE Internet Comp.*, **4**(6): 68–73, 2000.
30. D. Fensel, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, New York: Springer, 2001.
31. Y. Li and N. Zhong, Mining Ontology for Automatically Acquiring Web User Information Needs, *IEEE Trans. Know. Data Engineer.*, **18**(4): 554–568, 2006.
32. A. Maedche and S. Staab, Ontology learning for the semantic web, *IEEE Intell. Sys.*, **16**(2): 72–79, 2001.

33. M. Missikoff, R. Navigli, P. Velardi, Integrated approach to web ontology learning and engineering, *IEEE Computer*, **35**(11): 60–63, 2002.

34. N. Zhong, Representation and construction of ontologies for web intelligence, *Internat. J. Foundations Comp. Sci.*, **13**(4): 555–570, 2002.

35. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, The web and social networks, *IEEE Computer*, **35** (11): 32–36, 2002.

36. W. Li, N. Zhong, J. Liu, Y. Y. Yao, C. Liu, Perspective of Applying the Global E-mail Network, *Proc. 2006 IEEE/ WIC/ACM International Conference on Web Intelligence (WI'06)*, IEEE Computer Society Press, pp. 117–120, 2006.

37. S. Wasserman and K. Faust, *Social Network Analysis*, Cambridge, MA: Cambridge University Press, 1994.

38. R. Albert, H. Jeong, A. L. Barabasi, Diameter of the world-wide web, *Nature*, **410**: 130–131, 1999.

39. B. A. Huberman, P. L. T. Pirolli, J. E. Pitkow, R. M. Lukose, Strong regularities in world wide web surfing, *Science*, **280**: 96–97, 1997.

40. B. A. Huberman, L. A. Adamic, Growth dynamics of the world-wide web, *Nature,* **410**: 131, 1999.

41. S. Ohsuga, Framework of knowledge based systems - multiple meta-level architecture for representing problems and problem solving processes, *Knowledge Based Sys.*, **3**(4): 204–214, 1990.

42. H. Yamauchi and S. Ohsuga, Loose coupling of KAUS with existing RDBMSs, *Data & Knowledge Engineering*, **5**(4): 227–251, 1990.

43. W. Klosgen and J. M. Zytkow, *Handbook of Data Mining and Knowledge Discovery*, Oxford: Oxford University Press, 2002.

44. R. Stone, *Successful Direct Marketing Methods*, 6th ed., Lincolnwood, IL: NTC Business Books , 1996.

45. P. Van Der Putten, Data mining in direct marketing databases, in W. Baets (ed)., *Complexity and Management: A Collection of Essays*, Singapore: World Scientific, 1999.

46. Z. Pawlak, *Rough Sets, Theoretical Aspects of Reasoning about Data*, Dordrecht: Kluwer, 1991.

47. Y.Y. Yao and N. Zhong, Granular computing using Iinformation tables, in T. Y. Lin, Y. Y. Yao, and L. A. Zadeh (eds.), *Data Mining, Rough Sets and Granular Computing*, Berlin: Physica-Verlag, 2002, pp. 102–124.

48. P. C. Fishburn, Seven independence concepts and continuous multiattribute utility functions, *J. Math. Psycho.*, **11**: 294–327, 1974.

49. N. Zhong, C. Liu, and S. Ohsuga, Dynamically organizing KDD process, *Internat. J. Pattern Recog. Artif. Intell.*, **15**(3): 451–473, 2001.

50. N. Zhong, J. Z. Dong, C. Liu, and S. Ohsuga, A hybrid model for rule discovery in data, *Knowledge Based Sys.* **14**(7): 397–412, 2001.

51. Y. Sai, Y. Y. Yao, and N. Zhong, Data analysis and mining in ordered information tables, *Proc. 2001 IEEE International Conference on Data Mining (ICDM'01)*, Piscataway, NJ: IEEE Computer Society Press, 2001, pp. 497–504.

52. N. Zhong, Y. Y. Yao, J. Z. Dong, and S. Ohsuga, Gastric cancer data mining with ordered information, in J. J. Alpigini, et al. (eds.), *Rough Sets and Current Trends in Computing*, LNAI 2475, New York: Springer, 2002, pp. 467–478.

53. N. Zhong, Y. Y. Yao, and M. Ohshima, Peculiarity oriented multi-database mining, *IEEE Trans. Knowl. Data Engineer.*, **15**(4): 952–960, 2003.

54. J. Hu and N. Zhong, Organizing multiple data sources for developing intelligent e-Business Portals, *Data Mining Know. Dis.*, **12** (2–3): 127–150, 2006.

55. M. Cannataro, and D. Talia, The knowledge grid, *CACM*, **46**: 89–93, 2003.

56. N. Zhong, J. Hu, S. Motomura, J. L. Wu, and C. Liu, Building a data mining grid for multiple human brain data analysis, *Computat. Intell.*, **21**(2): 177–196, 2005.

57. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco, CA: Morgan Kaufmann, 1999.

58. I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, San Francisco, CA: Morgan Kaufmann, 2004.

59. J. Nabrzyski, J. M. Schopf, J. Weglarz, *Grid Resource Management*, Dordrecht: Kluwer, 2004.

FURTHER READING

A. Congiusta, A. Pugliese, D. Talia, and P. Trunfio, Designing Grid Services for distributed knowledge discovery, *Web Intell. Agent Sys,* **1**(2): 91–104, 2003.

J. A. Hendler and E. A. Feigenbaum, Knowledge is power: the semantic web vision, in N. Zhong, et al. (eds.), *Web Intelligence: Research and Development*, LNAI 2198, Springer, 2001, 18–29.

N. Zhong and J. Liu (eds.), *Intelligent Technologies for Information Analysis*, New York: Springer, 2004.

NING ZHONG
Maebashi Institute
of Technology
Maebashi City, Japan

JIMING LIU
Hong Kong Baptist
University
Hong Kong, China

YIYU YAO
University of Regina
Regina, Saskatchewan, Canada

R

ROUGH SET THEORY

INTRODUCTION

Rough set theory is a new mathematical approach to imperfect knowledge. The problem of imperfect knowledge has been tackled for a long time by philosophers, logicians, and mathematicians. Recently, it has also became a crucial issue for computer scientists, particularly in the area of artificial intelligence (AI). There are many approaches to the problem of how to understand and manipulate imperfect knowledge. The most successful one is, no doubt, the fuzzy set theory proposed by Zadeh (1).

Rough set theory (2) presents still another attempt to this problem. This theory has attracted attention of many researchers and practitioners all over the world, who have contributed essentially to its development and applications. Rough set theory overlaps with many other theories, despite which rough set theory may be considered as an independent discipline in its own right. The rough set approach seems to be of fundamental importance to AI and cognitive sciences, especially in the areas of machine learning, knowledge acquisition, decision analysis, knowledge discovery from databases, expert systems, inductive reasoning, and pattern recognition. The main advantage of rough set theory in data analysis is that it does not need any preliminary or additional information about data like probability distributions in statistics, basic probability assignments in Dempster–Shafer theory, a grade of membership, or the value of possibility in fuzzy set theory. One can observe the following about the rough sets approach:

– introduction of efficient algorithms for finding hidden patterns in data,
– determination of minimal sets of data (data reduction),
– evaluation of the significance of data,
– generation of sets of decision rules from data,
– easy-to-understand formulation,
– straightforward interpretation of obtained results, and
– suitability of many of its algorithms for parallel processing.

One of the issues discussed in connection with the notion of a set is vagueness. Mathematics requires that all mathematical notions (including set) must be exact (Gottlob Frege (3)). However, philosophers and, recently, computer scientists have become interested in vague (imprecise) concepts. For example, in contrast to odd numbers, the notion of a beautiful painting is vague, because we are unable to classify uniquely all paintings into two classes: beautiful and not beautiful. Sometimes it is not possible to decide whether some paintings are beautiful or not and thus they remain in the doubtful area. Thus, beauty is not a precise but a vague concept. Almost all concepts we are using in natural language are vague. Therefore, common-sense reasoning based on natural language must be based on vague concepts and not on classic logic, which is why vagueness is important for philosophers and recently also for computer scientists. Vagueness is usually associated with the boundary region approach (i.e., existence of objects that cannot be uniquely classified to the set or its complement), which was first formulated in 1893 by the father of modern logic Gottlob Frege (3), who wrote:

> *"Der Begriff muss scharf begrenzt sein. Einem unscharf begrenzten Begriff würde ein Bezirk ensprechen, der nicht überall ein scharfe Grentzlinie hätte, sondern stellenweise gantz verschwimmend in die Umgebung übergine. Das wäre eigentlich gar kein Bezirk; und so wird ein unscharf definirter Begriff mit Unrecht Begriff gennant. Solche begriffsartige Bildungen kann die Logik nicht als Begriffe anerkennen; es is unmäglich, von ihnen genaue Gesetze auszustellen. Das Gesetz des ausgeschlossenen Drititten ist ja eigentlich nur in anderer Form die Forderung, dass der Begriff scharf begrentz sei. Ein beliebiger Gegenstand x fällt entwerder unter der Begriff y, oder er fällt nich unter ihn: tertium non datur."*

Thus, according to Frege, the concept must have a sharp boundary. To the concept without a sharp boundary, there would correspond an area that would not have any sharp boundary-line all around. It means that mathematics must use crisp, not vague concepts, otherwise it would be impossible to reason precisely.

Lotfi Zadeh (1) introduced a very successful approach to vagueness. In this approach, sets are defined by partial membership, in contrast to crisp membership used in the classic definition of a set. Rough set theory (2) expresses vagueness, not by means of membership, but by employing the boundary region of the set. If the boundary region of the set is empty, it means that the set is crisp, otherwise the set is rough (inexact). The nonempty boundary region of the set means that our knowledge about the set is not sufficient to define the set precisely. Discussion on vagueness in the context of fuzzy sets and rough sets can be found in Ref. 4. Basic ideas of rough set theory and its extensions, as well as many interesting applications can be found in books (see Refs. 2, 5–17), special issues of journals (see Refs. 18–25), proceedings of international conferences (see Refs. 26–36), and on the Internet (see, e.g.,www. roughsets.org, logic.mimuw.edu.pl, rsds.wsiz.rzeszow.pl).

Recent years are witness to a rapid grow of interest in rough set theory and its applications worldwide. Many international workshops, conferences, and seminars have included rough sets in their programs. A large number of high-quality papers on various aspects of rough sets and their applications have been published in recent years. In this article, we present the basic concepts of rough set theory and outline some research directions on rough sets.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

BASIC PHILOSOPHY

The rough set philosophy is founded on the assumption that with every object of the universe of discourse we associate some information (data, knowledge). For example, if objects are patients suffering from a certain disease, symptoms of the disease form information about patients. Objects characterized by the same information are indiscernible (similar) in view of the available information about them. The indiscernibility relation generated in this way is the mathematical basis of rough set theory. This understanding of indiscernibility is based on the idea of Gottfried Wilhelm Leibniz that objects are indiscernible if and only if all available functionals take on identical values (Leibnizian indiscernibility). Any set of all indiscernible (similar) objects is called an elementary set, and forms a basic granule (atom) of knowledge about the universe. Any union of some elementary sets is referred to as crisp (precise) set, otherwise the set is rough (imprecise, vague).

Consequently, each rough set has boundary–line cases (i.e. objects that cannot with certainty be classified either as members of the set or of its complement). Obviously, crisp sets have no boundary-line elements at all, which means that boundary-line cases cannot be properly classified by employing the available knowledge.

Thus, the assumption that objects can be "seen" only through the information available about them leads to the view that knowledge has granular structure. Due to the granularity of knowledge, some objects of interest cannot be discerned and appear the same (or similar). As a consequence, vague concepts, in contrast to precise concepts, cannot be characterized in terms of information about their elements. Therefore, in the proposed approach, we assume that any vague concept is replaced by a pair of precise concepts—called the lower and the upper approximation of the vague concept. The lower approximation consists of all objects that surely belong to the concept and the upper approximation contains all objects that possibly belong to the concept. The difference between the upper and the lower approximation constitutes the boundary region of the vague concept. Approximations are two basic operations in rough set theory.

APPROXIMATIONS AND ROUGH SETS

As mentioned, the starting point of rough set theory is the indiscernibility relation, generated by information about objects of interest. The indiscernibility relation expresses the fact that, because of the lack of knowledge, we are unable to discern some objects employing available information, which means that, in general, we are unable to deal with each particular object but we have to consider granules (clusters) of indiscernible objects, as fundamental concepts of our theory.

Now we present the basic concepts more formally.

Suppose we are given two finite, non-empty sets U and A, where U is the *universe of objects* and A is a set of *attributes*. The pair (U, A) is called an *information table*. With every attribute $a \in A$, we associate a set V_a, of its *values*, called the *domain* of a. Any subset B of A determines a binary relation $I(B)$ on U, called an *indiscernibility relation*, defined by

$$xI(B)y \text{ if and only if } a(x) = a(y) \text{ for every } a \in B \qquad (1)$$

where $a(x)$ denotes the value of attribute a for object x.

Obviously, $I(B)$ is an equivalence relation. The family of all equivalence classes of $I(B)$ (i.e., the partition determined by B,) will be denoted by $U/I(B)$, or simply U/B; an equivalence class of $I(B)$ (i.e., the block of the partition U/B) containing x will be denoted by $B(x)$.

If $(x, y) \in I(B)$, we will say that x and y are *B-indiscernible*. Equivalence classes of the relation $I(B)$ (or blocks of the partition U/B) are referred to as *B-elementary sets*. In the rough set approach, the elementary sets are the basic building blocks (concepts) of our knowledge about reality. The unions of *B-elementary sets* are called *B-definable sets*.

The indiscernibility relation will be further used to define basic concepts of rough set theory. Let us define now the following two operations on sets:

$$B_*(X) = \{x \in U : B(x) \subseteq X\} \qquad (2)$$

$$B^*(X) = \{x \in U : B(x) \cap X \neq \emptyset\} \qquad (3)$$

assigning to every subset X of the universe U two sets $B_*(X)$ and $B^*(X)$ called the *B-lower* and the *B-upper approximation* of X, respectively. The set

$$BN_B(X) = B^*(X) - B_*(X) \qquad (4)$$

will be referred to as the *B-boundary region* of X.

If the boundary region of X is the empty set (i.e., $BN_B(X) = \emptyset$), then the set X is *crisp* (*exact*) with respect to B; in the opposite case (i.e., if $BN_B(X) \neq \emptyset$), the set X is referred to as *rough* (*inexact*) with respect to B.

A rough set can also be characterized numerically by the following coefficient:

$$\alpha_B(X) = \frac{|B_*(X)|}{|B^*(X)|} \qquad (5)$$

called the *accuracy of approximation*, where $|X|$ denotes the cardinality of $X \neq \emptyset$. Obviously, $0 \leq \alpha_B(X) \leq 1$. If $\alpha_B(X) = 1$, then X is *crisp* with respect to B (X is *precise* with respect to B), and otherwise, if $\alpha_B(X) < 1$, then X is *rough* with respect to B (X is *vague* with respect to B).

Several generalizations of the classic rough set approach based on approximation spaces defined as pairs of the form (U, R), where R is the equivalence relation (called indiscernibility relation) on the set U, have been reported in the literature. Let us mention two of them.

A generalized approximation space can be defined by a tuple $AS = (U, I, \nu)$, where I is the *uncertainty function* defined on U with values in the powerset $P(U)$ of $U(I(x)$ is the *neighboorhood* of x) and ν is the *inclusion function* defined on the Cartesian product $P(U) \times P(U)$ with values in the interval [0, 1] measuring the degree of inclusion of sets. The lower AS_* and upper AS^* approximation operations can be defined in AS by

$$AS_*(X) = \{x \in U : \nu(I(x), X) = 1\} \tag{6}$$

$$AS^*(X) = \{x \in U : \nu(I(x), X) > 0\} \tag{7}$$

In the standard case, $I(x)$ is equal to the equivalence class $B(x)$ of the indiscernibility relation $I(B)$; in case of tolerance (similarity) relation $\tau \subseteq U \times U$, we take $I(x) = \{y \in U : x\tau y\}$ (i.e., $I(x)$ is equal to the tolerance class of τ defined by x). The standard inclusion relation is defined for X, $Y \subseteq U$ by

$$\nu(X, Y) = \begin{cases} \dfrac{|X \cap Y|}{|X|} & \text{if } X \quad \text{is non-empty} \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

For applications, it is important to have some constructive definitions of I and ν.

One can consider another way to define $I(x)$. Usually, together with AS we consider some set F of formulas describing sets of objects in the universe U of AS defined by semantics $\|\cdot\|_{AS}$, i.e., $\|\alpha\|_{AS} \subseteq U$ for any $\alpha \in F$. Now, one can take the set

$$N_F(x) = \{\alpha \in F : x \in \|\alpha\|_{AS}\} \tag{9}$$

and $I(x) = \{\|\alpha\|_{AS} : \alpha \in N_F(x)\}$. Hence, more general uncertainty functions having values in $P(P(U))$ can be defined. Usually, there are considered families of approximation spaces with approximation spaces labeled by some parameters. By tuning such parameters according to chosen criteria (e.g., minimal description length), one can search for the optimal approximation space for concept description.

The approach based on inclusion functions has been generalized to the *rough mereological approach* (8,12,17,37). The inclusion relation $x\mu_r y$ with the intended meaning *x is a part of y to a degree at least r* has been taken as the basic notion of the rough mereology being a generalization of the Leśniewski mereology (38,39). Research on rough mereology has shown importance of another notion, namely *closeness* of complex objects (e.g., concepts), which can be defined by $xcl_{r,r'}y$ if and only if $x\mu_r y$ and $y\mu_{r'}x$.

Rough mereology offers a methodology for synthesis and analysis of objects in distributed environment of intelligent agents, in particular, for synthesis of objects satisfying a given specification to a satisfactory degree or for control in such complex environments. Moreover, rough mereology has been recently used for developing foundations of the *information granule calculi*, aiming at formalization of the Computing with Words paradigm, recently formulated by Lotfi Zadeh (40). More complex information granules are defined recursively using already defined information granules and their measures of inclusion and closeness. Information granules can have complex structures like classifiers or approximation spaces. Computations on information granules are performed to discover relevant information granules (e.g., patterns or approximation spaces for complex concept approximations).

ROUGH SETS AND MEMBERSHIP FUNCTIONS

Rough sets can be also introduced using a *rough membership function*, defined by

$$\mu_X^B(x) = \frac{|X \cap B(x)|}{|B(x)|} \tag{10}$$

Obviously, $0 \leq \mu_X^B(x) \leq 1$. The membership function $\mu_X(x)$ is a kind of conditional probability and its value can be interpreted as a degree of *certainty* to which x belongs to X.

The rough membership function can be used to define approximations and the boundary region of a set, as shown below:

$$B_*(X) = \{x \in U : \mu_X^B(x) = 1\} \tag{11}$$

$$B^*(X) = \{x \in U : \mu_X^B(x) > 0\} \tag{12}$$

$$BN_B(X) = \{x \in U : 0 < \mu_X^B(x) < 1\} \tag{13}$$

One of the consequences of perceiving objects by information about them is that for some objects one cannot decide if they belong to a given set or not. However, one can estimate the degree to which objects belong to sets, which is a crucial observation in building foundations for approximate reasoning. Dealing with imperfect knowledge implies that one can only characterize satisfiability of relations between objects to a degree, not precisely. One of the fundamental relations on objects is a rough inclusion relation describing that objects are parts of other objects to a degree. Rough mereological approach (8,12,17,37) based on such relation is an extension of the Leśniewski mereology (38).

DECISION TABLES AND DECISION RULES

Sometimes we distinguish in an information table (U, A) a partition of A into two classes C, $D \subseteq A$ of attributes, called *condition* and *decision* (*action*) attributes, respectively. The tuple $\mathcal{A} = (U, C, D)$ is called a *decision table*.

Let $V = \bigcup\{V_a | a \in C\} \cup V_d$. Atomic formulas over $B \subseteq C \cup D$ and V are expressions $a = v$ called *descriptors (selectors)* over B and V, where $a \in B$ and $v \in V_a$. The set $\mathcal{F}(B, V)$ of formulas over B and V is the least set containing all atomic formulas over B and V and closed with respect to the propositional connectives $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation).

By $\|\varphi\|_{\mathcal{A}}$, we denote the meaning of $\varphi \in \mathcal{F}(B, V)$ in the decision table $\mathcal{A}$, which is the set of all objects in U with the property φ. These sets are defined by $\|a = v\|_{\mathcal{A}} = \{x \in U | a(x) = v\}$, $\|\varphi \wedge \varphi'\|_{\mathcal{A}} = \|\varphi\|_{\mathcal{A}} \cap \|\varphi'\|_{\mathcal{A}}$; $\|\varphi \vee \varphi'\|_{\mathcal{A}} = \|\varphi\|_{\mathcal{A}} \cup \|\varphi'\|_{\mathcal{A}}$; $\|\neg\varphi\|_{\mathcal{A}} = U - \|\varphi\|_{\mathcal{A}}$. The formulas from $\mathcal{F}(C, V)$, $\mathcal{F}(D, V)$ are called *condition formulas of $\mathcal{A}$* and *decision formulas of $\mathcal{A}$*, respectively.

Any object $x \in U$ belongs to a *decision class* $\| \bigwedge_{a \in D} a = a(x)\|_{\mathcal{A}}$ of $\mathcal{A}$.

All decision classes of $\mathcal{A}$ create a partition of the universe U.

A *decision rule* for $\mathcal{A}$ is any expression of the form $\varphi \Rightarrow \psi$, where $\varphi \in \mathcal{F}(C, V)$, $\psi \in \mathcal{F}(D, V)$, and $\|\varphi\|_{\mathcal{A}} \neq \emptyset$.

Formulas φ and ψ are referred to as the *predecessor* and the *successor* of decision rule $\varphi \Rightarrow \psi$. Decision rules are often called "*IF* ... *THEN* ... " rules.

Decision rule $\varphi \Rightarrow \psi$ is *true* in $\mathcal{A}$ if and only if $\|\varphi\|_{\mathcal{A}} \subseteq \|\psi\|_{\mathcal{A}}$. Otherwise, one can measure its *truth degree* by introducing some inclusion measure of $\|\varphi\|_{\mathcal{A}}$ in $\|\psi\|_{\mathcal{A}}$. It is important to note that an inclusion measure expressed by the confidence, widely used in data mining (41), has been considered by Lukasiewicz (42) a long time ago in studies on assigning fractional truth values to logical formulas. Given two unary predicate formulas $\alpha(x)$, $\beta(x)$, where x runs over a finite set U, Lukasiewicz proposes to assign to $\alpha(x)$ the value $\frac{|\|\alpha(x)\||}{|U|}$, where $\|\alpha(x)\| = \{x \in U : x \, \text{satisfies} \, \alpha\}$. The fractional value assigned to the implication $\alpha(x) \Rightarrow \beta(x)$ is then $\frac{|\|\alpha(x) \wedge \beta(x)\||}{|\|\alpha(x)\||}$ under the assumption that $\|\alpha(x)\| \neq \emptyset$.

Each object x of a decision table determines a *decision rule* $\bigwedge_{a \in C} a = a(x) \Rightarrow \bigwedge_{a \in D} a = a(x)$.

Decision rules corresponding to some objects can have the same condition parts but different decision parts. Such rules are called *inconsistent (nondeterministic, conflicting, possible)*; otherwise, the rules are referred to as *consistent* (*certain*, *sure*, *deterministic*, *nonconflicting*) rules. Decision tables containing inconsistent decision rules are called *inconsistent* (*nondeterministic*, *conflicting*); otherwise, the table is *consistent* (*deterministic*, *nonconflicting*).

Numerous methods have been developed for different decision rule generation that the reader can find in the literature on rough sets. Usually, one is searching for decision rules (semi) optimal with respect to some optimization criteria describing quality of decision rules in concept approximations.

In case of searching for concept approximation in an extension of a given universe of objects (sample), typical steps are the following. When a set of rules have been induced from a decision table containing a set of training examples, they can be inspected to see if they reveal any novel relationships between attributes that are worth pursuing for further research. Furthermore, the rules can be applied to a set of unseen cases in order to estimate their classificatory power. For a systematic overview of rule application methods, the reader is referred to literature.

DEPENDENCY OF ATTRIBUTES

Another important issue in data analysis is discovering dependencies between attributes. Intuitively, a set of attributes D depends totally on a set of attributes C, denoted $C \Rightarrow D$, if the values of attributes from C uniquely determine the values of attributes from D. In other words, D depends totally on C, if there exists a functional dependency between values of C and D. Formally dependency can be defined in the following way. Let D and C be subsets of A.

We will say that D *depends on* C in a *degree* $k (0 \leq k \leq 1)$, denoted $C \Rightarrow_k D$, if

$$k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|} \tag{14}$$

where

$$POS_C(D) = \bigcup_{X \in U/D} C_*(X) \tag{15}$$

called a *positive region* of the partition U/D with respect to C, is the set of all elements of U that can be uniquely classified to blocks of the partition U/D, by means of C.

If $k = 1$, we say that D *depends totally* on C, and if $k < 1$, we say that D *depends partially* (to *degree* k) on C.

The coefficient k expresses the ratio of all elements of the universe, which can be properly classified to blocks of the partition U/D, employing attributes C and will be called the *degree of the dependency*.

It can be easily seen that if D depends totally on C, then $I(C) \subseteq I(D)$,which means that the partition generated by C is finer than the partition generated by D. Notice that the concept of dependency discussed above corresponds to that considered in relational databases.

In summation D is *totally* (*partially*) dependent on C if *all* (*some*) elements of the universe U can be uniquely classified to blocks of the partition U/D, employing C.

REDUCTION OF ATTRIBUTES

We often face a question whether we can remove some data from a data table preserving its basic properties, that is, whether a table contains some superfluous data.

Let us express this idea more precisely.

Let $C, D \subseteq A$ be sets of condition and decision attributes, respectively.

We will say that $C' \subseteq C$ is a *D-reduct* (reduct with *respect* to *D*) of C if C' is a minimal subset of C such that

$$\gamma(C, D) = \gamma(C', D) \tag{16}$$

The intersection of all D-reducts is called a *D-core* (core with *respect* to D). As the core is the intersection of all reducts, it is included in every reduct (i.e., each element of the core belongs to some reduct). Thus, in a sense, the core is the most important subset of attributes, because none of its elements can be removed without affecting the classification power of attributes.

Many other kinds of reducts and their approximations are discussed in the literature. It turns out that they can be efficiently computed using heuristics based on Boolean reasoning approach.

DISCERNIBILITY AND BOOLEAN REASONING

Tasks collected under labels of data mining, knowledge discovery, decision support, pattern classification, and approximate reasoning require tools aimed at discovering in data of *templates* (*patterns*) and classifying them into certain *decision classes*. Templates are, in many cases, most frequent sequences of events, most probable events, regular configurations of objects, the decision rules of high-quality, standard reasoning schemes. Tools for discovering and classifying of templates are based on *reasoning* *schemes* rooted in various paradigms (43). Such patterns can be extracted from data by means of methods based on Boolean reasoning and discernibility.

The discernibility relation is closely related to indiscernibility and is one of the most important relations considered in rough set theory.

The ability to discern between perceived objects is important for constructing many entities like reducts, decision rules, or decision algorithms. In the classic rough set approach, the discernibility relation $DIS(B) \subseteq U \times U$ is defined by $xDIS(B)y$ if and only if $non(xI(B)y)$, which, however, is in general not the case for the generalized approximation spaces (one can define indiscernibility by $x \in I(y)$ and discernibility by $I(x) \cap I(y) = \emptyset$ for any objects x, y).

The idea of Boolean reasoning is based on construction for a given problem P of a corresponding Boolean function f_P with the following property: The solutions for the problem P can be decoded from prime implicants of the Boolean function f_P . Let us mention that to solve real-life problems, it is necessary to deal with Boolean functions having large number of variables.

A successful methodology based on the discernibility of objects and Boolean reasoning has been developed for computing of many important for applications entities like reducts and their approximations, decision rules, association rules, discretization of real value attributes, symbolic value grouping, searching for new features defined by oblique hyperplanes or higher-order surfaces, pattern extraction from data, as well as conflict resolution or negotiation.

Most of the problems related to generation of the above-mentioned entities are NP-complete or NP-hard. However, it was possible to develop efficient heuristics returning suboptimal solutions of the problems. The results of experiments on many datasets are very promising. They show very good quality of solutions generated by the heuristics in comparison with other methods reported in literature (e.g., with respect to the classification quality of unseen objects). Moreover, they are very efficient from the point of view of time necessary for computing of the solution. It is important to note that the methodology makes it possible to construct heuristics having a very important *approximation property*, which can be formulated as follows: Expressions generated by heuristics (i.e., implicants) *close* to prime implicants define approximate solutions for the problem.

CONCEPT APPROXIMATION

In this section, we consider the problem of approximation of concepts over a universe U^{∞}(concepts that are subsets of U^{∞}). We assume that the concepts are perceived only through some subsets of U^{∞}, called samples, which is a typical situation in the machine learning, pattern recognition, or data mining approaches (41,44). In this section, we explain the rough set approach to induction of concept approximations using the generalized approximation spaces of the form $AS = (U, I, \nu)$ defined earlier.

Let $U \subseteq U^{\infty}$ be a finite sample. By Π_{U}, we denote a perception function from $P(U^{\infty})$ into $P(U)$ defined by $\Pi_U(C) = C \cap U$ for any concept $C \subseteq U^{\infty}$. Let $AS = (U, I, \nu)$ be an approximation space over the sample U.

The problem we consider is how to extend the approximations of $\Pi_U(C)$ defined by AS to approximation of C over U^{∞}. We show that the problem can be described as searching for an extension $AS_C = (U^{\infty}, I_C, \nu_C)$ of the approximation space AS, relevant for approximation of C, which requires to show how to extend the inclusion function ν from subsets of U to subsets of U^{∞}that are relevant for the approximation of C. Observe that, for the approximation of C, it is enough to induce the necessary values of the inclusion function ν_C without knowing the exact value of $I_C(x) \subseteq U^{\infty}$ for $x \in U^{\infty}$.

Let AS be a given approximation space for $\Pi_U(C)$ and let us consider a language L in which the neighborhood $I(x) \subseteq U$ is expressible by a formula $pat(x)$, for any $x \in U$. It means that $I(x) = \|pat(x)\|_U \subseteq U$, where $\|pat(x)\|_U$ denotes the meaning of $pat(x)$ restricted to the sample U. In case of rule-based classifiers, patterns of the form $pat(x)$ are defined by feature value vectors.

We assume that for any new object $x \in U^{\infty} \backslash U$ we can obtain (e.g., as a result of sensor measurement) a pattern $pat(x) \in L$ with semantics $\|pat(x)\|_{U^{\infty}} \subseteq U^{\infty}$. However, the relationships between information granules over U^{∞} like-sets $\|pat(x)\|_{U^{\infty}}$ and $\|pat(y)\|_{U^{\infty}}$, for different x, $y \in U^{\infty}$, are, in general, known only if they can be expressed by relationships between the restrictions of these sets to the sample U [i.e., between sets $\Pi_U(\|pat(x)\|_{U^{\infty}})$ and $\Pi_U(\|pat(y)\|_{U^{\infty}})$].

The set of patterns $\{pat(x) : x \in U\}$ is usually not relevant for approximation of the concept $C \subseteq U^{\infty}$. Such patterns are too specific or not enough general and can directly be applied only to a very limited number of new objects. However, by using some generalization strategies, one can search, in a family of patterns definable from $\{pat(x) : x \in U\}$ in L, for such new patterns that are relevant for approximation of concepts over U^{∞}. Let us consider a subset $PATTERNS(AS, L, C) \subseteq L$ chosen as a set of pattern candidates for relevant approximation of a given concept C. For example, in case of a rule-based classifier, one can search for such candidate patterns among sets definable by subsequences of feature value vectors corresponding to objects from the sample U. The set *PATTERNS(AS,L,C)* can be selected by using some quality measures checked on meanings (semantics) of its elements restricted to the sample U (like the number of examples from the concept $\Pi_U(C)$ and its complement that support a given pattern). Then, on the basis of properties of sets definable by these patterns over U, we induce approximate values of the inclusion function ν_C on subsets of U^{∞}definable by any of such pattern and the concept C.

Next, we induce the value of ν_C on pairs (*X,Y*), where $X \subseteq U^{\infty}$ is definable by a pattern from $\{pat(x) : x \in U^{\infty}\}$ and $Y \subseteq U^{\infty}$ is definable by a pattern from *PATTERNS* (*AS,L,C*).

Finally, for any object $x \in U^{\infty} \backslash U$, we induce the approximation of the degree $\nu_C(\|pat(x)\|_{U^{\infty}}, C)$ applying a conflict resolution strategy *Conflict_res* (a voting strategy, in case of rule-based classifiers) to two families of degrees:

$$\{\nu_C(\|pat(x)\|_{U^\infty}, \|pat\|_{U^\infty}) : pat \in PATTERNS\,(AS, L, C)\} \tag{17}$$

$$\{\nu_C(\|pat\|_{U^\infty}, C)\ :\ pat \in PATTERNS\,(AS, L, C)\} \tag{18}$$

Values of the inclusion function for the remaining subsets of U^∞ can be chosen in any way—they do not have any impact on the approximations of C. Moreover, observe that, for the approximation of C, we do not need to know the exact values of uncertainty function I_C—it is enough to induce the values of the inclusion function ν_C. Observe that the defined extension ν_C of ν to some subsets of U^∞ makes it possible to define an approximation of the concept C in a new approximation space AS_C .

In this way, the rough set approach to induction of concept approximations can be explained as a process of inducing a relevant approximation space.

MEREOLOGY AND ROUGH MEREOLOGY

Exact and rough concepts can be characterized by a new notion of an element, alien to naive set theory in which this theory has been coded until now. For an information system $\mathcal{A} = (U, A)$ and a set B of attributes, the mereological element $el_B^{\mathcal{A}}$ is defined by letting

$$x el_B^{\mathcal{A}} X \text{ if and only if } B(x) \subseteq X \tag{19}$$

Then, a concept X is B-exact if and only if either $x el_B^{\mathcal{A}} X$ or $x el_B^{\mathcal{A}}\, U \backslash X$ for each $x \in U$, and the concept X is B–rough if and only if for some $x \in U$ neither $x el_B^{\mathcal{A}} X$ or $x el_B^{\mathcal{A}}\, U \backslash X$.

Thus, the characterization of the dychotomy exact–rough cannot be done by means of the element notion of naive set theory, but it requires the notion of containment ($\subseteq$)(i.e., a notion of mereological element).

The Leśniewski Mereology (theory of parts) is based on the notion of a part (38,39). The relation π of part on the collection U of objects satisfies,

1. if $x\pi y$ then not $y\pi x$,
2. if $x\pi y$ and $y\pi z$ then $x\pi z$.

The notion of mereological element el_π is introduced as,

$$x el_\pi y \text{ if and only if } x\pi y \text{ or } x = y \tag{20}$$

In particular, the relation of proper inclusion $\subset$ is a part relation π on any non-empty collection of sets, with the element relation $el_\pi = \subseteq$.

Formulas expressing rough membership, quality of decision rule, quality of approximations, and so on can be traced back to a common root (i.e., $\mu(X,Y)$ defined by Equation (8). The value $\mu(X,Y)$ defines the degree of *partial containment* of X into Y and naturally refers to the Leśniewski Mereology. An abstract formulation of this idea in Ref. 37 connects the mereological notion of element el_π with this idea of partial inclusion in the idea of a *rough inclusion* as a relation $\mu \subseteq U \times U \times [0, 1]$ on a collection of pairs of objects in U endowed with part π relation, and such that

1. $\mu(x, y, 1)$ if and only if $x el_\pi y$,
2. if $\mu(x, y, 1)$ then (if $\mu(z, x, r)$ then $\mu(z, y, r)$),
3. if $\mu(z, x, r)$ and $s < r$ then $\mu(z, x, s)$.

Implementation of this idea in information systems can be based on *archimedean* t–norms (37); each such norm T is represented as $T(r, s) = g(f(r) + f(s))$ with f, g pseudo–inverses to each other, continuous and decreasing on [0, 1]. Letting for (U, A) and $x, y \in U$,

$$DIS(x, y) = \{a \in A : a(x) \neq a(y)\} \tag{21}$$

and

$$\mu(x, y, r) \text{ if and only if } g\left(\frac{|DIS(x, y)|}{|A|}\right) \geq r \tag{22}$$

defines a rough inclusion that satisfies additionally the transitivity rule

$$\frac{\mu(x, y, r), \mu(y, z, s)}{\mu(x, z, T(r, s))} \tag{23}$$

Simple examples here are as follows: The Menger rough inclusion in the case $f(r) = -lnr$, $g(s) = e^{-s}$ yields $\mu(x,y,r)$ if and only if $e^{-\frac{|DIS\,(x,y)|}{|A|}} \geq r$ and it satisfies the transitivity rule

$$\frac{\mu(x, y, r), \mu(y, z, s)}{\mu(x, y, r \cdot s)} \tag{24}$$

where the t–norm T is the Menger (product) t–norm $r \cdot s$, and the Lukasiewicz rough inclusion with $f(x) = 1 - x = g(x)$ yielding $\mu(x,y,r)$ if and only if $1 - \frac{|DIS(x,y)|}{|A|} \geq r$ with the transitivity rule

$$\frac{\mu(x, y, r), \mu(y, z, s)}{\mu(x, y, max\{0, r + s - 1\})} \tag{25}$$

with the Lukasiewicz t–norm.

Rough inclusions (37) can be used in *granulation of knowledge* (40). Granules of knowledge are constructed as aggregates of indiscernibility classes close enough with respect to a chosen measure of closeness. In a nutshell, a granule $g_r(x)$ about x of radius r can be defined as the aggregate of all y with $\mu(y, x, r)$. The aggregating mechanism can be based on class operator of mereology (like in rough mereology (37)) or on set theoretic operations of union.

Rough mereology (37) combines rough inclusions with methods of mereology. It employs the operator of mereological class that makes collections of objects into objects. The class operator Cls satisfies the requirements, with any non–empty collection M of objects made into the object

$Cls(M)$,

$$\text{if } x \in M, \text{ then } x el_{\pi} Cls(M) \quad (26)$$

$$\text{if } x el_{\pi}\, Cls\,(M), \text{then there exist } y, z \text{ such that } y el_{\pi} x,\ y el_{\pi} z, z \in M \quad (27)$$

In case of the part relation $\subset$ on a collection of sets, the class *Cls(M)* of a non–empty collection M is the union $\bigcup M$.

Granulation by means of the class operator *Cls* consists in forming the granule *$g_r(x)$* as the class $Cls(y : \mu(y, x, r))$. One obtains a granule family with regular properties (see Ref. 36).

RESEARCH DIRECTIONS IN ROUGH SET THEORY

In this section, we present a list of research directions on the rough set foundations and the rough set-based methods. For more details, the reader is referred to the bibliography on rough sets.

List of research directions on rough sets

- Boolean reasoning and approximate Boolean reasoning strategies as the basis for efficient heuristics for rough set methods.
- Tolerance (similarity)-based rough set approach.
- Rough set-based approach based on neighborhood (uncertainty) functions and inclusion relation, in particular, variable precision rough set model.
- Rough sets in multi-criteria decision analysis and preference modeling.
- Recurrent rough sets.
- Rough sets and nondeterministic information systems.
- Rough set-based clustering.
- Rough sets and incomplete information systems, in particular, missing value problems.
- Rough sets and noisy data.
- Rough sets and relational databases.
- Rough sets and inductive reasoning.
- Rough sets in modeling of decision systems and analysis of complex systems, in particular, rough sets and layered (hierarchical) learning.
- Rough sets as a tool for approximate reasoning in distributed systems, by autonomous agents, and in multiagent systems.
- Rough mereology foundations, in particular, rough mereological approach to synthesis and analysis of complex objects.
- Rough sets and rough mereology in granular computing. In particular:
  - modeling of approximation spaces for granular computing;
  - calculi of information granules;
  - approximate reasoning schemes and networks;
  - complex concept approximation from experimental data and domain knowledge;
  - spatio-temporal reasoning;
  - classification of complex objects and prediction;
  - rough set and rough mereological approach to computing with words and perception;
  - rough-neural computing.
- Relationships of rough sets to other approaches of reasoning under incomplete information like Dempster–Shafer theory of evidence, fuzzy sets, mathematical morphology, statistical inference, Bayesian reasoning, rough sets, and Petri nets.
- Logical calculi based on rough sets like specific modal, 3-valued, or information logics.
- Relationships of rough sets with logic programming.
- Algebraic structures corresponding to calculi on rough sets and logics on rough sets like quasi–Boolean algebras, double Stone algebras, Nelson algebras, Heyting algebras, and Wajsberg algebras.
- Relational calculi and rough sets.
- Topological aspects of rough sets.
- Philosophical aspects of rough sets.
- Hybridization of rough sets with soft computing approaches, in particular, with fuzzy sets, neural networks, genetic algorithms, and evolutionary computing.
- Rough sets in machine learning.
- Rough sets in pattern recognition.
- Rough sets in data mining and knowledge discovery.
- Rough sets and case-based reasoning.
- Rough sets and membrane computing and other molecular biology– inspired calculi.
- Rough sets and formal concept analysis.

A CHALLENGE FOR RESEARCH ON ROUGH SETS

There are many real-life problems that are still hard to solve using the existing methodologies and technologies. Among such problems are, for examples, classification of medical images, control of autonomous systems like unmanned aerial vehicles or robots, or problems related to monitoring or rescue tasks in multiagent systems. All these problems are closely related to intelligent systems that are more and more widely applied in different real-life projects.

One of the main challenges in developing of intelligent systems are methods for approximate reasoning from measurements to perception (i.e., from concepts close to sensor measurements to concepts expressed in natural language by human beings that are the perception results).

Today, new emerging computing paradigms are investigated attempting to make progress in solving problems related to this challenge. Further progress depends on a successful cooperation of specialists from different scientific disciplines such as mathematics, computer science, artificial intelligence, biology, physics, chemistry, bioinformatics, medicine, neuroscience, linguistics, psychology, and sociology. In particular, different aspects of reasoning from measurements to perception are investigated in psychology (45,46), neuroscience (47), layered learning (48), mathematics of learning (47), machine learning, pattern recognition (44), data mining (41), and also by researchers working on recently emerged computing paradigms, like computing with words and perception (40), granular computing (17), rough sets, rough-mereology, and rough-neural computing (17).

One of the main problems investigated in machine learning, pattern recognition (44), and data mining (41) is concept approximation. It is necessary to induce approximations of concepts (models of concepts) from available experimental data. The data models developed so far in such areas like statistical learning, machine learning, and pattern recognition are not satisfactory for approximation of complex concepts resulting in the perception process. Researchers from the different areas have recognized the necessity to work on new methods for concept approximation (see Refs. 49 and 50). The main reason is that these complex concepts are, in a sense, too far from measurements, which makes the searching for relevant (for their approximation) features infeasible in a huge space. There are several research directions aiming at overcoming this difficulty, one of which is based on the interdisciplinary research where the results concerning perception in psychology or neuroscience are used to help to deal with complex concepts (see Ref. 44). There is a great effort in neuroscience toward understanding the hierarchical structures of neural networks in living organisms (47,51). Also, mathematicians are recognizing problems of learning as the main problem of the current century (47).

The problems discussed so far are also closely related to complex system modeling. In such systems, the problem of concept approximation and reasoning about perceptions using concept approximations is one of the main challenges. One should take into account that modeling complex phenomena entails the use of local models (captured by local agents, if one would like to use the multiagent terminology (52)) that next should be fused. This process involves the negotiations between agents (52) to resolve contradictions and conflicts in local modeling. This kind of modeling will become more and more important in solving complex real-life problems that we are unable to model using traditional analytical approaches. The latter approaches lead to exact models. However, the necessary assumptions used to develop these models are causing the resulting solutions to be too far from reality to be accepted. New methods or even a new science should be developed for such modeling (53). One of the possible solutions in searching for methods for complex concept approximations is the layered learning idea (48). Inducing concept approximation should be developed hierarchically starting from concepts close to sensor measurements to complex target concepts related to perception. This general idea can be realized using an additional domain knowledge represented in natural language. For example, one can use principles of behavior on the roads, expressed in natural language, trying to estimate, from recordings (made, e.g., by camera and other sensors) of situations on the road, whether the current situation on the road is safe. To solve such a problem, one should develop methods for concept approximations together with methods aiming at approximation of reasoning schemes (over such concepts) expressed in natural language. Foundations of such approach are based on rough set theory (2) and its extension rough mereology (8,12,17,37), both discovered in Poland.

Objects we are dealing with are information granules. Such granules are obtained as the result of information granulation (40). Information granulation can be viewed as a human way of achieving data compression, and it plays a key role in implementation of the strategy of divide-and-conquer in human problem solving.

Computing with Words and Perception "derives from the fact that it opens the door to computation and reasoning with information which is perception-rather than measurement-based. Perceptions play a key role in human cognition, and underlie the remarkable human capability to perform a wide variety of physical and mental tasks without any measurements and any computations. Everyday examples of such tasks are driving a car in city traffic, playing tennis and summarizing a story" (40). The rough mereological approach (8,12,17,37) is based on calculi of information granules for constructing complex concept approximations. Constructions of information granules should be robust with respect to their input information granule deviations. In this way, a granulation of information granule constructions is considered. As a result, we obtain the so-called AR schemes (AR networks) (8,12,17,37). AR schemes can be interpreted as complex patterns (41). Searching methods for such patterns relevant for a given target concept have been developed (17). Methods for deriving relevant AR schemes are of high computational complexity. The complexity can be substantially reduced by using domain knowledge. In such a case, AR schemes are derived along reasoning schemes in natural language that are retrieved from domain knowledge. Developing methods for deriving such AR schemes is one of the main goals of our projects.

The outlined research directions create foundations toward understanding the nature of reasoning from measurements to perception is a challenge and crucial for constructing intelligent systems for many real-life projects.

CONCLUSIONS

In this article, basic concepts of rough set theory are presented. It has turned out, however, that the "basic model" of rough sets—presented here—has not been sufficient for many applications and is in need of some extensions. Besides, theoretical inquiry into the rough set concept also has led to its various generalizations. Some of them have been mentioned in the article.

A variety of methods for decision rules generation, reducts computation, and continuous variable discretization are very important issues not discussed here. We have only emphasized the developed powerful methodology based on discernibility and Boolean reasoning for efficient computation of different entities including reducts and decision rules. Also, the relationship of rough set theory to many other theories has been extensively investigated. In particular, its relationships to fuzzy set theory, the theory of evidence, Boolean reasoning methods, statistical methods, and decision theory have been clarified and seem now to be thoroughly understood.

There are reports on many hybrid methods obtained by combining the rough set approach with other approaches such as fuzzy sets, neural networks, genetic algorithms, principal component analysis, and singular value decomposition.

Recently, it has been shown that the rough set approach can be used for synthesis and analysis of concept approximations in the distributed environment of intelligent agents. We outlined the rough mereological approach and its applications in information granules calculi for synthesis of information granules satisfying a given specification to a satisfactory degree.

Readers interested in the above issues are advised to consult the enclosed references.

Many important research topics in rough set theory such as various logics related to rough sets and many advanced algebraic properties of rough sets were only mentioned in the article. The reader can find details in the cited books, articles, and journals.

Finally, we have outlined a challenge for research on rough sets related to approximate reasoning from measurements to perception.

ACKNOWLEDGMENTS

The authors would like to thank Professor James Peters for his valuable comments on a draft version of the article.

The research of Andrzej Skowron has been supported by the Ministry of Scientific Research and Information Technology of the Republic of Poland. Lech Polkowski was supported by grants from Polish-Japanese Institute of Information Technology and University of Warmia and Mazury.

BIBLIOGRAPHY

1. L.A.: Zadeh, Fuzzy sets, *Inform. Control*, **8**: 338–353. 1965.
2. Z. Pawlak, Rough Sets: Theoretical Aspects of Reasoning about Data. Volume 9 of System Theory, Knowledge Engineering and Problem Solving. Dordrecht, The Netherlands Kluwer Academic Publishers, 1991.
3. G. Frege, Grundgesetzen der Arithmetik, 2. Jena, Germany: Verlag von Hermann Pohle, 1903.
4. S. Read, Thinking about Logic: An Introduction to the Philosophy of Logic, New York: Oxford University Press, 1994.
5. R. Słowiński, (ed.), Intelligent Decision Support – Handbook of Applications and Advances of the Rough Sets Theory. Volume 11 of D: System Theory, Knowledge Engineering and Problem Solving. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1992.
6. T.Y. Lin, N. Cercone, (eds.), Rough Sets and Data Mining – Analysis of Imperfect Data, Boston, MA: Kluwer Academic Publishers, 1997.
7. E. Orłowska (ed.), Incomplete Information: Rough Set Analysis. Volume 13 of Studies in Fuzziness and Soft Computing, Heidelberg, Germany: Springer-Verlag/Physica-Verlag, 1997.
8. L. Polkowski, A. Skowron, (eds.), Rough Sets in Knowledge Discovery 1: Methodology and Applications. Volume 18 of Studies in Fuzziness and Soft Computing, Heidelberg, Germany: Physica-Verlag, 1998.
9. L. Polkowski, A. Skowron, (eds.), Rough Sets in Knowledge Discovery 2: Applications, Case Studies and Software Systems, Volume 19 of Studies in Fuzziness and Soft Computing, Heidelberg, Germany: Physica-Verlag, 1998.
10. S.K. Pal, A. Skowron, (eds.), Rough Fuzzy Hybridization: A New Trend in Decision-Making, Singapore: Springer-Verlag, 1999.
11. I. Duentsch, G. Gediga, Rough set data analysis: A road to noninvasive knowledge discovery, Bangor, UK: Methodos Publishers, 2000.
12. L. Polkowski, T.Y. Lin, S. Tsumoto, (eds.), Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems. Volume 56 of Studies in Fuzziness and Soft Computing, Heidelberg, Germany: Springer-Verlag/ Physica-Verlag, 2000.
13. T.Y. Lin, Y.Y. Yao, L.A. Zadeh, (eds.), Rough Sets, Granular Computing and Data Mining. Studies in Fuzziness and Soft Computing, Heidelberg, Germany: Physica-Verlag, 2001.
14. L. Polkowski, Rough Sets: Mathematical Foundations. Advances in Soft Computing, Heidelberg, Germany: Physica-Verlag, 2002.
15. S. Demri, E. Orłowska, (eds.), Incomplete Information: Structure, Inference, Complexity, Monographs in Theoretical Computer Science, Heidelberg, Germany: Springer-Verlag, 2002.
16. M. Inuiguchi, S. Hirano, S. Tsumoto, (eds.), Rough Set Theory and Granular Computing. Volume 125 of Studies in Fuzziness and Soft Computing, Heidelberg, Germany: Springer-Verlag, 2003.
17. S.K. Pal, L. Polkowski, A. Skowron, (eds.), Rough-Neural Computing: Techniques for Computing with Words, Cognitive Technologies, Heidelberg, Germany: Springer-Verlag, 2003.
18. R. Słowínski, J. Stefanowski, (eds.), Proc. of the First International Workshop on Rough Sets: State of the Art and Perspectives, Kiekrz, Poznán, Poland, 1992.
19. W. Ziarko, (ed.), Special issue, *Intell. Int.J.*, **11 (2)**, 1995.
20. W. Ziarko,(ed.), Special issue, *Fundamenta Informaticae*, **27** 2–3, 1996.
21. T.Y. Lin, (ed.), Special issue. *J. of the Intell. Automation and Soft Computing*, **2** (2): 1996.
22. J.Peters, A.Skowron, (eds.), Special issue on a rough set approach to reasoning about data, *Internat. J. of Intell.* Syst., **16(1)**: 2001.
23. N. Cercone, A. Skowron, N. Zhong (eds.), Special issue, *Computat. Intell.* **17 (3)**: 2001.
24. S.K.Pal, W. Pedrycz, A. Skowron, R. Swiniarski, (eds.), Special volume: Rough-neuro computing, *Neurocomputing* **36:** 2001.
25. A.Skowron, S.K.Pal,(eds.), Special volume: Rough sets, pattern recognition and data mining, *Pattern Recog. Lett.* **24**(6) 2003.
26. W. Ziarko, (ed.), Rough Sets, Fuzzy Sets and Knowledge Discovery: *Proc. of the Second International Workshop on Rough Sets and Knowledge Discovery (RSKD'93)*, Banff, Alberta, Canada, 1993.
27. T.Y. Lin, A.M. Wildberger, (eds.), Soft Computing: Rough Sets, Fuzzy Logic, Neural Networks, Uncertainty Management, Knowledge Discovery, San Diego,CA: Simulation Councils, Inc., 1995.
28. S. Tsumoto, S. Kobayashi, T. Yokomori, H. Tanaka, A. Nakamura (eds.), *Proc. of the The Fourth Internal Workshop on Rough Sets, Fuzzy Sets and Machine Discovery*, University of Tokyo, Japan, 1996.

29. L. Polkowski, A. Skowron, (eds.), *First International Conference on Rough Sets and Soft Computing RSCTC*. Warsaw, Poland, Springer-Verlag, 1998.
30. A. Skowron, S. Ohsuga, N. Zhong (eds.), *Proc. of the 7-th International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing (RSFDGrC'99)*, Yamaguchi, Japan, 1999.
31. W. Ziarko, Y. Yao, (eds.), *Proc. of the 2nd International Conference on Rough Sets and Current Trends in Computing (RSCTC'2000)*, Banff, Canada, 2000.
32. S. Hirano, M. Inuiguchi, S. Tsumoto, (eds.), *Proc. of International Workshop on Rough Set Theory and Granular Computing (RSTGC-2001)*, Matsue, Shimane, Japan, 2001.
33. T. Terano, T. Nishida, A. Namatame, S. Tsumoto, Y. Ohsawa, T. Washio, (eds.), *New Frontiers in Artificial Intelligence, Joint JSAI 2001 Workshop Post-Proceedings*, 2001.
34. J.J. Alpigini, J.F. Peters, A. Skowron, N. Zhong, (eds.), *Third International Conference on Rough Sets and Current Trends in Computing (RSCTC'02)*, Malvern, PA, 2002.
35. A. Skowron, M. Szczuka (eds.), *Proc. of the Workshop on Rough Sets in Knowledge Discovery and Soft Computing at ETAPS*, 2003.
36. G. Wang, Q. Liu, Y. Yao, A. Skowron, (eds.), *Proc. of the 9th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing (RSFDGrC'03)*, Chongqing, China, 2003.
37. L. Polkowski, A. Skowron, Rough mereology: A new paradigm for approximate reasoning. *International Journal of Approximate Reasoning*, **15**: 333–365, 1996.
38. S. Leśniewski, Grungźuge eines neuen systems der grundlagen der mathematik, *Fundamenta Matematicae*, **14**: 1–81, 1929.
39. S. Leśniewski, On the foundations of mathematics, *Topoi* **2**: 7–52, 1982.
40. L.A. Zadeh, A new direction in AI: Toward a computational theory of perceptions, *AI Magazine* **22:** 73–84, 2001.
41. W. Kloesgen, J. Zytkow, (eds.), Handbook of Knowledge Discovery and Data Mining, Oxford: Oxford University Press, 2002.
42. J. Łukasiewicz, Die logischen grundlagen der wahrscheinilchkeit srechnung, 1913, in L. Borkowski, (ed.), Jan Łukasiewicz – Selected Works, Amstardam, London, North Holland Publishing Company, Polish Scientific Publishers, 1970.
43. R. Duda, P. Hart, R. Stork, (eds.), Pattern Classification, New York: Wiley, 2002.
44. J.H. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Heidelberg, Germany: Springer-Verlag, 2001.
45. L.W. Barsalou, Perceptual symbol systems, *Behavioral and Brain Sciences*, **22**: 577–660, 1999.
46. S. Harnad, Categorical Perception: The Groundwork of Cognition, New York: Cambridge University Press, 1987.
47. T. Poggio, S. Smale, The mathematics of learning: Dealing with data, *Notices of the AMS*, **50**: 537–544, 2003.
48. P. Stone, Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer, Cambridge, MA: The MIT Press, 2000.
49. L. Breiman, Statistical modeling: The two cultures, *Statistical Science*, **16**: 199–231, 2001.
50. V. Vapnik, Statistical Learning Theory, New York: Wiley, 1998.
51. M. Fahle, T. Poggio, Perceptual Learning, Cambridge, MA: MIT Press, 2002.
52. M. Huhns, M. Singh, Readings in Agents, San Mateo, CA: Morgan Kaufmann, 1998.
53. M. Gell-Mann, *The Quark and the Jaguar – Adventures in the Simple and the Complex*, London: Brown and Co., 1994.

ZDZISŁAW PAWLAK*
Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, and University of Information Technology and Management
Warsaw, Poland

LECH POLKOWSKI
Polish–Japanese Institute of Information Technology
Warsaw, Poland
University of Warmia and Mazury
Olsztyn, Poland

ANDRZEJ SKOWRON
Institute of Mathematics, Warsaw University
Warsaw, Poland

*Deceased

AD HOC AND SENSOR NETWORKS

INTRODUCTION

In a wireless ad hoc network, each node is equipped with one or more wireless radio transceivers. A node can communicate with nodes in its radio range directly (called *single-hop* wireless); otherwise, it relies on the intermediate nodes to relay its message to a non-neighbor node. The latter mechanism is called *multi-hop* wireless communication. In contrast, infrastructure-based networks such as wireless local area networks (WLANs) and cellular networks use only single-hop wireless communication. Moreover, a wireless ad hoc network usually does not have special-purpose relay nodes similar to routers in conventional networks — every node is a potential router (that can relay the data of other nodes). Furthermore, the network topology of a wireless ad hoc network is usually much more dynamic than that of a conventional network, because of node failures and/or node mobility.

A wireless ad hoc network has several benefits over wired infrastructure-based networks, which make it a compelling choice for networking in certain application scenarios. We discuss some of these benefits here:

- **Quick to Deploy**: A wireless ad hoc network, by definition, does not require an existing infrastructure, such as wall power or wiring. This quality significantly reduces the time to deploy a wireless ad hoc network and have it up and running. Sometimes, the time to deploy may be minutes or seconds as opposed to days or weeks for an infrastructure-based network.
- **Suitable for a Wider Range of Environment**: A wireless ad hoc network can be deployed easily in remote places such as in forests, under water (e.g., rivers, oceans, etc.), on mountain tops, on moving troops in a battlefield, in toxic areas, or on other planets.
- **More Resilient to Failures**: Wireless ad hoc networks typically are more resilient to failures than infrastructure-based networks. This resihence is because communication among nodes can be over multiple hops using intermediate nodes (each of which can act as a router) and because the protocols developed do not assume any existence of infrastructure. Most ad hoc network protocols are/can be designed to reconfigure quickly upon failure; therefore, communication among surviving nodes is possible even if several or the majority of nodes have failed (as in a battlefield scenario).
- **Offers Freedom of Mobility**: Because no wiring exists among nodes, the nodes in a wireless ad hoc network can move freely and still maintain communication with other nodes in the network. The protocols also are designed to adapt quickly to mobility (which may cause frequent changes in the set of neighbors). This characteristic makes ad hoc networks especially useful in mobile applications such as among a group of moving soldiers, a fleet of moving vehicles, and a fleet of aircrafts flying together, as well as in disaster locations.
- **Economical**: Because of its low set-up overhead (e.g., no wiring and labor), deploying a wireless ad hoc network is more economical compared with its wired counterpart in several application scenarios.

Although wireless ad hoc networks have several advantages over infrastructure-based networks, they have their own limitations. For example, the nodes usually have limited lifetime because they typically run on batteries. Moreover, it is often more challenging to develop efficient protocols for the wireless ad hoc networks, because of the mobility and the limitations of the wireless communication medium.

wireless ad hoc networks can be classified into several categories. Below are three major categories of ad hoc networks, each of which has become a fertile research area in its own right:

1. **Mobile Ad Hoc Network (MANET)**: A MANET is a network of mobile computing devices, such as laptops and PDAs. The purpose of forming a MANET is to facilitate communication among mobile host devices that make up this network.
2. **Wireless Sensor Network (WSN)**: WSNs are composed of small wireless sensors, such as motes (2) that can monitor their surrounding physical environment by using various on-board sensors. The purpose of a WSN is to monitor the environment in which it is embedded either to collect data of interest or to detect events of interest, such as monitoring its surrounding for illegal intrusion activity.
3. **Wireless Mesh Networks**: A wireless mesh network consists of wireless devices mainly used as routers to provide a wireless infrastructure to other devices. A wireless mesh network can provide Internet access to computational devices, such as laptops and PDAs, without having to deploy a wired infrastructure.

Each of the above categories can be classified additionally based on the devices and communication technology it employs. For example, the mobile devices in a MANET can be laptops, PDAs, or even cell phones, and the communication technology used by these devices can be 802.11, Bluetooth, ZigBee, and so forth.

Our focus in this article is on the first two categories–MANETs and WSNs (see Ref. (3) for a survey of wireless mesh networks). MANETs and WSNs share several key characteristics; both of them rely on the wireless medium for communication and use multi-hop wireless routing. However, they have important differences as well because

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

of the intrinsic differences in their potential applications. For example, typical applications of MANETs include communication on a battlefield and during disaster recovery; therefore, research on MANETs has been focusing on supporting human communication in the face of unconstrained mobility. Alternately, WSNs are used to monitor the physical environment, such as natural habitats and volcanoes, as well as to detect intrusions in a highly secure area. The sensor nodes usually are stationary and need to last months without human intervention, so energy-efficiency is a critical issue for WSNs, whereas ensuring connectivity despite user-induced (and hence uncontrolled) mobility has not been a major focus of WSN research.

In the remainder of this article, we first discuss the common issues in MANETs and WSNs, all of which are important for wireless mesh networks, as well. Then, we describe differences between MANETs and WSNs. Finally, we conclude the article.

COMMON ISSUES IN WIRELESS AD HOC NETWORKS

The Issue of Connectivity

A wireless ad hoc network, by its very definition, does not have a preplanned network topology. At the same time, the network needs to facilitate communication among different nodes. For this to be possible, the network needs to have some form of connectivity. Depending on the particular network, we may want all the nodes in the network to form a connected graph, most of the nodes to form a connected graph, or the network to provide delay-tolerant connectivity (4). Sometimes, for fault tolerance or to balance the routing load among nodes, k-connectivity in the network may be desired such that k node-disjoint paths exist between every pair of nodes.

Connectivity (or k-connectivity) can be made possible by increasing the density of nodes, by adjusting the transmission range of individual nodes, or by moving (controlled or uncontrolled) the nodes. If the nodes are mostly static and their location distribution can be approximated by a Poisson process or a random uniform process, then a critical relation exists between the transmission range and node density (5,6). Given one of these two parameters, the other can be derived. If the density is given, then the required transmission is called critical, and vice versa. The term critical (in say transmission range) intuitively means that if the transmission range is less than the critical value, then the network is disconnected with high probability. However, if the transmission range is higher than the critical value, then the network is connected with high probability. The connectivity of the network is said to have a phase transition (from disconnected to connected) at this critical value.

When nodes are mobile, similar results exist for the critical relation between the transmission range and density (7). These results assume that node movements can be approximated by certain mobility models.

Although critical density provides a guidance as to what behavior can be expected from a randomly deployed network in terms of connectivity, the results are not directly usable by a practitioner who would like to have a guarantee on connectivity for finite deployment regions. This situation is because the results derived for critical density are asymptotic, by definition. Recently, a new technique has been proposed to derive density estimates for random deployments that are quite reliable for finite deployment regions (8). Such work bridges the gap between theory and practice in the area of connectivity because theoretical results now can be readily used in practice.

Another area of research that has received considerable attention is called “Delay-tolerant connectivity.” It means that the network may not be connected at every instant in time, but movement of nodes may facilitate occasional communication among pairs of disconnected nodes. In the extreme case, data mules (9) may be deployed whose sole purpose is to ferry data between source–destination pairs. In other scenarios, nodes that have data packets destined to another node may wait till they come in direct contact with each other or pass the message to one of their current neighbors, who repeats the process until the data reaches its destination or until its time to live runs out, in which case it is dropped. Figuring out a good approach for message delivery in a mobile network that is not always connected currently is a highly active area of research.

Distributed Medium Access Control

Because the wireless medium is broadcast in nature, collision can occur when two nodes within the transmission range of each other send packets at the same time. In an infrastructure-based wireless network, such as a cell phone network, the access point (or base station) can allocate a different frequency band or a different transmission slot to each node in the same cell. However, in an ad hoc network, no centralized controller exists. Therefore, the first issue that needs to be addressed in any wireless ad hoc network is how to coordinate the transmissions of different nodes without using a centralized controller. The major objectives are to avoid collision (that may lead to loss of all colliding messages and hence loss of bandwidth) in a distributed manner, make efficient use of scarce wireless bandwidth, ensure fairness among the nodes, provide real-time guarantees to high-priority packets, and achieve all these tasks with a mechanism that scales to large network sizes. A protocol that achieves these objectives (or a subset of these) in a distributed manner is called a Distributed Medium Access Control (MAC) protocol. [1]

Several distributed MAC protocols have been proposed. Most of them can be classified in two categories:

- **Competitive Protocols**: These protocols subscribe to the philosophy that each node should compete for access to the common wireless channel by itself. Each node makes a local decision on whether to transmit its packets at a given time instant or not. These decisions are based on rules that are expected to maximize the chances of a successful transmission. One

[1] Some infrastructure networks such as wireless LANs also may use distributed MAC protocols to avoid collision.

common technique is to sense the channel for idleness before starting a new transmission, which is referred to as Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA). The main advantage of these protocols is simplicity and, hence, scalability. The main disadvantage is inefficient use of the wireless channel, especially when the number of nodes competing for the channel is high. Examples of such protocols include Multiple Access Collision Avoidance for Wireless LANs (MACAW), Floor Acquisition Multiple Access Protocol (FAMA), Busy Tone Multiple Access (BTMA), Dual BTMA (DBTMA), Receiver Initiated BTMA(RI-BTMA), Multiple Access Collision Avoidance by Invitation (MACA-BI), and Media Access with Reduced Handshake (MARCH).

- **Cooperative Protocols**: These protocols follow a different approach; they are based on the philosophy that nodes should cooperate on deciding a schedule, for example, who has the right to use the channel at a particular time. In most of these protocols, time is divided in slots and nodes work together on deciding which slots are assigned to which nodes. The main advantage of these protocols is efficient use when most nodes have continuous data to send. Another advantage is the guarantee of an upper bound on the delay that any node will experience in sending its packets. The major disadvantage is its complexity, which arises from its core philosophy of requiring cooperation among nodes. Examples of such protocols include Distributed Packet Reservation Multiple Access (D-PRMA), Collision Avoidance Time Allocation (CATA), Hop Reservation Multiple Access (HRMA), Soft Reservation Multiple Access with Priority Assignment (SRMA/PA), and Five Phase Reservation Protocol (FPRP).

Some protocols use a combination of the two philosophies, for example, use reservation for real-time traffic that needs a delay guarantee and use competitive access for regular traffic. An example of such a protocol is MACA with Piggy-Backed Reservation (MACA/PR).

We refer the reader to chapter 6 in Ref. 10 for a description of all the MAC protocols listed above. Some issues that are unique to a MAC protocol in WSN are described in the section “Energy Efficiency” below.

Neighbor Discovery and Multi-Hop Routing

Because nodes in a wireless ad hoc network depend on their neighbors to relay packets for them, each node needs first to discover its neighbors after initial deployment (“neighbor discovery”) and needs to update this information as neighboring nodes fail or move out of its transmission range. Moreover, each node needs to figure out to which neighbor a particular packet should be forwarded so that the packet can reach its destination most efficiently. This task is accomplished using a distributed ad hoc routing protocol, which typically takes into consideration the unique characteristics of wireless ad hoc networks, for example, frequent topology changes, limited power source, and low bandwidth resources.

Numerous ad hoc routing protocols have been proposed to date (see Ref. 11 and chapter 7 in Ref. 10) [2]. These protocols perform either flat or hierarchical routing. In flat routing, a node potentially can obtain a route to all the other nodes in the network. In hierarchical routing, the network usually is divided into many non overlapping clusters. Each cluster has a clusterhead that handles inter cluster routing, whereas other nodes only need to discover routes to nodes within their own cluster. Hierarchical routing protocols usually are more suitable for large networks. Below, we focus our discussion on flat routing protocols. Readers are referred to Ref. 11 for more discussion of hierarchical routing protocols. Issues that are unique to a routing protocol in WSN are described in the “Typical Traffic Pattern” section below.

Flat routing protocols can be classified into one of the following three categories, based on when the routes are discovered:

- A *proactive* routing protocol always maintains a route to every destination in a network, regardless of whether such a route will be used. The routes usually are computed using a distance vector algorithm or a link state algorithm. The protocols in this category are closest to traditional routing protocols, but they typically include optimizations that reduce bandwidth and processing overhead. They also are able to detect obsolete routes faster, for example, by adding more information in routing messages. Examples of proactive ad hoc routing protocols include DSDV [Destination-Sequenced Distance Vector (12)], OLSR [Optimized Link State Routing Protocol (13)], TBRPF [Topology Dissemination-based on Reverse-Path Forwarding (14)], and WRP [Wireless Routing Protocol (15)].
- A *reactive* routing protocol performs route discovery only when a node receives a packet to a particular destination that has no associated route in the routing table of the node. In other words, routing overhead will not be incurred for destinations that have no traffic destined to them. Therefore, reactive protocols usually have a lower processing, storage, and bandwidth overhead than proactive protocols. The reactive approach is especially suitable for networks with highly dynamic nodes, as the costs of maintaining routes to the dynamic destinations are extremely high. However, the overhead reduction also depends heavily on the traffic pattern in the network. If traffic is evenly distributed among all the destinations, the overhead saving may not be significant. Moreover, because route discovery takes time to complete, networks using reactive routing protocols may have a longer delay in packet delivery. Examples of reactive ad hoc routing protocols include AODV [Ad-hoc On-demand Distance Vector (16)], DSR [Dynamic Source Routing (17)], and DYMO [DYnamic Manet On-demand Routing (18)].

[2] Several of these protocols are being standardized by the Internet Engineering Task Force (IETF) MANET working group (http://www.ietf.org/html.charters/manet-charter.html).

- A *hybrid* routing protocol maintains pre computed routes to some destinations and performs on-demand route discovery for the other destinations. This type of protocol is designed for large networks, where a pure proactive protocol may incur too much control traffic and a pure reactive approach may have too high a packet delay and/or too much control traffic. One example of hybrid routing protocols is ZRP [Zone Routing Protocol (19)].

We now briefly describe DSR and ZRP as an illustration of how ad hoc routing protocols work.

Dynamic Source Routing. Each node in Dynamic source routing (DSR) (17) maintains a cache of discovered routes. When a sender needs to communicate with a new destination, it broadcasts a Route Request (RREQ) message to its neighbors. Each neighbor checks its cache to see if a route to the destination has been discovered before. If not, the node appends its address to the RREQ message and broadcasts this message to its neighbors. This process continues until at least one node identifies a route to the destination in its cache. This node then sends a Route Reply (RREP) message to the original sender of the RREQ message with the entire path in the reply. If no intermediate nodes have a path to the destination, the destination eventually will receive the RREQ message and send a RREP to the sender.

DSR uses source routing in packet delivery, for example, the sender of a packet specifies the entire path in the header of each data packet. Source routing allows a node to use multiple paths to reach the same destination while avoiding packet loops. However, it incurs more message overhead as each packet needs to carry the entire path in its header. Another downside is that the source route may become obsolete when a packet is still–route to its destination, especially when the nodes are highly mobile.

Zone Routing Protocol. In Zone Routing Protocol (ZRP) (11), each node maintains routes proactively to all the nodes within a certain number of hops. This set of nodes is called a *zone* for the node, and the number of hops is called a *zone radius*. If a node needs to deliver a packet to a destination outside its zone, it just sends a route request message to the nodes on the boundary of its zone. Those nodes, in turn, forward the message to the nodes on their zone boundary until a node can locate the destination in its own zone. Because the zone radius determines the routing traffic both within a zone and between zones, the main research issue is how to determine the appropriate zone radius to minimize the overall routing traffic.

Reliable Data Delivery

The wireless medium typically has a higher error and loss rate than the wired medium because of path loss, multi path fading, and interference. Path loss means that the signal strength weakens after the wireless signal travels for some distance. The remaining signal strength usually is a function of the distance. Multi path fading occurs when the wireless signal propagates in different directions and finally all the signals arrive at the same destination. These different versions of the original signal may have different phases and strength, so the combination of them may look very different from the original signal. Interference is caused by signals transmitted at frequencies close to each other. It can be reduced to a certain extent by using guard bands between frequency bands and minimizing the transmission range of each node (as described in “The Issue of Connectivity” section).

Given the higher error and loss rate of the wireless medium, how to ensure the reliable data delivery without negatively impacting end-to-end throughput becomes a key issue in wireless ad hoc networks. First, unlike wired networks that can rely solely on end-to-end recovery, wireless ad hoc networks also need hop-by-hop link-level error recovery to minimize delay, improve throughput, and reduce unnecessary retransmissions by end nodes. Second, the transport layer needs to distinguish losses caused by errors from those caused by congestion. The most popular reliable transport layer protocol is Transmission Control Protocol (TCP). It was designed for wired networks in which most of the losses are caused by congestion, so a TCP sender reduces its speed drastically whenever a loss is detected. This reaction is considered inappropriate for error-triggered losses as the sender should probably be as aggressive as before. As a result, the TCP performance in a wireless network could be problematic. Several extensions to TCP and alternative protocols have been proposed to address these problems. We refer the reader to Ref. 10 chapter 9 for details. A new trend in this research area is for the lower layer to expose more information to the transport layer so that the overall system will be more efficient and effective. Such *cross layer optimization* has been proposed for solving other problems in wireless ad hoc networks as well.

Security

Securing wireless ad hoc networks is especially challenging (20,21). First, privacy and integrity are more difficult to ensure in a wireless network than in a wired network because it is easy for an attacker to snoop on a wireless channel and modify ongoing transmission. Second, because of the infrastructureless nature of wireless ad hoc networks, authenticity is difficult to establish; no trusted central authority exists. Third, because the wireless nodes are more portable than computers in a traditional network, they may be easier to lose and be used later by attackers to inject false information. Furthermore, conventional security mechanisms usually have high computational and storage demands that may make their implementation difficult on wireless nodes.

WHAT SETS MANETS AND WSNS APART?

Although discussions of wireless ad hoc networks (which mostly refers to MANETs) often include wireless sensor networks (WSN) as a special case, these two areas each have blossomed into exciting research areas in their own right. The reason for this is because these two networks possess several unique characteristics that set them apart. Below we discuss some major characteristics that are unique to each of these networks.

Typical Usage

MANETs are used mostly for communication between human-operated devices, such as laptops, PDAs, or cellular phones, whereas wireless sensor networks are deployed mostly for data collection and event monitoring. We now discuss some representative applications of each network.

We first describe two applications of MANETs.

- **Facilitating Communication Among a Troop of Soldiers**: Each soldier carries a computing device with ad hoc networking ability. The devices hosted on the soldiers form an ad hoc network as soon as they are turned on. This network allows messages from any node to reach any other node even though the soldiers are allowed to move freely to achieve their operational goals (their movements are not constrained to maintain a connected network). Therefore, the network of devices needs to take care of maintaining connectivity.
- **Facilitating Communication in Remote Locations**: Cellular phone towers do not cover remote areas (such as mountains and forests). If mobile phones are equipped with ad hoc networking capability (as is being planned), then an ad hoc network among the various mobile phones can be formed. This ad hoc network will enable data and possibly voice communication among users even if no cellular phone towers are in the neighborhood to provide regular coverage.

Now we describe two applications of WSNs.

- **Detecting Illegal Crossing on an International Border**: Wireless sensor nodes are sprayed from an aircraft on the international border. Once these sensors land on ground, they form a multi-hop wireless network. They start monitoring for people or vehicles crossing the border. As soon as such an event is detected by one or more sensors, a detection message is dispatched to a manned station for possible action. The message takes less than a couple of seconds to reach a manned station that may be situated several miles from the point of occurrence of the intrusion event. This system has the potential to improve significantly the border surveillance at a low cost. With this system, the entire border can be monitored continuously instead of the spotty surveillance that is done today.
- **Monitoring a Fabrication Plant to Prevent Downtime**: Wireless sensors can be deployed in a fabrication plant to monitor the vibration and acoustic signatures of critical equipments. If the signature matches some specific patterns that typically precede failures, a message is immediately dispatched to a manned station and preventive actions are taken to ensure no downtime occurs. This system has the potential to save millions of dollars by preventing downtime of critical equipment.

As illustrated by the above-mentioned applications, the purpose of deploying a MANET is very distinct from that of deploying a WSN. The implication is that new research issues emerge in a WSN that had not been so critical in a MANET, such as the issues of coverage (i.e., ensuring that a WSN provides the desired quality of monitoring), tolerance to new types of faults, focus on energy efficiency, and so forth. Even those issues that are common to both networks, such as the design of medium access control, routing, and other protocols (discussed in the “Common Issues in wireless Ad Hoc Networks” section), need to be revisited for WSNs. Below, we elaborate on these and other differences between MANETs and WSNs.

Typical Traffic Pattern

Because the typical uses of the two networks are distinct, their typical traffic patterns are quite distinct as well. In a MANET, traffic pattern usually is point to point or point to multipoint. In other words, traffic originating from one node may be destined to one particular subset of nodes at a given instant, whereas traffic originating from another node or from the same node but at a different instant may be destined to a different subset of nodes.

In a wireless sensor network, however, data traffic either flows from sensor nodes to one or a set of base stations, called source to sink, or from the base station(s) to some or all nodes, called sink to source. Examples of source-to-sink traffic are event detection messages from sensors or sensor data about the environmental variations. Examples of sink-to-source traffic are the dissemination of a new program to all (or a subset of) sensors or the dissemination of a new value of some parameters to all (or a subset of) sensors. Base stations sometimes are referred to as *sinks* to emphasize this traffic pattern.

Because the traffic pattern in a WSN is so distinct from that in a MANET, the routing protocol used in these two networks is different as well. As mentioned in the previous paragraph, two types of traffic need to be supported by a WSN, information from sensors to sink(s) and from sink(s) to sensors. Traffic from sensors to sink(s) is referred to as *data gathering*, and that from sink(s) to sensors is referred to as *data dissemination*. The major issues that need to be addressed in a routing protocol to support each of these traffic patterns are very distinct, and hence two different categories of routing protocols have been developed to cater to these two traffic types. MintRoute (22) is an example of a data gathering routing protocol, and Deluge (23) is an example of a data dissemination routing protocol.

Attended Versus Unattended—Implications for Fault-Tolerance

MANETs typically consist of human-operated devices and therefore are attended mostly by a human being. Several types of faults easily may be detected and repaired (by resetting the device). Battery exhaustion also is not a major concern as the human operator may recharge the device when needed.

A wireless sensor network typically is deployed outdoors and may remain unattended for long periods of time. This unattended nature has several fault-tolerance implications. First, sensor nodes are subject to new types of faults that may come from outdoor environmental conditions such as wind, rain, excessive heat or cold, physical tampering, and so forth. Excessive heat or cold or excessive battery depletion may cause other types of failures that qualify as byzantine failures (24). Second, node failures are more frequent in a wireless sensor network. Further, node failures may not be detected immediately and sometimes and not be detected at all (for example, if message from a healthy sensor node cannot reach the base station). Third, physically repairing or replacing individual nodes may not be feasible (e.g., if the sensors are deployed in inhospitable terrain or in enemy territory), and, hence, only remote repair of failures is feasible. Fourth, battery recharging may not be feasible (especially if the nodes are not equipped with energy scavenging mechanisms as in solar cells).

Consequently, the protocols developed for wireless sensor network needs to be adaptive to these new types of failures. These failure types are not prevalent in a MANET.

Resource Constraints

The computational capacity, memory size, buffer capacity, and network bandwidth available to a sensor node is an order of magnitude lower than that available to a node in a typical MANET. See Table 1 for a comparison of the hardware specification of a typical WSN device with that of a typical MANET device. Observe that the processor is at least 50 times slower in a WSN and that RAM size is at least 6,400 times lower. This implies that the protocols and algorithms developed for a WSN need to be considerably simpler than that developed for a typical MANET.

Energy Efficiency

Sensor nodes, being deployed outdoors and unattended, run on batteries that may not be replaced. Hence, the issue of energy efficiency and network longevity are high-priority considerations, whereas this problem is less severe in MANETs that mostly consist of personal digital devices that can be recharged. As a result, every protocol or algorithm developed for wireless sensor network should be designed with a consideration of energy efficiency. For example, the MAC protocols proposed for MANETs are not very appropriate for use in a WSN because energy efficiency is not as critical in a MANET. In a WSN, even keeping the radio in listening mode for an extended period of time can drain significant energy. Hence, the radio may be completely turned off to save energy and turned on only periodically or when needed to receive or transmit data. If the radio is not always in the listening mode, communication (especially of real-time data like the detection of an intruder) becomes nontrivial. Several MAC protocols to ensure timely communication while ensuring energy-efficiency have been proposed. An example of such a protocol is B-MAC (25).

Table 1. Comparison of the key hardware properties of a typical WSN device (telosb mote), a pocket PC (HP iPAQ), and a typical laptop

Property	WSN Device	Pocket PC	Laptop
Processor speed	8 MHZ	400 MHZ	1.8 GHZ
RAM size	10 KB	64 MB	1 GB
Persistent storage	1 MB	64 MB	60 GB
Radio data rate	250 kbps	11 mbps	54 mbps

The issue of energy efficiency also is critical in the process of deployment. If redundant sensors are deployed, then the redundant sensor nodes are put to sleep, taking turns, to maximize the lifetime of the sensors (as discussed in “The Issue of Coverage” section).

Mobility

The nodes in a typical MANET are assumed to be frequently mobile. The nodes in a WSN, however, are mostly static, unless moved by wind or other external phenomenon. In the future, some sensor networks may consist of mobile nodes, (26). In these cases, however, the motion of sensors will be dictated by the network requirement [such as to facilitate data collection from a sensor node disconnected from the base station (9) or to provide temporary coverage in place of a failed sensor node (26)] as opposed to a user-induced motion as in a typical ad hoc network. This difference in the mobility pattern affects how the protocols for the two networks are designed.

Security Threats

New types of security threats are possible in a sensor network because of outdoor and unattended deployment, such as physical capture and physical destruction. Because sensor nodes have the ability to receive new program code to replace the currently active program code via a wireless channel, an adversary may inject malicious program onto sensor nodes. False sensory data or bogus events also can be injected in the network. Communication can be jammed by accompanying a malicious target (that the network is supposed to detect) with a jammer device. Because the sensors have limited energy reserve, attacks can be played to deplete sensors of their energy, such as by sending too many messages (from a more powerful device) or by causing too many event detections. Designing protocols to mitigate these and other security threats in a WSN currently is an active area of research.

The Issue of Coverage

Because the main purpose of a WSN is data collection and event monitoring, the issue of coverage becomes a key issue in the deployment and maintenance of sensor networks. The issue of coverage is that of determining methods of initial deployment and subsequent maintenance of the network topology (over time) to ensure that a WSN provides the desired quality of monitoring (27,28). This issue does not arise in MANETs because their main purpose is not to monitor events.

When a sensor network is to be deployed, several critical deployment issues arise, such as how many sensors should be deployed and in what pattern. Determining how many sensors to deploy becomes more challenging when sensors cannot be deployed at desired locations, as when spraying them from an aircraft. Once sensors have been deployed, mechanisms are needed to detect whether the network continues to provide the desired quality of monitoring, as some sensors may fail unexpectedly because of environmental factors. In the event that the network can no longer provide the desired quality of monitoring, additional sensors may need to be deployed, or if the sensors have movement ability, then some sensors may need to be repositioned to repair the network. Designing efficient methods of redeployment or reconfiguration continues to be an active area of research.

To tolerate unanticipated sensor failures, some redundant sensors may be deployed. In such a case, mechanisms are needed to determine a sleeping schedule (29) for the redundant sensors such that the batteries of the active nodes get depleted at a slower rate, ensur which a longer life for the network.

Localization

Because the main purpose of a wireless sensor network is to monitor events or collect information about the environment, it often is critical to associate location information with the data collected by a sensor node. For example, if a sensor network is deployed to detect fire, then it is not sufficient to learn that fire has erupted. Location of the fire eruption is a critical part of the information. Additionally, because installing GPS at every sensor node is prohibitively expensive and energy consuming, the process of localization needs to be performed in a sensor network such that each sensor node knows its absolute location. Either the process of localization is not so critical in a typical MANET, or installing a GPS unit on each device is within the budget.

Various mechanisms have been proposed to perform localization. For example, a mobile unit with GPS mounted on it can traverse through the network broadcasting its location (30). Sensors can localize themselves using this broadcast. Alternatively, some anchor nodes who know their location (possibly using a GPS) can be placed in the network. These nodes then help other nodes determine their locations by using a localization algorithm. Some mechanisms for localization use the time difference of arrivals of radio or acoustic signals (31), whereas others use radio interferometric techniques where radio signals are transmitted to cause interference (and hence phase difference) at the receivers (32). Localization in a WSN still is an active area of research.

Time Synchronization

Because the main purpose of a WSN is to monitor events or collect information about the environment, often it is critical to associate time information with the data collected by a sensor node. For example, if a sensor network is deployed to track the trajectory of a moving target, then the time of detection of the target at a specific sensor is necessary to chart the trajectory of the target movement.

Time synchronization also is useful in MANETs (especially for implementing some cooperative MAC protocols). However, the clocks of MANET nodes usually are more accurate than that of sensor networks. Also, in MANET devices such as cell phones, time synchronization is provided by a centralized infrastructure. Consequently, the problem of time synchronization is more critical in a WSN than in a MANET and requires a nontrivial solution.

Several protocols exist for time synchronization in a WSN. They can be classified in two categories: *proactive* and *reactive*(33). In proactive protocols, a virtual global reference time across the entire network is established and maintained via the exchange of messages. Reference Broadcast Synchronization (RBS) (34) and Flooding Time Synchronization Protocol (FTSP) (35) are examples of proactive protocols. In reactive protocols, time is not synchronized at all. Packets are time-stamped using local unsynchronized times. Synchronization is done after the detection of events. An example of such a protocol is Routing Integrated Time Synchronization (RITS) protocol (36,33).

CONCLUSION

Wireless ad hoc networks have revolutionized the world of communication by enabling quick and infrastructureless communication at the point of need, whether it is in a battlefield, on a mountain, under water, or on a different planet. Wireless sensor networks, alternately, are revolutionizing many disciplines by providing the unprecedented ability to observe our environment. By enabling the unobtrusive collection and accessibility of real-time data from the environment, new research capability now is available in several scientific disciplines, such as biology, geology, oceanology, medicine, and elderly care. Also, by enabling real-time and continuous monitoring of the environment, new capabilities in surveillance have become possible such as efficient and comprehensive border surveillance. Both of these disciplines, wireless ad hoc network and wireless sensor network, are relatively young disciplines with highly active research communities. As new applications emerge and as the technologies mature, these two technologies potentially can have a greater impact on our lives than personal computers and the Internet have.

BIBLIOGRAPHY

1. H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*, John Wiley & Sons, 2005.
2. M. Horton, D. E. Culler, K. Pister, J. Hill, R. Szewczyk and A. Woo, The commercialization of microsensor motes, *Sensors Magazine*, **19**(4): 40–48, 2002.
3. I. F. Akyildiz, X. Wang and W. Wang, Wireless mesh networks: A survey, *Computer Networks*, **47**(4): 445–487, 2005.
4. K. Fall, A delay-tolerant network architecture for challenged internets, *Proc. ACM SIGCOMM*, Karlsruhe, Germany, 2003.
5. P. Gupta and P. R. Kumar, Critical power for asymptotic connectivity in wireless networks, *IEEE 37th Conference on Decision and Control*, Tampa, FL: 1998, pp. 1106–1110.
6. X. Y. Li, P. J. Wan, Y. Wang and C. Yi, Fault tolerant deployment and topology control in wireless networks, *International Symposium on Mobile Ad Hoc Networking and Computing (ACM MobiHoc)*, Annapolis, MD: 2003, pp. 117–28.
7. P. Santi, The critical transmitting range for connectivity in mobile ad hoc networks, *IEEE Trans. in Mobile Computing*, **4**(3): 310–317, 2005.

8. P. Balister, B. Bollobás, A. Sarkar and S. Kumar, Reliable density estimates for achieving coverage and connectivity in thin strips of finite length, *International Conference on Mobile Computing and Networking (ACM MobiCom)*, Montreal, Canada, 2007.
9. S. Jain, R. C. Shah, W. Brunette, G. Borriello and S. Roy, Exploiting mobility for energy efficient data collection in wireless sensor networks, *J. Mobile Networks and Applications*, **11**(3): 327–339, 2006.
10. C. S. R. Murthy and B. S. Manoj, *Ad Hoc Wireless Networks: Architectures and Protocols*. Prentice Hall, 2004.
11. E. M. Belding-Royer, Routing approaches in mobile ad hoc networks, chapter 10, in *Mobile Ad Hoc Networking*. Wiley-IEEE Press, 2004.
12. C. Perkins and P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, 1994, pp. 234–244.
13. P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum and L. Viennot, Optimized link state routing protocol for ad hoc networks, *Proc. 5th IEEE Multi Topic Conference (INMIC 2001)*, 2001.
14. R. Ogier, F. Templin and M. Lewis, Topology dissemination based on reverse path forwarding (TBRPF), Feb. 2004.
15. S. Murthy and J. J. Garcia-Luna-Aceves, An efficient routing protocol for wireless networks, *Mobile Networks and Applications*, **1**(2): 183–197, 1996.
16. C. Perkins, E. Belding-Royer and S. Das, Ad hoc on-demand distance vector (AODV) routing, July 2003.
17. D. B. Johnson and D. A. Maltz, Dynamic source routing in ad hoc wireless networks, *Mobile Computing*, 353, 1996.
18. I. Chakeres and C. Perkins, Dynamic manet on-demand routing, Mar. 2006.
19. Z. J. Haas, A new routing protocol for the reconfigurable wireless networks, *Proc. of 6th IEEE International Conference on Universal Personal Communications (IEEE ICUPC'97)*, 1997, Vol. 2, pp. 526–566.
20. Y.-C. Hu and A. Perrig, A survey of secure wireless ad hoc routing, *IEEE Security & Privacy, special issue on Making Wireless Work*, **2**(3): 28–39, 2004.
21. A. Mishra and K. M. Nadkarni, Security in wireless ad hoc networks, pp. 499–549, 2003.
22. A. Woo, T. Tong and D. Culler, Taming the underlying challenges of reliable multihop routing in sensor networks, *ACM Conference on Ebmedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, 2003.
23. J. W. Hui and D. Culler, The dynamic behavior of a data dissemination protocol for network programming at scale, *ACM Conference on Ebmedded Networked Sensor Systems (Sensys)*, 2004.
24. S. Bapat, V. Kulathumani and A. Arora, Analyzing the yield of exscal, a large scale wireless sensor network experiment, *IEEE International Conference on Network Protocols (ICNP)*, Boston, MA, 2005.
25. J. Polastre, J. Hill and D. Culler, Versatile low power media access for wireless sensor networks, *ACM Sensys*, 2004.
26. J.-P. Sheu, P.-W. Cheng and K.-Y. Hsieh, Design and implementation of a smart mobile robot, *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Montreal, Canada, 2005, Vol. 3, pp. 422–429.
27. S. Kumar, T. H. Lai and J. Balogh, On *k*-coverage in a mostly sleeping sensor network, *International Conference on Mobile Computing and Networking (ACM MobiCom)*, Philadelphia, PA, 2004, pp. 144–158.
28. S. Kumar, T. H. Lai and A. Arora, Barrier coverage with wireless sensors, *International Conference on Mobile Computing and Networking (ACM MobiCom)*, Cologne, Germany, 2005, pp. 284–298.
29. S. Kumar, T. H. Lai, M. E. Posner and P. Sinha, Optimal sleep wakeup algorithms for barriers of wireless sensors, *IEEE BROADNETS*, Durham, NC, 2007.
30. A. Galstyan, B. Krishnamachari, K. Lerman and S. Pattem, Distributed online localization in sensor networks using a moving target, *Third International Conference on Information Processing in Sensor Networks (IPSN)*, Berkeley, CA, 2004.
31. L. Girod, M. Lukac, V. Trifa and D. Estrin, The design and implementation of a self-calibrating distributed acoustic sensing platform, *The Fifth ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Boulder, CO, 2006.
32. B. Kusy, A. Ledeczi and X. Koutsoukos, Tracking mobile nodes using rf doppler shifts, *The Fifth ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Sydney, Australia, 2007.
33. J. Sallai, B. Kusy, A. Ledeczi and P. Dutta, On the scalability of routing integrated time synchronization protocol, *European Workshop on Wireless Sensor Networks (EWSN)*, Zurich, Switzerland, 2006.
34. J. Elson, L. Girod and D. Estrin, Fine-grained network time synchronization using reference broadcasts, *Proc. Fifth Symposium on Operating System Designand Implementation (OSDI)*, Boston, MA, 2002, pp. 147–163.
35. M. Maroti, B. Kusi, G. Simon and A. Ledeczi, The flooding time synchronization protocol, *ACM Sensys*, Baltimore, MD, 2004.
36. B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi and D. Culler, Elapsed time on arrival: A simple and versatile primitive for canonical time synchronization services, *Int. J. Ad Hoc and Ubiquitous Computing*, **1**(4): 239–251, 2006.

Santosh Kumar
Lan Wang
The University of Memphis
Memphis, Tennessee

COMMUNICATION-INDUCED CHECKPOINTING PROTOCOLS AND ROLLBACK-DEPENDENCY TRACKABILITY: A SURVEY

INTRODUCTION

A *checkpoint* is a snapshot of the current state of a process, saved on nonvolatile storage. A process periodically takes a checkpoint so that it can reduce the amount of lost work upon a failure. To survive a failure, the process reloads the state recorded in the latest checkpoint into volatile memory and restarts from that checkpoint. Such a procedure is called *rollback recovery*.

A distributed computation is composed of multiple processes connected by a communication network. Processes communicate and synchronize only by exchanging messages via the network. The execution of each process produces a sequence of events, and all the events produced by a distributed computation can be modeled as a partially ordered set with the well-known Lamport's *happened-before* relation (1). In a distributed computation, the states of all involved processes and those of the underlying communication channels constitute the system state. Upon a failure, lost process states may create *orphan* messages that result in an *inconsistent* state, i.e., a state that is impossible to reach via any failure-free distributed execution. A message is called *orphan* with regard to an ordered pair of checkpoints if the receiving event of such a message happens before the latter checkpoint in the pair but its sending event occurs after the former one. Hence, an ordered pair of checkpoints is *consistent* if there are no orphan messages with respect to this pair. Furthermore, a *global checkpoint* is a set of checkpoints, one from each process. A global checkpoint is consistent if all pairs of its component checkpoints are consistent (2). In particular, the consistent global checkpoint that can minimize the total rollback distance upon a failure is called the *recovery line*. Computing a consistent global checkpoint, preferably the recovery line, is fundamental to any rollback-recovery protocol after inconsistencies happen due to failures.

If checkpoints are taken independently, it is possible that cascading rollback propagation, which is required to eliminate all orphan messages, may occur during the course of finding the recovery line. In the worst case, no consistent global checkpoints can be found (except for the set of all initial checkpoints), and this is the well-known *domino effect* problem (3). Many checkpointing protocols have been proposed to selectively take checkpoints to avoid this problem. For more details, see the survey paper (4). Among them, *coordinated checkpointing*(2,5) avoids the domino effect by synchronizing the checkpointing actions of all processes through explicit control messages. In contrast, *communication-induced checkpointing (CIC)*(6) accomplishes coordination by piggybacking control information on application messages. Specifically, in addition to taking application-specific *basic* checkpoints, each process can also be directed by the protocol to take extra *forced* checkpoints according to certain checkpoint-inducing conditions, to ensure the progression of the recovery line. Such conditions typically predicate on information piggybacked on messages as well as on local control variables.

CIC protocols can also be used to achieve a stronger property, called *rollback-dependency trackability* (RDT) (7). Besides the inconsistencies resulting from causal dependency, two checkpoints can have a noncausal, *zigzag* dependency that makes it impossible for them to belong to the same consistent global checkpoint (8). A CIC protocol satisfies RDT if all such hidden dependencies are guaranteed to be online trackable through a simple transitive dependency vector. The RDT property can both eliminate the domino effect and ensure that any set of checkpoints that are not causally related pairwise can be extended to form a consistent global checkpoint. Moreover, it allows efficient decentralized calculation of the recovery line (7).

PRELIMINARIES

In this section, we define terms that are essential for understanding the CIC protocols. Associated with a distributed computation, the set of messages and the set of local checkpoints constitute the *checkpoint and communication pattern*. In a pattern, $C_{i,x}$ represents the xth checkpoint of process P_i. The sequence of events occurring at P_i between $C_{i,x-1}$ and $C_{i,x}$ $(x > 0)$ constitute a *checkpoint interval* (or *interval* for short), which is denoted by $I_{i,x}$.

A *Z-path* is defined as a sequence of messages in which the sending event of every message except for the first one happens in the same or a later interval than the receiving event of the preceding message (9). Furthermore, a Z-path is from checkpoint $C_{i,x}$ to $C_{j,y}$ if its first message is sent after $C_{i,x}$ and its last message is received before $C_{j,y}$. A Z-path denotes that its terminating checkpoint has a rollback dependency on its starting one. More specifically, with a Z-path from $C_{i,x}$ to $C_{j,y}$, if process P_i rolls back to $C_{i,x}$ upon a failure, process P_j also needs to rollback to $C_{j,y-1}$ in order to eliminate inconsistency. Hence, a Z-path from a checkpoint $C_{i,x}$ to itself, which is also called *Z-cycle*, is the cause of the domino effect since it will induce recursive, cascading rollback propagation. The checkpoint $C_{i,x}$ involved in this Z-cycle is thus unable to belong to any consistent global checkpoint (8) and considered *useless* in Ref. 10. Intuitively, the ultimate goal of a CIC protocol is to eliminate all Z-cycles from a checkpoint and communication pattern. A protocol can take an additional forced checkpoint prior to a condition representing a Z-cycle to remove this Z-cycle. Such a forced checkpoint is also consistent with the involved useless checkpoint. One major challenge is that not all Z-cycles are on-the-fly detectable. Consequently, CIC protocols employ various techniques to discover suspect conditions that may result in a Z-cycle.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

A Z-path is *causal* if the receiving event of each message aside from the last one precedes the sending event of the next message in the sequence. A causal Z-path is sometimes referred to as a *causal path*. A Z-path is *noncausal* if it is not causal. A causal path means that the information in its starting checkpoint can be online transmitted to its terminating one through the piggybacking technique. In addition, a noncausal Z-path sent in interval $I_{i,x}$ and arriving in $I_{j,y}$ is *causally doubled* if a causal path exists from $I_{i,x'}$ to $I_{j,y'}$ such that $x \leq x'$ and $y' \leq y(9)$. The idea of *causal doubling* is that for this noncausal Z-path, the information in its starting checkpoint can still be online forwarded to its terminating one via the doubling causal path. Hence, if all Z-paths in a checkpoint and communication pattern are either causal or causally doubled, all rollback dependencies between checkpoints can be on-the-fly trackable with a transitive dependency vector and the pattern, by definition, satisfies RDT. Moreover, because a Z-cycle can never be causally doubled, RDT protocols suppress the formation of Z-cycles altogether (11).

The most extreme method to prevent the domino effect by enforcing RDT is to direct a process to take a forced checkpoint whenever a message is received. A better way to this end is to force a checkpoint before every message-receiving event with a preceding message-sending event in the same interval (12). In doing so, the two protocols can ensure that all message-receiving events precede all message-sending events within every interval to prevent noncausal Z-paths from being formed. In the next two sections, we will introduce several more sophisticated CIC protocols and RDT protocols, respectively.

CIC PROTOCOLS

CIC protocols can be divided into two distinct categories: *index-based* and *model-based*(4). An index-based protocol associates every checkpoint with a sequence number similar to the Lamport's logical clock (1). In contrast, a model-based protocol does not use a time-stamping mechanism; rather, it prevents the formation of certain checkpoint and communication patterns during the execution.

Index-based CIC protocols have been extensively studied in the literature (10,13–17). A common technique among them is to guarantee that sequence numbers of checkpoints always *increase* along a Z-path (14,18). Doing this technique can eliminate all Z-cycles since the sequence number of a checkpoint cannot be larger than itself. Furthermore, checkpoints with the same sequence number from different processes are consistent because a Z-path will never be formed from one checkpoint to another with the same sequence number. Such a consistent global checkpoint can thus be used for recovery upon a failure. For example, the checkpoint-inducing condition of the protocol, introduced in Ref. 13, is expressed as "$m.sn > sn_i$", where sn_i represents the current sequence number of a process P_i and $m.sn$ the sequence number carried on a message m received by P_i. The intuition behind this condition is that when a process P_i receives a message m with $m.sn > sni$, it will take a forced checkpoint with the sequence number set to $m.sn$ prior to delivering m so that P_i can contribute a checkpoint to the construction of the new consistent global checkpoint with sequence number $m.sn$.

Because forcing extra checkpoints incurs runtime overhead, it is desirable to take as few forced checkpoints as possible while avoiding the domino effect. To this end, one fundamental principle of improved CIC protocols is to reuse existing checkpoints as much as possible as part of a coming consistent global checkpoint with a higher sequence number to avoid the requirement of some forced checkpoints. For instance, a protocol proposed in Ref. 10 will direct a process to force a checkpoint only when the "$m.sn > sn_i$" condition is encountered after at least one message-sending event in the same interval. If there is not any message-sending event between the last checkpoint of process P_i and the receiving event of message m, it is impossible to have a Z-path from the last checkpoint of P_i to a checkpoint of other processes prior to delivering m. Therefore, despite receiving one message with a larger sequence number, P_i can still employ its last checkpoint as part of the coming consistent global checkpoint corresponding to $m.sn$. In addition, by subtly collecting as much information as it can from the causal past, another protocol in Ref. 10 achieved an even more restrictive checkpoint-inducing condition, at the expense of piggybacking much more control variables on messages than just a single sequence number. In practice, much of the causal information is too obsolete to be helpful to checkpointing decisions. Accordingly, the protocol presented in Ref. 15 discarded some obsolete information from the causal past to reduce the size of piggybacked information to a small constant, while achieving nearly as good performance as the previous protocol, especially on a tree-shaped communication network (15).

Another way to reuse existing checkpoints is to adopt a different indexing strategy from the classic one. The sequence number of the underlying indexing strategy used in Refs. 10, and 13–15 is maintained in the classic way of Ref. 1 in that it is increased by one each time a basic checkpoint is taken. Hence, if a process takes basic checkpoints at a higher rate than other processes and consequently has a larger sequence number, forced checkpoints may be induced when other processes receive messages from this process. To deal with this asymmetry, the *lazy indexing* strategy is presented in Ref. 16. With such a strategy, if one process P_i has only received messages with sequence numbers smaller than its own in the current interval, it is unnecessary for P_i to increase the sequence number when the next basic checkpoint is taken. The reason is that, in such a situation, the new checkpoint of P_i can still be consistent with existing checkpoints of other processes that are originally consistent with its preceding one so that it does not need to belong to another consistent global checkpoint with a higher sequence number. Furthermore, a more sophisticated lazy indexing strategy is proposed in Ref. 17. This strategy precisely traces orphan messages to allow the consistent global checkpoint corresponding to the current sequence number to gradually progress to succeeding checkpoints as best it can. Such an improved strategy can increase the sequence number at a lower speed than the previous lazy indexing scheme.

Next, model-based CIC protocols, like those introduced in Refs. 19 and 20, are more complex protocols that track the the checkpoint and communication pattern to prevent particular patterns that can potentially result in a Z-cycle. In general, a model-based protocol needs more control information carried on a message than an index-based protocol. Moreover, simulation experiments showed that the former is more eager to remove a suspect Z-cycle than the latter, which results in many more forced checkpoints (21).

Finally, a common intuition about CIC protocols is if a protocol forces a checkpoint only at a stronger condition, then it must take at most as many forced checkpoints as a protocol based on a weaker condition. It has been proved that such an intuition is in fact false because any forced checkpoint may affect subsequent condition testings (22). This result implies that the usual approach of sharpening the checkpoint-inducing condition by piggybacking more information on each message may not always yield a more efficient protocol. But interestingly, comparisons of some existing protocols can indeed be based solely on comparing their conditions (22). The analysis also led to an impossibility result: An optimal online CIC protocol cannot exist that always takes fewer forced checkpoints than any other protocol (22).

RDT PROTOCOLS

Given a checkpoint and communication pattern, a CIC protocol does not need to examine that every noncausal Z-path is causally doubled to ensure the RDT property. Causally doubling a certain subset of noncausal Z-paths is sufficient. Such a subset is called an *RDT characterization* in Ref. 11. Important RDT characterizations are all derived from the notion of *prime* causal paths. A causal path from a checkpoint $C_{i,x}$ to a process P_j is *prime* if it arrives at P_j the first among all causal paths from $C_{i,x}$ to P_j. Intuitively, such a causal path is the first causal path causing P_j to have a dependency on $C_{i,x}$.

The first RDT characterization is the *PCM-path*, which is a noncausal Z-path formed by concatenating a prime causal path and a single message (11). A PCM-path is the first Z-path that cannot transmit to its arriving process the information about the rollback dependency on the starting checkpoint, if it is not causally doubled. Hence, a safe strategy to satisfy RDT is to *break* any non-causally doubled PCM-path with a forced checkpoint prior to meeting the involved prime causal path. Moreover, for an online protocol, the information of being causally doubled must be contained in the causal past of a process at the moment it detects a PCM-path for the checkpointing decision. This concept is called *visible doubling*(23). An online protocol can achieve RDT if it breaks all PCM-paths that are not visibly doubled. Several protocols based on the PCM-paths are derived in Ref. 11, where each protocol breaks a certain subset of PCM-paths, containing at least all non-visibly doubled PCM-paths. Among them, a protocol with a stronger condition generally needs more control information carried on a message. A comprehensive comparison of their performance can be found in Ref. 24.

A more constrained RDT characterization, called the *EPSCM-path*, is proposed in Ref. 11 as well. An EPSCM-path is a PCM-path such that the component prime path is both *elementary* and *simple*. A causal path is *elementary* if it merely traverses a process once, whereas a causal path is *simple* if it does not include any checkpoints. This characterization is the minimal subset of non causal Z-paths that have to be *causally doubled* to satisfy the RDT property. A few protocols based on EPSCM-paths are also presented in Ref. 11.

Recently, it was proved in Ref. 25 that *visibly doubling* all *PMM-paths* in a pattern suffices to satisfy RDT, where a PMM-path is a noncausal Z-path composed of just two messages with the first being prime. So it is the minimal noncausal Z-path allowed in the computation model. Several RDT protocols can be derived from PMM-paths as well. Interestingly, it has been demonstrated in Ref. 26 that for an RDT protocol, the last elementary and simple part of every prime causal path it encounters online is still prime and so is the last message. Thus, RDT protocols will always encounter a PCM-path, an EPSCM-path, and a PMM-path simultaneously. Moreover, several protocols derived from these three kinds of Z-paths, respectively, have the same behavior for all patterns (26).

The most important benefit of the RDT property is that it allows us to find the recovery line in an efficient, distributed manner because all checkpoint dependencies are online trackable. But an RDT protocol typically needs more forced checkpoints and requires more control information piggybacked on a message. Such protocols can be classified into the model-based category because it prohibits some particular patterns from occurring. Finally, an impossibility result was presented in Ref. 27, stating that it is not possible to design a *scalar, clock-based* CIC protocol that carries only one integer on a message, while satisfying RDT.

CONCLUSIONS

CIC protocols allow each process to take its basic checkpoints autonomously. No special coordination messages are exchanged to ensure consistency among all processes. Furthermore, the calculation of a consistent global checkpoint upon a failure can be accomplished in an efficient and decentralized manner. But every application message needs to carry extra control information. More importantly, the behavior of taking forced checkpoints highly depends on the number of processes and on the communication pattern. Also, the number of checkpoints induced by a protocol may be a considerable burden. Therefore, the main challenge for CIC protocols is to control the unpredictable checkpointing behavior and to reduce the number of forced checkpoints while preserving the desirable properties.

BIBLIOGRAPHY

1. L. Lamport, Time, clocks and the ordering of events in a distributed system, *Commun. ACM*, **21**(7): 558–565, 1978.
2. K. M. Chandy and L. Lamport, Distributed snapshots: Determining global states of distributed systems, *ACM Trans. Comput. Syst.*, **3**(1): 63–75, 1985.

3. B. Randell, System structure for software fault-tolerant, *IEEE Trans. Soft. Eng.*, **1**(2): 220–232, 1975.
4. E. N. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson, A survey of rollback-recovery protocols in message-passing systems, *ACM Comput. Surveys*, **34**(3): 375–408, 2002.
5. R. Koo and S. Toueg, Checkpointing and rollback-recovery for distributed systems, *IEEE Trans. Soft. Eng.*, **13**(1): 23–31, 1987.
6. B. Janssens and W. K. Fuchs, Experimental evaluation of multiprocessor cache-based error recovery, *Proc. Int'l Conf. Parallel Process.*, 1991, pp. 505–508.
7. Y. M. Wang, Consistent global checkpoints that contain a given set of local checkpoints, *IEEE Trans. Comp.*, **46**(4): 456–468, 1997.
8. R. H. B. Netzer and J. Xu, Necessary and sufficient conditions for consistent global snapshots, *IEEE Trans. Parallel and Distrib. Syst.*, **6**(2): 165–169, 1995.
9. R. Baldoni, J. M. Helary, A. Mostefaoui, and M. Raynal, A communication-induced checkpointing protocol that ensures rollback-dependency trackability, *Proc. Int'l Symp. Fault-Tolerant Comput.*, 1997, pp. 68–77.
10. A. Mostefaoui, J. M. Helary, R. H. B. Netzer, and M. Raynal, Communication-based prevention of useless checkpoints in distributed computations, *Distrib. Computing*, **13**(1): 29–43, 2000.
11. R. Baldoni, J. M. Helary, and M. Raynal, Rollback-dependency trackability: A minimal characterization and its protocol, *Inform. and Comput.*, **165**(2): 144–173, 2001.
12. D. L. Russell, State restoration in systems of communicating processes, *IEEE Trans. Soft. Eng.*, **6**(2): 183–194, 1980.
13. D. Briatico, A. Ciufoletti, and L. Simoncini, A distributed domino-effect free recovery algorithm, *Proc. IEEE Symp. Reliab. in Distrib. Soft. and Database Syst.*, 1984. pp. 207–215.
14. D. Manivannan and M. Singhal, A low overhead recovery technique using quasi-synchronous checkpointing, *Proc. IEEE Int'l Conf. on Distrib. Comput. Syst.*, 1996, pp. 100–107.
15. J. Tsai, An efficient index-based checkpointing protocol with constant-size control information on messages, *IEEE Trans. Dependable and Secure Comput.*, **2**(4): 287–296, 2005.
16. G. M. D. Vieira, I. C. Garcia, and L. E. Buzato, Systematic analysis of index-based checkpointing algorithms using simulation, *Proc. IX Brazilian Symp. Fault-Tolerant Comput.*, 2001, 31–41.
17. R. Baldoni, F. Quaglia, and P. Fornara, An index-based checkpointing algorithm for autonomous distributed systems, *IEEE Trans. Parallel and Distrib. Syst.*, **10**(2): 181–192, 1999.
18. J. M. Helary, A. Mostefaoui, and M. Raynal, Virtual precedence in asynchronous systems: Concept and applications, *Int'l Workshop Distrib. Algor.*, 1997, pp. 170–184.
19. I. C. Garcia and L. E. Buzato, Checkpointing using local knowledge about recovery lines, *Technical Report, TR-IC-99-22*, University of Campinas, Brazil, 1999.
20. F. Quaglia, R. Baldoni, and B. Ciciani, On the no-Z-cycle property in distributed executions, *J. Comput. and Syst. Sciences*, **61**(3): 400–427, 2000.
21. L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. DeMel, An analysis of communication-induced checkpointing, *Proc. Int'l Symp. Fault-Tolerant Comput.*, 1999. pp. 242–249.
22. J. Tsai, Y. M. Wang and S. Y. Kuo, Evaluations of domino-free communication-induced checkpointing protocols, *Inform. Process. Lett.*, **69**: 31–37, 1999.
23. R. Baldoni, J. M. Helary, and M. Raynal, Rollback-dependency trackability: Visible characterizations, *Proc. 18th ACM Symp. Principles of Distrib. Comput.*, 1999, pp. 33–42.
24. J. Tsai, S. Y. Kuo, and Y. M. Wang, Theoretical analysis for communication-induced checkpointing protocols with rollback-dependency trackability, *IEEE Trans. Parallel and Distrib. Syst.*, **9**(10): 963–971, 1998.
25. I. C. Garcia and L. E. Buzato, On the minimal characterization of the rollback-dependency trackability property, *Proc. IEEE Int'l Conf. Distrib. Comput. Syst.*, 2001, pp. 342–349.
26. J. Tsai, On properties of RDT communication-induced checkpointing protocols, *IEEE Trans. Parallel and Distrib. Syst.*, **14**(8): 755–764, 2003.
27. R. Baldoni, J. M. Helary, and M. Raynal, Impossibility of scalar clock-based communication-induced checkpointing protocols ensuring the RDT property, *Information Processing Lett.*, **80**(2): 105–111, 2001.

JICHIANG TSAI
National Chung Hsing University
Taichung, Taiwan
YI-MIN WANG
Microsoft Corporation
Redmond , Washington

COORDINATION AND SYNCHRONIZATION: DESIGNING PRACTICAL DETECTORS FOR LARGE-SCALE DISTRIBUTED SYSTEMS

INTRODUCTION

Large-scale distributed systems such as PlanetLab (1), peer-to-peer systems (e.g., (2–4)) , Grid networks (5), and so on, have exploded in popularity in the past few years. It is well known that such systems are failure-prone, for example, "nodes" (client machines or computer hosts) can join and leave the system at will (a phenomenon called churn), and messages can be dropped by the underlying network.

Several distributed applications have began to run atop such clusters, for example, distributed computations, cooperative file sharing, multimedia and content streaming, resource discovery, and application-level DNS. To enable coordination and synchronization, in each of these distributed applications, the application must keep track of the behavior of each node involved in the application. Intuitively, each node has individual "personalities" from the viewpoint of its cooperativeness or willingness to contribute to the overall good of the system. Thus, it is important to keep track of the individual characteristics of these nodes in a distributed fashion.

On the one hand, at the most basic level, simple node-level failures must be detected. For instance, when a node fails (or joins the system), some other nodes that are currently in the system need to be made aware of the change in the *membership*. Similarly, some nodes may be modifying messages maliciously or deviating from the core protocols specified as a part of the system—it is important to detect (and then perhaps punish) such nodes.

At the other extreme, several applications must detect system-wide properties. We consider one interesting class of detectors that fall at this end of the spectrum—one requires nodes to be aware of the approximate size of the system, that is, the number of non-faulty nodes present in the system currently. In between these two extremes, some applications track the individual availability history of nodes, which includes, their up/down characteristics. This availability of information can then be used to place replicas (of files or services) so as to maximize the availability of the service being replicated, to ensure that the multicast reliability at recipient nodes varies as a function of the node's availability, and so on.

We broadly call the above problems of measuring node-specific or aggregated, system-wide properties as the problem of *detection.* A variety of detectors for distributed systems exist, and it is possible that a book-length article could be written to cover these various detectors! To maintain brevity, this article focuses on a "sliver" of detectors from across the spectrum of node-level to system-wide detectors.

Specifically, we will focus only on the failure-, Byzantine-, and availability-related classes of detectors . We will mention, at appropriate places, other detector classes that are not covered here and that the reader may be interested in researching. The reader should use this article as a beginning step to understand more about the topic.

Notice that the latter extreme of detection is related to "statistics collection" and aggregation (6,7), but we are interested only in the actual availability-related or behavior-related characteristics of nodes, and not in collecting statistics that are specific to a particular application. In other words, most solutions to the detection problems we will discuss can be used by a wide variety of distributed applications. Furthermore, we hope this article will motivate the reader to read existing literature on other problems such as termination, deadlock detection, snapshots, and reputation mechanisms.

We focus on practical solutions with the two following characteristics: (1) they have been implemented and validated in experimental evaluation or practice and (2) they are based on novel ideas and on strong theory. Our goal here is to enable and to enhance the understanding of such viable and practical solutions for practitioners to use in real systems.

Thus, this article considers four main classes of detection problems.

1. *Crash Failure Detectors:* When the given node in the system crashes, other nodes that knew about it should be informed that it crashed.
2. *Byzantine Failure Detectors:* When the given node deviates from the specified application protocol behavior, other nodes that are non-Byzantine must be informed.
3. *Availability Detectors:* The system (or a small set of nodes) maintains information about the availability history of each node.
4. *System Size Estimators:* An initiator node (or all nodes in the system) must know about the approximate number of non-faulty nodes present as a part of its distributed group. This group could be either a one-shot or a continuous estimation problem.

Two points must be noted here. First, we are interested primarily in *fully distributed solutions* to these problems. That is, protocols that operate in a peer-to-peer fashion, without requiring a central server, are of the most interest to us. Second, we will discuss rarely the action taken by the application when such a detection is triggered. In some of the referenced papers, such reactive application behavior may be discussed. However, our discussion in this article presents detectors in a modular fashion so they can be used in a plug-and-play manner with a variety of applications.

Although the main goal of this article is to make the reader aware of practicalities of detection problems and solutions, we do highlight relevant theoretical results that form the context or indicate the difficulties of a problem.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

CRASH FAILURE DETECTORS

Here, we consider the failure detection of nodes under the *fail-stop* failure model. Under this model, either a node is non-faulty (or correct) or it has crashed. Any node can crash, and once it has done so it never executes any more instructions (i.e., it never recovers).

Crash failure detection is the core of all peer-to-peer systems and distributed systems that attempt to operate in a non-centralized manner.

Before solving any problem (such as that of crash failure detection), it is important to discuss under what *system model* (i.e., assumptions) the problem must to be solved. Primarily, two types of system models exist for distributed systems:

1. *Synchronous System Model:* Each non-faulty node has a maximum known time bound on the time taken to execute any instruction. Furthermore, a maximum known time bound on the delay is faced by a message sent by one non-faulty process to another non-faulty process. An example of a system that follows this model are multiprocessor systems such as supercomputers. Notice that nodes are still allowed to fail in this model.
2. *Asynchronous System Model:* Unlike the above model, the asynchronous model imposes no limits on either the time taken by a non-faulty node to execute any instruction nor the message delays. In other words, messages can be delayed an arbitrarily long time, and nodes can be arbitrarily slow without being faulty. Most practical networks follow the asynchronous system model, for example, Internet, wireless networks, and sensor networks.

Fail-Stop failure detectors are interested in two properties:

- Completeness: The percentage of failures that are detected eventually by all concerned non-faulty nodes.
- Accuracy: The percentage of detections that correspond to a failed node.

Notice that it is easy to guarantee trivially either 100% completeness (each node always consider all other nodes as crashed all the time) or 100% accuracy (each node never considers any other node as crashed at any point of time).

Chandra and Toueg (8) showed that it is impossible to guarantee both 100% completeness and 100% accuracy in an asynchronous system model. In the synchronous system model, however, implementing a complete and accurate failure detector is straightforward—any one of the following detectors for asynchronous systems can be used, along with timeouts that are decided based on the message delay and the instruction processing bounds.

In view of the above impossibility, most distributed applications have come to expect 100% completeness (and thus probabilistic accuracy) from the underlying crash failure detector. Each crash of a node in a distributed application must be followed by a repair or recovery operation in that application—thus, it is important to detect each failure, but it is alright to have mistaken detections. All algorithms we discuss below guarantee 100% completeness.

Chandra and Toueg (8) were the first to present failure detectors with a view for solving the problem of consensus for a bit, in a process group. Specifically, they provide a taxonomy of detectors in Ref. (8), which includes the weakest failure detector to solve consensus. Substantial work has occurred in the theoretical community since then on failure detectors for a variety of system models, e.g., see Ref. (9). However, we preclude such papers (even though classic) because either they do not scale to large distributed systems with thousands of nodes or they have not been validated in practice.

For asynchronous systems, practical failure detectors for fail-stop failures tend to be of two types: (1) heartbeating based, and (2) ping-based.

Heartbeating-Based Failure Detectors

Each node *n* sends an “I am alive” (heartbeat) message periodically (once every *hb* seconds) to a subset of other nodes in the system. Successive heartbeat messages are numbered with monotonically increasing sequence numbers to be distinguishable. Each other node that is aware of node *n* maintains the time since the last heartbeat was received from node *n*. When this time crosses a timeout threshold *(timeout* seconds), the node *n* is marked as failed.

First, this satisfies 100% completeness—once node *n* fails, it will stop sending heartbeats, and because *timeout* is finite, all previously sent heartbeats by *n* will be received and the timeout will expire eventually (at any given recipient node). In practice, the value of *timeout* is typically much larger than message transmission delays; hence, the actual detection time is *timeout* seconds.

However, this algorithm does not guarantee accuracy, especially in an asynchronous network where heartbeat messages can be delayed for an arbitrarily long time. This delay can cause another message to timeout while waiting for heartbeats from a (correct) node *n*, and consequently mark *n* as crashed mistakenly. Notice that the larger the value of *timeout* is compared with message delays, the more is the accuracy of the protocol, that is, the smaller is the false-positive rate. However, larger *timeouts* also entail longer detection times; hence, the value of *timeout* will trade off between detection time and accuracy.

Heartbeat transmission can be implemented in one of two ways—*explicit* or *implicit.* Explicit heartbeat creates separate messages for heartbeats, whereas implicit heartbeat either piggybacks heartbeat messages, atop application messages or, in some cases, uses application messages themselves as heartbeat messages.[1] For the rest of our discussion, we will assume explicit heartbeating; however, our discussion applies to the implicit variety too.

Several variants of heartbeat-based failure detectors exist—the difference between these detectors is based on which is the “subset” of nodes that receive the heartbeats

[1] We will ignore this last option mentioned because our goal in this section is to focus on application-independent protocols.

from the given node *n*. This choice is decided based on the *overlay*, or the *membership graph* (i.e., the graph defined by a node's neighbors). Below, we describe several different types of such overlays, along with the associated heartbeat-based protocol.

Simple Overlays. The classical approach was a *ring-based* overlay, with nodes arranged in a virtual ring (with no necessary correlation to their actual locations). Each node merely sent heartbeats to its clockwise neighbor (in addition, the anticlockwise neighbor was also used to increase the fault-tolerance), and these neighbors would be the only ones to detect failure of this node. In a system with N nodes, the overhead of this scheme is $O(N)$ since every node sends heartbeats periodically. The drawback of this scheme was that multiple simultaneous failures could cause an unnecessarily long delay for detecting failures, especially if a sequence of nodes in the ring failed in succession. Because the likelihood of this occurrence increases as the total number of nodes increased, the ring-based algorithm was not scalable.

A different, simpler alternative is to send the heartbeat to *all* other nodes in the system. Although this is clearly more fault-tolerant than the ring, this scheme has a very high overhead ($O(N^2)$ messages) and could have lower accuracy. Any slow node could mark a very large set of other nodes as faulty because it did not receive several heartbeat messages in a timely manner.

Gossip-style Heartbeating. Van Renesse et al. (10) made the above all-to-all heartbeating model more accurate by not having each node send its heartbeats directly to every other node, but instead *gossip* the latest heartbeat counters for several other nodes. At any node *n*, gossiping selecting entails periodically selecting a few other random nodes and sending them the array of the latest heartbeat counters (from other nodes) known at node *n*.

Van Renesse et al. (10) showed that if all heartbeats could be included in each gossip message, and each node gossiped with a constant other randomly selected gossip targets every second (on average), it took $O(\log(N))$ seconds for any node's updated heartbeat information to spread to all other nodes with high probability. Here, N is the number of nodes in the system. Thus, the timeouts could be set in this range (if one knew an upper bound on the value of N). Thus, the failure detection times are small—because $\log(N)$ is a small number, and it grows very slowly, even for values of N up to 2^{32} (the number of possible IPv4 addresses), the value of $\log_2(N) = 32$.

Distributed Hash Table-based Overlays (or Structured Overlays). Distributed hash tables (DHTs), also known as structured overlays,[2] are overlays that follow a specific structure. For instance, the Pastry p2p overlay follows a hypercube-type structure, with nodes that maintain overlay "neighbors" based on prefix matches of id's assigned to nodes, these id's are assigned by hashing the node's IP address (e.g., by using SHA-1 or MD-5), but that fact is orthogonal to our discussion here. In turn, each node sent heartbeats to its neighbors in this overlay.

Similarly, in other DHTs such as Chord, a heartbeat-style strategy was used to detect failures. Information about a node failure would propagate to its immediate neighbors and might cause these nodes to select other, "better" neighbors that were non-faulty.

Random Partial Membership Graphs. Although DHTs such as Pastry and Chord follow a specific pattern of "neighbor" selection of nodes, to make resource discovery and file insertion operations very efficient, a separate class of overlays have been designed for other applications that do not use the resource-discovery functionality primarily. For instance, publish-subscribe and multicast applications often rely on the presence of a connected overlay graph among the nodes. Yet, the protocol attempts to achieve this by having each node maintain only a *small random subset* of other nodes in the system as its neighbors.

Below, we discuss the core design of one such random partial membership graph system briefly. The reader is encouraged to research other algorithms in this class, such as T-Man (11).

Scamp. Scamp (12, 13) attempts to maintain a *uniform random* overlay graph among nodes, with each node maintaining $O(\log(N))$ neighbors in this graph. This is achieved by the following mechanisms: (*1*) Each node *n* maintains a list of neighbors in the overlay, denoted as Neighbor Set(*n*), as well as the list of other nodes than point to them (the in-neighbor list), (*2*) [Node Join] When a new node joins the system, it obtains at least *c* contacts (*c* is a fixed parameter), and forwards its subscription (joining) information to *c* of these nodes. A node *n* that receives a new joining node's information will include it in with probability $1/(1 + |\text{Neighbor Set}(n)|)$; otherwise, it forwards this subscription to one of its neighbors, selected at random. (*3*) [Node Departure] A voluntarily leaving node *n* asks the highest-id *c* neighbors of itself to delete *n* from their neighbor lists. Every in-neighbor of *n* is asked to point to another of the previous neighbors of *n* (which excludes the *c* selected above)—duplicate selections may be allowed.

Although the basic SCAMP assumes voluntary departures only, each node sends heartbeats periodically to all of its neighbors. This avoids a node from being partitioned (isolated) out from the network—when a node has not received *any heartbeats* from any other node, it knows that it is partitioned.[3]

The authors show in Ref. (12) that this protocol causes each node to have an expected $(c+1)\log(N)$ neighbors, and that the distribution of neighbor selection is random (i.e., the probability distribution of the number of in-neighbors at a node has a small standard deviation).

It is easy to see how SCAMP can be extended to handle fail-stop failures—all neighbors of a given node would time

[2] To be more precise, a structured overlay is the actual underlying overlay, whereas a DHT is layered atop this overlay and provides get- and put-style functionalities to an application. However, today, the two terms are often used as synonyms by several sections of the distributed computing community, and hence we treat "DHT" and "structured overlays" as synonyms in this article.

[3] Note that this does not avoid a large subgraph from being partitioned out of the overlay!

out waiting for a heartbeat and then execute actions similar to the voluntary unsubscriptions described above. However, it is not clear whether this would continue to maintain the uniform randomness of the overlay. Furthermore, false positives could occur—any node that misses a heartbeat would propagate a failure notification, and a suspected node would be forced to leave the group. This problem is addressed in the SWIM system discussed in the next section.

Ping-Based Failure Detectors

Unlike heartbeat-based failure detectors, ping-based failure detectors do not use any kind of heartbeat messages. Instead, each node n is pinged periodically by a subset of other nodes in the system. If the node is unresponsive, the pinging nodes could retry the pinging. If several retries do not lead to a response, the node n is marked as crashed. Below, we describe two such ping-based failure detectors: SWIM and CYCLON.

Swim. The SWIM system (14) by Das et al. has each node periodically (once every T seconds) select *one* other node (say n) uniformly at random from across the system and ping this remote node. If the remote node is unresponsive (T is assumed to be larger than the typical round-trip time in the system), then the pinging node may ask up to K (value fixed) other nodes to ping the node n indirectly and return replies (if any). If either the direct or any one of the the indirect pings results in a positive reply from n, the pinging node takes no additional action. However, in the absence of a response, the pinging node marks node n as crashed.

Clearly, this protocol satisfies 100% completeness—a crashed node will be picked eventually as a ping target by some node in the system, and be detected as failed. Furthermore, the authors showed that this protocol has a constant failure detection time *on expectation*, for examples, for $K = 0$, the expected time between failure of node n and the *first* other node that detects this failure is $\frac{T}{1 - e^{-1}}$ seconds. It is important to note that this time does not depend on the size of the system, this is a desirable and scalable property, especially in a really large distributed system. Furthermore, the authors show how to tune the value of K to obtain a tradeoff between the detection time, the false positive rate (the inaccuracy rate), and the overhead (messages per second per node). The reader is referred to Ref. (14) for more details.

Cyclon. CYCLON (15) is another membership protocol that attempts to maintain a uniform, random membership graph while having each node maintain only a small number of neighbors. Briefly, each node maintains an *age* for each of its neighbors, which denotes the time since that neighbor entry was created at node n. Each node does the following two actions periodically—eliminate the neighbor with the maximum age and exchange neighbor lists with this oldest aged neighbor. CYCLON then describes a specific way to update the neighbor lists to maintain the uniform randomness of the overlay graph. However, notice that this selection of the oldest age neighbor is *implicit failure detection*, but in the heartbeat style. If this oldest neighbor does not respond, it is deleted. Thus, failed nodes disappear eventually from neighbor lists. If the size of neighbor lists is $O(\log(N))$, then the failure detection time is also $O(\log(N))$, which is small!

BYZANTINE FAILURE DETECTORS

Unlike the fail-stop failure model discussed in the previous section, the *Byzantine* failure model specifies that nodes can behave in any arbitrary and perhaps malicious manner, that is, a Byzantine-faulty node could deviate from the protocol specified by the application in arbitrary ways. For instance, it could execute instructions that are unauthorized or do not result from the applying the specified protocol on its received messages, it could send messages with malicious intent or junk content, or claim to have received messages that it never received. In short, the Byzantine model is the most general of all models of failure. Clearly, it encompasses the fail-stop failure model.

Yet, the Byzantine model is a very realistic model. Hosts whose security has been compromised, by viruses, worms, or human hackers, as well as a process based on buggy program code, all follow the Byzantine model.

The traditional approach to handling Byzantine failures has, until very recently, been to *mask*, rather than to detect, these types of failures. Most protocols for Byzantine fault tolerance are replicated state machines with a focus on solving problems such as atomic commit and consensus (8, 16–19), that is, where all nodes must agree on the value of a variable. These protocols assume that at most f faulty nodes exist in the system, and at least $3f + 1$ total nodes exist in the system (faulty or not). Several such protocols have been specified in theory (20,21) and in practice (22,23). These protocols are designed to allow the non-faulty nodes to solve the agreement problem in the presence of up to f Byzantine nodes among them. The reader would be interested to know that it has been proved (24) that one cannot implement Byzantine fault-tolerant consensus when more than one-third of the nodes are faulty, hence these protocols have "optimal" tolerance.

Although these protocols [especially Castro and Liskov's (22)] are highly practical and perform well in real systems, they are unable to tolerate more than f failures. If one used a Byzantine failure detector instead, the following advantages could be obtained (25):

- More than f failures could be detected (and tolerated, if the application is equipped with mechanisms to respond to detected failures). In fact, no upper bound exists on the number of Byzantine nodes in the system.
- The common case (where all nodes are non-Byzantine) becomes very efficient w.r.t. performance metrics such as throughput, latency, and scalability. Simplicity of design is preserved because typically detectors are designed to fit in very modularly with the rest of the application.
- Many applications do not need to solve the consensus problem, and Byzantine failures are interested in other problems that are not related to consensus.

For these problems, applications require information about the nodes that might be faulty. We remind the reader that one cannot implement Byzantine fault-tolerant consensus when more than one-third the nodes are faulty (24).

The same properties of completeness and accuracy apply to Byzantine failure detectors (thus no failure detector can achieve both properties with a 100% guarantee). Below, we briefly describe two systems—LOCKSS and PeerReview—that provide some semblance of Byzantine failure detectors. Besides these two systems, other systems exist that come close to providing a detector, but do not provide one that is fully specified. Aiyer et al. (25) provide a mechanism to monitor quorum systems so that an alarm is raised when failure assumptions are about to be violated. Intrusion detection systems work at the level of a single node. Reputation systems (see, e.g., Ref. 27) monitor the behavior of nodes in a p2p system but do not provide a notion of detection of Byzantine failure.

Before we discuss these systems, we note an important point—a Byzantine failure detector depends, to some extent, on the application itself, for example, what is considered to be unacceptable behavior by a node. Yet, the LOCKSS system is generic enough to be applicable to any distributed storage solution, whereas PeerReview applies modularly to any distributed application that allows auditing actions on application logs.

LOCKSS (Lots Of Copies Keeps Stuff Safe). The LOCKSS system by Maniatis et al. (28) provides a protocol to maintain a consistency of replicas—LOCKSS is implemented in the context of a digital library archive, where archival units (AUs) are the basic blocks that are replicated across multiple nodes. The challenge is that even though the AUs are immutable, attacks by either adversaries or bit-rot may cause some of the replicas of the AU to become corrupted as time progresses. The goal of the LOCKSS system is to (*1*) maintain the correctness and consistency of these replicas and (*2*) enable detection of an ongoing attack, especially when a large number of replicas are in disagreement with one another.

LOCKSS meets the above challenges by (*1*) building a continuously-changing (churned) overlay among nodes and (*2*) using this overlay to execute periodic polling on the replicas of the AU (to check for and correct their consistency). We do not describe here the intricate details of the protocol, viz., or the actual quotas on how much of the list is churned for each of the above actions. The reader is encouraged to read Ref. (28) for all details and adversary attacks on the protocol.

In brief, the protocol works in the following manner. Each node n:

1. Maintains two types of neighbors—inner circle neighbors (more trusted) and outer circle (less trusted) neighbors. At any time, the inner circle consists of a random subset of other nodes that have agreed with the recent *votes* of this node n. In addition to these two circles, node n maintains a list of *friends*—other nodes on whom it places a very high level of trust.
2. Initiates a *Voting procedure* periodically. This procedure is done by querying the inner circle neighbors, each of which in turn nominates a few nodes for n's outer circle. Then n chooses a small random subset from each nomination and asks these nodes to vote. Each vote is classified as either "agreeing" or "disagreeing" with n's own vote. This calculation is based on the hash of the replica of the AU in question, that is, the entire contents of the AU replica are hashed to generate a signature and this signature is matched. (In addition, the LOCKSS protocol marks each vote as either valid or invalid based on a proof of computational effort. For our purposes, an invalid vote will result in the offending voter being ignored and removed from the neighbors lists at n.) Finally, if V total votes were requested and received, then three cases may arise: *(1)* if the number of agreeing votes is at least $V-D$, the poll was successful and n retains its replica, (*2*) if the number of agreeing votes was no more than D, the poll was a failure and n repairs its replica (from a random disagreeing neighbor), and (*3*) if the number of agreeing votes is between D and $V-D$, n raises an *alarm* (the effects of an alarm are described below). D is a configurable parameter.
3. *Churns* its neighbor lists. After each vote, the inner circle neighbors who have disagreed or who have not voted for awhile are eliminated. A random subset of the remaining nodes is left in, a few random recently agreeing nodes (no voting) from the outer circle are brought in, and finally a few random friends are brought in. The goal of this churning of neighbors is to ensure that malicious nodes do not gain a foothold on the neighbor list of a node n for too long.

The authors of the LOCKSS system show that under a variety of adversary attacks, if most of the replicas of the AU are good (resp. bad), then most polls will end successfully (resp., in failure). However, and most importantly, it takes a very long time for an AU with predominantly good replicas to transition to a state with predominantly bad replicas. Hence, the *alarm* condition raised in the specification above will have enough time (and enough alarms) to detect this shift. In the authors' words—"The rate at which at an attack can make progress is limited by the smaller of the adversary's efforts and the efforts of the victims." Put another way, LOCKSS slows down the conversion of good replicas into bad replicas (which occurs because of the presence of malicious nodes) so much that victims (i.e., good nodes) are able to fix the bad replicas. Thus, even a delayed and slow human response to such an alarm would restore the correctness of the system because the adversaries are slowed down considerably by LOCKSS.

Notice that even though the above protocol does not *detect* Byzantine nodes explicitly in the system, but it is able to detect *disagreeing* votes. In the case of alarms raised for the AU, compromised nodes can be detected easily (via their proposed hashes for the AU) and thus repaired. Logs of the votes obtained can be used to detect faulty nodes (albeit perhaps with human involvement), and if a particular group of nodes is raising alarms, a local spoofing alarm could be raised to audit local nodes.

PeerReview. The PeerReview system (25) shares some common design characteristics with the LOCKSS system designed above. However, unlike it, PeerReview provides for explicit Byzantine failure detection with interesting completeness and accuracy properties (see below). Specifically, PeerReview ensures that a correct node will never be declared as being faulty (assuming that the node is indeed responsive). *This is a major difference from the fail-stop failure detectors of Crash failure Detectors*.

The PeerReview protocol has each node monitor application protocol-compliance of all other nodes in the group. It is potentially expensive and inefficient (it involves $O(N^2)$ messages in the system), however it is a good first-cut at this difficult problem. Among the several assumptions made by PeerReview, the most important ones are: (*1*) Messages sent by correct nodes are eventually received by the recipient (if it is correct) and (*2*) the application protocol for which compliance must be checked is a replicated state machine (29).

First, each node *n* maintains a log of all its previous protocol actions and uses this to sign messages. Top-level hashes of the log are taken periodically and on-demand—such *authenticators* are piggybacked on top of all messages sent out by *n*. In other words the log is maintained as a hash chain. All messages must be acknowledged (acknowledgment messages also carry authenticators). Besides the authenticator, each message sent by *n* also contains a short *proof* that the latest message is the latest action in the local log. Finally, node *n* periodically forwards to other nodes the authenticators it knows for other nodes; this ensures eventual dissemination of any authenticator.

Second, each node *n* is *audited* periodically by other nodes *j*. Node *j* can show that *n* is faulty if either (*1*) it has an authenticator and a log both from *n*, both signed by *n*, but disagreeing with each other or (*2*) a signed log segment from *n* that fails a conformance check. During the audit phase, node *j* can begin to *suspect n* if the latter is either unresponsive or noncompliant. Otherwise, node *j* performs a consistency check to see if the log matches the recent authenticators it has for *n* (this is for rule (*1*) above). Then, node *j* extracts all authenticators from the log segment and forwards them to all other nodes—this ensures eventual dissemination of these authenticators to all other correct nodes. Finally, *j* performs a conformance check for step (*2*) above. This phase is perhaps the most computationally expensive operation in the protocol. node *j* instantiates a local copy of the application state machine i.e., algorithm replays all inputs form the log, and checks whether outputs match the ones in the log.

Notice that any deviation based on the above checks can be forwarded to other interested nodes, who can then verify for themselves whether node *n* is faulty, either by repeating the checks for itself or by contacting node *n* directly to re-do the checks.

This helps PeerReview to ensure a nice variant of the Accuracy property—*no non-faulty node will be suspected or detected by another non-faulty node.* The completeness property is not guaranteed either, but an interesting variant of it is guaranteed. Although it is possible that a faulty node may in fact escape detection forever, it is true that if many faulty nodes exist in the system, at least one node will be detected eventually. Thus, a finite number of bad nodes can affect the good nodes for only so long.

PeerReview has been implemented and found to perform well in practice—readers are referred to Ref. (25) for more details. However, at the time of writing this article, it remains to be seen what alternative Byzantine failure detectors can be designed. Furthermore, whether this detector class can be made scalable at all remains a million dollar question!

AVAILABILITY DETECTORS

After having discussed detectors for *online* individual node-level characteristics (crash and Byzantine), we transition to the problem of *availability detection.* The failure model considered here is the *crash-recovery* model, where a node can leave or fail away from the system and rejoin the system later with the same node identifier.

The availability detection problem is to estimate the *short-term* or *long-term* up/down characteristics of each node *n*. The earlier detectors informed other nodes of the immediately recent failure of a node *n*—that is not our goal here; instead, tracking the up/down characteristics of *n* is our goal.

Availability detection is an absolutely essential component in the design of many peer to peer storage systems, e.g., see Refs (30) and (31). In these systems, the availability histories of nodes are used to select the best set of nodes to hold replicas for a given object to increase the system-wide availability of the object. In these systems, availability detection is sometimes tied to an *availability predictor*, which predicts the future availability of node *n* based on its history.

Availability detection is also useful in trying to satisfy reliability predicates, where the reliability of an application protocol (e.g., multicast) at a recipient node is tied to the availability of that node, e.g., see Ref. (32).

Below, we describe different types of availability detection schemes. Notice that detection schemes typically have two subcomponents: *who monitors node n*, and *how the availability history of n is maintained at other monitoring nodes*. We discuss both these issues below. Furthermore, in cases where availability prediction is possible, it is described briefly, as well.

Group-Based Master Detectors

The Total Recall system (30) uses a *master node* in a group (of replica-holding nodes) to detect the availability of the nodes that hold replicas, and to maintain availability history, as well as to predict availability. It uses this to select the best set of replicas. The master node is selected on a per-object basis, and it is responsible to monitor (via pings or heartbeats) the availability of two types of nodes: *inode storage nodes* and *data storage nodes*. Higher-granularity availability information is maintained for the former set of nodes (and those lost replicas repaired eagerly by the master), whereas the latter set has lower-granularity availability detection (and those lost replicas are repaired lazily).

Group-Based Distributed Detectors

Carbonite (31) and HBHC (33) each use more distributed schemes than group-based master detectors, but once again, these schemes work within small groups of nodes (holding replicas of a given object).

Carbonite's availability detection works by creating a spanning tree [of height $O(\log(N))$] rooted at each node in the group, which contains other nodes at its leaves. The spanning tree is created using the routing algorithm of the underlying p2p DHT (distributed hash table). Each node sends out heartbeat messages to its children periodically, and the heartbeat is propagated down the tree to its leaves. If a heartbeat is missed, the monitoring node triggers a repair for every object stored on the node detected as down. In a manner, this scheme is a crash-recovery protocol, but we include it in this section because it is used by Carbonite to measure the availability history of individual nodes.

HBHC (33) is another system for replica maintenance. The availability monitoring in HBHC is also fully distributed within the replica group. In brief, each node pings each other node in the group periodically, that is, it is an all-to-all pinging scheme. This information is also disseminated periodically to all other nodes in the group using a gossip-style (epidemic-style) dissemination (34).

System-Based Detectors

AVCast (32) is a system that links the multicast reliability at recipient nodes to their availability. The availability monitoring occurs on a system-wide basis, without assuming replica groups. Thus, it is a general scheme. To start, availability monitoring can be done either by having each node report its own availability individually or by using the overlay structure itself to decide which nodes monitor the availability of a given node n. The former approach is infeasible because nodes can lie about their own availability, whereas the latter scheme does not generalize easily because in power-law overlays [e.g., Gnutella (2)], higher-degree nodes would have a higher monitoring overhead.

Instead, AVCast's detector says that a node m will monitor another node n if the condition $\text{Hash}(m, n) < K/N$, where Hash is a consistent hashing function with range [0,1], m and n are id's of the nodes, K is a small fixed constant, and N is the approximate system size (a fixed quantity at all nodes). If the actual system size stays within a constant factor of N, each node will have an expected $O(K)$ other *random* nodes that monitor its availability via ping and reply messages. Besides ensuring load balance, this scheme is *verifiable*; any third node can verify (using the hash condition above) if two nodes m and n are in fact related by a monitoring relationship. Thus, it is very difficult for a node n to cheat others either by reporting a higher availability for itself or by colluding with other nodes. Reference (32) describes additional optimizations in the algorithm, where the value of K is changed adaptively—the reader is encouraged to read the paper for details.

Types of Availability History

Although the detectors of the previous sections merely maintained a straightforward history of the availability of a given node n, and calculated its availability as the average of all previous availability-test points (i.e., times at which the availability of the node was explicitly measured), other approaches to *maintaining history* are possible. References (32) and (35) discuss some of these approaches very well in the context of the goal of *availability prediction*, and we describe some below. Notice that most of these history-maintenance schemes can be used orthogonally along with the availability monitoring schemes above. However, we do not discuss integration issues here.

RightNow. This is just the current up/down status of the node.

Aged. Reference (32) uses an aged detector, where the last k availability tests on a node are weighed in an aged manner, with more recent availability tests weighed exponentially heavily compared with older tests. This aged equation is used to estimate the availability probability of node n. This aging rule is similar to the aging-based prediction of run times of tasks in operating systems.

SatCount. In this scheme (35), the availability of a node is marked as one of 4 values (using a 2-bit counter), based on its history. These values are -2 (strongly offline), -1 (weakly offline), $+1$ (weakly online), $+2$ (strongly online). This categorization is based on the results of the past k availability-testing points for node n.

de Bruijn Graph-Based. For each node, the last k points of availability testing are maintained, with the most recent tests being in the lowest significant bits. A left-shift operation is done with each new test. Using this as a basis, a state machine based on a de Bruijn graph can be set up among the 2^k possible availability states for node n (for the *k-bit* availability history). In a de Bruijn graph, each of the 2^k states leads into the two other states obtained by left-shifting it. Reference (35) describes how to predict availability of a node n based on this—either by following the most likely path from the current availability state, or by following multiple paths, or by using a linear predictor (this works best for short-term-stable availability behavior). These techniques are based on digital signal processing approaches. Finally, a hybrid detector combines all the above using an adaptive tournament scheme. Readers are encouraged to read Ref. (35) for more details. Using availability traces collected from two different clusters, the authors showed that in practice, the hybrid detector and predictor work very well for home and office clusters and moderately well for geographically distributed clusters like PlanetLab.

SYSTEM SIZE ESTIMATORS

Finally, we discuss how to detect system-wide properties related to failures. Specifically, we discuss different approaches to solving the *System Size Estimation* problem in large-scale distributed systems. Informally, the problem involves finding the "current" (at initiation time) number of non-faulty processes present in a distributed system, since nodes can join and leave at any point of time. First, notice that an accurate estimate is impossible to achieve—messages have non-zero latencies, and departure or failure of even a single node, immediately after its last message with respect to the estimation protocol, will lead to an inaccurate estimate (and this is very likely to occur in large-scale distributed systems).

Such estimation protocols are extremely useful in many distributed systems, which includes p2p overlays whose design depends on the value of system size N(36,37), nodeID assignment schemes (36), for estimating the latency of lookups in some log(N)—p2p overlays (3,4). Finally, estimated system sizes can be used to monitor and to audit performance of distributed applications, (e.g., on PlanetLab), as well as for dynamic partitioning of Grid applications.

Like our prior detection protocols, we desire our estimation protocols to be scalable, efficient, fault-tolerant, and practical. In addition, increasing the (probabilistic) accuracy is an important goal. The estimation problem comes in two flavors—*one-shot* detection involves a one-time estimation of system size, whereas *continuous* detection involves estimating the system size continuously. Accuracy can be defined as either the root mean square of the error between the estimated system size and the current system size or as the standard deviation of these errors. The former metric measures how *close* the estimate is to the actual size; the latter metric measures how *consistently* the estimated size shadows the actual size.

Protocols for system size estimation come in two varieties—*active protocols* and *passive protocols*. Active protocols must be initiated by a single node and they involve passing messages around inside the group until the initiator receives enough responses or information to draw an estimate. This style of protocol is a one-shot solution, but it can be repeated for a continuous implementation.

Passive protocols, on the other hand, do not involve exchanging any estimation messages actively. Instead, these protocols attempt to *snoop* on messages sent by the application or by a membership protocol (such as the ones discussed in Crash Failure Detectors) to obtain an estimate. By nature, they are continuous estimators.

Below, we discuss a small subset of active and passive estimation protocols. Following the theme of this paper, we choose only protocols that are the most practical, are implemented easily and have the least assumptions to hinder their transition into practice.

Active Estimation Protocols

Bawa et al. and Sample & Collide. Both Bawa et al. (39) and Massoulie et al's Sample and Collide scheme (40) use the birthday paradox to estimate the system size. These protocols initiate a random walk within the distributed system—each node uses its neighbor information (provided by any of the group membership protocols such as the ones discussed in Crash Failure Detectors). If the number of nodes in the system is N, it takes an expected number of $\sqrt{2N}$ steps to get back to a node that was already traversed by the random walk. Based on this, the system size is estimated.

Aggregation-Based Protocols. Several aggregation protocols have been proposed for the distributed system. These protocols calculate the sum, average, min, and max, of a set of values provided by the nodes in the system. Jelasity et al. (41) use one such aggregation protocol to derive an estimation protocol. Basically, once the protocol is initiated, each node keeps an estimate of the current size of the system. At node n, this value is initialized to 1 when the initiating message is received. Periodically, node n exchanges its value with one neighbor chosen at random and replaces its current estimate with the average of these two estimates. The authors of Ref. (41) then show that the estimate converges in time that is logarithmic in the group size. Several other aggregation protocols such as those by Kempe et al. (42) could also be used potentially to derive a system size estimate similarly.

Hops Sampling. This scheme (43,44) involves disseminating a *gossip* message (also called *epidemic* message) into the group and measuring the average latency of the receipt times of this gossip. Because the dissemination latency of a gossip varies logarithmically with the system size, an estimate for the latter can be derived. The basic gossiping model works as follows: when a gossip message is received at node n, this node (once every T seconds) selects a fixed constant number of gossip targets (nodes) periodically at random and sends them copies of the gossip. In addition, the Hops Sampling approach carries the *hopcount* variable (initialized to 0 by the initiating node); when a node n first receives the initiating message, it notes the hopcount, increments it by 1, and then starts to gossip the initiating message with the new hopcount piggybacked on top of it. Finally, after $O(\log(N))$ rounds, the initiating node queries a small subset of nodes in the system to sample their hopcounts, relates this to log(N), and obtains a system size estimate. The latency of this protocol is also logarithmic in the system size.

Comparing the Above Three Approaches. Le Merrer et al. have compared the above three active approaches quantitatively via simulations (45). They found that using aggregation (with estimates over last 50 rounds) provides the best accuracy, whereas Hops Sampling (with the estimate averaged over last 10 runs) provides lower accuracy comparatively. Admittedly, this particular comparison lacks a common baseline across the algorithms (e.g., the number of messages exchanged), but it is clear that Hops Sampling uses significantly fewer messages than aggregation, whereas Sample&Collide uses the least messages and has somewhat middling accuracy. Overall, the comparison does show that these different active protocols define an overhead-accuracy trade-off. This opens the door for the design of adaptive estimation protocols (e.g., try to achieve a given level of accuracy while trying to stay within an overhead budget).

Other Active Estimators. Awerbuch and Scheideler (46) assign special id's to nodes and organize them in a hierarchy to enable estimation. Malkhi and Horowitz (47) use a ring-based algorithm for estimation, however unlike the above schemes this scheme could have a very high error. Finally, several systems have been proposed to estimate the size of p2p overlays [e.g., Stutzbach and Rejaie's crawler-based approach (48)].

Passive Estimation Protocols

Passive protocols for size estimation do not initiate one-shot runs of the protocol. Instead, they snoop on application or membership protocol messages to estimate the system size. Furthermore, this class of protocols enables *each and every node* in the system to have an estimate without restricting this knowledge to a privileged initiating node.

The Interval Density Scheme (44) is one such passive estimation protocol that works by snooping on the messages passed along by a gossip-style membership protocol such as the one by van Renesse et al. (10) (discussed in Crash Failure Detectors). Basically, given a membership protocol (such as Ref. (10)) that enables each node n in the system to eventually (and perhaps quickly) learn information about the id or IP address of each other node joining into, or failing or departing from the system, the node n can estimate the system size *by only remembering a small fraction of these node ids*.

Each node uses a consistent hash function (e.g., one based on SHA-1 or MD-5) to hash a heard-of node IP address into the real interval [0,1]. In a nutshell, node n is interested only in those other nodes whose IP addresses hash into a sub-interval I of the interval [0,1]. If I is of size $O(K/N)$, where K is a constant, and N is the (approximate) system size, then the memory use at node n because of this estimation protocol is merely $O(K)$. Furthermore, using snooping on the gossip-style membership protocol, it turns out that the time for a node join or failure to reflect at the estimate at all other nodes is $O(\log(N))$. Finally, the inventors of this scheme showed that it suffices for K to be $O(\log(N))$ to derive an accuracy of the protocol that goes to 1 as the actual system size increases toward infinity.

Reference (44) describes several ways to adjust both the size and the centerpoint of the interval I at node n so as to obtain an accurate detection—the reader is encouraged to read the referenced paper for more details.

Active vs. Passive Approach. The active Hops Sampling approach was compared with the passive Interval Density scheme in Ref. (43). Both these algorithms are available as part of an open-source software called *Peer-Counter*(44). Overall, if the group size is more or less static, the passive approach yields a better accuracy. If the group size is highly dynamic, the algorithms perform comparably when one considers the root mean square error. However, the passive scheme has better performance w.r.t. the standard deviation of these errors (i.e., it is able to shadow the variation of system size better).

SUMMARY

In this article, we have discussed online detectors for several types of problems in large-scale distributed systems. We have seen (*1*) heartbeat- and ping-based detectors for crash failures, (*2*) implicit and explicit Byzantine failure detectors, (*3*) master-based, group-based and fully distributed availability monitors, and (*4*) active and passive system size estimation schemes. Our focus was on approaches that were practical yet novel at their core. This continues to be a flourishing area of research, which implements ideas in a variety of real systems.

REFERENCES

1. L. Peterson, T. Anderson, D. Culler, and T. Roscoe, A blueprint for introducing disruptive technology into the internet, Proc. *HotNets-I*, 2002.
2. The Gnutella protocol specification. Available: http://www9.limewire.com/.
3. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, *Proc. ACM SIGCOMM Conference*, 2001, pp. 149–160.
4. A. Rowstron and P. Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, *Proc. IFIP/ACM Middleware*, 2001.
5. I. Foster, C. Kesselman, and S. Tuecke, The anatomy of the Grid: Enabling scalable virtual organizations, *Internat. J. Supercomp. Appl.*, 2001.
6. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, TinyDB: An acqusitional query processing system for sensor networks, *ACM TODS*, 2005.
7. R. vanRenesse, K. Birman, and W. Vogels, Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining, *ACM Trans. Comp. Sys.*, **21** (2): 164–206, 2003.
8. T. D. Chandra and S. Toueg, Unreliable failure detectors for reliable distributed systems, *J. ACM.*, **43** (2): 225–267, 1996.
9. W. Chen, S. Toueg and M. K. Aguilera, On the quality of service of failure detectors, *Proc. 30th International Conference on Dependable Systems and Networks (ICDSN/FTCS)*, 2000.
10. R. van Renesse, Y. Minsky, and M. Hayden, A gossip-style failure detection service, *Proc. Middleware 98*, 1998, pp. 55–70.
11. M. Jelasity and O. Babaoglu, T-Man: Gossip-based overlay toplogy management, *Self-Organising Systems: ESOA*, LNCS 3910: 1–15, 2005.
12. A. Ganesh, A.-M. Kermarrec, and L. Massoulie, Peer-to-peer membership management for gossip-based protocols, *IEEE Trans. Comp.*, **52** (2): 139–149, 2003.
13. A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie, SCAMP: peer-to-peer lightweight membership service for large-scale group communication, *Proc. 3rd NGC*, LNCS 2233, 2001, pp. 44–55.
14. A. Das, I. Gupta, and A. Motivala, SWIM: Scalable Weakly-consistent Infection-style process group Membership protocol, *Proc. IEEE DSN*, 2002, pp. 303–312.

15. S. Voulgaris, D. Gavidia, and M. vanSteen, CYCLON: Inexpensive membership management for unstructured P2P overlays, *J. Network Syst. Managem.*, **13** (2): 197–217, 2005.

16. B. Chor, M. Merritt, and D. B. Shmoys, Simple constant-time consensus protocols in realistic failure model, *Proc. 4th ACM PODC*, 1985, pp. 152–160.

17. A. Doudou, B. Garbinato, R. Guerraoui, and A. Schiper, Muteness failure detectors: Specification and implementation, *EDCC*, 1999, pp. 71–87.

18. M. J. Fischer, N. A. Lynch, and M. Patterson, Impossibility of distributed consensus with one faulty process, *J. ACM*, **32** (2): 374–382, 1985.

19. K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, Byzantine fault detectors for solving consensus, *Comp. J.*, **46** (1): 16–35, 2003.

20. B. Chor and C. Dwork, Randomization in byzantine agreement, *Adv. Comp. Res.*, **5**: 443–497, 1989.

21. M. O. Rabin, Randomized Byzantine generals, *Proc. 24th IEEE FOCS*, 1983, pp. 403–409.

22. M. Castro and B. Liskov, Practical byzantine fault tolerance and proactive recovery, *ACM Trans. Comp. Sys.*, **20** (4): 398–461, 2002.

23. J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, Separating agreement from execution for byzantine fault tolerant services, *Proc. ACM SOSP*, 2003, pp. 253–267.

24. L. Lamport, R. Shostak, and M. Pease, The Byzantine generals problem, Proc. *ACM TOPLAS*, **4** (3): 382–401, 1982.

25. A. Haeberlen, P. Kouznetsov, and P. Druschel, The case for byzantine fault detection, *Proc. Usenix HotDep*, 2006.

26. A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, BAR fault tolerance for cooperative services, *Proc. ACM SOSP*, 2005, pp. 45–58.

27. E. Damiani et al., A reputation-based approach for choosing reliable resources in peer-to-peer networks, *Proc. 9th ACM CCS*, 2002.

28. P. Maniatis, M. Roussopoulos, T. J. Giuli, D. S. H. Rosenthal, and M. Baker, The LOCKSS peer-to-peer digital preservation system, *ACM Trans. Comp. Sys.*, **23** (1): 2–50, 2005.

29. F. Schneider, The state machine approach: A tutorial, Technical Report TR 86–800, Cornell University, 1986.

30. R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, Total Recall: System support for automated availability management, *Proc. Usenix NSDI*, 2004.

31. B.-G. Chun, et al., Efficient replica maintenance for distributed storage systems, *Proc. Usenix NSDI*, 2006, pp. 45–58.

32. T. Pongthawornkamol and I. Gupta, Avcast : New approaches for implementing availability-dependent reliability for multicast receivers, *Proc. IEEE SRDS*, 2006.

33. T. Schwarz, Q. Xin, and E. L. Miller, Availability in global peer-to-peer storage systems, *Proc. WDAS*, 2004.

34. A. J. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson, Epidemic algorithms for replicated database maintenance, *Proc. 6th ACM PODC*, 1987, pp. 1–12.

35. J. W. Mickens and B. D. Noble, Exploiting availability prediction in distributed systems, *Proc. Usenix NSDI*, 2006, pp. 73–86.

36. G. S. Manku, M. Bawa, and P. Raghavan, Symphony:distributed hashing in a small-world, *Proc. 4th USITS*, 2003, pp. 127–140.

37. P. B. Godfrey and I. Stoica, Heterogeneity and load balance in distributed hash tables, *Proc. IEEE Infocom*, 2004.

38. G. S. Manku, Balanced binary trees for id management and load balance in distributed hash tables, *Proc. ACM PODC*, 2004, pp. 197–205.

39. M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, Estimating aggregates on a peer-to-peer network, Technical report, Stanford University, 2003.

40. L. Massoulie, E. L. Merrer, A.-M. Kermarrec, and A. Ganesh, Peer counting and sampling in overlay networks: random walk methods, *Proc. ACM PODC*, 2006.

41. M. Jelasity and A. Montresor, Epidemic-style proactive aggregation in large overlay networks, *Proc. 24th ICDCS*, 2004.

42. D. Kempe, A. Dobra, and J. Gehrke, Computing aggregate information using gossip, *Proc. 44th IEEE FOCS*, 2003.

43. D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, Active and passive techniques for group size estimation in large-scale and dynamic distributed systems, Manuscript currently under preparation, 2006.

44. D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, Decentralized schemes for size estimation in large and dynamic groups, *IEEE NCA*, 2005.

45. E. Le Merrer, A. M. Kermarrec, and L. Massoulie, Peer to peer size estimation in large and dynamic networks: A comparative study, *Proc. 15th IEEE HPDC*, 2006, pp. 7–17.

46. B. Awerbuch and C. Scheideler, Robust distributed name service, *Proc. 3rd IPTPS*, 2004.

47. D. Malkhi and K. Horowitz, Estimating network size from local information, *ACM Inform. Process. Lett.*, **88** (5): 237–243, 2003.

48. D. Stutzbach and R. Rejaie, Characterizing unstructured overlay topologies in modern p2p file-sharing systems, *Proc. IMC*, 2005, pp. 49–62.

INDRANIL GUPTA
University of Illinois at Urbana-Champaign
Urbana, Illinois

DISTRIBUTED DATABASES

INTRODUCTION

The development of network and data communication technology has resulted in a trend of decentralized processing in modern computer applications, which includes distributed database management. Naturally, the decentralized approach reflects the distributed organizational structure, allows the improved availability and reliability of data, and allows improved performance and easier system expansion.

We can define a *distributed database*(1,2) as a collection of data that belong logically to a single database but are stored physically in several databases over the sites of a network. Two important aspects in the definition of a distributed database exist. First, a distributed database is distributed physically within several databases called *local databases* on different sites of a network; this aspect distinguishes a distributed database from a centralized database. Second, a user may have an illusion that a distributed database is a single database (i.e., a virtual database called a *global database*); this aspect distinguishes a distributed database from a set of networked databases.

The fact that a distributed database is spread physically over several local databases, yet it is viewed logically as a whole brings challenging tasks for a distributed database management system (DDBMS), a software that is used to manage distributed databases. A distributed database system (DDBS) consists of a DDBMS and the distributed databases that it manages. The key issue of a DDBS is the support of *transparency*. With transparency, users may access and update a distributed database through a single *global schema* by using an ordinary query language such as SQL in the same way as they do to a centralized database. Three fundamental tasks must be supported by a DDBS: distributed database design, distributed query processing, and distributed transaction management.

Apart from data distribution, heterogeneity and autonomy are two other aspects of a DDBS. In terms of heterogeneity, a DDBS may be classified as homogeneous or heterogeneous. A *homogeneous DDBS* has identical local DBMSs on all sites, whereas a *heterogeneous DDBS* allows differences in their local DBMSs. Sometimes, local DBMSs of a heterogeneous DDBMS may be of different types: relational, hierarchical, network, and object-oriented. In terms of autonomy, a DDBMS with high autonomy of local DBMSs is called a *federated* DBS (FDBS) or a *mutidatabase* system (3).

Date (4) has listed twelve rules for a DDBMS. They are as follows:

1. Local autonomy. The sites in a distributed system should be autonomous. In this context, autonomy indicates that local data is locally owned, local operations remain purely local, and all operations at a given site are controlled by that site.
2. No reliance on a central site. There should be no single site without which the system cannot operate.
3. Continuous operation. Ideally, a need should never exist for a planned system shutdown.
4. Location independence. The user should be able to access all data from any site as if it were stored at the user's site, regardless of where it is stored physically.
5. Fragmentation independence. The user should be able to access the data, regardless of how it is fragmented.
6. Replication independence. The user should be unaware that data has been replicated. Thus, the user should not be able to access a particular copy of a data item directly, nor should the user have to update all copies of a data item.
7. Distributed query processing. The system should be capable of processing queries that reference data at more than one site.
8. Distributed transaction processing. The system ensures that both transactions at local and global levels conform to ACID properties (i.e., atomicity, consistency, isolation, and durability).
9. Hardware independence. It should be possible to run the DDBMS on a variety of hardware platforms.
10. Operating system independence. It should be possible to run the DDBMS on a variety of operating systems.
11. Network independence. It should be possible to run the DDBMS on a variety of disparate communication networks.
12. Database independence. It should be possible to have a DDBMS that consists of different local DBMSs. In other words, the system should support heterogeneity.

Recently, we have observed the rapid development of Internet technology and the use of XML as a standard for data formatting and exchange on the Internet. The effective management and integration of huge amounts of XML data resources on the Internet brings new topics for distributed database management.

The rest of this paper is organized as follows: Three fundamental tasks of a DDBMS (i.e., distributed database design, distributed query processing, and distributed transaction management) are introduced in the first 3 Sections, respectively. In the next Section, we discuss the problems in FDBSs. In the final Section, we discuss distributed database related research topics on the Web and XML.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

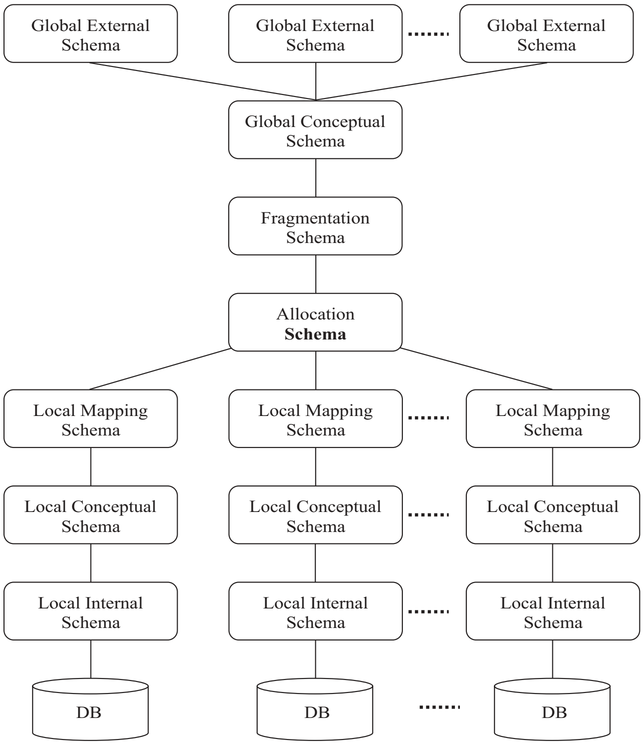


Figure 1. Distributed database architecture.

DISTRIBUTED DATABASE ARCHITECTURE AND DESIGN

The ANSI/SPARC three-level architecture provides a reference architecture for a centralized database. This architecture can be extended for a distributed database as shown in Fig. 1(5). Given that the global virtual database of a distributed database is used by end users, whereas local databases of the distributed database are actually used to store and to manage real data, the architecture only has global external schemas and local internal schemas. The task of distributed database design is to map a global conceptual schema into a set of local conceptual schema. Three steps can be followed, which result in three schemas, fragmentation schema, allocation schema, and local mapping schema. The fragmentation schema describes how global relations are partitioned into subrelations called *fragments*. The allocation schema describes on which site(s) a fragment is placed, which takes into account any replication. The local mapping schema maps fragments of allocation schema into relations in local conceptual schema. The local conceptual schema at each site defines the entire local database at the site, and the local internal schema is a physical level representation of a local database.

Fragmentation

Two types of fragmentation exist: *horizontal* and *vertical*. A horizontal fragment is a subset of tuples, and a vertical fragment is a subset of attributes of a global relation. Two rules must be followed during fragmentation: *completeness*—all the data of a global relation must be mapped into the fragments and *reconstruction*—a global relation must be able to construct from its fragments. For horizontal fragmentation, an additional rule must be followed, which is *disjointness*—no overlapping exits between any two fragments. For vertical fragmentation, the disjointness rule is allowed to be violated because a replicated attribute is required to reconstruct the global relation. The following is global schema with three global relations:

```
SALESPERSON(sid, name, commission, branch)
CUSTOMER(cid, name, address)
ORDER(oid, orddate, totamt, cid, sid)
```

A global relation can be fragmented horizontally into fragments by using a *selection* operation σ and be reconstructed from its fragments by a *union* operation ∪. For example, assume that every salesperson works in either Sydney or Melbourne but not in both branches, then we have the following:

$$\mathrm{SALESPERSON}_{\mathrm{SYD}} = \sigma_{\mathrm{branch}=\text{`SYD'}}\ \mathrm{SALESPERSON}$$
$$\mathrm{SALESPERSON}_{\mathrm{MEL}} = \sigma_{\mathrm{branch}=\text{`MEL'}}\ \mathrm{SALESPERSON}$$
$$\mathrm{SALESPERSON} = \mathrm{SALESPERSON}_{\mathrm{SYD}} \cup \mathrm{SALESPERSON}_{\mathrm{MEL}}$$
$$\mathrm{SALESPERSON}_{\mathrm{SYD}} \cap \mathrm{SALESPERSON}_{\mathrm{MEL}} = \varnothing$$

A global relation can also be fragmented horizonally into fragments that depend on the horizontal fragmentation of another global relation (derived horizontal fragmentation) by using a *semi-join* operation **SJ**. For example,

$\mathrm{ORDER}_{\mathrm{SYD}} = \mathrm{ORDER}\ \mathbf{SJ}\ \mathrm{SALESPERSON}_{\mathrm{SYD}}$
$\mathrm{ORDER}_{\mathrm{MEL}} = \mathrm{ORDER}\ \mathbf{SJ}\ \mathrm{SALESPERSON}_{\mathrm{MEL}}$
$\mathrm{ORDER} = \mathrm{ORDER}_{\mathrm{SYD}} \cup \mathrm{ORDER}_{\mathrm{MEL}}$
$\mathrm{ORDER}_{\mathrm{SYD}} \cap \mathrm{ORDER}_{\mathrm{MEL}} = \varnothing$

A global relation can be fragmented vertically into fragments by using a projection operation π and be reconstructed from its fragments by a natural join operation **NJ**.

$\mathrm{SALESPERSON}_{\mathrm{COMM}} = \pi_{\mathrm{sid,\ commision}}\ \mathrm{SALESPERSON}$
$\mathrm{SALESPERSON}_{\mathrm{DETAIL}} = \pi_{\mathrm{sid,\ name,\ branch}}\ \mathrm{SALESPERSON}$
$\mathrm{SALESPERSON} = \mathrm{SALESPERSON}_{\mathrm{COMM}}\ \mathbf{NJ}$
$\quad \mathrm{SALESPERSON}_{\mathrm{DETAIL}}$

Mixed horizontal and vertical fragmentations may be used for a global relation. Sometimes, a global relation may not need to be fragmented.

Allocation

After the fragmentation step, we have a set of fragments. The problem of allocation is how to distribute this set of fragments to the set of sites such that the distribution is optimal to a predefined set of dominant applications, which can be modeled as a set of retrieval and update references to fragments. Two basic alternatives to allocate fragments exist: *nonredundant* or *redundant*. The former places each fragment into a single site, whereas the latter may place a fragment into multiple sites. The general allocation problem is NP-hard (6). Therefore, the proposed solutions are based on heuristics.

Levels of Transparencies

From Fig. 1, a DDBMS may provide transparencies at different levels, which depend on users' requirement. The highest level of transparency is the fragmentation transparency. At this level, users do not need to know that global relations are fragmented and where the fragments are placed. Therefore, global schema is used for any retrieval and update requests. The middle level of transparency is the location transparency. At this level, users must use fragments specified in the fragmentation schema for any retrieval and update requests. However, users do not need to know the locations of these fragments and how many copies of these fragments exist. The local mapping transparency is the lowest level of transparency. At this level, users must use the allocation schema to specify not only the fragment, but also which copy of the fragment on a given site. The only thing users may not know is how the fragment is represented in the local conceptual schema.

DISTRIBUTED QUERY PROCESSING AND OPTIMIZATION

In a distributed database, a global query is expressed as references to global relations defined in the global schema. A DDBMS must transform this global query into several subqueries; each subquery executes on a local database and then combines the results of subqueries to form the result of the global query (7). The set of subqueries, the queries for combining results of subqueries, and the order for executing these queries constitute a *distributed query execution plan*. For a global query, many such query execution plans exist. The task of the distributed query optimization is to find an optimal plan such that either minimum total cost or minimum response time for executing a global query is achieved. Unfortunately, finding such an optimal plan has been proved a NP-hard problem; therefore, most of the proposed solutions are based on heuristics.

Distributed query optimization is much more complicated than its centralized counterpart. For centralized systems, the primary factor for the cost of a particular execution plan is the cost of local processing. In a distributed system, more factors must be considered.

1. The distribution of data. As global relations are fragmented and allocated, possibly with more than one copy, to several sites as local relations. Much space exists to choose which copy of a fragment to use for a global query. This process is called *materialization*.
2. Communication cost. As data is spread over different sites of a network, data transmission between sites is inevitable. For wide area networks, the speed of data transmission is much slower than that of disk access. As such, communication cost becomes a dominant factor toward the measurement of cost of a global query.
3. Potential parallelism. As subqueries are executed by local DBMSs, it is possible to parallelize the processing of these subqueries, provided they are not dependant in the execution order. As such, performance gain can be achieved by parallelism. Exploring parallelism is especially important if minimum response time is selected as the criterion of optimization.

In addition, accurate database profiles are important to estimate the cost of operations and the size of intermediate relations. A database profile contains statistic information of the databases, such as the size of relations and attributes, the data distribution information, the selectivity of operations such as selection and join, and so on.

Several optimization strategies have been used for distributed query optimization. These strategies include transformation, semi-join based, and join based strategies.

Query Transformation

Rules are available that can be applied to a query (as a expression of relational algebra) to rewrite it into an equivalent expression, (1,8). Let U and B stand for unary and binary algebraic operations, respectively. We may have the following algebraic laws for some relational algebraic operations where R, S, and T are relations.

- Commutativity: $U_1\ U_2\ R \leftrightarrow U_2\ U_1\ R$, $R\ B\ S \leftrightarrow S\ B\ R$.
- Associativity: $R\ B\ (S\ B\ T) \leftrightarrow (R\ B\ S)\ B\ T$.
- Idempotence: $U\ R \leftrightarrow U_1\ U_2\ R$.

- Distributivity: U (R B S) → U(R) B U(S).
- Factorization: U(R) B U(S) → U(R B S).

A query can be represented as an operator tree where the leaf nodes of the tree are relations and nonleaf nodes are operators. Obviously, the operators closest to leaf nodes will be executed first, and the root operator will be executed the last. The objective of query optimization based on equivalent transformation is to find an operator tree with the minimum cost for execution. In centralized databases, heuristics can be used for better performance of queries (e.g., use idempotence of selection and projection to generate appropriate selections and projections for each relation, push selections and projections down in the tree as far as possible).

In distributed databases, a global relation may be fragmented horizontally and/or vertically into fragments, and real data is stored in local relations that represent physical copies of those fragments. Therefore, a global relation in a global query must be replaced as union (horizontal fragmentation) and/or natural join (vertical fragmentation) of fragments. It is always beneficial to reduce the size of fragments before they are transmitted to other sites. Sometimes, while pushing selection down the tree to union of horizontal fragments, a contradictory qualification may be achieved for some fragments, which means no result will be obtained from those fragments. Consequently, those fragments can be removed from the query. Similarly, while pushing projection down the tree to join of vertical fragments, those fragments that do not contain the projected attributes can be removed from the query. To distribute joins that appear in the global query, unions that represent collections of fragments must be pushed up beyond the joins that we want to distribute. Although a join between two global relations with one horizontally fragmented depends the other, only joins between correspondent fragments are needed.

Semi-Join Strategy

A join operation is even more expensive in distributed databases than in centralized ones when data transmission cost is the dominant factor for query optimization. To reduce the cost of a join across sites, it is ideal to reduce the size of operand relations fully. A semi-join operator can be a reducer in most cases. The theory of semi-joins is well defined by Bernstein and Chiu (9).

Let R and S be two relations, in which A and B are the join attributes that belong to R and S, respectively, and **SJ** stand for semi-join, then R $\mathbf{SJ}_{A=B}$ S is a subset of tuples of R, constituted by those tuples that give a contribution to the join of R with S. The benefit of this is that the tuples that are not concerned with join will be filtered out before the real join operation.

A semi-join program for a join between R and S can be done by one of the following strategies:

1. R $\mathrm{JN}_{A=B}$ (S $\mathrm{SJ}_{B=A}$ π_A R).
2. S $\mathrm{JN}_{B=A}$ (R $\mathrm{SJ}_{A=B}$ π_B S).
3. (R $\mathrm{SJ}_{A=B}$ π_B S) $\mathrm{JN}_{A=B}$ (S $\mathrm{SJ}_{B=A}$ π_A R).

If R and S are from different sites and we take the first strategy, then the following program can be used to implement R $\mathrm{JN}_{A=B}$ S.

1. Send π_A R to the site of S.
2. Compute S' = S $\mathrm{SJ}_{B=A}$ π_A R at the site of S.
3. Send S' to the site of R.
4. Compute R $\mathrm{JN}_{A=B}$ S' at the site of R.

The cost of the above semi-join program is the cost of step 1 and step 3, whereas the cost of a join-based algorithm is that of transferring relation S. The semi-join approach is better if size(π_A R) + size(S $\mathrm{SJ}_{B=A}$ π_A R) < size(S), i.e.,

$$\mathrm{size}(\pi_A R) < \mathrm{size}(S) - \mathrm{size}(S\,\mathrm{SJ}_{B=A}\pi_A R)$$

Notice that the right side of the above inequity is the reduced tuples of S. The semi-join approach is better if the semi-join acts as a sufficient reducer (i.e., if a few tuples of S participate in the join). The join approach is better if most of the tuples of S participate in the join, because the step 1 of semi-join requires additional cost.

The semi-join can be useful to reduce the size of the operand relations involved in a multiple-join query. The size of an operand relation may be reduced by more than one semi-join. For example, R in a multiple join query of operand relations R, S, and T can be reduced by R' = R SJ (S SJ T). Such a sequence of semi-joins is called a semi-join program for R. For an operand relation, several potential semi-join programs exist. One of these programs is optimal and it is called the full reducer. Given that the number of semi-join programs is exponential in the number of operand relations, the cost of the full reducer program is sometimes greater than the benefit. In the DDBMS prototype SDD-1, a semi-join based algorithm has been proposed (10) based on a hill-climbing algorithm for centralized query optimization.

Join-Based Strategy

In a DDBS in which data transmission cost is much more expensive than local processing cost, the use of semi-joins can improve the performance of a query significantly. If we also consider the cost of local processing to evaluate alternative execution plans, then the direct use of joins as a query processing tactic is often more convenient than the use of semi-joins. For example, R* query optimization algorithm (11) uses joins rather than semi-joins. It uses a compilation approach in which an exhaustive search of all alternative execution plans is performed to choose one with the least cost. Both data transmission and local processing costs are considered in Ref. 11.

DISTRIBUTED TRANSACTION MANAGEMENT

The objectives of distributed transaction management are the same as those of centralized transaction management [i.e., the guarantee of ACID properties (12–14)]:

Atomicity requires that either all or none of the transaction's operations be performed. In other words, if a transaction fails to commit, its partial results cannot remain in the database.

Consistency requires that a transaction to be correct. In other words, if a transaction is executed alone, it takes the database from one consistent state to another. When more than one transaction is executed concurrently, the database management system must ensure the consistency of the database.

Isolation requires that an incomplete transaction cannot reveal its results to other transactions before its commitment. This function can avoid the problem of cascading abort (i.e., the necessity to abort all the transactions that observed the partial results of a transaction that was later aborted).

Durability means that once a transaction has been committed, all the changes made by this transaction must not be lost even in the presence of system failures.

Two types of transactions we need to consider in a distributed database system are *local* and *global* transactions. A local transaction may access and update data in only one local database, whereas a global transaction may access and update data in several local databases. Thus, a global transaction consists of a set of *subtransactions*, each of which involves data residing on one site. A *transaction manager* at each site ensures ACID properties of local transactions as well as subtransactions at that site. For global transactions, the task is much more complicated, because several sites may be participating in execution. The concurrent global transactions must be serializable and recoverable in the distributed database system. In consequence, each subtransaction of a global transaction must be either performed in its entirety or not performed at all.

Serializability in a Distributed Database

It is well understood that the maintenance of the consistency of each single database does not guarantee the consistency of the entire distributed database. It follows, for example, from the fact that serializability of executions of the subtransactions on each single site is only a necessary (but not sufficient) condition for the serializability of the global transactions. To ensure the serializability of distributed transactions, a condition stronger than the serializability of single schedule for individual sites is required.

In the case of distributed databases, it is relatively easy to formulate a general requirement for correctness of global transactions. The behavior of a distributed database system is the same as a centralized system but with distributed resources. The execution of the distributed transactions is correct if their schedule is serializable in the whole system. The equivalent conditions are as follows:

- Each local schedule is serializable.
- The subtransactions of a global transaction must have a compatible serializable order at all participating sites.

The last condition indicates that for any two global transactions G_i and G_j, their subtransactions must be scheduled in the same order at all the sites on which these subtransactions have conflicting operations. Precisely, if G_{ik} and G_{jk} belong to G_i and G_j, respectively, and the local serializable order is G_{ik} precedes G_{jk} at site k, then all the subtransactions of G_i must precede the subtransactions of G_j at all sites where they are in conflict.

Various concurrency control algorithms such as two phase locking (2PL) (15,16) and timestamp ordering approaches (17,18) have been extended to distributed database systems. Because the transaction management in a distributed database system is implemented by several identical local transaction managers, the local transaction managers cooperate with each other for the synchronization of global transactions. If the timestamp ordering technique is used, a global timestamp is assigned to each subtransaction and the order of timestamps is used as the serialization order of global transactions. If a 2PL algorithm is used in the distributed database system, the locks of a global transaction cannot be released at all local sites until all the required locks are granted. In distributed systems, the data item might be replicated. The updates to replicas must be atomic (i.e., the replicas must be consistent at different sites). The following rules may be used to lock with n replicas:

- Writers need to lock all n replicas, readers need to lock one replica.
- Writers need to lock all m replicas ($m > n/2$), readers need to lock $n - m + 1$ replicas.
- All updates directed first to a primary copy replica (one copy has been selected as the primary copy for updates first and then the updates will be propagated to other copies).

Any one of the above rules will guarantee consistency among the duplicates.

Atomicity of Distributed Transactions

In a centralized system, transactions can either be processed successfully or be aborted with no effects left on the database in the case of failures. Normally, the failures cause loss of volatile or nonvolatile storage data. In a distributed system, however, additional types of failure may occur.

For example, network failures or communication failures may cause network partition, and the messages sent from one site may not reach the destination site. If a partial execution of a global transaction at a partitioned site existed in a network; it would not be easy to implement the atomicity of a distributed transaction. To achieve an atomic commitment of a global transaction, it must be ensured that all of its subtransactions at different sites are capable and available to commit. Thus, an agreement protocol must be used among the distributed sites. The most popular atomic commitment protocol is the two phase commitment (2PC) protocol.

In the basic 2PC, the site where a global transaction is issued serves as a *coordinator*. The participating sites that execute the subtransactions must commit or abort the transaction unanimously. The coordinator is responsible to make the final decision to terminate each subtransaction. The first phase of 2PC is to request from all *participants* the information on the execution state of subtransactions. The participants report to the coordinator, who collects the answers and makes the decision. In the second phase, that decision is sent to all participants. In detail, the 2PC protocol proceeds in two phases for a global transaction T_i (1).

Phase 1. Obtaining a Decision.

1. Coordinator asks all participants to prepare to commit transaction T_i:
   a. add [prepare T_i] record to the log
   b. send [prepare T_i] message to each participant
2. When a participant receives [prepare T_i] message it determines if it can commit the transaction:
   a. if T_i has failed locally, respond with [abort T_i]
   b. if T_i can be committed, send [ready T_i] message to the coordinator.
3. Coordinator collects responses:
   a. all respond *ready*, decision is commit
   b. at least one response is *abort*, decision is abort
   c. at least one fails to respond within time-out period, decision is abort.

Phase 2. Recording the Decision in the Database.

1. Coordinator adds a decision record ([abort T_i] or [commit T_i]) in its log.
2. Coordinator sends a message to each participant informing it of the decision (commit or abort).
3. Participant takes appropriate action locally and replies *done* to the coordinator.

The first phase is that the coordinator initiates the protocol by sending a *prepare-to-commit* request to all participating sites. The *prepare* state is recorded in the log and the coordinator is waiting for the answers. A participant will reply with a *ready-to-commit* message and record the *ready* state at the local site if it has finished the operations of the subtransaction successfully. Otherwise, an *abort* message will be sent to the coordinator and the subtransaction will be rolled back accordingly.

The second phase is that the coordinator decides whether to commit or abort the global transaction based on the answers from the participants. If all sites answered *ready-to-commit*, then the global transaction is to be committed. The final *decision-to-commit* is issued to all participants. If any site replies with an *abort* message to the coordinator, the global transaction must be aborted at all the sites. The final *decision-to-abort* is sent to all the participants who voted the *ready* message. The global transaction information can be removed from the log when the coordinator has received the *completed* message from all the participants.

The basic idea of 2PC is to make an agreement among all the participants with respect to committing or aborting all the subtransactions. The atomic property of global transaction is then preserved in a distributed environment.

The 2PC protocol is subject to the blocking problem in the presence of site or communication failures. For example, suppose that a failure occurs after a site has reported *ready-to-commit* for a transaction, and a global commitment message has not yet reached this site. This site would not be able to decide whether the transaction should be committed or aborted after the site is recovered from the failure. Three phase commitment (3PC) protocol (19) has later been introduced to avoid the blocking problem. But, 3PC is too expensive.

The 2PC protocol is used not only in distributed databases, but also in parallel databases for transactions, which contain subtransactions to be executed in different partitions of a parallel database (20).

FEDERATED DATABASE SYSTEMS

A *federated database system* (FDBS) is a collection of cooperating but autonomous database systems called *component DBSs* that are integrated to various degrees (3). The software that provides controlled and coordinated manipulation of the component DBSs is called a *federated database management system* (FDBMS). A component DBS in an FDBS can participate in more than one federation. A *multidatabase system* (MDBS) differs from an FDBS in that only a single federation schema is defined for a multidatabase. The DBMS of a component DBS, or component DBMS, can be a centralized or distributed DBMS or another FDBMS. Several significant aspects of an FDBS are as follows:

1. Local autonomy. Component DBSs are often under separate and independent control. Those who control a database are often willing to let others share the data only if they retain control. A component DBS can continue its local operations and can participate in a federation at the same time. Normally, no difference to a component DBS exists between a local applications or a global applications at federated levels.
2. Heterogeneity. Usually, component DBMSs are different; they can differ in such aspects as data models, query languages, and transaction management capabilities.
3. Pre-existing distribution. Usually, multiple component DBSs are built before an FDBS is built. Therefore, discrepancy in semantics and conflicts may exist among those component databases.

Schema Architecture and Design

Figure 2 shows a five-level schema architecture of a FDBS proposed by Sheth (3).

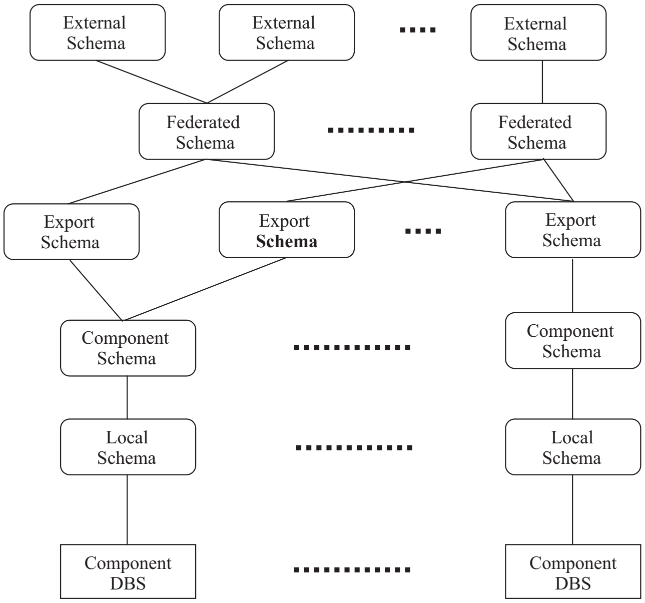


Figure 2. Five-level schema architecture of an FDBS.

- Local Schema. A local schema is the conceptual schema of a component DBS. A local schema is expressed in the native data model of the component DBMS; and hence, different local schemas may be expressed in different data models.
- Component Schema: A component schema is derived by translating local schemas into a data model called the canonical or common data model (CDM) of the FDBS. Two reasons for defining component schemas in a CDM are 1) they describe the divergent local schemas using a single representation and 2) semantics that are missing in a local schema can be added to its component schema.
- Export Schema. Not all data of a component DBS may be available to the federation and its users. An export schema represents a subset of a component schema that is available to the FDBS. It may include access control information regarding its use by specific federation users. The purpose of defining export schemas is to facilitate control and management of association autonomy.
- Federated Schema. A federated schema is an integration of multiple export schemas. A federated schema also includes the information on data distribution that is generated when integrating export schemas. Some systems use a separate schema called a distribution schema or an allocation schema to contain this information. Multiple federated schemas may exist in an FDBS, one for each class of federation users. A class of federation users is a group of users and/or applications who perform a related set of activities.
- External Schema. A subschema or a view defined over a federated schema primarily for a pragmatic reason of not having to define too many federated schemas or to tailor a federated schema for smaller groups of federation users than that of a federated schema.

As component databases normally pre-exist in a FDBS, we can take a bottom-up design approach for federated databases. This approach is in contrast to the top-down design approach discussed in a previous section. The major tasks of the bottom-up design are schema translation and schema integration.

Schema Translation. As local schemas of different component databases may be defined in different data models, the specification of a CDM for defining federated schemas is required. Relational data model and object-oriented data model are often chosen as a CDM. Mapping rules must be studied between data models (e.g., relational model, DBTG or network model, hierarchical model, object-oriented model, and more recently XML data model).

Schema Integration. After schema translation, component schemas are generated for component databases. After that, export schemas are generated from component schemas for integration to different federated schemas. Four steps can be followed for schema integration.

- Pre-Integration. Pre-integration is required to establish the *rules* of the integration process before actual integration occurs. For example, candidate keys in each schema must be identified; equivalent domains of attributes must be described in terms of mappings from one representation to another.
- Comparison. During this phase, both the naming and the structural conflicts are identified. Naming conflicts include *synonym* (two identical entities or attributes with different names) and *homonym* (two different entities or attributes with the same name). Structural

conflicts include 1) *type conflicts*: the same object is represented by an attribute in one schema and by an entity in another, 2) *dependency conflicts*: different relationship types are used to represent the same thing in different schemas (1:m vs m:n), 3) *key conflicts*: different candidate keys are available and different primary keys are selected in different schemas, and 4) *behavioral conflicts* are implied by the modeling mechanism (e.g., deletion of the last employee causes the dissolution of the department).

- Conformation. Conformation is the resolution of the conflicts that are determined at the comparison phase.
- Merging and Restructuring. All schemas must be merged into a single database schema and then restructured to create the best federated schema.

Global Query Processing and Optimization

In a loosely coupled FDBS, the FDBMS can support little or no query optimization. In a tightly coupled FDBS, the FDBMS can perform extensive query optimization. Query processing involves converting a query against a federated schema into several queries against the export schemas and executing these queries. Query processing in an FDBMS is similar to that in a distributed DBMS. In an FDBMS, however, several additional complexities may be introduced because of heterogeneity and autonomy. The cost of performing an operation may be different in different component DBSs. The component DBMSs may differ in their abilities to perform local query optimizations. The system and database operations provided by each of the component DBMSs and the FDBMSs may be different. Landers and Rosenberg (21) discuss optimization problems and solutions adopted for some of the above issues in Multibase.

Global Transaction Management

Supporting global transaction management in an environment with multiple heterogeneous and autonomous component DBSs is very difficult. The challenge is to permit concurrently global updates to the underlying databases without violating their autonomy. Two types of transactions to be managed exist: *global transactions* submitted to the FDBMS by federation users and *local transactions* submitted directly to a component DBMS by local users. The basic problem in supporting global concurrency control is that the FDBMS does not know about local transactions because a component DBMS is autonomous. That is, local wait-for relationships are known only to the transaction manager of the component DBMS. Without knowledge about local as well as global transactions, it is highly unlikely that efficient global concurrency control can be provided. Because of the existence of local transactions, it is very difficult to recognize when the execution order differs from the serialization order at any site (22). Additional complications occur when different component DBMSs and the FDBMS support different concurrency control mechanisms (23). Georgakopoulos et al. (24) proposed to incorporate additional data manipulation operations on *tickets* in the subtransactions of each global transaction and show that if these operations create direct conflicts between substransactions at each participating component DBS, indirect conflicts can be resolved even if the FDBS is not aware of their existence. However, all the published solutions often make unrealistic and pessimistic assumptions, or support a low level of concurrency or sacrifice autonomy to obtain higher concurrency. It is unlikely that a theoretically elegant solution exists that provides conflict serializability without sacrificing performance (i.e., concurrency and/or response time) and availability.

Work on weaker consistency criteria (25) and advanced transaction models (26) provide techniques to specify and to execute transactions that provide ACID properties selectively. A concept of *S-Transactions* (27) is proposed for semantic transactions suited for a banking environment that consists of a network of highly autonomous systems. It may be desirable to devise solutions that do not meet the conflict serializability criteria but that are practical and meet a desired level of consistency. Du and Elmagarmid (22) propose a weaker consistency criterion called *Quasi Serializability* that works if no value dependencies (e.g., referential integrity constraints) exist across databases. Garcia-Molina and Salem (28) propose a concept of *Sagas* that provides semantic atomicity but does not serialize execution of global transactions.

THE WEB AND DISTRIBUTED DATABASES

The last decade has seen the emergence of the Web as the central forum for data storage and exchange. More recently, XML has been proposed as a standard for data exchange and storage. Compared with the relational data model, the de facto standard for database systems and its structural primitives for building trees of elements with attributes offer much more flexibility in data organization and format. The Web and XML provide many avenues for database researchers. The Web bears a similarity to a bottom-up designed federated database in terms of intergrating structured data sources from different websites; however, the Web is much more loosely coupled. In the following, we address two areas that are relevant to distributed databases.

Information Integration on the Web

Data integration is a pervasive challenge faced in applications that need to query across multiple autonomous and heterogeneous data sources (29). Data integration is crucial in large enterprises that own a multitude of data sources. Integrating information from data resources over the Internet requires creating some form of integrated view to allow for distributed querying. The context of the Internet raises several issues for information integration that are far more difficult than those of multidatabase systems (3). First, the number of data sources may be very high, which makes view integration and conflict resolution a problem. Second, the space of data resources is very dynamic, so adding or dropping a data source should be done with minimal impact on the integrated view. Third, the data sources may have different computing capabilities, which range from full-featured

DBMS to simple files. This feature is unlike multidatabase systems, which assume data sources with an SQL-like interface. Finally, data sources may be unstructured or semi-structured, which provides virtually no information for view integration. To address these problems, the database research community has revisited the multidatabase architecture (i.e., the architecture of a FDBS with a single federation schema) with data source wrappers and mediators. For each data source, a wrapper exports some information about its source schema, data, and query capabilities (30). For the whole integration system, a mediator centralizes the information provided by the wrappers in a unified global view of all available data, decomposes global queries into subqueries executable by wrappers on data sources, and gathers the partial results and computes the answer to the global queries. This wrapper–mediator architecture differs from a data warehouse in that integrated global view is not materialized.

Two basic approaches exist in data integration, GAV and LAV (31–33). These two approaches have also been used in the context of data integration on the Web. GAV (Global As View) defines a global schema as a view over a set of source schemas, whereas LAV (Local As View) defines source schemas as views over the global schema. GAV has been used in FDBSs and multidatabase systems, in which the quality depends on how well we have compiled the sources into the global schema through mapping. Whenever a source changes or a new source is added, the global schema must be reconsidered. Query processing can be based on some sort of rewriting. Each element in the user's query corresponds to a substitution rule just as each element in the global schema corresponds to a query over the source. Query processing is simply expanding the subgoals of the user's query according to the rule specified in the mediator, and thus the resulting query is likely to be equivalent. LAV has high modularity and reusability. Once the global schema is well designed, changes on a source only affect the definition of the source. The quality depends on how well we have characterized the sources. In LAV systems, queries undergo a more radical process of rewriting because a mediator does not exist. The integration system must execute a search over the space of possible queries to find the best rewrite. The resulting rewrite may not be an equivalent query but maximally contained, and the resulting tuples may be incomplete.

Recently, *Semantic Web* (34) has attracted great attentions from both research communities and standard organizations. Semantic Web is supposed to be an extension of the Web where the semantics of data is available to and processable by machines. At the core of this new technology are the languages that are used to describe the semantics of XML documents and ontology, such as RDF, DAML+OIL, and OWL. Ontology can be used to define global schemas, which makes information integration on the Web easy.

Publishing Relational Data on the Web

Although XML is emerging as the universal format to publish and to exchange data on the Web, most business data is still stored and maintained in relational database systems. As a result, an increasing need exists to publish relational data efficiently as XML documents for Internet-based applications. One approach to publish relational data is to create XML views of the underlying relational data. Through the XML views, users may access the relational databases as though they were accessing XML documents. Once XML views are created over a relational database, queries in an XML query language like XML-QL or XQuery can be issued against these XML views for the purpose of accessing relational databases. SilkRoute (35) is one of the systems that takes this approach. In SilkRoute, XML views of a relational database are defined using a relational to XML transformation language called RXL, and then XML-QL queries are issued against these views. The queries and views are combined together by a query composer and the combined RXL queries are then translated into the 1corresponding SQL queries. XPERANTO (36) takes a similar approach which uses XQuery for user queries. DTD directed publishing is introduced in Ref. 37 where an attribute translation grammar (ATG) is designed to creating XML views of relational databases. Another approach (38) to publish relational data is to provide virtual XML documents for relational data via an XML schema that is transformed from the underlying relational database schema such that users can access the relational database through the XML schema. In this approach, the process of XML schema generation preserves integrity constraints of the underlying relational schema, which makes a difference compared with the view approach taken by SilkRoute.

BIBLIOGRAPHY

1. S. Ceri and G. Pelagatti, *Distributed Databases - Principles and Systems*, New york: McGraw-Hill, 1984.
2. M. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, 2nd ed., EngleWood Cliffs, NJ: Prentice-Hall, 1999.
3. A. Sheth and J. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases, *ACM Computing Surveys*, **22** (3): 183–236, 1990.
4. C. Date, *An Introduction to Database Systems*, Vol. **1**, 4th ed., Reading, MA: Addison-Wesley, 1986.
5. T. Connolly and C. Begg, *Database Systems—A Practical Approach to Design, Implementation, and Management*, 3rd ed., Reading, MA: Addison Wesley, 2002.
6. C. T. Yu, et al., File allocation in distributed databases with interaction between files, *Proc. 9th Int. Conf. Very Large Data Bases*, 1983, pp. 248–259.
7. C. Yu and C. Chang, Distributed query processing, *ACM Computing Surveys*, **16** (4): 399–433, 1984.
8. J. Ullman, *Principles of Database Systems*, 2nd ed., Rockville, MD: Computer Science Press, 1982.
9. P. A. Bernstein and D. W. Chiu, Using semi-joins to solve relational queries, *J. A.C.M.*, **28** (1): 25–40, 1981.
10. P. A. Bernstein, et al., Query processing in a system for distributed databases (SDD-1), *ACM Trans. Database Sys.*, **6** (4): 602–625, 1981.
11. P. G. Selinger and M. E. Adiba, Access path selection in distributed database management systems, *Proc. 1st Int. Conf. Databases*, 1980, pp. 204–215.
12. T. Härder and A. Reuter, Principles of transaction-oriented database recovery, *ACM Comput. Surv.*, **15** (4): 287–317, 1983.

13. J. Gray, The transaction concept: virtues and limitations, *Proc. 7th Int. Conf. Very Large Data Bases*, 1981, pp. 144–154.
14. Y. Zhang and X. Jia, Transaction processing, In J. Webster (ed.), *Wiley's Encyclopedia of Electrical and Electronics Engineering*, vol. 22, 1999, pp. 298–311.
15. K. P. Eswaran, et al., The notions of consistency and predicate locks in a database system, *Commun. ACM*, **19** (11): 624–633, 1976.
16. J. Gray, Notes on data base operating systems, *Lect. Notes Comput. Sci.*, **6**: 393–481, 1978.
17. P. A. Bernstein and N. Goodman, Timestamp-based algorithms for concurrency control in distributed database systems, *Proc. 7th Int. Conf. Very Large Data Bases*, 1980, pp. 285–300.
18. L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Commun. ACM*, **21** (7): 558–565, 1978.
19. C. Date, *An Introduction to Database Systems*, Vol. 2, 2nd ed., Reading, MA: Addison-Wesley, 1982.
20. C. Liu, et al., Capturing global transactions from multiple recovery log files in a partitioned database system, *Proc. 29th Int. Conf. Very Large Data Bases*, 2003, pp. 987–996.
21. T. Landers and R. Rosenberg, An overview of multibase, in H.-J. Schneider (ed.), *Distributed Databases*. Amsterdam: North-Holland, 1982, pp. 153–184.
22. W. Du and A. Elmagarmid, Quasi serializability: A correctness criterion for global concurrency control in interbase, *Proc. 15th Int. Conf. Very Large Data Bases*, 1989, pp. 347–355.
23. V. D. Gligor and R. Popescu-Zeletin, Transaction management in distributed heterogeneous database management systems, *Inf. Syst.*, **11** (4): 287–297, 1986.
24. D. Georgakopoulos, M. Rusinliewicz, and A. Sheth, Using tickets to enforce the serializability of multidatabase transactions, *TKDE*, **6** (1): 166–180, 1994.
25. Y. Breitbart, H. Garcia-Molina, and A. Silberschatz, Overview of multidatabase transaction management, *VLDB J.*, **2**: 181–239, 1992.
26. A. Elmagarmid (ed.), *Database Transaction Models For Advanced Applications*. San Mateo, CA: Morgan Kaufmann, 1992.
27. J. Veijalainen, F. Eliassen, and B. Holtkamp, The S-transaction model, in A. Elmagarmid (ed.), *Database Transaction Models For Advanced Applications*. San Mateo, CA: Morgan Kaufmann, 1992, pp. 467–513.
28. H. Garcia-Molina and K. Salem, Sagas, *Proc. 1987 ACM SIGMOD Int. Conf. Management of Data*, 1987, pp. 249–259.
29. A. Halevy, A. Rajaraman, and J. Ordlille, Data integration: The teenage years, *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 9–16.
30. S. Cluet, et al., Your mediators need data conversion, *Proc. 1997 ACM SIGMOD Int. Conf. Management of Data*, 1997, pp. 177–188.
31. A. Halevy, Answering queries using views: A survey, *VLDB J.*, **10** (4): 270–294, 2001.
32. J. D. Ullman, Information integration using logical views, *Theor. Comput. Sci.*, **239** (2): 189–210, 2000.
33. M. Lenzerini, Data integration is harder than you thought, *Proc. CoopIS*, 2001, pp. 22–26.
34. T. Berners-Lee, J. Hendler, and O. Lassila, The Semantic Web, *Scientific American*, **May**: 2001.
35. M. Fernandez, W. Tan, and D. Suciu, SilkRoute: Trading between relations and XML, *Proc. WWW*, 2000, pp. 723–725.
36. M. Carey, et al., XPERANTO: Middleware for publishing object-relational data as XML Documents, *Proc. 26th Int. Conf. Very Large Data Bases*, 2000, pp. 646–648.
37. M. Benedikt, et al., DTD-directed publishing with attribute translation grammars, *Proc. 28th Int. Conf. Very Large Data Bases*, 2002, pp. 838–849.
38. C. Liu, M. Vincent, and J. Liu, Constraint preserving transformation from relational schema to XML schema, *World Wide Web J.*, **9** (1): 93–110, 2006.

CHENGFEI LIU
Swinburne University of
Technology
Melbourne, Australia

YANCHUN ZHANG
Victoria University
of Technology
Melbourne, Australia

DISTRIBUTED FILE SYSTEMS

INTRODUCTION

A distributed file system, also known as a network file system, is a means for to access data transparently across a network. A user should not have to know whether a file actually resides locally or on a remote server. In the typical distributed file system architecture in Fig. 1(a), a client uses the same standard file system calls that are translated into network requests at runtime. If the access to remote data is not transparent, a client might have to retrieve explicitly a file locally, make modifications, and then move explicitly the file back to the server as in Fig. 1(b).

Distributed file systems are a broad topic in computer science. We have organized our discussion into four distinct areas. We begin by focusing on high availability techniques that improve uptime and provide support for mobile or disconnected operation. Then, we closely examine closely several protocol standards for compatibility and implementation-independent optimizations. The next section describes several distributed file systems that are tuned for high-performance computing (HPC) applications. Finally, we discuss several distributed file systems that have application-specific features.

HIGH AVAILABILITY

Users who need reliable and distributed access to files across various networks (i.e., university campuses and large companies) use “highly available” distributed file systems. Distributed file systems with high availability can be characterized generally by a fair degree of fault tolerance: addressing client, network, and server failures. Files must be accessible easily to users from multiple locations, and although concurrent, or near concurrent, writing to shared files is rare, reading shared files is vital for collaboration. Data availability is critical to virtually every organization. Typical techniques include both replication and logs. Industry demand for highly available distributed file systems has resulted in a plethora of proprietary solutions.

The serverless approach to highly available storage bears a striking resemblance to peer-to-peer systems. Although high availability may be the most high profile goal, any production read-write file system must address security.

The Network File System (NFS) and Common Internet File System (CIFS) are described in the next section from a design and protocol perspective. The file systems themselves do not address high availability explicitly. Availability is left to proficient system administration and fault tolerant storage techniques.

In this section, we focus on the Andrew file system (AFS), Coda, and a few serverless file systems.

Andrew File System

The Andrew File System (AFS) (1,2) was started at Carnegie Mellon University as part of the Andrew distributed computing environment. It has gone through three iterations: AFS-1, AFS-2, and AFS-3. The primary focus of AFS is scalability: namely across many client workstations in a large institution. In AFS, a pool of trusted file servers are known collectively as *Vice*. Clients, who each run a *Venus* process, are untrusted and must have local disks. The benefit of this security model is that administrators need to worry about only a small percentage of the entire system. Users are aware of only directory structure, and not of physical location of files. This transparency is not only important to interactive users, but to user applications as well.

AFS-1 was a pilot vehicle for the basic AFS architecture, and it was used for only about a year. Local caching is done at the file granularity, and client-side cache coherence is handled in a very simplistic but inefficient manner. Before any locally cached version of a file can be used, the client first verifies its validity with Vice. Although cached copies of files can be read and written, updates to cached directories go to the servers directly.

AFS-2 is built on the lessons learned with the deployment of AFS-1 as well as additional performance evaluations. Notable changes were made to cache management, the global name space, and the server design.

Effectively, caching entire files is a prefetching technique, and it is proved beneficial, if not vital, to performance. AFS-2 introduced the *callback*, an explicit agreement made between a client and server when a client first caches some given data. A callback allows servers to notify actively the appropriate clients when their cached data becomes invalid. Until then, a client will assume that the particular cached data is still safe. In AFS-2, callbacks reduce the validation traffic significantly. Updated data is passed to the servers on file close.

AFS-2 also adopted the notion of *volumes*. Volumes consist of a set of files that form partial subtrees in Vice. Collectively, all the volumes in Vice form the entire file system name space. The typical division of volumes is about one volume per user. Volumes can be archived easily and migrated to different servers.

AFS-3 is focused on administrative improvements, but it is worth noting that Venus was moved into kernel space, which allows it to cache data 64KB at a time. Caching data at a finer grain than entire files improves latency and allows operation on very large files that do not otherwise fit on the client disk. The task of administering AFS-3 is decentralized using *cells*. Cells are composed of servers, clients, system administrators, and users. Multiple cells can coordinate to aid collaboration between sites.

AFS-3 was adopted for commercialization by Transarc, and they were bought subsequently by IBM. IBM then branched and opened the code as OpenAFS. The AFS-2 design became the starting point for the Coda file system.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

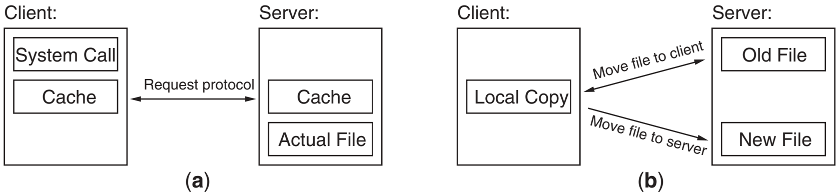


Figure 1. (a) Typical distributed file system access. (b) Explicit remote file access (for example, through FTP).

Coda

Coda (3,4) is based on the AFS-2 design with several major differences. This close relationship is clear in Table 1. AFS-2 was subject to debilitating failures if any server crashed. Coda addresses server failure with server replication. Disconnected operation in Coda is treated as a temporary state, but it can be tolerated for an extended period of time.

In Coda, a volume is replicated on multiple servers that compose a *volume storage group* (VSG). Reads are done from a single replica, and updates are performed on all copies. The subset of servers in the VSG that is accessible makes up the accessible VSG (AVSG). On a cache miss, all the servers in the AVSG are contacted, and the most recent data from one of the servers is cached. Any servers with stale data are also notified.

Two contexts exist for disconnections: brief temporary failures and extended disconnections. Because Coda already caches entire files on the client, disconnected operations are relatively easy: One just needs to ensure that the appropriate files are cached before they are remove from the network. During disconnected operations, any cache miss cannot be resolved. For brief, unintentional disconnections, the regular LRU cache policy may suffice and cache misses may be avoided. Extended disconnections may be the result of more severe failures in either the network or the servers, or it may also be the result of a mobile device that is disconnected intentionally. For longer disconnections, the LRU policy will not likely be sufficient to avoid cache misses. To this end, Coda provides the user with a mechanism to prioritize files and directories for caching.

Disconnected operation in a distributed system entails implicitly some mechanism for reconnection. As soon as the AVSG is available again, updates are pushed upstream. If no conflicts exist because of modifications by other clients, this process is completely transparent. Any files and directories without conflicts are simply updated, and Coda provides tools for users to resolve and to update conflicts.

Intermezzo (5) later sought to replicate the benefits and useful features of Coda with a simpler ground-up implementation.

Serverless File Systems

Serverless file systems make no distinction between client and server nodes. Separate client and server processes may exist, but no real technical restrictions exist on what machines they might run on. Because of this, the security and administrative models are different from those of AFS and its derivatives. Some common trends in serverless file systems are leveraging routing/storage systems as well as using versioning or logs to store immutable file modifications. This basic design is illustrated in Fig. 2. The separate routing and storage systems can be optimized independently to exploit locality, load balancing, and other low-level aspects involved with storage. Salient characteristics of several serverless file systems are compiled in Table 2.

OceanStore (6,7) is an ambitious wide-area storage system built on fundamentally untrusted servers. The OceanStore prototype, Pond features "location-independent routing, Byzantine update commitment, push-based update of cached copies though an overlay multicast network, and continuous archiving to erasure-coded form (8)." Nodes in OceanStore are not symmetric, but they are tiered. The unit of storage in OceanStore is the *data object*; this corresponds to a file in Pond. Data objects are versioned completely, so data is never modified in-place or deleted. The stream of versions for a particular object is called an *active global unique identifier* and each version is called a *version global unique identifier* (VGUID). Only the "delta blocks," or changes, of subsequent VGUIDs are stored. Versioning allows atomic updates, simpler replication, and easy rollback to earlier versions of data. The *inner ring*, a small group of coordinated servers, manages updates to data objects and has the final say on the *primary replica* for an object. Although the inner ring need not be trusted explicitly and it is designed to be fault tolerant to a

Table 1. A comparison of AFS-3 and Coda

Category	AFS-3	Coda
Dedicated servers	Yes	Yes
Consistency semantics	Session	Transactional
Consistency enforcement	Callbacks	Callbacks
Caching	Client disk chunked	Client disk file-level
Fault tolerance	Local disk cache	Local disk cache /Hoarding
Replication	Manual	ROWA volume replication
Security	Needham-Schroeder	Needham-Schroeder

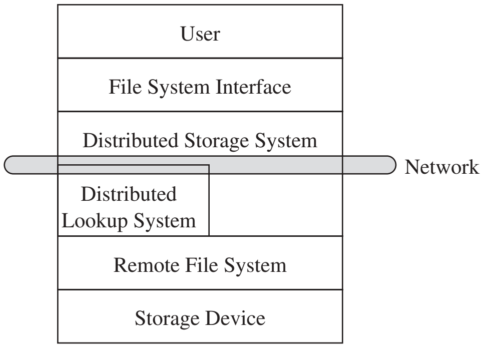


Figure 2. Typically, are serverless file systems composed of a file system interface, a distributed storage system, and a distributed lookup system.

degree, it should be composed of fairly reliable, well-connected servers. The underlying blocks are stored as cryptographically-secure hashes. Object location and routing is handled by a scalable overlay network called Tapestry (9). Unlike distributed hash tables, Tapestry can store data blocks anywhere because hosts publish the GUIDs of their resources. With this flexibility, Tapestry can shuffle data for better locality and lookup performance. OceanStore uses replication primarily for performance purposes. For archiving, OceanStore uses erasure codes to store blocks. New blocks are erasure-coded and distributed across OceanStore servers. Because reconstructing data from erasure-coded blocks can be expensive, frequently used whole-blocks are cached. User applications can access secondary replicas of data, but updates are also sent directly to the primary replica as well as a few random secondary replicas. The secondary replicas then propagate the updates amongst themselves epidemically. When the inner ring commits the update, the commit signal is multicasted to the appropriate secondary replicas.

The Cooperative File System (CFS) (10) is a peer-based read only file system built on the Chord (11) distributed hash system and Self-certifying File System (SFS). File blocks are distributed and balanced across clients using Chord to maintain their locations. For increased availability, CFS uses block-level replication across nodes.

Ivy (12) was designed subsequently with read and write capabilities. Writes are log based, and each client updates its own log records. The log records themselves are distributed using Chord. File modifications become visible at least upon closing the file. Reads are done from all clients' logs. To aid performance, clients cache recent copies of the file system state. Essentially, the clients' combined logs for a particular file make up the file. A special *view* block maintains the appropriate log-heads globally for each client that is part of the entire file system. The view itself cannot change once the participating nodes have been established for an Ivy file system. To add or remove a node, the nodes must coordinate explicitly to create a new Ivy file system. If nodes become partitioned, operations continue without any consistency guarantees. Ivy relies on the replication mechanisms within Chord for availability. Upon reconnection, Ivy retains all updates because of its intrinsic log design, but the user or application must use the provided tools to detect and resolve any atomicity issues.

Like Ivy, Frangipani (13) is also built over a distributed storage system, in this case, Petal (14). Also similar to Ivy, Frangipani derives features such as scalability and fault tolerance from Petal. Frangipani itself is intended to run within a single administrative umbrella; consequently, implicit trust exists between nodes. The Petal interface provides a distributed virtual disk image that is actually made up of some number of Petal servers, each of which may contain multiple disks. Petal allows for easy addition and removal/crashes of Petal servers, which makes these events transparent to Frangipani. Petal servers need not be dedicated machines, and they can be run on users' machines. Although Petal provides a disk-like interface, Frangipani hides Petal with a more user-friendly file system interface. User programs access Frangipani files through a virtual file system mechanism and a locally running Frangipani file server module. Alternatively, a Frangipani server process can be described as a local daemon on the "client" machine that only provides services to the local user. The "server," in the sense that it is across the network, is the Petal virtual disk. Frangipani accesses file data directly on Petal, but each Frangipani server maintains its own log of pending metadata changes also on the Petal disk. If a Frangipani server crashes, its log can be used by another Frangipani server to recover. Frangipani itself, however, does not log file updates, only metadata updates. By design, Frangipani servers do not communicate directly with each other, which makes additions, removals, and tolerance more simple.

Table 2. A comparison of serverless high-availability file systems

Category	OceanStore	Ivy	Frangipani	Pastis
Dedicated servers	No	No	No	No
Consistency semantics	POSIX	Close-to-open	Metadata	Close-to-open
Consistency enforcement	Single *primary replica*/Inner ring	Versioning	Locking/leases	Global timestamp
Caching	Client disk (Shared) block-level	Client memory Block-level	Client memory Sector-aligned chunks	Client disk file-level
Fault tolerance	Erasure-coded versioning	DHash - temporary partitioning	Automatic recovery	Pastry - node addition & failure
Replication	Two-level block	DHash	Petal option	*k* copies
Security	Cryptographically-secure hash	Self-certifying	Trusted hosts	Smartcard

The Pastis (15) file system is layered very similarly to Ivy. It is built on the PAST (16) peer-to-peer storage service, which in turn uses Pastry (17), a fault tolerant and self-organizing routing system based on a distributed hash table. PAST itself ensures high availability through replication. Pastis is structured internally similar to traditional file systems, and it uses its own *inodes* stored in PAST. Like Frangipani, Pastis operates on a block abstraction. File updates in Pastis are also nondestructive because PAST blocks are immutable.

PROTOCOL STANDARDS AND RELATED OPTIMIZATIONS

Several distributed file system protocols have been established to ensure vendor and enterprise interoperability. A partial list of distributed file system protocols includes the Common Internet File System, the Apple Filing Protocol, the NetWare Core Protocol (NCP), and the Network File System (NFS). We direct our discussion towards the two most pervasive protocols in use today: NFS and CIFS.

Network File System

NFS was developed by Sun Microsystems to provide transparent file access in a networked, distributed environment. Since 1989, NFS has been an internet engineering task force standard protocol (18–22). The most popular revision currently in use is NFS version 3. It is implemented on a variety of operating systems and provides file sharing among a collection of heterogeneous computers. To improve performance in the broader Internet environment, a major revision of NFS, version 4, has been defined to integrate several new or improved features, such as file locking, security, operation coalescing, and file delegation (23–25).

NFS consists of two components, namely client-side and server-side systems. Figure 3 demonstrates how NFS components interact. The client-side systems process local user requests for files stored at a remote server. All client–server communication is handled with remote procedure calls (RPCs). RPCs allow programs on a local machine to initiate procedures on remote machines. When a local process calls a procedure on a remote machine, the calling process is suspended, and instructions to run the procedure are sent over the network to the remote machine where the procedure is executed. The execution results are transported back over the network to the caller process. The client-side system uses RPCs to transmit user requests to remote NFS servers. The server is responsible to carry out the requested file operations and to send the results back to the client to conclude the RPC call. NFS version 4 introduces compound RPC procedures that enable the encapsulation of related operations into a single RPC, which creates new opportunities for better I/O (input-output) performance.

File System Model. In the NFS model, files and directories are organized as a hierarchical tree graph in which internal nodes and leaves represent directories and files, respectively. An NFS server makes a local directory available to clients by *exporting* that directory. Directories from different locally mounted file systems can be exported. An NFS client mounts the exported directories to its local file system so that user processes on the client machine can access the NFS mounted directories as if they are part of the local file system. To access a file, a client must first look up the filename and obtain the associated file handle from the server. A file handle is unique to all file systems exported by the same server. Each file contains the attributes *fsid* and *fileid* to identify a file system uniquely on a server and the file/directory within that file system. Other attributes include permission modes, owner ID, group ID, file size, last access time, and last modification time, last metadata modification time.

Client Caching and File Locking. In NFS version 3, servers are not required to preserve any client file access state. The stateless approach eliminates the need to recover client state after a server crash. An NFS version 3 server does not need to record which clients have open files. When a client accesses a file, the RPC request provides all the necessary information such as file ID and the current offset for the server to function correctly. This stateless approach has been abandoned in version 4 to adapt to the modern wide-area network environment. In particular, the new client-side caching and file locking protocols enable more effective use of cached data and efficient cache consistency control. Although client-side file caching is left out of the NFS version 3 protocol, many implementations make extensive use of caching to improve performance. Because version 3 is stateless, caching is performed independently on the clients, and servers retain no clients' caching state. Data cached on clients can be stale without the server knowing. Some implementations allow cached data to be stale for up to 30 seconds. Cache coherence is left to application developers to enforce. The most common approach uses a separate lock manager, such as the network lock manager protocol (NLM), to provide advisory locks. Unlike mandatory locks, an advisory lock does not block other applica-

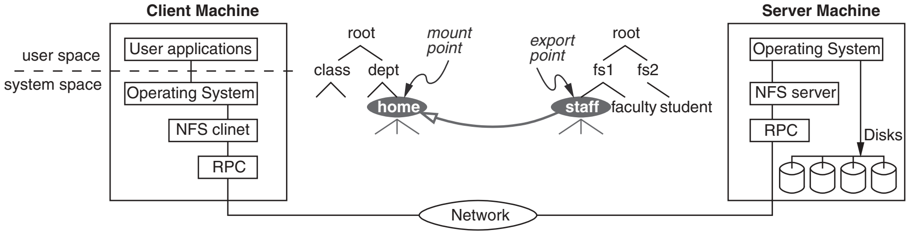


Figure 3. An illustration of the general NFS components and their interaction.

tions forcibly from accessing a locked file region. Advisory locks are only useful between cooperating processes.

NFS version 4 integrates a caching protocol and supports weak coherence. Because data can be cached in client memory and/or a server's local disk cache, the new cache protocol requires that dirty data is flushed to the server when the file is closed. The cached file data can remain in the client's memory after the close, but it must be revalidated if the file is opened again by any process on the particular machine. This close-to-open consistency is sufficient for many applications and users. A new open delegation protocol is designed to address the common situation where a file is accessed by a single client. *Delegations* allow the server to shift responsibility for a file's opens, closes, and locking operations to a client. This action eliminates the server validation costs for operations on a file from different processes that reside on the same client. The delegation state of a file is recorded on the server. When a process on a different client machine requests access rights to the same file, the server must either deny the request or recall the delegation. The revocation is accomplished with an RPC callback to the client. Callbacks are another difference between version 4 and version 3 in that RPCs are only initiated by clients in version 3. The NFS version 4 locking protocol is similar to NLM, but it introduces leases for lock management. During a lease time interval, the server denies the lock requests from other applications. To prevent the removal of a granted lock, a client must renew its lease before it expires.

Fault Tolerance. In NFS version 3, recovering from a server crash is very simple because no state exists to lose. A client is not aware of a server crash and will retry its request until the server responds. In version 4, however, it is essential to recover the state stored at the server after a system reboot because clients rely on the stateful locking protocol to access safely cached data locally. A grace period equal in duration to the lease period is executed at the server after reboot to allow clients to reclaim locks. During the grace period, the server must reject read, write, and nonreclaiming locking requests. Lock recovery from a client crash is simpler. Because a lock is leased with a time constraint, the server removes the lock when it is expired. After reboot, a client needs to request the lock again. Although important, lock recovery is different from data recovery. If a file has been delegated to a client and that client crashes with dirty data, the data is lost.

NFS version 4 handles RPC recovery from a network partition by introducing a duplicate request cache at the server. A client inserts a unique transaction ID in each RPC request and the server caches the ID to identify duplicate requests from a client during retry requests after a time out. The results of a file operation are also stored in the cache in case the server response was lost. Thus, the server can retrieve the cached results for retransmission without duplicating the requested actions.

Security. NFS version 3 only covers the user authentication and file access permission check. Because NFS is built on top of the RPC protocol, authentication is established on two RPC authentication parameters: a credential and a verifier. If an NFS implementation chooses not to implement authentication, these two parameters are ignored. The protocol defines three types of authentication, and a server may support several different flavors of authentication at once. The first is UNIX-style authentication in which a client passes the user ID, group ID, and groups to the server and the server checks the permission rights for file access. This method relies simply on the security at the client machine, because it assumes all users have passed the client security check. The second uses DES-encrypted host names and session keys exchanged between clients and servers via a public key scheme. The third also uses the DES-encrypted method, but it functions instead with Kerberos secret keys.

One of the NFS version 4 goals is to use a strong security protocol for a wide-area network environment. It supports not only authentication, but also message confidentiality through cryptography. NFS version 4 employs the RPCSEC_GSS security framework, which is based on the Generic Security Service API. This framework allows for the use of various security mechanisms at the RPC layer. RPCSEC_GSS can perform integrity checksums and encrypt the entire RPC request and response. NFS version 4 also requires RPCSEC_GSS to support Kerberos version 5 and LIPKEY public-key mechanisms.

Common Internet File Systems

Since the 1990s, use of the Internet and the World Wide Web can be characterized primarily by read-only access. The most popular examples are the web browsing in HTTP and document transfer in FTP. As the Internet continues to increase in bandwidth and in availability, the demand to share files with both read/write permissions increases. The Common Internet File System (CIFS) proposed by Microsoft, defines a distributed file system protocol to enable document sharing over a wide-area network (26). Although CIFS is based on the file system developed for Windows operating systems, the protocol is platform-independent. CIFS is derived from the standard server message block (SMB) protocol (27), an Open Group standard for personal computers and UNIX interoperability since 1992.

CIFS uses TCP/IP for client–server communication and the Internet domain name service (DNS) to resolve server IP addresses. A uniform resource locator address is used to identify a file at a remote server. Clients parse the URL character string to separate the server host name and the file location within that server. The SMB message format is used to communicate between clients and servers. An SMB message header contains the command code, error code, directory ID, caller process ID, user ID, command parameters, and data buffer.

Security. The CIFS protocol requires server authentication for users before file accesses are allowed, and each server authenticates its own users. A client system must send authentication information to the server to gain access to its resources. A CIFS server keeps an encrypted form of a client's password using DES encryption in block mode. Two methods are defined and can be selected by the server for security: share level and user level.

Table 3. A comparison of NFS and CIFS file systems

Category	NFS v3	NFS v4	CIFS
Dedicated servers	Yes	Yes	Yes
Consistency semantics	Close-to-open	Close-to-open	POSIX
Consistency enforcement	NLM	Leased lock protocol	Opportunistic lock protocol
Caching	—	Client memory	Client memory
Fault tolerance	Stateless servers	Grace period at reboot	Grace period at reboot
Replication	—	Alternative locations	—
Security	RPC authentication	RPCSEC_GSS	DES encryption

At the share level, an optional password may be required to gain access to an available resource at the server. To access the resource, a user must know the name of the server, the location of the resource on that server, and the password. Share level security servers may use different passwords for different levels of access.

A user-level server requires clients to provide a user name and a corresponding user password to gain access to the resource. Hence, different levels of access for the same resource can be set for different users. When a client's authentication is validated, the server will generate and return an identifier to represent that authenticated instance to the client in the user ID field of the response SMB message. This user ID must be included in all additional requests made on behalf of the user from that client. In contrast, a share level server does not set the user ID field in the returning SMB message.

Client Caching and File Locking. A CIFS implementation is expected to use client-side file caching to enhance network performance. The protocol supports both read-ahead and write-behind file caching. Three types of opportunistic locks are defined for cache coherence control. An exclusive lock allows a client to open a file for exclusive access, a batch lock allows a client to keep a file open on the server even if the local user on the client machine has closed the file; and a level II lock indicates that there are multiple readers of a file and no writers. When a client opens a file, it makes a request to the server for a particular type of lock on the file. The response from the server indicates the type of lock granted to the client. The client uses the granted lock type to adjust its caching policy.

An exclusive lock is intended for single client access to a file. It provides optimized file access by allowing the client to work on a local copy of the file. When a second client requests to open the same file, the server will break the lock granted to the first client. In breaking an exclusive lock, the former lock possessor must flush its dirty data to the server and purge read-ahead data.

Batch locks are useful particularly in a slow network environment. For a sequence of commands that involve repeated open and close operations to the same file, batch locks allow the client to skip the extraneous open and close requests. If the server receives either a rename or a delete request for the file that has a batch lock, it must inform the client who has possession of the lock that it will be broken. The client can then switch to a mode where the file will be opened and closed. When a batch lock is broken, the client must flush its dirty data and synchronize with the server. Most of the time, this process involves closing the file. Once the file is closed, the open request from the initiating client may be completed.

Level II locks are used to protect shared files for read-only operations even though the files are opened in read-write access mode. Multiple clients can be granted level II locks to the same file if no client writes to the file. When a client holds an exclusive lock on a file and another client opens subsequently the same file, the exclusive lock held by the first client is broken and downgraded to a level II lock. After the first client synchronizes its cached data with the server, a level II lock is granted to the second client. The level II lock may be broken if any of the clients write to the file. Once the level II lock is broken, all file requests must be executed on the server across the network.

CIFS opportunistic locks are somewhat similar to the NFS file delegation. NFS delegations differ from opportunistic locks in that a delegation is initiated by the NFS server and opportunistic locks are requested by the CIFS clients. Table 3 summarizes the features of the two file systems.

HIGH-PERFORMANCE COMPUTING

Large-scale scientific simulations, which include those in astrophysics, computational chemistry, bioinformatics, nuclear testing, energy and petroleum, finance, and many others, dominate the field of high-performance computing (HPC). The TOP500 list, maintained by Hans Meuer, Erich Strohmaier, Horst Simon, and Jack Dongarra, describes the top supercomputers in the world, and they are classified architecturally as clusters, massively parallel processing machines, and constellations. Typically, applications in the HPC domain are optimized heavily for performance and scalability.

Most of these applications use the message passing interface (MPI) (28), the most commonly used portable, parallel API in the HPC community. Its portability allows scientists to run their applications on a variety of supercomputing platforms with minimal effort. In 1997, the MPI-2 standard was created by the MPI Forum to address parallel I/O (MPI-IO) as well as add other useful new features for portable parallel computing. ROMIO (29) is the reference MPI-IO implementation distributed with Argonne National Laboratory's MPICH library. Other MPI distributions, such as OpenMPI and LAM, often use ROMIO directly or as the basis for their own MPI-IO implementations. Frequently, higher-level libraries (for example, netCDF and HDF5) are built on top of MPI-IO to leverage its portability across different I/O systems and

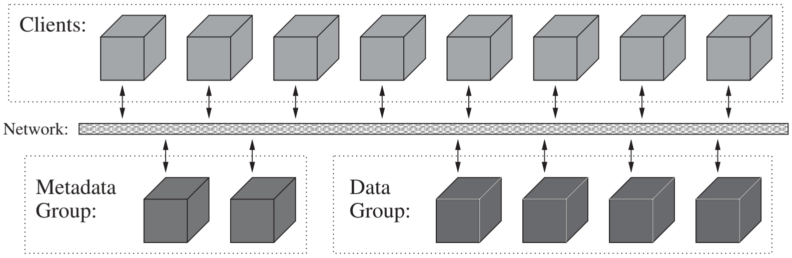


Figure 4. Typical parallel file system configuration. Clients have parallel access to components within the metadata and data groups.

to provide features specific to particular user communities. As the gap between processor and hard disk technologies continues to widen, I/O becomes an increasingly severe performance bottleneck. Parallel file systems, as shown in Fig. 4, help to narrow that gap by scaling up the number of hard disks to increase aggregate I/O bandwidth.

The HPC file system domain can be divided into production file systems and research file systems. Typically, production file systems are stable commercial products used in production machines. Some examples of production file systems include Lustre (30), Panasas (31), GPFS (32), SGI's CXFS, IBRIX FusionFS (33), and GFS (34). Research file systems are used primarily for trying out new ideas that may one day make it into production if appropriate. Several research file systems exist, which include PVFS (35,36), Clusterfile (37), Ceph (38), LWFS (39), Galley (40), Sorrento (41), and many more. We have chosen to focus our discussion on the three most used HPC production file systems: (Table 4), Lustre, Panasas, and GPFS, in the following three sections. We also describe three prominent HPC research file systems (PVFS, LWFS, and Ceph) in the sections that follow (Table 5).

Lustre

Lustre (30), from Cluster File Systems, gets its name from a portmanteau of the terms "Linux" and "cluster." As of the June 2006 TOP500 list, over 70 of the 500 supercomputers use Lustre technology, which includes the number one computer (Lawrence Livermore National Laboratory's BlueGene/L machine). The Lustre architecture consists of clients, metadata servers (MDSs), and object-storage targets (OSTs). MDSs maintain a transactional record of high-level file system changes, such as the location of related objects and stripe sizes. They are protected from failure through MDS replication and failover techniques. OSTs are responsible for actual file data and locking. Clients make requests to objects on the OSTs, in which an object is simply a container of data that may have attributes associated with it. In the future, object-based disks (OBDs) may be able to offload the work necessary to translate file system requests into physical storage requests. Currently, Lustre uses OBD device drivers to implement OBD functionality on top of ext3 or other Linux file systems. Failure of an OST is handled by failover techniques. If a failover OST is unavailable, clients will get errors when trying to access the failed OST and new file create operations will avoid the failed OST.

Lustre uses a distributed lock manager (DLM) to ensure POSIX compliance. The DLM helps Lustre to maintain its globally coherent collaborative cache. Although locks for an arbitrary byte-range may be requested, OSTs round the granted locks to file system block boundaries. Metadata operations use "intent based" locks (lock requests combined with data requests) for efficient atomic operations that do not require lock revocations. Additionally, Lustre provides snapshots, rollback, and copy-on-write semantics. Lustre uses secure network attached disk features for authentication, authorization, and encryption. A preliminary Lustre driver for the ROMIO MPI-IO implementation has not yet been integrated into the ROMIO distribution.

Panasas

Panasas (31,42) is used on many TOP500 supercomputers and was chosen to be deployed on the Los Alamos National Laboratory's new Roadrunner petascale supercomputer. Many application domains, which include energy research, high energy physics, atmospheric science and weather prediction, seismic data analysis, automotive design and simulation, as well as many others, have chosen Panasas as their storage solution. Panasas's main product is the ActiveScale Storage cluster, which uses the Panasas ActiveScale File System (PanFS). The core PanFS architecture is based on the decoupling of the datapath from the control path and the *object* abstraction of file data, similar to

Table 4. A comparison of HPC production file systems

Category	Lustre	Panasas	GPFS
Dedicated servers	Yes	Yes	Yes
Consistency semantics	POSIX	POSIX	POSIX
Consistency enforcement	DLM	MDS	DLM
Caching	Client memory Block-level	Client memory Block-level	Client memory Block-level
Fault tolerance	Fail-over servers	OSD-level RAID	Log-based & disk-level RAID
Replication	MDS	OSD-level RAID	Two copies & RAID
Security	Capability	Capability	OpenSSL

Table 5. A comparison of HPC research file systems

Category	PVFS	LWFS	Ceph
Dedicated servers	Yes	Yes	Yes
Consistency semantics	MPI-IO	N/A	POSIX
Consistency enforcement	Servers	Library	OSD locks
Caching	Server memory Block-level	Library	Client memory Block-level
Fault tolerance	Fail-over servers	Library	RADOS
Replication	Server RAID	Library	RADOS
Security	In progress	Capability	Capability

Lustre. The PanFS client module accepts POSIX file system commands from the operating system and addresses and stripes the objects across multiple OSDs. The OSD component in PanFS manages data storage, handles storage-side caching and prefetching, and contains the metadata associated with its objects. Using OSDs instead of the typical block-based storage interface shifts some of the burden of fine-grain layout information to the OSDs. The PanFS metadata server (MDS) coordinates the layout of a file across OSDs, helps maintain RAID integrity, manages file and directory access, and keeps client caches coherent with file locks.

PanFS uses client-side data caching in the Linux buffer/page caches to complement the caching done by the OSDs. It aggregates writes on the client for more efficient I/O operation and also supports prefetching. The MDS handles client cache coherency with a single writer/shared readers protocol with invalidation and flushing. PanFS allows files to use different RAID levels individually across objects. To limit *incast* behavior and too many senders overflowing the network buffers, a two level striping layout is used to limit simultaneous accesses to the number of OSDs in a parity stripe. Therefore, files are striped across all the OSDs for maximum bandwidth, and the OSDs are broken up into RAID parity groups whenever appropriate (with a maximum of 13 objects per parity group). Panasas OSDs each have two SATA disk drives, a processor, RAM, and a Gigabit Ethernet network interface. An OSD battery-backed RAM cache allows data to be committed even if a power failure occurs.

General Parallel File System (GPFS)

IBM has designed many of the world's top supercomputers, which include the recent BlueGene/L architecture. Its flagship file system, GPFS (32), is available for its AIX and Linux clusters, and most recently on the BlueGene/L architecture as of December 2005. Although GPFS is designed primarily for high-performance computing, it is also used in industries such as media and entertainment, ISPs, finance, telecommunications, electronics, and retail. GPFS uses a shared-disk architecture, in which file system nodes have access to all disks through the network fabric. The disks are assumed to use the conventional block I/O interface (as opposed to the object based interfaces used by Lustre and Panasas). GPFS clients communicate directly with file system nodes, which perform I/O on their behalf. GPFS guarantees single-node equivalent POSIX semantics for file system operations across all nodes through the use of distributed locking. The only exception to POSIX compliance is that access time updates are not visible on all nodes immediately. The metanodes that handle metadata in GPFS are allocated dynamically with the help of the global lock manager.

The GPFS DLM is composed of a centralized global lock manager and the local lock managers on each file system node. Lock tokens are passed out by the global lock manager to the local lock managers that grant locks. A lock token is revoked only when another node requests conflicting lock operations to the same object. As with Lustre and Panasas, lock tokens play a large role to maintain cache consistency between nodes. Locks are acquired with byte-range granularity in GPFS and are rounded to block boundaries. The first node to write a file will receive a byte-range lock from zero to infinity. When the second node begins to write to the same file, the first node will relinquish part of its byte-range lock token until the offset of the second node's write. As more nodes write to the file, the byte-range lock tokens are further divided. In this way, GPFS attempts to keep locks as large as possible to avoid the increasing overhead of a plethora of locks.

Parallel Virtual File System (PVFS)

The first generation of PVFS (35) began at Clemson University to serve as a research-oriented open source parallel file system for Linux clusters. Since its inception, it has grown tremendously in popularity. A second generation version of PVFS (36) was released initially in late 2003 and has stabilized during the last couple years. This second generation of PVFS is intended to serve as a production file system as well as to quickly incorporate novel research ideas because of its highly modular architecture. The Argonne Leadership Computing Facility (ALCF) has selected PVFS as its storage solution. In this section, we describe the second generation PVFS storage system.

The PVFS architecture has clients and I/O servers. The I/O servers may manage metadata, data, or both. Clients communicate directly with I/O servers to access file metadata, file distribution information, and file data, similar to other parallel file systems. Modularity has been introduced in the networking subsystem through the buffered messaging interface to abstract access to various underlying networking technologies. Similarly, the trove storage interface provides APIs for various storage implementations. PVFS was redesigned, in part, to handle noncontiguous data access efficiently through its request system. PVFS requests understand and process derived datatypes built on basic datatypes such as contigs, vectors, and structs, similar to MPI derived datatypes. In addition, PVFS has a highly optimized MPI-IO device driver that can, in most cases, make a one-to-one mapping between MPI-IO calls and PVFS system calls. To improve fault-tolerance, PVFS has stateless clients and servers to minimize the impact of failing components. Failover high-availability solutions can be used by PVFS if multiple machines have access to shared storage.

Light Weight File System (LWFS)

Catamount, a lightweight operating system for Red Storm (currently number two in the TOP500 as of November 2006) at Sandia National Laboratories (SNL), implements only the required underlying services while avoiding functionality that could compromise application scalability. In the same spirit, the LWFS (39) project is a joint collaboration between SNL and the University of New Mexico to investigate the viability of a "lightweight" approach to I/O. The LWFS core only implements a thin layer of software above the hardware, which includes infrastructure to provide controlled access to distributed data across multiple storage severs, to expose the parallelism of multiple storage servers, and to allow the client implementation to create additional functionality. Because many more compute nodes exist than I/O nodes, LWFS servers determine when to move data. LWFS clients make asynchronous RPCs and servers either "pull" data for writes or "push" data for reads (43). All data movement is performed over the Portals message passing interface that supports one-sided operations.

In accordance with U.S. Department of Energy security requirements, LWFS provide scalable mechanisms for authentication, authorization, and "immediate" revocation of access permissions when policies change. LWFS has coarse-grain access control to *containers* of objects, in which every object belongs to a single container. All objects in the same container are subject to the same access control policy. Higher-level libraries are responsible to organize objects in containers as LWFS does not manage the relationship of objects in a container. To enable scalable security, LWFS uses fully transferable credentials and capabilities. To support "immediate" revocation, LWFS invalidates cached entries on each of the storage servers.

Ceph

Ceph (38) is a research-oriented file system from the University of California at Santa Cruz. It has three major components: clients that export a near-POSIX file system interface; a cluster of OSDs that collectively store all metadata and data; and a metadata cluster responsible to manage the namespace and coordinating security, consistency, and coherence. As with the other object-based file systems, Ceph separates file metadata management from data storage. Ceph uses its reliable autonomic distributed object store (RADOS) to protect against OSD failures. Primary OSDs forward updates to their replicas in an asynchronous manner for better performance, and reads are only serviced by the primary OSD to reduce synchronization costs.

In the metadata cluster, Ceph employs dynamic distributed metadata management that is based on dynamic subtree partitioning. In essence, dynamic distributed metadata maps subtrees of the directory hierarchy to metadata servers based on their workload. Individual directories are hashed across multiple nodes only if they become hot spots.

For data distribution, Ceph uses the controlled replication under scalable hashing (CRUSH) algorithm (44). CRUSH relies heavily on a suitably strong multi-input integer hash function. Using the hash function, CRUSH can locate any object with a placement group and an OSD cluster map. Placement rules help CRUSH map the placement groups onto OSDs based on the desired level of replication as well as other constraints. CRUSH also helps Ceph adapt to the addition and removal of storage devices with low overhead.

APPLICATION-SPECIFIC

In distributed file systems, design decisions are made to balance performance, scalability, reliability, usability, and security. Ideally, all these aspects would be maximized, but some desirable traits conflict inevitably with each other, which leads to tradeoffs. For example, achieving a usability characteristic like strong consistency typically hinders performance and scalability, and the converse is also true. Typically, general purpose distributed file systems attempt to provide reasonable support for most of the above features. A file system that has been designed for a specific application, however, can relax certain restrictions (based on the requirements of the application) to improve certain behaviors. For example, search engines store very large files that rarely are deleted or overwritten, as most of the requests involve appending or reading files in bulk. In this particular case, a relaxed consistency model is acceptable for scalability and efficiency reasons.

We will discuss three such special purpose file system areas. In the next section, we consider web application optimizations on the Google file system that supports the Google search engine. In the section after that, we examine security optimizations for the self-certifying file system (SFS). Last, we discuss how distributed file systems are used for virtualized storage. The characteristics of all file systems in this section are shown in Tables 6 and 7.

Table 6. A comparison of GFS and SFS file systems

Category	Google FS	SFS
Dedicated servers	Yes	Yes
Consistency semantics	Application-specific	N/A
Consistency enforcement	MDS	NLM
Caching	Metadata caching at client	N/A
Fault tolerance	Shadow masters, logging, checksum	Stateless servers
Replication	Three copies (default)	N/A
Security	Capability	Separate key management

Table 7. A comparison of virtual machine distributed file systems

Category	VMFS	VxFS
Dedicated servers	Optional	Optional
Consistency semantics	Guest OS	Guest OS
Consistency enforcement	On-disk locks	I/O fencing
Caching	OS-dependent	OS-dependent
Fault tolerance	Journal-based recovery	VCS backup
Replication	Unknown	FlashSnap
Security	SAN	SAN

Google File System

The Google file system (GFS) (45) is a scalable, distributed file system designed for Linux platforms. GFS evolved out of BigFiles (46), which was developed at Stanford in the early days of the Google search engine. At that time, BigFiles existed primarily to store multi-Gigabyte files efficiently. Today, GFS runs on hundreds or thousands of commodity Linux machines and is tailored for high-performance data-intensive applications as well as storing a few million very large files. Because multi-Gigabyte files are common, the overall system should be optimized for large files. Small file access is also supported, but not optimized. The largest of the current GFS clusters is 1000 nodes with a 300 TB storage capacity. It is accessed concurrently by hundreds of clients (45). These machines, both cheap and unreliable, often fail. On average, at least one machine will fail everyday at Google, so it can be assumed that not all of them will be working at any given time (47). Some challenges for GFS are fault tolerance and fast recovery support. GFS runs a persistent monitoring mechanism that helps make it fault tolerant and automatically recoverable.

Because GFS was designed to support a search engine, architectural decisions are based on several domain-specific characteristics: files are rarely deleted, overwritten, or shrunk; most of the workload involves large contiguous writes when files are being appended; small writes are supported but are not optimized to keep high efficiency for large writes; high sustained bandwidth is preferable when compared with low latency for individual reads or writes; and autonomy with minimal synchronization overhead is essential to allow multiple clients the ability to append to the same file concurrently.

Architecture. The GFS architecture consists of a single *master*, multiple *chunkservers*, and the client library, as illustrated in Fig. 5. GFS employs commodity Linux machines with user-level server processes that run on each of them. Files are divided into fixed size (default 64 MB) chunks, each of which is identified using a unique 64-bit chunk handler. Chunks are stored on local disks as Linux files. For fault tolerance and recovery, each chunk is replicated on at least three chunkservers. The master maintains the metadata for the entire file system: mostly namespace and access control information, the mapping from file to chunks, and the current locations of every chunk. The master communicates with chunkservers through *HeartBeat* messages to give instructions and to collect states.

Because of the single master architecture, larger file systems can be supported on the master at the cost of adding extra memory to handle the additional metadata load. The master does not keep a consistent record of which chunkservers have replicas on a per-chunk granularity. During startup, the chunkserver provides this information to the master, and subsequently the master keeps itself updated with HeartBeat messages. This protocol avoids consistency issues when a chunkserver crashes. The GFS client library is linked with each application and it reads and it writes on behalf of the linked applications. A client interacts with the master to acquire metadata, but all data

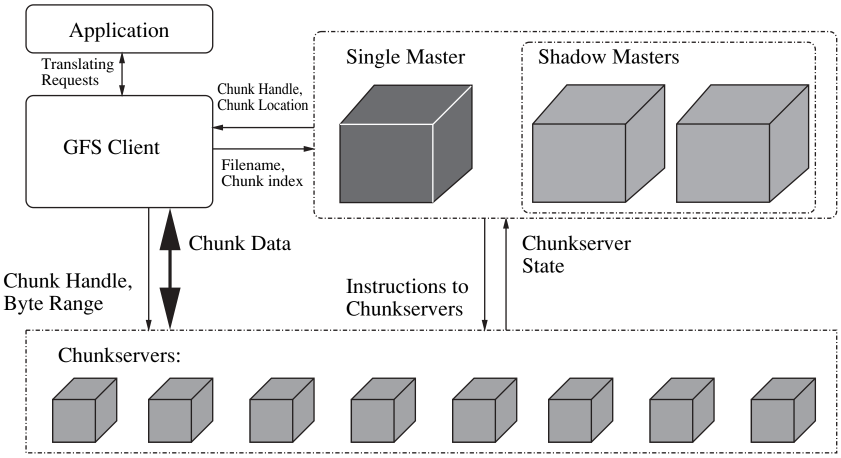


Figure 5. The Google File System architecture.

communication is handled strictly between a client and the chunkservers. The client does not cache data because most of the Google applications stream through huge files; however, they do cache metadata to avoid repeated access to the master. Chunkservers cache frequently accessed data in the Linux buffer cache because chunks are stored as local files on Linux machines.

Record Append and Consistency. GFS has a relaxed, simple, and efficient consistency model. Most of the target applications for GFS involve large sequential writes and large streaming reads. Small random reads or writes at arbitrary positions are also supported, but need not be highly efficient. A unique result of the GFS design is that appending to a file is more efficient than overwriting it. A write causes data to be written at an application specified offset. A record append, on the other hand, appends a record atomically at most once even in the presence of concurrent mutations at an offset decided by GFS. The client only specifies the data, and the offset returned to the client is the beginning of the record region. GFS may insert padding or record duplicates between appended records. The reader may deal with occasional padding and duplicates using checksums, or can remove duplicates by using unique identifiers in records.

Fault Tolerance. The master and the chunkservers are designed to restore their state seconds after a failure. As already mentioned, every chunk is replicated on at least three different chunkservers. If a chunkserver goes down or corrupted data is detected through checksum calculations, then other replicas of this chunk are used to recover the correct data. The master state is also replicated for reliability. Operation logs and checkpoints are replicated on multiple machines. A mutation is considered committed only after its log record has been flushed to disk. The master is in charge of all mutations as well as garbage collection. If it fails, it must be restarted immediately. A new master can be created by using replicas of the operation log. There are also *shadow masters* that provide read-only access to the file system while the actual master is unavailable (Fig. 5). Shadow masters are not mirrors; they lag behind the master by fractions of a second. GFS generates many operation logs to record significant events. These files can be deleted right away but are kept as long as space exists. RPC logs contain requests and responses, but not data. These logs can serve for load testing and for later analysis.

Self-Certifying File System

In almost any system, file system or other, often a tradeoff occurs between security and performance. In the case of distributed file systems, an additional tradeoff is scalability. To ensure secure and transparent transfers we need remote file transfer protocols to operate securely between physically dispersed workstation environments. These protocols should also be able to provide confidentiality, authentication, and data integrity. Some applications of secure file systems are financial transactions, multimedia streaming, medical records, and devices that use digital rights management.

Replicas are used commonly in distributed file systems for better locality as well as fault tolerance. Secure file systems can either store unencrypted replicas on trusted servers or encrypted replicas on untrusted servers. A built-in key management system may be required to ensure security.

Most file systems come with a key management system; some examples include Kerberos (48) and SSL (49). Internet file sharing deals with such wide diversity that managing encryption keys becomes very cumbersome, and establishing a secure web server with SSL can take a significant amount of time. The Self-certifying File System (SFS) tries to solve the above mentioned problems of security, key management, and extensibility.

Related Work. Before we discuss SFS, it is appropriate to understand some of the security mechanisms present in other distributed file systems. The Andrew File System (AFS) (1,2,50) is one of the earliest and most successful secure distributed file systems. It uses a message authentication code to protect the integrity between client and server. AFS uses password authentication to guarantee the integrity of remote files. After logging into an AFS client machine, a user is able to obtain a key shared by the file server. If malicious users gain access to a session key, they can pollute the client disk cache, buffer cache, and name cache for parts of the file system that they supposedly should not have permission. If multiple users log on the same AFS client machine, they must either trust each other or the operating system (OS) must maintain separate secure caches for each user.

SFS Design. SFS claims to provide better security and extensibility without key management. SFS (51) is a secure distributed file system that removes key management from the file system entirely. Like AFS, it also provides a shared namespace, but it introduces self-certifying pathnames (filenames) that effectively contain the appropriate remote server's public key. It makes sharing of files over the Internet secure by allowing the local area network to gain control of a remote file by using self-certifying pathnames. Because pathnames already specify the public key, SFS doesn't need a separate key management mechanism to communicate with file servers. By moving the key management scheme out of the file system, many key management policies can coexist in the same file system. This makes SFS extensible while securely working over the untrusted Internet.

The overall security of SFS can be divided in two parts: file system security and key management. SFS provides only file system security, so a malicious user can't read or change the file system without permission. SFS ensures that an attacker can do no worse than delay the file system's operation, and any data that a client receives can be verified as authentic. Clients and read-write servers always communicate over a secure channel that guarantees secrecy and data integrity. Although self-certifying pathnames solve the problem of authenticating file servers to a user, SFS must also authenticate users to servers. When a user first accesses an SFS file system, the client delays the access and notifies its authentication agent of this event. The agent can then authenticate the user to the remote server before file access begins. A server-side authentication server program performs user authentication. The agent and authentication server pass messages to each other through SFS using a protocol opaque to file system. Security, extensibility and portability are achieved at a performance cost attributed mostly to the underlying encryption overhead in SFS.

SFS Read-Only. For a read-write secure file system, expensive encryption/decryption becomes the critical path and performance does not scale with the number of processes. The SFS read-only file system (52) is a distributed file system that allows a high number of clients to access public read-only data securely with acceptable performance. The data of the file is stored in a database and is signed off-line with the private key of a system stored in a database to replicate on many untrusted machines. In online certificate authorities, frequent disk accesses are avoided by having copious amounts of memory. The SFS read-only server performs better than the SFS read-write server because no online cryptography operation exists. Public key decryption is a performance bottleneck for a SFS read-write server. The SFS read-only server pushes the cost of cryptographic operations from the server to the clients, which allows the server to support a large number of clients.

Virtual Machine Distributed Storage

In the enterprise domain, virtualization has become an important technology that aids high availability, server consolidation, and reduced testing complexity. Typically, the concept of virtualization revolves around the idea of a virtual machine, or hardware virtualization, that is based on adding a virtualization layer between the hardware and the OS. This virtualization layer allows multiple virtual machines to run concurrently on a single actual machine that shares its physical resources among them. Some example virtualization vendors include VMware, Xen, Qemu, Parallels, and Innotek.

Virtualized storage refers to the abstraction of logical storage from physical storage. Although an OS may believe that it has a single, SATA-connected hard drive, in reality, the physical storage device may be network-based, storage device-based, or host-based (typically, distributed file systems). Several virtualization vendors use distributed file systems to support virtual storage for their virtual machines. A virtual disk may be as simple as a file on a remote server that allows any client with connectivity to the server to resume the virtual machine. Distributed file systems that support virtualized storage efficiently include VMware's VMFS and Symantec's VxFS. In 2003, Red Hat purchased Sistina to use the global file system for its upcoming virtualization platform; however, a product is yet to be released. These distributed file systems are corporate products, which makes it difficult to find detailed information.

VMFS. VMware provides several virtualization products for both desktops and servers. Their storage virtualization solution that is optimized for virtual machines is the Virtual Machine File System (VMFS). VMFS was designed to allow virtual machine state to be stored in a centralized repository. VMFS-3 is the latest revision that addresses manageability, availability, scalability, and performance issues and can use a wide range of Fibre Channel and iSCSI SAN equipment.

Several VMFS features improve manageability and availability. Distributed journaling and journal-based recovery allow for faster recovery during server failure. An exhaustive file system check would take a long time before the server could come back online. VMFS can hot add virtual disks to running virtual machines to handle increased application requirements or provide backup capability. Logical unit numbers are discovered automatically and are mapped to VMFS volumes. VMFS-3 now supports many files using techniques similar to other file systems rather than the flat address space in VMFS-2. On-disk locking ensures that operations are atomic across shared virtual storage.

To support its performance and scalability goals, VMFS has several optimizations. Block sizes are adaptive and can adjust for both max file size limits and better backend resource use. Because backend storage devices are disks, increasing the access sizes has a lot of potential to improve performance and to reduce network traffic. Caching is used for nonvirtual disk-based files because guest OSs expect syncing the disk to push data to the storage devices. Changes in the on-disk locking protocol allow better scalability with respect to the number of files open in a virtual machine. Older versions of VMFS stored on-disk locks per file in different (noncontiguous) sectors, on disk. VMFS-3 now stores all locks in a single sector that supports the same number of open files as other VMware products.

VxFS. Recently, Symantec acquired Veritas in 2004 to consolidate its enterprise operations. Instead of writing its own core virtualization software, Symantec makes virtualization products, such as the Veritas Cluster Server (VCS), based on VMware and Xen technology. Symantec also provides virtualized storage solutions based on its Veritas Storage Foundation Cluster File System, which includes the Veritas File System (VxFS) and the Veritas Volume Manager (VxVM). VxVM volumes are used as boot disks for guest OSs to enable easy cloning. VxFS is used in the guest OS for better reliability and performance.

VCS improves administrator efficiency through reduced management. The cluster nodes share a single set of configuration and data files, which requires the administrator to "manage" only a single node regardless of the number of nodes in the cluster. VxFS also enables VCS backup/recovery operations by means of shared access and FlashSnap, a point-in-time copy of production information. If file systems are checkpointed, the file system can be "rolled back" to a consistent point in time. VxFS is an integral part of VCS's ability to handle both application and node failures. If a node fails, the application will be migrated dynamically to an available node in the cluster. Additionally, because storage resources are consolidated and abstracted from the virtual machines, maintenance costs are reduced.

From a performance point of view, VxFS provides *Dynamic Storage Tiering*, a technology that moves unimportant or out-of-date files transparently to less expensive storage hardware. The policies can be dynamically set, are centrally managed, and work on heterogeneous server and storage infrastructure. RAID support provides performance and reliability as per user needs. To ensure data integrity, VxFS uses I/O fencing through the SCSI-3 persistent group reservation technology. VxFS can remove access from "errant" nodes using I/O fencing. Automatic performance tuning helps the system adjust to dynamically changing workloads.

BIBLIOGRAPHY

1. J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Side-botham, and M. J. West, Scale and performance in a distributed file systems, *ACM Trans. Comput. Syst.*, **6**(1): 1988
2. M. Satyanarayanan, Scalable, secure, and highly available distributed file access, *IEEE Computer*, **23**(5): 1990.
3. M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, Coda: a highly available file system for a distributed workstation environment, *IEEE Transactions on Computers*, **39**(4): 447–459, 1990.
4. J. J. Kistler and M. Satyanarayanan, Disconnected operation in the coda file system, in *Thirteenth ACM Symposium on Operating Systems Principles*, volume 25, Asilomar Conference Center, Pacific Grove: ACM Press, 1991, pp. 213–225.
5. P. Braam, M. Callahan, and P. Schwan. The intermezzo filesystem, 1999.
6. D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, and J. Kubiatowicz. Oceanstore: An extremely wide-area storage system, Technical Report, University of California, Berkeley, 2000.
7. J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weath-erspoon, W. Weimer, C. Wells, and B. Zhao, Oceanstore: An architecture for global-scale persistent storage, *Proceedings of ACM ASPLOS*. ACM, 2000.
8. S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, Pond: the oceanstore prototype, in *Proceedings of the Conference on File and Storage Technologies*. USENIX, 2003.
9. B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, *IEEE J. Selected Areas Commun.***22**(1): 41–53, 2004.
10. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, Wide-area cooperative storage with CFS, *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, Chateau Lake Louise, Banff, Canada, 2001.
11. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001, pp. 149–160.
12. A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, Ivy: a read/write peer-to-peer file system, in *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.
13. C. A. Thekkath, T. Mann, and E. K. Lee, Frangipani: a scalable distributed file system, *Symposium on Operating Systems Principles*, 1997, pp. 224–237.
14. E. K. Lee and C. A. Thekkath, Petal: distributed virtual disks, in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, 1996, pp. 84–92.
15. F. Picconi, J.-M. Busca, and P. Sens, Exploiting network locality in a decentralized read-write peer-to-peer file system, in *Proceedings of International Conference on Parallel and Distributed Systems*, 2004.
16. A. Rowstron and P. Druschel, Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, SOSP, 188–201, 2001.
17. A. Rowstron and P. Druschel, Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems, *Lecture Notes in Computer Science*, **2218**: 329+, 2001.
18. Sun Microsystems, Inc., *NFS: network File System Protocol Specification*, Internet Engineering Task Force Network Working Group, RFC 1094, 1989.
19. B. Callaghan, B. Pawlowski, and P. Staubach, *NFS Version 3 Protocol Specification*, Internet Engineering Task Force Network Working Group, RFC 1813, 1995.
20. B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, NFS version 3: Design and implementation, in *USENIX Summer*, 1994, pp. 137–152.
21. B. Callaghan, *NFS Illustrated*. Reading, MA: Addison-Wisley, 1999.
22. H. Stern, *Managing NFS and NIS*, Sebastopol, CA: O'Reilly & Associates Inc., 1991.
23. S. Shepler, C. Beame, R. Callaghan, M. Eisler, D. Noveck, D. Robinson, and R. Thurlow, *Network File System (NFS) version 4 Protocol*, Internet Engineering Task Force Network Working Group, RFC 3530, 2003.
24. B. Pawlowski, S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson, and R. Thurlow, The NFS version 4 protocol, *Proceedings of the 2nd International System Administration and Networking Conference (SANE2000)*, 2000, p. 94.
25. A. Tanenbaum and M. vanSteen, *Disributed Systems - Principles and Paradigms*. Englewood Cliffs, NJ: Prentice Hall, 2002.
26. P. Leach and D. Naik, *Common Internet File System (CIFS) Technical Reference Revision: 1.0*, 2002.
27. T. O. Group, *Protocols for X/Open PC Interworking: SMB, Version 2*, 1992.
28. *Message passing interface forum*. Available: http://www.mpi-forum.org.
29. ROMIO: *A high-performance, portable MPI-IO implementation*. Available: http://www.mcs.anl.gov/ronio.
30. *Lustre*. Avaibable: http://www.lustre.org.
31. D. Nagle, D. Serenyi, and A. Matthews, The Panasas Active Scale storage cluster - delivering scalable high bandwidth storage, in *Proceedings of the 2004 ACM/IEEE Supercomputing Conferencence*, 2004.
32. F. Schmuck and R. Haskin, GPFS: a shared-disk file system for large computing clusters, in *Proceedings of the Conference on File and Storage Technologies*, San Jose, CA, 2002.
33. *IBRIX FusionFS*. Available: http://www.ibrix.com/.
34. *Global file system*. Available: http://www.rodhat.com/software/rha/gfs/.
35. P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, PVFS: A parallel file system for Linux clusters, *Proceedings of the 4th*

Annual Limxx Showcase and Conference, Atlanta, GA, 2000, pp. 317–327.

36. The parallel virtual file system 2 (PVFS2). Available: http://www.pvfs.org/pvfs2/.
37. F. Isaila and W. Tichy, Clusterfile: A flexible physical layout parallel file system, *Proceedings of the IEEE International Conference on Cluster Computing*, Newport Beach, CA, 2001.
38. S. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, Ceph: A scalable, high-performance distributed file system, Proceeding of the 7th Conference on Operating Systems Design and Implementation (OSDI' 06), 2006.
39. R. A. Oldfield, A. B. Maccabe, S. Arunagiri, T. Kordenbrock, R. Riesen, L. Ward, and P. Widener, Lightweight i/o for scientific applications, *in Proc. 2006 IEEE Conference on Cluster Computing*, Barcelona, Spain, 2006.
40. N. Nieuwejaar and D. Kotz, The Galley parallel file system, Technical Report PCS-TR96-286. Hanover, NH: Dept. of Computer Science, Dartmouth College, 1996.
41. H. Tang, A. Gulbeden, J. Zhou, W. Strathearn, T. Yang, and L. Chu, A self-organizing storage cluster for parallel data-intensive applications, *Proceedings of ACM Supercomputing Conference*, 2004.
42. Panasas. Available: http://www.panasas.com.
43. R. A. Oldfeld, P. Widener, A. B. Maccabe, L. Ward, and T. Kordenbrock, Efficient data-movement for lightweight i/o, *2006 IEEE Internat. Conf. on Cluster Computing*, 2006.
44. S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, Crush: controlled, scalable, decentralized placement of replicated data, *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, New York, 2006, pp. 122.
45. S. Ghemawat, H. Gobioff, and S.-T. Leung, The google file system, *SOSP*, 2003.
46. How google works. Available: http://www.baaelinemag.com.
47. Google's secret of success? Dealing with failure. Available: http://news.zdnet.com.
48. J. G. Steiner, B. C. Neuman, and J. I. Schiller, Kerberos: an authentication service for open network systems, in *USENIX Winter*, 1988.
49. T. Y1önen, Ssh - secure login connections over the internet, *USENIX Security Symposium*, 1996.
50. M. Satyanarayanan, Integrating security in a large distributed system, *ACM Trans. Comput. Syst.*, **7**(3): 1989.
51. D. Mazires, M. Kaminsky, M. F. Kaashoek, and E. Witchel, Separating key management from file system security, in *SOSP*, 1999.
52. K. Fu, M. F. Kaashoek, and D. Mazières, Fast and secure distributed read-only file system, *ACM Trans. Comput. Syst*, **20**(1): 2002.

Avery Ching
Kenin Coloma
Arifa Nisar
Wei-keng Liao
Alok Choudhary
Northwestern University
Evanston, Illinois

ELECTRONIC DATA INTERCHANGE

INTRODUCTION

Electronic data interchange (EDI) is the process by which a business organization exchanges business transactions between application systems in electronically processable forms. In this process, an automated business application system originates the transaction, the value-added network (VAN) transmits it to the receiver, and an automated business application system at the receiver adequately responds to the transaction. For example, at a store, the bar code scanner at the cash register can update the inventory of each item sold. If the inventory falls below some predetermined number, the bar code scanner system triggers an ordering system. The ordering system creates an order and hands it over to the EDI system. The EDI translator translates the purchase order into a standardized transaction set according to ANSI ASC X12.850 standards and electronically sends the purchase over to a vendor's mailbox using an EDI VAN. Human intervention is not needed in any step of the whole process. It is clear from the above example that for EDI to be successful, integration must exist among various business application systems and the EDI software. The EDI system should support the seamless location, transfer, and integration of business information in a secure and reliable manner.

EDI uses computers to transmit business transactions and, in the process, eliminates paperwork significantly. With this paperless transfer of data, one does not have to rekey the information at the receiving end. Therefore, errors, time, and cost incurred in the rekeying of data are saved. This automatic creation and transfer of business transactions enables organizations to improve accuracy of business data, better serve their customers, improve relationships with suppliers, and effectively compete in the global market. For example, just-in-time (JIT) inventory control practices that have significantly cut inventory costs will be difficult to implement without EDI.

In addition to the above-mentioned direct benefits, EDI provides many indirect benefits. EDI standardizes business transactions for the whole industry, as participants in EDI must agree in advance on what data are to be exchanged, in what order, and what format needs to be used. This standardization helps in streamlining the transaction process, as parties do not have to go back and forth asking for clarifications or missing data. The federal government as well as major companies expect their suppliers to use EDI. For example, the U.S. Department of Defense will not transact business with a vendor any other way except through EDI. So a vendor has no choice but to have EDI capabilities. This article details EDI system components and processes needed to implement EDI.

EDI SYSTEMS AND PROCESSES

To automate transactions processing among different business partners, a successful EDI system has the integrated components shown in Fig. 1 (1,2).

STANDARDS

Every industry has a set of transactions. Different terms have specific meaning and usage in a specific industry. Standards are needed so that transactions are formatted in a structure that can be processed by the transaction processing systems of the industry. Standards provide the framework for formatting any specific transaction. ANSI ASC X12 and EDIFACT are the two predominant standards.

ANSI ASC X12

The American National Standards Institute (ANSI) is the national body that coordinates the development of standards in all areas of business. ANSI created the Accredited Standards Committee (ASC) X12 and gave it a charter to develop a set of standards for electronic exchange of business transactions. ANSI ASC X 12 standards define the data structures and the rules for encoding business transactions. Following; are the structures used in ANSI ASC X12 standards:

Data Element

A data element is the very basic or elementary unit of information, for example, item number, quantity, item description, and so on. The characteristics of each data element are defined. A group of simple data elements that represents a single named item is known as a composite data element. For example, if a piece of metal has to undergo seven different machining processes, then 1c234de represents those seven machining processes.

Data Segments

A data segment consists of a group of related data elements. These logically related data elements are arranged in a predefined sequence to generate a data segment. For example, an address segment consists of a group of data segments, that is, company name, city, state, and zip code. A segment contains some data elements that are essential, whereas other data elements may be optional. Some of the optional data elements may not be applicable for a business; therefore, they are omitted in the transaction. When a data element is omitted, the data element separator should

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

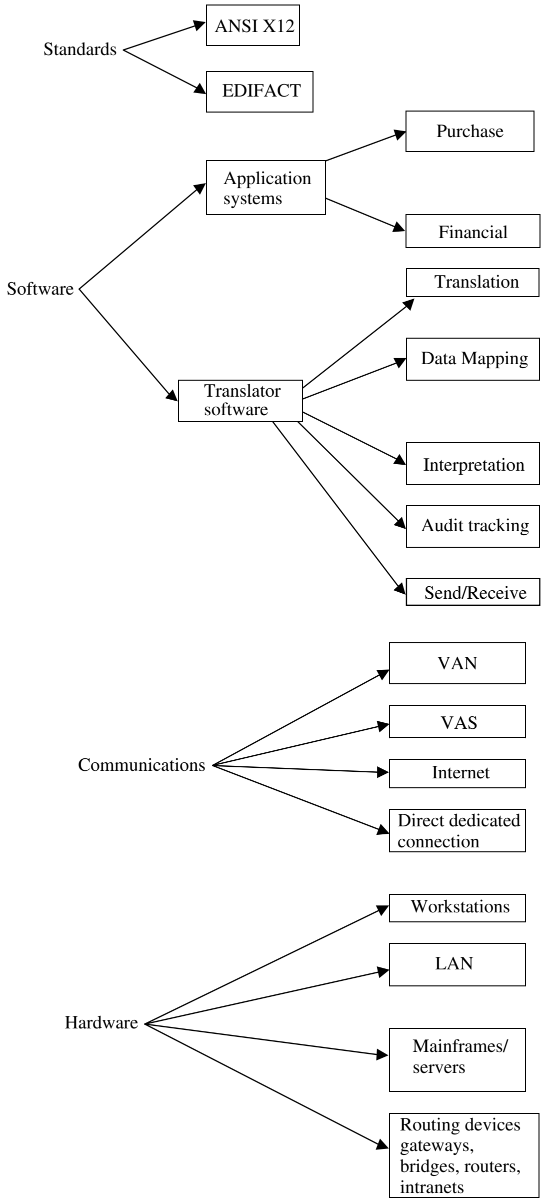


Figure 1. EDI system and processes.

explicitly indicate such an omission. For example, a purchase order can be sent as follows:

PO1**100*EA50.00**VC*P123

*	Element separator
PO1	Purchase order1
100	Quantity
EA	Each
**	Omitted data element
50.00	Price
VC	Vendor catalog
P123	Part number P123

Quantity: 100
Unit: Each
Description: Part No 123
Unit Price: 50.00
Total: 5,000.00

Transaction Set

A transaction consists of a group of related data segments that must be present to provide information for a viable business transaction. For example, transaction set X12 840 is a request for quotation (RFQ). This transaction set X12 840 provides the information about different data segments and data elements that are required to make RFQ a meaningful transaction. Similarly, X12 850 is a purchase order and X12 855 is a purchase order acknowledgement. To create a format for a transaction, one has to define the following:

- segments to be used,
- the structure of each segment,
- data elements to be used in each segment, and
- the characteristics of each data element.

Functional Group

A functional group consists of a group of similar transaction sets. For example, if there are three RFQs for three different items to be sent to the same trading partner, the EDI software will create one interchange with one RFQ functional group. This RFQ functional group will contain three different transaction sets, one each for three different items. Some EDI translators allow several different functional groups to be included in one interchange. For example, if two responses for RFQs and five purchase orders are being sent to the same trading partner, the EDI translation software will create one interchange that contains two functional groups, that is, one RFQ response functional group and one order functional group.

Envelope

An EDI envelope is a specialized segment that contains (1) routing information; that is, it provides addresses of both the sender and the receiver of the transmission. The address segment marks the beginning of the transmission, (2) the date and timings of the EDI interchange; (3) the unique control number used for tracking the transaction; (4) the authorization and security information; (5) the EDI standards and version of the interchange; and (6) the number of functional groups in the interchange. Figure 2 explains the structure of EDI envelope, arrangement of functional groups, and transaction sets (1,2).

EDIFACT

For international trade, the United Nations rules for EDI for administration, commerce, and transport (EDIFACT) provide a set of standards, directories, and guidelines that

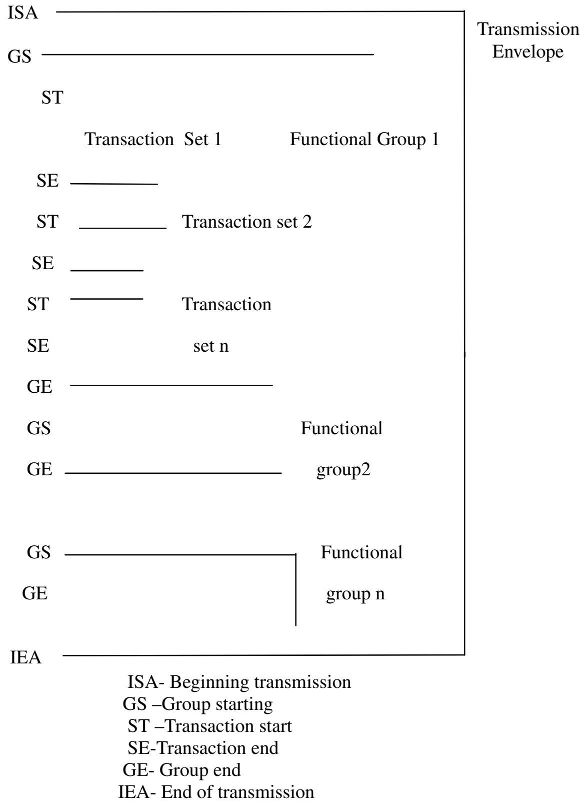


Figure 2. EDI envelope and group mapping.

have been internationally agreed upon for electronic exchange of structured business transactions. EDIFACT is a global attempt to standardize such information exchanges so all computers involved are speaking the same language, which will create an open system that anyone can join at any point. EDIFACT is designed to be independent of software, hardware, or communication media, thus accomplishing universal connectivity. The International Organization for Standardization (ISO) adopted the EDIFACT syntax in 1987.

To achieve global open EDI, one can use EDIFACT document syntax rules, X.400 message handling systems, and X.500 directory services. X.500 directory services can be used to store product information so that purchase managers can order electronically. These X.500 directory services are a powerful tool that allows EDI to take place between organizations without prior EDI agreements.

Both ANSI ASC X12 and EDIFACT standards perform the same functions. ANSI ASC X12 is an older standard and provides many more functions than EDIFACT. The EDIFACT organization is trying to develop additional functions. The two standards have different syntax and therefore it is difficult to convert transactions from one system to the other. In January 1995, the ANSI ASC X12 development body decided to follow the syntax and standards of EDIFACT so that full compatibility is achieved. EDIFACT can be used for both domestic and international interchanges, whereas ANSI ASC X12 is mainly for domestic interchanges. One can obtain a complete listing of both these standards from the Data Interchange Standards Association.

EDI SOFTWARE

An organization may have automated applications in the area of finance, marketing, accounting, production and operations, and human resources management. Data are entered into these application systems and transactions are generated that may have to be communicated to business partners. These business information systems call on EDI software to establish and maintain standards and handshaking rules for communicating among business partners. EDI software defines the methods, timing, and routines for receiving, transmitting, storing, and updating transactions among application systems (see Fig. 3). EDI software makes the exchange transparent; that is, it hides the complexity of the underlying communication protocols from the end user. A good integrated EDI software package provides the following functions:

- an application interface,
- translation, and
- data communication.

Application Interface Software

As the term indicates, the application interface software is the software bridge that facilitates the interface between the business application system and the EDI standards translation software. This software enables transparent flow of transactions between business partners. After the required data have been entered in the application software and the transaction is ready to be transmitted to the receiver, this software retrieves transaction data from the application database and places them into a flat file for subsequent conversion into EDI-formatted data before to transmission to trading partners. Flat files are used to pass transaction data between an application system and the EDI translation software. System interface software is important for both outgoing and incoming transactions, as it either reads or writes flat files of transaction data. For incoming transactions, this software retrieves data from a flat file and prepares them for acceptance by the application system. Some transaction software packages may not use a flat file because they exchange data directly with the application system database, thereby eliminating the need for interface software.

Standard Translation Software

A business organization transacts business with many trading partners. Some degree of flexibility is needed to support communication with the various trading partners because a need may exist to modify a trading partner's data to ensure compliance with the standards or to facilitate integration with the user's application system. EDI translation software allows for both the semantic translation and the syntax translation of the data element. A summary of

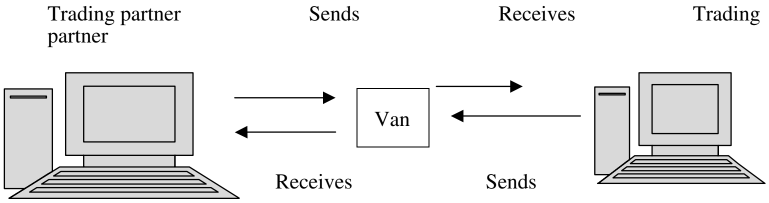


Figure 3. EDI process.

the characteristics of the standard translation software listed by the National Institute of Standards and Technology is as follows:

Transaction Set Mapping. Translation software translates data retrieved from an application database into a standard EDI format before it is transmitted to trading partners. It also converts EDI-formatted data, for example, in ANSI ASC X12 format, received from trading partners into a file format that the application system recognizes. Before the translator can translate data, it must know the location of the data to be translated. Some translators require some users to create a separate flat file formatted as an ASCII text file. Such a flat file helps in the standardization of data from various files and different formats. Some translators have a utility called "transaction set mapper." The transaction set mapper cross-references the contents of the flat file with an EDI standard set and subsequently translates the flat-file information into the desired transaction set. Mapping from/to the standards to/from the application formats is one of the key functions of translation software. The mapper reduces the amount of programming for application system interface. Data manipulators map internal data fields to applications according to an ANSI ASC X12 transaction set, which enables different trading partners to exchange transactions.

Character Set Convention. If business applications of the trading partners use different character sets (ASCII and EBCDIC), the need may exist to convert one to the other. Sometimes EDI software may do the conversion or, if VAN is used, it will do the required set conversion.

Code Conversion. Codes used in a vendor's application program might be different from the EDI codes. For example, the X12 ID Qualifier for serial number is SN, whereas the user application might use the code SRNUM to identify a serial number. The EDI software converts the standard codes to and from the user's code to facilitate integration between the user's application and EDI software.

Automatic Compliance Correction. For both inbound and outbound data, EDI software verifies the identity of trading partners, the syntax of the data, and whether it complies with the EDI standards and version being used. To accomplish this verification, EDI software references its tables of EDI standards at the user's trading partner profiles. Some simple errors are automatically corrected by adjusting the data to make them comply with the standards.

Manual Compliance Correction. Some compliance verification errors may be so severe that EDI software cannot automatically correct them. In such circumstances, the software suspends the processing so that the end user can review the transaction, correct the errors, and submit the transaction for reprocessing.

Duplicated Number Detection. Some EDI software tracks the use of business document numbers, such as purchase order numbers. If a number is duplicated, the software identifies the duplication and can take several different actions. It can either display or log error messages or it can suspend processing of the transactions until the end user can correct the duplication.

Functional Acknowledgement. Senders of transactions would like to know if the recipient received the information. The ANSI ASC X12 997 transaction set is known as functional acknowledgement. The recipient uses functional acknowledgement to send the sender an acknowledgement of the EDI transaction. It verifies the acceptance or rejection of a transaction set and reports any syntactical errors. Generally, EDI translators are so configured as to automatically return functional acknowledgement.

Document Type Sequencing. Control numbers are used to identify functional groups in an exchange. There may be several different kinds of document types within the multiple functional groups. These document types are also identified using control numbers. Each trading partner may have a set of functional group and document control numbers sequentially. It is easy to find a missing document from transmission by viewing the lapses in document control numbers.

Multiple Functional Groups. Some EDI translators permit multiple functional groups in one interchange. For example, if three invoices (ANSI ASC X12 810) and two RFQ (ANSI ASC X12 840) responses are being sent to the same trading partner, the EDI software creates one interchange containing two functional groups, that is, one functional group for invoices and the other for RFQ responses. If the software does not support multiple functional groups, then two interchanges would be needed, one for each functional group. The second interchange would cause increased overhead in terms of double transmission costs and greater storage requirements.

DATA COMMUNICATION SOFTWARE

The communication software establishes the communication link between the sender and the receiver. One can use a general-purpose data communication software for modem dialing and connecting to VANs. To complete this job, the communication software has to perform several tasks.

Protocol(s) Support

Communications software must support the required protocol(s). Some EDI software include: asynchronous transmission; others provide bisynchronous transmission. These programs would provide seamless transmission if they were fully integrated with simple mail transfer protocol (SMTP) or X.435.

VAN Script Files

For communicating with the VAN, the sender initiates a session. The session is governed by a predefined set of commands called "VAN script," which are specific to the VAN's host computer. The functions of a VAN's script are as follows: (1) It dials into the VAN, (2) it recognizes the login name and password for allowing access, (3) it deposits EDI messages to be delivered to trading partners, and (4) it retrieves EDI messages from the mailbox.

Unfortunately, there is not a standardized set of commands for communicating with VANs. Different VANs may have different VAN scripts. Therefore, when purchasing EDI software, the user should make sure that it has the VAN script that enables the user to communicate with the available VAN services. VAN providers know this difficulty and therefore generally provide the VAN subscribers the software required for communicating with the application systems. A software vendor that offers scripts for several different EDI VANs is a desired choice for purchasing EDI software.

Multiple VAN Support

Trading partners of an EDI user may subscribe to many different VANs. Therefore, the communications software must be flexible so that it can connect to many different VANs.

Direct Trading Partner

Some trading partners may use VAN services whereas others may not. Those who are not using VAN services must be connected directly by the EDI software. For receiving messages from these direct trading partners, a dedicated computer system is required, as no VAN exists to provide storage or message-forwarding capabilities.

Script Building Tool

In some cases, a trading partner may have to connect to a VAN or to a mainframe computer for which no communication script is available. EDI software that has capabilities of building scripts can help in such situations by creating custom scripts for connecting to other VANs or directly to mainframes.

Communication Audit Trails

Communication audit trails can be used for verification that a transaction was communicated among trading partners. An audit trail may include the following:

- times and dates of communication,
- identifiers,
- acknowledgements, and
- errors encountered, if any, and others.

Viewing Utility

Large amounts of information are generated in EDI processes such as audit trails, configuration data, functional acknowledgements, and others. Manually viewing or editing all these data may be cumbersome. Viewing utilities help in viewing various aspects of communication data.

Installation, Maintenance and Support

Several of the following functions are essential to install and maintain EDI software. Automated installation routines make it easier to install EDI software and to update periodically. EDI software has to keep pace with the changes in standards and versions. Tracing facilities in the software provide a trace or show the way a transaction is processed. It helps in debugging translator software. Logging functions provide the ability to maintain a computerized log of all data interchanges and, therefore, provide an audit trail. Need may exist to permanently store some data interchanges among trading partners for a long period of time. The archiving function helps in this long-term storage of data, either in regular format or compressed format. Over a period of time, a lot of data from interchanges may accumulate. Automated purging utilities provide the ability to automatically purge data based on some criteria such as starting and ending dates, particular partner, specific item, and others. As a result of power failure or other reasons, the EDI process may fail during transaction interchange. Data recovery and restart utilities automatically recover the data and retransmit transactions that were not completed beacuse of the earlier failure.

EDI COMMUNICATION NETWORK

EDI needs a communication network that will transmit, receive, and store EDI messages and transactions so that the entire communication process is fully automated. These networks can be classified as follows: (1) VANs and value-added services (VASs), (2) Internet, and (3) direct dedicated communications.

VAN

VAN is a store-and-forward mechanism for exchanging business transactions. VAN performs EDI requirements as VAN acts as the communication facilitator that provides the function of transmitting, receiving, and storing messages (see Fig. 4). The easiest way to start communicating

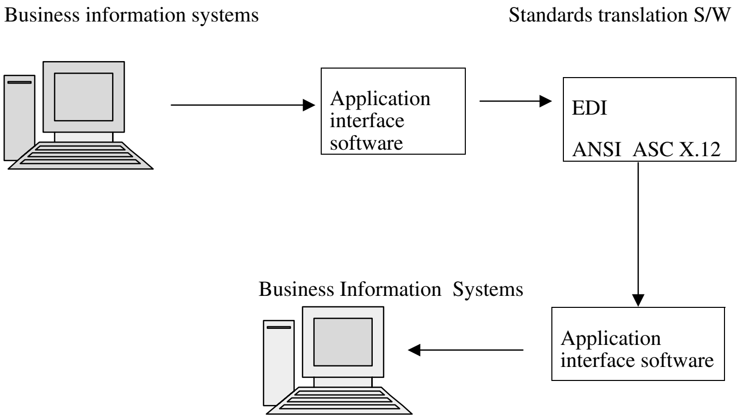

Business information systems
Standards translation S/W
Application interface software
EDI
ANSI ASC X.12
Business Information Systems
Application interface software

Figure 4. Commercial and value-added network.

with the trading partners is to subscribe to a VAN. A VAN operator provides the EDI communication expertise and equipment necessary for electronic communication. VAN providers also provide VASs such as consulting and training in the mapping of EDI transactions, coding VAN communication script, on-site EDI software and hardware installation, and others.

VANs are the most widely used communication networks for EDI communication. Increased competition among the VAN providers has resulted in low prices for VAN services, which has facilitated organizations to outsource the delivery of data and message services. In an increasingly competitive marketplace that demands fast responses to customer needs, an organization may ask the following question: Why struggle single-handedly trying to support national and international voice and data traffic when VAN service providers are ready to assume those responsibilities at very competitive prices? VAN services provide the current technology, economies of scale, customer service fault management, and so on. VAN provides a single communications access point, 24-hour access and support, control reports on EDI traffic, and reliability of services. Advantages of VANs are as follows (1–3):

- VAN is generally available throughout the day, 24 hours a day.
- Any trading partner is just a call away to VAN.
- VAN provides a mailbox capability; that is, messages are routed, stored, and forwarded any time of the day.
- VAN capabilities are available irrespective of geographical location or time.
- VANs support different speeds and protocols.
- VANs provide reliable connectivity to trading partners.
- VANs provide security for transactions.

Users can schedule when the VAN scripts is executed. Execution of VAN scripts can be automated or manual. Automated execution is the preferred way. In a manual system, the communications process will have to be started manually whenever desired. With manual control, the communications errors can be noted and corrected in real-time.

There are several requirements that a VAN must fulfill before it can be used (1–3):

1. A VAN must support the protocol (asynchronous or bisynchronous) being used by the communication software. Some VANs may not support the X.25 protocol.
2. A VAN must support the standards such as ANSI ASC X12, UN/EDIFACT, or industry-specific TDCC, VICS, and so on.
3. No conflict should exist in the data segment and data element delimiters used by the trading partners and the VAN.
4. A VAN should support the access method desired by the user, such as dial-up lines, leased lines, and so forth.
5. Data backup and recovery functions must be available.
6. Data security features should provide transmission status reports and usage accounting data.
7. Transmission timing should be short.
8. Additional VASs must be provided.

Support by VAN Service Providers

Support is essential for someone who has just bought EDI software. Users need guidance in installation, maintenance, and use of any new EDI software. Such user support can be provided both by the software and the vendors. For example:

- user documentation provides narrative text concerning the daily use of the EDI software,
- technical documentation,
- help success,
- online tutorial,
- vendor services,

- training, and
- user groups.

Internet

The Internet provides the retailers and other businesses with the ability to communicate business documents electronically. The Internet provides a more convenient form of business communication. These online business transactions are more efficient and flexible. As no intermediary is involved, the cost of business transactions using the Internet is lower compared with VAN-assisted electronic commerce. With the growth in Internet and related services, it has become possible for retailers to access a worldwide network of customers. VANs, as compared with the Internet's worldwide connectivity, have very limited connectivity to only a few thousand other paying subscribers. The Internet also provides interactive capabilities rather than just store-and-forward functions provided by VANs. These interactive functions provide browsing abilities to users and help retailers to market their products to a much larger audience. One major problem with the Internet is security, which is discussed in a later section.

Direct Dedicated Connections

There are many transmission and switching mechanisms that can make it feasible to have direct dedicated connection. Synchronous digital hierarchy, frame relays, and asynchronous transfer mode provide the potential for direct partner interface, mainly from LAN to LAN.

HARDWARE REQUIREMENTS

For operating the EDI software, communication software, and application systems, a business needs workstations, servers, and mainframe computers. For communicating with other organizations, LAN, WAN, intranets, Internets, and other networks are needed. Routing devices such as gateways, bridges, routers, brouters, and others are needed for packet, message, or circuit switching. The detailed explanation of these hardware devices, network management devices, switching mechanisms, and communication protocols are beyond the scope of this article.

SECURITY

EDI demands that an organization become a part of the network. Once an organization becomes a part of a network, it faces challenges from unauthorized intruders and hackers. A list of control activities is provided to ensure that interchange of data takes place while maintaining the integrity of the computer systems.

Access Control

Access controls are required at initiation, transmission, and destination. These controls can be achieved by using password, user ID, storage lockout, and different levels of storage and function access.

Data Integrity

Authentication, acknowledgement protocol, computerized log, digital signatures, and edit checks can be used for detecting errors during the process of input or transmission. Authentication, integrity, confidentiality, and nonrepudiation can be achieved through public key cryptosystems that employ digital signature, encryption, and key exchange technologies. Nonrepudiation can be accomplished through the use of certification authority. Upon user authentication, traditional access control or role-based access control methods can be employed to define access rights. For security, many competing algorithms exist and may give rise to interoperability problems.

Digital certificates, electronic forms that encrypt and authenticate both ends of the same transaction, are crucial in enabling EDI over the Internet. They provide the level of security EDI users are accustomed to with existing VAN service providers. Digital certificates exist that are compatible with the standard ANSI ASC X12 data types. The Internet could prove to be a much simpler and cheaper transmission medium for EDI than VANs if adequate security is developed.

Transaction Completeness

To avoid loss or duplication of a transaction during transmission, one can use batch totaling, sequential numbering, and one-to-one checking against the control file.

Availability

Viruses, Trojan horses, programming errors, and hardware and software errors may interrupt the availability of EDI systems. One can use anti-virus packages to prevent viruses. By planning, developing, installing, and operating error-free software, one can eliminate the problems of Trojan horses, viruses, and other software errors that lead to interruption of services. Fault-tolerant systems including off-site backup, redundant arrays of independent disks (RAID), disk mirroring, tandem computers, and other techniques help in avoiding interruption because of sabotage or natural causes.

SUMMARY

EDI is being used for accelerating the flow of business transactions among business partners. Advances in computer and communication technologies have made it possible to create transactions in a few minutes and transmit them to trading partners in seconds. Standardization must exist among transaction formats for computerized communication to take place between application systems of different organizations. ANSI ASC X12 and EDIFACT are two dominant formats for domestic and international interchanges, respectively. The output of the sender's application system is sent to the receiver's application system with the help of application interface, standard translation software, and communication software. Understanding of the different components and their integration requirements helps in the successful implementation of EDI. Such a successful implementation reduces transaction costs, provides flexibility, and improves the competitive advantage.

ACKNOWLEDGMENTS

This article is based on the fundamental concepts explained in Guidelines for the Evaluation of Electronic Data Interchange Products and Electronic Data Interchange (1,2). The framework of this article and many details are repeated from these documents.

BIBLIOGRAPHY

1. J. J. Garguilo and P. Markowitz, *Guidelines for the Evaluation of Electronic Data Interchange Products*, Gaithosburg, MD: National Institute of Standards and Technology. Available: http://www.snad.ncls.gov/.
2. Anonymous, *Electronic Data Interchange*, National Institute of EDI. Available: http://www.fie.com/web/era/introedi/index.html/.
3. Anonymous, *Your Introduction to Electronic Commerce*, Business Handbook. Available: http://ch5.htm at net.gap.net/.

Rajesh Aggarwal
Middle Tennessee State University
Murfreesboro, Tennessee

FAILURE DETECTORS FOR ASYNCHRONOUS DISTRIBUTED SYSTEMS: AN INTRODUCTION

WHY FAILURE DETECTORS?

To stop waiting or not to stop waiting? That is the question! Asynchronous distributed systems are characterized by the fact that there is no bound on the time it takes for a process to execute a computation step, or for a message to go from its sender to its receiver. This is why these systems are usually called *"time-free"* systems. The major part of the software that addresses non-real-time problems implicitly considers a time-free underlying system. This has several advantages. The main one is "generality." As it is does not require that the underlying system satisfies specific timing assumptions, the software can be safely executed on any system. Moreover, the understanding and the correctness proof are usually easier as they do not rest on particular timing assumptions.

Unfortunately, the previous advantage can become useless as soon as there are failures. Assuming there is a process per node (processor), let us consider the case where a node can crash. The problem is then for a process p to know whether another process q has or has not crashed. The bad news is that the combination of crashes and asynchrony creates a context where p has no safe means to know whether q has or has not crashed. If, thinking q has crashed, p stops waiting from q after some time, it can be wrong as maybe q has not crashed and the message from q to p is only very slow. If, after it stops waiting from q and before it gets q's message, p takes an irrevocable decision (motivated by the fact that it thinks that q has crashed), this decision is wrong (and the safety property of the upper layer application can consequently be violated[1]). On the other side, let us assume that q has crashed. To prevent the bad previous scenario from occurring, p must wait until it gets q's message. It is easy to see that p will wait forever, and the liveness property of the application will never be satisfied. This is one of the main problems we are faced with when designing fault-tolerant distributed algorithms in asynchronous systems prone to failures (2).

Do the same as ancient Greeks did: Ask an oracle! To solve the previous dilemma, Chandra and Toueg have introduced and investigated the notion of failure detectors (3). A failure detector can be seen as a distributed oracle related to the detection of failures.[2] Such oracles do not change the pattern of failures that affect the execution in which they are used. Their essential characteristic is related to the guess they provide about failures. As defined by Chandra and Toueg (3), a failure detector class is basically defined by two properties, namely, a completeness property and an accuracy property. *Completeness* is on the actual detection of failures, while *accuracy* restricts the mistakes a failure detector can make.

Why use failure detectors? There are several good reasons for using a failure detector. One lies in the design approach it favors. More precisely, a failure detector is not defined in terms of a particular implementation (involving network topology, message delays, local clocks, etc.) but in terms of abstract properties (related to the detection of failures) that allow problems to be solved despite process crashes. Thus, the failure detector approach allows a modular decomposition that not only simplifies protocol design but also provides general solutions. More specifically, during a first step, a protocol is designed and proved correct assuming only the properties provided by a failure detector class. So, this protocol is not expressed in terms of low-level parameters, but it depends only on a well-defined set of abstract properties. The implementation of a failure detector *FD* of the assumed class can then be addressed independently. Additional assumptions can be investigated, and the ones that are sufficient to implement *FD* can be added to the underlying distributed system in order to get an augmented system on top of which *FD* can be implemented. In that way, *FD* can be implemented in one way in some context and in another way in another context, according to the particular features of the underlying system. It follows that this layered approach favors the design, the proof, and the portability of protocols.

Another important advantage of failure detectors lies in the approach they promote to address problems that are impossible to solve in time-free asynchronous distributed systems prone to failures. One of the most famous of them is the consensus problem that cannot be solved in asynchronous systems as soon as a process can crash (5).[3] When faced with a problem *Pb* that cannot be solved in an asynchronous system prone to failures, a natural and fundamental question that comes to mind is as follows:

> *Which is the weakest failure detector FD_{min}(Pb) the underlying asynchronous system has to be equipped with in order for the problem Pb to be solved?*

Answering this question is important from both a practical and a theoretical point of view. From a theoretical

[1] A problem can be defined with a *safety* and a *liveness* property. The safety property stipulates that "nothing bad ever happens," while the liveness property stipulates that "something good eventually happens" (1).

[2] Let us notice that the *oracle* notion has first been introduced in language theory. An oracle is a *language* whose words can be recognized in one step from a particular state of a Turing machine (4). The main characteristic of such oracles is to hide in a single "observed" step a sequence of computation steps or the use of an uncomputable function. They have been used to provide hierarchies of problems with respect to complexity or computability.

[3] Failure detectors have initially been introduced to cope with the impossibility to solve consensus in time-free systems prone to process crashes (3).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

point of view, the properties defining such a weakest failure detector state the necessary and sufficient conditions under which the problem Pb can be solved. This means that $FD_{min}(Pb)$ defines the borderline beyond which Pb cannot be solved. More precisely, Pb can be solved only in asynchronous systems enriched with additional mechanisms able to implement $FD_{min}(Pb)$. This has an immediate practical consequence. To solve Pb, we need a system where the $FD_{min}(Pb)$ properties can be implemented.[4] Of course, if the system we are provided with satisfies stronger properties, Pb can be solved. This means that when a (provably correct) protocol does not work in a system, it is only because the properties assumed by this protocol are not satisfied by the underlying system. (Let us notice that the same argument applies to hierarchies of failure modes (6).)

Failure detectors further a deeper understanding of distributed computing problems in the presence of failures, in the sense that they allow us to know whether a problem Pb_1 is “more difficult” to solve than a problem Pb_2 or whether Pb_1 and Pb_2 require different kinds of assumptions. Pb_1 is said to be more difficult to solve than Pb_2 if it requires that the underlying system satisfies additional assumptions not necessary for solving Pb_2; that is, formally, $FD_{min}(Pb_1) \Rightarrow FD_{min}(Pb_2)$. If we have neither $FD_{min}(Pb_1) \Rightarrow FD_{min}(Pb_2)$ nor $FD_{min}(Pb_2) \Rightarrow FD_{min}(Pb_1)$, Pb_1 and Pb_2 are incomparable, which means that they require underlying systems with different properties in order for them to be solved.

Can failure detectors be implemented? Guided by practical motivations, we only consider failure detectors that cannot guess the future. Those failure detectors have been called *realistic* (7). They actually do correspond to the failure detectors that can be implemented in a synchronous system (i.e., a system with known upper bounds on both message delays and the time it requires for a process to execute a step).

So, a simple way to implement a failure detector is to use an underlying synchronous system as an additional subsystem. This subsystem is only used to implement the required failure detector and is not directly accessible by the processes. They do not ever know the existence of it. They evolve in a computation model defined by an asynchronous system enriched with the appropriate failure detector.

As we will see later, it is possible to solve some problems with *eventually accurate* failure detectors. Those failure detectors are assumed to satisfy their accuracy property (restriction on the mistakes they can make) only after some finite but unknown time. It appears that the use of such failure detectors generally requires a majority of correct processes (this constraint can be seen as the price that has to be paid to cope with *eventual* accuracy).

Interestingly, the use of eventually accurate failure detectors allows the design of *indulgent* algorithms (8), i.e., algorithms that never violate their safety property, whatever the behavior of the failure detector they use. This means that, if the failure detector never meets its accuracy property, these algorithms cannot terminate, but if they terminate, they terminate correctly. Such eventually accurate failure detectors are very interesting for a simple reason. They have best effort implementations in asynchronous systems; namely, these implementations provide failure detector outputs that a priori can only be considered as *approximate* outputs. But, very interestingly, when the underlying system behaves synchronously during a long enough period, the outputs are no longer approximate but become correct. The periods during which the underlying asynchronous system behaves synchronously are usually called *“stable”* periods. An interesting consequence is that, as we can see, the protocols implementing such failure detectors can run concurrently with the application processes on the same asynchronous system. This is practically relevant.

It is important to notice that there is no “magic” behind a failure detector. This means that, if a failure detector allows solving an otherwise impossible problem P in a given computation model M, then that failure detector cannot be implemented in the model M. The impossibility to solve P in the model M is not circumvented; it is only moved “inside” the failure detector.

Due to its inherent modularity, a main advantage of the failure detector approach is the very clean separation between the properties offered by a failure detector and the machine/network-dependent requirements that allow for implementation of it (those encapsulate low-level synchrony requirements that can change from one implementation to another one).

ASYNCHRONOUS SYSTEM MODELS

Process model. We consider a system consisting of a finite set of n processes $\Pi = \{p_1, p_2, \ldots, p_n\}$. A process can fail by crashing, i.e., prematurely halting. It behaves correctly (i.e., according to its specification) until it (possibly) crashes. By definition a process is *correct* (during a run) if it does not crash (during that run); otherwise, it is *faulty*. There is no assumption on the time it takes for a (non-crashed) process to execute a step. In the following, $t \leq n - 1$ denotes the maximum number of processes that can crash, and $f \leq t$ the actual number of process crashes during a given run.

Communication model. Processes communicate and synchronize by exchanging messages through links. Every pair of processes is connected by a link. We consider two types of links.

- The link connecting p_i to p_j is *reliable* if it does not create or duplicate messages, and every message sent by p_i to p_j is eventually received by p_j (if p_j is correct).
- The link connecting p_i to p_j is *fair lossy* if, while it does not create or duplicate messages, it can lose messages, but if p_i sends an infinite number of messages to p_j and p_j executes receive actions infinitely often, then it receives an infinite number of messages from p_i.

A process p_i sends a message m to a process p_j by invoking *“send* (m) *to* p_j*”*; p_j receives it when it terminates the invocation of “*receive*( )”. The *send*( ) and *receive*( ) primitives are provided by the underlying communication

[4] Or (in some cases) approximated, as we will see later.

network. The notation *"broadcast* (m)*"* is used as a shortcut for "**forall** $j \in \{1, \ldots, n\}$ **do** *send* (m) to p_j **enddo**". If p_i crashes while executing "*send* (m) *to* p_j", either m is sent or m is not sent at all (i.e., *send*( ) is atomic, while *broadcast*( ) is not.)

Computation models. In the following we consider two types of asynchronous computation models:

- The FLP computation model that considers crash-prone processes and reliable links.[5]
- The FLL computation model that considers crash-prone processes and fair lossy links.

SOLVING CONSENSUS

The Consensus Problem

The *consensus* problem is a paradigm of agreement problems. It appears, in one form or another, as soon as processes have to agree, e.g., on a common action to execute, on the same decision to take, etc. A well-known example of where consensus appears is *atomic broadcast.* That problem, which appears as a basic software layer in a lot of replication-based, fault-tolerant distributed systems, requires that the correct processes deliver the same set of messages in the same order. So, it is at the same time a communication problem (all the correct processes have to deliver the same set of broadcast messages), and a consensus problem (as they have to deliver them in the same order) (3). So, in the consensus problem, every correct process p_i *proposes* a value v_i and all correct processes have to *decide* on some value v, in relation to the set of proposed values. More precisely, the *consensus problem* is defined by the following three properties (3,5):

- C-Termination: Every correct process eventually decides on some value.
- C-Validity: If a process decides v, then v was proposed by some process.
- C-Agreement: No two correct processes decide differently.

The agreement property applies only to correct processes. So, it is possible that a process decides on a distinct value just before crashing. *Uniform consensus* prevents such a possibility. It has the same Termination and Validity properties plus the following agreement property:

- C-Uniform Agreement: No two processes (correct or not) decide differently.

In the following we consider the uniform consensus problem.

An Eventually Accurate Failure Detector

As indicated in the beginning of the article, the consensus problem cannot be solved in asynchronous systems prone to even a single process failure (5). It is to circumvent this impossibility that Chandra and Toueg proposed the failure detector concept (3). Among the several classes of failure detectors they have proposed, the one denoted $\diamond S$ has been shown to be the weakest to solve consensus (9). Each process p_i is equipped with a local failure detector module that provides it with a set *suspected*$_i$; p_i can only read this set that contains the identities of the processes that are currently suspected to have crashed. Any failure detector module is inherently unreliable. It can make mistakes by not suspecting a crashed process or by erroneously suspecting a correct one. Moreover, suspicions are not necessarily stable. A process p_j can be added to or removed from a set *suspected*$_i$ according to whether p_i's failure detector module currently suspects p_j. We say "process p_i suspects process p_j" at some time, if at that time we have $p_j \in suspected_i$.

To be useful a failure detector class has to satisfy some properties, and those have to be as weak as possible while allowing the problem of interest to be solved. The class $\diamond S$ includes all the failure detectors satisfying the following properties:

- Strong Completeness: Eventually, every process that crashes is permanently suspected by every correct process.
- Eventual Weak Accuracy: There is a time after which some correct process is never suspected by the correct processes.

The implementation of failure detectors of the class $\diamond S$ has been addressed in Refs. 10–14. As noted, all these implementations assume that the underlying system is eventually *stable.* If the stability assumption is satisfied during a long enough period, the sets *suspected*$_i$ satisfy the properties defining $\diamond S$. "Long enough" means here "a duration allowing the protocol using the failure detector to terminate."

A $\diamond S$-based Consensus Protocol

The protocol that follows considers the FLP model. It is indulgent, so it enjoys the nice property of never violating consensus safety (validity and uniform agreement), whatever the sequence of (correct or bad) values read from the *suspected*$_i$ sets; moreover, it terminates (at least) when these sets contain correct values during a long enough period (namely, the period during which the consensus protocol needs the failure detector).

The protocol presented in Fig. 1 is a particular instance of the generic protocol introduced in Ref. 15. It requires a majority of correct processes ($t < n/2$), which has been shown to be a necessary requirement for indulgent protocols (8). Its principles are surprisingly simple. The processes (p_i) proceed by asynchronous consecutive rounds (r_i). Each round r is coordinated by a process p_c such that $c = (r \bmod n) + 1$ (hence, if the round number never stops increasing, each process is ensured to be the coordinator of a future round).

[5] The name "FLP" is coined from the first letters of Fischer, Lynch, and Paterson who proved the impossibility of solving consensus in this system model (5). This abbreviation is of general use in the literature.

```
Function Consensus($v_i$)

Task $T1$:
(1)  $r_i \leftarrow 0$; $est_i \leftarrow v_i$;
(2)  while $true$ do
(3)      $c \leftarrow (r_i \bmod n) + 1$; $r_i \leftarrow r_i + 1$; % $1 \le r_i < +\infty$ %

         ——— Phase 1 of round $r$: from $p_c$ to all ———
(4)      if $(i = c)$ then broadcast PHASE1($r_i$, $est_i$) endif;
(5)      wait until (PHASE1($r_i$, $v$) has been received from $p_c \vee c \in suspected_i$);
(6)      if (PHASE1($r_i$, $v$) received from $p_c$) then $aux_i \leftarrow v$ else $aux_i \leftarrow \perp$ endif;

         ——— Phase 2 of round $r$: from all to all ———
(7)      broadcast PHASE2($r_i$, $aux_i$);
(8)      wait until (PHASE2 ($r_i$, $aux$) msgs have been received from a majority of proc.);
(9)      let $rec_i$ be the set of values received by $p_i$ at line 8;
         % We have $rec_i = \{v\}$, or $rec_i = \{v, \perp\}$, or $rec_i = \{\perp\}$ where $v = est_c$ %
(10)     case $rec_i = \{v\}$       then $est_i \leftarrow v$; broadcast DECISION($est_i$); stop $T1$
(11)          $rec_i = \{v, \perp\}$ then $est_i \leftarrow v$
(12)          $rec_i = \{\perp\}$    then skip
(13)     endcase
(14) endwhile

Task $T2$: when DECISION($est$) is received: broadcast DECISION($est_i$); return($est$)
```

Figure 1. A simple $\diamond S$-based consensus protocol ($t < n/2$) (15).

Let v_i be the value initially proposed by p_i. The local variable est_i represents p_i's estimate of the decision value. During a round r, its coordinator p_c tries to impose its current estimate as the decision value. To attain this goal, a round is made up of two phases. During the first phase, (1) p_c sends est_c to all the processes (line 4), and (2) any process p_i waits until it receives p_c's estimate or suspects it (line 5). According to the result of its waiting, a process p_i sets a local variable aux_i to the received value $v = est_c$, or sets it to a default value $\perp$ (line 6). It is important to notice that due to the completeness property of the underlying failure detector, no process can block forever at line 5.

Then, the processes start the second phase of round r, during which they exchange the values of their aux_i variables (line 7). Let us observe that, due to the "majority of correct processes" assumption, no process can block forever at line 8. Moreover, it is important to notice that only two values can be exchanged: $v = est_c$ or $\perp$. Consequently, the set rec_i of values received by a process p_i can only have the values $\{v\}$, $\{v, \perp\}$, or $\{\perp\}$. Moreover, due to the "majority of correct processes" assumption, it is impossible for two sets rec_i and rec_j to be such that $rec_i = \{v\}$ and $rec_j = \{\perp\}$, so we also have the following invariant (for each pair of processes p_i and p_j, that have not crashed):

$$rec_i = \{v\} \Rightarrow (\forall\, p_j : (rec_j = \{v\}) \vee (rec_j = \{v, \perp\}))$$
$$rec_i = \{\perp\} \Rightarrow (\forall\, p_j : (rec_j = \{\perp\}) \vee (rec_j = \{v, \perp\}))$$

This invariant dictates the behavior of p_i:

- $rec_i = \{v\}$ (line 10). In this case, p_i decides the value v. It can safely do so, since in this case, a process that does not decide adopts v as its new estimate value. Moreover, to prevent possible deadlock situations, p_i broadcasts its decision value.
- $rec_i = \{v, \perp\}$: (line 11). In this case, consistently with the previous item, p_i adopts v as its new estimate value, and proceeds to the next round.
- $rec_i = \{\perp\}$ (line 12). In this case, p_i proceeds to the next round without modifying est_i.

The proof that this $\diamond S$-based consensus protocol is correct is relatively easy. It is left to the reader (who can also find it in Ref. 15). The strong completeness property is used to show that the protocol never blocks. The eventual weak accuracy property is used to ensure termination (there will be a round coordinated by a correct nonsuspected process). The majority of correct processes are used to prove consensus agreement.

Other $\diamond S$-based consensus protocols can be found in Refs. 3,16–18.

Interactive Consistency

This problem has first been introduced in the context of synchronous systems where some processes can behave in a Byzantine way (19). Here we consider the interactive consistency problem in the FLP model.

This problem is harder than consensus in the following sense: The processes have to agree not on a proposed value but on the *vector* of proposed values. So, each process p_i proposes a value v_i and has to decide a vector D_i such that the following properties are satisfied (we consider here the *uniform* version of the problem):

- IC-Termination: Every correct process eventually decides on a vector.
- IC-Validity: Any decided vector D is such that $D[i] \in \{v_i, \perp\}$, and is v_i if p_i does not crash.
- IC-Agreement: No two processes decide differently.

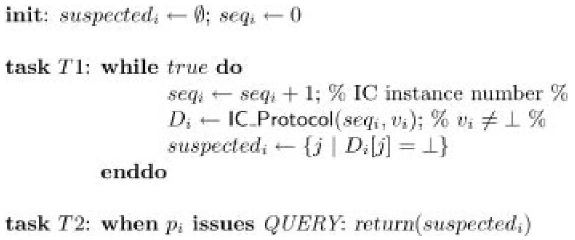
It is shown in Ref. 20 that the weakest failure detector class that allows the interactive consistency problem to be solved in the FLP model is the class of perfect failure detectors. This class, denoted $\mathcal{P}$, contains all the failure detectors that satisfy the following properties (3):

- Strong Completeness: Eventually, every process that crashes is permanently suspected by every correct process.
- Strong Accuracy: No process is suspected before it crashes.

As we can see, a perfect failure detector never makes mistakes. A $\mathcal{P}$-based interactive consistency protocol is described in Ref. 21. Interestingly, this protocol that proceeds by consecutive asynchronous rounds, is as efficient as the "best" synchronous interactive consistency protocol ("best" from a time complexity point of view, i.e., when we count the maximum number of rounds that are required, namely, min($f + 2$, $t + 1$, n)).

While consensus can be solved in the FLP model (with a majority of correct processes) equipped with $\diamond S$, it is not possible, assuming a solution to the consensus problem, to design a protocol building a failure detector of $\diamond S$. On the contrary, we show here that, in the FLP model, the construction of a perfect failure detector and interactive consistency are equivalent problems in the sense that one can solve either of them as soon as we are provided with a solution to the other.

Interactive consistency protocols based on a perfect failure detector are described in Refs. 20 and 21. A protocol



```
init: $suspected_i \leftarrow \emptyset$; $seq_i \leftarrow 0$

task $T1$: while $true$ do
        $seq_i \leftarrow seq_i + 1$; % IC instance number %
        $D_i \leftarrow$ IC_Protocol($seq_i, v_i$); % $v_i \neq \perp$ %
        $suspected_i \leftarrow \{j \mid D_i[j] = \perp\}$
    enddo

task $T2$: when $p_i$ issues $QUERY$: $return(suspected_i)$
```

Figure 2. From interactive consistency to a perfect failure detector (20).

providing the inverse construction is described here (Fig. 2). Assuming a solution to the interactive consistency problem (subroutine protocol called IC_Protocol(x,v)), this protocol implements a perfect failure detector. The protocol consists of two tasks and is very simple.[6] Task $T1$ repeatedly invokes the interactive consistency protocol and suspects a process p_j as soon as the output D_i returned by an invocation is such that $D_i[j] = \perp$. Task $T2$ processes the queries issued by the upper layer. It returns the current value of *suspected*$_i$. The reader can easily check that the sets *suspected*$_i$ satisfy strong completeness and strong accuracy.

SOLVING NON-BLOCKING ATOMIC COMMIT

The Non-Blocking Atomic Commit Problem

Originated from databases, the *non-blocking atomic commit* problem (NBAC) is certainly one of the oldest agreement problems encountered in distributed computing. According to its local state, each process first issues a vote (*yes* or *no*). Then, according to the set of votes and the fact that some processes possibly crashed, the non-crashed processes have to decide on a single value, namely, *commit* or *abort*. More precisely, the problem is defined by the following properties:

- NBAC-Termination: Every correct process eventually decides.
- NBAC-Validity: A decided value is *commit* or *abort*. Moreover:
    —NBAC-Justification: If a process decides *commit,* all processes have voted *yes*.
    —NBAC-Obligation: If all processes vote *yes* and there is no crash, then the decision value is *commit*.
- NBAC-Agreement: No two processes decide differently.

It is easy to see that the justification property relates the *commit* decision to the *yes* votes, while the obligation property eliminates the trivial and useless solution where all processes would always decide *abort*. Actually, this property defines what is a "good" run. It is a run in which all the processes want to commit (they voted *yes)* and the environment behaves correctly (no process crashes). The decision can only be *commit* in good runs.

As the reader can see, a major difference between the specification of consensus and the specification of NBAC lies in the fact that the latter mentions explicitly process crashes occurring during a protocol execution.

An Appropriate Failure Detector

Solving NBAC in the FLP model requires the model to be enriched with appropriate failure detectors. Such failure detectors are studied and investigated in Refs. 22 and 23. We consider here *timeless* failure detectors, i.e., failure detectors that do not provide information on when exactly (in the sense of global time) failures occurred. (Let us notice that $\mathcal{P}$ and $\diamond S$ define classes of timeless failure detectors.)

To address this question, a failure detector class, denoted $?\mathcal{P}$ and called the class of *anonymously perfect* failure detectors, has been proposed in Ref. 23. This class is defined as follows:

- Anonymous completeness: If a crash occurs, eventually every correct process is permanently informed that some crash occurred.
- Anonymous accuracy: No crash is detected unless some process crashed.

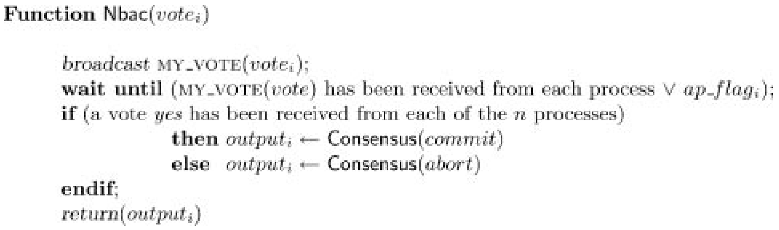
The class of failure detectors denoted $?\mathcal{P}+\diamond S$ includes all the failure detectors that satisfy both $?\mathcal{P}$ and $\diamond S$. In the following, a failure detector module of that class is represented at p_i as a boolean variable *ap_ flag*$_i$ ("approximate flag") that is *true* iff a crash is detected.

Figure 3 describes an NBAC protocol based on a failure detector of $?\mathcal{P} + \diamond S$. This protocol actually reduces NBAC to consensus (which, as we have seen, can be solved in the FLP model enriched with $\diamond S$ when $t < n/2$). The protocol is pretty simple. From a methodological point of view, it is interesting to see how each of $?\mathcal{P}$ and $\diamond S$ are used: the first is for ensuring the validity of NBAC, the second to be able to use the subroutine consensus protocol.

The weakest failure detector to solve the NBAC problem has been investigated and solved in Ref. 22. That failure detector, denoted $(\Psi, ?\mathcal{P})$, has a hierarchical structure. It is made up of two components (ψ and $?\mathcal{P}$), which means that at any time it outputs a pair of values, one related to ψ, the other one related to $?\mathcal{P}$. The failure detector ψ is in turn composed of three failure detector components Ω, Σ (see below), and a new instance of $?\mathcal{P}$. It initially outputs the default value $\perp$, and after some finite time behaves as the pair (Ω, Σ) at all processes, or (only in case a failure has previously occurred) it *may* behave as $?\mathcal{P}$ at all processes. (The switch from $\perp$ to (Ω, Σ) or $?\mathcal{P}$ need not occur simultaneously at all processes, but the same choice is made at all processes. It is important to note that the choice from $\perp$ to $?\mathcal{P}$ is allowable only if a crash has occurred. Moreover, if a failure occurs, any of the switches from $\perp$ to $?\mathcal{P}$ or (Ω, Σ) can occur.[7])

[6] The difficult construction is the other one: solving interactive consistency from a perfect failure detector (20,21).

[7] This means that, after a finite time, $(\Psi, ?\mathcal{P})$ behaves as $(?\mathcal{P}, ?\mathcal{P})$ or as $((\Omega, \Sigma), ?\mathcal{P})$.



```
Function Nbac($vote_i$)

    broadcast MY_VOTE($vote_i$);
    wait until (MY_VOTE($vote$) has been received from each process $\vee$ $ap\_flag_i$);
    if (a vote $yes$ has been received from each of the $n$ processes)
            then $output_i \leftarrow$ Consensus($commit$)
            else $output_i \leftarrow$ Consensus($abort$)
    endif;
    return($output_i$)
```

Figure 3. A simple $?\mathcal{P} + \diamond S$-based NBAC protocol ($t < n/2$) (23).

.

IMPLEMENTING QUIESCENT COMMUNICATION

This section continues our visit with the failure detector concept by considering the problem that consists of implementing *quiescent communication* despite process crashes and fair lossy links, i.e., in the FLL model (24).

The Quiescence Problem

Considering two processes p_i and p_j that do not crash connected by a fair lossy link, a basic communication problem consists in building a reliable link on top of that fair lossy link. This problem is well known, and basic mechanisms such as retransmission and acknowledgments allow it to be solved. Retransmission allows message losses to be tolerated, while acknowledgments allow their retransmission to be eventually stopped. When a receiver receives a message *m,* it sends back *ack*(m), and for each message *m* it wants to send, the sender repeatedly resends it until it gets an *ack*(m). This simple protocol is *quiescent* in the sense that, after some time, no process sends or receives messages related to the transmission of *m.*

Let us now consider the case where the receiver p_j can crash. In this case, it is possible that p_j crashes before receiving *m* and the sender will consequently send copies of *m* forever. The protocol is no longer quiescent. So, the problem is to provide quiescent implementations of communication primitives in the FLL model. This problem is addressed and solved in Ref. 24. This article first shows that the *quiescent communication* problem cannot be solved in a pure FLL model. To solve it, that model has to be appropriately enriched with a failure detector. This is not at all counter-intuitive since, to stop retransmitting a message, the sender has to know—in a way or another—whether the receiver has crashed.

Ref. 24 shows that the weakest class of failure detectors solving the quiescent communication problem is the class of eventually perfect failure detectors, i.e., the failure detectors that, after some unknown but finite time, suspect all the crashed processes and only them. Unfortunately, such a failure detector cannot be implemented in FLL. So, the authors investigated another class of implementable failure detectors capable of providing quiescent communication protocols. They called it the class of *heartbeat* failure detectors.

A Heartbeat Failure Detector

A heartbeat failure detector outputs at each process p_i an array $HB_i[1 \cdots n]$ of non-decreasing counters satisfying the following properties:

- HB-completeness: If p_j crashes, then *HB_i[j]* stops increasing.
- HB-accuracy: If p_j is correct, then *HB_i[j]* never stops increasing.

As we can see, heartbeat failure detectors can be implemented, but their implementation is not quiescent. In that sense these failure detectors allow the non-quiescent part of a communication protocol to be isolated. Moreover, the use of a heartbeat failure detector favors design modularity and eases correctness proofs. Additionally, a single heartbeat failure detector "service" can be used by several upper layer applications.

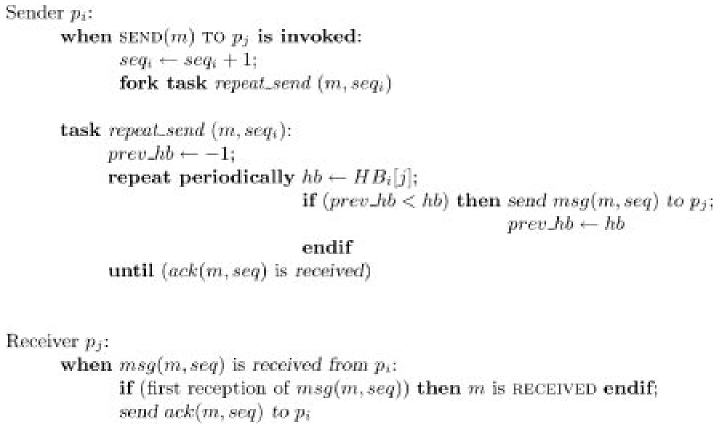
A Quiescent Implementation

Figure 4 presents a quiescent protocol providing a reliable link in an FLL system model equipped with a heartbeat failure detector. The protocol on the sender side provides an implementation of the SEND() primitive invoked by the upper layer application. To that end, it uses the send() primitive provided by the underlying communication layer. Similarly, on the receiver side, RECEIVE() notifies the upper layer that a new message has arrived, while receive() is used to receive a message from the underlying communication layer. The protocol is particularly simple and self-explanatory. The *seq_i* variable (initialized to 0) is used by the sender as a sequence number generator. It is easy to see that, after some unknown but finite time, p_i either receives *ack*(m) (due to the fairness of the underlying channel) or stops retransmitting as *HB_i[j]* no longer increases (when p_j has crashed).

This protocol shows an important difference between a *quiescent* protocol and a *terminating* protocol. The protocol described in Fig. 4 is quiescent as for each message *m* sent by p_i to p_j, there is a time after which no more protocol messages are exchanged. However, this protocol is not terminating. This is because, until it receives *ack*(m), p_i has no means to know whether it has to retransmit *m*. If p_j crashes and all *ack*(m) it sent before are lost, p_i will never terminate the task *repeat_send* (*m, seq_i*). The reader can observe that this is the best that can be done. This important difference is discussed in detail in Ref. 25.

Failure Detectors in Synchronous Systems

While atomic broadcast, consensus, NBAC, etc. cannot be solved in the FLP computation model, they can be solved in synchronous systems, i.e., in systems where there are



```
Sender $p_i$:
    when SEND($m$) TO $p_j$ is invoked:
        $seq_i \leftarrow seq_i + 1$;
        fork task $repeat\_send\ (m, seq_i)$

    task $repeat\_send\ (m, seq_i)$:
        $prev\_hb \leftarrow -1$;
        repeat periodically $hb \leftarrow HB_i[j]$;
                    if $(prev\_hb < hb)$ then $send\ msg(m, seq)\ to\ p_j$;
                                              $prev\_hb \leftarrow hb$
                    endif
        until $(ack(m, seq)$ is $received)$

Receiver $p_j$:
    when $msg(m, seq)$ is $received\ from\ p_i$:
        if (first reception of $msg(m, seq)$) then $m$ is RECEIVED endif;
        $send\ ack(m, seq)\ to\ p_i$
```

Figure 4. A quiescent implementation of a reliable link (24).

bounds on processing time and message transfer delay. So, failure detectors are useless in these systems from a decidability point of view. They add no computational power. Nevertheless, failure detectors suited to synchronous systems have recently been introduced. Their aim is to help design more efficient protocols, i.e., protocols with a "best case" time complexity that cannot be attained in pure synchronous systems. To illustrate this idea, we show here how *fast failure detectors* can be used to expedite consensus in synchronous systems (26).

Synchronous System Model

As previously mentioned, a synchronous systems is characterized by the existence of a bound on the time it takes to receive and process a message, and the fact that this bound is known by the processes. In order to simplify the presentation and without loss of generality, we assume in the following that local computations take no time and transfer delays are upper bounded by D. Thus, a message sent at time t is not received after $t + D$ (D-timeliness). The links are reliable (no creation, duplication, or loss). Moreover, processes have access to a common clock.

The combination of D-timeliness and no-loss properties with the possibility of process crashes makes possible the following behaviors when, at time t, a process p_i sends a message m to processes p_j and p_k. If p_i does not crash at time t, both p_j and p_k receive m by time $t + D$. However, if p crashes at time t, different scenarios are possible. Namely, it is possible that neither p_j nor p_k receives m, or that only one of them receives m (by time $t + D$) while the other does not receive it, or that both of them receive m (by time $t + D$).

Fast Failure Detectors

Such failure detectors have been introduced in Ref. 26. A *fast perfect* failure detector provides the processes p_i with sets $suspected_i$ that satisfy the following properties (where $d < D$):

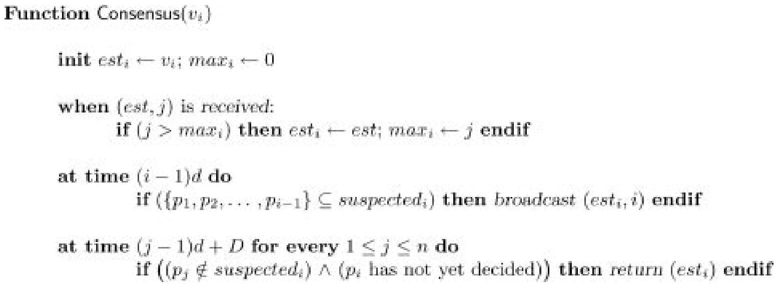
- d-Timely completeness. If a process p_j crashes at time t, then, by time $t + d$, every alive process suspects it permanently.
- Strong accuracy. No process is suspected before it crashes.

Let us observe that, if a process crashes between times t and $t + d$, then some, but not necessarily all, processes may suspect it at $t + d$.

As indicated in Ref. 26, fast failure detectors can be implemented with specialized hardware (with provides $d << D$). From a user point of view (the one in which we are interested here), they can be used to attain time complexity lower bounds that are better than what can be attained in a pure synchronous system.

To illustrate this, let us consider the fast failure detector-based synchronous consensus protocol described in Fig. 5 (26). This protocol enjoys the following early deciding property. Started at time $T = 0$, it allows the processes to decide by time $D + fd$ (let us remember that f is the actual number of process crashes). Thus, its time complexity is $D + fd$, which is much better than the best that can be done without using failure detectors, namely, $\min(f + 2, t + 1)D$.

Let us now describe in detail the behavior of a process p_i. Let us first observe that, during the time period $[0, (i - 1)d)$, p_i can only receive messages. Then, at time $(i - 1)d$, if p_i suspects all the processes with a smaller id, it sends its current estimate of the decision value (est_i) to all the processes. When a process p_i receives an estimate value *(est),* it updates its own estimate (est_i) only if it is coming from a process whose id is larger than max_i (a local variable initialized to a value smaller than any id). In that way, the successive values of est_i are coming from processes with increasing ids. Finally, at times $(j - 1)d + D$, for $j = 1, \ldots, n$, p_i decides if it trusts the corresponding process p_j. A proof of the protocol can be found in Ref. 26. It is easy to see that the processes decide by D time units when the process p_1 does not crash (in that case they decide the value v_1 proposed by p_1). If p_1 crashes while p_2 does not, they decide by time $d + D$; according to



```
Function Consensus($v_i$)

    init $est_i \leftarrow v_i$; $max_i \leftarrow 0$

    when $(est, j)$ is received:
        if $(j > max_i)$ then $est_i \leftarrow est$; $max_i \leftarrow j$ endif

    at time $(i-1)d$ do
        if $(\{p_1, p_2, \ldots, p_{i-1}\} \subseteq suspected_i)$ then broadcast $(est_i, i)$ endif

    at time $(j-1)d + D$ for every $1 \leq j \leq n$ do
        if $((p_j \notin suspected_i) \wedge (p_i$ has not yet decided$))$ then return $(est_i)$ endif
```

Figure 5. Synchronous consensus with a fast failure detector (26).

the failure pattern, the decided value is the value v_1 proposed by p_1 or the value v_2 proposed by p_2(it is the value v_1 proposed by p_1, if p_1 succeeded in sending v_1 to p_2). Etc.

ADDITIONAL REMARKS

Other problems. In addition to the previous distributed computing problems that we have shortly visited, the failure detector approach has been used to circumvent other impossibility results. We list here two of them.

The first is the construction of a *reliable atomic register* in the FLP model. It has been shown that such a construction is possible if and only if $t < n/2$. Intuitively, the majority of correct processes assumption can be used to ensure that the last value[8] of the register can always be accessed (27). A natural question is then: "Which is the weakest failure detector to implement an atomic register in the FLP model when $n/2 \leq t < n$ (i.e., when any number of processes can crash)?" This question is answered in Ref. 22 with the failure detector denoted Σ, and called the *quorum* failure detector. That failure detector outputs a set $trusted_i$ at each process p_i (the value of each of these sets can change with time). Let $trusted_i^{\tau}$ be the value of $trusted_i$ at time τ. Intuitively, $trusted_i$ contains a set of processes that p_i currently trusts as being alive. More formally, the sets $trusted_i$, $1 \leq i \leq n$, satisfy the following properties:

- Safety: $\forall i, j, \tau, \tau' : trusted_i^{\tau} \cap trusted_i^{\tau'} \neq \theta$.
- Liveness: There is a time after which, for any process p_i that has not crashed, $trusted_i$ contains only correct processes.

So, Σ provides the processes with intersecting sets, and eventually each of these sets contains only correct processes.

Σ-based algorithms building an atomic register are described in Refs. 28 and 29. (Interestingly, Σ can be built in the FLP model when $t < n/2$. This is not at all surprising as the atomic register problem can be solved in the same context without the help of a failure detector.) For the interested reader, let us mention that a protocol solving the consensus problem, despite up to $t \leq n$ crashes, in the FLP model enriched with both Σ and $\diamond S$ is described in Refs. 30.

The second problem we mention here is related to communication, namely the design of a *uniform reliable broadcast* primitive. This primitive allows the processes to broadcast messages, and it ensures that (*1*) at least the messages broadcast by the correct processes are delivered to all the correct processes, and (*2*) if a process (correct or faulty) delivers a message m, then all correct processes deliver m. So, this primitive ensures that no message from a correct process is "lost," and that no message delivered by a process is missed by a correct processes. The correct processes deliver the same set of messages, and a faulty process delivers a subset of it.

As previously mentioned, this problem can be easily solved in the FLP model when $t < n/2$ and requires additional assumptions when $n/2 \leq t < n$. The main difficulty lies in ensuring item (2). The interested reader will find in Ref. 31 an appropriate (optimal) failure detector and a uniform reliable broadcast protocol based on such a failure detector. For the interested reader, a uniform reliable broadcast protocol that additionally satisfies the quiescence property is described in Refs. 14 and 32.

Other failure detectors. Other failure detectors have been proposed. One of the most well known is the *eventual leader* failure detector, denoted Ω (9). That failure detector provides the processes with a function leader satisfying the following properties:

- Validity: Each invocation of leader returns a process name.
- Eventual Leadership: There is a time t and a correct process p such that, after t, every invocation of leader by a correct process returns p.

A failure detector of the class Ω actually provides the processes with an eventual leader election capability. But, let us notice that there is no knowledge of when the leader is elected. This means that several leaders can coexist during an arbitrarily long period of time, and there is no way for the processes to learn when this "confusing" period is over.

Ω-based consensus protocols are described in Refs. 33 and 34. The requirement $t < n/2$ is necessary for such protocols (3). Moreover, it has been shown that Ω and $\diamond S$ have the same computational power (32,35). No one allows the solution of a problem that could be solved without the other.

Consensus is a particular case of a more general problem, namely the k-set agreement problem. In that

[8] "Last" refers here to physical time, as the consistency criterion considered for the register is atomicity.

problem, the processes can decide up to k different (proposed) values. Consensus is consequently a 1-set agreement. The weakest failure detector to solve the k-set agreement problem has been investigated in Ref. 36.

On the methodology. Since the implementation of some failure detectors (e.g., eventually accurate failure detectors such as $\diamond S$) can be only approximate during some periods (when the underlying system is unstable), it is interesting to use a failure detector only in "extreme" cases, which means that the use of a failure detector has to be avoided whenever possible.

Considering the atomic broadcast problem, several articles (37–39) have provided atomic broadcast implementations that use a failure detector-based consensus black box only in extreme cases. Such protocols are said to be *thrifty* (or non-trivial) with respect to the underlying oracle.

A few references. The reader interested in the implementation of failure detectors should consult the following references (9–11, 13, 40–44). Protocols implementing failure detectors of the class Ω can be found in Refs. 12,40,45 and 46. The reader interested in the classification of distributed computing problems in presence of failures can consult Refs. 47–49. The reader interested in the weakest failure detector classes to solve some fundamental distributed computing problems can consult Refs. 9 and 22. A class of failure detectors (and related problems) that can be implemented in asynchronous systems is presented in Ref. 50.

Random oracles have also been investigated to solve distributed computing problems in the presence of crash failures (51,52). A combination of random oracles with failure detectors is addressed in Refs. 53–55.

The condition-based approach to solve agreement problems consists in characterizing the largest set of input vectors for which it is possible to solve the problem (56). As a "trivial" example, we can consider the case where we know that more than a majority of processes do propose the same value. It is easy to solve consensus for such input vectors despite one process crash. Roughly speaking, a condition-based protocol solves the corresponding agreement problem each time the input vector belongs to the condition (or when there are no failures), and it does its best to terminate when the input vector does not belong to the condition and there are failures. Very recently, a new class of failure detectors it has been proposed that allows combining the power of conditions with the information on failures required to solve agreement problems (54).

CONCLUSION

The failure detector approach was introduced by Chandra and Toueg. Initially designed for asynchronous systems, it allows a statement of the weakest assumptions that have to be added to these systems in order to solve problems that otherwise could not be solved. So, in this type of system, they allow the barrier separating impossibility and decidability to be crossed. From a more practical software engineering point of view, they strongly favor a modular approach (they allow hiding the timing assumptions needed to solve problems that are otherwise impossible to solve in pure asynchronous systems). Failure detectors have then been extended to synchronous systems. In these systems, they allow the attainment of time complexity lower bounds that could not be attained in purely synchronous systems.

BIBLIOGRAPHY

1. L. Lamport, Proving the correctness of multiprocess programs, *IEEE Trans. Soft. Engineer.*, **3**(2): 125–143, 1977.
2. M. Raynal, Detecting crash failures in asynchronous systems: what? why? how?*Proc. Int. Conference on Dependable Systems and Networks (DSN'04)*, Florence, Italy, 2004.
3. T. D. Chandra and S. Toueg, Unreliable failure detectors for reliable distributed systems, *J. ACM*, **43**(2): 225–267, 1996. (First version published in the proceedings of the *10th ACM Symposium on Principles of Distributed Computing,* 1991.)
4. J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Reading, MA: Addison Wesley, 1979.
5. M. J. Fischer, N. Lynch, and M. S. Paterson, Impossibility of distributed consensus with one faulty process, *J. ACM*, **32** (2): 374–382, 1985.
6. D. Powell, Failure mode assumptions and assumption coverage, *Proc. of the 22nd Int'l Symposium on Fault-Tolerant Computing (FTCS-22)*, Boston, MA, 1992, pp. 386–395.
7. C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, A realistic look at failure detectors, *Proc. IEEE Inter. Conference on Dependable Systems and Networks (DSN'02)*, Washington D. C., 2002, pp. 345–352.
8. R. Guerraoui, Indulgent algorithms, *Proc. 19th ACM Symposium on Principles of Distributed Computing, (PODC'00)*, Portland, OR, 2000, pp. 289–298.
9. T. D. Chandra, V. Hadzilacos, and S. Toueg, The weakest failure detector for solving consensus, *J. ACM*, **43**(4): 685–722, 1996.
10. C. Fetzer, M. Raynal, and F. Tronel, An adaptive failure detection protocol, *Proc. 8th IEEE Pacific Rim Int. Symposium on Dependable Computing (PRDC'01)*, Seoul, Korea, 2001, pp. 146–153.
11. I. Gupta, T. D. Chandra, and G. S. Goldszmidt, On scalable and efficient distributed failure detectors, *Proc. 20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, Newport, RI, 2001, pp. 170–179.
12. M. Larrea, A. Fernández, and S. Arèvalo, Efficient algorithms to implement unreliable failure detectors in partially synchronous systems, *Proc. 13th Symposium on Distributed Computing (DISC'99)*, Bratislava (Slovakia), Berlin: Springer Verlag LNCS #1693, 1999, pp. 34–48.
13. M. Larrea, A. Fernández, and S. Arèvalo, Optimal implementation of the weakest failure detector for solving consensus, *Proc. 19th Symposium on Reliable Distributed Systems (SRDS'00)*, Nuremberg, Germany, 2000, pp. 52–60.
14. A. Mostefaoui, E. Mourgaya, and M. Raynal, Asynchronous implementation of failure detectors, *Proc. Int. IEEE Conference on Dependable Systems and Networks (DSN'03)*, San Francisco, CA, 2003, pp. 351–360.
15. A. Mostefaoui and M. Raynal, Solving consensus using Chandra-Toueg's unreliable failure detectors: a general quorum-based approach, *Proc. 13th Symp. on DIStributed*

Computing (DISC'99), Berlin: Springer Verlag LNCS #1693, Bratislava, Slovakia, 1999, pp. 49–63.

16. M. Hurfin, A. Mostefaoui, and M. Raynal, A versatile family of consensus protocols based on Chandra-Toueg's unreliable failure detectors, *IEEE Trans. Comp.*, **51**(4): 395–408, 2002.
17. M. Hurfin and M. Raynal, A simple and fast asynchronous consensus protocol based on a weak failure detector, *Distrib. Comput.*, **12**(4): 209–223, 1999.
18. A. Schiper, Early consensus in an asynchronous system with a weak failure detector, *Distrib. Comput.*, **10**: 149–157, 1997.
19. L. Pease, R. Shostak, and L. Lamport, Reaching agreement in presence of faults, *J. ACM*, **27**(2): 228–234, 1980.
20. J.-M. Hélary, M. Hurfin, A. Mostefaoui, M. Raynal, and F. Tronel, Computing global functions in asynchronous distributed systems with process crashes, *IEEE Trans. Par. Distrib. Syst.*, **11**(9): 897–909, 2000.
21. C. Delporte-Gallet, H. Fauconnier, Helary J.-M., and M. Raynal, Early stopping in global data computation, *IEEE Trans. Parallel Distrib. Syst.*, **14**(9): 909–921, 2003.
22. C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg, The weakest failure detetors to solve certain fundamental problems in distributed computing, *Proc. 23h ACM Symposium on Principles of Distributed Computing (PODC'04)*, St-John's, Newfoundland, Canada, 2004, pp. 338–346.
23. R. Guerraoui, Non-blocking atomic commit in asynchronous distributed systems with failure detectors, *Distrib. Comput.*, **15**: 17–25, 2002.
24. M. K. Aguilera, W. Chen, and S. Toueg, On quiescent reliable communication, *SIAM J. Comput.*, **29**(6): 2040–2073, 2000.
25. R. Koo and S. Toueg, Effects of message loss on the termination of distributed protocols, *Informat. Proc. Lett.***27**: 181–188, 1987.
26. M. K. Aguilera, G. Le Lann, and S. Toueg, On the impact of fast failure detectors on real-time fault-tolerant systems, *Proc. 16th Symposium on Distributed Computing (DISC'02)*, Berlin: Springer-Verlag LNCS #2508, 2002, pp. 354–369.
27. H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, New York: McGraw-Hill, 1988.
28. C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, Failure detection lower bounds on registers and consensus, *Proc. 16th Symposium on Distributed Computing (DISC'02)*, Berlin: Springer-Verlag LNCS #2508, 2002, pp. 237–251.
29. R. Friedman, A. Mostefaoui, and M. Raynal, Asynchronous bounded lifetime failure detectors, *Informat. Proces. Lett.*, **94** (2): 85–91, 2005.
30. R. Friedman, A. Mostefaoui, and M. Raynal, A weakest failure detector-based asynchronous consensus protocol for $f < n$, *Informat. Proces. Lett.*, **90**(1): 39–46, 2004.
31. M. K. Aguilera, S. Toueg, and B. Deianov, Revisiting the weakest failure detector for uniform reliable broadcast, *Proc. 13th Int. Symposium on DIStributed Computing (DISC'99)*, Berlin: Springer-Verlag LNCS #1693, 1999, pp. 21–34.
32. M. Raynal, Quiescent uniform reliable broadcast as an introduction to failure detector oracles, *Proc. 6th Int. Conference on Parallel Computing Technologies (PaCT'01)*, Novosibirsk, Springer Verlag LNCS #2127, 2001, pp. 98–111.
33. R. Guerraoui and M. Raynal, The information structure of indulgent consensus, *IEEE Trans. Comput.*, **53**(4): 453–466, 2004.
34. A. Mostefaoui and M. Raynal, Leader-based consensus, *Parallel Proc. Lett.*, **11**(1): 95–107, 2001.
35. F. Chu, Reducing Ω to $\Diamond\mathcal{W}$, *Informat. Process. Lett.*, **76**(6): 293–298, 1998.
36. P. Zielińsky, Anti-Ω: the weakest failure detector for set agreement, *Tech Report #694*, University of Cambridge, UK, 2007.
37. M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, Thrifty generic broadcast, *Proc. 14th Symposium on Distributed Computing (DISC'00),* Berlin, Springer-Verlag LNCS #1914, pp. 268–282, 2000.
38. A. Mostefaoui and M. Raynal, Low-cost consensus-based atomic broadcast, *7th IEEE Pacific Rim Int'l Symposium on Dependable Computing (PRDC'2000)*, IEEE Computer Society Press, UCLA, Los Angeles, CA, 2000, pp. 45–52.
39. F. Pedone and A. Schiper, Handling message semantics with generic broadcast protocols, *Distrib. Comput.*, **15**(2): 97–107, 2002.
40. M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, On implementing Ω with weak reliability and synchrony assumptions *Proc. 22th ACM Symposium on Principles of Distributed Computing (PODC'03)*, Boston, MA, 2003, pp. 306–314.
41. M. Bertier, O. Marin, and P. Sens, Implementation and performance evaluation of an adaptable failure detector, *Proc. Int. IEEE Conference on Dependable Systems and Networks (DSN'02)*, Washington, D. C., 2002, pp. 354–363.
42. W. Chen, S. Toueg, and M. K. Aguilera, On the quality of service of failure detectors, *IEEE Trans. Comput.*, **51**(5): 561–580, 2002.
43. A. Mostefaoui, D. Powell, and M. Raynal, A hybrid approach for building eventually accurate failure detectors, *10th IEEE Pacific Rim Int. Symposium on Dependable Computing (PRDC'2004)*, Papeete, Tahiti, France, 2004, pp. 57–65.
44. M. Raynal and F. Tronel, Group membership failure detection: a simple protocol and its probabilistic analysis, *Distrib. Syst. Engineer. J.*, **6** (3): 95–102, 1999.
45. M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, Communication-efficient leader election and consensus with limited link synchrony, *Proc. 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, St-John's, Newfoundland, Canada, 2004, pp. 328–337.
46. A. Mostefaoui, M. Raynal, and C. Travers, Time-free and timer-based assumptions can be combined to get eventual leadership, *IEEE Trans. Parall. Distrib. Syst.*, **17**(7): 656–666, 2006.
47. C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, Shared memory *vs* message passing, *Tech Report* IC/2003/77, EPFL, Lausanne, December 2003.
48. E. Fromentin, M. Raynal, and F. Tronel, On classes of problems in asynchronous distributed systems with process crashes, *19th IEEE Int. Conf. on Distributed Computing Systems (ICDCS'99)*, Austin, TX, 1999, pp. 470–477.
49. V. Hadzilacos and S. Toueg, Reliable broadcast and related problems, In S. Mullender (ed.), *Distributed Systems*, New York: ACM Press, 1993, pp. 97–145.
50. V. K. Garg and J. R. Mitchell, Implementable failure detectors in asynchronous systems, *Proc. 18th Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (F-ST & TCS'98)*, Chennai, India, Berlin: Springer Verlag LNCS #1530, 1998.
51. M. Ben-Or, Another advantage of free choice: completely asynchronous agreement protocols, *2nd ACM Symposium on Principles of Distributed Computing, (PODC'83)*, Montréal, Canada, 1983, pp. 27–30.

52. M. Rabin, Randomized byzantine generals, *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, Los Alamitos, CA, 1983, pp. 116–124.

53. M. K. Aguilera and S. Toueg, Failure detection and randomization: a hybrid approach to solve consensus, *SIAM J. Comput.*, **28**(3): 890–903, 1998.

54. A. Mostefaoui, S. Rajsbaum, and M. Raynal, Versatile and modular consensus protocol, *Int. IEEE / IFIP Conf. on Dependable Systems and Networks (DSN'02)*, Washington, DC, 2002, pp. 364–373.

55. A. Mostefaoui, M. Raynal, and F. Tronel, The best of both worlds: a hybrid approach to solve consensus, *Proc. Int. Conference on Dependable Systems and Networks (DSN'00)*, New York City, 2000, pp. 513–522.

56. A. Mostefaoui, S. Rajsbaum, and M. Raynal, Conditions on input vectors for consensus solvability in asynchronous distributed systems, *J. ACM*, **50**(6): 922–954, 2003.

57. A. Mostefaoui, S. Rajsbaum, and M. Raynal, The combined power of conditions and information on failures to solve asynchronous set agreement, *24th ACM SIGACT-SIGOPS Int. Symposium on Principles of Distributed Computing (PODC'05)*, Las Vegas, NV, 2005, pp. 179–188.

FURTHER READING

H. Attiya, A. Bar-Noy, and D. Dolev, Sharing memory robustly in message passing systems, *J. ACM*, **42**(1): 121–132, 1995.

M. Chor and C. Dwork, Randomization in byzantine agreement, *Adv. Comp. Res.*, **5**: 443–497, 1989.

A. Mostefaoui, E. Mourgaya, and M. Raynal, An introduction to oracles for asynchronous distributed systems, *Fut. Generat. Comput. Syst.*, **18**(6): 757–767, 2002.

MICHEL RAYNAL
IRISA, Université de Rennes
Rennes Cedex, France

G

GRADIENT-BASED OPTIMIZATION TECHNIQUES FOR DISCRETE EVENT SYSTEMS SIMULATION

INTRODUCTION

Often, decision makers shy away from system simulation modeling as a tool because of the prohibitive computational cost of generating random realizations. The development, in recent years, of efficient "single-run" approaches, as well as dramatically reduced computer cost through improved hardware, make simulation more attractive. Now that the roadblocks have been eased, a new awareness of the prescriptive simulation modeling applications, such as optimization, goal-seeking, and controlling of the prescribed design input values, is needed. This article brings together the new approaches to prescriptive simulation modeling.

Discrete event systems simulation modeling has been an important and effective technique in describing systems in several areas, such as business and engineering. Simulation modeling is an effective way of pretesting proposed systems, plans, or policies, before developing expensive prototypes, field tests, or actual implementations.

Many human-made systems can be modeled as discrete event systems (DESs); examples are computer systems, communication networks, flexible manufacturing systems, production assembly lines, and traffic transportation systems. DESs evolve with the occurrence of discrete events, such as the arrival of a job or the completion of a task, in contrast with continuous dynamic processes, such as aerospace vehicles, which are primarily represented by differential equations. Because of the complex dynamics that result from *stochastic interactions* of such discrete events over time, the tasks of performance analysis and optimization of DES can be difficult. At the same time, because such systems are becoming more widespread as a result of modern technological advances, it is important to have tools for analyzing and optimizing the parameters of these systems.

Analyzing complex DESs often requires computer simulation. In these systems, the objective function may not be expressible as an explicit function of the input parameters; rather, it involves some performance measures of the system whose values can be found only by running the simulation model or by observing the actual system. However, because of the increasingly large size and inherent complexity of most human-made systems, purely analytical means are often insufficient for optimization. In these cases, simulation is an alternative; its chief advantage is *its generality applicability*, and its primary disadvantage is its cost in terms of *time* and *money*. Although the price for computing resources continues to dramatically decrease, only a statistical estimate as opposed to an exact solution can be obtained. For practical purposes, a statistical estimate derived from a simulation is sufficient.

These human-made DESs are costly, and therefore it is important to operate them as efficiently as possible. These high costs make it necessary to find more efficient means of conducting simulation and optimizing its output. Consider optimizing an objective function with respect to a set of continuous and/or discrete controllable parameters subject to some constraints.

In almost all simulation models, an expected value can express the system's performance. Let

$$J(\theta) = E\{L[X(\theta)]\} \tag{1}$$

be the expected steady-state performance measure of a stochastic system. For the transient period analysis, see Equation (1). In Equation (1), $X(\theta)$ is a random vector with a known joint probability density function (pdf) $f(x;\theta)$, depending on a decision parameter θ which could be a vector $\theta \in \Theta \subseteq R^N$, and $L[X(\theta)]$ is the performance function. For example, in a queuing system, $L[X(\theta)]$ might be the steady-state waiting time in the system, $f(x;\theta)$ the underlying pdf of the service time, and θ the service rate.

Simulation is an option when $L[X(\theta)]$ is either unknown or too complicated to calculate analytically. An estimator of $J(\theta)$ for a given value $\theta=\theta_0$ is obtained by averaging over n independent replications; that is

$$\hat{J}(\theta_0) = \sum_{i=1}^{n} L[X_i(\theta_0)]/n \tag{2}$$

where x_i, $i=1, 2,\ldots, n$ are random vector realizations, and n is the sample size (number of independent replications, regenerative cycle, batches, etc.). This equation is an asymptotically unbiased estimator and converges to $J(\theta)$ by the law of large numbers. The goal is to optimize $J(\theta)$, subject to $\theta \in \Theta$

Figure 1 shows a general scheme to integrate simulation with gradient-based optimization techniques.

The optimizer and simulator modules are closely coupled, with the output of each module directly providing input for the other module. When users specify a performance measure $J(\theta)$ to be optimized, they must also select one or more of the optimization techniques provided in the following sections. The simulation *begins* with the processing of the initial input parameter θ. The *output data*—an estimate based on Equation (1) for $J(\theta)$—are in turn an input to the optimizer. The optimizer then generates output parameter θ, which in turn is a new input parameter for the simulator. This process is performed iteratively until one reaches a satisfactory solution. A general framework for integrating simulation and optimization as shown in Fig. 1 is proposed in Ref. 1. The inputs are subdivided into two categories: controllable inputs and uncontrollable inputs. For example, in a queuing system, the service rate (μ) and arrival rate (λ) are controllable and uncontrollable inputs, respectively; whereas the expected waiting time in the system $J(\lambda, \mu)$ could be the expected system performance measure.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

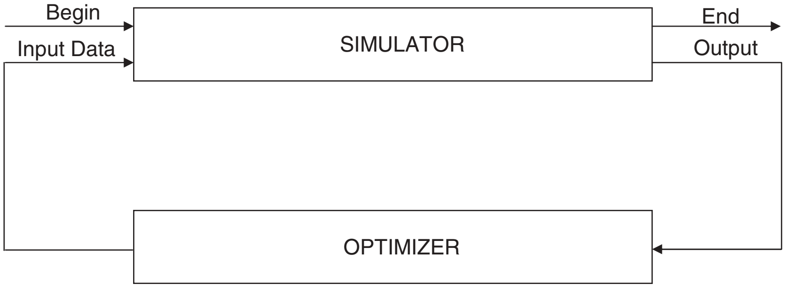


Figure 1. An optimizer support system.

Descriptive analysis includes problem identification, problem formulation, data collection and analysis, computer simulation model development, validation and verification, and output analysis.

Prescriptive analysis includes sensitivity estimation. The traditional approach to obtaining sensitivity information (derivative, Hessian, etc.) combines Crude Monte Carlo estimation with finite "differencing." This approach is impractical when each simulation run takes hours of computer time on even the fastest computers. Methods, which yield enhanced efficiency in estimating sensitivity information at little additional computational not simulation cost, are of great value. In recent years, the first two methods—-infinitesimal perturbation analysis (PA) and score function (SF), also known as likelihood ratio (LR)—have been proposed for estimating sensitivity while simulating the nominal system. Combined with appropriate variance reduction techniques, SF or LR method for sensitivity information could enlarge substantially the class of optimization and goal-seeking problems that can be solved.

Optimization is trying to find the best course of action, for example, optimizing the performance function J over μ. Optimization in simulation has challenged researchers for many years as shown in Fig. 2.

Traditionally, two approaches have been undertaken for optimization of simulation. First, when any part of a simulation can be solved analytically, simulation analysis is combined with analytical modeling. Second, attention is focused on statistical design, such as fractional factorial design and response surface design. Nowadays, the single-run sensitivity estimation is incorporated in the stochastic algorithms, such as Keifer-Wolfowitz, using PA. In using SF or LR sensitivity estimate, because the performance J and the sensitivity are estimated from the *same* sample, they are dependent; moreover, the effect of these correlated errors are usually significant. Therefore, slower procedures, such as the Robbins-Monro (R-M) algorithm, are more satisfactory.

Goal seeking, or targeting the parameter design problem, means trying to solve design problems to meet the target value, that is, to solve the inverse simulation

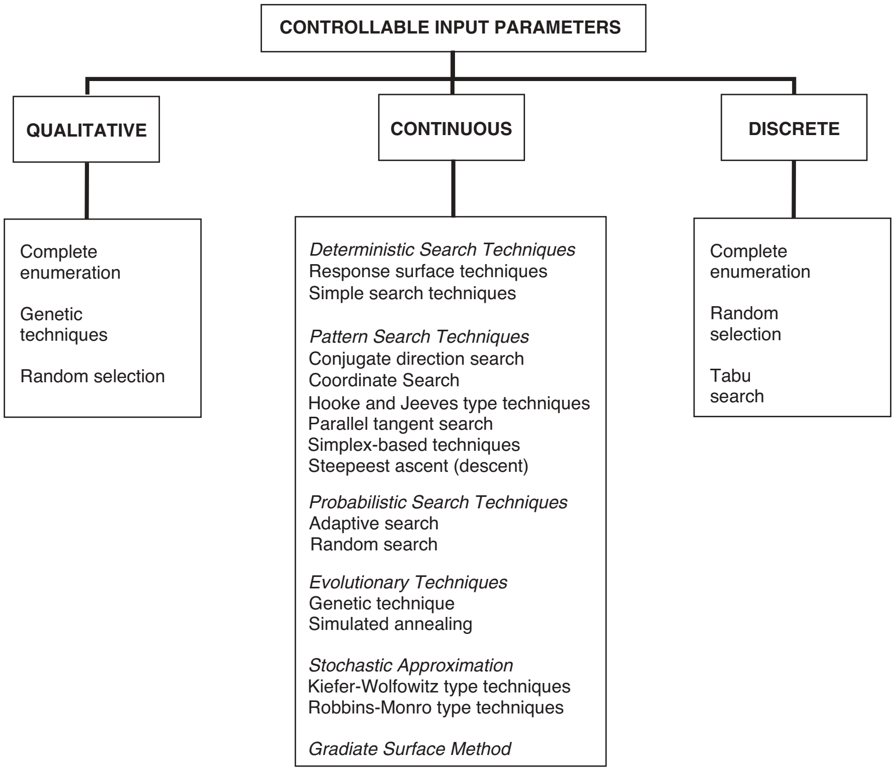


Figure 2. A classification of optimization techniques via simulation.

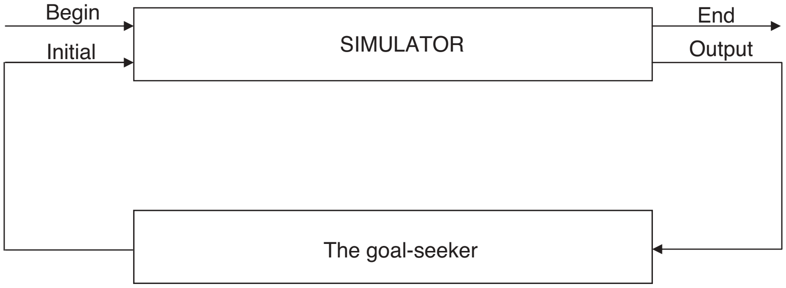


Figure 3. Goal seeking or targeting the parameter design problem.

problem. The "goal-seeking" problem considers the inverse question of descriptive simulation; namely, what must be the controllable input parameter value to achieve a desired output? A simulation-based approach is to estimate the derivative of the performance measure function with respect to the input parameter for a nominal system in a single simulation run, as shown in Fig. 3.

This estimated derivative is used in first-order local approximation of the stochastic version of Newton's root-finding algorithm to estimate the necessary controllable input parameter with a desired statistical confidence.

Postprescriptive analysis includes stability and the what-if analysis. In these analyses, the controllable input parameters are varied to study stability of the solution and to measure the responsiveness of the estimated performance measure to the change in uncontrollable input parameters; that is, to perform the so-called what-if analysis. Simulation models are often subject to serious errors propagated from statistical errors in estimating the parameters of the input distributions. Postprescriptive analysis establishes confidence in the prescriptive results with respect to small changes of parameters in the input distributions. It also provides systematic guidelines for allocating scarce organizational resources to data-collection and data-refinement activities. For example, this estimator requires only one sample path to estimates; any number of small changes in $\mu 1$ are computed simultaneously from a single simulation run while simulating the nominal system.

The remainder of the article is organized as follows. "Gradient Estimation" presents an introduction to gradient estimation. The finite difference (FD), simultaneous perturbation (SP), PA, the SF/LR, and harmonic analysis gradient estimators are presented in the next four sections, respectively. The next section provides a short comparison among these different views of gradient estimators. Then, gradient-based optimization techniques are introduced. Stochastic approximation techniques are presented next, whereas the section that follows is devoted to gradient surface methods. Postoptimality tools are derived in "Postsolution Analysis." Concluding remarks are given last. All techniques are presented in English-like format and therefore can be implemented in a variety of operating systems and machines, which provides unlimited portability.

GRADIENT ESTIMATION

In the design, analysis, and operation of DESs any information about the sensitivity or gradient $dJ(\theta)/d\theta$ is useful to both engineers and managers. The sensitivity $dJ/d\theta$ can be used in conjunction with various optimization algorithms whose function is to adjust θ gradually until a point is reached where $J(\theta)$ is maximized (or minimized). If no other constraints on θ are imposed, then we expect $dJ/d\theta = 0$ at this point. A typical algorithm is the R-M stochastic approximation described by

$$\theta_{n+1} = \theta_n + \eta_n (dJ/d\,\theta)_{\theta=\theta n}\; n = 1, 2, \ldots. \qquad (3)$$

where the parameter is adjusted gradually from some initial value θ_0 until $dJ/d\theta = 0$ for some θ_n. The amount of adjustment is proportional to the value of the derivative evaluated at $\theta = \theta_n$, with properly selected coefficients η_n, $n = 1, 2, \ldots$. These coefficients are referred to as step sizes, scaling factors, or learning rates, which could be any positive sequence of numbers that converge to zero. In our case, the derivative is replaced by its estimate (2–6).

The gradient estimate can be used as a subroutine in a nonlinear programming algorithm, such as the *Augmented Lagrangians* method (7), to obtain the optimal solution of the following stochastic nonlinear program:

Problem P:

$$\min \mathrm{J}(\theta) = \min \mathrm{E}[\mathrm{L}(\mathrm{X}(\theta))]$$

subject to:

$$\begin{aligned} \mathrm{g}_i(\theta) &= \mathrm{E}\,\mathrm{G_i}[(\mathrm{X}(\theta))] \leq 0,\ i = j, \ldots, \mathrm{m} \\ \mathrm{h}_j(\theta) &= \mathrm{E}\,\mathrm{H_j}[(\mathrm{X}(\theta))] = 0,\ \mathrm{j} = \mathrm{I}, \ldots, \mathrm{p} \end{aligned}$$

The objective function and perhaps some constraints are not known analytically, so their functional evaluation and their gradients must be replaced by their estimates via simulation (2, 4, 5).

FINITE DIFFERENCE (FD)

Suppose that we have a system performance parameterized random variable $L(\theta)$ for each θ in some interval, say [a, b]; that is, a stochastic process $\{L(\theta)\text{: } \theta\ \varepsilon[\mathrm{a, b}]\}$. For example, $L(\theta)$ could be the amount of time that a customer waits in the system. The parameter θ could be the mean service rate of the server.

Suppose that the measure of interest is $E[L(\theta)]$ and that the decision maker is interested in the sensitivity of the measure with respect to the parameter value. Also, assume that this measure is differentiable in θ on [a, b]. The brute force FD method consists of the following steps:

Step 1: *Generate* N *sample points*

Generate N sample points ω_1, ω_2, ω_3, …. ω_N, using $\theta - \Delta\theta$ for some small $\Delta\theta$ to generate N realizations. This task is generally done in simulation by generating N sequences of pseudo-random numbers. Then N realizations can be generated for $L(\theta - \Delta\theta)$, namely $L(\theta - \Delta\theta, \omega_1)$, $L(\theta - \Delta\theta, \omega_2)$, $L(\theta - \Delta\theta, \omega_3)$, …. $L(\theta - \Delta\theta, \omega_N)$. In other words, the simulation is performed N times with the parameter set at $\theta - \Delta\theta$.

Step 2: *Perform another* N *simulations*

Typically using the same pseudo-random numbers, that is, using common random variate as a variance reduction technique (VRT), generate N realizations of:

$$L(\theta + \Delta\theta, \omega_1), L(\theta + \Delta\theta, \omega_2), L(\theta + \Delta\theta, \omega_3), \ldots. L(\theta + \Delta\theta, \omega_N).$$

Step 3: *Estimate the derivative*

$$dE[L(\theta)]/d\theta\_[\Sigma L(\theta + \Delta\theta, \omega_i) - \Sigma L(0 - \Delta\theta, \omega_i)]./2N\Delta\theta \quad (4)$$

where the sums are over $i = 1, 2, \ldots, N$. The estimate should get better as N gets larger.

The first fact to notice about this estimate is that it requires $2N$ simulation runs. If θ were a parameter vector and the decision maker were interested in sensitivity analysis with respect to each of the components, then $2N$ simulations would have to be run for each component of θ. This method is inefficient. The second fact to notice about this estimate is that it may have a very poor variance, and it may result in numerical calculation difficulties, because for very small $\Delta\theta$ there is not much statistical difference between $L(\theta + \Delta\theta)$ and $L(\theta - \Delta\theta)$, and we are dividing this difference by a small number. Finally, this estimate is unbiased for each non-zero $\Delta\theta$, because the slope of a secant line is approximating the slope of the tangent line. This result leads to the difficult question of which $\Delta\theta$ to use. To minimize the "secant error," we should use a small $\Delta\theta$, but using such a small value contributes to the variance and numerical difficulties mentioned above. The well-known Kiefer-Wolfowitz stochastic approximation algorithm (8) is based on the FD gradient approximation.

SIMULTANEOUS PERTURBATION (SP)

The SP algorithm introduced by Spall (10, 11) has attracted considerable attention. SP is based on an easily implemented and highly efficient gradient approximation that relies on measurements of the performance function, not on measurements of the gradient of the performance function. The gradient approximation is based on only *two*function measurements, regardless of the dimension of the gradient vector θ. This finding contrasts with the FD approach, which requires several function measurements proportional to the dimension of the gradient vector. The fundamental (and perhaps surprising) theoretical result is that under reasonably general conditions, SP and Kiefer-Wolfowitz FD-based stochastic approximation (for optimization - purposes) achieve the *same* level of statistical accuracy for a given number of iterations, even though SP uses fewer function evaluations than FD.

The step-by-step summary below shows how to produce the gradient estimate.

Step 0: *Initialization and Coefficient Selection*

Set counter index k equal to 0. Pick an initial guess for the non-negative coefficients C and for η in the SP gain sequence

$$C_k = C(k+1)^{-\eta} \quad (5)$$

A practically effective (and theoretically valid) value for η is 0.101; C may be determined based on prior knowledge and/or numerical experimentation. In the latter case, C is approximately equal to the standard deviation (noise) when running the simulation several times with the same input, using different random number streams.

Step 1: *Generation of a simultaneous perturbation vector*

Generate by Monte Carlo a p-dimensional random vector Δ_k where p is the dimension of θ, and each of the p components of Δ_k is independently generated from a Bernoulli $(-1, +1)$ distribution with a probability of 1/2 for each outcome.

Step 2: *Performance measure evaluations*

Obtain two measurements of the performance function J(.) based on N realizations of simultaneous perturbation around θ: $L(\theta + C_k\,\Delta_k)$ and $L(\theta - C_k\,\Delta_k)$ with the C_k and Δ_k from Steps 0 and 1.

Step 3: *Gradient approximation*

Generate the simultaneous perturbation approximation to the (unknown) p-dimensional gradient of $J(\theta)$ by averaging over N:

$$\begin{matrix} \dfrac{J(\theta + C_k\,\Delta_k) - J(\theta - C_k\,\Delta_k)}{2C_k\,\Delta_{k1}} \\ \vdots \\ \dfrac{J(\theta + C_k\,\Delta_k) - J(\theta - C_k\,\Delta_k)}{2C_k\,\Delta_{kp}} \end{matrix} \quad (6)$$

where Δ_{ki} is the ith component on the Δ_k vector (Δ_{ki} may be a Bernoulli random variable, as discussed in Step 1). Note that only the denominators change in the p components of the gradient estimation; the numerators reflect the simultaneous perturbation of all components of θ in contrast to the component-by-component perturbation in the FD gradient approximation.

For SP-based optimization, refer to Ref. 6. SP can be seriously limited by, for example, the stability constraints of the system (traffic intensity must remain less than 1 for steady-state sensitivity estimation).

One-measurement PA analog to Equation (5) is:

$$\begin{matrix} \dfrac{J(\theta + C_k\Delta_k)}{2C_k\,\Delta_{k1}} \\ \vdots \\ \dfrac{J(\theta + C_k\,\Delta_k)}{2C_k\,\Delta_{kp}} \end{matrix} \quad (7)$$

A closely related approach to SP is the random directional SP. Let $d_k \in \mathrm{R}^p$ denote a vector that contains p mutually independent variables generated by the standard normal distribution. Then the kth component of random direction gradient estimate is:

$$[\mathrm{J}(\theta + \mathrm{C}_k \mathrm{d}_k) - \mathrm{J}(\theta - \mathrm{C}_k \mathrm{d}_k)]/2\mathrm{C}_k \tag{8}$$

SP uses two different estimators. A single-run version of SP is given in Ref. 10.

The following approaches avoid any numerical problems associated with taking the difference as an approximation to the gradient; they are based on *a single simulation run*, and the methods have the potential for real-time applications.

PERTURBATION ANALYSIS (PA)

PA computes (roughly) what simulations would have produced had θ been changed to θ + Δθ without actually making this change (11, 15). The intuitive idea behind PA is that a sample path constructed using θ is frequently structurally very similar to the sample path using θ + Δθ. A large amount of information is the same for both of them. It is wasteful to throw this information away and to start the simulation from scratch with θ + Δθ. In PA, moreover, we can let Δθ → 0 to get a derivative estimator without numerical problems.

The effect of a parameter change on the performance measure is important. However, we would like to realize this change by keeping *the order of events exactly the same*. The perturbations will be so small that only the durations, not the order, of the states will be affected. This effect should be observed in three successive stages:

Step 1: How does a change in the value of a parameter vary the sample durations related to that parameter?

Step 2: How does the change in individual sample duration reflect itself as a change in a subsequent particular sample realization?

Step 3: Finally, what is the relationship between the variation of the sample realization and its expected value?

PA calculates the gradient of performance measure for every realization ω. The value of L(θ, ω) is obtained by a simulation run. PA calculates the value of L(θ + Δθ, ω) in parallel. This statement means that PA calculates all derivatives from a single simulation run. Under mild conditions, for every realization ω, a bound Δ(ω) exists, so that the PA estimator equals the exact path derivation of the performance measure for all $|\Delta\theta| \leq \Delta(\omega)$. Averaging the PA estimator over independent replication yields E[dL(θ, ω)/dθ]. PA works by computing a *sample path derivative* from the simulated sample path, which is used as an estimate of E{d[L(θ)]/dθ}. This random variable is the derivative d [L(θ)]/dθ}, which is the derivative of a random variable. If this estimator is statistically unbiased, then

$$\mathrm{E}\{\mathrm{d}[\mathrm{L}(\theta)]/\mathrm{d}\theta\} = \mathrm{d}\,\mathrm{E}[\mathrm{L}(\theta)]/\mathrm{d}\theta \tag{9}$$

This equation says that unbiasedness corresponds to the interchange of the operations of differentiation and expectation. Much theory of PA involves finding conditions that justify this interchange. Now, if the interchangeability holds, then the estimate of dE[L(θ)] / dθ is given by

$$\mathrm{dE}[\mathrm{L}(\theta)]/\mathrm{d}\theta = \mathrm{E}\{\mathrm{dL}(\theta)]/\mathrm{d}\theta\} - \Sigma\, \mathrm{dL}(\theta,\, \omega_i)/\mathrm{N}\mathrm{d}\theta \tag{10}$$

where the sum is over all $i = 1, 2, \ldots, N$, and dL(θ, ω_i) is the random variable dL(θ) / dθ evaluated at the sample point ω_i. The strong law of large numbers implies that

$$\mathrm{dE}[\mathrm{L}(\theta)]/\mathrm{d}\theta = \lim \Sigma\, \mathrm{dL}(\theta,\, \omega_i)]/N\mathrm{d}\theta \quad \text{as}\, N \to \infty \tag{11}$$

where the sum is over all $i = 1, 2, 3, \ldots N$.

Let X(θ) be a parameterized family of random variables with F(x, θ) as its cumulative probability distribution function. Assume F(x, θ) is continuously differentiable in x and θ, and for Δθ > 0, F(x, θ) ≥ F(x, θ + Δθ). We can view X(θ) as the inverse transformation $\mathrm{X}(\theta) = \mathrm{F}^1(\mathrm{U}, \theta)$ where U is uniformly distributed on [0, 1]. The random variable dX(θ)/dθ can now be defined by

$$\begin{aligned}\mathrm{dX}(\theta)/\mathrm{d}\theta &= \lim_{\Delta\theta \to 0} [\mathrm{F}^{-1}(\omega,\, \Delta\theta) - \mathrm{F}^{-1}(\omega,\, \theta)]/\Delta\theta \\ &= \mathrm{dF}^{-1}(\omega,\, \theta/\mathrm{d}\theta)\end{aligned} \tag{12}$$

for each ω ε [0, 1] for which this limit exists. Under differentiability properties of F(x, θ), it can be shown (16) that

$$\mathrm{dX}(\theta)/\mathrm{d}\theta = -\{\partial[\mathrm{F}(\mathrm{X}(\theta), \theta)]/\partial\theta\}/\partial[\mathrm{F}(\mathrm{X}(\theta), \theta)]\}/\partial\mathrm{x} \tag{13}$$

This result is useful because it expresses the derivative of a random variable as a function of the random variable. Note that the denominator of the right side of Equation (13) is the probability density function of X(θ). It is also useful to note that this derivative has the minimum variance among all possible definitions of the sample path derivative (17).

We have already observed that we can view X(θ) as the inverse transformation $\mathrm{X}(\theta) = \mathrm{F}^{-1}(\mathrm{U}, \theta)$ where U is uniformly distributed on [0, 1]. In other words, the expectation may be taken over u ∈ [0, 1] instead of x, with u = F(x, θ); that is,

$$\mathrm{J}(\theta) = \int \mathrm{L}[x(\mathrm{u}, \theta)]\mathrm{du} \tag{14}$$

The interpretation is that the sample path really depends on a uniformly distributed random variable over which the expectation is taken, which specifies another random variable X(U, θ), in turn determines the sample function

L[X(U, θ)]. If we now differentiate Equation (14) with respect to θ, we get

$$\mathrm{dJ}/\mathrm{d}\theta = \int \{\partial \mathrm{L}[x(\mathrm{u}, \theta)]/\partial\theta\}\, \mathrm{du} = \mathrm{E}\{\partial \mathrm{L}[x(\mathrm{u},\, \theta)]/\partial\theta\} \tag{15}$$

If the interchange of expectation and differentiation is allowed, then the sample deviation dL/dθ is an unbiased estimator of the performance derivative dJ/dθ. Note that the sample derivative can be written as follows:

$$\partial L[x(u,\theta)]/\partial\theta = \partial L[x(u,\theta)]/\partial x \quad . \quad \partial x(u,\theta)/\partial\theta \tag{16}$$

This result is useful because it reflects that PA consists of two parts: *perturbation generation* [i.e., how changes in θ introduce changes in X(θ)], and *perturbation propagation* [i.e., how a change in X(θ) ultimately affects the sample function L]. Observe that in this whole process, U (the underlying random number) is kept fixed (16).

Frequently, a PA estimator of d E [L(θ)]/dθ for a performance measure random variable L(θ) will fail to be unbiased or strongly consistent, if it is not the case that L(θ) is a continuous function of θ. When this happens, the treatment involves conditioning on another suitably chosen random variable K(θ). Conditioning tends to "smooth out" discontinuities. Hence, the random variable dE[L(θ) | K(θ)]/dθ can be used as a potential estimator, provided it is a continuous function of θ with probability 1, and if this derivative can be observed from the sample space. Now, *if* this estimator is unbiased, then we have

$$E\{d\,E[L(\theta)|K(\theta)]/d\theta\} = d\,E\{E[L(\theta)|K(\theta)]/d\theta]\} = d\,E[L(\theta)]/d\theta$$

Obviously, PA algorithms cannot be used for discrete parameters. Furthermore, for some systems, the common probability space derivatives do not give relevant information about the derivative of the objective function. For these systems, the finite difference estimator is used. Generally, these algorithms are concerned with calculating:

$$L(\theta+\Delta\theta,\,\omega) = h\,[z_1(\theta+\Delta\theta,\,\omega),\ldots,z_N(\theta+\Delta\theta,\omega)]$$

where a finite ω is given. In this case, the function h is not necessarily the same as the nominal sample path; in fact, it could be different, because a finite perturbation in the parameter value often produces a perturbed sample path that is very different from the nominal sample path. The conclusion is that, except for some limited systems, the information on the nominal sample path cannot be used readily to calculate L(θ + Δθ, ω). Instead, one obtains estimates of E[L(θ)] and E[L(θ + Δθ)], which are to obtained simultaneously by viewing a simulation as a function of the system parameters (including θ) and a sequence of uniform random variables into the performance measure L. Changing the value of θ by a small amount is equivalent to using a slightly different mapping. This method leads to the cut-and-paste concept, which involves making the sample paths obtained from different parameter values behave similarly by appending and cutting pieces of the trajectories appropriately. This result allows for obtaining estimates of the performance measure at different values of the parameter by running parallel simulations.

Related to the cut-and-paste concept is the standard clock method. Under the assumption that the times between events of type i are exponentially distributed with rate λ_i, events are generated at the rate $\Sigma\lambda_i$ which is the maximal rate at which events could possibly occur. Some of these events will be rejected because of perturbations or infeasibility, which allows for many simulations to be run in parallel. Several approximations allow the Markov assumption to be relaxed. The standard clock method is particularly suited for answering "what if" questions in real-time.

SCORE FUNCTION OR LIKELIHOOD RATIO METHOD (SF/LR)

Using S(x, θ) = f(x, θ)/g(x) (the so-called likelihood ratio) and g(x) as the new probability density function, the performance measure can be rewritten as

$$\begin{aligned} J(\theta) &= \int L(x)\,f(x,\,\theta)\,dx = \int L(x)\,S(x,\theta)\,g(x)\,dx \\ &= E_g\{L(x)S(x,\theta)\} \end{aligned} \tag{17}$$

The derivative of the performance measure becomes

$$\begin{aligned} dJ/d\theta &= \int L(x)[\partial S(x,\,\theta)/\partial\theta]\,g(x)\,dx \\ &= E_g[L(x)\,S(x,\,\theta)[\partial \ln f(x,\,\theta)/\partial\theta] \end{aligned} \tag{18}$$

which is the likelihood ratio gradient estimator. The gradient can be estimated simultaneously, at any number of different parameter values, in a single-run simulation. Here L(x) is written instead of L[x(θ)] because L, which is a specific realization of the performance metric, is fixed as θ varies. Thus, this approach is referred to as common realization. The new density function g(x) is usually assumed to be of the form g(x) = f(x, θ_0), where θ_0 is some fixed value of the parameter, which is called the reference parameter. The optimal value of θ_0, to minimize the variance of the gradient estimate, is estimated by simulation (5).

If we set g(x) = f(x, θ), then it is called the score function method of estimating the gradient

$$dJ/d\theta = \int L(x)\,[\partial\,f(x,\,\theta)/\partial\theta]\,dx \tag{19}$$

provided that the interchange of expectation and differentiation in Equation (18) is allowed (18). Moreover, observing that

$$\partial \ln f(x,\,\theta)/\partial\theta = [\partial f(x,\theta)/\partial\,\theta]/f(x,\,\theta)$$

we get, assuming f(x, θ) ≠ 0,

$$\begin{aligned} dJ/d\theta &= \int L(x)\,[\partial \ln f(x,\,\theta)/\partial\,\theta]\,f(x,\,\theta)dx \\ &= E\{L(X)[\partial \ln f(X,\theta)/\partial\,\theta]\} \end{aligned} \tag{20}$$

In this way, an alternative unbiased estimator of dJ/dθ is obtained. This estimator is known as the SF estimator, which is sometimes also referred to as the LR estimator. Again, in this case, the parameter θ is viewed as affecting the probability distribution of values of the sample function but not the particular value observed. This method must be contrasted to the PA approach where we view θ as affecting the sample function itself.

Recent work on the SF or LR approach includes Refs. 19 and 20. The basic idea is that the gradient of the performance measure function, $J'(\theta)$, is expressed as an expectation with respect to the *same* distribution as the performance measure function itself. Therefore, the sensitivity information can be obtained with little computational (not simulation) cost, while estimating the performance measure. For example, a gradient estimate of customer sojourn time in a GI/G/1 queuing system could be

$$\sum_{i=1}^{N} \mathrm{q_i}[\partial \ln \mathrm{f}(x_i,\ \theta)/\partial\theta]/N \tag{21}$$

where q_i is the sojourn time of the ith customer and $\mathrm{f}(x_i, \theta)$ is the pdf of the service time X with service rate θ.

It is well known that the crude form of the SF estimator suffers from the problem of linear growth in its variance as the simulation run increases (19). However, in the steady-state simulation, the variance can be controlled by run length. Furthermore, information about the variance may be incorporated into the simulation algorithm. A recent flurry of activity has attempted to improve the accuracy of the SF estimates. Under regenerative conditions, the estimator can be modified easily to alleviate this problem, yet the magnitude of the variance may be large for queuing systems with heavy traffic intensity. A heuristic decomposition method of estimating the SF-based sensitivity is proposed in Ref. 4. The heuristic idea is to treat each component of the system (e.g., each queue) separately, which synchronously assumes that individual components have "local" regenerative cycles. This approach is promising, because the estimator remains unbiased and efficient although the global regenerative cycle is very long.

Now, look at the general (nonregenerative) case. In this case, any simulation will give a biased estimator of the gradient, as simulations are necessarily finite. If n (the length of the simulation) is large enough, then this bias is negligible. However, as noted earlier, the variance of the SF sensitivity estimator increases with n; so, a crude SF estimator is not even approximately consistent. Many ways can be used to attack this problem. Most variation in an estimator comes *from the score function*. The variation is especially high when all past inputs contribute to the performance and the scores from all are included. When batch means are used, the variation reduces by keeping the length of the batch small.

A second method is to reduce the variance of the score to such an extent that we can use simulations long enough to eliminate the bias effectively. This approach is the most promising. The variance may be reduced further by using standard VRT, such as importance sampling (5). Finally, we can simply use many iid replications of the simulation.

As a way to reduce the amount of simulation, the n observations are divided into k groups, or batches, of m consecutive observations. Batch means are calculated for each of the k batches. However, the batches are not truly independent. The mean-squared error of the resulting sensitivity estimates depends highly on the choice of m, but the optimal value of m is unknown. The estimate of sensitivity is:

$$\sum_{t=1}^{n} \mathrm{L}_i\ \mathrm{S}_t(\theta)/n \tag{22}$$

where $n = km$ and $\mathrm{S}_t(\theta)$ is the score associated with L_t. The batch size used to obtain this sensitivity estimate will depend on the autocorrelation structure of the sequence of the batch performances. The score associated with a particular performance L_t should be calculated from all previous inputs on which the performance depends. The performance will be approximately independent of inputs that occurred in the past. The more L_t depends on past inputs, the more inputs should be used in the calculation of the score and the larger m must be. For example, for an M/M/1 queue, a reasonable batch length is about 10 times the expected length of the busy period. Even this amount leads to large bias for high-traffic intensities. Because m is not infinite, the estimator will be biased. In steady-state estimation, the performance depends mainly on recent inputs, is approximately independent of more remote inputs, and is completely independent of future inputs. In this case, a long simulation is run. However, instead of using the score of all the inputs in the estimate, a weighted score is used:

$$\sum_{t=1}^{n} \mathrm{L}_t \sum_{i=0}^{t-1} \mathrm{a}_i\ \mathrm{S}(x-\mathrm{i};\theta)/\mathrm{n} \tag{23}$$

with the weights decreasing to 0 as i goes to infinity. Two weighing schemes have been examined in Ref. 21. The first is termed the moving batch SF estimate. Its weights are:

$$\begin{aligned} \mathrm{a}_i = 1;\quad & \text{for}\quad i = 0,\ldots,\mathrm{m}-1 \\ \mathrm{a}_i = 0;\quad & \text{for}\quad i = \mathrm{m},\ldots,\mathrm{n}-1 \end{aligned} \tag{24}$$

in other words, only the m most recent inputs are considered in the score. This gives

$$\sum_{t=1}^{n} \mathrm{L_t} \sum_{\mathrm{i}=0}^{\mathrm{m}-1} \mathrm{S}(\mathrm{x_t}-\mathrm{i};\theta)/\mathrm{n} \tag{25}$$

By dropping all but the m most recent inputs, the variance of the resulting sensitivity estimator is reduced. Thus, it makes sense to drop the m terms, because for some terms L_i is independent of future inputs, so these terms have an expectation equal to zero. Moreover, the scores are only approximately independent of the performance, so that the expectation of these scores is approximately zero. This estimate of the gradient is closely related to the batch means. The equations for these estimates for k batches of length m, $n = km$ are:

$$\sum_{j=1}^{k} \sum_{j=(i-1)\mathrm{m}+1}^{im} \mathrm{L}_i \sum_{\mathrm{t}=(i-1)\mathrm{m}+1}^{j} \mathrm{S}(x_t;\theta)/n \tag{26}$$

or

$$\sum_{j=2}^{k+1} \sum_{j=(i-1)\mathrm{m}+1}^{im} \mathrm{L}_i \sum_{\mathrm{t}=(i-2)m+1}^{j} \mathrm{S}(\mathrm{x_t};\theta)/n \tag{27}$$

In each case, only product terms with nonzero expectations are included. However, with the moving batch estimate, each score is calculated from the same number of inputs. The scores for the first estimate are calculated using from t to m terms; those of the second estimate are calculated using from t to $2m$ terms. Some simulation results for these estimators are given in Ref. 21, which shows substantial reduction in variance for both estimators.

A disadvantage of these batch estimates is that they take somewhat longer to compute than a standard batch estimator. As another alternative, exponential weighing may be used weights of the form $a_i = a^i$, for some a <1. Other possible scores for use with a batch estimate are, for $i = 1, \ldots k$, and $(i-1) < t \leq im$:

$$S_{1t}(\theta) = \sum L_i \, \underset{j=(i-1)m+1}{S_j^t(\theta)/n} \tag{28}$$

$$S_{2t}(\theta) = \sum L_i \, \underset{j=(i-2)m+1}{S_j^t(\theta)/n} \tag{29}$$

$$S_{3t}(\theta) = \sum L_i \, \underset{j=(i-1)m+1}{S_j^{im}(\theta)/n} \tag{30}$$

$$S_{4t}(\theta) = \sum L_i \, \underset{j=(i-2)m+1}{S_j^{im}(\theta)/n} \tag{31}$$

In Equation (28), the score is computed from all inputs from the beginning of the batch to the present observation. Equation (29) also includes the total score of the previous batch. The estimate using Equation (30) is the score of the entire batch, which is a crude type of estimator. Equation (31) is the sum of the scores of the present and the last batch, which is also a crude type of estimator. A crude estimate is expected to do more poorly than these efficient estimates. The performance at time t is independent of future inputs. Efficient estimators discard these product terms with expectation zero. The idea behind S_{2t} and S_{4t} is as follows. Adding the score of the previous batch decreases the bias, at the cost of increasing the variance. At moderate to high-traffic intensities, this process will lead to a significant reduction in the mean squared error. At low intensities, the estimates are already approximately unbiased; so, the mean square error will be increased. In the case of the crude estimates, this should do substantially better than merely doubling the batch size. With S_{4t}, all the additional terms in the score are correlated with all the performance estimates in the batch which is not the case when the batch size is doubled. An improvement in the efficient estimates is expected, as the minimum number of terms in the score associated with L_t is $m + 1$ rather than 1. These estimators can be considered as the method of batch means with a "window" of variable width as a function of different estimators using, for example, its own score function in that window, up to that window, or even through the whole process.

Another effective VRT is by *conditioning*. The idea behind conditional expectation is simple. If LS is the output of interest, then E(LS) = E{E{LS | Z)} and Var(LS) = E[Var(LS | Z)] + Var[E(LS | Z) for any random variable Z for which the conditional expectation exists. In other words, Var(LS) $\geq$ Var[E(LS | Z). An estimator based on E(LS) has a variance no larger than one based on LS. The approach is to compute E(LS | Z) analytically and to estimate the expected value of this quantity using Monte Carlo methods. Score function estimates of sensitivity are of the form $L(x)S(\theta)$. Note that

$$E_\theta[L(X)\,S(\theta)] = E_\theta\{E_\theta[L(X)|S(\theta)]S(\theta)\} \tag{32}$$

A new sensitivity estimate can be obtained by estimating the inner expectation, by generating the inputs X conditionally on the value of $S(\theta)$, and then by calculating the outer expectation. Thus, the estimator based on conditioning takes the form

$$\int L(X|S(\theta) = s)\, s\, f_{s(\theta)}(s)\, ds \tag{33}$$

where $f_{S(\theta)}$ (s) is the density of the score function. It may be necessary to evaluate this integral numerically, using the quadrature rule (21, and refs. therein). Other conditioning methods are proposed in Ref. 4.

A *control variate* is another alternative VRT. Consider SF estimates of the gradient of $J(\theta)$. It is natural to consider the efficient score function [Ln(f)]' as the control variate. Under some mild regularity conditions, use $E\{[dLn(f)/d\theta\}] = 0$. Therefore, as another estimator, Equation (20) can also be written as:

$$J'(\theta) = \int \{L(x)\} f'(x;\theta)\, dx = \mathrm{Cov}\,[L(X),\ S] \tag{34}$$

Using the usual estimate for this covariance can lead to large variance reduction (22). Another alternative is:

$$J'(\theta) = E[L(X)\,.\,S] + \alpha\, E[S] \tag{35}$$

where α could be the optimal linear control $\alpha^* = \mathrm{Cov}(L, S)/\mathrm{Var}(S)$, which can be estimated by substituting the usual estimates of variance and covariance for the true values.

Note also that the gradient can be written as

$$\begin{aligned} J'(\theta) &= \int \{L(x)\}\, f'(x;\theta)/f(x;\theta)\, dx \\ &= \int L(x) \frac{f'(x;\theta)}{\varphi(x;\theta), f(x;\theta)} \varphi(x;\theta) dx \end{aligned} \tag{36}$$

The best choice for φ is the one proportional to $L(x).|f'(x;\theta)|$. This function minimizes the variance of $J'(\theta)$, but this optimal φ depends on the performance function $L(x)$ which for most cases is not known in advance. One may use the empirical version of $L(x).|f'(x;\theta)|$.

The inherent variability of the SF makes variance reduction techniques extremely important if SF is to be useful. We have outlined a few of ways to run a simulation to obtain a SF estimate of the gradient of the performance measure. Although the type of simulation being run has a large effect on the precision of the sensitivity estimate, the precision of the estimate of performance, which is obtained at the same time, is insensitive to the type of simulation.

HARMONIC ANALYSIS (HA)

Another strategy for estimating the gradient simulation is based on the frequency domain method, which differs from the time domain experiments in that the input parameters are deterministically varied in sinusoidal patterns during the simulation run, as opposed to being kept fixed as in the time domain runs. The range of possible values for each input factor should be identified. Then the values of each input factor within its defined range should be changed during a run. In time series analysis, t is the time index. In simulation, however, t is not necessarily the simulation clock time. Rather, t is a variable of the model that keeps track of certain statistics during each run. For example, to generate the interarrival times in a queuing simulation, t might be the variable that counts customer arrivals.

In frequency-domain simulation experiments, a separate frequency ω is assigned to each factor. By basing the analysis on Fourier series, ω varies between 0 and 1/2 in multiples of $1/T$, where T is the number of observations. The highest frequency is obtained when ω is chosen to be 1/2, and the lowest frequency is achieved when ω is chosen to be $1/T$. The lowest frequency completes one cycle, whereas the highest frequency completes $T/2$ cycles during each run. The contribution of each frequency to the variability of a time series is measured by a spectrum function. Furthermore, the theory of linear systems indicates that a sinusoidal input to a linear system, in steady state, results in a sinusoidal output at the same frequency. Consequently, the output spectrum and the input factor spectrum are related.

The simulation performance function $L(t\,|\,\theta)$ can be approximated by two different meta-models: a polynomial meta-model and a trigonometric meta-model. The polynomial meta-model is obtained from a Taylor series expansion, whereas the trigonometric meta-model is obtained from a Fourier series expansion. The harmonic gradient estimator is then derived by matching or equating the coefficients of these two meta-models.

Frequency-domain simulation experiments identify the significant terms of the polynomial that approximates the relationship between the simulation output and the inputs. Clearly, the number of simulation runs required to identify the important terms by this approach is much smaller than those of the competing alternatives, and the difference becomes even more conspicuous as the number of parameters increases (23). The implementation steps are as follows:

Step 1: *Initialization*

Assume the input/output relationship for a simulation model can be approximated by a *quadratic dynamic polynomial response surface* meta-model (24). Set the simulation length T (even), the p oscillation frequencies $\{\omega_j\}$, and the p oscillation amplitudes $\{a_j\}$ $(j = 1, 2, \ldots, p)$. The oscillation frequencies are Fourier frequencies of the form $\omega_j = 2\pi h_j/T$, where $h_j \varepsilon \{1, 2, \ldots, T/2\} \subset Z^+$ are from the tables in Ref. 25 to ensure no confounding of any distinct frequencies when gradient components can be estimated. The simulation run length T is set such that $\max\{|\omega_j|\} < \beta$ for some $\beta > 0$ small. The p oscillation amplitudes $\{a_j\}$ are set such that each input parameter remains feasible and $\max\{|a_j|\} < \gamma$ for some $\gamma > 0$,. The value of T and $\{a_j\}$ should also be set based on information in Table 1 in Ref. 25 to allow the interchange of the derivative operator and the expectation operator.

Step 2: *Signal simulation run*

Make a signal simulation run by varying the p input parameters during the run, as follows:

$$\theta_j = \theta_j(0) + a_j \operatorname{Sin}(\omega_j t), j = 1, 2, \ldots, p,\ t = 1, 2, \ldots, T \tag{37}$$

Step 3: *Estimate signal run harmonic coefficients*

Compute

$$A_s(\omega_j) = (2/a_j T) \sum_{t=1}^{T} L_s(t) \operatorname{Sin}(\omega_j t),\ j = 1, 2, \ldots, p \tag{38}$$

Step 4: *Noise control variate simulation run*

Make a second (i.e., noise) simulation run with the p input parameters all held fixed during the run (i.e., $\theta_j = \theta_j(0)$, $j = 1, 2, \ldots, p$, $t = 1, 2, \ldots, T$). Common random number streams should be used for both simulation runs.

Step 5: *Estimate noise-run control-variate harmonic coef ficients*

Compute

$$A_N(\omega_j) = (2/a_j T) \sum_{t=1}^{T} L_s(t) \operatorname{Sin}(\omega_j t),\ j = 1, 2, \ldots, p \tag{39}$$

Step 6: *Set the control variate weighing parameters* $\eta\,(\omega_j)$

Compute

$$\eta(\omega_j) = \sum_{t=1}^{T} L(t|\theta(0) + a \operatorname{Sin}(\omega t) \times (Y(t|\theta(0))/\sum_{t=1}^{T} L^2(t|\theta(0) \tag{40}$$

Step 7: *Compute gradient estimate of the quadratic polynomial response surface*

Compute

$$A_s(\omega_j) - \eta(\omega_j) A_N(\omega_j),\ j = 1, 2, \ldots, p \tag{41}$$

A SHORT COMPARISON

L'Ecuyer (25) showed that PA and SF, (or LR), can be unified. He concluded that PA can be viewed as a (degenerate) special case of SF. Extensive comparison of the PA and SF approaches reveals several interesting differences. Both approaches require an interchange of expectation and differentiation. However, the conditions for this interchange in PA heavily depend on the nature of $L(\theta)$, and they must be verified for each application, which is not the case in SF. Therefore, in general, it is easier to satisfy SF unbiasedness conditions. However, SF requires $f(x,\ \theta) \neq 0$. PA assumes that the order of events in the perturbed path is the same as the order in the nominal path, for a small enough $\Delta\theta$, which allows calculation of $dL/d\theta$, the sensitivity of the sample performance for a particular simulation. For example, if the performance measure is the mean number of customers in a busy period, the PA estimate of the gradient with respect to any parameter is zero! The number of customers per busy period will not change if the order of events does not change.

In terms of the information required to implement these estimators, both approaches require knowledge of the cdf (or pdf) in which θ is as a parameter. SF needs to compute $\partial \ln f(x,\ \theta)/\partial\,\theta$. However, in PA it may be sufficient to know that θ is a scale or location parameter, without full knowledge of the associated event lifetime cdf. In addition, the nature of θ, which can only affect event lifetime distributions in PA, may be more general in the SF approach.

In terms of ease of implementation, PA estimators may require considerable analytical work on the part of algorithm developer, with some ”customization” for each application, whereas SF has the advantage of remaining a general definable algorithm whenever it can be applied. The derivation of the SF estimator, because $L(\theta)$ is directly observed and $\partial \ln f(x,\theta)/\partial\theta$ is readily evaluated; However, the evaluation of $dL/d\theta$ in PA usually requires some analysis and is not immediate. this aspect makes SF attractive. For discrete random variables (r.v.) the comparison is rather interesting. Consider a discrete r.v. that takes the values α_i with probability $p_i (i = 1, 2, \ldots, n)$. A SF algorithm can be developed with respect to the p_i parameters, but not the α_i parameters, whereas PA algorithms can be developed with respect to the α_i but not the p_i parameters.

Perhaps the most important criterion for comparison lies in the question of accuracy of an estimator, which is typically measured through its variance. If an estimator is strongly consistent, its variance is gradually reduced over time and ultimately goes to zero. The speed with which this happens may be extremely important. Because in practice, decisions normally have to be made in a limited time, an estimator whose variance decreases fast is highly desirable. For some simple systems (e.g., $J(\theta) = c$, with c a constant), where it is possible to compute *explicitly* variances of both PA and SF estimators, it can be shown that the variance of a SF estimator is significantly larger than that of its PA counterpart. In general, when PA does provide unbiased estimators, the variance of these estimators is small. PA fully exploits the structure of DESs and their state dynamics by extracting the needed information from the observed sample path, whereas SF requires no knowledge of the system other than the inputs and the outputs. Therefore, when using SF methods, variance reduction is necessary. The question is whether or not the variance can be reduced enough to make the SF estimator useful in all situations to which it can be applied. The answer is certainly yes. Using standard variance reduction techniques can help, but the most dramatic variance reduction occurs using new methods of VR such as conditioning (27), which is shown numerically to have a mean squared error that is essentially the same as that of PA.

It is much easier to extend the SF to higher derivatives, for example,

$$d^2 J(\theta)/d\theta^2 = E[L.S - d^2 \ln f(\theta, x)/d\theta^2] \tag{42}$$

Finally, the PA approach explicitly seeks to exploit the structure of event-driven sample paths in evaluating dL/ dθ. The SF approach does not attempt to do so. In this respect, SF is a general-purpose methodology for obtaining sensitivity of performance metrics of stochastic processes, not necessarily DES.

The SF estimate of the gradient can be related to the FD estimate. Equation (20) can be rewritten as

$$\begin{aligned}&\{\int \{L(x)\} f(x, \theta + \Delta\theta) dx - \int \{L(x)\} f(x, \theta)\, dx\}/\Delta\theta = \\ &\{\int L(x) \frac{f(x, \theta + \Delta\theta)}{f(x, \theta)} f(x, \theta)\, dx - \int \{L(x)\} f(x, \theta)\, dx\}/\Delta\theta \\ &\{E[L(X) \frac{f(X, \theta + \Delta\theta)}{f(X, \theta)}] - E[L(X)]\}/\Delta\theta\end{aligned} \tag{43}$$

However, the SF gradient estimate [Equation (20)] can be rewritten as

$$\begin{aligned}&\int L(x)\,[\partial \ln f(x, \theta)/\partial\theta]\, f(x, \theta) dx = E\{L(x)\,[\partial f(x, \theta)/\partial\theta]/f(x, \theta)\} = \\ &E\{[L(X)/f(X, \theta)]\, \mathrm{Lim}_{\Delta\theta \to 0}[f(X, \theta + \Delta\theta) - f(X, \theta)]/\Delta\theta\} = \\ &\mathrm{Lim}_{\Delta\theta \to 0}\{E[L(X) \frac{f(X, \theta + \Delta\theta)}{f(X, \theta)}] - E[L(X)]\}/\Delta\theta\end{aligned} \tag{44}$$

This shows that the SF estimate is the limit value of the FD as $\Delta\theta \to 0$.

Feuerverger *et al* (28) showed that if the performance depends on an indeterminate, possibly infinite number of inputs, then the SF gradient estimate is $2\pi f(L, S)$, where $f(L, S)$ is the *cross spectral density* of $\{L_i, S_i\}$ where S_i is the score of X_i. The main ideas and interrelationships of the various methods are shown in Fig. 4.

OPTIMIZING SIMULATED SYSTEMS

Simulation is the primary analysis tool for designing complex DESs. However, the simulation must be linked with a mathematical optimization technique to be used effectively for systems design.

Stochastic approximation procedures include Kiefer-Wolfowitz and Robbins-Monro types of techniques. Both types use a gradient estimate. The gradient can be estimated by any of the following approaches: finite difference, simultaneous perturbation, frequency domain, likelihood ratio, or infinitesimal perturbation analysis. Introductory concepts on gradient estimations may be found in Refs. 29–43. In Refs. 44 and 45, theoretical bases are provided.

Gradient surface methods combine the advantages of response surface methodology and efficient derivative esti-

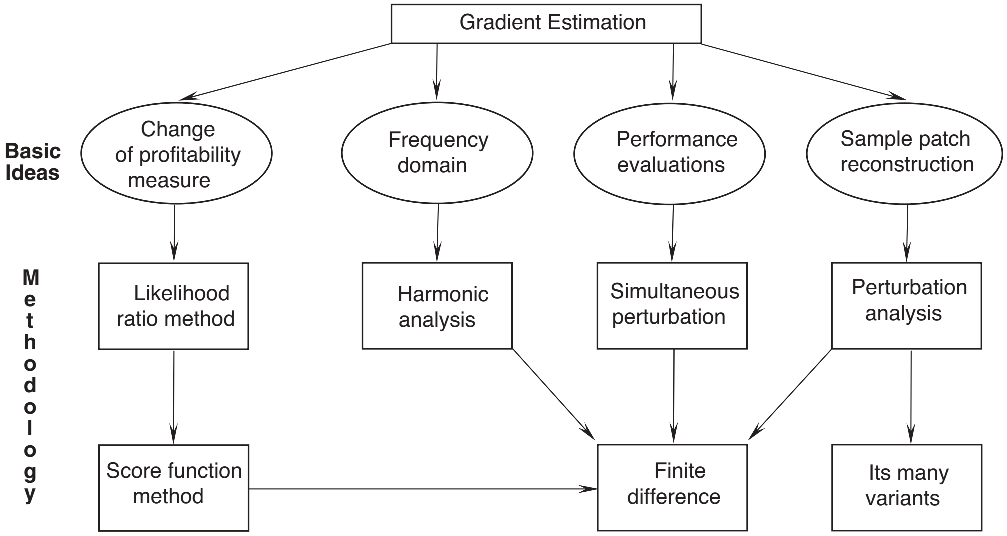


Figure 4. Classification and unification of gradient estimation methods.

mation techniques. Introductory discussions are given in Refs. 46 and 47.

STOCHASTIC APPROXIMATION TECHNIQUES

Two related stochastic approximation techniques have been proposed, one by Robbins and Monro (48) and one by Kiefer and Wolfowitz (8). The first technique was not useful for optimization until an unbiased estimator for the gradient was found. Kiefer and Wolfowitz (8) developed a procedure for optimization using finite differences. Both techniques are useful in the optimization of noisy functions, but they did not receive much attention in the simulation field until recently. Generalization and refinement of stochastic approximation procedures give rise to a weighted average (21), which deals with constraints, nondifferentiable functions, and some classes of nonconvex functions, among other things.

Kiefer-Wolfowitz Type Techniques

Kiefer and Wolfowitz (8) proposed a finite difference approximation to the derivative. One version of the Kiefer-Wolfwitz (K-W) technique uses two-sided finite differences that result in the following recursive procedure:

$$\theta_{n+1} = \theta_n - (\mathrm{a}_n/2\mathrm{b}_n)\,[\mathrm{J}(\theta_n + \mathrm{b}_n) - \mathrm{J}(\theta_n - \mathrm{b}_n)] \tag{45}$$

where the two sequences a_n and b_n satisfy the following conditions:

$$\sum \mathrm{a}_n = \infty \tag{46}$$

$$\lim\,(\mathrm{b}_n) = 0 \tag{47}$$

$$\sum (\mathrm{a}_n/\mathrm{b}_n)^2 < \infty \tag{48}$$

Then the sequence in Equation (45) converges in mean square and with probability 1 to a local optimum of $\mathrm{J}(\theta)$. However, because of the FD approximation for the gradient, the K-W procedure has a slow asymptotic convergence rate, which is typically of order $n^{-1/3}$. In general, any technique that attempts to estimate the gradient via finite differences will converge at a rate slower than $n^{-1/2}$ in the computational efforts (47).

The first fact to notice about the K-W estimate is that it requires $2N$ simulation runs, where N is the dimension of vector parameter θ. If the decision maker is interested in gradient estimation with respect to each of the components of θ, then $2N$ simulations must be run for each component of θ. This method is inefficient. The second fact is that it may have a very poor variance, and it may result in numerical calculation difficulties, because for small b_n, there may not be much statistical difference between estimates of $\mathrm{J}(\theta_n + \mathrm{b}_n)$ and $\mathrm{J}(\theta_n - \mathrm{b}_n)$, and we are dividing this difference by a small number. Finally, this estimate is unbiased for each nonzero b_n, because the slope of a secant line is approximating the slope of the tangent line. This result leads to the difficult question of which b_n to use. To minimize the "secant error," a small b_n should be used; but using such a small value contributes to the variance and numerical difficulties mentioned above.

Robbins-Monro Type Techniques

The original Robbins-Monro (R-M) technique is not an optimization scheme, but rather a root finding procedure for functions whose exact values are observed with noise. Its application to optimization is immediate: Use the procedure to find the root of the gradient of the objective function.

For simplicity, consider a simple parameter; that is, θ is a scalar. Let $\mathrm{H}(\theta)$ be the unknown function. The goal is to find θ^* such that $\mathrm{H}(\theta^*) = 0$. To do this, a sequence of experiments is performed with starting point θ_1 and successive values of θ according to the recursive relation

$$\theta_{n+1} = \theta_n - \mathrm{a}_n.\mathrm{H}(\theta_n) \tag{49}$$

where a_n is one of a sequence of decreasing and divergent positive numbers (such as $1/n$), and $H(\theta_n)$ is the noisy observation obtained from the nth experiment at θ_n. Assuming E $[H(\theta_n) \mid \theta_n = \theta] = H(\theta)$, and some additional mild conditions, the sequence $\{\theta_n\}$ converges to θ^* in mean square error.

Consider the case in which the parameter space is constrained to a bounded set Δ of the form $\Delta = (\theta\colon q_i(\theta) \leq 0, i = 1, \ldots, s)$, where $q_i(\theta)$ are continuously differentiable and Δ is the closure of its interior. Let X_(y) denote any closest point in Δ to y. The projected analog of Equation (16) is

$$\theta_{n+1} = X[\theta_n - a_n H(\theta_n)] \tag{50}$$

Interest was renewed in the R-M technique as a means of optimization, with the development of the perturbation analysis, likelihood ratio also known as score function, and frequency domain estimates of derivatives. Optimization for simulated systems based on the R-M technique is known as a "single-run" technique. These procedures optimize a simulation model in a single run simulation with a run length comparable with that required for a single iteration step in the other methods. This result is achieved essentially by observing the sample values of the objective function and, based on these observations, updating the values of the controllable parameters while the simulation is running (that is, without restarting the simulation). This observing-updating sequence is done repeatedly, which leads to an estimate of the optimum at the end of a single-run simulation. Besides having the potential of large computational savings, this technique can be a powerful tool in real-time optimization and control, where observations are taken as the system is evolving in time.

Generalizing Equation (49) to higher dimension, the Robbins-Monro type technique takes the following form:

$$\theta_{n+1} = \theta_n - a_n D_n \nabla J_n(\theta_n) \tag{51}$$

where a_n is the step size sequence, D_n is a matrix of appropriate dimensions, and $\nabla J_n(\theta_n)$ is the estimate of the gradient $\nabla J(\theta)$ at θ_n. The main idea behind Equation (51) is a trade-off based on the signal-to-noise ratio of the gradient estimate $\nabla J_n(\theta_n)$ to the error of the estimate. Initially, when θ_n is far from the optimal solution θ^*, this ratio is large, and a_n and D_n can be large. As θ^* is approached, notice $\nabla J(\theta) \to 0$, and the expectation of $\nabla J_n(\theta_n)$ is mostly noise. For convergence purposes, a_n should approach zero, but not too fast to avoid being stuck at a nonoptimum. These coefficients are referred to as step sizes, scaling factors, or learning rates. The R-M sequence is asymptotically normal, but it has some problems. It converges slowly when the function is very flat, and it may not converge when the function is steep. For example, solving the following equation $\exp(-3\theta) - 0.9\exp(-4\theta) - 0.1 = 0$, by R-M does not converge to its solution $\theta^* = 0.5073$.

More recently, R-M has been extended to multistep methods to provide the basic optimization technique for reasonable problems by "filtering" past observations. One such step uses *two independent* estimates:

$$\theta_{i+1} = \theta_i - a_n \left\{ \frac{J'_1(\theta_i)}{\min\{\varepsilon, |J^{1}{}_2(\theta_i)|\}} + \frac{J\_2'(\theta_i)}{\min\{\varepsilon, |J'_1(\theta_i)|\}} \right\} \tag{52}$$

where $\varepsilon > 0$ is the scaling factor, and J'_1 and J'_2 are the two independent estimates of J' at θ^i. When unbiased estimators are used, it converges at the rate of $n^{-1/2}$, which is the fastest. The technique is very sensitive to ε; that is, the smaller ε is, the more robust the technique is (29).

A practical concern is how to estimate the gradient $\nabla J_n(\theta_n)$ of the objective function $J(\theta)$, when the function $J(\theta)$ itself has to be estimated from simulation.

GRADIENT SURFACE METHOD

One may combine the gradient-based techniques with the response surface methods (RSMs) for optimization purposes. A response surface is constructed with the aid of n response points and the components of their gradients.

The gradient surface method (GSM) combines the virtue of RSM with that of the single-run, gradient estimation techniques such as perturbation analysis and the score function (or likelihood ratio) techniques. A single simulation experiment with little extra work yields $N + 1$ pieces of information (i.e., one response point and N components of the gradient). This method is in contrast to *crude* simulation, where only one piece of information, the response value, is obtained per experiment. Thus, by taking advantage of the computational efficiency of single-run gradient estimators, in general, N-fold fewer experiments will be needed to fit a global surface compared to the RSM. At each step, instead of using Robbins-Monro techniques to locate the next point locally, a candidate for the next point is determined globally, based on the current global fit to the performance surface. In particular, the gradient estimation information is used at this step and its previous steps, to find a function $\nabla J_n(\theta)$ to approximate the true gradient surface $\nabla J(\theta)$. Then, the zero point of $\nabla J_n(\theta)$ is found and taken as a candidate for the next iteration point, θ_{n+1}.

The GSM approach has the following advantages. The technique can quickly get to the vicinity of the optimal solution because its orientation is global (22, 39). Thus, it produces *satisfying* solutions quickly. Like RSM, it uses all accumulated information. And in addition, it uses gradient surface fitting, rather than direct performance response-surface fitting via single-run gradient estimators. This technique significantly reduces the computational efforts compared with RSM. Similar to RSM, GSM is less sensitive to estimation error and local optimality. And, finally, it is an on-line technique, which mean the technique may be implemented while the system is running.

A typical optimization scheme involves two phases: a search phase and an iteration phase. Most results in analytic computational complexity assume that good initial approximations are available, and the results deal with the iteration phase only. If enough time is spent in the initial search phase, then the time needed in the iteration phase is reduced. The literature contains papers that provide conditions for the convergence of a process (40); a process has to be more than convergent in order to be computationally interesting. It is essential to bound the cost of computation. In this sense, GSM can be thought of as helping the search phase and as an aid to bounding the cost of computation. Standard or simple devices can be adopted for such issues as stopping rules. It is a complementary tool for use in combination with other techniques.

Under smoothness and convexity conditions imposed on $J(\theta)$, the minimum can be found by solving $\nabla J(\theta) = 0$ that is, to optimize $J(\theta)$ is equivalent to finding the zero of $\nabla J(\theta)$, without a closed form expression for it. A linear function is used to estimate $\nabla J(\theta)$ directly fitting $J(\theta)$ by a quadratic surface. Although derivative information can also be used to fit the response surface, it is not done for the following reasons. Denote the gradient surface fitting function $F(\theta) = A\theta + b$, where A is an $N \times N$-dimensional matrix and b is a N-dimensional vector, it is popular to use the least-squares method to estimate A and b. Then, any quadratic response surface will yield gradient solutions, at most, as good as the least squares estimate but no better. Furthermore, fitting a quadratic surface will require one more parameter. Also, the linear function for the estimated gradient surface is used for convenience only; in principle, more complex surfaces can be used. Linear GSM, however, is satisfactory for small-size problems.

Assuming that we have $n(n \geq p)$ estimates of the gradient, the least-squares method provides the best estimate denoted by (A_{n+1}, b_{n+1}). The zero point of the gradient surface $(\theta_{n+1} = -A_{n+1}^{-1} \cdot b_{n+1})$ is taken as a candidate for the next iteration point. Surface fitting is accomplished based on the information about all n points (global behavior of the response surface). As a result, a small change in the estimation error at a particular point will have less effect on the result of surface fitting and its zero point than in ordinary gradient techniques, which can be regarded as a special case of surface fitting, in which only one point is used to define the gradient surface.

Initially, all data points are used to get an approximate global fit of the gradient surface (GS). As we get closer to the optimum, older data points are discarded and data points that are closer together are used to fit the GS and ensure convergence. These considerations suggest the use of a "window size" for surface fitting and also raise the following questions. First, how many points should be chosen to fit the GS? Second, at each iteration, should we use the same number of points to fit GS? And finally, whether a fixed window size or a dynamic window size is used, what is the rule to add or to drop points from the window; for example, should the latest L points where L is the size of the window always be used?

All of these questions suggest that there should be different phases in the search for an optimum, as follows. In the first step, initial points are randomly chosen. With no initial special knowledge, the minimum number of points is used to determine the GS. If the results of this fitting yield a solution that is not acceptable (e.g., the solution is outside the feasible region known to contain the optimal solution); the solution point is discarded. The window size is then increased by one by just randomly adding another new point. When an acceptable candidate is found, the window size is fixed. This iteration stage the "initial phase," or Phase I.

At the next stage, within the fixed window, each new candidate solution is generated using the same procedure as in Phase I. At this stage, it is determined whether the new candidate should be added. If yes, determine which old point should be dropped. Here is a dropping and adding rule. Assume the window size is fixed to be L. Remove the worst point; in minimization problem, $\theta_j = \max [J(\theta_{k-L+1}), \ldots, J(\theta_k)]$. If $j \neq k$; then the new candidate θ_k can be added. This stage of iteration is called Phase II. When θ_k cannot be added (i.e., θ_k is the worst point), then instead of using a randomly chosen new point, use the stochastic approximation techniques to get a new point and then switch back to Phase II. When each new candidate is very close to its predecessor, the optimum may be approaching. At this point, even switching back to Phase II may not produce a good-fitting function.

Then, using pure stochastic approximation techniques may provide a better approach to the optimal solution. This stage of iteration after switching is called the "ending phase" or Phase III. Switching back to Phase II avoids the use of the pure stochastic approximation technique at the beginning stage, which generally causes slow convergence. But to keep switching from Phase III to Phase II would waste simulation time. Hence, a maximum switching number (MSN) should be given. Actually, Phase III is just a degenerate gradient surface method, where the local gradient is used to define the gradient surface $g_k(\theta)$. The detailed algorithm is presented below:

Phase I

Step 0: Given N (minimum window size), M (maximum window size), MSN set $S = 0$, $L = N + 1$.

Step 1: Randomly choose N initial points $\theta_1, \theta_2, \ldots\ldots, \theta_N$.

Step 2: Use any single-run gradient estimators to find the corresponding derivatives $z(\theta_i)$, $i = 1, \ldots L-1$.

Step 3: Use the least-squares method to find the GSM function g_L with θ_i and $z(\theta_i)$, $i = 1, \ldots\ldots, L-1$.

Step 4: Find the zero point of g_L as θ_L.

Step 5: If $L > M$, then switch to Phase III; otherwise,

Step 6: If θ_k is not acceptable (e.g., θ_k does not satisfy some constraints), then discard θ_k and randomly choose a new point, $L = L + 1$, go to 2; otherwise, set the window size to L, and go to Phase II.

Phase II

Start with $k + L$.

Step 7: Use any single-run gradient estimation technique to find the corresponding derivatives $z(\theta_k)$.

Step 8: Remove the worst point $\theta_j = \text{worst}\,(\theta_1, \ldots\ldots, \theta_k)$; if $j = k$ then $S = S + 1$, switch to Phase III; otherwise, go to the next step.

Step 9: Use least-squares method to find the GSM function g_{k+1} with θ_i and $z(\theta_i)$.

Step 10: Find the zero point of g_{k+1} as estimate for θ_{k+1}

Step 11: Set $\theta_{k+1} = (1 - \rho_k)\theta_k + \rho_k\theta_{k+1}$, $0 < \rho_k < 1$ (53)

Step 12: If the stopping rule is satisfied, stop; otherwise, $k = k + 1$, go to 7.

Phase III

Step 13: Use one point to determine estimate

$$\theta_k = \theta_{k-1} - z(\theta_{k+1}) \tag{54}$$

$$\text{set} \quad \theta_k = (1 - \rho_k)\theta_{k-1} + \rho_k\theta_k \tag{55}$$

Step 14: Is $S > \text{MSN}$? No, switch back to Phase II.

Step 15: Is the stopping rule satisfied? If yes, stop; otherwise, $k = k + 1$, go to Step 13.

Notice that the maximum window size M is used to keep from infinitely iterating at Phase I. A different stopping rule may be chosen such as $|J(\theta_{k+1}) - J(\theta_k)| \leq \varepsilon$. Phase II uses the zero point of g_{k+1}, (θ_{k+1}) and the previous point θ_k to predict the new point θ_{k+1}. This is an alternative. If $\rho = 1$, θ_k is just the zero point of g_k, then implement the least squares solution exactly. Whenever $\rho < 1$, decide whether or not to go all the way to the zero point from the current point. This method enables control of how aggressively the least squares solution is implemented.

Similar remarks apply to Phase III. Phase III is simply the stochastic approximation technique. In this technique, use only one previous point to predict a new candidate. Determination of MSN depends on each problem. Usually, if derivative estimates do not require excessive computation, then MSN can be large. Otherwise, using a large MSN will waste simulation resources. The least-squares solution suggests second-order methods, such as the variable metric (VM) method or Davidon-Fletcher's method. The least squares solution of GSM is different in substance and in philosophy because $\theta_{k+1}-\theta_k$ need not be orthogonal to the previous descent direction as in the variable metric method. Furthermore, in GSM the matrix A is determined by using least-squares fit over many more points than the dimension of the θ vector, whereas VM is primarily a deterministic technique. Therefore, GSM is less a stochastic version of the variable metric method than a more efficient version of the response surface method via gradient estimators. Finally, the experimental design problem: In particular, given a window size N, how should points $\{\theta_i\}$, $i = 1, \ldots, N$ be chosen initially? In the technique given above, random sampling design can be used to get these points.

For online optimization, a new design in GSM called "single direction" design may be used. Because for online optimization it may not be advisable or feasible to disturb the system, random design usually is not suitable. Single-direction design gives a starting point and then uses gradient descent steps (i.e., Phase III), to determine the successive $\{\theta_2, \ldots, \theta_{m+1}\}$.

The primary consideration is whether the design is concentrated or widely spread. If the points of the design are concentrated, then most information comes from the estimates of the derivative. Using the performance measures alone gives no information about the optimum. With a design of this type, if the points are centered at the optimum, asymptotically no advantage exists to including the estimates of the performance with the estimates of the derivatives. When the design is not centered, an advantage is gained by including the performance estimates, if they are correlated with the derivative estimates. The higher the correlation, the stronger the advantage will be. As the estimates tend to be highly correlated, and the probability that the design is centered at zero, there is an advantage to including the performance estimates.

When the quadratic model may be assumed to be more than locally reasonable, and the points of the design may be spread out, the advantage clearly goes to the estimates which include the performances. Again, if the performance and derivative estimates are possibly correlated, then it is better to use both.

POSTSOLUTION ANALYSIS

Stochastic models typically depend on various uncertain and uncontrollable input parameters that must be estimated from existing data sets. Focus is given on the statistical question of how input–parameter uncertainty propagates through the model into output–parameter uncertainty. In the case when θ is a controllable or uncontrollable parameter, the decisionmaker is interested in estimating $J(\theta)$ for a small change in $\theta = \theta^*$ to $\theta = \theta^* + \Delta\theta$, where θ^* is the optimal solution or the solution to the goal-seeking problem; that is, the so-called postsolution analysis for the purpose of stability with respect to the solution θ^*, or the "what-if" analysis. This information can be obtained by the following single-run simulation perturbation technique.

Similar to Equation (1), the expected performance measure with perturbed parameter $\theta^* + \Delta\theta$ is:

$$J(\theta^* + \Delta\theta) = \int L(x).f(x; \theta^* + \Delta\theta)dx \tag{56}$$

$$= \int L(x)\frac{f(x; \theta^* + \Delta\theta)}{f(x; \theta)}f(x; \theta)dx \tag{57}$$

$$= E[L(x)].W \tag{58}$$

where the likelihood ratio

$$W = f(x; \theta^* + \Delta\theta)/f(x; \theta) \tag{59}$$

provided $f(x; \theta)$ does not vanish within the set Θ.

Similar to Equation (1), the expected performance measure with perturbed parameter $\theta^* + \Delta\theta$ is obtained while simulating the system at the nominal value θ^*, all in a single run. Notice that the expectation in Equation (58) is with respect to pdf, $f(x; \theta)$ (i.e., the nominal system). This result was achieved by using the Radon-Nikodym measure (49), in Equation (58) to transform all perturbed probability space to the nominal one. This estimator requires only one sample path of the system with parameter θ^*. Using Equation (58), estimates for any number of $\Delta\theta$ are computed simultaneously from a single-run simulation while simulating the nominal system. Because it adds only moderate computational cost, it is efficient.

Although this technique closely resembles Importance Sampling used as a VRT, its intent is different. The rationalization of Equation (58) is that the generated random vector X is roughly representative of X, with pdf $f(x;\theta^*)$. However, each of these random observations could also have hypothetically come from $f(x; \theta^* + \Delta\theta)$. The factor W weighs the observations according to this phenomenon. Regardless of the fact that this approach is not costly, it produces a larger variance than crude simultaneous perturbation, which reruns the simulation for each $\Delta\theta$. Because for regular probability functions,

$$\mathrm{E[W]} = \int \frac{f(x;\theta^* + \Delta\theta)}{f(x;\theta)} f(x;\theta)dx = \int f(x;\theta^* + \Delta\theta)dx = 1 \tag{60}$$

Equation (58) can be written as

$$J(\theta^* + \Delta\theta) = E[L(x)W] = E[L(x).W]/E(W) \tag{61}$$

Other alternative estimators could be:

$$\hat{J}(\theta^* + \Delta\theta) = \sum(L_i\, W_i)/\sum W_i \tag{62}$$

$$\hat{J}(\theta^* + \Delta\theta) = \sum(L_i W_i)/n + \alpha\Big[\sum W - n\Big]\sum L_i \tag{63}$$

and

$$\hat{J}(\theta^* + \Delta\theta) = \sum(L_i\, W_i)/\sum W_i + \alpha\Big[\sum W_i - n\Big]\sum L_i \tag{64}$$

where the sums are over all i, $i = 1, 2, 3,\ldots, n$, and α is the usual *optimal linear control factor*. As always, some pilot runs must be performed to study the effectiveness of these and other filters before implementing them. These estimators have been shown to have less variation than the crude one based on Equation (58) using M/G/1 queues in Ref. 22 and refs. therein. Some of these estimators are biased, but they give reasonable results if the perturbation is small and $L(\theta)$ is well behaved. Clearly, jackknifing and bootstrap may be used to reduce the bias substantially. Other single-run perturbation techniques are proposed in Ref. 22. The ideas in this section are also useful as a *single-run perturbation* to be embedded in optimization techniques, such as simplex type techniques, random search techniques, and simulated annealing, in the earlier stage of implementation. This method saves some computational costs.

THE FUTURE OF OPTIMIZATION BY SIMULATION

With the growing incidence of computer-based modeling and simulation in many diverse fields, such as business, economic and government systems (50), the scope of simulation domain must be extended to include much more than traditional optimization techniques. Optimization techniques for simulation must also account specifically for randomness inherent in estimating the performance measure and satisfying the constraints of stochastic systems.

Given the input parameters θ (which could be a vector) and a performance measure $J(\theta)$, the problem of estimating the sensitivities (i.e., gradient, Hessian, etc.) of $J(\theta)$ with respect to θ can be tackled through FD, SP, PA, the SF Method, the LR Method, and HA, as shown in Fig. 4.

Sensitivity estimation is useful for local information, structural properties, response surface construction, goal-seeking problem, and optimization of DES simulation. This aticle described step-by step the existing methods of sensitivities estimators with the aim of theoretical unification and a few comparisons. All algorithms were presented in English-like format and therefore can be implemented in a variety of operating systems and machines, which provides unlimited portability.

PA yields an unbiased estimate if the interchange of derivative and expectation is permitted. This method imposes constraints on the nature of the sample function and the structure of the system that can be analyzed. For SF and LR, this condition is easier to satisfy. The advantage of SF and LR methods is that estimates of any number of derivatives (gradient) can be computed in a single-run simulation. The major difficulty in applying the SF and LR methods is that the variance of derivatives estimator increases as the length of the simulation increases. Therefore, effective variance reduction techniques must be used to make SF and LR competitive with PA. The PA fails to provide unbiased estimates when the sample function $L(\theta)$ is discontinuous in θ. This problem can be overcome at the expense of obtaining more information during simulation run. Implementation of PA and many of its variants are all problem dependent, whereas SF, LR, FD, SP, and HA are not. However, SP and FD can be seriously limited by, for example, the required stability of the system. For example, traffic intensity must remain less than one for steady-state sensitivity estimation.

Most work has been performed for θ being the parameters of input distribution, and strong theoretical results are available when these methods are implemented on relatively simple systems. More extensions are required to estimate sensitivities for more complex systems. Comparisons between different methods can rarely be performed theoretically in terms of bias, variance, and computational complexity. Many studies (51–53) rely on computer simulations to compare different methods. However, computational complexity in terms of the total computational effort, for reduction in both the bias and variance of the gradient estimator depend on the computational budget allocated for any simulation application. Simulation time is a crucial bottleneck in the sensitivity estimation. Future directions for research in gradient estimation lie in parallelization and distributed versions of the techniques discussed in this study. For example, gradient estimation by these methods for a computer system may require days to simulate only a few second of the system's operation.

We described the most widely used optimization techniques that can be integrated effectively with a simulation model. We also described techniques for postsolution analysis with the aim of theoretical unification of the existing techniques. All techniques were presented in step-by-step format to facilitate implementation in a variety of operating systems and computers, which improves portability.

General comparisons among different techniques in terms of bias, variance, and computational complexity are not possible. However, a few studies rely on real computer simulations to compare different techniques in terms of accuracy and number of iterations. Total computational effort for reduction in both the bias and variance of the estimate depends on the computational budget allocated for a simulation (54, 55).

No single technique works effectively and/or efficiently in all cases. The most promising techniques are the stochastic approximation, simultaneous perturbation, and the gradient surface methods. Stochastic approximation techniques that use perturbation analysis, score function (or likelihood ratio), or simultaneous perturbation gradient estimators optimize a simulation model in a *single simulation run*. These techniques do so by observing the sample values of the objective function; based on these observations, the stochastic approximation techniques update the values of the controllable parameters while the simulation is running and without restarting the simulation. This observing-updating sequence, done repeatedly, leads to an estimate of the optimum at the end of a single-run simulation. Besides having the potential of large savings in computational effort in the simulation environment, this technique can be a powerful tool in *real-time* optimization and control, in which observations are taken as the system is evolving over time.

Response surface methods have a slow convergence rate, which makes them expensive. The gradient surface method combines the advantages of the RSMs and efficiency of the gradient estimation techniques, such as infinitesimal perturbation analysis, score function (or likelihood ratio), simultaneous perturbation analysis, and frequency domain technique. In the GSM, the gradient is estimated, and the performance gradient surface is estimated from observations at various points, similar to the RSM. Zero points of the successively approximating gradient surface are then taken as the estimates of the optimal solution. GSM is characterized by several attractive features: It is a single-run technique and more efficient than RSM; at each iteration step, it uses the information from all data points rather than just the local gradient. It tries to capture the global features of the gradient surface and thereby quickly arrives in the vicinity of the optimal solution, but close to the optimum, it takes many iterations to converge to stationary points. Search techniques are therefore more suitable as a second phase. The main interest is to figure out how to allocate the total available computational budget across the successive iterations.

We expect the four areas for future research in the field of optimization techniques for simulation:

1. Computational studies of techniques presented in this article, for systems with many of controllable parameters and constraints.
2. Effective combinations of several efficient techniques to achieve the best results under constraints on *computational resources*.
3. Development of parallel and distributed schemes
4. Development of an expert system that incorporates all available techniques.

With respect to the third area of future research above, not all techniques lend themselves to parallel and distributed schemes. The last item in the above list would allow a framework for comparing and selecting the right technique(s) for the problem at hand. Realistic visual interfaces could also be devised and linked to the simulation prescriptive and postprescriptive process to provide an accurate visual representation.

ACKNOWLEDGMENTS

I am most appreciative to Dr. David Proctor and the reviewers for their suggestions and comments. This work is supported by the National Science Foundation under grant CCR-9505732.

BIBLIOGRAPHY

Note: Additional and interesting references in particular applications in the areas of systems engineering and general network system are available at: http://home.ubalt.edu/ntsbarsh/Business-stat/RefSim.htm.

1. G. Guariso, M. Hitz, and H. Werthner, An integrated simulation and optimization modelling environment for decision support, *Decision Support Systems*, **16**, 103–117, 1996.
2. J. Kleijnen and R. Rubinstein, Optimization and sensitivity analysis of computer simulation models by score function method, *Eur. J. Operational Res.*, **88**, 413–427, 1996.
3. J. Dussault, D. Labrecque, P. L'Ecuyer, and R. Rubinstein, Combining the stochastic counterpart and stochastic approximation methods, *Discrete Event Dynamic Sys. Theory Appl.* **7**, *5–28, 1997.*
4. R. Rubinstein and B. Melamed, *Modern Simulation and Modeling*, New York: Wiley, 1998.
5. R. Rubinstein and A. Shapiro, *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*, New York: Wiley, 1998.
6. P. Sadegh, Constrained optimization via stochastic approximation with simultaneous perturbation gradient approximation, *Automatica*, **33**, 889–892, 1997.
7. D. Pierre and M. Lowe, *Mathematical Programming Via Augmented Lagrangians: An Introduction with Computer Programs*, Reading, MA: Addison-Wesley, 1975.
8. J. Kiefer and J. Wolfowitz, Stochastic estimation of the maximum of a regression function, *Annals Math. Stat.*, **23**, 462–466, 1952.
9. J. Spall, Multivariate stochastic approximation using a simultaneous perturbation gradient approximation, *IEEE Tran. Auto. Control*, **37**, 332–341, 1992.
10. J. Spall, A one-measurement form of simultaneous perturbation stochastic approximation, *Automatica*, **33**, 109–112, 1997.
11. X-R. Cao, Perturbation analysis of discrete event systems: Concepts, algorithms, and applications, *Eur. J. Operational Res.*, **91**, 1–13, 1996.
12. M. Fu, Optimization via simulation: A review, *Annals Operations Res.*, **53**, 199–247, 1994.
13. M. Fu, and J-Q. Hu, *Conditional Monte Carlo: Gradient Estimation and Optimization Applications*, Norwell, MA: Kluwer, 1997.
14. F. Vazquez-Abad, Sensitivity analysis for stochastic DEDS: An overview, *Aportaciones Matematicas, Notas de Investigacion*, **7**, 163–182, 1992.

15. R. Suri, Perturbation analysis: The state of the art and research issues explained via the GI/G/1 queue, *Proc. of IEEE*, **77**, 114–137, 1989.

16. P. Glasserman, *Gradient Estimation via Perturbation Analysis*, Norwell, MA: Kluwer, 1991.

17. C. Cassandras, *Discrete Event Systems: Modeling and Performance Analysis*, Irwin, MA: CRC Press, 1993.

18. P. L'Ecuyer, On the interchange of derivative and expectation for likelihood derivative estimation, *Manage. Sci.*, **41**, 738–748, 1995.

19. H. Arsham, A. Feuerverger, D. McLeish, J. Kreimer, J. and R. Rubinstein, Sensitivity analysis and the what-if problem in simulation analysis, *Int. J. Math. Comput. Mode.*, **12**, 193–219, 1989.

20. P. Glynn, Likelihood ratio gradient estimation for stochastic systems, *Commun. ACM*, **33**, 75–84, 1990.

21. J. Dippon and J. Renz, Weighted means in stochastic approximation of minima, *SIAM J. Control Optimiz.*, **35**, 1811–1827, 1997.

22. H. Arsham, Performance extrapolation in discrete event systems simulation, *J. Syst. Sci.*, **27**, 863–869, 1996.

23. Y. Ho, S. Leyuan, D. Liyi, and W. Gong, Optimizing discrete event dynamic systems via the gradient surface method, *Discrete Event Dynamic Syst: Theory Appl.*, **2**, 99–120, 1992.

24. S. Jacobson, Convergence results for harmonic gradient estimators, *ORSA J. Comput.*, **6**, 381–397, 1994.

25. S. Jacobson, A. Buss, and L. Schruben, Driving frequency selection for frequency domain simulation experiments, *Operations Res.*, **39**, 917–924, 1991.

26. P. L'Ecuyer, A unified view of the IPA, SF and LR gradient estimation techniques, *Manage. Sci.*, **36**, 1364–1383, 1990.

27. F. Liang, *Weighted Markov Chain Monte Carlo and optimization*, Doctoral Dissertation Shatin, Hong Kong, Chinese University of Hong Kong, 1997.

28. A. Feuerverger, D. McLeish, and R. Rubinstein, A cross spectral method for sensitivity analysis of computer simulation models, *Comtes Rendus: Mathematical Reports Academy Sciences, Royal Society Canada*, **8**, 335–339, 1986.

29. S. Andradottir, Simulation optimization, in *Handbook on Simulation*, J. Banks (ed.), New York: Wiley, 1998.

30. V. Borkar, Asynchronous stochastic approximations, *SIAM J. Control Optimization*, **36**, 224–239, 1998.

31. M. Johnson and J. Jackman, Infinitesimal perturbation analysis: A tool for simulation, *J. Operational Res. Soc.*, **40**, 243–254, 1989.

32. M. Nakayama and P. Shahabuddin, Likelihood ratio derivative estimation for finite-time performance measures in generalized semi-Markov processes, *Manage. Sci.*, **44**, 1426–1441, 1998.

33. R. Hurrion, An example of simulation optimization using a neural network metamodel: Finding the optimum number of kanbans in a manufacturing system, *J. Operational Res. Soc.*, **48**, 1105–1112, 1997.

34. P. L'Ecuyer, N. Giroux, and P. Glynn, Stochastic optimization by simulation: Some experiments with the M/M/1 queue in steady-state queue, *Manage. Sci.*, **40**, 1245–1261, 1994.

35. H. Walk, Foundations of stochastic approximation, in *Stochastic Approximation and Optimization of Random System*, L. Ljung, G. Pflug, G., and H. Walk, (eds.), Basel, Switzerland: Birkhauser, 1992.

36. Y. Ho, *Discrete Event Dynamic Systems*, New York: IEEE press, 1992.

37. R. Suri and M. Leung, Single run optimization of discrete event simulation-An empirical study using the M/M/1 queue, *IIE Trans.*, **21**, 35–49, 1989.

38. A. Shapiro, Simulation-based optimization: Convergence analysis and statistical inference, *Commun. Stat. Stochastic Mod.*, **12**, 425–432, 1996.

39. M. Safizadeh, Optimization in simulation: Current issues and the future outlook, *Naval Res. Logistics*, **37**, 807–825, 1990.

40. G. Pflug, *Optimization of Stochastic Models: The Interface Between Simulation and Optimization*, London, UK: Kluwer, 1996.

41. S. Jacobson and L. Schruben, Techniques for simulation response optimization, *Operations Res. Letters*, **8**, 1–9, 1989.

42. Y. Ermoliev and V. Norkin, Normalized convergence in stochastic optimization, *Annals Operations Res.*, **30**, 187–198, 1991.

43. A. Benveniste, M. Metivier, and P. Priouret, *Adaptive Algorithms and Stochastic Approximations*, New York: Springer-Verlag, 1990.

44. D. Clark, Necessary and sufficient conditions for the Robbins-Monro method, *Stochastic Processes and their Applications*, **17**, 359–367, 1984.

45. S. Robinson, Analysis of sample-path optimization, *Math. Operations Res.*, **21**, 513–528, 1996.

46. J. Donohue, E. Houck, and R. Myers, Simulation design for the estimation of quadratic response surface gradients in the presence of model mis-specification, *Manage. Sci.*, **41**, 244–262, 1995.

47. P. Heidelberger, X. Cao, M. Zazanis, and R. Suri, Convergence properties of infinitesimal perturbation analysis estimates, *Manage. Sci.*, **34**, 1281–1302, 1988.

48. H. Robbins and S. Monro, A stochastic approximation method, *Annals Math. Stat.*, **22**, 400–407, 1951.

49. H. Arsham, J. Kreimer, and R. Rubinstein, Application of Radon-Nikodym theorem for simulation of queueing system, in *Discrete Event Simulation and Operations Research*, H. Adelsberger and F. Broceckx (eds.), Belgium: SCS Publisher, 1987.

50. R. Paul, *Simulation in Action*, London, UK: Springer-Verlag, 1998.

51. M. Fu, J. Hu, and R. Nagi, Comparison of gradient estimation techniques for queues with non-identical servers, *Comput. Operations Res.*, **22**, 715–729, 1995.

52. T. Lacksone and P. Anussornnitisarn, Empirical comparison of discrete event simulation optimization techniques, *Proc. of the 27th Annual Summer Computer Simulation Conference*, 1995, pp. 96–101.

53. D. Chin, Comparative study of stochastic algorithms for system optimization based on gradient approximation, *IEEE Trans. Syst. Man, Cybernetics - Part B: Cybernetics*, **27**, 244–249, 1997.

54. H. Arsham, Monte Carlo techniques for parametric finite multidimensional integral equations, *Monte Carlo Methods Appl.*, **13**, 173–195, 2007.

55. H. Arsham, Input parameters to achieve target performance in stochastic systems: A simulation-based approach, *Inverse Problems Eng.*, **7**, 363–384, 1999.

HOSSEIN ARSHAM
University of Baltimore
Baltimore, Maryland

INFORMATION AGE

INTRODUCTION

The Information Age is a period of time, or era, in which information itself, in its various forms, constitutes a key or dominant ingredient in our delivery of products and services. The Information Age began to emerge in a most serious manner about halfway through the twentieth century, when, for the first time, white collar workers started to outnumber blue collar employees. This newly born and growing Information Age was facilitated, in large measure, by

- pre-existing successes in telegraphy, telephony, radio, and television;
- bringing large-scale computers into our businesses, led by IBM;
- the success of xerography and copying machines, notably by Xerox; and
- the response to the challenge represented by the 1957 launch of Sputnik by the Russians.

All of the above provided initial fuel for the Information Age engine by proving the importance of information-related technologies and demonstrating their economic value in the marketplace. Two additional and extremely significant sets of events included the refinement and market penetration of

- the personal computer (PC) along with its resident applications, and
- connectivity and the Internet.

These events were acknowledged, in the aggregate, as"trends that were shaping the 1980s" in shifting from an industrial society to an information society (1). In doing so, at least four key points were considered to be critical:

- the reality and influence of the information-oriented society,
- computers and communications coming together to support each other,
- new information technologies migrating from old industrial activities to new processes, and
- the need for our education system to step up to difficult challenges associated with the new information orientation.

The above also supported and empowered the"triumph of the individual" (2) in moving from the twentieth into the twenty-first century. As the information age is brain-intensive in distinction to being capital-intensive, the individual with the right idea at the right time has a real opportunity to bring new information-related products and services into the marketplace. At the same time, this distinction has placed pressure on the individual to recognize and deal effectively with new problems as well as opportunities (3).

THE PERSONAL COMPUTER (PC)

The personal computer was a major factor in supporting the Information Age during its early years. In the late 1970s and early 1980s, the PC grew from a sophisticated hobby (Altair, Commodore) into a marginal but useful device (Apple II, TRS-80) and then into a serious home and business machine (IBM PC, Macintosh), with a variety of available software applications (e.g., word processors, spreadsheets, databases) (4). The elementary hardware architecture expanded, and soon we had powerful workstations such as those provided by Apollo Computer and Sun Microsystems. But the PC distinguished itself by continuously improving its performance while retaining its competitive and accessible price (e.g., less than $3,000) so that businesses did not hesitate in providing one to essentially all professional employees. In addition, more and more households found it to be an attractive asset for all members of the family. This unprecedented double-front (office and home) assault had a lot to do with ushering in and sustaining the Information Age. Increasing numbers of people, from all walks of life, at home and at the office, were becoming comfortable with, and dependent on, the PC. And the PC manufacturers, sharpening their products in a flourishing market, responded well to the challenge as new and successful companies were born and thrived (e.g., Compaq, Dell).

The combination of high performance, low price, and extensive software availability changed forever the way in which large numbers of people were dealing with computers and information. Setting a firm foundation for the Information Age, the PC, from palm to lap to desktop, was arguably the most critical ingredient. And the peculiar nature of affordable software, the PC's heart and soul, literally gave"power to the people" in ways never experienced before.

INFORMATION SYSTEMS

From the first"Killer App" (application) known as Visicalc, a spreadsheet produced by Dan Bricklin and Bob Frankston (4), large numbers of software developers have been building new and better application packages for the PC. Early software addressed a single function such as a spreadsheet or a word processor. Soon, multi-function packages were offered such as Microsoft's Office and Lotus' Smartsuite. Application areas continued to spring up that developers felt could be used by general businesses (e.g., accounting packages) as well as the home user (e.g., an encyclopedia). These areas were further expanded to meet

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Table 1. Illustrative Examples of Types of Information Systems

1. accounting	16. office automation
2. transportation	17. network browser & search engines
3. logistics support	18. process reengineering
4. risk management	19. information security
5. contracts management	20. data mining/warehousing
6. financial management	21. presentation graphics
7. geographic information	22. sales and marketing systems
8. legal information	23. decision support
9. enterprise resource planning (ERP)	24. point-of-sale systems
10. configuration management	25. word processing
11. database management	26. project management
12. inventory control	27. voice recognition
13. human resources	28. operations management
14. correspondence tracking	29. purchase/expenditure tracking
15. executive information	30. utility systems

the needs of both government and industry and appeared under the general category of"information systems." However, despite this broad title, they came in numerous subtypes, as illustrated by the list in Table 1. Based on the first 50 years or so of the Information Age (the last half of the twentieth century), there is good reason to believe that even the various system types cited in Table 1 will be expanded substantially during the twenty-first century as the Information Age continues to mature and express itself.

CONNECTEDNESS AND THE INTERNET

The next most powerful force in bringing about the Information Age, given the PCs and workstations, relates to the fact that we learned how to establish computer interconnections in an efficient and cost-effective manner. The stand-alone desktop computer would become only one of many nodes in a network that provided a means for people to communicate and exchange massive amounts of data. As we moved from local area networks (LANs) to wide area network (WANs) to the Internet, all of this connectivity greatly enhanced information exchange and productivity. For example, Lotus Notes was an effective way for everyone within an enterprise to communicate; the Internet was the way to communicate person-to-person(s) and enterprise-to-enterprise, all over the world, and with great speed.

THE ROLE OF SOFTWARE

Underlying the computers and networks, and ultimately the Information Age, is software. As it ranges from operating system to application package to multi-functional integrated information system, software is the "central intelligence" that makes all of it work. Behind the power of the software is the seminal idea, brainpower, and skill of the software creator, the new"warrior" of the Information Age.

Considering the central position of software as we move into the twenty-first century, it remains one of our most serious unsolved problem areas. We are getting better at it, and we are also creating new approaches to it, as in moving from procedural languages to object-oriented techniques (e.g., from "C" to "C++"). To some extent, it is a confusion of plenty, in search of a better way that most will use and benefit from. So whether it be reuse, or the more widespread use of commercial-off-the-shelf (COTS) software, or the renaissance of artificial intelligence (AI), or yet another development paradigm, the software challenge is likely to have many new solutions fitted to it as we move through the Information Age.

EDUCATION

Our education system deals generically with both information and learning and so must at least be cited as an important part of the Information Age. First, our educational institutions need to evolve new and better ways to prepare us to function as well as possible in the Information Age. To accomplish this feat, they must understand what the Information Age is and where it might be going. They need to stay close to industry, as much of it will be in uncharted territories driven by businesses. Second, they need to carry out applied research that will go beyond the foundations of the Information Age and into the problems that industry is and will be facing. Other areas of concern and interest in which our colleges and universities can participate are addressed below.

IMPORTANT ISSUES OF THE INFORMATION AGE

In Need of a Theory

Although we are continually producing information systems, we do not have an adequate "theory" of information that will explain what it is, how it provides value and influence, and how, for example, to convert information into knowledge. In the mid-1950s, a theory was formulated by the name"information theory," but it was applied largely to matters of analyzing and building communications channels and systems (5). One would hope and expect that the academic world will be able to provide the basis for a theory that will help business and government navigate through the Information Age.

Information Security

Information Security may turn out to be the most serious problem of the Information Age. It has a large number of dimensions, ranging from assuring that people are not able to access your bank account to preventing viruses from crashing your system, however large or small, to maintaining your personal and corporate privacy (6). One can be sure that lack of security will be an invitation to prankster as well as criminal intrusion, and all sectors of society must be concerned, from businesses to government to academia to the individual. This problem will also be exacerbated by the continued introduction and expansion of wireless systems.

Information Junk Mail and Overload

Along with all the true and useful information will come increasing amounts of junk (SPAM) as well as information overload. Humans, we know, have limited personal information handling and processing speeds and capabilities. We will therefore need better ways to rapidly discern true information from junk, discard the latter, and deal with the former. Filters to help us through this process are and will continue to be available; but they too may be defeated by the determined junk-mailer.

Information and Intelligence

An integral part of the Information Age is to add the appropriate amounts and types of intelligence to our products and services. Various products, therefore, will“know” some things that are best“remembered” by the product instead of the owner or user. Products will have the improved ability to self-diagnose difficulties in their own operation, pointing the way for the user to initiate a simple switch reset or other type of automated repair routine. Companies will have to understand how much intelligence to add, such that they will constantly be providing additional value to the consumer. These patterns have already shown themselves, for example, in the automobile, but can be expected to be greatly extended during the twenty-first century.

Human Interaction

As suggested above, in the Information Age, both products and services will incorporate unprecedented types and forms of information for use by industry, government, academia, and the individual consumer. In each domain, humans will find themselves part of the loop and at least the following two pressure points will be evident:

- the information transfers and systems will have to take full account of possible ways to optimize the human interactive role, and
- the humans will have to adapt (modify their own behavior) to maintain high levels of utility, effectiveness, and productivity.

Decision support systems in industry are good examples of information systems for which the above factors will be of increasing importance (see the systems listed in Table 1).

Information Technology versus Information Need

Trends during the last twenty years of the twentieth century exhibit an expansive growth of the technology that is able to organize and process bits and bytes ever faster and less expensively. This “technology push,” however, has often outpaced the needs of users, resulting in a gap between what the information technology has provided and what the user truly requires, which has led to overpromising and underdelivering from the perspective of the user. As the information age matures in the twenty-first century, an important issue is whether this gap will, on the whole, increase or decrease. If the former, we will have more solutions in search of a problem, and more problems going unsolved. Enterprises that understand and reduce the gap are likely to outdistance their competitors.

Building Software

As noted earlier, software is likely to remain the underlying force that gives life to the Information Age. We may therefore expect that there will be frontal attacks on the problem of improving the effectiveness and efficiency of software development, with new paradigms and languages coming on the scene and older ones (e.g., C, C++, Java) reengineered for improvements. Promising approaches might well lie in the following directions:

- extensive reuse of software components;
- software that is able to write new software;
- improved notions relative to software systems architecting;
- better metrics that aid in software decomposition, design, and management;
- integrated computer-aided software engineering (CASE) tools; and
- enhanced software team performance.

The reader is likely to find additional related notions in many subject areas of this encyclopedia.

Valuing the Information Enterprise

The economics of the Information Age, in several important dimensions, need to be substantially clarified. One such dimension is the Internet-based enterprise in which many of the prior rules for valuation are being rewritten. Another has to do with the perceived value to the consumer of adding information to a variety of products and services. A third aspect is that of the degree to which historical growth patterns in revenues and profits will be important. The business marketplace is likely to point the direction toward some of these answers. Others, hopefully, will be developed in our universities and financial institutions.

e-Commerce

Electronic commerce (e-commerce) includes any and all ways in which computers and networks are used to rapidly transfer information between a variety of corporate and individual users, which includes electronic data interchange (EDI), electronic funds transfer (EFT), electronic mail (e-mail), on-line catalogs and databases, and the use of the World Wide Web (www) and Internet (7). A part of e-commerce involves business-to-business interactions for purposes of distributing products and information related thereto. Another dimension relates to business-to-consumer transactions (such as amazon.com). Both modes of interaction have been growing and show no signs of abatement. All of the above uses of e-commerce need to be assisted by standards as well as the appropriate levels of security, with its related technologies (e.g., encryption), to maintain trust and viability throughout the Information Age.

Creating and Maintaining the Learning Organization

In the Information Age, it will be particularly important to assure the health and well-being of the Learning Organization (8). This statement would appear to be self-evident as new types and forms of software and information are created and used. The five disciplines that form the basis for the Learning Organization are as follows:

- personal mastery,
- mental models,
- building shared vision,
- team learning, and
- systems thinking.

Enterprises that fail to master the Information Age requirements for learning are not likely to maintain their competitiveness in a fast-changing world. A vibrant learning organization, however, is also likely to be a necessary but insufficient condition for success.

Business Adaptation and Transformation

Related to the above matter of the learning organization is the issue of business and government adaptation as well as transformation (i.e., how well, and how quickly, these take place). Main line businesses (e.g., automobile dealers, hardware stores, appliance manufacturers) will have to answer at least the following key questions:

- How can I utilize the available information technology and systems so as to improve the effectiveness and efficiency of my business?
- How can I extend and advance current information systems to assure that I stay ahead of my competitors?

Businesses that adapt to the changing environment and transform themselves with respect to the integration of information systems of all types are likely to flourish in the Information Age. Government organizations will have to do much the same in order to survive over the long run.

Knowledge Management

Knowledge Management (KM) may be defined as“the way companies generate, communicate, and leverage their intellectual assets” (9). This relatively new term is part of a progression that starts with data, operates upon that to produce information, and then engages in some process that uses such information as an important element in creating knowledge. In this context, knowledge may be viewed as a meta-form of information. Knowledge may also be thought of as that which is created when various“-packets” of information are combined to establish a new level of understanding that did not exist previously. Considerable efforts are underway to try to grasp what it is that constitutes knowledge, the specific stages of knowledge development and how organizations need to approach the matter of assuring the creation and application of knowledge (9). Many of the foundations of and issues related to Knowledge Management are being examined on a continuous basis (e.g., *Knowledge Management* magazine; see www.kmmag.com). Knowledge Management is considered to be an important field that will have to be further explored and instantiated as we continue to move through the Information Age.

Management in the Information Age

Management of our enterprises during the Information Age will have to meet the challenges represented by the above-cited issues, but also may expect organic changes to be taking place in these enterprises. As an example, Peter Drucker, one of our leading management commentators, suggests the following (9):

- a very sharp reduction in the number of management levels and the total number of managers (e.g., one-half the levels, and one-third the managers);
- the strong emergence of specialists to be able to create information, which he defines as “data endowed with relevance and purpose”;
- reliance on task forces (of the above specialists) that will transcend traditional departmental structures;
- providing a sufficient vision as well as motivation to unify an organization of information specialists;
- making sure that top management people are prepared and tested for success in the information-based organization; and

- building the highly competitive enterprise of the Information Age is the "managerial challenge of the future."

BIBLIOGRAPHY

1. J. Naisbitt, *Megatrends*, New York: Warner Books, 1982.
2. J. Naisbitt and P. Aburdene, *Megatrends 2000*, New York: Avon Books, 1990.
3. H. Eisner, *Reengineering Yourself and Your Company: From Engineer to Manager to Leader*, Norwood, MA: Artech House Publishers, 2000.
4. R. X. Cringely, *Accidental Empires.*, New York: Harper Business, 1992.
5. C. Shannon and W. Weaver, *The Mathematical Theory of Communication*, Urbana, IL: The University of Illinois Press, 1949.
6. D. F. Linowes, *Privacy in America*, Urbana, IL: University of Illinois Press, 1989.
7. D. Kosiur, *Understanding Electronic Commerce*, Redmond, WA: Microsoft Press, 1997.
8. P. Senge, *The Fifth Discipline*, New York: Doubleday/Currency, 1990.
9. Harvard Business Review, Discussions on Knowledge Management, Boston, MA: Harvard Business School Press, 1998.

Howard Eisner
The George Washington
University
Washington, D. C.

METROPOLITAN AREA NETWORKS

INTRODUCTION

A metropolitan area network (MAN) is a network that covers the distances of most cities, about 50 Km. Traditionally, MANs have been data networks. However, as packet voice over the Internet, IP voice, becomes more widely accepted, the distinction between voice and data networks is becoming blurred. Originally, MANs were predominantly used to interconnect the users in a single organization, such as the databases and tellers in the branches of a bank in a single city. However, the Internet has increased the demand for high-speed communications to individual users. The use of MANs has shifted from communications between users in the same metropolitan area to communications between users and the global infrastructure.

A MAN is a shared facility, and Internet traffic is bursty. Internet users require a high data rate for a period of time and then are silent for a longer period. Because of the bursty nature of the communications, the average bandwidth that a user requires is much less than the peak bandwidth that is needed to avoid waiting to download large amounts of data. A shared MAN makes it possible for a user to acquire a large bandwidth during data transmission and to relinquish that bandwidth to other users during silent intervals. When there are many users, the total bandwidth in a shared network can approach the average bandwidth of the users and provide them with their peak bandwidth most of the time when it is needed. In effect, a user can purchase slightly more than his average bandwidth on a shared network and obtain performance close to a dedicated channel at his peak bandwidth.

The utilization statistics during busy hours have long been used to design telephone networks. Before the widespread use of the Internet, and the increased duration of telephone connections, the number of connections in a local telephone switch was about one sixth the number of incoming lines. Statistics showed that the smaller number of connections is sufficient to place almost all of the incoming call requests during the busiest hours. The widespread use of the Internet has decreased the ratio of connections to incoming lines to about one third, but we still do not require a switch connection for every incoming line. Similarly, we can use the utilization statistics of data to design shared MANs with bandwidths that are much less than the peak requirements of the users.

Local area networks (LANs) take advantage of the utilization statistics of data to share the communications facilities. For instance, the Ethernet protocol gives the entire bandwidth to a single user for the duration of his transmission. The users contend for the bandwidth using a protocol called carrier sense multiple access with collision detection (CSMA/CD). In CSMA/CD, a user listens to the channel before transmitting and does not transmit if other users are transmitting (CSMA). When a user transmits, he continues to listen to the channel in case another user has started to transmit at the same time. If more than one user transmits at the same time, there is a collision and all of the users stop transmitting (CD). After detecting a busy channel or a collision a users waits a random amount of time before trying to transmit again. The random wait makes it unlikely that the same users will interfere with one another multiple times. As more users contend for a channel, users experience longer access delays before acquiring the channel. By using the busy hour utilization statistics to limit the number of users sharing a single LAN, the access delays are tolerable.

LANs are designed to cover small distances of a few kilometers, and the protocols take advantage of this characteristic. The CSMA/CD protocol works well when the time it takes to transmit data is much less than the time it takes the packet to propagate between users. However, when the distances and propagation delays increase, so that the propagation delay is greater than the time that it takes to transmit the packet, the signal that the source and receiver detect is different. The source cannot use the signal that it detects to reliably determine when there is interference from other users at the receiver.

A MAN covers greater distances than a LAN, so that the propagation delays are longer. In addition, a MAN is designed for more users than a LAN, so that the shared transmission rate must be higher and the time to transmit a message is less. The protocols that are designed for a LAN cannot be applied directly to a MAN. The first generation of MANs used protocols and network topologies that are specifically designed to span the greater distances and operate at the higher transmission rates that occur in a MAN. In the next section, we describe three of these protocols: the fiber distributed data interface (FDDI), the dual bus distributed queue protocol (DQDB), and the Manhattan Street Network(MSN).

The first generation of MAN protocols is not widely used. The main reasons are economic. In consumer applications, the cost of the device that is at the customer location is particularly important. When television was first introduced, the designers went to great lengths to reduce the cost of TV sets at the expense of the broadcast equipment. In electronics, the first few devices bear the development costs and the cost decreases rapidly as more devices are deployed. Ethernets were common in businesses before consumer MANs evolved. Multilayered MANs that use an Ethernet interface in the consumer location have a possibly insurmountable advantage over any new technology.

All three of the first-generation MANs were designed to operate over fiber-optic networks. In a MAN, the cost of new transmission infrastructure that reaches every location can be prohibitive. Fiber to the home has not been realized. Networks that use existing infrastructure, such as the cable TV network, or reduced infrastructure, such as wireless networks, have an economic advantage over any network that requires new facilities. (In this work, we use

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

CATV to refer to cable TV networks. Although CATV originally signified community access TV, it has become common to use CATV to refer to cable TV.)

In addition to the economic reasons, the DQDB network and the MSN have failed to become accepted because they are compatible with the asynchronous transfer mode (ATM) protocols rather than the Internet protocols that are used in routers. In the mid-1990s, ATM switches could support more high rate communications lines than routers. ATM is based on fixed size cells that can be exchanged between multiple inputs and outputs in a space division switch. A space division switch allows many inputs and outputs to be switched in parallel. The Internet protocols are based on variable size packets that routers passed through a single processor. The processor was a bottleneck that constrained the total input and output rate of routers. Increasing line rates in fiber-optic networks made it likely that ATM switches would replace routers. Many of the current generation of routers partition the variable size packets into fixed-size cells and use a space division switch internally. Once the bottleneck in routers was eliminated, ATM switches failed to replace routers.

In the third section we will describe the most popular MANs. Three technologies are currently used, one based on the telephone network, the second on the CATV network, and the third on wireless networks.

Telephone networks use adaptive equalizers to increase the data rates that can be transmitted on the existing telephone lines. The data rates that can be obtained depend on the quality of the lines between the central office and the subscriber premises. The lines that are being installed have fewer loading coils and alternative branches, which are referred to as "dog legs," than the lines that were installed before data became an important service, and can support higher data rates. Typically, it is possible to obtain data rates between 1.5 Mbps and 6.3 Mbps on local telephone lines. The technology is referred to as digital subscriber loop(DSL). The data rate can be used in a half duplex mode where the entire data rate is first used to transmit from the home to the central office, then used to transmit from the central office to the home, or it can be partitioned so that part of the bandwidth is available in each direction. When more bandwidth is provided in one direction than the other, DSL is referred to as asymmetric DSL (ADSL). ADSL is justified in Internet applications because users download more information from the Internet than they send to the Internet. DSL and ADSL provide dedicated lines to a central office and are not a shared MAN. They are described elsewhere in this encyclopedia.

The CATV infrastructure is evolving from a tree topology, with cables from the head end of the network to each home, to a hub-and-spoke topology, with fibers from the head end to distribution points, the hubs, and smaller trees from the hubs to each home. CATV MANs use one protocol to collect data from the homes connected to a tree that emanates from a hub, a second protocol to send data to the homes, and a third protocol or dedicated lines to transfer data between the hub and the head end of the CATV network. The hub-and-spoke topology makes it possible to use conventional Ethernet interfaces at the customer sites. In the third section, we will describe a variant of the Ethernet protocol that can be used on CATV networks and show how the protocol can be modified to also carry voice communications.

Wireless MANs use separate techniques to collect data from users in a local area and to transfer that data across the metropolitan area. Two LAN technologies are described, the IEEE 802.11 standard, also referred to as WiFi for wireless fidelity, and Bluetooth. The WiFi protocol is a variant of the Ethernet protocol called CSMA/CA, where CA stands for collision avoidance. We also describe a polled mode in this protocol that can be used for voice transmission and multihop protocols that can be used to extend the range of this network. WiFi is becoming widely used because the cost of IEEE 802.11 chips are decreasing and are included in most laptop computers. We also describe the evolving IEEE 802.16 protocol, which cover the distances in a metropolitan area and can lead to an entire wireless MAN. The wireless nature of this solution makes it possible to deploy this network with less investment in infrastructure than in wired networks.

MANs are evolving quickly. The solutions that were described in the previous version of this article have been replaced. In the conclusion we will attempt to predict future changes. These changes are fueled by the need for improved reliability, the emergence of IP voice, the increased use of wireless technologies, the overuse of the cellular bands, the eventual deployment of fiber to the home, and a resurgence of ATM-like, cell-based transmission.

THE FIRST-GENERATION MANS

FDDI

FDDI (1) is a token passing loop network that operates at 100 Mbps. It is the American National Standards Institute (ANSI) X3T9 standard and was initially proposed as the successor to an earlier generation of LANs. FDDI started as a LAN; however, it is capable of transmitting at the rates and spanning the distances required in a MAN.

Token Passing Protocol. FDDI uses a token passing protocol to give each station on the loop a chance to transmit data. The information that is transmitted on the loop is framed by a unique character. A frame may contain a token or a data message. When a station on the loop receives the token, it may remove the token and transmit data. When the station has completed its data transmission, it transmits the token so that the next station on the loop has a chance to transmit. A station that does not have the token forwards the data frames that it receives on the loop, and a station that has the token discards any data frames that it receives. The discarded data were either transmitted by one of the stations that had the token before the token site or was transmitted by the token site. If the data were inserted by another station, it must pass that station to reach the current token site. In either case, the data has circulated around the loop at least once and every station has had a chance to receive it.

In a simple token passing protocol, one station can hold the token for a very long period of time and delay other stations. FDDI uses a target token rotation time (TTRT) to control the amount of time that a station may hold the token and thereby avoids long delays. Two types of stations on an FDDI network exist, priority stations and asynchronous stations. Priority stations can transmit up to a prescribed amount of data each time that the token is received. These stations can be used to transmit real-time voice and video. Asynchronous stations can use all of the transmission capacity that is not currently being used to transmit priority data. These stations can be used to transmit large quantities of bursty data.

Each station uses the same TTRT, which is the time that we would like to have the token take to circulate around the loop. Each station tracks the token rotation time (TRT), which is the time that is left before the token exceeds the TTRT at the current station. When a token arrives earlier than expected, an asynchronous station can hold the token, and transmit data, up to its local calculation of TRT. If the token arrives late, the asynchronous station does not transmit data. In this way, an asynchronous station does not increase the delay of tokens that are late. In actuality, there is a minimum-size data packet, and an asynchronous station that receives the token just before it is due to arrive may forward it slightly later than the TTRT.

Each priority station i can send up to X_i bits, which may be different for each priority station, each time it receives the token. The time it takes to transmit the bits is the maximum token hold time for the station THT_i. The total amount of priority traffic is constrained to $\sum_i \mathrm{THT}_i + \Delta \leq \mathrm{TTRT}$, where Δ is the delay around the loop, including the propagation time and the delay inserted by each station. The constraint guarantees that the token can circulate in less than TTRT when all of the priority stations are transmitting data.

It is shown in Ref. 2 that the maximum time between token arrivals is less than 2 × TTRT and the average time between token arrivals is less than TTRT. When the token does not arrive at a station by 2 × TTRT, it is presumed to be lost, either because of a transmission error or because a token site failed, and a token recovery procedure is initiated. State machines that depict the operation of priority and asynchronous stations are shown in Fig. 1.

Isochronous Traffic. FDDI-II adds the ability to send isochronous, or circuit-switched, traffic on an FDDI loop. An isochronous channel is a regularly occurring slot that is assigned to a specific station.

FDDI-II is implemented by transmitting fixed-size frames. A central station sends out a framing signal every 125 μs. The first part of the frame is used for isochronous channels, and the second part of the frame is used to transmit the bits in the FDDI token passing protocol. An isochronous station that is assigned one byte per frame has a 64-Kbps channel. This channel is adequate for telephone quality voice.

In FDDI-II, the stations that implement the token passing protocol must switch between the two modes of operation when they receive framing signals. When a station that enters the circuit switched mode, it forwards the bits it receives. When the circuit-switched mode ends, the station resumes the token passing protocol where it left off.

Architecture. A single failure of a node or a link disconnects the stations on the loop. Poor reliability prevents loop networks from connecting the number of users associated with MANs. The reliability of an FDDI is improved with a second loop that does not carry data during normal operation, but it is available when failures occur.

FDDI networks have three components as shown in Fig. 2. The type "A" units connect user devices to the primary loop and implement the token passing protocol. The type "B" units manage the reliability. They are connected to both loops and one or more type "A" units. Type "B" units monitor the signal returning from type "A" units and bypass type "A" units that have stopped operating. They also monitor the signal on the two loops and bypass links or other type "B" units that have failed. There is one type "C" unit on an FDDI network that is responsible for signal timing and framing.

The outer loop in Fig. 2 is the primary loop and normally carries the information. The inner loop is the secondary loop and is used to bypass failed links or failed type "B" units. The signal on the secondary loop is transmitted in the opposite direction from the primary loop. Normally type "B" units forward the signal that they receive on the secondary loop. However, when a primary loop failure is detected, by a loss of received signal on that loop, a type "B" unit replaces the lost signal with the signal it receives from the secondary loop and stops transmitting on the secondary

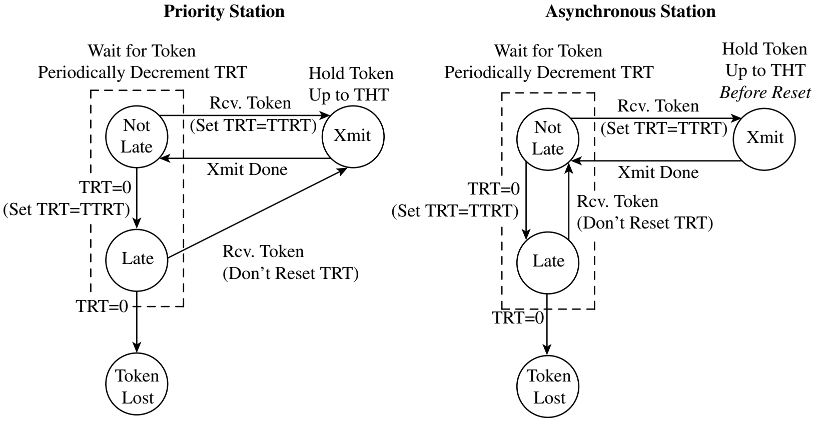


Figure 1. State machines for priority and asynchronous stations.

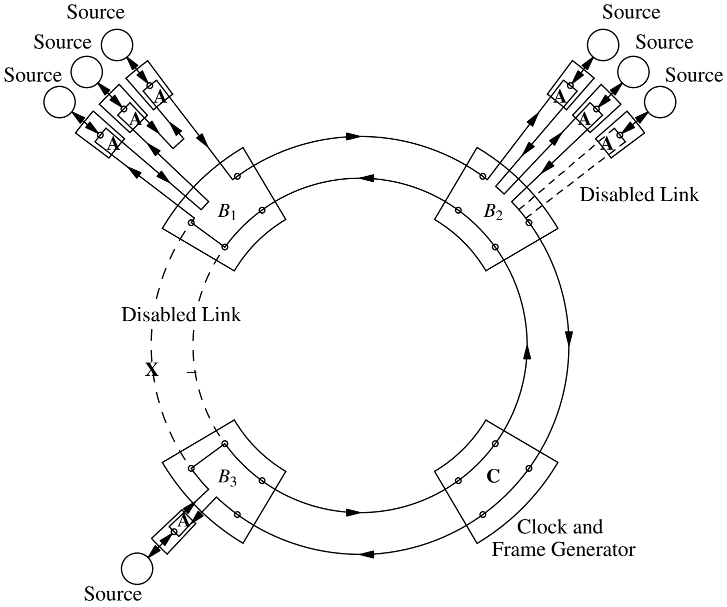


Figure 2. FDDI loop.

loop. When a type "B" unit stops receiving signal on the secondary loop, it replaces that signal with the signal it would have transmitted on the primary loop and stops transmitting on the primary loop. As an example, in Fig. 2, the "X" signifies a link failure. The unit B_1 stops receiving the signal on the primary loop, substitutes the signal that it receives from the secondary unit, and stops transmitting on the secondary loop. The unit B_3 stops receiving the signal on the secondary loop, transmits the signal it would have transmitted on the primary loop on the secondary loop, and stops transmitting on the primary loop. The entire secondary loop replaces the single failed link on the primary loop.

The Distributed Queue Dual Bus (DQDB)

DQDB (3,4) is the IEEE 802.6 standard for MANs. It uses two buses that pass each station, transmits information in fixed size slots, and uses the distributed queue protocol to provide fair access to all stations. Signals on the two buses propagate in opposite directions. A station selects the appropriate bus to communicate with a specific station and uses the other to reserve slots on that bus.

DQDB uses two passive, directional taps on each bus for each station. The first tap reads the signal, and the second tap adds signal to the bus. The taps read and write data on a bus without breaking the bus and are common components in both CATV and fiber-optic networks. The inability to remove signals makes it necessary for the the bus to have a break in the communications path where signals can leave the system.

The taps distinguish directional buses from loop networks, which use signal regenerators. In loop networks, there is a point-to-point transmission link between each station. Each station receives the signal on one link and transmits on the next link. A station can add or remove the signal on the loop. However, a failure in the electronics in a station breaks the communications path. By contrast, the stations on a directional bus networks do not interrupt the signal flow, and a failure in the electronics in a station does not break the communications path.

The directional taps are passive and do not contain active elements that can amplify the signal on the bus. Each tap removes energy from the signal path, and the signal must be restored to its full strength after passing several stations. The DQDB standard provides for erasure nodes (5,6). Erasure nodes are regenerators, similar to the station interfaces on a loop network. They restore the signal to its full strength and remove slots that have already passed their destination, so that the slots may be reused.

The Access Protocol. A baseband signal is transmitted. Energy exists in a bit position when a "1" is transmitted, and no energy is in the bit position when a "0" is transmitted. The station at the beginning of each bus, the head end, periodically transmits a sync signal to divide the transmission time into fixed-size slots.

The first bit in the slot is a "busy" bit, that is initially "0" and is changed to a "1" when the slot is being used. The read tap precedes the write tap at each station. The directional characteristic of the taps makes it possible for a station to read what upstream stations have transmitted on the bus, independent of what the station is transmitting. When a station has data to send, it transmits a "1" in the busy bit, while simultaneously reading what upstream stations have transmitted. If the busy bit was "0," the station transmits its data. If the busy bit was "1," then the slot is occupied and the station stops transmitting. There is no harm in adding energy to the "1" in the busy bit.

The stations on the bus that are closer to the head end have priority access over stations that are further away. In a dual bus network, reservations are used to construct a distributed first-in–first-out (FIFO) queue that services all stations in the order that they arrive. When slots arrive,

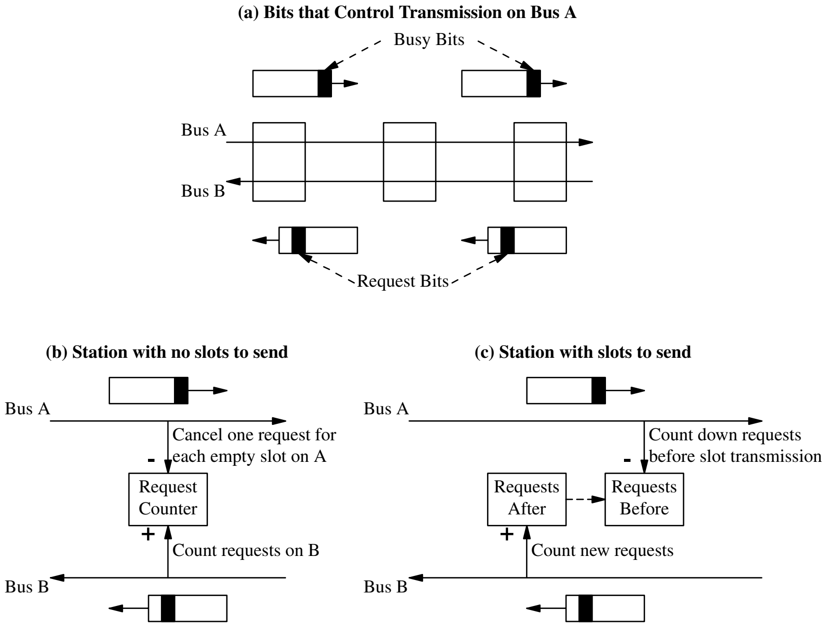

(a) Bits that Control Transmission on Bus A
Busy Bits
Bus A
Bus B
Request Bits
(b) Station with no slots to send
Bus A
Cancel one request for each empty slot on A
-
Request Counter
+
Count requests on B
Bus B
(c) Station with slots to send
Bus A
Count down requests before slot transmission
-
Requests After
Requests Before
+
Count new requests
Bus B

Figure 3. Queue formation on Bus A.

a station notifies the upstream stations by transmitting a reservation on the bus traveling in the opposite direction from the direction that it will transmit the slot. Each station maintains a queue of the requests from downstream stations and its slots, for each bus. When an empty slot arrives on a bus, the station examines the queue. If the next entry in the queue is a request from a downstream station, the station allows the empty slot to pass in order to service that request, and it removes that request from the queue. If the next entry in the queue is its own slot, it transmits that slot and removes it from the queue.

The queue at each station is a time-ordered list of its own arrivals and the arrivals at downstream stations. The queue at the head end has the complete list of arrivals, and the head end places its own messages in the slots they would have acquired if the actual messages were all in the queue. The next station on the bus does not have the list of arrivals at the head end in its local queue. However, the slots that these messages would have acquired, if they were in the queue, are busy and are not available to service the queue. By using the remaining empty slots to service its queue, the station places its own arrivals after the arrivals from downstream stations that arrived earlier than its own message. The station at the end of the bus only has its own arrivals in its queue, but all arrivals at upstream stations that arrived before the arrivals in this queue have acquired slots.

To prevent long messages from blocking short messages, the slots from each station are serviced in a round-robin order. Round-robin service can be implemented by maintaining a separate queue of slot requests for each downstream station and servicing each queue in order as empty slots arrive. An equivalent implementation in a reservation system is to have a station issue one slot request at a time, and not issue the next request until the previous request is serviced. DQDB implements a round-robin, FIFO queue with two counters that count the reservation requests that precede its own request, and those that follow it.

To preserve fixed-size slots, DQDB approximates the reservation system with a single reservation per slot. The second bit in each slot is a reservation bit and is initially set to zero. A station sets the bit to one to make a reservation. If two stations try to make a reservation in the same slot, the second station receives a one in that slot and must wait for a susequent slot to make its reservation.

Reservation requests are transmitted to the upstream stations on the opposite bus as the data. Two separate reservation systems exist, one for transmitting data on each bus. In each system, the bus that is used to transmit data is referred to as the data bus, and the other bus is the reservation bus. In Fig. 3, we depict the queue formation on bus A. Figure 3(a) shows the bits transmitted on each bus, (b) shows the operation of the counter in a station that is not transmitting slots, and (c) shows the operation of the counter in a station with a slot to send. In Fig. 3C, the count down counter contains the number of requests that preceded the slot from the local station. When this counter reaches zero, the next empty slot is used to transmit a slot and then the request counter, which is the number of requests that arrived after the slot from the local station, is transferred to the countdown counter and precedes the next slot from this station.

In a DQDB network with multiple priority levels for data, there is one reservation bit and two counters for each priority level. When empty slots are received, the counters for the higher priority levels are emptied first.

The DQDB protocol does not provide guarantees on delay or bit rate. An isochronous mode, similar to FDDI, has been added to support real-time traffic. The slots leaving the head end are grouped into 125 μs frames. In some slots the busy bit is zero and the slots are available for the DQDB protocol. In other slots, the busy bit is one. These slots are reserved for real-time traffic. A station that reserves a single byte in a frame acquires a 64-Kbps channel with at most a 125-μs delay. This same guarantee is provided by the telephone system for digital voice.

Protocol Unfairness. The description of the distributed queue ignores the propagation delay on the buses. The distance-bandwidth product of IEEE 802.6 standard networks creates a potential for gross unfairness (7). The standard was modified to include bandwidth balancing (BWB) (8), which eliminated most of the unfairness.

The IEEE 802.6 standard is designed to operate at 155 Mbps, with 53-byte slots, and is compatible with ATM. At these rates, a cell is only about 0.4 miles long. The standard spans up to 30 miles. Therefore, there may be 75 cells simultaneously on the bus. Assume that a station near the head end of the bus has a long file transfer in progress when a station 50 cells away requests a slot. In the time it takes the request to propagate to the upstream station, that station transmits 50 slots. When the request arrives, the upstream station lets an empty cell pass and then resumes transmission. An additional 50 slots are transmitted before the empty cell arrives at the downstream station. When the empty slot arrives, the downstream station transmits one slot and submits a request for another. The round trip for this request to get to the upstream station and return an empty slot is another 100 slots. As a result, the upstream station obtains 100 times the throughput of the downstream station.

A similar imbalance can occur in favor of the downstream station when that station starts transmitting first. Although the downstream station is the only source, it transmits in every cell, while placing a reservation in every slot. When the upstream station begins transmitting, there are no reservations in its counter, but there are 50 reservations on the bus. Although the upstream source transmits a slot, a reservation is received. Therefore, the upstream station must allow one slot to pass before transmitting its second slot. During the time it takes to service the reservation and the upstream station's next transmitted slot, two reservations arrive. Therefore, the upstream station lets two empty slots pass before transmitting its third slot. The reservation queue at the upstream station continues to build up each time it transmits a slot, and the upstream station takes fewer of the available slots. An imbalance between the upstream and the downstream station is sustained indefinitely because the downstream station places a reservation on the bus for each of the empty slots that the upstream station releases. The imbalance is not as pronounced as when the upstream station starts first, but it is considerable. The exact imbalance depends on the distance between two stations and the time that they start transmitting relative to one another (8).

The BWB mechanism is based on two observations:

1. Each station can calculate the fraction of the slots that are used, whether or not the data pass the station.
2. It is possible to exchange information between stations by using the fraction of the slots that are not used.

Table 1. Convergence of Rates when two Stations use 90% of the Slots Available to them

Station A		Station B	
Measure Bsy+Rqst	Take	Measure Bsy+Rqst	Take
0	0.9×1 = 0.9	–	–
0	0.9×1 = 0.9	0.9	.9×.1 = .09
0.09	.9×.91 = .82	0.82	.9×.18 = .16
0.16	.9×.84 = .76	0.76	.9×.24 = .22
0.22	.9×.78 = .7	0.7	.9×.3 = .27
0.27	.9×.73 = .66	0.66	.9×.34 = .31
0.31	.9×.69 = .62	0.62	.9×.38 = .34
. . .	. . .	. . .	. . .
0.474	.9×.526 = .474	0.474	.9×.526 = .474

A station sees a busy bit for every slot transmitted by an upstream station and a reservation for every slot transmitted by a downstream station. By summing the fraction of the busy bits and reservation bits and adding the fraction of the slots that the station transmits, the station calculates the total fraction of the slots that transmit data on the bus.

Table 1 shows how stations can communicate by using the fraction of unused slots. Each stations tries to acquire 90% of the unused bandwidth on a channel. Station A starts first and uses 90% of the total slots. When station B arrives, only 10% of the slots are available. Station B does not know whether the slots are being used by a single station taking its allowed maximum share or many stations. Station B uses 90% of the available slots or 9% of the slots in the system. Station A now has 91% of the slots available. When station A adjusts its rate to 90% of 91% of the slots, it uses 82% of the slots, making 18% of the slots available to station B. Station B adjusts its rate up to 90% of 18%, which causes the station A to adjust its rate down, and so on until both stations arrive at a rate of 47.4%. Note that this mode of communications cannot be used when stations try to acquire 100% of the slots.

The implementation of BWB in the standard is particularly simple. A station acquires a fraction of the slots available by counting the slots it transmits and by placing an extra reservation in the local reservation queue when the count reaches a prescribed value. In this way, a station lets a fraction of the slots that are available remain empty. For instance, if a station wants to take 90% of the slots that are available, it counts the slots that it transmits and inserts an extra reservation in the reservation counter after every ninth slot that it transmits. As a result, every tenth slot that the station could have taken remains empty.

With BWB, the fraction of the throughput that station i acquires, T_i is a fraction α_i, of the throughput left behind by the other stations:

$$T_i = \alpha_i \left\{ 1 - \sum_{j \# i} T_j \right\}$$

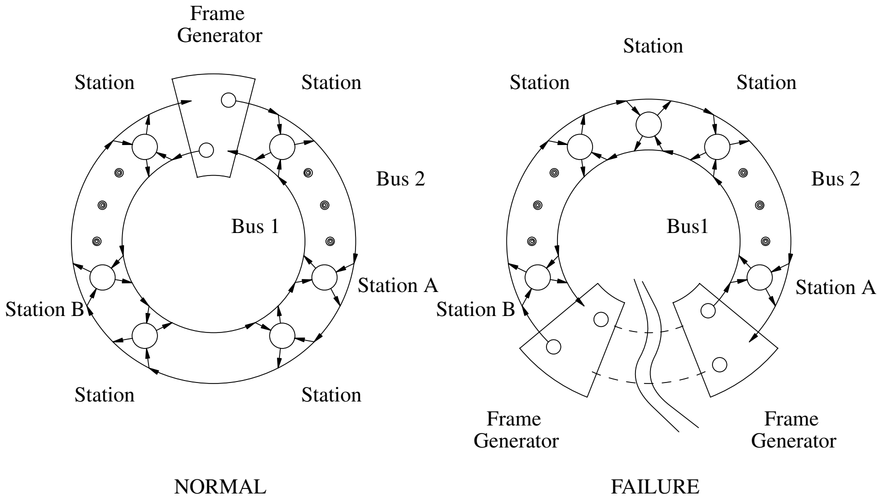


Figure 4. DQDB network before and after a failure.

When N stations contend for the channel, and use the same value of $\alpha = \alpha_\iota$ they each acquire a throughput

$$T = \frac{\alpha}{1 + \alpha(N - 1)}$$

The total throughput of the system increases as α approaches one or the number of users sharing the facility becomes large. The disadvantage with letting α approach one is that it takes the network longer to stabilize. We can see from the example in Table 1 that the network converges exponentially toward the stable state. However, as $\alpha \to 1$, the time for convergence goes to infinity. The original DQDB protocol uses $\alpha = 1$.

Reliability. The dual bus in a DQDB network is configured as a bidirectional loop, as shown in Fig. 4. The signal on the outer bus propagates clockwise around the loop, and the signal on the inner bus propagates counterclockwise. The signal does not circulate around the entire loop, but it starts at a head end on each bus and is dropped off the loop before reaching the head end.

To communicate, a station must know the location of the destination and the head ends and transmit on the proper bus. For instance, station A transmits on the outer bus to communicate with station B, and station B transmits on the inner bus to communicate with station A.

The dual bus is configured as a loop so that the head end can be repositioned to form a contiguous bus after a failure occurs. The head end for each bus is moved so that the signal is inserted immediately after the failure and drops off at the failure. This system continues to operate after any single failure.

The ability to heal failures increases the complexity of stations on the DQDB network. To heal failures, the station that assumes the responsibility of the head end must be able to generate clock and framing signals. In addition, after a failure each station must determine the new direction of every other station. For instance, after the failure in Fig. 4 is repaired, station A must use the inner bus, rather than the outer bus, to transmit to station B.

Manhattan Street Network (MSN)

The MSN (9) is a two-connected network of 2 × 2 switches. A station is attached to each switching node. The connectivity between nodes in the MSN is the same as in an FDDI network and a DQDB network, except that the logical topology of the network resembles the grid of one-way streets and avenues in Manhattan, as shown in Fig. 5. Fixed-size cells are switched between the two inputs and outputs using a strategy called deflection routing. The fixed-size cells can encapsulate ATM cells, so that the MSN is compatible with wide-area ATM networks.

In the MSN, packets are routed independently at each node that they traverse so that the overhead associated with establishing and maintaining circuits is eliminated.

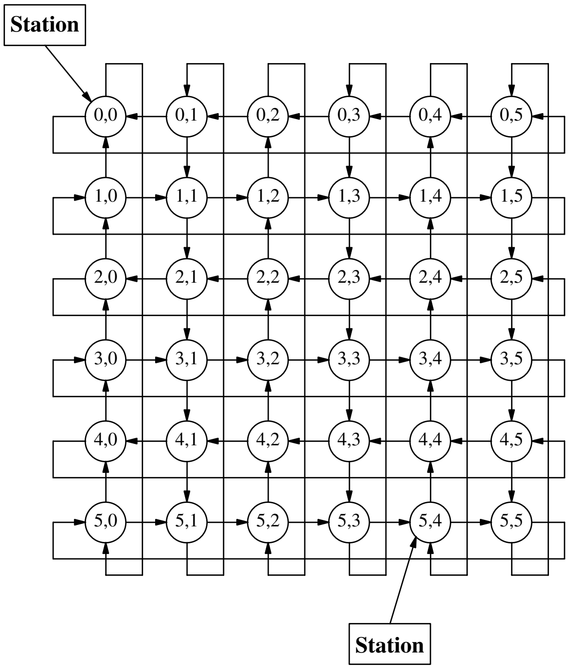


Figure 5. The Manhattan Street Network.

The direction of one-way streets and avenues alternate. By numbering the streets and avenues properly it is possible to get to any destination without having a complete map, and when failures occur, detours around the failure can be determined. The grid is logically constructed on the surface of a torus instead of a flat plane. The wraparound links on the torus decrease the distance between the nodes and eliminate congestion in the corners.

In deflection routing, packets can be forced to take an available path rather than waiting for a specific path. It operates on any network where the nodes have the same number of inputs and outputs and the network transmits fixed-size cells. The cells are aligned at a switching point in a node. In a two-connected network, if both cells select the same output, and the output buffer is full, one cell is selected at random and forced to take the other link. The cell that takes the alternate path is deflected.

Deflection routing gives priority to cells passing through the node. Cells are only accepted from the local source when empty cells are arriving at the switch. Therefore, the number of cells arriving at the switch never exceeds the number of cells that can be transmitted, and cells are never dropped because of insufficient buffering. The link capacities is shared between bursty sources without large buffers and without losing packets because of buffer overflows. The operation of a deflection routing node is shown in Fig. 6. Deflection routing is also used for routing inside some ATM switches (10,11).

The MSN is well suited for deflection routing for three reasons:

1. At any node many destinations are equidistant on both output links. Cells headed for these destinations have no preference for an output link and do not force other cells to be deflected.
2. When a cell is deflected, only four links are added to the path length. The worse that happens is that the cell must travel around the block.
3. Deflection routing can guarantee that cells are never lost, but it cannot guarantee that they will not be deflected indefinitely and never reach their destination. It has been found that this type of livelock does not occur in the MSN when the cell that is deflected is selected randomly (12).

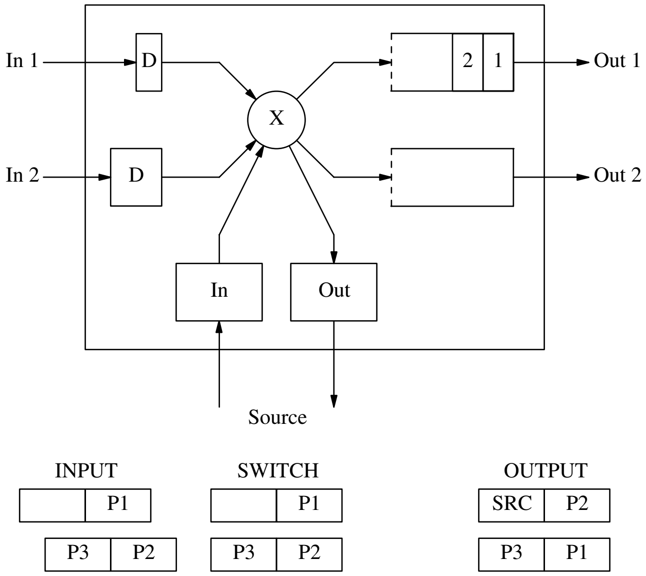


Figure 6. Deflection routing node.

Deflection routing is similar to the earlier hot potato routing (13), which operated with variable-size packets, on a general topology, with no buffers. Fixed-size cells, the MSN topology, and two or three cells of buffering converted the earlier routing strategy, which had very low throughputs, to a strategy that can operate at levels exceeding 90% of the throughput that is achieved with infinite buffering.

Reliability. The MSN topology has several paths between each source and destination. The alternate paths can be used to communicate after nodes or links have failed.

There are two simple mechanisms to survive failures in the MSN, as shown in Fig. 7. Node failures are bypassed by two normally closed relays that connect the rows and columns through. The missing node in the grid in Fig. 7 has failed. Link failures are detected by a loss of signal, as in loop networks. Nodes respond to the loss of signal by not transmitting on the link at right angles to the link that has stopped. When one link fails, three other links are removed from service and the node at the input to the

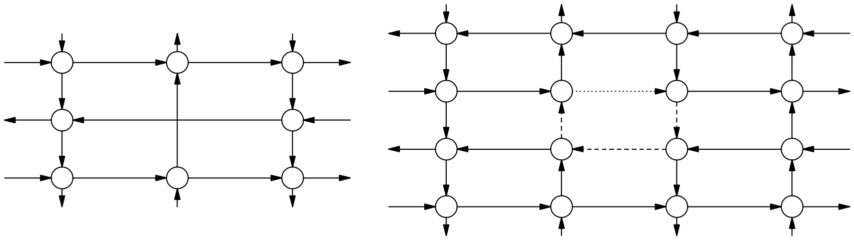


Figure 7. Failure recovery mechanisms in the MSN.

failed link stops transmitting on it. The dotted link in Fig. 7 has failed, and nodes stop transmitting on the dashed links. This link removal procedure works with any number of link failures. The number of inputs equals the number of outputs, so that deflection routing continues to operate without losing cells. In addition, it has been found that the simple routing rules that are designed for complete MSNs continue to work on networks with failures.

Comparison of FDDI, DQDB, and the MSN

DQDB and FDDI are linear topologies. The average number of links that data traverses increases linearly with the number of nodes in the network, and the average throughput that each user can obtain decreases linearly with the number of users. By contrast, in the MSN, the distance between nodes increases as the square root of the number of nodes in the network. As a result, the reduction in the throughput per user, which occurs as networks become large, is much less in the MSN than in the FDDI or DQDB network.

In the DQDB network, the penalty for large networks can be reduced by breaking the network into segments and erasing data that have already been received when it reaches the end of a segment. This strategy works particularly well when communities of users communicate frequently. When those users are placed on the same segment of the bus, the traffic between them does not propagate outside the segment and interfere with users in other segments. When a community in the middle of the bus becomes congested in a DQDB network, communications between nodes at opposite edges of the bus must still pass through that community.

In the MSN, communities of users are supported in a very natural way. If nodes that communicate frequently are located within a few blocks, they only traverse the paths in those few blocks and do not affect the rest of the network. Special erasure nodes are not needed because the protocol removes cells that reach their destination. In addition, when there is heavy traffic within a neighborhood, communications between other neighborhoods can continue without passing through that neighborhood. With deflection routing, cells naturally avoid passing through congested neighborhoods.

Both DQDB and FDDI can survive single failures. However, when multiple failures occur, the network is partitioned into islands of nodes that cannot communicate with one another. Nodes in the MSN are not cut off from one another until at least four failures occur. When four failures have occurred, the likelihood of nodes being disconnected, and the number that are actually disconnected, is small. A quantitative comparison of the reliability of MSN, DQDB, and FDDI networks is presented in Ref. 14.

An advantage of linear topologies is that routing is relatively simple. All data that enter an FDDI system is transmitted on a single path, and there is only one path to select at any intermediate node. In a DQDB network, the source must decide which of the two paths leads to the destination, but once the data are in the network, there are no choices to make. The MSN network has a simple rule to select a path, but a choice must be made at each node.

An important consideration in any large network is how easily it can be modified to add or delete users. In early LANs, there was a correspondence between the topology of the network and the physical distribution of users. A loop network was a daisy chain between adjacent offices and a bus network passed down a hallway. It is more difficult to change the wiring between offices than to have all offices connected to a wiring cabinet and change the interconnections in that cabinet. As a result, most linear networks are physically a star network between users and a wiring cabinet. The users are connected inside to a wiring cabinet to form a logical loop or bus or mesh network. The number of wires that must be changed in the wiring cabinet determines how difficult it is to add or delete users from a network.

In a bidirectional loop or bus network, adding or deleting a user is a relatively simple operation. To add a user, the connection between two users is broken and the new user is inserted between them. In the wiring cabinet, two wires are deleted and four are added. In a complete MSN, two complete rows or columns must be added to retain the grid structure. There are, however, partial MSNs in which rows or columns do not span the entire grid, and a technique is known for adding one node at a time to a partial MSN to eventually construct a complete MSN (15). With this technique, the number of links that must be changed in the wiring cabinet is the same as in the loop or bus network.

DQDB and FDDI have an isochronous mode of operation that provides dedicated circuits to support realtime traffic. The isochronous mode is well integrated into the DQDB protocol. Nodes that only require the data mode do not have to change any protocols or hardware when isochronous traffic is added to the network. The only change that these nodes notice is that more slots are busy. By contrast, when isochronous traffic is added to an FDDI network, every node must be able to perform context switching to move between the data and the circuit modes. The MSN does not have an isochronous mode of operation. Real-time traffic operates like IP voice on the Internet and is dependent on low network utilizations.

THE CURRENT GENERATION MANs

CATV

The CATV network is an existing MAN that is designed to deliver TV programs to a large number of homes. The network is designed for unidirectional delivery of the same signal to a large number of receivers. In most CATV networks, many channels carry signals from the head end to the home in the downstream direction, and a smaller number of channels carry signals in the opposite direction, the upstream direction. The network taps are also directional and receive signals from downstream but not from upstream. Signals transmitted through the directional taps travel upstream. The lines in many homes are not properly terminated and insert significant noise into the network, but this noise travels upstream, it is not received by the other homes, and does not degrade TV reception from the head end. The upstream channel is a noisy channel that cannot be used to carry high-quality analog signals, but it can be used to carry digital data.

The CATV network is increasingly being used to provide high bandwidth data access to the Internet. Several competing standards committees, including the IEEE 802.14 standards committee and the Multimedia Cable Network Systems group (MCNS), are working on standards that are not compatible with one another.

The current practice and the evolving standards have some common characteristics. All channels use 6-Mhz bands that are used for TV transmission. Many homes share the upstream channel to send data to the Internet. The protocols that share this channel include reservation protocols, which allow homes to acquire scheduled slots, and contention-based protocols, which are similar to the CSMA/CD protocol used in Ethernet but may have more complicated contention resolution schemes based on tree searches. The upstream channel is relatively noisy, and cable modems transmit between 1.6 and 10 Mbps in these channels. The downstream channel carries addressed packets from the Internet to the many homes.

The data from the Internet come from a single source, a router, so that there is no contention for this channel. Typically, in Internet applications there is much more data to the home than from the home. The downstream channel has a much higher signal-to-noise ratio than the upstream channel, and cable modems transmit up to 40 Mbps in the downstream channels.

Instead of describing the many contending standards, we will demonstrate the use of the CATV network with the IEEE 802.3 Ethernet standard protocol. This approach has the advantage that the home terminals use standard Ethernet chips, which have become inexpensive because they are widely used, to share the upstream channel. Furthermore, Ethernets typically transmit encapsulated IP packets that can be forwarded directly to the Internet. The CSMA/CD protocol that is used on Ethernets cannot be applied directly to the upstream CATV channel because

1. The stations that are transmitting on the upstream channel cannot listen to the other stations to determine when the channel is busy or when a collision occurs.
2. The distances spanned by a CATV network are much greater than the distances spanned by a local network so that the CSMA/CD protocol become less efficient.

CATV networks have evolved from a tree topology to a hub-and-spoke topology. In the tree topology, cables from the head end of the network are connected to each home. In the hub-and-spoke topology, fibers from the head end carry signals to each hub and smaller cable trees connect the hub to the individual homes. The hub-and-spoke topology has fewer amplifiers and delivers TV signals with a higher signal-to-noise ratio. The hub-and-spoke topology also reduces the distance between users connected to the same hub. The decreased distance between users and the lower transmission rates on the upstream channel make it possible to use CSMA/CD to share the upstream channel between users that are connected to the same hub. When necessary, a transmission plan, called homenets (16), can be used to partition the trees emanating from a hub into several smaller Ethernets.

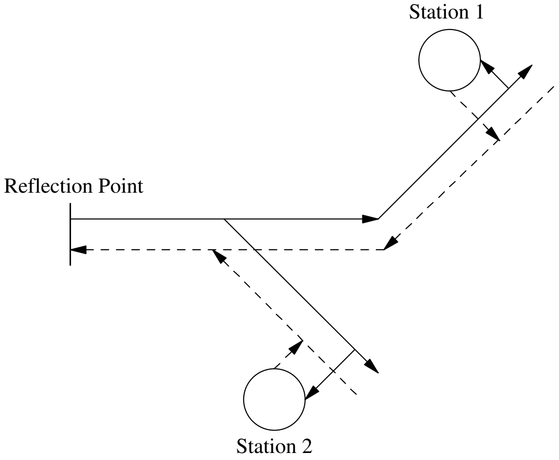


Figure 8. Transmission strategy for CSMA/CD access.

Each hub limits the number of data users who share a single Ethernet. Increasing the number of users on an Ethernet reduces the bandwidth that is available to each user. Proper placement of the hubs can limit the contention and provide a desired service level for data. The same principles can be used to engineer the sharing of the CATV data network as have been used to engineer the sharing of local office switches in the telephone network.

Each user must be able to listen to the signals transmitted by all other users to perform CSMA/CD. At each hub, the signal from the upstream channel is translated to a frequency that is used by a downstream channel and retransmitted on the tree originating at the hub, as shown in Fig. 8. The home terminals listen to the second channel to determine when the channel is busy or when there are collisions.

Movable Slot TDM. As IP voice gains wider acceptance, the data channels on CATV networks will be used to carry voice. The reservation systems that are being considered by the standards committees can provide better delay and bandwidth guarantees than the standard Ethernet protocol. A variation of the CSMA/CD protocol called Movable Slot TDM (MSTDM) (16,17) makes it possible to obtain high-quality voice on the upstream channel without modifying the current Ethernet chips. MSTDM makes it possible to place voice on the upstream channel without modifying the operation of data-only users

MSTDM gives voice packets priority over data packets. The data packets follow the standard IEEE 802.3 protocol. The voice packets listen before transmitting but do not perform collision detection. When a voice and data packet collide, the data packet stops transmitting but the voice packet continues to transmit. There is a preempt interval at the beginning of each voice packet that does not contain bits that are needed to receive the voice packet and is long enough to guarantee that the data packet has stopped

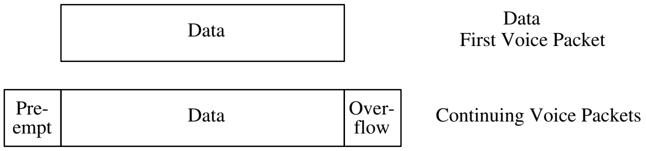


Figure 9. Format of MSTDM packets.

transmitting before the useful data in the voice packet is transmitted. If the channel is busy when a voice packet tries to transmit, it waits until the channel becomes idle and retransmits immediately.

The first packet in a voice connection uses the same protocol as the data packets. Subsequent voice packets in a connection are transmitted a fixed period T_v after the last successful transmission, whether or not the previous packet is delayed. If the previous packet is delayed, it places any voice samples that arrive while it is being delayed into an overflow area in the packet, so that the same number of voice samples are waiting at each scheduled transmission time. The only constraint on the data packet is that its length is less than or equal to the length of a fixed-size voice packet. The packet formats are shown in Fig. 9.

Scheduled voice sources never collide. All scheduled voice packets have the same length, require time X_V to transmit, and are scheduled at least X_V apart. A scheduled voice source preempts a data source that collides with it. A scheduled voice source is delayed by less than X_V by a data source that is currently transmitting, because the data packet length is less than the voice packet length. If a scheduled voice source is delayed, it cannot be further delayed by a data source because it transmits as soon as the channel becomes idle and preempts any data source that starts transmitting at the same time. When a scheduled voice packet is delayed less than X_V, it delays the next voice source by an amount less than or equal to its own delay. Therefore, the delay of successive voice sources is nonincreasing and is less than X_V. And voice sources never collide. The access rule for scheduled voice sources is CSMA, rather than CSMA/CD. CSMA can be implemented with a single NAND gate that turns off collision detection. The NAND gate is external to the commercially available chips that implement the Ethernet protocol.

When a scheduled voice source is delayed, the overflow area is occupied by the samples that arrive during the delay. The overflow area need only be large enough to accommodate the voice samples that arrive in X_V. The delayed packet adopts a new schedule, and the overflow area is empty for successive packets that are not delayed. The upper bound on the delay for a voice sample is $T_V + X_V$. In Fig. 10 we show the operation of the protocol when scheduled packets are delayed by a data source or a new voice source.

When the system bandwidth is completely used by voice sources, MSTDM becomes a simple TDM system and new voice sources and data sources cannot disrupt the operation of the system. Figure 11 depicts the operation of a system that can support 3.5 voice sources. The small number of sources is only used to demonstrate the operation of the protocol. A 10-Mbps system, with 32-Kbps voice sources, can support several hundred active voice sources. In Fig. 11 there is enough bandwidth for half of a new voice source. Source 4 joins the system and delays source 1, 2, and 3. The delayed source 3 delays the newly scheduled source 4, which once again delays sources 1, 2, and 3. The delay is nonincreasing, so the scheduled sources never collide, but the sources are always delayed, so that the overflow area always has samples. Any new voice or data sources find a busy channel or collide with a scheduled source and are preempted.

Wireless Networks

Wireless networks are the most rapidly evolving metropolitan area networks. Wireless networks require less of an investment in infrastructure than wired networks,

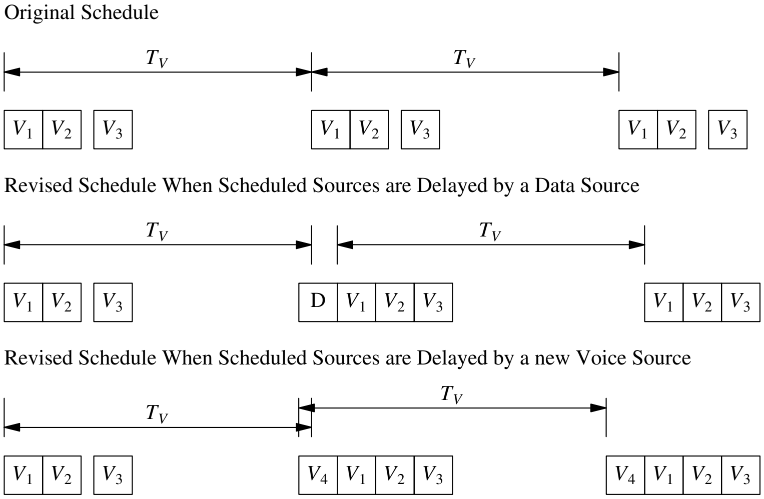


Figure 10. Scheduled voice sources that are delayed by a data source or a new voice source.

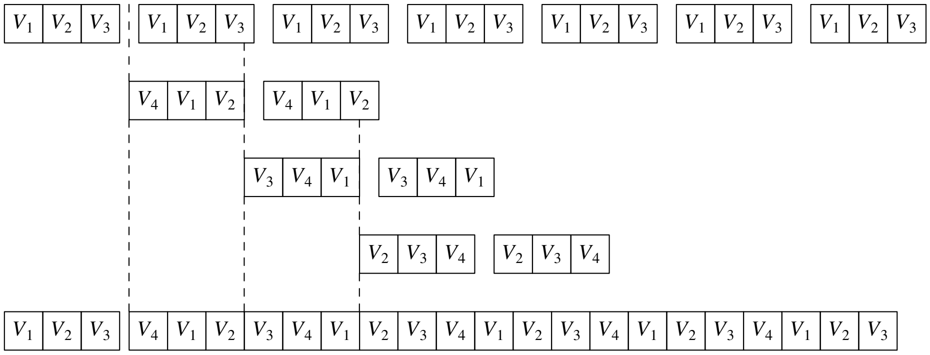

V_1 V_2 V_3 V_1 V_2 V_3 V_1 V_2 V_3 V_1 V_2 V_3 V_1 V_2 V_3 V_1 V_2 V_3 V_1 V_2 V_3
V_4 V_1 V_2 V_4 V_1 V_2
V_3 V_4 V_1 V_3 V_4 V_1
V_2 V_3 V_4 V_2 V_3 V_4
V_1 V_2 V_3 V_4 V_1 V_2 V_3 V_4 V_1 V_2 V_3 V_4 V_1 V_2 V_3 V_4 V_1 V_2 V_3 V_4 V_1 V_2 V_3

Figure 11. Fully utilized MSTDM network.

particularly in metropolitan areas where installing new cables may involve digging up a street. Deploying wireless networks makes it possible to try new services without investing in new cables. In addition IEEE 802.11 wireless interfaces have come down in cost and are included in most new laptop computers. Wireless interfaces in battery- operated computers make it possible for users to work where they they are, rather than searching for power or information outlets. And the wireless interface is a single standard, unlike the many incompatible cable interfaces.

Wireless metropolitan area networks are composed of access networks that interface to the users, and cover relatively small distances, and backbone networks that cover the metropolitan area distances and carry the user traffic to the wide area network. In this article we discuss two access networks, the IEEE 802.11 standard network and Bluetooth. IEEE 802.11 networks operate at higher bit rates and cover longer distances than Bluetooth networks. Initially, Bluetooth interfaces were to cost much less than IEEE 802.11 interfaces. However, the larger number of IEEE 802.11 units that have been deployed has made them less expensive than Bluetooth interfaces. It is likely that IEEE 802.11 networks will dominate and that there will be very few Bluetooth networks. Bluetooth is included in this article because it has had a great impact on the evolving IEEE 802.16 standard. The IEEE 802.16 networks are wireless networks that can cover the distances spanned by metropolitan areas and are possible backbone networks for the wireless access networks.

IEEE 802.11 Networks. IEEE 802.11 networks have two modes of operation, a point coordination function, which is polled, and a distributed coordination function, which uses a protocol called carrier sense multiple access with collision avoidance, CSMA/CA. The polled operation assumes a master station that assigns slots to the active transmitters. The master station also sends unassigned slots in which new sources can transmit when they want to be added to the polling list. When multiple sources transmit during an unassigned slot, there is a collision and the sources must retry. The CSMA/CA protocol is similar to the Ethernet protocol and does not require a master station. Currently, the CSMA/CA protocol is better defined and is more widely used than the polled protocol. However, the polled protocol uses the scarce radio bandwidth more efficiently and can provide the guarantees that are needed for voice communications. When WiFi networks are used to access a base station that connects it with a backbone network, the base station is the logical master station

The CSMA/CD protocol that is used in wired Ethernets cannot be applied directly to wireless networks. It is possible to listen to the channel before transmitting (CSMA) to determine whether another source is transmitting, but it is not possible to listen to the channel while transmitting (CD) to determine whether another source is also transmitting. In addition, hidden nodes interfere with the source at the destination, but they cannot be detected by and cannot detect the source, as depicted in Fig. 12. The area A_{SX} is the region in which the signal from the source can be detected. This area includes the destination. The area A_H is the hidden nodes. A node in A_H cannot detect the signal from the source, but if it starts to transmit, its signal will interfere with the signal from the source at the destination. The area A_{HX} is the region in which nodes detect the signal from a hidden node and includes the destination.

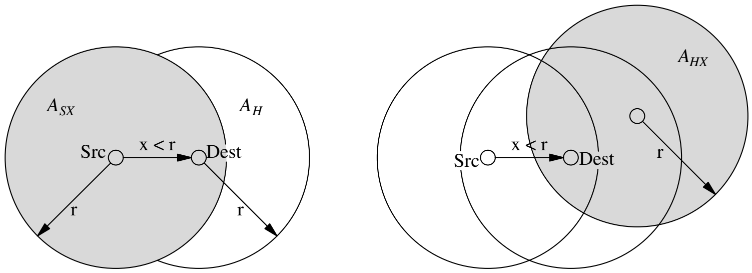


Figure 12. Hidden nodes.

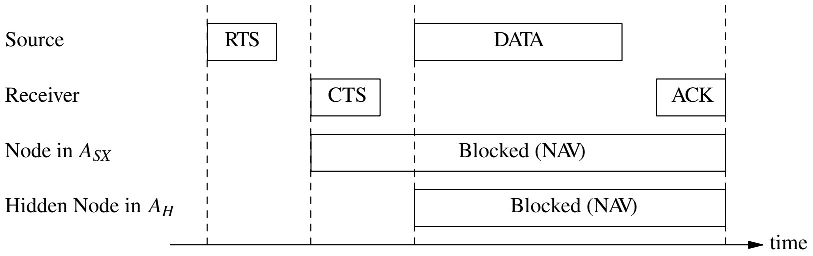


Figure 13. Three-way handshake, CSMA/CA—with ACK.

The CSMA/CA protocol uses a three-way handshake to avoid collisions with hidden nodes, as depicted in Fig. 13. The source senses the channel, and if there is no transmission in progress, it sends a request to send (RTS) to the destination. If the destination receives the RTS, one node that is hidden from the source is not transmitting, the destination sends a clear to send (CTS). When the source receives a CTS, it sends the data packet. If the receiver correctly receives the data packet, it sends an acknowledgment (ACK). The error rate in wireless networks is typically higher than that in wired networks, and the ACK is an integral part of the wireless protocol. In addition, the packet size in wireless networks is typically smaller than in wired networks to increase the probability of correct reception. The nodes in A_{SX} receive the RTS from the source, but it may not receive the CTS from the receiver. The node in A_{SX} sets a network allocation vector (NAV) that stops it from transmitting for a period of time that is long enough for the receiver to transmit an ACK. A node A_H receives CTS but not RTS or the data. The node in A_H sets a shorter NAV that is long enough for the source to send the data and the receiver to send an ACK.

The original IEEE 802.11 networks operated at 1 Mbps. The more recent versions have variable rates and can operate up to 54 Mbps. A source tries to transmit at the highest rate. If the signal is not correctly received, the source reduces its rate, to increase the signal-to-noise ratio, until the signal is correctly received. The achievable rate is related to the distance between the source and the destination. The closer the destination, the higher the signal-to-noise ratio, and the higher the rate. Table 2 is the approximate relationship between the distance and the achievable rate for the current versions of the IEEE 802.11 standard.

Table 2. Approximate Relationship between Distance and Transmission rate in IEEE 802.11 Networks

Rate (Mbps)	802.11a	802.11b	802.11g
1	250	200	250
2	175	175	175
6	175	100	175
9	140	70	140
12	140	40	140
18	80		80
24	70		70
36	40		40
48	20		30
54	10		20

The distance spanned by an IEEE 802.11 network can be increased by using multihop techniques. If the source cannot reach the access point to the MAN directly, it transmits the data to an intermediate node, which forwards the data toward the destination. The intermediate nodes operate as routers and select the next node on the path to the destination. The path to the destination is not fixed and may change as nodes enter and leave the network. The techniques to find paths are covered in the literature on ad hoc, multihop, radio networks. This currently is an active research area and is beyond the scope of the current article.

Bluetooth. Bluetooth is a polled network. It is organized as piconets with one master and up to seven slave nodes. All communications are between the master and a slave node, with half of the slots assigned to the master node. The total bit rate in a piconet is 1 Mbps.

The number of nodes in a network is increased by connecting piconets into a scatternet. A slave node that is in both piconets operates as a gateway between the piconets, as depicted in Fig. 14. Multihop communications is implemented on paths that traverse source, to master, to bridge, to master, . . ., to master, to destination.

IEEE 802.16. The IEEE 802.16 MAN standard is similar to Bluetooth networks. The network is polled, but the distance is longer and the bit rates are higher. The network can span 30 Km, and the slaves can operate at a bit rate that is 50 Mbps, 100 Mbps, or 150 Mbps, depending on the signal-to-noise ratio.

The stations that are polled are dropped from the polling list if they remain inactive for seven consecutive polls. In addition, the master and slave stations can obtain different slot rates to reflect asymmetries in the traffic. IEEE 802.16 networks can be interconnected as scatternets.

CONCLUSION

Metropolitan area networks have evolved rapidly in the last decade because of changes in technology and applications. The cell switching technology of ATM networks has lost to the routing technology of IP networks, and the MANs that were based on cell switching have disappeared. Consumer applications that connect individuals to the Internet have become much more important in MANs than interconnecting users in a corporate network that spans a metropolitan area. As a result, the price of network access has become a much more important consideration and older technologies that are further along the learning

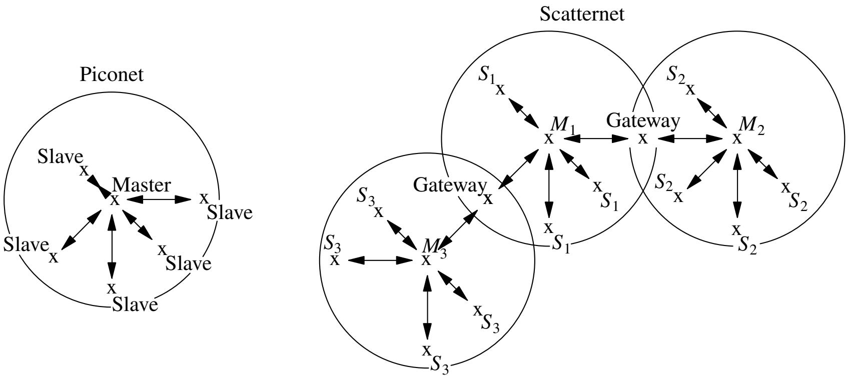


Figure 14. Bluetooth networks.

curve have dominated. Users have come to expect tetherless access to networks, and wireless end-user devices have become the norm for all of our communications.

In the next decade, we expect the rate of change of MANs to increase. Initially, the use of wireless backbone networks will increase, as we explore new and different communications services. Wireless technologies provide the fastest, most economical way to construct new networks. However, as some services succeed, there will not be enough wireless bandwidth in the backbone to support the new demand. The wireless backbone will be replaced by more efficient, higher rate, fiber-optic networks. Wireless will not go away. The end user is hooked on tetherless access. However, the part of the network that is invisible to the user will apply the most cost-effective technologies.

The Internet has been the defining application for the current generation of MANs. IP voice is likely to be the defining application of the next generation of MAN. IP voice is more efficient than circuit-switched voice because the channel is not used during silent intervals. However, the success of IP voice will more likely depend on new services, such as the walkie-talkie functions that are being built into cell phones. If this happens, traffic within the MAN will increase with respect to traffic to the wide-area networks. Networks that are being designed to reflect the traffic imbalance between end users and servers in the Internet will once again be replaced by networks with balanced loads in both directions.

The success of IP voice and the reduced cost of IEEE 802.11 chips is likely to completely change the current cellular voice networks. The success of cellular voice has created a bandwidth crisis in many metropolitan areas. The smaller distances spanned by IEEE 802.11 networks makes it possible to reuse the bandwidth more often, and multihop techniques provide a means to redistribute the traffic in congested areas. In addition, IEEE 802.11 base stations are less visible than the current microwave towers.

As we become more dependent on MANs, reliability will become a more important issue. Currently it is almost impossible to buy multipath reliability in a metropolitan area. Even if we buy two lines from different service providers, both lines may belong to a single provider or traverse the same conduits. Future MANs are likely to have a different protocol architecture than our current layered structure. The new architecture will make more physical attributes visible, rather than hiding them. In addition, more reliable mesh structures, such as the MSN, are likely to replace the current tree, hub-and-spoke, and ring architectures.

Finally, we expect a resurgence of cell transmission in the heart of the network. Routers switch cells internally. Eventually there will be a standard that allows routers to exchange the cells, rather than having multiple conversions between cells and IP packets. The end-user applications are not affected by the internal operation of the network. Cell transmission will also allow the routers to provide quality of service. It is unlikely that the heavyweight ATM standards, or the virtual circuits that are part of ATM, will return, but these standards are not needed to implement cell transmission. The return of cell transmission may make us reconsider the first generation of MANs that also used cell transmission.

BIBLIOGRAPHY

1. F. E. Ross, An overview of FDDI: The fiber distributed data interface, *IEEE J. Select Areas Commun.*, **7** (7): 1043–1051, 1989.
2. M. J. Johnson, Proof that timing requirements of the FDDI token ring protocol are satisfied, *IEEE Trans, Commun.*, **COM–35** (6): 620–625, 1987.
3. R. M. Newman, Z. L. Budrikis, and J. L. Hullett, The QPSX man, *IEEE Commun. Mag.*, **26** (4): 20–28, 1988.
4. R. M. Newman and J. L. Hullett, Distributed queueing: A fast and efficient packet access protocol for QPSX, *Proc. 8th Internatl. Conf. on Comp. Comm.*, Munich, F.R.G., Sept. 15–19, 1986, pp. 294–299.
5. M. Zukerman and P. G. Potter, "A protocol for eraser node implementation within the DQDB framework," *Proc. IEEE GLOBECOM '90*, San Diego, C., Dec. 1990, pp. 1400–1404.
6. M. W. Garrett and S.-Q. Li, "A study of slot reuse in dual bus multiple access networks," *IEEE J. Select. Areas Commun.*, **9** (2): 248–256, 1991.
7. J. W. Wong, "Throughput of DQDB networks under heavy load," *EFOC/LAN-89*, Amsterdam, The Netherlands, June 14–16, 1989, pp. 146–151.
8. E. L. Hahne, A. K. Choudhury, and N. F. Maxemchuk, "Improving the fairness of distributed-queue dual-bus

networks," *INFOCOM '90*, San Francisco, CA, June 5–7, 1990, pp. 175–184.

9. N. F. Maxemchuk, "Regular mesh topologies in local and metropolitan area networks," *AT&T Tech. J.*, **64** (7): 1659–1686, 1985.
10. S. Bassi, M. Decina, P. Giacmazzi, and A. Pattavina, "Multistage shuffle networks with shortest path and deflection routing for high performance ATM switching: The open loop shuffleout," *IEEE Trans. Commun.*, **42** (10): 2881–2889, 1994.
11. A. Krishna and B. Hajek, "Performance of shuffle-like switching networks with deflection," *Proc. INFOCOM '90*, June 1990, pp. 473–480.
12. N. F. Maxemchuk, "Problems arising from deflection routing: Live-lock, lockout, congestion and message reassembly," in G. Pujolle (ed.), *High Capacity Local and Metropolitan Area Networks*. New York: Springer-Verlag, 1991, p. 209–233.
13. P. Baran, "On distributed communications networks," *IEEE Trans, Commun. Syst.*, **cs–12** (1): 1–9, 1964.
14. J. T. Brassil, A. K. Choudhury, and N. F. Maxemchuk, "The Manhattan Street Network: A high performance, highly reliable metropolitan area network," *Comput. Networks ISDN Syst.*, **26** (6–8): 841–858.
15. N. F. Maxemchuk, "Routing in the Manhattan Street Network," *IEEE Trans, Commun.*, May **COM–35** (5): 503–512, 1987.
16. N. F. Maxemchuk and A. N. Netravali, "Voice and data on a CATV network," *IEEE J. Select. Areas Commun.*, **SAC–3** (2): 300–311, 1985.
17. N. F. Maxemchuk, "A variation on CSMA/CD that yields movable TDM slots in integrated voice/data local networks," *BSTJ*, **61** (7): 1527–1550, 1982.

N. F. Maxemchuk
Columbia University
New York, New York

MOBILE AND UBIQUITOUS COMPUTING

OVERVIEW

The development of wireless communication technology and portable computing devices in late 1980s and early 1990s has led to a new computing paradigm, *mobile computing,* in which mobile devices capable of wireless communications are used to perform various computing tasks (1). Because of the characteristics of mobile environments, such as user mobility and severe resource constraints of mobile devices, many challenges exist in mobile computing, which include wireless ad hoc communications, mobility, portability, scalability, resource constraints, and adaptability (2–5). Since the 1990s, substantial effort has been made in various areas, which include networking, database, security, and software engineering, to address these challenges. However, many challenging issues still need to be addressed.

The concept of *ubiquitous computing* was first introduced by Weiser in 1991 (6) based on the idea that the most powerful and useful technologies ever invented are those that become indistinguishable from our daily life (6). In his view, ubiquitous computing represents a new computing paradigm in which information processing needed by a person is done by hundreds of computing devices of various scales, from PDAs to desktop PCs and to even supercomputers, connected through wired or wireless networks transparently. Comparing ubiquitous computing with mainframes shared by many users and personal computers owned by individual users, Weiser considered ubiquitous computing the third wave in computing in which many computers serve one user (7). Ubiquitous computing makes computing invisible to people and allows them to focus more on their uses rather than their computers.

Ubiquitous computing has many applications, from intelligent environmental control and smart home appliances, to interactive workspaces, mobile patient-care systems, context-aware tourist guides, and smart classrooms. An exemplified scenario can be found in Ref. 6 to illustrate a day of life in a world of ubiquitous computing. In this scenario, tiny electronic tabs affixed with various objects are used to identify and to locate useful items. In-vehicle devices are used to display the traffic condition and to find parking spaces. Handheld computers as well as large interactive display systems are used to create virtual offices for collaborative works.

Because of user mobility, and because of heterogeneous and dynamically changing computing environments, many research issues need to be addressed in ubiquitous computing, such as design of tiny, inexpensive, and energy-efficient mobile devices; interconnection of wired and wireless networks; lightweight system software for ubiquitous computing devices; techniques and tools for developing smart ubiquitous computing application software; and various security and privacy issues in ubiquitous computing environments. Since early 1990s, much research has been performed to develop the enabling techniques for ubiquitous computing to address these issues, and some technical terms have been created and used to describe similar computing technologies for ubiquitous computing, such as "pervasive computing," "ambient intelligence," and "invisible computing (8–10)." Currently, a common definition for ubiquitous computing is that it is a model of human—computer interaction in which information processing has been integrated thoroughly into everyday objects and activities, or, simply "computing anytime, anywhere (11)."

Mobile computing is related closely to ubiquitous computing, but it has a different emphasis. Mobile computing emphasizes mobility (i.e., the capability to continuously perform computing tasks in mobile environments). Although mobility is one of the important requirements for ubiquitous computing, the major concern of ubiquitous computing is how to make the interactions between human and computers transparent to human users. This concern requires ubiquitous computing systems to be aware of users' needs and the ambient environment and to adapt themselves to provide satisfactory services to users continuously in dynamically changing ubiquitous computing environments. Usually this feature is referred to as *context- / situation awareness*. In addition, computing devices used in ubiquitous computing environments are not limited to mobile devices, and they are connected through wired and/or wireless networks.

Despite these differences between mobile and ubiquitous computing, the technologies for mobile computing are still important for ubiquitous computing because ubiquitous computing environments also have many characteristics similar to those of mobile environments, such as user mobility and usage of wireless networks. Hence, some researchers consider that the research in these two areas should be combined (12), and some researchers even consider that the research of ubiquitous computing subsumes that of mobile computing (10). In this article, we will not distinguish the research in these two closely related areas, and we will use the term "mobile and ubiquitous computing" to refer to the combination of these two areas.

Research on mobile and ubiquitous computing spans across many different aspects, which include networking, databases, artificial intelligence, operating systems, software engineering, security and privacy, and so forth. It is impossible to enumerate and to discuss all the important research issues in this article. Hence, in this article, we will summarize the current state of research in the following four aspects: wireless ad hoc networks, context-/situation-awareness in mobile and ubiquitous computing environments, techniques for developing mobile and ubiquitous computing software, and privacy issues in mobile and ubiquitous computing. The research results in these four aspects not only include important enabling techniques for

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

developing mobile and ubiquitous computing systems, but also provide the most distinct features of mobile and ubiquitous computing systems.

WIRELESS AD HOC NETWORKS

The main advantage of mobile and ubiquitous computing environments is the capability of integrating heterogeneous mobile computing devices in various network domains to provide users with the capability of "anywhere, anytime" computing.

Being different from the Internet in which terminal hosts are connected to routers via a fixed network infrastructure, the heterogeneous mobile computing devices in mobile and ubiquitous computing environments essentially communicate with each other in an autonomous manner to construct a wireless ad hoc network. A *wireless ad hoc network* is a self-configuring network of mobile nodes connected by wireless links. Without a fixed network infrastructure, every node acts as a terminal host and a router simultaneously for the data transmission in the network. The network topology is self-organizing and geographically dispersed in the sense that a link only exists between two mobile nodes if they are within the physical communication range with each other. A wireless ad hoc network may operate in a stand-alone fashion, or may be connected to the larger Internet.

In wireless ad hoc networks, many research challenges have been identified from the mobility and energy constraints of the individual handheld computing devices, which are the major types of nodes in wireless ad hoc networks. In the following sections, these challenges and solutions are reviewed briefly in the categories of different layers of the network protocols hierarchy.

Medium Access Control (MAC) Layer

The major challenge for the MAC layer in the wireless ad hoc networks is how the MAC layer protocols are designed to allocate the communication resources, such as the available bandwidth of wireless channels, and to optimize the network performance efficiently, which can be measured in term of throughput, transmission delay, and fairness. In this subsection, we will discuss briefly the major MAC protocols currently used.

Carrier Sense Medium Access (CSMA). Because of the lack of centralized control in wireless ad hoc networks, the MAC protocols in this area are primarily contention based. CSMA is one of the earliest mechanisms adopted for wireless ad hoc networks. In CSMA, a transmitter first senses the wireless channel in the vicinity and refrains itself from transmission if the channel is already in use. Various methods such as ALOHA (13) and *n*-persistent algorithms (14), can be used to determine how long the deferred node should wait before the next attempt.

Medium Access Collision Avoidance (MACA). MACA is a "virtual sensing" mechanism instead of physical sensing. Such a mechanism is also called packet sensing. Typically, the virtual sensing mechanisms rely on the transmitter and the receiver to perform a handshake prior to the transmission of the data packet. Specifically, the MACA method (15) conducts the handshake via a pair of request-to-send (RTS) and clear-to-send (CTS) messages. When a node wants to send data to another node, it first sends a short RTS packet to the destination. The receiver responds with a CTS packet. On receiving the CTS packet, the sender sends its queued data packet(s). All other nodes that overhear the CTS message will defer themselves from sending out any packets until the predicted transmission period indicated in the CTS packet is passed. Any node that overhears the RTS signal, but not CTS, is allowed to send out packets in a certain time period as either the RTS/CTS handshake is not completed or it is out of range of the receiver.

IEEE 802.11. The IEEE 802.11 MAC protocol (16) is another example of using both physical sensing and RTS/CTS handshake mechanisms. IEEE 802.11 is defined actually as the standard MAC and physical protocols for wireless LANs, not specially designed for multihop wireless ad hoc networks. The MAC layer consists of two core functions: a distributed coordination function (DCF) and a point coordination function (PCF).

DCF controls the medium access through the use of carrier sense multiple access with collision avoidance (CSMA/CA) and a random backoff algorithm. Carrier sense in CSMA/CA is performed using both physical and virtual mechanisms. Basically, a node can access the channel only if no signal is physically detected. RTS/CTS mechanism in IEEE 802.11 can also be used in the situations, where multiple wireless networks utilizing the same channel overlap, as the medium reservation mechanism works across the network boundaries.

Although DCF is designed for asynchronous contention-based medium access, the IEEE 802.11 MAC protocol also defines PCF which is based on DCF and supports allocation-based medium access in the presence of an access point (AP). An AP plays the role of a point coordinator and polls each participating node in a round robin fashion (17) to grant medium access on allocation basis. PCF is not suitable for wireless ad hoc networks because it requires centralized control by the AP, which is not available in such networks.

Routing Protocols in Wireless Ad Hoc Networks

Routing in wireless ad hoc networks encounters severe challenges from node mobility/dynamics, with potentially very large numbers of nodes and limited communication resources, such as the network bandwidth and energy of mobile nodes. The routing protocols for wireless ad hoc networks have to adapt quickly to frequent and unpredictable topology changes and must be efficient in terms of the communication overhead. Furthermore, because bandwidth is scarce in wireless ad hoc networks and the sizes of such networks are usually small compared with the wired Internet, the scalability issue for wireless multihop routing protocols is concerned mostly with excessive routing message overhead caused by the increase of network population and mobility. In this subsection, we will discuss briefly some major routing protocols currently used.

Classifications of Routing Protocols. Generally, routing protocols in wireless ad hoc networks use either distance-vector or link-state routing algorithms (18), both of which find shortest paths to destinations.

In distance-vector routing, a vector that contains the communication cost (e.g., hop distance) and next hops to all the routing destinations is kept and exchanged at each node. Distance-vector protocols suffer from slow route convergence and a tendency to create loops in mobile environments.

The link-state routing algorithm overcomes the problem by maintaining global network topology information at each router through periodical flooding of link information on its neighbors. However, such link-state advertisement scheme generates larger routing control overhead than that of the distance-vector protocols.

In large wireless ad hoc networks, the transmission of routing information will consume most of the bandwidth and unconsequently with block applications, which renders it unfeasible for bandwidth limited wireless ad hoc networks. Thus, reducing routing control overhead becomes a key issue to achieve routing scalability. Such scalability is more challenging in the presence of high-node mobility. When nodes in the network are moving, the hierarchical partitioning must be updated continuously. Mobile IP solutions work well if a fixed infrastructure exists. However, when all nodes are moving, such solutions cannot be applied directly.

Routing protocols in wireless ad hoc networks can be classified into two categories: proactive and reactive. Many proactive protocols stem from conventional link-state routing and will cause large communication overhead in dynamic network topology. *On-demand routing,* however, is an emerging reactive routing in wireless ad hoc networks. It differs from conventional routing protocols in that no routing activities and no permanent routing information are maintained at network nodes if no communication occurs in the network. Hence, it provides a scalable routing solution. Such a feature makes the on-demand routing protocol efficient in controlling the communication overhead in wireless ad hoc networks. Because on-demand routing protocols are the mainstream protocols in wireless ad hoc networks, in the remainder of this section, we will focus our discussion on the on-demand routing protocols only.

On-Demand Routing Protocols. The design of on-demand routing protocols is based on the idea that each node tries to reduce routing overhead by only broadcasting routing requests when the communication of the node is awaiting. Representative examples include ad hoc on demand distance vector routing (AODV) (19), dynamic source routing (DSR) (20), and temporally ordered routing algorithms (TORA) (21). Among these protocols, AODV and DSR have been evaluated extensively in the wireless ad hoc networks literature and are being considered by the Internet Engineering Task Force (IETF) MANET Working Group as the leading candidates for standardization.

Typically, on-demand algorithms have a route discovery phase. Query packets are flooded into the network by the sources in search of a path. This phase completes when a route is found or when all possible outgoing paths from the source are searched. Different approaches for discovering routes exist in on-demand algorithms. In AODV, on receiving a query, the intermediate nodes “learn” the path to the source and enter the route in the forwarding table. Eventually, the intended destination receives the query and can respond “using the path traced by the query.” This function permits the establishment of a full duplex path. DSR uses an alternative scheme for tracing on-demand paths, (i.e., source routing) in which a source indicates in a data packet’s header the sequence of intermediate nodes on the routing path. In DSR, the query packet copies in its header the IDs of the intermediate nodes it has traversed. The destination then retrieves the entire path from the query packet, and it uses the retrieved path (via source routing) to respond to the source, which provides the source with the path at the same time. Data packets carry the source route in the packet headers, and a DSR node caches the routes aggressively to minimize the cost incurred by the route discovery.

Generally, AODV and DSR are used in flat network architectures. However, when the size of the wireless ad hoc network increases beyond a certain threshold, the flat routing schemes become infeasible because of the exponential increase of link and processing overhead. One way to solve this problem and to produce scalable and efficient solutions is hierarchical routing. Hierarchical routing in wireless ad hoc networks is based on the idea of organizing nodes in groups and then assigning nodes different functionalities inside and outside a group. Both routing table size and update packet size are reduced by including only part of the network instead of the whole network; hence, the communication overhead is reduced. The most popular way of building hierarchy is to group nodes geographically close to each other into explicit clusters. Each cluster has a leading node (*clusterhead*) to communicate to other nodes on behalf of the cluster (22). An alternate way is to have implicit hierarchy, in which each node has a local scope, different routing strategies are used inside and outside the scope, and communications pass across overlapping scopes. Because mobile nodes have only a single omni-directional radio for wireless communications, this type of hierarchical organization is referred to as *logical hierarchy* to distinguish it from the physical hierarchy of network structure. Representative examples of hierarchical routing protocols include Clusterhead-Gateway Switch Routing (CGSR) (23) and Zone Routing Protocol (24).

TCP in Wireless Ad Hoc Networks

Transmission control protocol (TCP) is the transport layer protocol that provides reliable end-to-end data delivery in unreliable networks. Because of its wide use in the Internet, it is desirable to keep using TCP remains to provide reliable data transfer services within wireless ad hoc networks. Unfortunately, wireless ad hoc networks differ from wired Internet significantly in terms of bandwidth, propagation delay, and link reliability. The implication of these differences is that packet losses are no longer caused mainly by network congestion. Instead, most packet losses are caused by high bit error rate in wireless channels and route breakages in dynamic network topology. Hence, the TCP performance faces the following challenges in wireless ad hoc networks, in which the network topology is highly dynamic:

- **Channel errors.** In wireless channels, relatively high bit error rate caused by multi path fading and shadowing may corrupt packets in transmission, leading to loss of TCP data segments or acknowledgments (ACKs). If a TCP sender cannot receive the ACK within the retransmission timeout, it reduces its congestion window to one segment immediately, exponentially backs off (25) its retransmission timeout (RTO), and retransmits the lost packets. Thus, intermittent channel errors may cause the congestion window size at the sender small, which results in low TCP throughput.
- **Mobility.** Mobility may cause link breakage and route failure between two neighboring nodes, when one mobile node moves out of the other's transmission range. In turn, link breakage causes packet losses. Because TCP cannot distinguish between packet losses caused by route failures and packet losses caused by congestion, TCP congestion control mechanisms react adversely to such losses caused by route breakages (26). Meanwhile, discovering a new route may take longer time than TCP sender's RTO. If route discovery time is longer than RTO, the TCP sender will invoke congestion control after timeout. The throughput, which has already reduced, will decrease even more because of the packet loss. It will become worse if the sender and the receiver of a TCP connection belong to different network partitions. In such a case, multiple consecutive RTO timeouts lead to inactivity for one or two minutes even if the sender and receiver finally get reconnected.
- **Multi-path routing.** Routes in wireless ad hoc networks are short-lived because of frequent link breakages. To reduce the delay caused by route recomputation, some routing protocols, such as TORA (21), maintain multiple routes between a sender-receiver pair, and they use multi path routing to transmit packets. In such a case, packets that come from different paths may not arrive at the receiver in the same order as they were sent out. Not aware of multi path routing, the TCP receiver will misinterpret such out-of-order packet arrivals as a sign of congestion. The receiver will then generate duplicated ACKs that cause the sender to invoke congestion control algorithms like fast retransmission (on reception of three duplicated ACKs according to TCP protocol).
- **Congestion.** The attempt of TCP to use the network bandwidth fully would easily make wireless ad hoc networks congested. Because of the factors such as route change and unpredictable variable MAC delay, the relationship between congestion window size and the tolerable per-link data rate is no longer maintained in ad hoc networks. The congestion window size computed for the old route may be too large for the newly found route, which results in network congestion as the sender still transmits at the full rate allowed by the old congestion window size.

Three types of performance enhancement schemes have been proposed to improve TCP performance over wireless ad hoc networks. The first scheme (27,28) improves the TCP performance using feedback schemes. Through the use of feedback information to signal non congestion-related causes of packet losses, the feedback approaches help TCP distinguish between true network congestion and other problems, such as channel errors, link contention, and route failures. The second scheme (26,29) makes TCP adapt to route changes without relying on feedback from the network, in light of the concern that feedback mechanisms may cause additional complexity and cost in wireless ad hoc networks. The third scheme (30,31) tailors lower layers, such as routing layer and MAC layer, according to TCP congestion control algorithms.

CONTEXT-AWARE/SITUATION-AWARE COMPUTING

As described in the first section, context-aware/situation-aware computing is a major feature of mobile and ubiquitous computing. In this section, we will discuss the basic concepts and applications of context-aware/situation-aware computing, and how contexts and situations are modeled. Techniques for developing context-aware/situation-aware software will be covered in the next section. Becasue of limited space, other research issues related to context-aware/situation-aware computing, including context sensing, and contextual and situation information management, are not covered in this article. Readers interested in these topics are referred to the conferences and periodicals listed in the last section.

Basic Concepts

Although several issues related to context-aware computing have been discussed in Refs. 32–35, the term "context-aware computing," which was first introduced in 1994 (36), is defined as the "ability of a mobile user's applications to discover and react to changes in the environment they are situated in," and the term *context* is defined as "the location of use, the collection of nearby people and objects, as well as the changes to those objects overtime".

Since then, much research has been done in the mobile and ubiquitous computing community on context-aware computing. Various definitions of context have also been made (36–44). For example, Dey and Abowd (40) defined context as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." Chen and kotz (41) defined context as "the set of environmental states and settings that either determines an application's behavior or in which an application event occurs and is interesting to the user."

Based on various definitions of context, a context has the following properties: (*1*) A context changes over time and

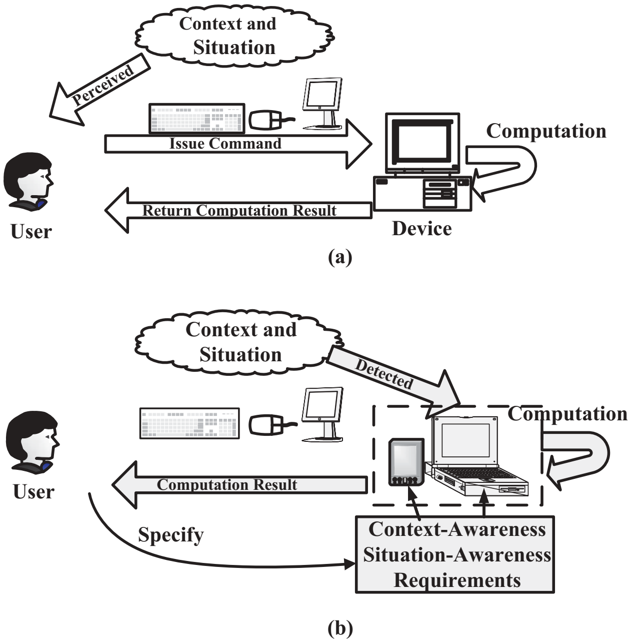


Figure 1. Different types of human-computer interactions: (a) user-initiated human-computer interaction and (b) context-aware/situation-aware human-computer interactions.

is meaningful only when it is associated with a time instant, (*2*) A context must be detectable with appropriate hardware or software support so that it could be used in computing, and (*3*) A context must be relevant to the interactions between a user and an application. Based on these properties, Yau, et al. (42,43) defined context as "any instantaneous, detectable, and relevant property of the environment, the system, or users."

Besides the differences in the definitions of context, various ways exist to categorize contexts (37,40,41). Schilit, et al. (37) considered context in computing, user, and physical environments. Dey and Abowd (40) considered location, identity, time, and activity as four primary types of contexts, and they considered other types of contextual information as secondary pieces because they can be indexed by the four primary types of contexts. Chen and Kote (41) suggested to categorize contexts as computing, user, physical, and time. Nevertheless, these characterizations of context are informal and only aim at helping developers identify context to be used in their applications.

Another concept related closely to context is situation. Although context is considered sometimes the same as situation (45,46), normally context and situation are considered different (41–43,47). In Refs. 42 and 43, a *situation* is defined as a set of context attributes of users, systems, and environments over a period of time that affect future system behavior. Although other definitions exist for situation (48,49), it is agreed commonly that "situation" is a higher-level concept built upon "context," and a situation provides a higher-level understanding of the status of users or other objects involved in computing than a context. Similar to context-aware computing, situation-aware computing can be defined as the awareness of situations and adapting the system's behavior based on situation changes (42,43).

The importance of context-aware/situation-aware computing in mobile and ubiquitous computing environments lies on the new way of human-computer interaction in context-aware/situation-aware computing systems. Dey and Abowd (50) defined context-aware computing system as follows: "A system is *context-aware* if it uses context to provide relevant information and/or services to the user, relevancy depends on the user's task." As shown in Fig. 1, unlike traditional computing systems that operate on explicit inputs (data and commands) from users, context-aware computing systems consider contexts as implicit inputs and operate on both contexts and the explicit inputs from users (51). Hence, the interactions between human users and computers in context-aware computing are no longer initiated only by human users. Changes in contexts can also trigger and guide interactions between users and computers. This technique is extremely useful in mobile and ubiquitous computing environments, in which computing is expected to be invisible or distraction-free to users.

Applications of Context-Aware/Situation-Aware Computing

Context-aware/situation-aware computing is very useful in various mobile and ubiquitous computing applications, which are usually categorized based on how contextual and situation information is used in applications. Schilit, et al. (37) first classified context-aware applications in the following four categories:

- Proximate selection, which refers to applications that automatically find and display objects, such as documents, printers, and monitors, based on a user's location to allow easier access to these objects.

- Automatic contextual reconfiguration, which refers to applications that automatically add, remove, or change their components to satisfy users' needs in dynamically changing environments.
- Contextual information and commands, which refer to applications that augment or parameterize users' commands with context to allow users to access appropriate data or functions based on current contexts.
- Context-triggered actions, which refer to applications that automatically take actions under certain conditions based on predefined rules provided by users.

This categorization of context-aware applications has been refined, augmented, or simplified by others (40,52,53). Among these, Dey (40) provided the simplest and most general categories that describe possible features provided by context-aware applications:

- *Presentation* of information and services to users (based on contexts). An example for this category is an application that provides location-based services, such as searching nearby restaurants and printing documents on the nearest printer.
- Automatic *execution* of services (based on contexts). An example for this category is an environment control application that automatically adjusts brightness of lights at home based on residents' activities.
- *Tagging* of context to information for later retrieval. An example for this category is an address book application that provides not only the phone numbers, but also the current status (busy or free) of an addressee so that a caller can determine whether he wants to postpone the call (52).

Context and Situation Modeling

To facilitate the development and the operations of context-aware/situation-aware computing systems, proper models for contexts and situations are needed for specifying context-aware/situation-awareness requirements.

Context models have been classified into the following types: key-value pairs, object-oriented models, logic-based models, markup language-based models, and graph-based models (41,46). A key-value pair is the simplest form of context models, in which a key is the identifier of a particular context, and the corresponding value is the actual contextual information. In object-oriented models, contexts are properties of objects that represent physical or conceptual entities in mobile and ubiquitous computing environments. Logic-based models use logic programming languages, such as Prolog, to represent and reason contextual information. Markup language-based models describe contexts using various elements defined in markup language-format documents, such as XML. Graph-based models represent contexts and various activities to be performed based on contexts as nodes in graphs, and they represent relations among contexts and activities as edges in graphs.

Inspired by the development of the semantic web, recent research on context modeling has resulted in ontology-based models, such as CONON (54) and SOUPA (55), which are usually based on standard ontology languages like OWL (56). Compared with earlier work on context modeling (41,46), ontology-based context models can express semantic-rich information related to contexts and have better support for reasoning with contexts. These advantages of ontology-based models make them more suitable for developing ubiquitous computing applications, which need to adapt to environment changes intelligently.

Early work on situation modeling was mainly conducted in the artificial intelligence community. Situation Calculus and its extensions (57–60) were developed for describing and for reasoning how actions and other events affect the world, and assume that all actions and events that change the world are known or predictable. In Situation Calculus, a situation is considered as a complete state of the world, which leads to the well-known Frame Problem and Ramification Problem (58). Barwise (61) defined a situation as a part of "the way the world M happens to be," which supports the truth of a sentence Φ in M. In addition, Barwise formally defined the semantics of a situation (62) based on a "scene," which is a "visually perceived situation" that consists of not only the objects and individual properties associated with the objects, but also the relationships among objects. Barwise's definition of situation is more practical compared with the definition of situation in Situation Calculus because it allows the precise description of situations, and it can be supported easily by the prevailing object-oriented modeling techniques. Currently, many researchers have adopted Barwise's definition of situation and developed their own formalisms of situations for various purposes, such as supporting information fusion (63,64), situation-aware software development (65–67), and effective human-computer interaction (68). For example, a core situation awareness (SAW) ontology was introduced in (63,64) based on a similar view of situations as Barwise's, which defines a situation as a collection of situation objects, including objects and relations as well as other situations. Another example is a declarative SAW model presented by Yau et al, (65,66), which provides developers the capability to specify the situations of interest, the contexts required for analyzing the situations, and the relations among various situations and actions graphically. Software tools have been developed based on this declarative SAW model to translate graphical specifications automatically to specifications based on formal languages, such as F-Logic and AS^3 logic, and to synthesize software agents automatically for distributed context acquisition and situation analysis (65–67).

TECHNIQUES FOR DEVELOPING MOBILE AND UBIQUITOUS COMPUTING SOFTWARE

How to develop software for mobile and ubiquitous computing is a problem, which has attracted substantial attention since the very beginning of mobile and ubiquitous computing research. The major challenges in developing mobile and ubiquitous computing software include location transparency, disconnected operations, interoperability, context-awareness/situation-awareness (1,69). Research in this area has mainly focused on middleware and software toolkits that provide appropriate development and runtime support to address these challenges. Hence, in this section we will focus on middleware and software toolkits facilitating the development of mobile and ubiquitous computing software. Readers interested in other related research topics, such as testing mobile and ubiquitous computing software, are referred to the conferences and periodicals listed in the last Section.

Existing middleware for mobile and ubiquitous computing can be divided into two major categories based on how they support the coordination among mobile devices: (1) object based, and (2) tuple-space based. Notable work from the first category includes ALICE (70), Mobiware (71), GAIA (72), RCSM (42,43), and MobiPADS (73), which are based on object-oriented middleware architecture like CORBA (Common Object Request Broker Architecture) (74). Notable work in the second category includes LIME (75) and TSpace (76). Their tuple-space based coordination model supports location transparency and disconnected operations, and mobility is viewed as transparent changes in data stored in the tuple space (75,76).

Besides middleware for mobile and ubiquitous computing, various embedded operating systems, such as Windows CE (77), embedded Linux (78) and Plan 9 (79), along with platform-specific software development kits have been developed to facilitate the development of mobile and ubiquitous computing software.

Among the major challenges in developing mobile and ubiquitous computing software, the incorporation of context-awareness/situation-awareness has attracted most attention because it has not been previously addressed in traditional distributed computing research. Many challenging issues for developing context-aware/situation-aware software in mobile and ubiquitous computing environments have been identified, which include the discovery and the management of heterogeneous context sources, persistent storage of contextual and situation information, analysis of collected context data for determining the situation, interfacing context-aware/situation-aware software with heterogeneous hardware platforms (40,50,80,81).

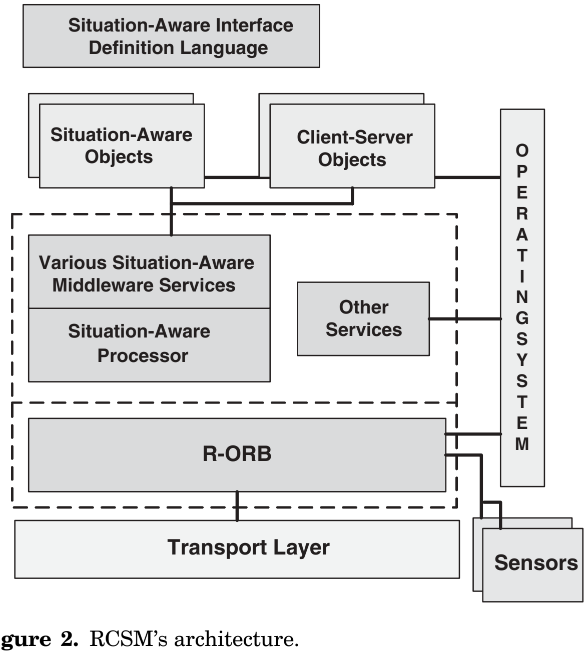

Situation-Aware Interface Definition Language
Situation-Aware Objects
Client-Server Objects
OPERATING SYSTEM
Various Situation-Aware Middleware Services
Situation-Aware Processor
Other Services
R-ORB
Transport Layer
Sensors

Figure 2. RCSM's architecture.

Several frameworks, toolkits and infrastructures have been developed for providing support to context-aware application development. Notable results include CALAIS (82), Context Toolkit (50), CoolTown (83), MobiPADS (73), GAIA (72,84), TSpaces (76,85) and RCSM (42,81). CALAIS (82) focuses on applications accessible from mobile devices, and supports acquisition of context about users and devices, but it is difficult to evolve existing applications when the requirements for context acquisition and the capabilities and availabilities of sensors change. Context Toolkit (50) provides architectural support for context-aware applications, but it does not provide analysis of complex situations. CoolTown (83) supports applications that display contexts and services to end-users. MobiPADS (73) is a reflective middleware designed to support dynamic adaptation of context-aware services, and hence enables runtime reconfiguration of context-aware applications. GAIA (72,84) provides context service, space repository, security service and other QoS for managing and interacting with active spaces. TSpaces (76,85) uses tuple spaces to store contexts and allows tuple space sharing for application software to read and write, but it ignores the status of the device where the application software executes, network conditions, and the surrounding environment as part of the overall context. RCSM (Reconfigurable Context-Sensitive Middleware) (42,81) provides development and runtime support for situation-aware (SA) application software. Because of limited space, we will only give a brief overview in the following subsections on two middleware, RCSM and MobiPADS, which provide context-awareness/situation-awareness. Readers interested in this are referred to the conferences and periodicals listed in the last Section.

RCSM

RCSM is a lightweight SA middleware, which provides development and runtime support for SAW, dynamic service discovery and group communication for ubiquitous computing applications (42,43,80,81). A conceptual architecture of RCSM, which is shown in Fig. 2, consists of the following major components:

1. SA Processor provides the runtime services for situation analysis and manages the SAW requirements of SA objects. The SAW requirements of SA objects are defined using situation-aware interface definition language (SA-IDL). An SA-IDL compiler was developed to generate the situation-aware object skeleton codes and corresponding configuration files, to be used by the SA Processor to perform situation analysis. The SA object skeleton codes provide the standard interfaces for SA objects to interact with the SA Processor.
2. RCSM object request broker (R-ORB) provides the runtime services for context discovery and acquisition, and SA communication management. The context manager in R-ORB implements an efficient context discovery protocol (86) to support adaptive context discovery and acquisition in ubiquitous computing environments based on the requirements on

contexts extracted from the configuration files of SA applications by the SA processor. SA object discovery protocols enable efficient and spontaneous communication between distributed SA objects.

Using SA-IDL, contexts can be described precisely as context objects, and situations can be composed by not only the current values of multiple contexts, but also the historical values of multiple contexts over a period of time. The SA processor is designed to cache and analyze the context history to determine the situation. In addition, the SAW requirements, such as the definitions of situations, can be modified in runtime through the SA Processor. Once the requirements are changed, the R-ORB and SA Processor will reconfigure themselves to collect the necessary contexts and perform situation analysis based on the new requirements.

MobiPADS

Mobile platform for actively deployable service (MobiPADS) (73) is a reflective middleware, which serves as an execution platform for context-aware mobile computing. MobiPADS enables active service deployment and reconfiguration in response to context changes, and hence it can optimize the performance of mobile applications when the context changes.

MobiPADS consists of two types of agents: MobiPADS server agents, which reside in the network infrastructure and are responsible for most of the optimization computations, and MobiPADS client agents, which reside in the mobile devices and provide various services for mobile applications. MobiPADS adopts the idea of mobile codes, and stores the codes of service objects in MobiPADS agents. Service objects can be deployed on either the client or server agent, and can migrate between the client and server agents when needed (e.g., when the device, where the client agent resides, moves), to enable flexible reconfiguration of mobile applications. Each MobiPADS agent also has a set of system components for managing system (MobiPADS client and server, and service objects) configurations, migrating service objects between MobiPADS server and client, recording known services, contextual event notification, and establishing virtual communication channels between service objects. Each MobiPADS service is a pair of mobilets: a slave mobilet at the server agent for providing actual processing capabilities, and a master mobilet at the client agent for instructing the slave mobilets and for presenting results to the client. The mobilets can be chained together to support necessary service composition for mobile applications, similar to workflows in workflow systems. An XML-based language was developed in MobiPADS to describe how service objects interact with each other and how they are configured.

MobiPADS uses an event subscription-notification model to provide context-awareness. The idea is similar to the ECA (event-condition-action) model in active databases (87). All contexts are modeled as event sources to generate contextual events when certain conditions are satisfied. All entities (system components, mobilets and mobile applications) can subscribe to contextual events of interests, and they are notified when certain events occur to achieve context-awareness. MobiPADS also supports event composition, which allows combining multiple events from different context sources to express complex semantics. However, MobiPADS only focuses on the current events, and it does not consider historical events, which are important for achieving SAW.

PRIVACY ISSUES IN MOBILE AND UBIQUITOUS COMPUTING

Currently, most research in mobile and ubiquitous computing focuses on how to connect users and service providers in heterogeneous and dynamically changing computing environments, and how to develop applications cost-effectively. However, lack of privacy protection in mobile and ubiquitous computing would hinder its practical usage, and hence should be considered seriously from the beginning of system design. Although many privacy techniques are available to protect digital communications, the context-aware/situation-aware property of mobile and ubiquitous computing creates many new challenges and makes many existing techniques unsuitable (88–91). In such an environment, we need to consider the following two important aspects: (*1*) the protection of context information and (*2*) the authentication based on context information. In this section, we will focus on these two aspects. Other privacy issues, such as identity protection, secure communication, and key management, can be addressed using existing techniques (88–91) developed for general distributed computing systems, and hence they will not be discussed in this chapter.

Context Information Privacy

In mobile and ubiquitous computing, the contexts of users are very important because service providers may control or adapt their services according to their users' contexts, such as providing local weather information on users' locations. Hence, context information of users should be available to service providers to help them improve service qualities and provide personalized services. However, context information should be protected because revealing such information, which may be considered sensitive in certain applications, creates significant privacy risk, such as allowing malicious service providers to trace users. These two conflicting requirements on context information make the protection of context information difficult. So far, most research in this area focuses on the protection of location information (92).

A possible solution for these two conflicting requirements is to develop anonymity techniques, which will break the associations between users' identities and their contexts. Thus, service providers can use users' context information, but they cannot link two different contexts to the same user. The anonymous use of location information can be accomplished easily if a centralized trusted location server (93–95) exists, which serves as the proxy between users and service providers. The location server receives the location information from users and disseminates it to service providers without revealing the information sources. However, whereas the centralized trusted location server can perfectly make the location information anonymous, it is the performance bottleneck and may become the single point of failure. Although distributing the service provided by the centralized trusted location server to several servers may reduce such negative impact, it is difficult for administrators to protect and for users to trust multiple servers. A possible solution presented in Ref. 96 is to separate the protection of identities and the protection of locations. The distributed servers will only be responsible for the protection of users' identities. Users' locations can be protected by grouping several users together and revealing only the group locations.

Other researchers have developed techniques to reduce the risk of revealing users' location information rather than making the location information anonymous (96). Obfuscation was proposed to provide only inaccurate information to service providers, like referring the exact location as the location around a landmark (97), or replacing one user' location with the location of a group (98). Because different services have different requirements on the precision of locations (e.g., city names are sufficient for weather services, but more accurate location is required to find the nearest hospital), the idea of obfuscation is developed even more to allow users and service providers to negotiate for the precision of location information (99,100), which ensures that only the information required for providing satisfactory services is revealed to service providers.

Although many approaches have been developed to protect users' locations, these approaches cannot be extended easily to protect other context information. Hence, more investigations on how to protect other context information are needed.

Context-Based Authentication

Authentication is to ensure that users and service providers have the identities as they have claimed. In mobile and ubiquitous-computing environments, besides the requirement of the users' identities, additional requirements exist on the users' environment because some service providers only provide services for users with a particular context, such as in a certain building. Hence, the authentication in such environments should include not only users' identities, but also users' contexts, such as the characteristics of communication channel (101) or users' locations (102–105).

To authenticate users' contexts, the service providers need to collect users' context information. This task can be done using various types of sensors, such as infrared beams (102), laser beams (106), and ultrasound (107). Based on the properties of sensors, various types of sensors can collaborate when the requirement on users' context information is too complex to be collected and preprocessed by a single mechanism. For example, the RF sensor and the ultrasound sensor (105) can be combined to determine whether a user is in the room and whether three users are in line.

During the context information collection process, the authentication for the identities of service providers is required to distinguish one service provider from others to avoid revealing users' context information to malicious service providers. To solve this problem, various physical and software-based solutions have been developed to ensure the associations between users and service providers (102,106,108–111).

Because context-based authentication also authenticates users' context information, the collected context information should be integrated into the authentication protocols. In spatial reference (105), the distance between a user and a service provider is represented as time latency. Once the user receives an RF signal from the service provider, the user inserts a certain delay before sending the service provider a response. This delay represents the distance between the user and the service provider. However, this approach will degrade the performance, especially when the encoded information is large. Another solution for integrating the context and the authentication is to derive the key from the collected context information directly (112,113).

Similar to existing research on the protection of context information, the research on context-based authentication focuses mainly on location-based authentication. Authentication based on other context information need to be investigated even more.

SUMMARY

In this chapter, we have presented a brief overview of mobile and ubiquitous computing, and we reviewed four important research areas in mobile and ubiquitous computing: wireless ad hoc networks, context-aware/situation-aware computing, techniques for developing mobile and ubiquitous computing software, and privacy issues in mobile and ubiquitous computing. Because of limited space, the materials are presented at a relatively high level. Readers interested in this topic are referred to the references. More references can be found in the following conferences and periodicals: Annual International Conference on Mobile Computing and Networks (MobiCom), International Conference on Distributed Computing Systems (ICDCS), International Conference on Ubiquitous Computing (Ubicomp), International Conference on Pervasive Computing and Communications (PerCom), International Conference on Mobile Data Management (MDM), Annual International Computer Software and Application Conference (COMPSAC), Network and Distributed System Security Symposium (NDSS), *IEEE Transactions on Software Engineering, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, Journal of Parallel and Distributed Computing, IEEE Personal Communication, IEEE Pervasive Computing, Journal of Systems and Software, Journal of Software Practice and Engineering,* and *International Journal of Network Security.*

BIBLIOGRAPHY

1. D. Duchamp, S. K. Feiner and G. Q. Maguire Jr., Software Technology for Wireless Mobile Computing, *IEEE Trans. Network*, **5** (6): 12–18, 1991.

2. G. H. Forman and J. Zahorjan, The challenges of mobile computing, *IEEE Comput.*, **27** (4): 38–47, 1994.
3. T. Imielinski and B. R. Badrinath, Mobile wireless computing, *Commun. ACM*, **37** (10): 19–28, 1994.
4. L. Kleinrock, Nomadic computing: An opportunity, *Comput. Commun. Rev.*, **25** (1): 36–40, 1995.
5. M. Satyanarayanan, Fundamental Challenges in Mobile Computing, *Proc. 15th ACM Symp. on Principles of Distributed Computing*, 1996, pp. 1–7.
6. M. Weiser, The computer for the 21st century, *Scientif. Amer.*, **265** (3): 94–104, 1991.
7. M. Weiser and J. S. Brown, The coming age of calm technology, *Beyond Calculation – The Next Fifty Years of Computing*, P. J. Denning and R. M. Metcalfe, (eds.), Berlin: Springer-Verlag, 1996, Chapter 6.
8. D. Norman, *The Invisible Computer*, Cambridge, MA: MIT Press, 1998.
9. ISTAG (Information Society and Technology Advisory Group), Scenarios for Ambient Intelligence in 2010. Available: ftp://ftp.cordis.europa.eu/pub/ist/docs/istagscenarios2010.pdf, February 2001.
10. M. Satyanarayanan, Pervasive computing: Vision and challenges, *IEEE Personal Commun.*, **8** (4): 10–17, 2001.
11. D. Saha, and A. Mukherjee, Pervasive computing: A paradigm for the 21st century, *IEEE Comput.*, **36** (3): 25–31, 2003.
12. Y. R. Chen, and C. Petrie, Ubiquitous mobile computing, *IEEE Internet Computing*, **7** (2): 16–17, 2003.
13. N. Abramson, The ALOHA system-another alternative for computer communications, *Proc. 1970 Fall Joint Computer Confi*, 1970, pp. 281–285.
14. F. A. Tobagi, and L. Kleinrock, Packet switching in radio channels: Part I – carrier sense multiple-access modes and their throughput-delay characteristics, *IEEE Trans. Commun.*, **23** (12): 1400–1416, 1975.
15. P. Karn, MACA-A new channel access method for packet radio, *ARRL / CRRL Amateur Radio 9th Computer Networking Conf.*, 1990, pp. 134–140.
16. IEEE, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, *ANSI / IEEE std 802.11*, 1999 Edition (R2003), Part 11.
17. Wikipedia, Round-robin scheduling algorithm. Available: http://en.wikipedia.org/wiki/Round-robin_scheduling.
18. S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, The Internet, and the Telephone Network*, Reading, MA: Addison Wesley, 1997.
19. C. E. Perkins and E. M. Royer, Ad-hoc on-demand distance vector routing, *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999, pp. 90–100.
20. D. B. Johnson and D. A. Maltz, Dynamic source routing in ad hoc wireless networks, in T. Imielinski and H. Korth (eds.), *Mobile Computing*, Dordrecht, Germany: Kluwer Publisher, 1996, pp. 153–181.
21. V. D. Park and M. S. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks, *Proc. 16th IEEE INFOCOM*, 1997, pp. 1405–1413.
22. J. Y. Yu and P. H. J. Chong, A survey of clustering schemes for mobile ad hoc networks, *IEEE Communication Surveys & Tutorials*, **7** (1): 32–48, 2005.
23. C. C. Chiang and M. Gerla, Routing and multicast in multihop, mobile wireless networks, *Proc. 6th IEEE Int'l Conf. on Universal Personal Communications Record*, 1997, pp. 546–551.
24. Z. J. Haas and M. R. Pearlman, The performance of query control schemes for the zone routing protocol, *ACM / IEEE Trans, Network.*, **9** (4): 427–438, 2001.
25. IETF, RFC793-Transmission Control Protocol. Available: http://www.faqs.org/rfcs/rfc793.html.
26. T. D. Dyer and R. V. Boppana, A Comparison of TCP performance over three routing protocols for mobile ad hoc networks, *Proc. 2001 ACM Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, 2001, pp. 56–66.
27. K. Chandran, et al., A Feedback-based scheme for improving TCP performance in ad hoc wireless networks, *IEEE Personal Commun.*, **8** (1): 34–39, 2001.
28. J. Liu, and S. Singh, ATCP: TCP for mobile ad hoc networks, *IEEE J. Selected Areas in Communications*, **19** (7): 1300–1315, 2001.
29. K. Chen, Y. Xue, and K. Nahrstedt, On setting TCP's congestion window limit in mobile ad hoc networks, *Proc. 2003 IEEE Int'l Conf. on Communications (ICC'2003)*, 2003, pp. 1080–1084.
30. V. Anantharaman, et al., TCP performance over mobile ad-hoc networks: A quantitative study, *J. Wireless Commun. Mobile Comput.*, **4** (2): 203–222, 2003.
31. Z. Fu, et al., The impact of multihop wireless channel on TCP throughput and loss, *Proc. IEEE INFOCOM*, 2003, pp. 1744–1753.
32. R. Want, et al., The active badge location system, *ACM Trans. Inform. Syst.*, **10** (1): 91–102, 1992.
33. B. N. Schilit, M. Theimer, and B. B.Welch, Customizing mobile application, *Proc. USENIX Symp. on Mobile and Location-Independent Computing*, August 1993, pp. 129–138.
34. M. Spreitzer and M. Theimer, Providing location information in a ubiquitous computing environment, *Proc. 14th ACM Symp. on Operating System Principles*, 1993, pp. 270–283.
35. A. Harter and A. Hopper, A distributed location system for the active office, *IEEE Network*, **8** (1): 62–70, 1994.
36. B. N. Schilit, and M. Theimer, Disseminating active map information to mobile hosts, *IEEE Network*, **8** (5): 22–32, 1994.
37. B. N. Schilit, N. Adams, and R. Want, Context-aware Computing Applications, *Proc. 1st IEEE Workshop on Mobile Computing Systems and Applications*, 1994, pp. 85–90.
38. P. G. Brown, J. D. Bovey, and X. Chen, Context-aware applications: From the laboratory to the marketplace, *IEEE Personal Commun.*, **4** (5): 58–64, 1997.
39. P. Brezillon and J. C. Pomerol, Contextual knowledge sharing and cooperation in intelligent assistant systems, *Le Travail-Humain*, **62** (3): 223–246, 1999.
40. A. Dey, and G. Abowd, Towards a better understanding of context and context-awareness, *Technical Report, GIT-GVU-99–22*, Atlanta,GA: Georgia Institute of Technology, 1999.
41. G. Chen, and D. Kotz, A survey of context-aware mobile computing research, *Technical Report TR2000-381*, Dartmouth College, 2000. Available: http://www.cs.dartmouth.edu/reports/abstracts/TR2000-381/.
42. S. S. Yau, Y. Wang and F. Karim, Development of situation-aware application software for ubiquitous computing environments, *Proc. 26th IEEE Int'l Computer Software and Applications Conf (COMPSAC 2002)*, 2002, pp. 233–238.
43. S. S. Yau, et al., Reconfigurable context-sensitive middleware for pervasive computing, *IEEE Pervas. Comput.*, **1** (3): 33–40, 2002.

44. P. Braione and G. P. Picco, On calculi for context-aware coordination, *Proc. 6th Int'l Conf. on Coordination Models and Languages (COORDINATION 2004)*, 2004, pp. 38–54.

45. B. Schiele, et al., Situation aware computing with wearable computers, in W. Barfield and T. Caudell (eds.), *Augmented Reality and Wearable Computers*, Matawan, NJ. Lawrence Erlbaum Press. 1999.

46. G. K. Mostefaoui, J. Pasquier-Rocha, and P. Brezillon, Context-aware computing: A guide for the pervasive computing community, *Proc. IEEE/ACS Int'l Conf. on Pervasive Services (ICPS'04)*, 2004, pp. 39–48.

47. A. Schmidt, *Ubiquitous Computing: Computing in Context*, Ph.D. Thesis, 2002, Lancaster University, UK.

48. P. Marti, et al., Situated interactions in art settings, *Proc. Workshop on Situated Interaction in Ubiquitous Computing at CHI2000*, 2000. Available: http://www.teco. edu/chi2000ws/papers/29_marti.pdf.

49. T. Selker and W. Burleson, Context-aware design and interaction in computer systems, *IBM Syst. J.*, **39**(3–4), 2000. Available: http://cac.media.mit.edu:8080/contextweb/jsp/index.htm.

50. A. K. Dey and G. D. Abowd, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Human-Computer Interact.*, **16**(2–4): 97–166, 2001.

51. S. Pokraev, et al., Context-aware services: State-of-the-art, *TI/RS/2003/137*, 2003. Available: https://doc.telin.nl/dscgi/ds.py/Get/File-27859/Context-aware_services-sota,_v3.0,_final.pdf.

52. J. Pascoe, Adding generic contextual capabilities to wearable computers, *Proc. 2nd Int'lSymp. on Wearable Computers*, 1998, pp. 92–99.

53. D. Chalmers, Contextual Mediation to Support Ubiquitous Computing, Ph.D. Thesis, Imperial College, London, England, 2002.

54. X. Wang, et al., Ontology-based context modeling and reasoning using OWL, *Proc. Context Modeling and Reasoning Workshop at the 2nd IEEE Annual Conf. on Pervasive Computing and Communications*, 2004, pp. 18–22.

55. H. Chen, et al., SOUPA: Standard ontology for ubiquitous and pervasive applications, *Proc. Int'l Conf. on Mobile and Ubiquitous Systems: Networking and Services*, 2004, pp. 258–267.

56. OWL-S 1.0. Available: http://www.daml.org/services/owl-s/1.0/.

57. J. McCarthy and P. J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, *Mach. Intell. 4*. 463–502, 1969.

58. J. A. Pinto, Temporal reasoning in the situation calculus, Ph.D. Thesis, University of Toronto, 1994.

59. J. McCarthy, Situation calculus with concurrent events and narrative, 2000. Available: http: //wwwformal. Stanford. edu/jmc/narrati ve/narrati ve. html.

60. D. Plaisted, A hierarchical situation calculus, *J. Comput. Res. Reposit. (CoRR)*, 2003.

61. J. Barwise, Scenes and other situations, *J. Philos.*, **77**: 369–397, 1981.

62. J. Barwise, The situation in logic, *CSLI Lecture Notes 17*, 1989.

63. C. J. Matheus, M. M. Kokar, and K. Baclawski, A Core ontology for situation awareness, *Proc. 6th Int'l Conf on Information Fusion*, 2003, pp. 545–552.

64. C. J. Matheus, et al., Constructing RuleML-based domain theories on top of OWL ontologies, *Proc. 2nd Int'l Workshop on Rules and Rule Markup Languages for the Semantic Web keep: Rule ML*, 2003, pp. 81–94.

65. S. S. Yau, et al., Situation-awareness for adaptable service coordination in service-based systems, *Proc. 29th Ann. Int'l Computer Software and Application Conf. (COMPSAC)*, 2005, pp. 107–112.

66. S. S. Yau, et al., Support for situation-awareness in trustworthy ubiquitous computing application software, *J. Soft. Pract. Eng. (JSPE)*, **36** (9): 893–921, 2006.

67. S. S. Yau, et al., Automated agent synthesis for situation awareness in service-based systems, *Proc. of 30th Annual Int'l Computer Software and Application Conf. (COMPSAC)*, 2006, pp. 503–510.

68. M. Endsley, and D. Garland, *Situation Awareness, Analysis and Measurement*, Mahwah,NJ: Lawrence Erlbaum Associates, 2000.

69. G. D. Abowd, Software engineering issues for ubiquitous computing, *Proc. 21st Int'l Conf. on Software Engineering*, 1999, pp. 75–84.

70. M. Haahr, R. Cunningham, V. Cahill, Supporting CORBA applications in a mobile environment, *Proc. 5th ACM/IEEE Int'l Conf. on Mobile Computing and Networking (MobiCom)*, 1999, pp. 36–47.

71. A. T. Campbell, et al., The Mobiware Toolkit: Programmable support for adaptive mobile networking, *IEEE Personal Commun.*, **5** (4): 32–43, 1998.

72. M. Roman, et al., A middleware infrastructure for active spaces, *IEEE Pervas. Comput. and S.N. Chuang*, **1** (4): 74–83, 2002.

73. A. T .S. Chan and S. N. Chuang, MobiPADS: A reflective middleware for context-aware computing, *IEEE Trans, Soft. Eng.*, **29** (12): 1072–1085, 2003.

74. Object Management Group, Common object request broker architecture specification v3.03, 2004. Available: http://www.omg.org/cgi-bin/apps/doc?formal/04-03 -12. pdf.

75. A. Murphy, G. Picco, and G.-C. Roman, LIME: A middleware for physical and logical mobility, *Proc. 21st Int'l Conf. on Distributed Computing Systems*, 2001, pp.524–533.

76. IBM Research, TSpaces Project. Available: http://www.almaden.ibm.com/cs/TSpaces/.

77. Microsoft, Windows CE home page. Available: http://msdn2.microsoft.com/en-us/embedded/aa731407.aspx.

78. Wikipedia, Embedded Linux. Available: http://en.wikipedia.org/wi ki/Embedded_Linux.

79. Bell Labs, Plan 9 operating system home page. Available: http://plan9.bell-labs.com/plan9/.

80. S. S. Yau and F. Karim, A context-sensitive middleware-based approach to dynamically integrating mobile devices into computational infrastructures, *J. Parallel Distrib. Comput.*, **64** (2): 301–317, 2004.

81. S. S. Yau, et al., Development and runtime support for situation-aware application software in ubiquitous computing environments, *Proc. 28th Annual Int'l Computer Software and Application Conf. (COMPSAC)*, 2004, pp. 452–457.

82. B. J. Nelson, Context-aware and location systems, Ph.D. thesis, University of Cambridge, 1998 Available: http://www.sigmobile.org/phd/1998/theses/nelson.pdf .

83. D. Caswell and P. Debaty, Creating web representations for places, *Proc. 2nd Int'l Symp. on Handheld and Ubiquitous Computing (HUC2K)*, 2000, pp. 114–126.

84. C. Hess, M. Roman, and R. H. Campbell, Building applications for ubiquitous computing environments, *Proc. Int'l Conf.*

Pervasive Computing, 2002. Available: http://choices.cs.uiuc.edu/gaia.

85. T. J. Lehman et al., Hitting the distributed computing sweet spot with Tspaces, *Comput. Networks*, **35** (4): 457–472, 2001.
86. S. S. Yau, D. Chandrasekar and D. Huang, An adaptive, lightweight and energy-efficient context discovery protocol for ubiquitous computing environments, *Proc. 10th Int'l Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, 2004, pp. 261–267.
87. U. Dayal, Active database systems, *Proc. 3rd Int'l Conf. on Data and Knowledge Bases*, 1988, pp. 150–170.
88. R. Campbell, et al., Towards security and privacy for pervasive computing, *Proc. Int'l Symp. on Software Security*, 2002, pp. 1–15.
89. H. Munirul and S. I. Ahamed, Security in pervasive computing: current status and open issues, *Int'l J. Network Secur.*, **3** (3): 203–214, 2006.
90. P. Bhaskar and S. I. Ahamed, Privacy in pervasive computing and open issues, *Proc. 2nd IEEE Int'l Conf. on Availability, Reliability and Security*, 2007, pp. 147–154.
91. S. I. Ahamed, N. Talukder and M. M. Haque, Privacy challenges in context-sensitive access control for pervasive computing environment, *Proc. 4th Annual Int'l Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS) SPEUCS Workshop*, 2007.
92. A. Gorlach, A. Heinemann and W. W. Terpstra, Survey on location privacy in pervasive computing, in P. Robinson, H. Vogt and W. Wagealla (eds.), *Privacy, Security and Trust within the Context of Pervasive Computing*, Berlin: Springer, 2005, pp. 23–34.
93. M. Gruteser and D. Grunwald, Anonymous usage of location-based services through spatial and temporal cloaking, *Proc. 1st Int'l Conf. on Mobile Systems, Applications, and Services*, 2003, pp. 31–42.
94. B. Gedik and L. Liu, Location privacy in mobile systems: a personalized anonymization model, *Proc. 25th IEEE Int'l Conf. on Distributed Computing Systems*, 2005, pp. 620–629.
95. M. F. Mokbel and C. Y. Chow, The new casper: query processing for location services without compromising privacy, *Proc. 32th Int'l Conf. on Very Large Data Bases*, 2006, pp. 763–774.
96. G. Ghinita, P. Kalnis and S. Skiadopoulos, PRIV'E: Anonymous location-based queries in distributed mobile systems, *Proc. 16th Int'l World Wide Web Conf.*, 2007, pp. 371–389.
97. J. I. Hong and J. A. Landay, An architecture for privacy-sensitive ubiquitous computing, *Proc. 2nd Int'l Conf. on Mobile Systems, Applications, and Services*, 2004, pp. 177–189.
98. C. Y. Chow, M. F. Mokbel and X. Liu, A peer-to-peer spatial cloaking algorithm for anonymous location-based services, *Proc. 14th ACM Int'l Symp. on Geographic Information Systems*, 2006, pp. 171–178.
99. E. Snekkenes, Concepts for personal location privacy policies, *Proc. 3rd ACM Conf. on Electronic Commerce*, 2001, pp. 48–57.
100. M. Duckham and L. Kulik, A formal model of obfuscation and negotiation for location privacy, *Proc. 3rd Int'l Conf. on Pervasive Computing*, 2005, pp. 152–170.
101. T. Kindberg, K. Zhang and N. Shankar, Context authentication using constrained channels, *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 2002, pp. 14–21.
102. D. Balfanz, et al., Talking to strangers: authentication in ad-hoc wireless networks, *Proc. of the Network and Distributed System Security Symp.*, 2002.
103. F. Riccardo, Using entity locations for the analysis of authentication protocols, *Proc. 6th Italian Conf. on Theoretical Computer Science (ICTCS)*, 1998, pp. 9–11.
104. R. Want, et al., The active badge location system, *ACM Trans. Inform. Syst.*, **10**: 91–102, 1999.
105. R. Mayrhofer, H. Gellersen and M. Hazas, Security by spatial reference: using relative positioning to authenticate devices for spontaneous interaction, *Proc. 9th Int'l Conf. on Ubiquitous Computing*, 2007, pp. 199–216.
106. T. Kindberg and K. Zhang, Secure spontaneous devices association, *Proc. 5th Int'l Conf. on Ubiquitous Computing*, 2003, pp. 124–131.
107. T. Kindberg and K. Zhang, Validating and securing spontaneous associations between wireless devices, *Proc. 6th Int'l Conf. on Information Security*, 2003, pp. 44–53.
108. F. Stajano and R. Anderson, The resurrecting duckling: security issues for ad-hoc wireless networks, *Proc. 7th Int'l Workshop on Security Protocols*, 1999, pp. 172–194.
109. L. E. Holmquist, et al., Smart-its friends: a technique for users to easily establish connections between smart artifacts, *Proc. 3rd Int'l Conf. on Ubiquitous Computing*, 2001, pp. 273–291.
110. L. Feeney, B. Ahlgren and A. Westerlund, Spontaneous networking: An application-oriented approach to ad hoc networking, *IEEE Commun. Magazine*, **39** (6): 176–181, 2001.
111. Shared Wireless Access Protocol (Cordless Access) Specification (SWAP-CA), Revision 1.0, The Home RF Technical Committee, 1998.
112. R. Mayrhofer, The candidate key protocol for generating secret shared keys from similar sensor data streams, *Proc. 4th European Workshop on Security and Privacy in Ad-hoc and Sensor Networks (ESAS)*, 2007, pp. 1–15.
113. D. Bichler, et al., Key generation based on acceleration data of shaking processes, *Proc. 9th Int'l Conf. on Ubiquitous Computing*, 2007, pp. 304–317.

Stephen S. Yau
Dazhi Huang
Wei Gao
Yin Yin
Arizona State University
Tempe, Arizona

M

MULTICAST PROTOCOLS AND ALGORITHMS

INTRODUCTION

One way to characterize communication is by the number of parties involved. The traditional communication modes are *unicast*, i.e., one-to-one, and *broadcast*, i.e., one-to-all. Between these two extremes we find *multicast*, the transmission of a message or datastream to an arbitrary set of receivers, i.e., one-to-many. Multicast can be seen as a unifying communication mode, as it is a generalization of both unicast and broadcast. Multicast is examined separately, however, because the specification of receivers as a set introduces features and complications that are not present in traditional unicast and broadcast. A more general term is *multipoint communication*, which implies many-to-many bidirectional data exchange.

The multicast model of communication is ideal for applications where data and control are partitioned over multiple entities. Examples include updating replicated databases, contacting any one of a group of distributed servers of which the composition is unknown (more appropriately termed *anycast*) and interprocess communication between multiple cooperating processes. The prototypical multicast applications, however, are real-time interactive multimedia conferencing and near real-time media distribution to multiple receivers.

Multicast efficiency is a fundamental requirement for the success of many group applications. Selective multicast replaces indiscriminate broadcasting to everyone, reducing the waste of resources caused by transmitting information to all receivers. To be more economical than unicast, multicast must conserve resources via *sharing*: Instead of transmitting information from a sender to each receiver separately, routes to receivers that share links must carry the information only once over each shared link. We can picture a multicast route as a *tree* rooted at the sender with a receiver at each leaf and, possibly, some receivers on internal nodes. This tree must be designed to maximize link sharing and thus minimize resource consumption.

Besides the obvious issue of how to construct multicast routing trees, multicast also raises other issues related to the extension of unicast mechanisms to a multicast context. For example, the sender in a reliable transport protocol can recover from communication errors based on error reports from the receiver. By simply extending this mechanism to multicast, we run the risk of *feedback implosion*, when many receivers send such reports toward the sender, thus swamping the network and the source with control information. In addition to the scalability issues raised by this approach, another issue is how the sender should react when conflicting reports arrive from different receivers.

MULTICAST MODELS

The difference between multicasting and separately unicasting to several destinations is best captured by the Internet host-group model of Cheriton and Deering (1): A host-group is a set of network entities sharing a common identifying multicast address. All group members receive any data packets addressed to this multicast address. The senders have no knowledge of group membership and may or may not belong to the group, corresponding to *closed* or *open* groups, respectively. Multicast messages on the Internet are sent on a best-effort basis, like unicast messages; i.e., they may be reordered, lost, or duplicated.

This definition allows group behavior over time to be unrestricted in multiple dimensions; it may have local (LAN) or global (WAN) membership, be transient or persistent, and have static or dynamic membership. From the sender's point of view, the multicast service interface is identical to unicast; only the address differs. Therefore, it is the network's responsibility to manage the multicast communication, transparently to the users. This extra work compared with unicast is expected to result in a more efficient usage of resources, which is the primary motive for network providers to support multicast in the first place.

The host-group model imposes specific requirements for the implementation of the multicast service. First, there must be a means for routing packets from a sender to all group members, which implies that the network must locate all members of the group and make appropriate routing arrangements without any assistance from the sender. Second, as group membership is dynamic, the network must continuously track membership during a session's lifetime, which may range from short to very long periods of time. Tracking is required both to start forwarding data to new group members and to stop the wasteful transmission of data to members that have left the group. This dynamic nature of multicast groups has a considerable impact on multicast routing.

It should be noted that the Internet host-group model is by no means unique for multicasting. Some applications require delivery of messages addressed to a group to be *atomic*; that is, each message sent to a multicast group must be received by either all receivers in the group or none at all. Atomicity further implies that all received messages must be processed in the same order by all receivers, i.e., that multicasts are also totally ordered. As atomic multicast is complex to implement, it is usually built on top of a simpler multicast facility, such as the one offered by the Internet.

MULTICAST ROUTING ALGORITHMS

Unicast routing attempts to minimize either transmission cost or delay, depending on the metric used for optimization. Although these goals seem different, from an

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

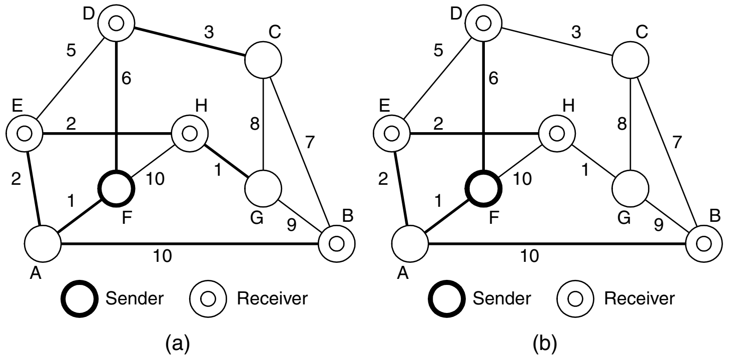


Figure 1. (a) The broadcast tree formed by the shortest paths from the sender to all nodes. (b) The multicast tree obtained by pruning all links that do not lead to receivers from the broadcast tree.

algorithmic point of view, they are both equivalent to finding shortest paths over a network with cost-labeled links; link costs may stand for either transmission cost or delay. A shortest path algorithm finds optimal routes between one node (the sender) and *all* other nodes in the network. Two common examples of such algorithms are those due to Dijkstra and Bellman–Ford. An optimal route minimizes the sum of the costs of all links included in the route. The union of all these routes forms a *shortest path tree* rooted at the sender. As this is a broadcast tree, as shown in Fig. 1(a) a straightforward (but not optimal) solution to the multicast routing problem is to prune off that tree all links that do not lead to any members of the group, as shown in Fig. 1(b).

The advantage of these algorithms is that they are easy to implement and deploy, as they are direct extensions of existing ones. Each path is optimal by definition, regardless of changes in group membership, and this optimality comes essentially for free, because shortest paths need to be computed anyway for unicast routing. The disadvantage of these algorithms is that they concentrate on pairwise optimizations between the sender and each receiver and only conserve resources as a side effect, when paths happen to overlap. For large networks with widely dispersed group members, either the scale of the network or the continuous network changes will necessarily restrict the use of these algorithms to subnetworks, requiring a hierarchical routing technique to support global multicasting.

To achieve the economies promised by multicasting, optimization must be viewed from the perspective of the entire distribution tree. This requires building trees that exploit link sharing as much as possible, duplicating packets only when paths diverge, so as to minimize the total distribution cost even at the expense of serving some receivers over longer paths. In algorithmic terms, what is needed is a minimal cost tree that reaches all receivers, possibly using additional nodes on the way. This is equivalent to the *Steiner tree* problem, analyzed by Hakimi (2). In this problem, a cost-labeled graph and a set of nodes, the Steiner points, are given and a minimal cost tree is sought connecting all Steiner points, including both the sender and all receivers, as shown in Fig. 2(a).

If all nodes were Steiner points, the above problem would coincide with the *spanning tree problem*, which can be solved efficiently. Unfortunately, the Steiner tree problem belongs to the class of NP-complete problems, as shown by Garey et al. (3). Fortunately, approximation algorithms exist for it with proven constant worst-case bounds and very good average behavior. As an example, trees built with the heuristic by Kou et al. (4) have at most twice the cost of Steiner trees, whereas simulations of realistic network topologies described by Kabada and Jaffe (5) have shown their cost to be within 5% of the optimum.

The advantage of Steiner tree algorithms is their overall optimality with respect to a single cost metric, such as transmission cost. Their disadvantages are also important, however: These algorithms must be run in addition to the unicast routing algorithm, and they suffer from scaling problems for large networks. Furthermore, optimality is generally lost after changes in group membership and

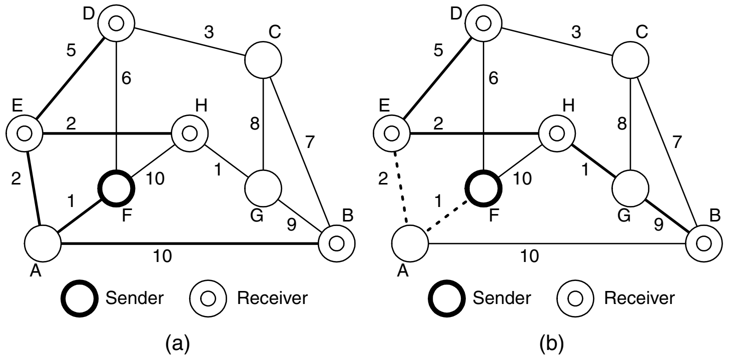


Figure 2. (a) The Steiner tree obtained by minimizing the overall cost from the sender to all receivers. (b) The core-based tree formed by the shortest paths from the core (node H) to all receivers. The sender uses the shortest path to the core for data transmission.

network reconfigurations, unless the tree is repeatedly recomputed from scratch. Approaches for extending these algorithms to deal with changes in group membership without tree recomputation include extending the existing tree in the cheapest way possible to support new group members and pruning redundant links when group members depart. The quality of the tree will deteriorate over time after several such modifications, eventually leading to the need for tree recomputation. Thus, Steiner tree algorithms are best suited to static or slowly changing environments because changes eventually lead to expensive tree recomputation to regain optimality.

Shortest path trees and Steiner trees are optimal with respect to a sender; therefore a separate tree must be built for each sender in both cases. A different approach is to employ a *center-based tree*, which, instead of being rooted at the sender, is rooted at the topological center-of the receivers. A single center-based tree serves as a common infrastructure for all senders; therefore, maintenance of the tree is greatly simplified and nodes belonging to the tree need only maintain state for one shared tree rather than for many source-rooted trees. Even though such a tree may not be optimal for any one sender, it may be an adequate approximation for all of them together. Unfortunately, the topological center of the receivers, apart from being hard to find (this problem is also NP-complete), is not even permanent in a dynamic multicast environment.

A more practical proposal is to abandon the topological center as the root of the tree, keeping the basic idea of a single shared multicast tree for all senders to a group. In this approach, routing is performed by defining one or more arbitrarily selected *core* (or *rendez-vous*) points to serve as the basis for tree construction for a group. All senders transmit their data to the core using an optimal (in the unicast sense) route, and the core uses a shortest path tree to distribute these data to all group members, as shown in Fig. 2(b). As in any shortest path tree, merging of paths is exploited whenever possible, but it is not an explicit goal of the routing calculations. Due to the concentration of paths around the core, though, common paths are expected to arise. Although this approach uses an underlying unicast routing algorithm, it is independent of it.

The disadvantage of this approach is that a single shared multicast tree, especially if it is rooted at an arbitrary node, is not optimal in any strict sense. The advantages of shared multicast trees are numerous, however. First, the shared tree means that this approach scales well in terms of maintenance costs as the number of senders increases. Although there is still a tree emanating from each sender, these trees merge near the core and the distribution mesh is common from there on. Second, the trees can be made efficient by choosing appropriately the core points. Third, routing is performed independently for each sender and receiver, with entering and departing receivers influencing only their own path to the core points of the shared tree. This last property means that network and group membership dynamics can be dealt with without global recomputation. Finally, the independence from specific unicast routing schemes, coupled with the scalability of the shared trees, makes this approach ideal for use on large networks. The core points may even be selected to facilitate hierarchical routing; i.e., a top-level tree can distribute data to the core point of each subnetwork, and each core point can then distribute data to the group members in its subnetwork.

Despite the differences between the approaches discussed above, simulations have shown that even simple multicast routing using shortest path trees is not significantly worse in terms of total tree cost from the optimal solutions. For realistic network topologies, Doar and Leslie (6) have found that the cost of a shortest path tree is less than 50% larger than that of a near-optimal heuristic tree, whereas path delays for heuristic trees are 30% to 70% larger than shortest path delays. As shortest path trees are easily built and modified using the underlying unicast routing algorithm and they never deteriorate in terms of delay, but simply vary in their inefficiency in terms of total cost, an application prepared to accept this overhead can avoid special multicast tree construction and maintenance methods by simply employing the shortest paths.

A similar cost versus simplicity tradeoff is involved when using shared trees, for all senders to a group. With shared trees, optimality is hard to achieve and even harder to maintain; a simple approach is to choose the best core point among group members only. With this limitation, when path delay is optimized, simulations show that delays are close to 20% larger than with shortest paths, and tree cost is about 10% lower than that of shortest path trees. Furthermore, even though a single tree minimizes state and maintenance overhead, this approach suffers from traffic concentration, exactly due to the single tree used, because it routes data from all senders through the same links around the core. Simulations show that delay-optimal member-centered shared trees can cause maximum link loads to be up to 30% larger than in a shortest path tree.

FEEDBACK CONTROL

When the basic service offered by the network is a best-effort one, as in the Internet, generalizing it for multicast is straightforward: Just send the data along the multicast routing tree without providing any guarantees with respect to reliability, throughput, or delay. Many applications, however, cannot be satisfied by such a service; therefore, mechanisms such as flow, congestion, and error control have to be provided on top of this best-effort service. These mechanisms depend on feedback to the sender, which is based on either network- or receiver-generated reports.

Error control ensures that packets transmitted by the sender are received correctly. Packets may be received corrupted (detected by error-detection codes), or they may be lost (detected by missing sequence numbers). Flow control assures that the sender does not swamp the receiver with data that cannot be consumed in time. Congestion control limits the transmission rate of the sender to avoid overloading the intermediate network nodes on the way to the receiver. Although error and flow control require feedback from the receiver, congestion control would be best served by feedback from the intermediate nodes themselves. In best-effort networks like the Internet, however, it is only the receivers that provide feedback about packet losses to the sender, thus leading to confusion between error-induced and congestion-induced losses.

In the unicast case, flow, error, and congestion control rely on feedback from a unique receiver. For example, loss reports may cause the retransmission of lost packets. With multicast, however, this approach faces the feedback-implosion problem: If all receivers respond with status information, they will swamp the sender with, possibly conflicting, reports. Ideally, senders would like to deal with the multicast group as a whole, not on an individual receiver basis, following the host-group model. The sender cannot simply treat all receivers identically, though, because this requires either ignoring the feedback of some receivers or wasting resources by satisfying the worst-case receivers. For example, the sender could retransmit only packets lost by all receivers, thus ignoring some losses, or retransmit any packets lost by any receiver, thus duplicating some packets.

As there is no evident solution to this problem, several approaches exist emphasizing different goals. The simplest approach is to ignore the problem at the network and simply provide a best-effort service. Delegating the resolution of these problems to higher layers may be an adequate solution in many cases, because these layers may have additional information about application requirements and be able to implement more appropriate mechanisms than what is possible inside the network. Even in this case, though, higher layers will have to implement one of the alternative approaches discussed below.

A second solution sacrifices the host-group model's simplicity by keeping per-receiver state at the sender during multicasts. After transmitting a multicast packet, the sender waits until a stable state is reached before sending the next one. For example, in error control, retransmissions may be made until all receivers receive the data. To economize on resources, retransmissions may be multicast when many receivers lose a packet, or unicast when few do. To reduce the risk of feedback implosion, receivers should use negative rather than positive acknowledgments, i.e., send responses only when problems occur, rather than to confirm that packets were received correctly. Furthermore, these negative acknowledgments may be multicast to all receivers after waiting for a random period of time, so as to suppress identical negative acknowledgments from multiple receivers, as suggested by Towsley et al. (7).

Even with these optimizations, the scalability of such schemes is doubtful for large and widely dispersed groups, even when errors, overflows, and congestion are very rare, because the sender remains solely in charge of all receivers. In addition, with these schemes the service provided to a group member is the lowest common denominator, which may be the slowest or most overloaded receiver, or the slowest or most congested link. While more sophisticated variations of this approach exist, their complexity and inefficiency makes them appropriate only for specific applications.

A third solution is to distribute the feedback control mechanism over the entire multicast tree, so as to avoid propagating the receiver's feedback all the way to the sender. In a hierarchical scheme, the intermediate nodes may either respond directly to feedback from downstream receivers or merge their feedback into a summary message and recursively propagate it upstream. If the added complexity of making local decisions on each node (not only group members) is acceptable, this approach narrows down the impact of problems to specific parts of the tree, relieving the sender from dealing with individual receivers. Note that even though this scheme avoids feedback implosion, the problem of dealing with possibly conflicting requests remains.

An alternative non-hierarchical method for distributed feedback control, targeted especially to error control, is to let all receivers and senders cooperate in handling losses, as proposed by Floyd et al. (8). When receivers discover a loss, they multicast a retransmission request, and anyone that has that message can multicast it again. Both requests and replies can have local scope, if the network supports it, so as to avoid burdening the entire group. To avoid feedback implosion, these requests and replies are sent after a fixed delay based on the distance from the source of the message or the request, respectively, plus a randomized delay. The result is that most duplicate requests and replies are suppressed by the reception of the first one. By varying the random delays, the desired balance between recovery delay and duplicates can be achieved, and in contrast to hierarchical schemes, only group members participate in recovery.

A fourth solution is for the sender to act based on an estimation of the average conditions across the group. A scalable feedback mechanism for this estimation has been proposed by Bolot et al. (9): It first estimates the number of receivers in a group and then what the average quality of reception is, using probabilistic techniques. This method can be used to detect congestion problems and adapt the transmission rate (to relieve congestion) or the error redundancy factor (to increase the chances of error recovery). In a refinement of this approach, proposed by Cheung and Ammar (10), the sender splits the receivers into groups according to their capabilities and only sends to each group the data that it can handle. This scheme prevents very fast or very slow receivers from dragging the whole group toward one extreme case.

Finally, another approach (mostly orthogonal to the above) tries to minimize the need for feedback by taking preventive rather than corrective action. For error control, this is achieved by using forward error correction (FEC) rather than error detection codes and retransmissions. For flow and congestion control, this is achieved by reserving resources in advance so that both receivers and intermediate nodes can support the sender's data rate. Although FEC imposes considerable processing and transmission overhead, it requires no additional network mechanisms. Resource reservations, however, require additional network mechanisms to set up and maintain the resources for each session.

MULTIMEDIA MULTICASTING

A common use of multicasting is for multimedia communication, i.e., the exchange of multiple interdependent media types, such as text, audio, and video. As continuous media, i.e., audio and video, require considerable transmission bandwidth, the economies promised by multicasting are especially attractive in this context. The issues arising when multimedia are combined with multicasting are treated more extensively by Pasquale et al. (11).

Host and Network Heterogeneity

Several representational formats exist for each media type, and each participant in a multicast group may support a different set of formats. In unicast, translation is equally effective at either the sender, or the receiver. In multicast, translation at the sender would require the stream to be duplicated and translated for each different type of receiver, preventing link sharing over common paths, placing excessive load on the sender, and requiring the sender to be aware of each receiver's capabilities, thus violating the host-group model. Translation at the receiver is the most economical and scalable approach in this case, because it fully exploits sharing and moves responsibilities away from the sender.

As continuous media impose heavy demands on both networks and hosts, it is likely that not all receivers will be able to receive all of a sender's traffic. This argues in favor of prioritization of the traffic generated through *hierarchical* coding. Hierarchical or *layered* coding techniques decompose a signal into independent or hierarchically dependent components, subsets of which can be used to provide partial reconstruction of the original. Receivers can thus choose only those parts of the media that they can use or are most important to them. For example, a high-resolution component of a video could be dropped from a congested subnetwork, allowing low-resolution components to be received and displayed in that subnetwork, without impacting uncongested subnetworks.

To avoid complicating the host-group model, each component of a hierarchically coded stream may be transmitted to a different multicast group, making the choice of a particular component equivalent to subscribing to the corresponding group. Based on this approach, Vicisano et al. (12) have proposed a purely receiver-driven congestion control scheme, where each receiver estimates the capacity of the network based on packet losses and only subscribes to as many groups as can be realistically delivered by its subnetwork. The sender periodically doubles its transmission rate for each component so as to enable the receivers to decide whether improved network conditions allow the reception of additional media components.

Resource Reservations

For interactive multimedia applications to be practical, the network must be able to provide some type of bandwidth and delay guarantees. If any such guarantees are to be provided, resources must be reserved at the various network nodes traversed. The exact nature of the reservations depends on the required service guarantees and the approach taken toward satisfying them. In any case, the first component of any resource reservation scheme is a specification model for describing flow characteristics; this depends heavily on the model of service guarantees supported by the network. The second component is a protocol for communicating these specifications to the receivers and reserving resources along the transmission path so as to support the requested services.

The simplest unicast approaches to resource reservations are source-based. A setup message containing the flow specification is sent to the destination, with the intermediate nodes committing adequate resources for the connection, if available. Resources are normally overallocated early on in the path, so that even if nodes encountered further along the path are short on resources, connection setup may still succeed. After the setup message reaches its destination, and assuming the connection can be admitted along the path, a response message is returned on the reverse path, allowing intermediate nodes to relax any excessive commitments made on the first pass.

Similarly, for multicast, there must be a way for senders to notify receivers of their properties, so that appropriate reservations may be made. In a homogeneous environment, reservations should be made once on each outgoing link for all downstream receivers, so as to minimize resource usage. Reserved resources may even be shared among multiple senders to the same group. However, receiver and network heterogeneity often prohibits use of this simplistic scheme, because the amount of resources that are available at each part of the multicast tree may be quite different. A modified scheme is to allocate resources as before during the first message's trip and then have all receivers send back their relaxation (or rejection) messages. Each node that acts as a junction only propagates toward the source the most restrictive relaxation among all those received. However, as paths from such junctions toward receivers may have committed more resources than are now needed, additional passes will be required for convergence or resources will be wasted.

An alternative is to abandon reservations during the sender's setup message, instead reserving resources based on the modified specifications returned by the receivers. Again, resource reservations are merged on junction points, but as these requests are expected to be heterogeneous, each junction will reserve adequate resources for the most demanding receiver and reuse them to support the less demanding ones. This approach supports both heterogeneous requests and resource conservation, thus maximizing the possibility for a new session to be admitted. As this mechanism converges in one pass, the reservation state in the switches can be periodically refreshed, turning the fixed state of a static connection into adaptive state suitable for a dynamic environment. Therefore, this mechanism can accommodate both group membership changes and routing modifications without involving the sender.

Quality-of-Service Routing

When multicast is used for multimedia communications, link sharing can lead to considerable economies in transmission bandwidth, but routing must also take into account two additional factors: delay constraints, particularly for interactive applications, and media heterogeneity. Separate handling of media streams allows using the most effective coding technique for each stream. The question arises then of whether the same or separate distribution trees should be used for each stream. Considering the load that continuous media put on network links, separate trees seem preferable. Thus, each media stream could ask for the appropriate quality-of-service (QoS) parameters and get routed accordingly, with receivers choosing to participate in any subset of these trees. On the other hand, the management overhead of multiple trees per source may be prohibitive, whereas routing each media stream separately may complicate inter-media synchronization.

Turning to delay constraints, assuming that we use delay as the link cost during routing, we already saw that the shortest path tree and the Steiner tree are different: The former minimizes individual path delays, whereas the latter minimizes overall distribution delay and maximizes link sharing. As the global tree metric and the individual receiver-oriented metrics are potentially in conflict, we cannot hope to optimize both. We can, however, try to optimize the global metric subject to the constraint that the individual metrics are tolerable. As interactive applications can be characterized by upper bounds on end-to-end delay, it is reasonable to design the tree to optimize total cost while keeping individual paths within some bound. Normally, all receivers are satisfied by the same delay bound, as this is determined by human perception properties.

This problem is essentially a version of the Steiner tree problem with additional constraints on the paths. Even though it is also NP-complete, fast heuristic algorithms that are nearly optimal have been developed, for example, by Kompella et al. (13). Almost identical formulations are obtained when the constraints are delay *jitter*, i.e., the variation of delay, or a probabilistic reliability constraint. For example, in the case of independent link losses, a loss probability can be assigned to each link. By using logarithms, the reliability metric can be calculated in linear form between a source and each destination by adding the logarithms along the path. Thus, the problem reverts to tree cost minimization with a constraint on an additive path-based metric. Finally, the constraint may be a link capacity that must not be exceeded. Again, heuristic algorithms exist to solve this variant of the problem.

MULTICAST PROTOCOLS ON THE INTERNET

The IP Multicasting Model

The Internet, due to its open architecture, has been extensively used as a testbed for multicast algorithms and protocols. IP multicasting is based on special (class D) multicast IP addresses. By simply using a class D address as the destination of a *datagram*, i.e., an IP packet, it is multicast to all group members rather than unicast. To achieve multicasting in a wide-area network, such as the Internet, a mechanism is needed to keep track of the dynamic membership of each group and another mechanism is needed to route multicast datagrams from a sender to these group members without unnecessary duplication of traffic. IP multicasting implements these mechanisms in two parts: Local mechanisms track group membership and deliver multicasts to group members within a local network, and global mechanisms route datagrams between local networks.

In each local network, at least one router acts as a multicast router. A multicast router keeps track of local group membership and is responsible for forwarding multicast originating from its local network toward other networks, as well as for delivering multicasts originating elsewhere to the local network. The delivery of multicast datagrams to local receivers, as well as the reception of local multicasts by the router for subsequent propagation to other networks, depend on the underlying network technology. Therefore, the information needed within the local network regarding group membership in order to achieve multicast delivery may vary.

In contrast, cooperation among multicast routers for the delivery of multicast datagrams between networks is based on a network-independent interface between each network and the outside world. The information needed to decide whether multicasts should be delivered to target networks is whether at least one group member for a destination group is present there, regardless of the information the multicast router needs for local purposes. A multicast router uses the list of groups present on its attached local networks along with information exchanged with its neighboring routers to support wide-area multicasting. Based on this interface, alternative algorithms can be used for global routing without affecting local mechanisms. Conversely, as long as this interface is provided by the local mechanisms, they can be modified without affecting global routing.

Global Mechanisms

A variety of global, wide-area, multicast routing mechanisms exist. The earliest one, proposed by Deering and Cheriton (14), is the distance vector multicast routing protocol (DVMRP). The original version of DVMRP is a variant of the truncated reverse path broadcasting algorithm. Routers construct distribution trees for each source sending to a group, so that datagrams from the source (root) are duplicated only when tree branches diverge toward destination networks (leaves). To construct the tree, each router identifies the first link on the shortest path from itself to the source, i.e., on the shortest *reverse* path, using the Bellman–Ford distance vector unicast routing algorithm. Datagrams arriving from this link are forwarded toward downstream multicast routers, i.e., those routers that depend on the current one for multicasts from that source. A broadcast distribution tree is thus formed, with datagrams reaching all routers. As each router knows which groups are present in its local networks, redundant datagrams are not forwarded there and the tree is truncated at the lowest level. The latest version of DVMRP implements the improved reverse path multicasting algorithm, where links leading to networks with no members for a group are pruned off the tree and are grafted back when members appear for these groups. Although initially all data are broadcast, eventually the tree becomes a real multicasting one.

Another protocol proposed by Moy (15), multicast open shortest path first (MOSPF), extends Dijkstra's link state unicast routing algorithm. Routers flood their membership lists among them, so that each one has complete topological information concerning group membership. Shortest path multicast distribution trees from a source to all destinations are computed on demand as datagrams arrive. These trees are real multicast ones, but the flooding algorithm used to construct them introduces considerable overhead.

A radically different approach is the core-based tree (CBT) protocol proposed by Ballardie et al. (16), which employs a single tree for each group, shared among all sources. This tree is rooted on an arbitrarily chosen router, the *core*, and extends to all networks containing group members. It is constructed from leaf network routers toward the core as group members appear; thus, it is composed of shortest reverse paths. Sending to the group is accomplished by sending toward the core; when the datagram reaches any router on the tree, it is relayed toward tree leaves. Routing is thus a two-stage process that can be suboptimal, as datagrams may be sent away from the receivers during the first stage.

As both shortest path trees and center-based trees have advantages and disadvantages, the protocol independent multicast (PIM) protocol proposed by Deering et al. (17) provides both. PIM supports two modes, the dense mode and the sparse mode. The dense mode is similar to DVMRP but independent of the underlying unicast routing algorithm. The sparse mode starts similarly to CBT, constructing a shared tree for all receivers using a core, called a *rendez-vous* point in PIM, but it allows paths to individual receivers to be switched to shortest delay ones upon receiver request.

Networks supporting IP multicasting may be separated by multicast unaware routers. To interconnect such networks, tunnels are used, i.e., virtual links between two endpoints, composed of a, possibly varying, sequence of physical links. Multicasts are relayed between routers by encapsulating multicast datagrams within unicast datagrams at the sending end of the tunnel and decapsulating them at the other end. Multicast routers may choose to forward through the tunnels only datagrams that have time-to-live (TTL) values above a threshold, so as to limit multicast propagation across networks.

For unicast routing to scale, the Internet is divided into autonomous systems (AS), i.e., areas that internally run a single routing protocol, probably different for each AS. The border routers of each AS run a common routing protocol to achieve global unicast routing. Similarly, although multicast routing within an AS can use any of the above protocols, a common protocol, such as the border gateway multicast protocol (BGMP) proposed by Kumar et al. (18), must be used among the border routers of each AS to achieve global routing. BGMP constructs shared trees between those ASs containing group members, using as the core the AS where the multicast group was originally created, thus bridging the multicast routing protocols used within each AS.

Local Mechanisms

Unlike global mechanisms, only a single set of local mechanisms exists. These local multicasting and group management mechanisms are based on shared-medium broadcast-based networks, such as Ethernet. Delivery is straightforward on such networks, because each host can listen to all messages and select only those with the appropriate addresses. As an optimization, class D IP addresses may be mapped, if possible, to native multicast addresses so as to filter datagrams in hardware rather than in software.

On these networks, multicasts with local scope do not require any intervention by the multicast router, whereas externally originated multicasts are directly delivered to the local network by the router. The router monitors all multicast transmissions on the local network so that it may forward to the outside world those for which receivers exist elsewhere. The router does not need to track individual group members; the only information needed to decide whether an externally originated multicast must be delivered to the local network is whether at least one group member exists in the network. Therefore, the multicast router only requires a local group membership list.

The Internet group management protocol (IGMP) provides a mechanism for group management well suited to broadcast networks, because only group presence or absence is tracked for each group. In the original version of IGMP, the multicast router periodically sends a query message to a multicast address to which all local receivers listento. Each host, on reception of the query, schedules a reply to be sent, after a random delay, for each group in which it participates. Replies are sent to the address for the group being reported, so that the first reply will be heard by all group members and suppress their transmissions. The multicast router monitors all multicast addresses, updating its membership list after receiving each reply. If no reply is received for a previously present group for several queries, the group is assumed absent. When a host joins a group, it sends several unsolicited reports to reduce join latency if it is the first local member of the group. No explicit action is required when a host leaves a group, as group presence eventually times out.

During the time interval between the last host leaving a group and the router stopping multicast delivery for that group, called the *leave latency*, local transmissions to the group are wasted. To reduce this phenomenon, in the latest version of IGMP a host must send a leave message when abandoning a group if it was the last host to send a report for that group. As this last report may have suppressed other reports, the router must explicitly probe for other group members by sending a group-specific query to trigger membership reports for this group. The router can only assume the group absent if no reports arrive after several such queries. Group-specific queries may use much shorter response intervals than general queries, so as to minimize leave latency.

For networks consisting of point-to-point links, such as dialup links between home users and their Internet Service Providers, only a single member per multicast group can exist at the end-user side. An extension to IGMP for such networks, proposed by Xylomenos and Polyzos (19), uses only explicit join and leave messages from the end-user side to the multicast router, thus reducing both join and leave latency, as well as avoiding periodic queries and reports.

Related Protocols

In addition to protocols ensuring multicast delivery, many other protocols, directly or indirectly related to multicast, exist on the Internet. The multicast address-set claim (MASC) protocol, proposed by Kumar et al. (18), allows the entire range of class D IP multicast addresses to be distributed between ASs, so as to avoid addressing conflicts when multicast groups originating in different networks choose the same address.

The session announcement protocol (SAP) is used to announce the existence of multicast sessions along with their addresses, so that interested receivers may join the group. The session description protocol (SDP) describes the media formats comprising a session so that interested receivers will know what to expect after joining the group.

If the receivers require QoS guarantees, they may use the resource reservation protocol (RSVP) by Zhang et al. (20), to signal their requirements to the sender. When RSVP requests from multiple receivers meet on the way to the sender, they are merged into a single reservation that satisfies the most demanding request. As RSVP reservations are periodically refreshed, dynamic reservation modifications and network reconfigurations are supported.

Unresolved Issues

Although multicasting is widely considered to be a valuable service, it is still not universally supported over the Internet. Many explanations for this phenomenon are proposed by Diot et al. (21), including security and scalability issues. Although the traditional security issues raised by unicast, such as data confidentiality and integrity, are also valid for multicast, in the multicast context, they are more difficult to address. For example, secure group communication can be provided by using independent end-to-end secure unicast channels between all pairs of participants, albeit by negating the link sharing advantages of multicast. In the host-group model adopted by the Internet, group membership is unknown to the sender; therefore, it is impossible to set up security associations between the sender and the receivers without tracking additional information.

Another issue that became apparent when multicast started becoming popular is that the amount of forwarding state required at each multicast router for global multicasting does not scale well, because separate entries are needed for every multicast group, even if a single tree is used for delivery. If shortest path trees are used instead, the number of forwarding entries must be multiplied by the number of senders to the group. In unicast routing, this problem is solved by aggregating the forwarding state based on the fact that networks with similar unicast IP addresses are usually geographically close; therefore, routers only need a single aggregate entry for many different IP addresses, pointing in the appropriate direction. Unfortunately, multicast groups may have members everywhere on the Internet; therefore, this type of aggregation is not generally possible. Various other aggregation methods have been proposed, as described by Zhang and Mouftah (22).

BIBLIOGRAPHY

1. D. R. Cheriton and S. E. Deering, Host groups: A multicast extension for datagram internetworks, *Proc. of the Data Communications Symposium*, Vol. **9**, 1985, pp. 172–179.
2. S. L. Hakimi, Steiner's problem in graphs and its implications, *Networks*, **1**: 113–133, 1971.
3. M. R. Garey, R. L. Graham, and D. S. Johnson, The complexity of computing Steiner minimal trees, *SIAM J. on Appl. Math.*, **34**: 477–95, 1978.
4. L. Kou, G. Markowsky, and L. Berman, A fast algorithm for Steiner trees, *Acta Informatica*, **15**: 141–145, 1981.
5. B. K. Kabada and J. M. Jaffe, Routing to multiple destinations in computer networks, *IEEE Trans. on Commun.*, **31**: 343–351, 1983.
6. M. Doar and I. Leslie, How bad is naive multicast routing? *Proc. of the IEEE INFOCOM*, Vol. **12**, 1993, pp. 82–89.
7. D. Towsley, J. Kurose, and S. Pingali, A comparison of sender-initiated and receiver-initiated reliable multicast protocols, *IEEE J. Select. Areas Commun.*, **15**: 398–406, 1997.
8. S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang, A reliable multicast framework for light-weight sessions and application level framing, *Comput. Commun. Rev.*, **25** (4): 342–356, 1995.
9. J. C. Bolot, T. Turletti, and I. Wakeman, Scalable feedback control for multicast video distribution in the Internet, *Comput. Commun. Rev.*, **24** (4): 58–67, 1994.
10. S. Y. Cheung and M. H. Ammar, Using destination set grouping to improve the performance of window-controlled multipoint connections, *Comput. Commun.*, **19**: 723–736, 1996.
11. J. C. Pasquale, G. C. Polyzos, and G. Xylomenos, The multimedia multicast problem, *Multimedia Syst.*, **6** (1): 43–59, 1998.
12. L. Vicisano, J. Crowcroft, and L. Rizzo, TCP-like congestion control for layered multicast data transfer, *Proc. of the IEEE INFOCOM*, Vol **17**, 1993, pp. 996–1003.
13. V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, Multicast routing for multimedia communication, *IEEE/ACM Trans. Networking*, **1** (3): 286–292, 1993.
14. S. E. Deering and D. R. Cheriton, Multicast routing in datagram internetworks and extended LANs, *ACM Trans. Comput. Syst.*, **8** (2): 85–110, 1990.
15. J. Moy, Multicast routing extensions for OSPF, *Commun. ACM*, **37** (8): 61–66, 1994.
16. A. Ballardie, J. Crowcroft, and P. Francis, Core Based Trees (CBT) — An architecture for scalable inter-domain multicast routing, *Comput. Commun. Rev.*, **23** (4): 85–95, 1993.
17. S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, The PIM architecture for wide-area multicast routing, *IEEE/ACM Trans. Networking*, **4**: 153–162, 1996.

18. S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin and M. Handley, The MASC/BGMP architecture for interdomain multicast routing, *Comput. Commun. Rev.*, **28** (4): 93–104, 1994.

19. G. Xylomenos and G. C. Polyzos, IP multicast group management for point-to-point local distribution, *Comput. Commun.*, **21** (18), 1645–1654, 1998.

20. L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, RSVP: A new resource ReSerVation Protocol, *IEEE Network*, **7** (5): 8–18, 1993.

21. C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, Deployment issues for the IP multicast service and architecture, *IEEE Network*, **14** (1): 78–88, 2000.

22. B. Zhang and H. T. Mouftah, Forwarding state scalability for multicast provisioning in IP networks, *IEEE Commun.*, **41** (6): 46–51, 2003.

George C. Polyzos
George Xylomenos
Athens University of Economics
and Business
Athens, Greece

NETWORK FLOW AND CONGESTION CONTROL

INTRODUCTION

Communication networks are a conglomeration of processing units of so-called *nodes* that are connected via communication *links*. The processing units can be computers, mobile phones, routers, sensors, and so on, that have varying capabilities and limitations in the available memory (buffer) space, processing speed, and transmission rates. Similarly, the communication links can have varying reliability characteristics and capacities, and they can be classified as wireless or wireline (e.g., coaxial and fiber-optic cables). Although a wireless connection allows for mobility, its supported rate is smaller than what can be achieved by, say, a fiber-optic cable. Examples of communication networks include cellular networks, ad hoc wireless networks, the Internet, the Public Service Telephone Network, and sensor networks.

Typically, numerous applications are served by communication networks simultaneously. These applications vary in their characteristics and demands. For example, real-time traffic (also called *inelastic* traffic), such as a voice call or a video transmission, has very stringent delay characteristics and fixed-rate requirements. Such traffic does not benefit from a rate that is greater than its required amount. In comparison, *elastic* traffic such as a file download is tolerant to delays but prefers the highest possible average rate. In the remainder of this article, we will use the terms *congestion control* and *flow control* interchangeably.[1] Also, we will call each flow associated with an application *a session*. Different sessions will have differing quality-of-service (QoS) constraints associated with them based on their demands. The following subsections discuss the circumstances that motivate the need for flow control.

Buffer Overflows and Delay

Because the resources of a network are shared by all the sessions that coexist, the performance—in throughput, delay, energy dissipation, and so on—experienced by each session is dependent on the behavior of the other sessions with which it has to compete. When the network resources are insufficient to support all the sessions satisfactorily, the interdependence among the performance of the sessions may hurt the performance of *all* the sessions that share the same resources. This situation is demonstrated in the following example (see Ref. 1 for a deeper analysis).

Example 1. Consider the scenario depicted in Fig. 1 with two sessions, one from S_1 to D_1 and the other from S_2 to D_2. The capacities of the links are given in the figure in bits per second (bps). The relay node R maintains a single finite buffer of size B to serve both of the sessions that traverse it. We assume that R can send packets over each of its outgoing links independently (i.e., it can send Session-1 packets to D_1 and Session-2 packets to D_2 simultaneously). However, the total number of packets that R can hold is limited by B. Consider the scenario when the rate of Session-2 is fixed at a rate slightly less than 1 bps, and the rate of Session-1 is variable, denoted by λ bps. When λ is less than 1, the buffer at R is lightly loaded and the total throughput is approximately $1+\lambda$ bps, which can be as high as almost 2. But, as λ exceeds 1, the buffer is overloaded and starts dropping packets. Those packets that are dropped need to be retransmitted to R. In such a scenario, S_1 and S_2 will be almost always busy sending new as well as dropped packets. But because the link capacity between S_1 and R is four times the capacity between S_2 and R, Session-1 has four times the chance of capturing a vacancy at the buffer than Session-2. Thus, the throughput of Session-1 will be four times the throughput of Session-2. Because the throughput of the first session can be at most 1 bps (because the capacity of the link from R to D_1 is 1), the throughput of the second session can be at most 0.25 bps, which yields a total throughput 1.25 bps. Thus, the total throughput drops to approximately 1.25 bps from approximately 2 bps because Session-1 overflows the buffer, and Session-2 cannot fully use the link from R to D_2 because of the packet drops. ◇

This example shows that the overloaded system may perform more poorly than a moderately loaded system. Flow/Congestion Control is a mechanism that, either proactively or adaptively, regulates the traffic load on the network to prevent performance degradations caused by overloading of the system.

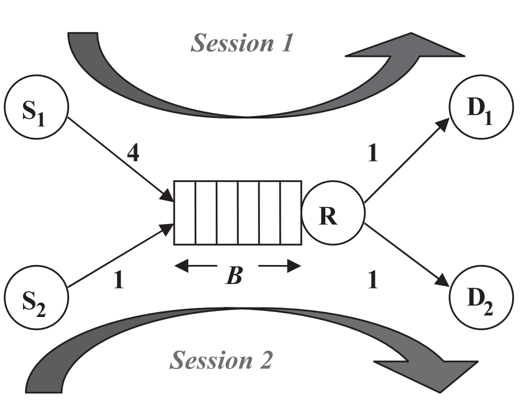


Figure 1. The scenario investigated in Example 1 that demonstrates the throughput performance degradation caused by overloading.

[1] To distinguish, the term congestion control may used to identify mechanisms that specifically aim to prevent congestion in the network, whereas the term flow control may be used to achieve other goals such as throughput, delay, fairness, or other quality-of-service constraints. We will avoid such subtle differences in this article and use these terms interchangeably.

Fairness

Because the network resources are limited, it is necessary to divide the available bandwidth fairly among the competing sessions. In the absence of a fairness mechanism, one session may dominate other and exhaust the resources of the network, leaving no service for others. There are various notions of fairness defined in the literature (see Ref. 2 and references therein for an extensive discussion). The following example introduces a canonical scenario to facilitate the discussion of different types of fairness.

Example 2. Three sessions exist as shown in Fig. 2, in which Session-1 traverses both links a and b, and Sessions-2 and 3 traverses link a and link b, respectively. If the capacity of each link is 1 bps, then what is a fair distribution of the service among the flows? We let x_1, x_2, and x_3 denote the rates of the three sessions, and we discuss two types of fairness: *max-min* fairness and *proportional* fairness.

- An allocation is said to be *max-min* fair if the rate of a session cannot be increased without decreasing the rate of another session with a smaller rate (than the one being increased). Thus, this fairness concept aims at the maximization of the minimum rate provided by the network. For the scenario in Fig. 2, a max-min fair allocation would be $x_1 = x_2 = x_3 = 0.5$ bps. Notice that according to this allocation, the total rate is 1.5 bps, although it is possible to achieve a total rate of 2 bps by setting $x_1 = 0$ and letting $x_2 = x_3 = 1$ bps.
- An allocation is said to be *proportionally* fair if for any other feasible allocation, the sum of the proportional changes in each user's rate is less than or equal to zero. Mathematically, $\mathrm{x}^\star = (x_1^\star, x_2^\star, x_3^\star)$ is a proportionally fair allocation if for any other feasible allocation x, we have $\sum_{i=1}^{3}(\frac{x_i - x_i^\star}{x_i^\star}) \leq 0$. A proportionally fair allocation for the scenario in Fig. 2 is $x_1^\star = 1/3$, and $x_2^\star = x_3^\star = 2/3$. Notice that the total rate provided by this allocation is 5/3, which is still smaller than the maximum but larger than the total rate of the max-min fair allocation. ◇

The fairness concept that is most suitable for a given scenario may be different. It is generally a difficult process to choose the most convenient fairness criterion for a given scenario. Once the fairness measure is determined, one of the primary goals of flow control is to achieve fairness among the sessions.

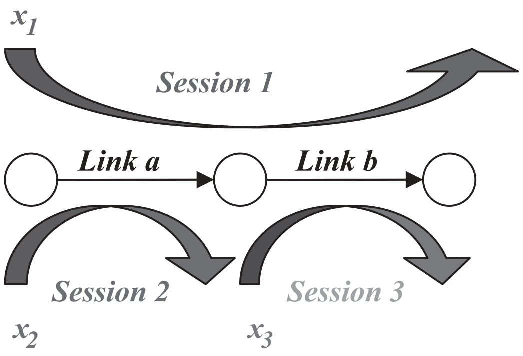


Figure 2. Different fairness goals will lead to different allocations (see Example 2).

Moreover, in a real network, no central authority decides how the resources should be allocated. Therefore, the flow control mechanism needs to operate in a decentralized manner to achieve the fairness goals.

MEANS OF CONGESTION CONTROL

Various techniques and strategies can be used to control congestion in communication networks. In the subsequent subsections, several such techniques are discussed. Some of these techniques are exemplified later in the document.

Buffer Management and Scheduling

It is sometimes useful to separate the buffers associated with different sessions. For example, in Example 1, if two separate buffers are maintained at node R, one for each session traversing it, then no performance degradation would be experienced by the second session because of the high load generated by the first session. Also, buffer management enables differentiated services by prioritizing service according to the importance or urgency of the sessions.

The presence of multiple buffers at a given node necessitates a scheduling mechanism. When packets of different sessions are waiting to be transmitted over a given link, the scheduling mechanism determines the order in which they should be transmitted. Thus, the scheduling mechanism critically affects the performance of the sessions. Numerous cross-layer mechanisms combine scheduling and congestion-control mechanism to achieve optimal performance (e.g., Refs. 3–8). We will return to this topic later in the section entitled "Mathematics of Congestion Control."

The buffer management mechanism may be first-in–first-out, in which packets are served in the order in which they enter the buffer, or some other priority mechanism. For example, in Ref. 9, more generalized processor-sharing mechanisms are investigated.

Packet Dropping/Marking

Another means to congestion control is through dropping and/or marking of packets that overload the network. Such operations may be used to send feedback to the source of the associated session and implicitly trigger it to reduce its traffic generation rate and reduce congestion. For example, in Example 1, if Session-1's packets are marked or dropped even before the buffer at R is full, then Session-2 could maintain its high-throughput level when the network is overloaded. We will observe in a later section how such a mechanism is used in practice.

Call Blocking

Sometimes it might be useful to block those calls that would congest the network if allowed. This strategy may also be thought of as a packet dropping at the session level. Call blocking may be needed if the session has some strict minimum rate requirement such as in real-time applications. In such a case, the call is admitted into the network if the minimum service can be guaranteed. Otherwise it is blocked. We will discuss this strategy more in the next section under the name of admission control.

TAXONOMY FOR CONGESTION/FLOW CONTROL

Various ways are available to classify congestion control mechanisms into broad categories. In the following subsection, we discuss some of these taxonomies.

Reservation-Based and Adaptive

The congestion-control mechanisms can be categorized into two classes: *reservation-based and adaptive*. Reservation-based mechanisms aim to prevent congestion from happening by predicting the effect of incoming sessions on the overall performance, and admitting or rejecting them based on the outcome. Such mechanisms, which are also called *admission-control mechanisms*, can be implemented in various ways (see Ref. 2):

1) *Resource Reservation*. Here, the strategy is to check whether the network has sufficient resources to support the QoS constraints of the incoming session. It aims at reserving resources for the incoming session so that it can be served satisfactorily. If all the links that will be used by the session have the sufficient capacity, then the session is admitted to the network. Otherwise, it is rejected. This strategy requires a fairly good understanding of the dynamics of the networks so that the potential effect of a new session can be accurately predicted. Also, it is less robust to changes in the network topology or capabilities.
2) *Probing*. In the strategy, a set of probing packets are transmitted to estimate the performance that will be experienced by the new session if it is admitted. If the estimated performance is acceptable, then it is admitted. Otherwise, it is rejected.

Admission control provides high performance to the admitted sessions. Typically, admission control mechanisms are used in circuit switched networks. Distributed admission control mechanisms are investigated in the literature (10).

Adaptive mechanisms, on the other hand, observe the congestion level in the network continuously and keep the session rates at an acceptable level. Also called *feedback-based controllers*, such mechanisms are robust to changes in the network conditions and require minimal information about the network topology or traffic characteristics. These qualities render such mechanisms attractive for large and changing networks such as the Internet and wireless networks. However, they may be less attractive for serving real-time traffic because of their dynamic nature.

Window-Based and Rate-Based

Congestion/flow control mechanisms generally determine the rate of flow that a session is allowed to pump into the network. Window and rate-based flow control are two different ways to implement this traffic generation. In this section, we discuss the general mechanism of both of these strategies along with their advantages and disadvantages. More references on this subject can be found in Refs. 1,2,11.

1) *Window-Based Flow Control*. The basic idea behind window-based flow control is to limit the number of packets that a session is allowed to have in the network at a given time. Many different versions of this strategy are available, such as router-centric or end-host-centric versions. The router-centric version performs a node-by-node implementation in which a window is maintained for each hop of a session's route. The end-host-centric version is implemented at the session sources and performs an end-to-end window implementation. Because it is the dominant version implemented in the networks, we describe the window-based flow control for the end-host-centric case.

We describe the operation of the mechanism for one of the sessions, say i. Each Session-i packet that is injected into the network has a packet ID, and traverses a set of intermediate links before it is received by its destination. At the successful reception of each packet, the destination sends a small acknowledgment (ACK) packet with the ID of the received packet back to the source. The duration between the time of injection of a packet into the network by the source and the time of reception of its ACK by the source is called the *round-trip time (RTT)*. RTT captures the queueing delay, processing time, and propagation delay that is experienced by the packets. Of course, this quantity is normally random and is dependent on the congestion level of the network. For the purposes of our discussion however, we will assume that the RTT of Session-i is fixed at T seconds.

We assume that the window size associated with Session-i is fixed to W. Thus, the maximum number of unacknowledged Session-i packets is limited to W. The window-based flow control operates as follows: The source of Session-i keeps track of the number of its packets that are in flight in a counter. It is allowed to inject new packets into the network as long as the counter is less than W. When the counter hits W it stops and awaits ACK(s) from the destination indicating that it has received one or more of the packets. If such ACKs arrive, then the counter is decreased by the number of acknowledged packets and new packets can be transmitted (see Fig. 3). If no ACKs are received within a certain duration, then the corresponding packets are assumed to be lost because of congestion. Then, the window is emptied and the packet transmission restarts with the unacknowledged packets.

Next, we discuss the rate achieved by Session-i for a given W and T. We assume that C gives the capacity that the network can provide to Session-i in packets per second. We can consider C as the maximum rate at which packets can be injected into the network by the source. Although C and T are actually dependent on the presence and rate of other

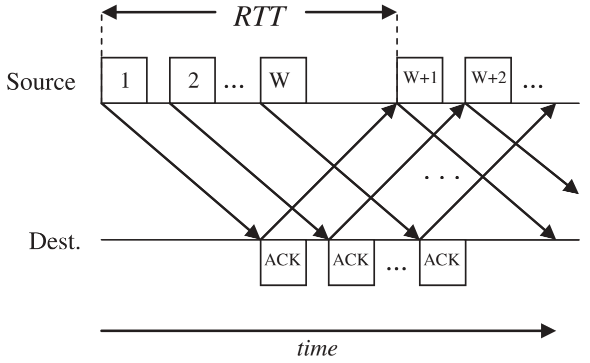


Figure 3. Operation of the end-to-end sliding window with a window size of W.

sessions that share the resources, for the moment, we assume that these values are fixed. When C is sufficiently large, the rate is limited by W and T as follows. Because W packets are in flight at any given time and each of these packets are acknowledged in T seconds, the rate of flow R is given by $R = W/T$. However, when C is smaller than this value, $R = C$. Combining these two arguments, we have that $R = \min(C, W/T)$.

Notice that this expression captures the effect of high RTT on the rate of the session for fixed W. In particular, when RTT is large, the session must have a small data injection rate to keep the congestion level low. For high-speed networks, where C is large, an efficient implementation of the window-based flow control requires W to be also large. This size reduces the reaction time to a potential increase in congestion.

So far, we assumed that the values of C and RTT are fixed. In reality, the choice of W directly affects the value of RTT. Specifically, as the W increases, the queueing delay and hence the RTT may increase dramatically. Therefore, a tradeoff occurs between throughput and delay. One metric to balance them is to consider the throughput/delay ratio, in which the goal would be to keep this ratio high. The following example investigates this metric for a simple network.

Example 3. Consider the single session in Fig. 4 that traverses a single relay node with a service rate C packets/sec. Assume that the queueing delay at the relay in terms of the arrival rate R is $\frac{1}{C-R}$ seconds.[2] Also assume that the processing and return time of the ACKs is fixed at d seconds. Then, the RTT is given by

$$d + \frac{1}{C - R}$$

We are interested in maximizing R/RTT. In Fig. 4, we plot the R/RTT as a function of R/C. We observe that this metric is maximized at a moderate load. This finding is intuitively meaningful because when R is low, the network is under-utilized, whereas when R is close to capacity, the delay is too high.

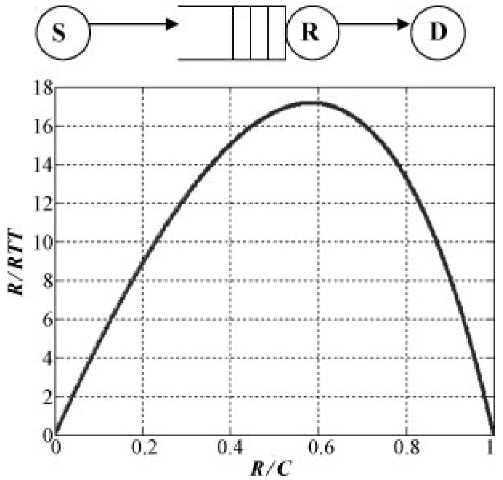


Figure 4. Throughput/delay ratio for the single session system depicted above when $d = 100$ milliseconds and $C = 10$ packet per second. The ratio is maximized at an intermediate level of load as indicated in the graph.

It is also true that when multiple sessions share the resources of a node, the capacity available for one session is influenced by the rate of the other session. Therefore, the window size of a session affects the C available to other flows.

Because both C and RTT are dynamic parameters that depend on the window size of coexisting sessions, it is often necessary to consider *adaptive* window flow mechanisms for congestion control. In fact, most current implementations use such adaptive schemes to achieve congestion control. We will investigate some mechanisms used in the Internet in the next section.

Window-based flow mechanisms react to congestion relatively quickly (within one window duration). In particular, if the congestion level increases, then the packet ACKs are received slowly and the packet injection rate is decreased. However, window-based flow control does not guarantee a minimum rate. Thus, it may be unsuitable for serving fixed-rate traffic, such as voice or video transmissions.

2) *Rate-Based Flow Control*. Another flow-rate regulation mechanism is rate-based flow control. This mechanism differs from the window-based mechanism in that it directly controls the rate of data injection into the network. Such a strategy is attractive for high-speed networks in which a window-based flow control would require extremely large window sizes and hence incur large delay. Also, rate-based control can guarantee minimum transmission rates, and hence it is well suited for serving real-time traffic.

The rate of packet injection of different flows must be determined based on their QoS constraints and the

[2] This is a well-known mean delay formulation of an M/M/1 queue (see Ref. 12).

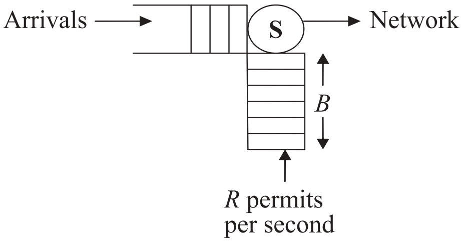


Figure 5. Leaky bucket implementation for rate-based flow control.

limitations of the network resources. For example, for voice traffic, it must be guaranteed that the minimum rate requirements are met, and the rate is no more than what is discernible by the listener.

Once the average rate is determined, say at a value of R packets per second, then rate-based flow control can be implemented in various ways. One way is to allow B packets to be injected into the network every B/R seconds, where B is some fixed parameter. The choice of B determines the amount of burstiness that the mechanism can accommodate.

Another common implementation is the *leaky bucket scheme*, in which a bucket of size B that holds *permits* is maintained at the source. A packet is injected into the network only when a permit is present in the bucket. One permit enters the bucket every $1/R$ seconds as long as the bucket is not full. This way, an average rate of R packets per second can be achieved for the flow.

Proper choice of bucket parameters is critical for the performance of the scheme. For example, if B is too small, then bursty traffic is delayed, whereas if B is too large, then the traffic may cause too much congestion in the network. As in the window-based flow control, rate-based flow control can also be adaptive. In particular, the allowed rate R and/or the bucket size B may be adjusted based on the congestion feedback to react to congestion.

CONGESTION CONTROL IN PRACTICE

In this section, we discuss several practical implementation mechanisms for congestion control in networks. We will focus on the Internet congestion control mechanisms, namely the Transmission Control Protocol (TCP) and its variants, because it is the dominant congestion-control mechanism implemented in today's communication networks.

TCP Flow Control

TCP is an adaptive window-based flow-control mechanism that is implemented in the Internet. It is primarily used to establish reliable, in-sequence delivery of a stream of packets from the source to the destination. A first version was originally developed by Jacobson in the late 1980s (13). In what follows, we describe the main components of Jacobson's algorithm as implemented in TCP, and then we describe several variants of this original version.

The basic idea behind all variants of TCP is to increase/decrease the window size of a session adaptively based on the congestion feedback as measured by delayed and/or lost ACKs. Jacobson's algorithm is composed of two phases of window-size adaptation:

1) *Slow-Start Phase*. The purpose of this phase is to increase the window size quickly to a relatively high level before congestion starts to kick in. In particular, it starts the window with a minimal size (usually 1) and it increments the window size W with every new ACK. For example, on the reception of the first ACK, W becomes 2, after another RTT two ACKs are received and W is increased to 4, and so on. Thus, the size grows to roughly twice its previous value in every RTT rounds. This phase continues until W reaches a threshold value Th. The threshold value Th is set at the beginning of the connection to half the maximum allowable window size. If packets are lost before Th is reached, then Th is set to half the window size at that time, and slow-start is restarted with the new Th value.
2) *Congestion-Avoidance Phase*. During this phase, the congestion level is regulated in the network. It implements an *additive increase/multiplicative decrease* operation to achieve this regulation. In particular, W is incremented by 1 for every W ACKs received. Notice that this increase is much slower than in the slow-start phase, and it is called additive increase. Whenever a packet loss is detected the size of the window is decreased, and different versions of TCP (Tahoe, Reno, NewReno, and SACK) implement different strategies. We will assume a particular multiplicative decrease strategy where Th is set to half the current window size, and the slow-start phase is restarted with the new Th value.

Because of the additive increase and multiplicative decrease nature of the implementation, a typical throughput pattern of TCP is in the form of a sawtooth, where the rate gradually increases up to the point where a packet loss is experiences when an abrupt decrease occurs. Figure 6 depicts the evolution of the window size under a typical TCP implementation. The slow-start phase is in operation between T_0 and T_1 where the window size quickly grows up to a level of W_0. Then the congestion avoidance phase kicks in and increases the window size gradually up to W_1 until time T_2 at what epoch the ACKs of the outstanding packets are no longer received by the source. After a time-out

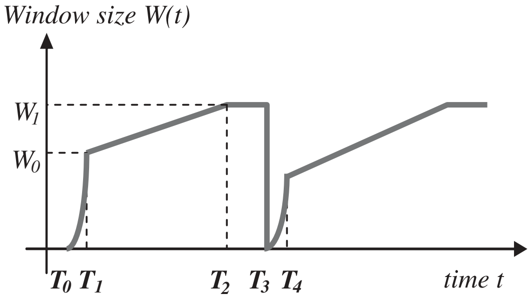


Figure 6. Typical evolution of the window size under TCP.

interval the window size is reduced to 1 at time T_3, and the slow-start phase is restarted with the window size threshold Th set to $W_1/2$. At T_4, this threshold is reached and the congestion avoidance phase is initiated, and so on.

All the variations of TCP described so far use packet losses as an indicator of congestion. Another measure of congestion can be the queueing delay at the links. TCP-Vegas (14) reacts to such indicators. The idea is similar: The source increases its window size when the queueing delay is low, and it decreases its window size when the queueing delay is high. This form of reaction is different in that it adapts the source rate before packet drops start to occur. More detail can be found in Refs. 2 and 14. One attractive feature of TCP-Vegas is that it has been shown to converge to a proportionally fair allocation (cf. Example 2). For a comparison of the performance of different versions of TCP, the reader is referred to Refs. 2,15–18.

Although TCP has been a success in the Internet, some of its basic assumptions render it unsuitable for next generation communication networks. In particular, TCP assumes that packet losses and delays as indicators of congestion. Whereas it is typically true for reliable wireline networks, it is not true for wireless and mobile networks. In a wireless network, packet drops and delays may also occur because of temporary channel fluctuations, even when the congestion level is very low. Thus, TCP performs poorly under wireless fading environments. References 19 and 20 discuss several improvements on TCP to deal with such deficiencies.

Congestion Avoidance Mechanisms

Most TCP implementations in operation require packet drops in their congestion control. This method may not be attractive because of the additional delay incurred because of the retransmission of lost packets. Several congestion avoidance mechanisms are proposed to avoid congestion rather than reacting to it. We discuss some of these in the following section.

1) *Random Early Detection (RED)*. RED is a mechanism introduced in Ref. 21 to inform the sources of potential future congestion in the network by marking or dropping packets based on queue-length levels. Specifically, based on its current average queue-length estimate, each intermediate node randomly decides to mark/drop some of its packets with a probability that is linearly related to the average queue length (see Fig. 7). Thus, when used together with TCP, such marking/dropping implicitly indicates that the intermediate node is becoming heavily congested. TCP reacts to such indicators by dropping its rate and reducing the congestion. RED helps the system reduce its average queue-length levels and hence decrease the queueing delay experienced by the flows.
2) *Explicit Congestion Notification (ECN)*. Another method to indicate congestion is through what is called an ECN mechanism (22–24). Here, a single bit is reserved in every packet as an indicator of congestion. As a packet traverses intermediate nodes, the nodes can set the ECN bit to one if their congestion level is high. For example, RED can use this bit to mark the packets when the congestion level is high. The ECN bit can also be used by other mechanisms such as the mechanisms to be discussed in the next section.

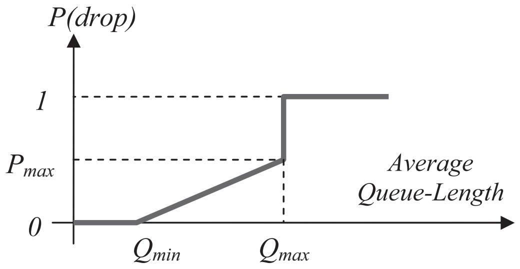


Figure 7. Dropping probability function of RED. The mechanism aims to discourage congestion by dropping packets when the average queue length exceeds $Q_{\min}$ with the probability of drop increasing linearly with the queue-length level.

MATHEMATICS OF CONGESTION CONTROL

Although congestion control mechanisms have been proposed and implemented in practice for decades, their mathematical formulation has been relatively recent. In this section, we will discuss a mathematical model of congestion control for general topology networks that is based on an optimization framework. This line of work is initiated by the seminal work of Kelly et al. (25,26), and then it is extended in subsequent works (e.g., see Refs. 1,27–29; also see Refs. 2 and 30 for excellent overviews).

Utility Maximization Framework

In this framework, the preferences of sessions and the aimed fairness characteristics are captured through a *utility function* formulation. For each session, say i, there exists a concave and strictly increasing function $U_i(x_i)$ that is a function of its mean rate. The concave form of the function reflects the law of diminishing returns, and says that the same increase in the rate is more valuable at a small rate than at a large rate. We assume that each session, say i, has a fixed route $R(i)$ associated with it, and each link, l, in the network has a fixed capacity denoted by c_l. Then, $y_l \triangleq \sum_{i:l \in R(i)} x_i$ gives the total rate of flow over link l. The goal of congestion control can be considered to set the session rates, (x_i), to optimize the following:

$$\begin{array}{lll} \max\limits_{(x_i)\geq 0} & \sum\limits_i U_i(x_i) & \\ \text{subject to} & y_l \leq c_l & \text{for each link } l \end{array} \qquad (1)$$

This model primarily captures the wireline network model with general topologies. It is very general in that it can capture various forms of fairness criteria (such as max-min or proportional fairness) depending on the choice the utility functions (see Ref. 31). For example, when $U_i(x_i) = \log(x_i)$

the solution of Equation (1) is the proportionally fair allocation.

Under the stated assumptions, Equation (1) is a convex optimization problem and various methods can be used to solve it (see Refs. 32 and 33). Amongst these methods is the Lagrangian method that leads to a *decentralized solution*, and hence is most attractive for the network setting. Although the details of the analysis can be found in the literature (e.g., Refs. 7,8,26 and 27), here we discuss the high-level description of the solution.

The operation can best be described through the perspective of two agents: the network and the source. These two agents interact with each other through a *pricing* mechanism. In particular, each link, l, in the network determine a price, p_l, for using its resources based on its current load y_l. The price is high if y_l is close to its capacity c_l, and it is low if y_l is much smaller than c_l. Thus, the price of each link measures its usage at the time. Given a set of link prices (p_l), the price associated with session-i is $p^i \triangleq \sum_{l \in R(i)} p_l$ which gives the total price of the links that session i traverses. p^i can be interpreted as the price that session i has to pay per unit flow of data it sends in the network. Then, the source of session i adjusts its data rate x_i so that the utility it gains from the network minus the cost is maximized. Mathematically, x_i maximizes $(U_i(x_i) - x_i p^i)$ for each session i. This operation is intuitively meaningful because those flows with high utility values will afford to pump more data into the network, whereas those with low utility values will have to reduce their rates, and consequently the total utility will be high. The adjusted set of session rates are then used at each link to update the prices such that the price of the under-utilized links are reduced whereas the overloaded links are increased. When this procedure of price adjustment by the network and rate adjustment at the sources are repeated indefinitely, the rate allocation vector $(x_i)_i$ converges to the solution of Equation (1). A pictorial overview of this iterative strategy is depicted in Fig. 8.

Various open problems require more attention. The assumed concave form of utility functions is well-suited for applications such as file transfer, they do not accurately capture real-time traffic. Several works have addressed the operation under nonconcave utility functions (e.g., Ref. 34). Yet, there is need for capturing QoS constraints of real-time traffic. Also, the question of convergence rate of the iterative algorithm is of interest for both theoretical and practical reasons.

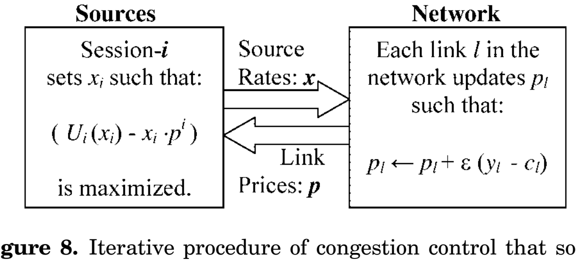

Sources
Network
Session-i sets x_i such that:
$(U_i(x_i) - x_i \cdot p^i)$
is maximized.
Source Rates: $\boldsymbol{x}$
Link Prices: $\boldsymbol{p}$
Each link l in the network updates p_l such that:
$p_l \leftarrow p_l + \varepsilon\,(y_l - c_l)$

Figure 8. Iterative procedure of congestion control that solves Equation (1). In the iteration, ε is a small positive constant that determines the step size of the evolution.

Cross-Layer Design

The utility-based formulation discussed in the previous section can also be used to develop cross-layer controllers that include scheduling and routing decisions in addition to congestion control. Here, the routes of the sessions are not fixed and are to be determined by the controller. Such mechanisms are investigated recently in the literature (e.g., Refs. 3–7,20,35). In what follows, we describe the high-level idea behind these schemes.

Again the operation of the cross-layer mechanism can be separated into two agents: the network and the source. Here, in contrast to the previous algorithm (see Fig. 8), the network is also responsible for the scheduling and routing decisions of packets. Thus, each node must determine which packets to transmit over which of its outgoing links. The routing capability allows the network to use underutilized links in the network, whereas the scheduling capability allows the network to give more priority to sessions that experience higher congestion. The source, as before, is responsible for the congestion control decisions.

The solution uses *buffer management* (cf. the second section) strategy to maintain separate queues at each node for those packets that are destined for the same end node. The lengths of these queues are used by the network and the source to make the scheduling-routing and congestion-control decisions. In particular, the network uses the queue-lengths to give priority to those queues with a high backlog compared with the queues in the neighboring nodes. Also called the *back-pressure* strategy (36), this strategy aims to spread packets of each session throughout the network to establish the most efficient routes. The source uses the queue-length information as the price that must be equalized to the utility to be gained from service. Thus, the cross-layer mechanism is coupled through the occupancy level of the queues maintained at the nodes. This high-level interaction is depicted in Fig. 9.

It is shown that such a scheme will provide the optimal performance (in terms of utility maximization) that any scheduling-routing-congestion control mechanism can provide. It is surprising that a loose coupling through appropriately maintained queue-lengths can provide such optimal behavior. All of these strategies extend to wireless networks (see Ref. 8).

FINAL REMARKS

In this brief overview, we observed that congestion/flow control aims at keeping the load of the network at a moderate level so that certain objectives can be reached. The objectives may be to provide certain QoS guarantees that

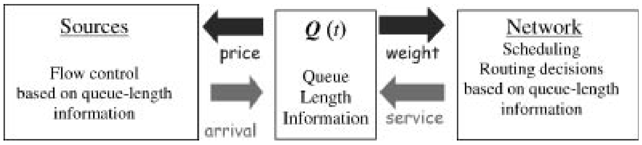

Sources
Flow control based on queue-length information
price
arrival
$Q(t)$
Queue Length Information
weight
service
Network
Scheduling
Routing decisions based on queue-length information

Figure 9. High level interaction between the network and the sources under the cross-layer scheme.

the competing traffic requires and/or to satisfy certain fairness criteria. No congestion control mechanism can achieve all these in the most general sense. We have discussed the most critical issues related to congestion control, as well as some methods and strategies that can be used to achieve the assumed objectives. Although there are some successful practical implementations of congestion control, these require significant improvements to be applicable to next generation high-speedwireless networks. We discussed several recent designs that are based on an optimization framework, and we have pointed out to some of the open problems.

BIBLIOGRAPHY

1. D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice Hall, 1987.
2. R. Srikant, *The Mathematics of Internet Congestion Control*. Boston, MA: Birkhäuser, 2004.
3. X. Lin and N. Shroff, The impact of imperfect scheduling on cross-layer rate control in multihop wireless networks. *Proceedings of IEEE Infocom*, Miami, FL, 2005.
4. A. Eryilmaz and R. Srikant, Fair resource allocation in wireless networks using queue-length based scheduling and congestion control. *Proceedings of IEEE Infocom*, Vol. 3, Miami, FL, 2005, pp. 1794–1803.
5. A. Stolyar, Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm. *Queueing Sys.*, **50**(4): 401–457, 2005.
6. M. J. Neely, E. Modiano, and C. Li, Fairness and optimal stochastic control for heterogeneous networks, *Proceedings of IEEE Infocom*, Miami, FL, 2005, pp. 1723–1734.
7. A. Eryilmaz and R. Srikant, Joint congestion control, routing and mac for stability and fairness in wireless networks. *IEEE J. Selected Areas Communicat., Special Issue on Nonlinear Optimization of Communication Systems*, **14**: 1514–1524, 2006.
8. X. Lin, N. Shroff, and R. Srikant, A tutorial on cross-layer optimization in wireless networks, *IEEE J. Selec. Areas Communicat., Special Issue on Nonlinear Optimiz. Communicat. Sys.*, **14**: 1452–1463, 2006.
9. A. Parekh and R. Gallager, A generalized processor sharing approach to flow control in integrated services networks: The single node case, *IEEE/ACM Trans. Network.*, 1993.
10. F. P. Kelly, P. B. Key, and S. Zachary, Distributed admission control. *IEEE J. Sele. Areas Communicat.*, **18**: 2617–2628, 2000.
11. L. Peterson and B. Davie, *Computer Networks: A Systems Approach*, 2nd ed. San Francisco, CA: Morgan Kaufmann Publishers, 2000.
12. L. Kleinrock, *Queueing Systems, Volume 1: Theory*. New York: Wiley–Interscience, 1975.
13. V. Jacobson, Congestion avoidance and control. *ACM Comput. Communicat. Rev.*, **18**: 314–329, 1988.
14. L. S. Bramko and L. L. Peterson, TCP Vegas: end-to-end congestion avoidance on a global Internet, *IEEE J. Sele. Areas Communicat.*, 1995, pp. 1465–1480.
15. J. Mo, R. J. La, V. Anantharam, and J. Walrand, Analysis and comparison of TCP Reno and Vegas. *Proceedings of Infocom*, 1999, pp. 1556–1563.
16. S. H. Low, L. Peterson, and L. Wang, Understanding vegas: A duality model. *J. of ACM*, **49**: 207–235, 2002.
17. T. V. Lakshman and U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. Network.*, 1997, pp. 336–350.
18. U. Madhow, T. V. Lakshman, and B. Suter, TCP/IP performance with random loss and bidirectional congestion. *IEEE/ACM Trans. Network.*, 2000, pp. 541–555.
19. H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. H. Katz, A comparison of mechanisms for improving TCP performance over wireless links, *IEEE/ACM Trans. Network.*, 1997.
20. H. Chaskar, T. V. Lakshman, and U. Madhow, TCP over wireless with link level error control: Analysis and design methodology, *IEEE/ACM Trans. Network.*, **7**(5): 605–615, 1999.
21. S. Floyd and V. Jacobson, Random early detection gateways for congestion avoidance, *IEEE/ACM Trans. Network.*, 1993, pp. 397–413.
22. S. Floyd, TCP and explicit congestion notification. *ACM Comp. Communi. Rev.*, **24**: 10–23, 1994.
23. K. K. Ramakrishnan and R. Jain, A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. *Proceedings of ACM Sigcomm*, 1988, pp. 303–313.
24. S. Kunniyur and R. Srikant, End-to-end congestion control: utility functions, random losses and ECN marks. *Proceedings of IEEE Infocom*, Tel Aviv, Israel, 2000.
25. F. P. Kelly, Charging and rate control for elastic traffic. *European Trans. Telecommunicat.*, **8**: 33–37, 1997.
26. F. P. Kelly, A. Maulloo, and D. Tan, Rate control in communication networks: Shadow prices, proportional fairness and stability, *J. Operat. Res. Soc.*, **49**: 237–252, 1998.
27. S. H. Low and D. E. Lapsley, Optimization flow control, I: Basic algorithm and convergence. *IEEE/ACM Trans. Network.*, **7**: 861–875, 1999.
28. R. La and V. Anantharam, Utility based rate control in the internet for elastic traffic. *IEEE/ACM Trans. Network.*, **10**(2): 272–286, 2002.
29. L. Chen, S. H. Low, M. Chiang, and J. C. Doyle, Jointly optimal congestion control, routing, and scheduling for wireless ad hoc networks, *Proceedings of IEEE Infocom*, Barcelona, Spain, 2006.
30. S. Shakkottai and R. Srikant, Network optimization and control. *Foundat. Trends in Networking*, **2**: 2007.
31. J. Mo and J. Walrand, Fair end-to-end window-based congestion control. *IEEE/ACM Trans. Network.*, **8**(5): 556–567, 2000.
32. D. Bertsekas, A. Nedic, and A. Ozdaglar, *Convex Optimization*. Belmont, MA: Athena Scientific, 2003.
33. S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.
34. J.-W. Lee, R. R. Mazumdar, and N. B. Shroff, Non-convex optimization and rate control for multi-class services in the internet. *IEEE/ACM Trans. Network.*, **13**: 827–840, 2005.
35. S. Sarkar and L. Tassiulas, End-to-end bandwidth guarantees through fair local spectrum share in wireless ad-hoc networks, *Proceedings of IEEE Conference on Decision and Control*, Maui, HI, 2003.
36. L. Tassiulas and A. Ephremides, Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks, *IEEE Trans. Auto. Control*, **36**: 1936–1948, 1992.

ATILLA ERYILMAZ
The Ohio State University
Columbus, Ohio

N

NETWORK RELIABILITY AND FAULT-TOLERANCE

When we make a telephone call, the call is connected through a communication network to the receiving party. Similarly, when we send an e-mail using the Internet, the message is sent through a communication network to the recipient. Such communication networks are made up of nodes and links that connect the nodes by hardware as well as the software components that allow for the functionality to communicate through such networks. Network reliability refers to the reliability of the overall network to provide communication in the event of failure of a component or components in the network. The term *fault-tolerant* is usually used to refer to how reliable a particular component (element) of a network is (e.g., a switch or a router.) The term *fault-tolerant network*, on the other hand, refers to how resilient the network is against the failure of a component.

Communication network reliability depends on the sustainability of both hardware and software. A variety of network failures, lasting from a few seconds to days depending on the failure, is possible. Traditionally, such failures derived primarily from hardware malfunctions that result in downtime (or "outage period") of a network element (a node or a link). Thus, the emphasis was on the element-level network availability and, in turn, the determination of overall network availability. However, other types of major outages have received much attention in recent years. Such incidents include accidental fiber cable cut, natural disasters, and malicious attack (both hardware and software). These major failures need more than what is traditionally addressed through network availability. For one, these types of failures cannot be addressed by congestion control schemes alone because of their drastic impact on the network. Such failures can, for example, drop a significant number of existing network connections; thus, the network is required to have the ability to detect a fault and isolate it, and then either the network must reconnect the affected connections or the user may try to reconnect it (if the network does not have reconnect capability). At the same time, the network may not have enough capacity and capability to handle such a major simultaneous "reconnect" phase. Likewise, because of a software and/or protocol error, the network may appear very congested to the user (1–3). Thus, network reliability nowadays encompasses more than what was traditionally addressed through network availability.

In this article, we will use the term *network reliability* in a broad sense and cover several subtopics. We will start with network availability and performability and then discuss survivable network design, followed by fault detection, isolation, and restoration as well as preplanning. We will conclude with a short discussion on recent issues and literature.

NETWORK AVAILABILITY AND PERFORMABILITY

Network availability refers to some measure of the reliability of a network. Thus, network availability analysis considers the problem of evaluating such a measure. [Note that in current literature, this is often termed as the network reliability analysis (4)]. Moore and Shannon did early work in this area (5). We discuss network availability through an example. Figure 1 shows that two telephones are connected by distribution segments (A) to local switches (S), while the switches are connected by the facility (B). The following allocation of outage/downtime percentage is assumed for the different elements: S 0.01%; A, 0.01%; B, 0.03%. Then, the availability of this connection is $(1-0.0001)^4(1-0.0003) = 99.93\%$; this translates to the maximum downtime of 368 min per year.

In general, network availability computation addresses the availability of a network in operational states, and discrete probability models are often used in analysis. Let $\mathscr{E}$ denote the set of elements of a network (for examples all the nodes and links). Each element may be in up or down state, where up refers to fully operational and down refers to total loss of the element. Let p_e denote that probability that element $e \in \mathscr{E}$ is up—this is also referred to as the availability of element e. Now consider the subset E_i of $\mathscr{E}$ consisting of the up elements of state i. Then, the probability that the network is in up state E_i is given by

$$Pr(E_i) = \prod_{e \in E_i} p_e \prod_{e \in \mathscr{E} \setminus E_i} (1 - p_e) \tag{1}$$

Note that there are $2^{|\mathscr{E}|}$ possible states (where $|\mathscr{E}|$ denotes the cardinality of the set $\mathscr{E}$); thus, usually network availability computation needs to deal with the problem of this exponential growth in states. A variety of algorithms for efficient computation have been developed over the years for different availability measures; the interested reader is directed to 4 and the references therein for additional information.

A related issue to availability is the performability. Most availability measures deal only with the connectivity aspect of the network; for example, what is the availability of a path from a source node to a destination node. However, when a failure occurs, the network may not be able to perform at the same level as when there was no failure. For example, the average network blocking in voice telephone networks (circuit-switched networks) is typically the measure for grade-of-service (GoS). A common value of GoS is 1% blocking under the normal operational mode, but under a specific outage, this may increase to more than 10% blocking; similarly, in a packet-switched network, the average packet delay may increase by an order of magnitude during a major failure compared to under the normal circumstances. Thus, the network failure performability addresses the performance of the network under various failure states. Consider a network with m elements that can

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

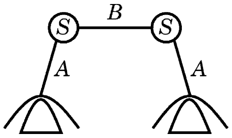


Figure 1. Network view for availability example.

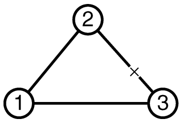


Figure 2. Three-node network.

each be either in operational or in a completely failed state; then, the total number of states is 2^m. The performability measure P is given by

$$\mathscr{P} = \sum_{k=1}^{2^m} Pr(k)X(k) \tag{2}$$

where $Pr(k)$ is the probability of state k, and $X(k)$ is the measure (e.g., network blocking in circuit-switched networks or average delay in packet-switched networks) in state k. Again, we face the issue of the exponential number of states. This can, however, be bounded by considering most probable t states as was first shown by Li and Silvester (6). Often, with the proper choice of t, the performability measure can be quite accurately computed. For example, if in a network, multiple simultaneous link failure scenarios are extremely unlikely, then the most probable states are the failure of each link independently. Accordingly, one may limit the computation to these states.

SURVIVABLE NETWORK CAPACITY DESIGN

While network availability and performability address important measures for evaluating the reliability of a network, designing the networks for survivability is extremely important for overall network reliability. In this section, we address this topic for the capacity design problem using separate examples for circuit-switched networks and packet-switched networks.

Circuit-Switched Traffic Networks Example

Consider the three-node circuit-switched network (Fig. 2) for which we are given that the availability of each link is 99.9%. We assume that the network has symmetric offered traffic (or load) and capacity. Offered load in circuit-switched networks is given in erlangs; this load is the product of the average call arrival rate and the average call holding time. For example, if the average call arrival rate is 200 calls/h and the average call holding time is 3 min, then the offered load is 10 erlangs (=3 × 200/60).

For the symmetric three-node network, offered load between any pair of nodes is assumed to be 10 erlangs, and the link capacity on each link is given to be 21 trunks (or circuits). We assume that the traffic between each pair of nodes is routed on the direct link that connects the end nodes of the pair, and we would like to know the call-blocking probability. For an offered load of a erlangs, and c trunks, and under the assumption that call arrival follows a Poisson process, Erlang-B loss formula can be used for computing the blocking probability, which is given by

$$E(c, a) = \frac{a^c/c!}{\sum_{k=0}^{c} a^k/k!} \tag{3}$$

Thus, in our example, we have $E(21, 10) = 0.000889 \approx 0.001$. That is, the network is providing a service quality (grade-of-service) of 0.1% blocking. (In actuality, the blocking for each pair of nodes is slightly different because any traffic blocked on the direct link can try the alternate route.)

Now, suppose that the link 2–3 fails; in this case, the network is still connected because node 1 is connected to node 3 via node 2. Assuming that the network still has the same amount of offered load, the load between node 2 and node 3 is now required to be routed through node 1; thus, the load offered to each link is 20 erlangs, whereas the capacity on each link is still 21 trunks. Thus, the blocking seen by traffic on each link is $E(21, 20) = 0.13144$, and the blocking seen by pair 2–3 traffic going through node 1 is even higher. Under the link independence assumption, the blocking on a path consisting of two links is given by $1 - (1 - b)^2$, where b is the link blocking probability. Thus, in our example, the blocking for traffic between 2–3 going through node 1 is $1 - [1 - E(21, 20)]^2 = 0.24558$.

Thus, we can see that, under no failure, the network provides a grade-of-service of 0.1%, whereas under a single link failure, the worst traffic pair blocking is 24.558%, although the network connectivity is still maintained. Recall that the link availability was assumed to be 99.9%; this means that the link can possibly be down for as long as 8 hours in a year. If we assume one event per link per year, then this link could conceivably be down for up to 8 hours straight! In some networks, this may be unacceptable given that the worst traffic pair blocking jumps to 24.558% from 0.01%. If we assume that the network should still provide a 0.1% blocking grade even under a single failure for every traffic pair, then to accommodate for the worst path blocking, we need link blocking on each of the remaining links to be b such that the path blocking for traffic between node 2 and node 3 using links 2–1 and 1–3 needs to satisfy $1 - (1 - b)^2 = 0.001$; this translates to $b = 0.0005$ for each link. Because we now have an offered load of 20 erlangs on each link, we need to find the smallest c such that $E(c, 20) = 0.0005$. Solving for integral c, we find that c needs to be at least 36 (i.e., we need to have 36 units of capacity on links 1–2 and 1–3 each). By the same argument, if we consider the failure of a different link independently, then the other two links each need 36 trunks. Thus, to cover for failure of each link independently, each link needs 36 trunks to provide the same level of blocking as was originally wanted for the network in the nonfailure mode. In other words, the network needs 80% more capacity to cover for a link failure compared to the no-failure case although network availability requirement was met.

Packet-Switched Networks Example

Consider this time a three-node packet-switched network. We will use Fig. 2 again. In packet networks, the offered traffic is usually given by the average packet arrival rate (packets per second, pps in short). If the average packet arrival rate to a network link is λ and follows a Poisson process, the average packet size is exponentially distributed with mean $1/\hat{\mu}$ kilobits, and the link speed is C kilobits per second (kbit/s), then the average packet delay (caused by the queueing phenomenon) can be obtained from the M/M/1 queueing system and is given by

$$T(\lambda, C, \hat{\mu}) = \frac{1}{\hat{\mu}C - \lambda} \tag{4}$$

For the three-node example, we assume unit mean packet size (i.e., $\hat{\mu} = 1$), in addition to assuming that the average arrival traffic between each pair of nodes is 10 packets per second and that the capacity of each link is 30 kbit/s. If all traffic between each node-pair is routed on the direct link, this provides an average delay of $T(10, 30, 1) = 0.05$ s, or 50 ms. Now suppose that the link 2–3 fails, then the traffic between node 2 and node 3 is routed through node 1; this induces an offered traffic of 20 pps on each remaining link. Thus, the average delay on each link (1–2 and 1–3) is 100 ms which is observed by traffic between nodes 1 and 2 and between nodes 1 and 3. On the other hand, the traffic between nodes 2 and 3 will go over two links and will thus experience a delay of $2 \times 100 = 200$ ms; this delay is four times more than under the no-failure situation.

If the network goal is to provide the average delay for any pair to be less than or equal to 50 ms under a single link failure, then to meet this condition we need link capacity C such that $2\,T(20, C, 1) = 2/(C - 20) = 0.05$ which implies that C needs to be 60 kbit/s on each of the remaining links. Similarly, if we consider the independent failure of a different link, then the other two links will require 60 kbit/s to provide the same level of service. Thus, in this network, we see that we need to double the capacity to provide the same level of service obtained under a single-link failure.

Discussion

We can see from these examples that if the network is *not* provided with additional capacity, then the traffic blocking can be very high in circuit-switched networks, which can result in excessive retry by users, or the packet backlog (queue) can build up in packet-switched networks. Thus, a transient effect can take place. From these two examples for two different networks, we can also see that, in some circumstances, the network capacity needs to be 80% to 100% more to provide the same level of service under a single link failure. This of course depends on the network objective (in our examples, we have used the objective that worst-pair traffic blocking or delay is minimized). In some networks, this near doubling of capacity can be cost-prohibitive; thus, the network performance requirement under failure may be relaxed. For example, under a single-element failure, it may be acceptable to have 5% blocking under a single link failure for the circuit-switched network case, or the average delay is acceptable to be 100 ms for the packet-switched network case. It is easy to see that this will reduce the additional capacity requirements in both cases.

Even though additional capacity can meet GoS requirement under a failure, the actual network topology layout and routing are also critical for survivable design (7). Thus, we also need to understand the network connectivity requirement for the purpose of survivability. For instance, a network needs to be minimally two-edge connected to address a single-link failure; this means that there must be two links connected to each node so that if one of them fails, a node can still be connected to the rest of the network through the other link; this avoids isolation of a node or a part of a network from the rest of the network. If a network is prone to multiple link failures at a time, this would require the network to have a higher degree of connectivity, which, in turn, would usually mean more network resource requirement to address for such failure situations. Survivable design for different node and edge connectivity level is extensively discussed in Ref. 8; the interested reader is directed to this reference for additional information.

Going back to the three-node examples, recall that the routing choice was limited to taking the *only* two-link path in the event of a failure. In a larger network, usually multiple routes between each origin and destination nodes are available; in the event of a failure, traffic can be sent on any of the unaffected paths. However, the actual flow on each path would depend on the actual routing rule in place as well as the availability of network capacity. Thus, it is not hard to see that the actual capacity requirement to address a failure in the network depends also on the actual routing schemes available in the event of a failure.

In any case, the overall network survivability and reliability depends on a number of issues. Network capacity design for survivability, as we see from these examples, plays an important part. In the next section, we discuss fault detection and isolation as well as network restoration—another key piece in network reliability.

FAULT DETECTION, ISOLATION, AND RESTORATION

Usually, different elements in a network are equipped with alarm generation capability to indicate the occurrence of any abnormal condition, which may cause the reduction or complete loss of the element. This abnormal condition is sometimes labeled as a fault. When an actual failure occurs, depending on the triggers set by various elements in the network, multiple alarms may be generated by a number of network elements—this is the fault-detection phase. Then, the network management system that monitors the network needs to determine the root cause of the fault. Fault isolation is the process of identifying the root cause of the fault. Thus, an issue that first needs to be addressed is correlation of alarms (9) to determine and isolate the actual point of failure in the network. Such fault-detection systems are needed to determine the cause of a fault quickly so that appropriate action can be taken. It is easy to see the relation of fault isolation to network reliability. The longer it takes to detect the cause of a fault, the longer it takes to fix it, and thus, conceivably the network is affected for a longer period of time, which decreases the performability of the network. Rule-based and model-based systems are used for fault isolation. Both centralized and distributed fault localization can be used; see Ref. 10 for a survey of different techniques.

Along with the fault-isolation phase, the restoration/repair phase begins. First, the network may be provided with additional capacity. If the additional capacity is provided so that even after failure the quality of service is met, then from the user's viewpoint, the failure is not perceived! Thus, a way of "restoring" the network is through additional capacity in the network (although, in actuality, the fault is not physically repaired yet). As we have already seen, to address for a single failure, the network may need twice the capacity, which may be sometimes cost prohibitive. Thus, the network may be provided with less than full spare capacity to address for a failure. In such cases, if the network has adaptive routing capability, then some of the traffic can be rerouted around the failure; thus, the users may not perceive the full impact of a failure.

Sometimes, the spare capacity can be provided in a different layer in the network because of cost and technological considerations. In the simplest architectural view of the communication network infrastructure, services such as voice or Internet are provided over *logical* switched or router-based networks; the capacity required for these logical networks is then provided over the physical transmission network, which may be connected by the digital cross-connect systems or SONET (Synchronous Optical Network) rings. For example, if a network is equipped with fast automated digital cross-connect system and/or SONET self-healing ring capability at the transmission network, the network where the services are provided may not perceive any failure because of fast automated restoration (11,12). At the same time, the transmission network level restoration schemes do not address failures such as a line card failure, or a switch or router failure; thus, restoration at the logical network level also needs to be triggered; this may include rerouting and automatic reconnection of affected connections. It is clear from this discussion that to restore from a failure, the network should be equipped with capacity as well as the proper network management system and software components to detect, isolate, and recover from a failure.

Other types of failures such as a software attack or a protocol operation failure cannot be addressed through the restoration process discussed earlier. An example is the SYN attack (2) in transmission control protocol (TCP), which severely affected an Internet service provider (TCP is the transport layer protocol on which services such as email, file transfer, and web browsing are provided in the Internet). In this case, the mechanism is needed to identify where such attacks are coming from so as to stop such attacks.

ADVANCED PREPARATION FOR NETWORK RELIABILITY

To provide network reliability, it is also important to do preplanning and/or advanced preparation. Of course, one way is to have additional spare capacity in the network. However, there can be a failure in the network that can actually take away the spare capacity if the network is not designed properly because of dependency between the logical network and the physical network (7). Thus, it is necessary to audit the network and find the vulnerable points in the network and then to equip the network with additional capabilities to avoid such vulnerabilities. For example,

1. The network may be provided with transmission-level diversity so that for any transmission link failure there is at least another path not on the path of the failure.
2. A redundant architecture at network nodes can be built to address for a node component or nodal failure; this may include dual- or multihoming to provide for multiple access and egress points to and from the core network.

To address for failures due to a software or protocol operations error or a software attack, different types of preparations are necessary. Several software errors that have occurred on various data networks such as the ARPANET, Internet, and SS7 Network (the data network that carries the signaling information for the public telephone network) (1,2) have caused such severe congestion in the network that it cannot be adequately addressed by normal congestion control schemes. Although enormous efforts go into developing robust software, it is not always possible to catch all possible software bugs (and sometimes bugs in the protocol operation). Should any software errors occur, the network should be provided with the capability to go to a known state in a speedy manner [e.g., speedy manual network reinitialization (1)]. If an error occurs as a result of a new feature, then it should have the ability to disable this feature and go to a known state for which the track record is good (3). To address for a software attack that can take advantage of a protocol's "loop hole," however, requires the development of intrusion-detection schemes.

RECENT ISSUES

Much research remains to be done to address network reliability in today's complex networking environment. We briefly touch on two areas in this regard: multilayered networking architecture and software errors/attacks.

Networking environment is evolving to various services being provided over multiple interconnected networks with different technologies and infrastructure. For example, the voice service is provided over circuit-switched networks, which are carried over the transmission network. Similarly, for Internet, applications such as web, email, and file transfers are carried over internet protocol (IP) layer connected by routers, which can be connected to the same transmission network or carried over an asynchronous transfer mode (ATM) or frame relay layer and then over the same transmission network. Thus, we are moving to an environment that we have coined the *multinetwork* environment. In such environment, in each of these networking layers, different types of failures/attacks and responses are possible. Some work in recent years has addressed this subject to some extent (7,13-17). It remains to be seen the impact of the failure propagation from one network to another, how the restoration process at each of these layers interacts with one another, whether they can make the best use of the network resources, and what type of network management coordination is needed for this purpose. Thus, network reliability in such interconnected multitechnology architecture needs further research.

Software/protocol operations errors and software attacks encompass the other area where mechanisms are needed to provide network reliability. This subject is relatively new—research on intrusion detection mechanisms is currently being explored to determine if an attack has occurred. Also, we need to see more work that helps us understand how severely the network will be affected in terms of network performance if a software attack or protocol failure occurs and how to recover from this anomaly. Also, the network architecture should be revisited to identify if there are ways to reconfigure the network after an attack so that parts of the network remain operational.

BIBLIOGRAPHY

1. B. A. Coan and D. Heyman, Reliable software and communication: III. Congestion control and network reliability, *IEEE J. Select. Areas Commun.*, **12**: 40–45, 1994.
2. S. Dugan, Cyber sabotage, *Infoworld*, **19**(6): 57–58, 1997.
3. D. J. Houck, K. S. Meier-Hellstern, and R. A. Skoog, Failure and congestion propagation through signalling controls, in J. Labetoulle and J. Roberts (eds.), *Proc. 14th Intl. Teletraffic Congr.*, Amsterdam: Elsevier, 1994, pp. 367–376.
4. M. O. Ball, C. J. Colbourn, and J. S. Provan, Network reliability, in M. O. Ball, et al., (eds.), *Network Models, Handbook of Operations Research and Management Science*, Vol. 7, Amsterdam: Elsevier, 1995, pp. 673–762.
5. E. Moore and C. Shannon, Reliable circuits using less reliable relays, *J. Franklin Inst.*, **262**: 191–208, 281–297, 1956.
6. V. O. K. Li and J. A. Silvester, Performance analysis of networks with unreliable components, *IEEE Trans. Commun.*, **32**: 1105–1110, 1984.
7. D. Medhi, A unified approach to network survivability for teletraffic networks: Models, algorithms and analysis, *IEEE Trans. Commun.*, **42**: 535–548, 1994.
8. M. Grötschel, C. L. Monma, and M. Stoer, Design of survivable networks, in M. O. Ball, et al. (eds.), *Network Models, Handbook of Operations Research and Management Science*, vol. 7, Amsterdam: Elsevier, 1995, pp. 617–672.
9. G. Jakobson and M. Weissman, Alarm correlation, *IEEE Netw.*, **7** (6): 52–59, 1993.
10. S. Kätker and K. Geihs, A generic model for fault isolation in integrated management systems, *J. Netw. Syst. Manage.*, **5**: 109–130, 1997.
11. W. D. Grover, Distributed restoration of the transport network, in S. Aidarous and T. Plevyak (eds.), *Telecommunications Network Management into the 21st Century*, Piscataway, NJ: IEEE Press, 1994, pp. 337–417.
12. T.-H. Wu, *Fiber Network Service Survivability*, Norwood, MA: Artech House, 1992.
13. R. D. Doverspike, A multi-layered model for survivability in intra-LATA transport networks, *Proc. IEEE Globecom'91*, 1991, pp. 2025–2031.
14. R. D. Doverspike, Trends in layered network management of ATM, SONET, and WDM technologies for network survivability and fault management, *J. Netw. Syst. Manage.*, **5**: 215–220, 1997.
15. K. Krishnan, R. D. Doverspike, and C. D. Pack, Improved survivability with multi-layer dynamic routing, *IEEE Commun. Mag.*, **33** (7): 62–69, 1995.
16. D. Medhi and R. Khurana, Optimization and performance of network restoration schemes for wide-area teletraffic networks, *J. Netw. Syst. Manage.*, **3**: 265–294, 1995.
17. D. Medhi and D. Tipper, Towards fault recovery and management in communication networks, *J. Netw. Syst. Manage.*, **5**: 101–104, 1997.

READING LIST

This list includes work on network reliability that address different failure and fault issues. This list is by no means exhaustive. This sampling should give the reader some feel for the wide variety of work available for further reading, as well as lead to other work in this subject.

Y. K. Agrawal, An algorithm for designing survivable networks, *AT&T Tech. J.*, **63** (8): 64–76, 1989.

D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed., Englewood Cliffs, NJ: Prentice-Hall, 1992.

C. Colbourn, *The Combinatorics of Network Reliability*, Oxford, UK: Oxford Univ. Press, 1987.

P. J. Denning (ed.), *Computers Under Attack: Intruders, Worms, and Viruses*, Reading, MA: ACM Press & Addison-Wesley, 1990.

B. Gavishet al., Fiberoptic circuit network design under reliability constraints, *IEEE J. Select. Areas Commun.*, **7**(8): 1181–1187, 1989.

B. Gavish and I. Neuman, Routing in a network with unreliable components, *IEEE Trans. Commun.*, **40**: 1248–1258, 1992.

A. Girard and B. Sansó, Multicommodity flow models, failure propagation, and reliable loss network design, *IEEE/ACM Trans. Netw.*, **6**: 82–93, 1998.

W. D. Grover, Self healing networks: *A distributed algorithm for k-shortest link-disjoint paths in a multigraph with applications in real time network restoration*, Ph.D. Dissertation, Univ. Alberta, Canada, 1989.

Fault Management in Communication Networks, Special Issue of *J. Netw. Syst. Manage.*, **5** (2): 1997.

Integrity of Public Telecommunication Networks, Special Issue *IEEE J. Select. Areas Commun.*, **12** (1): 1994.

Y. Lim, Minimum-cost dimensioning model for common channel signaling networks under joint performance and reliability constraints, *IEEE J. Select. Areas Commun.*, **8** (9): 1658–1666, 1990.

C. L. Monma and D. Shallcross, Methods for designing communications networks with certain two-connected survivability constraints, *Oper. Res.*, **37**: 531–541, 1989.

L. Nederlof et al., End-to-end survivable broadband networks, *IEEE Commun. Mag.*, **33** (9): 63–70, 1995.

B. Sansó, F. Soumis, and M. Gendreau, On the evaluation of telecommunication networks reliability using routing models, *IEEE Trans. Commun.*, **39**: 1494–1501, 1991.

D. Shier, *Network Reliability and Algebraic Structures*, Oxford, UK: Oxford Univ. Press, 1991.

D. Tipper et al., An analysis of congestion effects of link failures in wide-area networks, *IEEE J. Select. Areas Commun.*, **12**: 179–192, 1994.

DEEPANKAR MEDHI
University of Missouri–
Kansas City
Kansas City, Missouri

NETWORK SECURITY FUNDAMENTALS

OVERVIEW

Communication is at the heart of computer systems. Information stored at one location is moved to another location, combined with data from other sources, and processed to meet the needs of the users. Rarely can user commands or the information they access avoid traversing networks, which leaves the data and systems vulnerable to a variety of attacks. A malicious intruder might intercept, falsify, damage, or altogether prevent the networked communications. This chapter focuses on security goals, vulnerabilities, and defenses for networked systems. When we use the word *security*, it is in the context of networked computer systems; other use will be made clear by context.

In discussing security issues, we will often make use of examples, using the characters *Alice*, *Bob*, and *Eve*. Normally, Alice and Bob are attempting some sort of communication, a *message*, and Eve is maliciously interfering with or intercepting it. These examples are representative, but they should not be taken too literally. Alice and Bob are likely to be computer programs exchanging information, perhaps performing a *handshake* to initiate a session. Eve may be some sort of eavesdropping program logging communication between Alice and Bob, or she might be an *agent* actively introducing falsified messages. Therefore, in our context, Eve can be either a passive attacker or an active attacker.

A common theme in both networking and security is the idea of a *protocol*—a prescribed sequence of events designed to facilitate communication among the participants. In particular, *cryptographic protocols* are used to exchange information that should remain confidential.

Security Goals

A variety of goals can be identified with regard to network security, but three basic ones stand out: *confidentiality*, *integrity*, and *availability* (sometimes reduced to the acronym CIA). Information must be readily accessible when needed. Alice has sent Bob a message, and he wants to get it. It must be available to him. Information must be intact, and accurate, secure from both accidental and malicious change. The message Bob gets from Alice should be that message exactly as Alice sent it. Its integrity must be preserved as it crosses the network. Access must be restricted to confidential data. Eve should not be able to read the message Alice has sent to Bob. Things intended to be private must remain private.

Principles

A few key principles are recognized when discussing security First, security is an ongoing process rather than a tool or simple action (Bruce Schneier's dictum is "Security is a process, not a product"). Second, security may be dependent on technology, but both people and business processes are crucial elements and often vulnerable. If Bob keeps his password on a sticky note on his monitor, Eve may have an easy time reading Bob's messages from Alice! *Social engineering* is a term for attacking security simply by deceiving the people involved and lying to them to gain passwords or other *keys* to access systems or data. Eve calls Bob, claims to be from the IT department, and says she needs his password to correct a (nonexistent) problem with his e-mail. Third, security has an inescapable economic element—any security effort requires some cost and in exchange reduces some risks. Careful *risk analysis* is necessary to ensure that the security process is cost effective. Finally, *defense in depth* is necessary, with layers of security protecting resources. Any one layer might be compromised, but if yet more layers exist the information remains secure.

Policies

Any organization interested in security needs a *security policy*, a clear statement of the organization's specific goals, defining security for their information systems. Policies serve a variety of purposes. They define what is and is not acceptable use of an organization's information resources. They identify what *assets* are to be protected from what *threats*, and they establish priorities for dealing with the various risks. They provide plans, practices, and processes to follow to improve security and to deal with breaches of security. Among other roles, an organization's security policy can be an effective tool for educating personnel about both the importance of security and the practices for achieving it.

Elements

Security is a broad field, and it requires broad knowledge, particularly of software and networking. Our approach to organizing the topic is to first discuss the foundations of security in cryptography. Then we will discuss various security *services* offered on networks. The following section will discuss the various attacks against our security goals and services. After discussing *vulnerabilities* and attacks, we address defense mechanisms that can be used to thwart the attacks. Finally, we briefly discuss some issues specifically related to wireless network security.

FOUNDATIONS

When two or more computers want to talk in a "secure" way, they may expect their communication to meet one or any combination of the following security requirements, which are an extended set of the traditional security goals, CIA:

- *Confidentiality* (or privacy)—Ensuring that no one can access the information except the intended receiver.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

- *Integrity*—Ensuring that data are not maliciously tampered with during transmission and operation and can be altered only by those authorized.
- *Availability*—Ensuring that computer services are operable and resources are accessible throughout their lifetimes.
- *Authentication*—Ensuring that the identity of an entity in the communication is genuine.
- *Authorization* (access control)—Determining and enforcing who is allowed access to what resources, hosts, software, and network connections.
- *Non-repudiation*—Ensuring that it can be definitely established that an entity has sent a particular message, and that they cannot deny having done so.

To meet these requirements, use of *cryptography* is necessary, although not sufficient. Generally speaking, cryptography is the art of scrambling information into some unintelligible form and (usually) providing a secret method of recovering the original message. When a message is in its original, unscrambled form, it is called *plaintext* (or *cleartext*). The scrambled message is known as *ciphertext*. The process of converting plaintext to ciphertext is known as *encryption*, whereas recovering the original plaintext message from the encrypted ciphertext is called *decryption*.

Three main types of cryptographic schemes exist: *secret key* (also called *shared key* or *symmetric*) cryptography, hash functions, and *public key* (or *asymmetric*) cryptography. To implement such schemes, modern cryptographic technology makes extensive use of mathematics, especially *number theory*. As is obvious from the names, a *key* plays a special role in cryptography. A *key* is a parameter to a cryptography algorithm that controls details of its behavior. It is common practice to assume that someone attacking a cryptographic scheme knows all the algorithm and process details except for the key. Among other consequences, this dictates that *security through obscurity* is not an acceptable approach, and that open security processes and algorithms may be more secure as they can be thoroughly tested by the entire community.

Basic Number Theory

In this section, we will describe some concepts and theorems that have been commonly used in cryptographic algorithms.

Most public key cryptographic algorithms are based on *modular arithmetic*. Modular arithmetic is arithmetic where the results of operations are restricted to non-negative integers less than some fixed number n, a range of $[0\ldots(n-1)]$. An integer converted to this range is *modulo n* or *mod n*. Conceptually, modular operations perform ordinary arithmetic operations (such as addition, multiplication, or exponentiation) and then convert the result to the appropriate range by returning the remainder after division by n. So, for example, $23 \; mod \; 10 = 3$; and $(5 + 8) \; mod \; 10 = 3$.

First we define some basic concepts:

A *prime* number is a positive integer that is evenly divisible by exactly two positive integers (itself and 1).

The largest number that divides two numbers n and m is called the *greatest common divisor (gcd)* and denoted $gcd(n, m)$.

Two integers are *relatively prime* if and only if their *gcd* is 1.

The *multiplicative inverse* of x, denoted x^{-1} or $1/x$, is the number that, when multiplied by x, yields 1. For example, 7 is the multiplicative inverse of 3 *mod* 10, because $7 \times 3 \; mod \; 10 = 1$. A multiplicative inverse does not always exist. A positive integer x has a multiplicative inverse *mod n* if $gcd(x, n) = 1$.

Some modular arithmetic properties are important in cryptographic algorithms, such as:

Modular addition: $(a + b) \; mod \; n = [(a \; mod \; n) + (b \; mod \; n)] \; mod \; n$.

Modular subtraction: $(a - b) \; mod \; n = [(a \; mod \; n) - (b \; mod \; n)] \; mod \; n$.

Modular multiplication: $(a \times b) \; mod \; n = [(a \; mod \; n) \times (b \; mod \; n)] \; mod \; n$.

These three properties are particularly useful, because we can substitute the expressions on the right for those on the left, applying the *mod* operator early and often and keep the values in our calculations smaller than n.

Modular addition and multiplication form a commutative ring with the laws of (where a, b, and c are all positive integers):

Associativity: $[(a + b) + c] \; mod \; n = [a + (b + c)] \; mod \; n$.

Commutativity: $(a + b) \; mod \; n = (b + a) \; mod \; n$.

Distributivity: $[(a + b) \times c] \; mod \; n = [(a \times c) + (b \times c)] \; mod \; n$.

Fermat's theorem (or *Fermat's little theorem*)—If p is a prime number, then for any integer a, $a^{p-1} mod \; p = 1$.

Euler's totient function—the totient $\phi(n)$ of a positive integer n is the number of positive integers less than n and relatively prime to n. Therefore, if n is prime, $\phi(n) = n-1$. If $n = p \times q$, and p and q are primes, $\phi(n) = (p - 1)(q - 1)$.

Euler's Theorem—for every a and n that are relatively prime, $a^{\phi(n)} mod \; n = 1$. Then we can prove that, for modular exponentiation, we have $x^y \; mod \; n = x^{y \; mod \; \phi(n)} mod \; n$, where x and y are positive integers too.

The *Euclidean algorithm* (also called *Euclid's algorithm*)—one of the earliest algorithms developed, it can be used to find the *gcd* or to find the modular inverses of a number n, given n and a modulus m. The algorithm does not require factoring the two integers (a slow process) but works by repeatedly dividing the two numbers and their remainder in turns.

Secret Key Cryptography

Secret key cryptography uses a single, shared key for both encryption and decryption. It leads to the necessity of sharing the keys securely, called the *key exchange* or *key distribution* problem.

Based on how data are processed in encryption, secret key cryptography schemes are usually categorized as either *stream cipher* or *block cipher*.

Stream ciphers operate on plaintext digits one at a time and constantly changes the key used in transformation of successive digits. The sequence of keys used is called a *keystream* and is often generated from a simpler key. A stream cipher generates successive digits of the keystream based on an internal state. Based on the two essential ways that the state can be updated, stream ciphers are categorized into two types: *synchronous stream ciphers*, where the state is updated and therefore the keystream is generated independently of the plaintext and ciphertext, and *self-synchronizing ciphers*, where the state is updated based on the previous ciphertext.

Stream ciphers are often used when the length of the plaintext is unknown in advance. Examples of stream ciphers include RC4, LEVIATHAN, A5/1, A5/2, Chameleon and FISH. RC4 was designed by Rivest for RSA Security. It uses a keystream of variable size and operates on bytes. RC4 is used for file encryption. LEVIATHAN is a *seekable* stream cipher, which means that the user may efficiently skip forward to any part of the keystream. LEVIATHAN generates a keystream efficiently using its unique tree structure.

Block ciphers operate on a large block of plaintext digits each time, and the transformation uses a fixed key. Block ciphers can operate in different modes, of which the following four modes are common:

Electronic Code Book (ECB)—the simplest but the "worst" method. In ECB, each block is independently encrypted with the secret key and then all ciphered blocks are assembled together. A problem of ECB is that identical blocks will generate identical ciphertext. As such it is susceptible to brute-force attacks, and malicious alteration is also possible.

Cipher Block Chaining (CBC)—avoids the problem in ECB by exclusively-ORing (XORing) the previous block of ciphertext with the next plaintext before applying encryption with the secret key. As no previous block for the very first data block exists it uses an initial random number known as an Initialization Vector (IV). The use of an IV guarantees that repeated identical blocks of plaintext result in different ciphertext each time they are encrypted.

Output Feedback (OFB)—allows encryption of blocks of varying sizes. It generates a sequence of one-time pads by encrypting the previous feedback and then feeding it into a shift register. Only *k* bits (size of the blocks) are kept, and the remaining is discarded. The initial pad is generated from an IV. The one-time pad is XORed with plaintext to generate the ciphertext. OFB does not propagate errors but is vulnerable to message alteration if an attacker knows the plaintext and the ciphertext.

Cipher Feedback (CFB)—similar to OFB but takes the previous ciphertext (not the previous feedback) to generate the one-time pads.

A few secret key cryptographic algorithms in use today, which employ block ciphers, are *Data Encryption Standard (DES)*, *International Data Encryption Algorithm (IDEA)*, and *Advanced Encryption Standard (AES)*.

DES was published in 1977 by the National Bureau of Standards (NBS) [now the National Institute of Standards and Technology (NIST)]. It was designed by IBM based on their Lucifer cipher. DES uses a 56-bit key and operates on 64-bit blocks. DES employs complicated rules and rounds of transformations, which is asserted to be specifically designed to yield efficient implementation in hardware but relatively slow implementation in software. However, advances in CPUs have made it feasible to implement in software. DES is now considered to be insecure for many applications, mainly because of its small key size. Some other weaknesses have been proved in theory but remain infeasible to mount in practice.

Two variations of DES are Triple-DES (3DES) and DES-X (DESX). 3DES uses up to three 56-bit keys (to prevent brute-force attacks) and makes three encryption passes over each block (to prevent man-in-the-middle attacks). DESX was designed to increase the difficulty of brute-force attacks by XORing an extra 64-bit key to the plaintext before applying DES, and then XORing another 64-bit key after the encryption.

IDEA was developed by Xuejia Lai and James L. Massey of ETH Zurich and published in 1991. It was originally called IPES (Improved Proposed Encryption Standard). IDEA was designed to improve efficiency in software implementation. IDEA uses a 128-bit key and operates on 64-bit blocks as does DES. However, IDEA relates the encryption and decryptions keys in a more complicated manner. IDEA is patented by Ascom.

AES, also know as *Rijndael*, is a successor of DES and has replaced DES for many applications where 3DES was too slow. The algorithm was designed by Belgian cryptographers Joan Daemen and Vincent Rijmen. AES allows variable block and key sizes, and the latest specification allows a choice of any combination of block sizes and key sizes of 128 bits, 192 bits, or 256 bits.

Other secret-key cryptographic algorithms include *blowfish*—a 64-bit block cipher invented by Bruce Schneier, which is optimized for 32-bit processors with large data caches; *twofish*—a 128-bit block cipher using 128, 192, or 256 bit keys; and *CAST-128*—a DES-like substitution-permutation algorithm using a 128-bit key operating on a 64-bit block.

Public Key Cryptography

As secret key schemes require that the shared key(s) be exchanged before any secure communication can take place, an alternative that does not require prior preparation is appealing. *Public key encryption* makes use of two keys, a *public key*, used for encryption, and a *private key*, used by the recipient of a message to decrypt it. As the public key can only be used for encryption, it is safe for anyone to know and use, because all they can do is create a message that only the holder of the private key can decrypt. The two keys are related mathematically through a *trapdoor one-way* function—one that is easy to compute in the forward direction but hard to invert without the knowledge of the trapdoor (private key). The common (and widely used) example of such a trapdoor one-way function is *prime factorization*: It is easy to compute the product of two large prime numbers, but difficult to factor such a product, unless you already know one of the factors.

Public key schemes are used for a variety of purposes. The public key can be used to encrypt a message sent to the holder of the corresponding private key. The holder of a private key can use it to sign (encrypt a copy of) a message, which allows recipients to verify the signature using the public key. Public key schemes may also be used to securely exchange the shared keys needed for a secret key scheme. Secret key schemes are often markedly more efficient than public key schemes, so using public key encryption to exchange a (relatively small) key, which is then used to encrypt the bulk of the communications, is an effective use of computation resources.

The *Diffie–Hellman key exchange* is the earliest example in the literature. Rivest, Shamir, and Adelman developed what has become known as the *RSA* algorithm, perhaps the best known of public key schemes. More recent examples include *ElGamal* and the NIST Digital Signature Algorithm (DSA).

Hash Functions

Hash functions, also called *message digests* or *one-way transmissions*, are algorithms for creating small fixed-size digital "fingerprints" for any kind of data. The output of a hash function is called the *hash value*. Let H denote a hash function, m denote the input message, and h denote the hash value. Then $h = H(m)$. Hash functions used for security must be *one-way*; that is, given a hash value h, it is computationally infeasible to find an input x such that $H(x) = h$.

Besides the one-way property, properties required of hash functions include *randomness* and *collision free*. Randomness means that the resulting output should appear random and is not affected by the pattern of the message. Collision free means it should be computationally infeasible to determine two different messages with the same hash values (two different inputs with the same hash value is referred to as a *collision*). Collision-free hash functions protect message integrity because it will not be feasible to substitute a forged message for another message still produces the same hash value. Other properties of hash functions include *flexibility* (the function can be applied to messages of any block size), *convenience* (the function produces short output value), and *performance* (it is fast to compute a hash value).

Because of these properties, hash functions can be used to provide security services such as authentication of users, authentication of messages by generating a *message authentication code* (*MAC*), data integrity, and encryption. Because hash functions are not reversible, both encryption and decryption need to run the algorithm in the forward direction—usually a hash value is Exclusive-ORed with a message to produce the ciphertext, whereas the same hash value is Exclusive-ORed with the ciphertext to recover the original message.

The hash algorithms that are in common use today include:

Message digest (MD) produces a 128-bit hash from a message of arbitrary size. There are a series of MD algorithms, such as MD2, MD4, and MD5. MD2 was designed for systems with limited memory (1). It takes a message of an arbitrary number of octets and produces a 128-bit digest. MD4 was designed to be 32-bit word oriented for fast processing on 32-bit CPUs. MD4 can handle messages with an arbitrary number of bits, whereas MD2 requires the message to be an integral number of octets. MD4 was developed by Rivest. Also developed by Rivest to diminish potential weaknesses reported in MD4, MD5 makes more manipulation on the original data to achieve better security but with compromises on performance. MD5 has been implemented in a large number of products.

Secure hash algorithm (SHA) was proposed by NIST in the *secure hash standard* (*SHS*). The first member of the SHA family was published in 1993. SHA-1 was published two years later. It takes a message of at most 2^{64} bits long and produces a 160-bit hash value (2). Compared with MD5, SHA-1 is a little slower to execute but presumably more secure. SHS proposed four other versions of the algorithm: SHA-224, SHA-256, SHA-384, and SHA-512, which produce hash values of length 224-, 256-, 384-, or 512-bit, respectively.

RIPEMD is a series of hash functions that came from RACE Integrity Primitives Evaluation Message Digest. RIPEMD is based on the design principles used in MD4. RIPEMD-160 was developed by Hans Dobbetin, Antoon Bosselaers, and Bart Preneel, and first published in 1996. It is a 160-bit hash function and has similar performance to SHA-1. Other members of the RIPEMD family include RIPEMD-128, RIPEMD-256, and RIPEMD-320, which are 128-, 256-, and 320-bit versions of the algorithm, respectively. RIPEMD-256 only reduces the chance of accidental collisions and does not offer a higher level of security than RIPEMD-128. The reason lies in the fact that RIPEMD-256 is similar to RIPEMD-128 but initializes two parallel lines with different initial values and then exchanges a chaining variable between the two parallel lines after each round. Therefore, RIPEMD-256 does not introduce more complexity than RIPEMD-128 for attackers to launch *collision attacks* (to find two different inputs that will produce the same hash value). This feature also applies to RIPEMD-320 with respect to RIPEMD-160.

Some other hash functions include *HAVAL (HAsh of VAriable Length)*, a hash algorithm with many levels of security, and *Whirlpool*, a relatively new function.

Researchers have found collision attacks can be launched against MD5, SHA-0, RIPEMD, and other hash functions. Despite these attacks, there are many products that use these hash functions, and it will take years to substitute other functions (once such functions have been agreed on).

SECURITY SERVICES

Once cryptographic fundamentals are in place, one can deploy them as part of various services to provide security. Managing the keys for cryptography, authenticating users, and authorizing access to resources are all examples of such services. A clear distinction should be maintained between *authentication*, establishing a positive identification, and *authorization*, permitting a known individual to access some resource.

Key Management

Keys are an essential element of secure communications, and services must support their use. Services needed include generating keys (or key pairs in the case of public key systems); distributing keys—Alice must first obtain Bob's public key before she can encrypt a message using it; and storing keys—if we want to verify Alice's digital signature on a document, we need access to the public key corresponding to the private key she used to sign it. Such services provide a point of attack against cryptographic systems. If the stored keys can be compromised, then falsified documents can be sent and authentication efforts can be foiled. Often, discussions of system security assume that key management is secure—it is conducted by a *trusted third party*. Another (and necessary) service is the ability for users to revoke a key pair if their private key is ever compromised. The keys were (and remain) valid for messages transmitted prior to their revocation but can no longer be used for new messages.

Identity Authentication

A crucial service is authenticating the identity of an entity. When Alice and Bob communicate, they want to authenticate each other, because Eve might be masquerading as either of them. When users connect with a banking service, they want to be certain they are providing their PIN or password to their bank and not to someone who will use it to empty their account. *Phishing* attacks work by masquerading as a service provider and then collecting usernames, passwords, and other crucial information from victims. Authentication is usually by means of a *password* or *identity token* of some sort. Authentication testing relies on the information, physical item, or biological characteristic being unique to (and in possession of) the individual being identified. Iris scanning, fingerprinting, and DNA analysis make use of biological tokens. Keys, badges, or smart cards are physical items often used for authentication. Information tokens include passwords and such familiar security questions as, “What is your mother's maiden name?”

Password Authentication

A common approach to authentication in computer systems is the use of usernames and passwords. The system maintains a store associating usernames with passwords, and so long as a user provides a valid username and the corresponding password, they are authenticated (so far as the system is concerned). This approach assumes that the password has not been compromised. The simplest approaches transmit or store passwords in clear text. *telnet* and *ftp* were popular programs vulnerable because of cleartext passwords. Their use has largely been replaced by *ssh* or *sftp* (secure ftp). Common improvements on password based authentication include storage and transmission of passwords in a hashed form and the addition of *salt*, a random value hashed with the password. A good example is the LDAP (Lightweight Directory Access Protocol) Authentication Password Schema (3).

Digital Signatures

Similar to authentication of a user is the need to determine the author of a document. This determination is accomplished via digital signatures. Using an asymmetric encryption technique, Alice can sign a document using a hash of the document that she encrypts with her private key. Bob can verify the signature using Alice's public key. Assuming her private key has not been compromised, Alice cannot deny sending a message with her digital signature (*nonrepudiation*). As keys can be compromised, this a need then exists to revoke a public–private key pair. Documents signed with a key after revocation are no longer assumed valid.

SECURITY ATTACKS

A wide variety of attacks are aimed at breaking system security and privacy. To understand the need for certain defense mechanisms, it is vital to understand how a system might be insecure without it. In this section, we provide a brief discussion of different attacks.

Sniffing

Sniffing is a category of passive attack. An attacker can use a special program, denoted as *sniffer*, to passively monitor a computer network for key information without interfering much with normal activities in the network. The key information may be authentication information such as a password or any other information transmitted in packets such as IP address and TCP ports. In some case, even traffic characteristics of packets such as frequency, size, and interarrival time may be critical for security.

Sniffing's principle is simple. In a broadcast-based environment such as Ethernet, a network card can be used to monitor traffic on the media. If the frame's destination MAC address is the card's MAC address, the frame is accepted by the card and forwarded to upper layers of the protocol stack. Otherwise the network card discards the incoming frame. To deploy sniffing, a sniffer puts a network card into *promiscuous mode*, so that all frames on the network segment are accepted and captured. The sniffer generally listens on a special network programming socket, denoted as *raw socket*, and captures all packets. Typically sniffers run a protocol analysis of the captured packets and extract the interesting information. In a switched network, additional techniques are necessary to divert the traffic to a machine for capture and analysis.

A popular sniffing and protocol analysis tool is the open source Wireshark (www.wireshark.org, formerly known as Ethereal). Although Wireshark is available for both UNIX and Windows. Besides its sniffing capabilities, Wireshark is an excellent protocol analyzer of captured packets. Many other sniffing tools are tailored for special purposes, such as tcpdump (command line version of Wireshark), dsniff, ettercap, and Cain & Abel. Many of these tools are multipurpose programs, providing features in addition to sniffing.

Session Hijacking

Session hijacking is a category of active attack. In a session hijacking, an attacker puts herself in the middle of the communication path between a client and a server, takes over the communicating session, and pretends to be one of the participants. Session hijacking is usually an extension of sniffing.

An attacker may use a variety of other attacks to redirect the traffic through her own box. These attacks may include ARP poisoning, MAC flooding, port stealing, DHCP spoofing, DNS spoofing, and various routing games such as ICMP redirect. Once an attacker puts herself between the client and the server, she can change the content of a session. For example, if the client is downloading software, the attacker can attach a Trojan horse code to the download. An attacker may also hijack SSL and SSH sessions by providing fake certificates. Session hijacking may be used against both TCP and UDP sessions. To hijack TCP sessions, the attacker must make an extra effort to deal with sequence numbers and other TCP details.

Popular hijacking tools include ettercap, dsniff, and hunt. Ettercap runs on most popular systems such as Linux and Windows. Ettercap on Linux is stable. It is a multipurpose program used primarily for sniffing, capturing, and logging traffic on switched LANs by using attacks such as ARP poisoning. For example, ettercap can redirect an HTTP session through its host and change the HTTP page.

Spoofing

Spoofing is a category of active attack. In such an attack, an attacker intentionally "provides false information about a principal's identity in order to obtain unauthorized access to systems and their services" (4). In an IP spoofing attack, the attacker changes the source IP address within the IP header of a packet so that the packet source IP address can be random and the packets appear to have come from a different source other than the real sender. This is a simple way for an attack over a network to obscure its origin. E-mail headers are also easy to spoof so that the e-mail appears to be sent from another person. A web page may also be spoofed so that users think they are accessing a known site, but actually they are receiving web pages controlled by an attacker. This can be achieved in the following way: Recall the DNS service maps website names to their IP addresses. An attacker may use DNS spoofing to redirect user requests to a malicious website by replying to DNS requests with the IP of the malicious site rather than the IP of the actual site.

Password Cracking

In modern computer systems, passwords are not stored in clear text in case attackers break into the system and obtain the password store. Passwords are normally hashed and then stored with user names as indexes. Even when passwords are stored in a hashed form, if an attacker obtains the list of hashed passwords, they can use either a dictionary attack or a brute force attack to retrieve the original passwords.

In a dictionary attack, an attacker first obtains a list of common passwords. Those common passwords might be common names, words, place names, or acronyms. Then the attacker hashes all those common passwords and compares them with the victim's password hash. If there is a match, the attacker obtains the original password. The attacker may precompute hashes of these common passwords to speed the password cracking. If a dictionary attack does not work, the attacker may also resort to a brute force attack. In a brute force attack, the attacker tries all combinations of password elements such as letters, numbers, and special symbols until there is a hit.

A cracking dictionary might contain a great number of common words, names, and variations, and the size of such a popular dictionary could be over 19 MB. Therefore, it is important to choose a good password (i.e., one incorporating random elements) because it is just a matter of time for an attacker to crack the hashed password. A good password will greatly increase the cracking time and extend the life span of the password. Choosing an appropriate hash function is also important. Popular hash functions include DES (by early UNIX systems) and MD5. MD5 is safer than DES because MD5 creates a hash of 128 bits and generates a much larger password pool.

Popular password cracking tools include John the Ripper, Crack, L0phtCrack, and Cain & Abel. John the Ripper is a powerful password cracking tool that works under both UNIX and Windows.

Denial of Service

A *denial-of-service* (*DoS*) attack is a category of active attack. In a DoS attack, an attacker tries to exhaust a limited resource available to users. Basically, if some resource is limited, the attacker may use a variety of approaches to use it up so that no one else can access it. Thus, access to the resource is denied to authorized users.

An attacker may attack a resource locally. Those resources may include the local system process table, CPU time, disk space, and index node (*inode*) of a UNIX system. An attacker may also attack a resource remotely. These resources may include the entire remote system, a specific service, or network bandwidth. For example, in a ping of death attack, an oversized ping packet could cause a remote Windows 95 system memory leakage and crash the system. In a SYN flood attack, flooding SYN packets arriving at a host may overfill the TCP half-open connection buffer so that no more legal connections will be allowed.

Distributed Denial of Service (DDoS)

The *distributed denial-of-service* (*DDoS*) attack uses multiple attacking entities to prevent the legitimate use of a service. On the Internet, each entity such as a host, network, and service has limited resources. If these resources are consumed by too many users, no more users can access them. Because of the administration and privacy requirements, security mechanism deployment on the Internet is often not coordinated across multiple domains; yet Internet security is highly interdependent, which is why an attacker may deploy a DDoS attack on the Internet.

A DDoS attack has two phases. First, attackers compromise several hosts. These hosts become *masters*, which are also called *handlers*. Masters then compromise hundreds or even thousands of additional hosts, called *zombies*, and install DDoS flooding tools on the zombies. Zombies are also called *daemons*, *slaves*, or *agents*. This compromising process is normally automated and searches for a large number of vulnerable hosts, such as those without recent security patches. When attackers are ready to attack, a signal is transmitted from masters to all zombies, which all generate attacking traffic to throttle the target. Using masters allows attackers to hide their origin. To further hide their traces, attackers may access masters through a sequence of stepping stones, i.e., intermediary compromised machines, which may be scattered around countries across different continents. IP spoofing is often used by zombies to further obscure the attackers.

Popular DDoS tools include trinoo, Tribe Flood Network (TFN), Tribe Flood Network 2000 (TFN2K), stacheldraht, shaft, and mstream. TFN is made up of client (master) and zombie programs and is capable of deploying ICMP flood, SYN flood, UDP flood, and Smurf style attacks. It can also work as a backdoor and provide an "on-demand" root shell bound to a TCP port.

Cryptanalysis

Cryptanalysis is an attack that may retrieve secret information such as the encryption key from ciphertext (encrypted message). Four types of cryptanalysis exist. The *ciphertext only* cryptanalysis refers to the case where only ciphertext is available for an attacker. The *known plaintext* cryptanalysis refers to the case where a <plaintext, ciphertext> pair is available. The *chosen plaintext* cryptanalysis refers to the case where the attacker introduces specific plaintext and obtains the corresponding ciphertext. The *chosen ciphertext* cryptanalysis refers to the case where the attacker chooses a ciphertext and decrypts it with an unknown key in such a scenario as an unattended decryption machine. The above four cases are roughly ordered by the amount of information available to the attacker.

Many concrete cryptanalysis attacks exist. *Frequency analysis* is the study of the frequency of letters or groups of letters in a ciphertext. The method is very useful against mono-alphabetic ciphers such as the Caesar cipher, which does not change the plaintext character frequency in the ciphertext. In English, "e" tends to be common, whereas "q" is rare. Likewise, "st," "ng," "th," and "er" are common pairs of letters. So if "m" has the highest frequency in the ciphertext of a mono-alphabetic cipher message, it is highly possible that ciphertext "m" corresponds to plaintext "e." *Linear cryptanalysis* and *differential cryptanalysis* are two widely applicable attacks on modern block ciphers.

DEFENSE MECHANISMS

We have discussed a variety of attacks. Various security systems and protocols have been designed to combat those attacks. In the following discussion, we introduce a few of them, which may achieve security goals such as confidentiality, integrity, authentication, and nonrepudiation to different extents.

Kerberos

Kerberos is an authentication system developed at MIT, which uses secret key cryptography. The system is available for both UNIX and Windows platforms. The Kerberos protocol is named after the three-headed dog, Kerberos, from Greek mythology. The protocol also consists of three parts: the Key Distribution Center (KDC), the client (also known as the *principal*), and the server with the service the principle wishes to access. The KDC maintains a centralized authentication mechanism and provides two functions: Authentication Service (AS) and Ticket-Granting Service (TGS).

Kerberos is based on the Needham–Schroeder Protocol with minor modifications. In Kerberos, if a user wants to use a service available on a target server, the authentication follows the following procedure:

1. AS Exchange: The client and KDC share a secret key, which can be derived from the client password hash. The client sends a request of Ticket to Get Tickets (TGT) with her name to KDC. KDC searches for the client name in the centralized database, and AS replies to the client request. The AS reply has two sections: a TGT (encrypted with a key known only to TGS) and a session key (encrypted with the shared key between the client and KDC) to handle future communications between the client and KDC.
2. TGS Exchange: If the client wants to use the service, she sends the TGT to TGS, which decrypts this TGT. If approved, a service ticket is generated by TGS and sent to the client. The service ticket has two portions: client portion and server portion, both containing the same secret for the client and the server. The client decrypts the client portion of the service ticket by using the TGS session key obtained from the earlier AS reply. The client blindly sends the server portion of the TGS reply to the target server.
3. Client/Server Exchange: The server decrypts the server portion of the service ticket from the client by using its own long-term key with the KDC. Then an authentication protocol such as the challenge-response protocol based on the secret key cryptography can be used to authenticate the client. A service session is then built between the server and client.

IPsec

IPsec (Internet Protocol Security) is a set of protocols that support secure communication at the network layer. It was developed by the International Engineering Task Force (IETF) (5), to provide "interoperable, high quality, cryptographically-based security" for IPv4 and IPv6. IPsec can provide security services such as access control, connectionless integrity, data origin authentication, anti-replay service, and data confidentiality.

IPsec can be run in two encryption modes: *transport mode* and *tunnel mode*. Transport mode encapsulates only each packet's payload and provides secure connections between two endpoints in a network, or an endpoint and a gateway, if the gateway serves as the destination host. Tunnel mode encapsulates the entire IP packet (including not only the payload but also the header) and thus provides a secure path between two gateways. Tunnel mode is used to deploy a Virtual Private Network (VPN) as a secure virtual tunnel that can be established across the untrusted Internet.

IPsec provides two security services: *Authentication Header (AH)* (6) and *Encapsulating Security Payload (ESP)* (7). AH provides authentication, integrity, and optional anti-replay services, whereas ESP may provide all of the above as well as confidentiality (encryption). AH and ESP can be used alone or in combination with each other to provide desired security services.

In AH protocol, an authentication header is inserted between the IP header and the higher layer protocol header such as TCP and UDP (transport mode), or between a new IP header and the original IP header (tunnel mode). The authentication header contains a cryptographic hash-based message authentication code over nearly all the fields of the IP packet.

The ESP header is inserted after the IP header and before the higher layer protocol header (transport mode) or before the encapsulated entire IP packet (tunnel mode).

Both ESP and AH rely on *security associations* (SAs), which are collections of connection-specific parameters that specify some shared secrets such as key, algorithm, and policies to use. These secrets are established to seed the authentication function and to key the encryption algorithm. SAs are stored in the Security Associations Database (SADB).

IPsec Key Management

Encryption and authentication keys are used in IPsec for encoding and decoding. The two parties using IPsec in their communication share and exchange the keys that their security protocols use. Key management is an essential and important issue for IP security. The primary security protocol that supports this purpose is called "Internet Key Exchange" (IKE).

IKE allows IPsec-enabled devices to exchange their security associations to populate their security association databases. IKE is considered a "hybrid" protocol because it combines three key management protocols: ISAKMP (Internet Security Association and Key Management Protocol), Oakley, and SKEME.

ISAKMP is a generic protocol that supports many different key exchanges. It also defines the procedures for authentication, creation and management of SAs, key generation, and provisions for DoS and replay attacks.

Oakley describes a specific mechanism for key exchanges through definition of various key exchange modes. The keys generated using Oakley might be used to encrypt data with a long privacy lifetime, e.g., 20 years or more. Oakley is used to establish a shared key with an assigned identifier and associated authenticated identities for the two parties.

SKEME uses a different key exchange mechanism than Oakley. It provides several modes to perform fast and frequent rekeying. SKEME provides scalability and flexibility to key exchanges.

Several other key management protocols have been proposed, such as SKIP and Photuris. SKIP (Simple Key-management for Internet Protocols) is a key management protocol for sessionless datagram-oriented protocols such as IPv4 and IPv6. Photuris is a session-key management scheme that is used with AH and ESP. Photuris is primarily used for creating VPNs, establishing sessions for mobile nodes over bandwidth-limited channels, and short-lived sessions between a great number of clients and servers.

IP Traceback

DoS has become a pressing problem for today's Internet. DDoS has even more impact than DoS because a DDoS attacker uses many compromised slave systems to increase the impact when attacking. Highly automated attack tools have been developed and used to falsify the source ID supplied in the IP packets (called IP spoofing) obscuring the origin of the attack. The problem of finding the source of an IP packet is called *IP traceback*, and most IP traceback approaches have targeted DoS attack detection.

A brute force solution to traceback is to have every router mark every packet or keep a record of each packet as it is transmitted. However, this solution is not feasible because of the storage space and performance overhead required.

Most existing IP traceback approaches try to store some information about packets either in routers along the way or in the packet itself, and to reduce space and communication overhead. Some approaches are probabilistic, and some are deterministic. These approaches fall into four categories: *packet marking, logging, link testing,* and *ICMP-based traceback*.

Packet marking approaches work by inserting traceback data into the packet to be traced, to mark the packet when it passes through routers on its way to the destination. Stefan Savage et al. (8) proposed a Probabilistic Packet Marking (PPM). In PPM, routers mark the packet with low probability (e.g., 1/20,000), with either the router's IP address or the edges of the path that the packet has traversed before reaching the router. When enough packets are received, all edges and all fragments will be collected to reconstruct the attack path. The low probability of marking reduces associated overhead. Several modified PPM approaches have also been proposed.

Logging is an intuitive solution to establish the true origin of attack traffic. It logs packets handled by key routers throughout the Internet and then uses data mining approaches to extract information about the attack path. This approach allows accurate analysis of attack traffic, but the amount of processing and storage space for the logs is very demanding.

Link testing methods work through hop-by-hop tracing. The testing is started from the victim, and upstream links are tested to determine which one carries the attack traffic. The testing is recursively repeated until it reaches the origin of the attack. Link testing can only be carried out while an attack is active.

The ICMP-based traceback was proposed by Steven Bellovin. It works by probabilistically sending an ICMP traceback packet to the destination with a low probability (say, 0.005%). These ICMP packets contain partial path information, including information that indicates the origin of the packet, the time when it was sent, and its authentication. The low probability suppresses the processing overhead and the bandwidth requirement.

While traceback approaches have been deployed, other efforts are made to restrict illegitimate packets, such as *ingress filtering*. Ingress filtering restricts spoofed packets at ingress points by blocking traffic except from authorized source networks that can use the router.

SSL/TLS

Secure Sockets Layer (SSL) and its successor, *Transport Layer Security* (TLS), are cryptographic protocols that provide secure communications on the Internet.

SSL was developed in Netscape Navigator in 1995 and is now used by both Netscape and Internet Explorer. Many web services use SSL to protect communications between clients and servers, especially when clients need to provide confidential information such as credit card numbers. An example of the protocols for this service is *https*. Numerous other SSL-enhanced protocols (e.g., SSLtelnet, SSLftp, or stunnel) also take advantage of SSL. Two versions of SSL, versions 2 and 3 (v1 was only used internally at Netscape and was never released), are commonly used, and v3 is rapidly replacing v2.

TLS was developed by the IETF, based on and extending SSLv3.0. It is not compatible with SSL. TSL 1.1 is the current approved version of TLS. TLS 1.1 is very similar to TLS 1.0, but version 1.1 uses a modified format of encrypted RSA premaster secret, which is done to prevent an attack found in TLS 1.0.

SSL/TLS runs on top of TCP and beneath application protocols such as HTTP, FTP, and SMTP. The protocols use two keys, a public key and a secret key. SSL/TLS provides security services such as authentication, confidentiality, and integrity. It has two layers: *record protocol* and *handshake protocol*. The record protocol ensures that communication privacy is protected by using symmetric encryption and ensures the communication is reliable. The handshake protocol allows the server to authenticate itself to the client with public key cryptography and then allows the negotiation of symmetric cryptographic keys before the transmission. The reason for using a combination of public key encryption and symmetric key encryption is that public key encryption provides better authentication, whereas symmetric key encryption provides better performance. The handshake protocol involves four phases:

1. *Hello*—the client sends a *clientHello* message specifying a list of cipher suites, compression methods, and the highest version it supports. Then the server chooses from among the connection parameters that the client has offered and sends the choices back in a *serverHello* message.
2. *Server key exchange and authentication*—the server sends a certificate and a server_key_exchange message. The currently used certificates are based on X.509, but specifications for the use of OpenPGP based certificates are also available.
3. *Client key exchange and authentication*—the client sends a certificate if the server asked and a client_key exchange message. A certificate_verify message is also sent.
4. *Finish*—client and server send wrap-up messages.

In SSL/TSL, authentication is provided only on the server's side. To provide mutual authentication, PKI (public key infrastructure) needs to be deployed at the client.

Firewalls

A firewall is a system that sits between a private network (or a computer) and the rest of the network and attempts to keep malicious traffic away from the private network. All traffic entering or leaving the private network must pass through and be examined by the firewall, which will block traffic that does not meet the specified security criteria. Firewalls can provide controlled access to network information and protect against risks such as DoS, unauthorized access, or modification of internal data. Firewalls cannot protect against internal traffic or traffic that routes around the firewall.

A firewall can be implemented in hardware or software, or a combination of both. It must be configured correctly to function properly. Generally speaking, three types of firewall techniques exist:

Filter—deployed at the ISO network layer. Two types of filtering exist: *packet filtering* and *session filtering*. In packet filtering, decisions to blocking or transmit is made on a per-packet basis. No state information is examined or maintained during the filtering. Therefore, the firewall does not know whether a packet belongs to an existing connection or is trying to establish a new one. This type of firewall is also called *stateless*. An example of packet filtering is Linux iptables. The firewalls that use session filtering are *stateful firewalls*. That is, they extract and maintain "state" information of connections. In session filtering, decisions are made based on the context of the connection. If a packet is a new connection, the firewall will check against security policy; if the packet is part of an existing ongoing connection, the firewall will look it up in a state table and an update table, which maintain the state information. Filtering is fairly effective and transparent to users, but it is susceptible to IP spoofing and difficult to configure.

Circuit-level firewalls (or gateways)—applies security policies when a TCP or UDP connection is established. Once the connection has been established, the packets of the connection will be allowed through without additional checking. An example of a circuit-level gateway is SOCKS.

Application firewalls (or gateways)—applies security mechanisms to specific applications, such as telnet, ftp, or http servers. An application firewall examines packets more thoroughly and therefore is considered more secure than a circuit-level firewall. But application firewalls cost more in terms of money and resources. Another disadvantage of application gateways is that they may not be applicable to all types of connections.

Both application firewalls and circuit-level firewalls use proxy servers, which sit between the two hosts or networks and intercept all messages passing through. External hosts will establish connections with the proxy server, and the proxy server performs communications with the internal hosts. Proxy servers can hide the topology of a private network so that the external hosts only see the IP address of the proxy server and can only communicate with the internal hosts through the proxy. However, transparent application gateways have been introduced, which means the internal hosts do not have to be aware of the existence of a proxy server or to run special software to communicate with the server.

Secure Email

To understand the necessity of securing e-mail, we need to review how an e-mail message is sent and received. When a sender sends an e-mail, the sender's e-mail client software such as Thunderbird uses *Simple Mail Transport Protocol* (*SMTP*) to contact the sender's SMTP server. The sender's SMTP server relays the e-mail to the recipient's SMTP server, which delivers the e-mail to the inbox of the right e-mail account. The e-mail may be stored on any intermediate SMTP servers for later forwarding. Then a recipient uses *Post Office Protocol* (*POP*) or *Internet Message Access Protocol* (*IMAP*) to download a message stored at the recipient's SMTP server. In the case of webmail, the sender and recipient communicate to their respective SMTP servers through a *webmail server*. A sender first uses the HTTP protocol to put messages on the webmail server, which contacts its SMTP server for delivery. The recipient may have a webmail server, which uses POP/IMAP to download the user message to the webmail server. Then a recipient may use HTTP to get access to e-mail messages.

Normal e-mail messages are transmitted on the wire and stored at intermediate servers in cleartext. SMTP, POP, IMAP, and webmail may ask a user to input user name and password, which may lead to identity theft. To protect the user name and password, SSL should be used between a user and the corresponding SMTP server. Most modern SMTP servers and e-mail client software provide this capability. In the case of webmail, HTTPS should be used.

SSL only protects the e-mail path between a user and the corresponding SMTP server. Beyond that, the e-mail is still stored and transmitted in cleartext. To provide the end-to-end content protection, we may use S/MIME and PGP. Both protocols use public key cryptography. Each user has a key pair <public key, private key>. If Alice wants to send a message to Bob, Alice uses Bob's public key to encrypt the message and Bob uses his private key to decrypt the message. E-mail content confidentiality is maintained in this way. Alice may also use her own private key to create a hash of the message in order to use as a digital signature to her e-mail. Then Bob can use Alice's public key to decrypt the signature (encrypted email hash), compute his own version of the e-mail hash, and compare these two hashes. If they match each other, the message integrity is verified. This process is called *signature verification*. Such signatures also support authentication of Alice and nonrepudiation.

S/MIME is built into many e-mail clients like Microsoft Outlook, but a certificate needs to be bought from a third-party company such as Thawte.com or Verisign.com. PGP is open source and available for free. Enigmail is an extension to the mail client of Mozilla/Netscape and Mozilla Thunderbird, which allows users to access the authentication and encryption features provided by GnuPG for secure e-mail.

Virtual Private Networks (VPNs)

A VPN is a secure tunnel from a remote site, through the Internet, to the user's home network. When a VPN client logs onto a VPN server within a domain, the client computer will work as though it is in the same domain as the server. A VPN server can act as a gateway into a whole network or to a single computer. It listens for VPN clients attempting to connect to it. Using VPN, we can transparently integrate several physically separate working systems on the Internet as a single (virtual rather than physical) local network.

A VPN client may communicate with a VPN server with either of two protocols: the Point-to-Point Tunneling Protocol (PPTP) or Layer Two Tunneling Protocol with Internet Protocol security (L2TP/IPSec). PPTP has good encryption coupled with the function of user authentication. IPSec is safer because of its sophisticated encryption schemes but does not include authentication routines. L2TP is IPSec with authentication built in.

There are VPN servers for both Linux and Windows. Linux uses *iptables* and other software packages. Setting up a VPN server on a Windows XP Professional is straightforward. Many VPN client software packages exist.

Intrusion Detection

According to Amoroso in *Wykrywanie intruzów*, "intrusion detection is the process of identifying and responding to malicious activity targeted at computing and networking resources." An intrusion detection system (IDS) can be classified based on different criteria. Based on where an IDS is positioned, there are host based IDS (HIDS) residing on a single host and protecting that host, network based IDS (NIDS) monitoring an entire network segment, and perimeter IDS residing on a gateway or edge router and monitoring traffic between networks, usually an intranet and the Internet.

An IDS can also be classified based on intrusion detection approaches. An IDS may use anomaly detection, so host or network behaviors deviating from normal daily routine may be identified as suspicious. An IDS may also use attack signatures for detection. Packets and software for attacks often have unique patterns, such as special code that may cause a specific buffer overflow. Such patterns serve as the signature of an attack. Different attacks have different signatures. An IDS using signatures maintains a database of those signatures. An IDS may analyze audit trails, log files, processes, and network traffic for anomaly detection or signature detection.

There are a lot of commercial and open-source IDS tools such as Internet Security Systems' RealSecure IDS, Cisco's Secure IDS, and snort. Snort is a popular network intrusion open-source package. It allows a user to specify a set of rules that include the patterns to be detected in packets, along with corresponding IDS actions such as an alert for matched packets. A large database of rules for known attacks is included. Snort also allows the user to create custom plug-ins to extend detection beyond that available through the default pattern matching. Snort is used by many other packages and products. A packet passes through phases in Snort's detection engine: packet acquisition, packet decoder, preprocessor, detection engine, and intrusion report. Snort is often coupled with a database for storing alert data and an interface such as the Basic Analysis and Security Engine (BASE) for user-friendly alert display and data management.

Digital Forensics

Digital forensics involves obtaining and analyzing digital information for use as evidence in civil, criminal, and administration cases. There are a few phases during a digital forensics process (9): (*1*) Notification: an incident is detected, and the response team is informed; (*2*) preservation: make an exact copy of the digital crime scene; (*3*) survey: examine the crime scene for obvious pieces of digital evidence; (*4*) search: a more thorough search for additional evidence to support or refute hypotheses; (*5*) reconstruction: test the existing evidence and hypotheses to form a final theory; and (*6*) presentation: the final theory is presented to the parties requesting the investigation.

Digital forensics includes computer forensics and network forensics. Computer forensics is concerned with recovering, searching, and preserving digital evidence from floppy disk, hard disk, memory, CDs, and other media. The task of network forensics includes analysis of network traffic for violation evidence and traceback to the attacker. This task is where network forensics differs from intrusion detection, which focuses on detection of intrusions. For example, in the case of an e-mail Trojan horse, intrusion detection is concerned with detection and thwarting such a threat, whereas network forensics is also concerned with finding the source of the malicious e-mail.

Digital forensics is an active frontier for cyber security. Researchers and companies have been developing sophisticated software for safely preserving and recovering evidence from digital data. Tools include AccessData's Forensic Toolkit (FTK), Digital Intelligence's Encase Forensic Edition, and X-Ways Forensic Addition. These tools provide an integrated environment for recovering a variety of evidence such as deleted files from a storage media.

WIRELESS NETWORK SECURITY

Wireless technology provides a user the capability of communication with great flexibility and freedom. Wireless networks have been rapidly extending their capabilities and are becoming the communication infrastructure of choice. Wireless communication channels are also interoperable with the traditional Internet. With the increasing use of wireless technology, the security of wireless networks has become a serious concern.

The risk to users of wireless networks has been increasing exponentially as the service becomes more and more popular. Any security threats that exist in conventional wired networks also exist for wireless networks. Wireless networks use open shared media, and in many cases, communication is broadcast, making wireless networks more vulnerable to attacks such as eavesdropping, DoS (including signal jamming on communication channels and bogus requests and messages injection at the network level), identity theft, masquerading, and unauthorized access to wireless devices or networks. Besides these threats, malicious entities can also intrude on the privacy of legitimate users and track their physical movements.

Wireless networks are usually categorized into three types based on their coverage range: Wireless Wide Area Networks (WWANs), Wireless Local Area Networks (WLANs), and Wireless Personal Area Networks (WPANs). WWANs include wide area technologies such as 2G cellular, Global System for Mobile Communication (GSM), Cellular Digital Packet Data (CDPD), and Mobitex.

The IEEE 802.11 standard, the original WLAN standard, was first developed in 1997 to support medium-range, higher data rate applications and to address mobile and portable stations. The standard uses WEP and WPA to protect it security.

Wired Equivalency Privacy (WEP) was the original encryption standard for wireless communications. WEP comes in different key sizes. The commonly used key lengths are 128 and 256 bits. WEP intended to make wireless networks as secure as wired networks, but security flaws have been discovered and exploited. A demonstration held by a group from the FBI showed that publicly available tools can be used to break a WEP protected network, and it took only three minutes. WEP protection is better than nothing, but deployment of WPA encryption can be more secure.

WPA stands for *Wi-Fi Protected Access*. It is an early version of the 802.11i security standard and was developed by the WiFi Alliance to replace WEP. WPA has two improvements over WEP: improved data encryption through the temporal key integrity protocol (TKIP), and it provides user authentication, which is generally missing in WEP, through extensible authentication protocol (EAP).

Bluetooth is an industrial specification for WPAN. Bluetooth dynamically connects remote devices such as PDAs, cell phones, and laptops. Bluetooth provides security services such as authentication, confidentiality, and authorization. As with the 802.11 standard, Bluetooth does not address other security services such as nonrepudiation or audit. Bluetooth offers several security modes, and device manufacturers determine which mode to include in a Bluetooth-enabled device.

Besides wireless cellular networks that rely on an infrastructure of non-mobile access points (such as base stations), wireless networks without infrastructure, a mobile ad hoc network (MANET), have also been widely studied. MANETs are defined as peer-to-peer networks between mobile devices that do not have an access point in between. Mobile ad hoc networks are characterized by the absence of a fixed infrastructure, rapid topology change, and high node mobility. Absence of infrastructure makes ad hoc networks more difficult to secure, because it is hard to deploy control points. Additionally, limits on energy consumption, computation resources, and bandwidth force ad hoc network security mechanisms to be lightweight in both computation and communication. Asymmetric cryptography is usually considered too expensive for MANETs. Symmetric cryptographic algorithms and one-way functions are commonly used to protect data integrity and confidentiality.

A wireless sensor networks (WSN) is a large-scale mesh network that consists of a great number of small sensor nodes communicating via radio. WSNs can be applied in areas such as military, home, and health. Compared with ad hoc networks, WSNs tend to have even more rapidly changing topology, even severe constraints on power, computation, and space. Communications in sensor networks are usually broadcast, whereas most communication in an ad hoc network is point-to-point. End-to-end encryption is impractical in sensor networks. Usually hop-by-hop encryption mechanisms are used, in which sensor nodes store encryption keys with their immediate neighbors.

ACKNOWLEDGMENT

The book chapter is a brief survey of network security fundamentals. During the writing, we have referred to many online sources, articles, and books. We thank those authors for their brilliant work.

BIBLIOGRAPHY

1. B. Kaliski, The MD2 Message-Digest Algorithm (Request for Comments: 1319), 1992. Available: http://www.ietf.org/rfc/rfc1319.txt.
2. D. Eastlake, 3rd, P. Jones, US Secure Hash Algorithm 1 (SHA1) (Request for Comments: 3174), 2001. Available: http://www.ietf.org/rfc/rfc3174.txt.
3. K. Zeilenga, LDAP Authentication Password Schema (Request for Comments: 3112), 2001. Available: http://www.ietf.org/rfc/rfc3112.txt.
4. M. Kaeo, *Designing Network Security, Second Edition*, Indinapolis: 2003.
5. S. Kent, R. Atkinson, Secutiry Architecture for the Internet Protocol (Request for Comments: 2401), 1998. Available: http://www.ietf.org/rfc/rfc2401.txt.
6. S. Kent, R. Atkinson, IP Authentication Header (Request for Comments: 2402), 1998. Available: http://www.ietf.org/rfc/rfc2402.txt.
7. S. Kent, R. Atkinson, IP Encapsulating Security Payload (ESP) (Request for Comments: 2406), 1998. Available: http://www.ietf.org/rfc/rfc2406.txt.
8. S. Savage, D. Wetherall, A. Karlin, T. Anderson, Practical network support for IP traceback, in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm, Sweden, 2000*, pp. 295–306.
9. B. D. Carrier and J. Grand, A hardware-based memory acquisition procedure for digital investigations, *Journal of Digital Investigations*, **1**: 2004.

FURTHER READING

C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 2002.

R. Russell (ed.), D. Kaminsky, R. F. Puppy, J. Grand, K2, D. Ahmad, H. Flynn, I. Dubrawsky, S. W. Manzuik, and R. Permeh, *Hack Proofing Your Network*, 2nd ed. New York, Syngress, 2002.

J. Beale, A. R. Baker, B. Caswell, and M. Poor, *Snort 2.1 Intrusion Detection*, 2nd ed. New York: Syngress, 2004.

C. Prosise, K. Mandia, M. Pepe, *Incident Response and Computer Forensics*, 2nd ed. New York: McGraw-Hill, 2003.

G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code.* Reading, MA: Addison-Wesley, 2004.

J. Koziol, D. Litchfield, D. Aitel, C. Anley, S. “noir” Eren, N. Mehta, and R. Hassell, *The Shellcoder’s Handbook: Discovering and Exploiting Security Holes.* New York: Wiley, 2004.

D. A. Wheeler, Secure Programming for Linux and Unix HOWTO, 2006. Available: http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/.

W. G. Kruse II and J. G. Heiser, *Computer Forensics: Incident Response Essentials*, Reading, MA: Addison-Wesley, 2002.

STEVEN GRAHAM
XINWEN FU
Dakota State University
Madison, South Dakota
BIN LU
West Chester University of
Pennsylvania
West Chester, Pennsylvania

OPTICAL COMMUNICATION

Optical communication is a form of telecommunication that uses light as the transmission medium.

For most of human history, long-distance communication posed many challenges. Optical communication in its primitive form played a crucial role. In ancient China, the beacon towers on hilltops of the Great Wall that spans more than 4000 miles often played a key role in military communication in ancient war times. Once the enemy pressed toward the border, the signal from the beacon tower would be sent by beacon (fires or lanterns) during the night or by smoke signals in the daytime. When the city of Troy fell, the ancient Greeks learned the news from a system of fire beacons on adjacent islands that carried a prearranged signal nearly 400 miles. In the eighteenth century, a French visionary named Claude Chappe built a series of towers, each adorned with giant arms that could be clearly seen with a telescope from adjacent towers. These arms would be positioned differently to represent the various letters of the alphabet, and in this fashion, messages could be passed from tower to tower and across the whole of France. Over 100 years ago, Alexander Graham Bell invented the "photophone," an ingenious system that was beyond his time for sending sound on a light beam. This important invention is recognized as the progenitor of the modern fiber optical communication that carries an increasing amount of the world's telecommunication traffic.

Today, optics-enabled visual communication is still widely used in our daily lives. For example, aircrafts use the landing lights at airports to land safely, especially at night and under adversary weather conditions. Aircrafts landing on an aircraft carrier use a similar system to land correctly on the carrier deck. Ships often use a signal lamp to signal in Morse code or use international maritime signal flags to exchange messages. Distress flares are used by sailors in emergencies, whereas lighthouses and navigation lights are used to communicate navigation hazards.

In general, an optical communication system consists of a transmitter or a light source that encodes a message into an optical signal, a channel, or a waveguide that carries the signal to its destination, and a receiver that reproduces the message from the received optical signal. The modern guided optical communication is based on the light's total internal reflection principle, explained by the Snell's Law, which has been known for centuries and was used to illuminate streams of water in elaborate public fountains in Victorian times. The development of modern optical communication is indebted to technological breakthroughs and improvements realized in several areas: light source—LED and laser, materials and the manufacturing of low-loss lightwave guides—optical fibers, and other components and devices essential for effective optical transmission and communication as well as sophisticated electronics and signal processing techniques.

Optical fiber offers much higher bandwidths [nearly 50 terabits per second (Tb/s)] than copper cables and is less susceptible to various kinds of electromagnetic interferences and other undesirable effects. It possesses a lot of nice properties: low signal attenuation (as low as 0.2 dB/km), low signal distortion, low power requirement, low material usage, small space requirement, and relatively low cost. Optical fiber transmission has played a key role in increasing the bandwidth of telecommunications networks, especially in the last 20 years as the Internet has penetrated our daily lives. In the first-generation optical networks, optical fiber was used purely as a transmission medium, serving as a replacement for copper wires, and all the switching and processing of the bits were handled by electronics. These optical networks are widely deployed today in all kinds of telecommunications networks. Examples of first-generation optical networks are SONET (synchronous optical network) and SDH (synchronous digital hierarchy) networks, which form the core of the telecommunications infrastructure in North America and in Europe and Asia, respectively, as well as a variety of enterprise networks. Incorporating some switching and routing functions that were performed by electronics into the optical part of the network, the second-generation and future generation optical networks are capable of providing more functions than simple point-to-point transmission, for example, lightpath service, dynamic service provisioning, and so on. Optical communication networks will essentially serve as optical transport networks (OTNs) that enable everything over optics integration, e.g., IP over WDM (wavelength division multiplexing) integration, leading to the building of the next-generation optical Internet.

The applications of optical communication are abundant, including short-distance interconnection inside computers, digital optical audio cables, optical transport networks, backhaul networks that carry traffic and network control information between wireless base stations and other network elements, medical applications (such as gastroscope, endoscopy, and laparoscopic surgery), industrial fiberscope or borescope for inspecting anything hard to reach (such as jet engine interiors), and fiber-to-the-curb (FTTC) or fiber-to-the-home (FTTH) as a solution to the first-mile problem.

There are two main types of optical communications based on the medium or optical waveguide that optical signals traverse: (1) fiber optical communication and (2) free-space optical communication.

THE BASIC COMPONENTS

The research on optical communications was started substantially in 1960 by the invention of the laser and followed by studies simultaneously in three facets: light source, transmission line, and light detector. The optical communication system is composed of optical fiber to transmit light, laser to emit light, photodiode to detect light, and

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

various optic components to control the flow of light signals. In particular, the optical fiber has provided significant motive power to the growth of optical communications, because of its low loss and enormous capacity of signal transmission in the single mode fiber, excellent mechanical properties such as small diameter, and adequate tensile and bending strength.

Optical Fiber

The optical fiber works on the principle of total internal reflection. Total internal reflection is an optical phenomenon that occurs when light crosses materials with different refractive indices. When the light in the material with a larger refractive index strikes the medium boundary with a large enough angle of incidence with respect to the norm, the light will stop crossing the boundary altogether and instead totally reflect back internally. The development of optical fiber as the waveguide for effective signal transmission did not occur until the 1950s. At that time, researchers focused on the development of fiber bundles for image transmission, primarily for medical applications such as flexible gastroscopes. The first fiber optic semiflexible gastroscope was invented by Basil Hirschowitz, C. Wilbur Peters, and Lawrence E. Curtiss, researchers at the University of Michigan in 1956. Initially, optical fibers had relied on air as the low-index cladding material. During the development of the gastroscope, simply bundling a large number of such optical fibers together resulted in the loss of total internal reflection as well as the image transmitted. This loss prompted Curtiss to invent the first glass-clad fibers that use a layer of low-index glass as the cladding material that wraps the high-index glass core (Fig. 1).

However, fiber attenuation that light dims significantly after passing through just a few meters of glass was a big problem. In 1965, Charles K. Kao and George A. Hockham, then of British company Standard Telephones and Cables, first recognized that attenuation of contemporary fibers was caused by impurities in glass rather than by fundamental physical effects such as scattering, and they suggested how signal loss could be greatly reduced so that optical signals could run for kilometers instead of just a few meters. In a paper published in 1966, they reported, in concrete terms, on the potential of massive optical communications, through estimation of transmission capacity of optical fiber and transmission distance derived from the magnitude of expected loss and acceptable light power. They demonstrated that optical fiber could be a practical medium for communication if the attenuation could be reduced below 20 dB per kilometer. Subsequently, researchers Robert D. Maurer, Donald Keck, Peter Schultz, and Frank Zimar of American glass maker, Corning, Inc., developed a low-loss optical fiber with 17-dB/km optic attenuation in 1970 by doping silica glass with titanium, to make a long stride to the commercialization of low-loss, large-capacity optical fiber communication. In 1977, General Telephone and Electronics (GTE) sent the first live telephone traffic through fiber optics in Long Beach, California. The first transatlantic telephone cable using optical fiber went into operation in 1988.

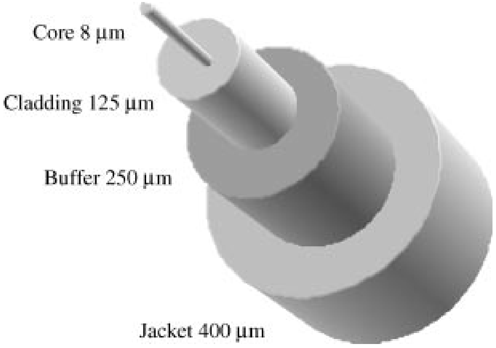


Figure 1. A typical single-mode optical fiber with diameters of the component layers.

Optical fiber is a composite material constructed of two layers of glass or plastic (Fig. 1). Typically optical fiber consists of a silica-based core and cladding surrounded by one or two layers of polymeric material that provides mechanical protection of the glass surface. The inner layer, the core, has a higher refractive index than the outer layer, the cladding. Light injected into the core and striking the core-to-cladding interface at greater than the critical angle will be reflected back into the core. Two basic types of fiber exist: multimode fiber and single-mode fiber. Figure 1 shows a typical single-mode fiber with component layers. A mode is a path that a light signal takes through a fiber. Modes result from the fact that light will only propagate in the fiber core at discrete angles within the cone of acceptance.

Multimode Fiber

A multimode fiber is one in which the guided light ray takes different paths through the fiber. Multimode fiber, the first to be manufactured and commercialized, is best designed for short transmission distances. This type of fiber has a core diameter from 50 μm to 85 μm, much larger compared with single-mode fiber, and therefore it is easier to couple light into the core of multimode fiber than of single-mode optical fiber. The most common multimode fiber deployed in the late 1970s and early 1980s had a core diameter of 50 μm. The large core allows many modes or rays of light to propagate. Therefore, light entering the fiber at the same time may arrive at the other end at slightly different times, which results in the light being spread out, a phenomenon called *modal dispersion*(1). The effect of modal dispersion is that the signal (i.e., the digital pulse) becomes smeared as it travels down the fiber. Today, multimode fiber is used almost exclusively for private premises systems in which distances are usually less than 1 km.

Single-Mode Fiber

A way to eliminate modal dispersion is to reduce the core's diameter until the fiber will propagate only one mode efficiently; i.e., the energy of the light signal travels in the form of one mode. A single-mode fiber is one in which the guided light ray takes one path through the fiber. The diameter of single-mode fiber is only 8–10 μm. Single-mode fiber allows for a higher capacity to transmit information because it can retain the fidelity of each light pulse over longer distances and exhibits no dispersion caused by multiple modes. Single-mode fiber also enjoys lower fiber attenuation than multimode fiber. Like multimode fiber, early single-mode fiber was generally characterized as step-index fiber, which means the refractive index of the fiber core is a step above that of the cladding rather than graduated as it is in graded-index fiber. Modern single-mode fibers have evolved into more complex designs such as matched clad, depressed clad, and other exotic structures. Because of the smaller core diameter, coupling light into the core of single-mode fiber is more difficult. The requirements for other single-mode connectors and splices are also much more demanding. Single-mode fiber suffers from *chromatic dispersion* (1) in that different wavelengths of light travel at different speeds in the fiber, or the spectral components of the pulse travel at different speeds, which makes the optical signal smeared as it travels down the fiber.

Three basic classes of single-mode fiber are used in modern telecommunications systems. The oldest and most widely deployed type is *non-dispersion-shifted fiber* (NDSF). These fibers were initially intended for use near 1310 nm because of zero-dispersion property in the 1310-nm region. Later, 1550-nm communication systems made NDSF fiber undesirable because of its very high chromatic dispersion at the 1550 nm wavelength. To address this shortcoming, fiber manufacturers developed *dispersion-shifted fiber* (DSF) that exhibits zero dispersion in the 1550-nm region. However, when multiple, closely spaced wavelengths in the 1550-nm region were transmitted in the dense wavelength division multiplexing (DWDM) systems, DSF exhibits serious nonlinearities as data rates, power, and number of channels increase. A new class of fibers, *non-zero-dispersion-shifted fibers* (NZ-DSF), was developed to address the problem of nonlinearities. The fiber is available in both positive and negative dispersion varieties and is rapidly becoming the fiber of choice in new fiber deployment, especially for DWDM systems.

Although most fibers are made of silica glass, low-cost plastic fibers can be made of polymers that are cheap materials and allow a simple production by extrusion. Plastic optical fibers have found widespread applications in consumer markets (e.g., home networks), the automotive and aircraft industry, and so on.

One of optical fiber's key performance properties is fiber attenuation that characterizes the decay of light signal's strength as it travels down the fiber. Figure 2 shows the spectral attenuation performance of typical silica-based multimode fiber and single-mode fiber. Notice the two low-loss wavelength windows: 1310-nm and 1550-nm regions. The 1550-nm window is preferred for long-haul transmission because of its low attenuation, width, and the availability of optical amplifiers.

Light Source

Light is part of the electromagnetic spectrum. In optical communication and networking, the practice is to use wavelength rather than frequencies to define an optical channel. The wavelength is usually measured in nanometers (nm) or micrometers (μm). The relationship between frequency and wavelength of a signal is given by Frequency (in Hertz) = Speed of light in a vacuum (in meters/second) / Wavelength (in meters). Therefore, the higher the frequency of the signals, the shorter the wavelengths. For example, a frequency of 192.1 THz operates with a wavelength of 1560.606 nm, whereas a frequency of 194.7 THz operates with a wavelength of 1539.766 nm. Three low-loss windows in the 0.8-, 1.3- and 1.55-μm infrared wavelength bands are used for optical communication because optical fibers transmit infrared wavelengths with less attenuation and dispersion.

Many different types of light sources can be used for optical communication. The most important one is the laser. A laser (light amplification by stimulated emission of radiation), first demonstrated in 1960, is an optical source that emits photons in a coherent beam (1). The early lasers were multilongitudinal mode (MLM) Fabry-Perot lasers. These MLM lasers emit light over a fairly wide spectrum of several nanometers to tens of nanometers. The actual spectrum consists of multiple discrete spectral lines, which can be thought of as different longitudinal modes, hence, the term MLM. For high-speed optical communication systems, the spectral width of the source must be as narrow as possible to minimize the effects of chromatic dispersion. Likewise, a narrow spectral width is also needed to minimize cross-talk among channels in WDM systems. A single-longitudinal mode (SLM) laser emits a narrow single-wavelength signal in a single spectral line that reduces the spectrum of the transmitted optical signal

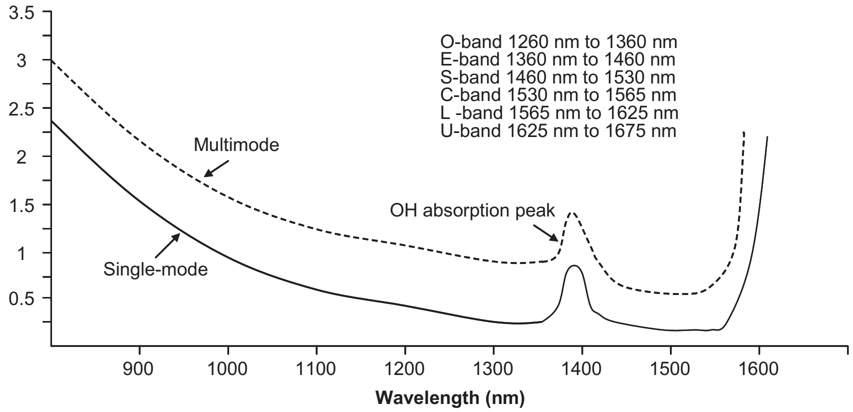


Figure 2. Fiber spectral attenuation performance.

to close to its modulation bandwidth. The penalty from chromatic dispersion is significantly reduced. Lasers have been used as transmitters and to pump or power optical signal amplifiers. Semiconductor lasers are the most popular light sources for optical communication systems.

A distributed-feedback (DFB) laser is a laser where the whole cavity or resonator consists of a periodic structure, which acts as a distributed reflector in the wavelength range of laser action, and contains a gain medium that compensates for the resonator losses. The DFB laser structure, which excels in wavelength precision, ensures stable single-wavelength oscillations of laser emissions. DFB lasers are required in almost all high-speed transmission systems today.

A tunable laser is a device that can tune over a range of wavelengths. The following tuning mechanisms are typically used: (*1*) Injecting current into a semiconductor laser causes a change in the refractive index of the material, which in turn changes the lasing wavelength; (*2*) temperature tuning; and (*3*) mechanical tuning that is used in lasers that use a separate external cavity mechanism. Some successful types of widely tunable lasers are the superstructure grating distributed Bragg reflector laser (SSGDBR), the grating-assisted codirectional coupler with sampled grating reflector laser (GCSR), and the sampled grating DBR laser (SGDBR) (2). All of these devices are capable of continuous tuning ranges greater than 40 nm. Another way to obtain a tunable laser source is to use an array of wavelength-differentiated lasers and turn one of them on at any time; e.g., an array of DFB lasers can be fabricated, each of them at a different wavelength. Tunable lasers are highly desirable components for WDM networks because fewer lasers are needed to support a multichannel WDM system and network operators need to stockpile fewer spare ones in the event that transmitters fail in the field and need to be replaced. Tunable lasers are also one key enabler of reconfigurable optical networks as well as for optical packet-switched networks, where data need to be transmitted on different wavelengths on a packet-by-packet basis.

Light-emitting diodes (LEDs) (1,3) provide a cheaper alternative to laser in many applications where the communication data rates are low and distances are short. An LED is a forward-biased *pn*-junction in which the recombination of the injected minority carriers (electrons in the *p*-type region and holes in the *n*-type region) by the spontaneous emission process produces light. The light output of an LED has a fairly broad *continuous* spectrum of several nanometers to tens of nanometers. LEDs are not capable of producing high-output powers, and typical output powers, are on the order of −20 dBm. They cannot be directly modulated at data rates higher than a few hundred megabits per second. A laser provides higher output power than an LED and therefore allows transmission over greater distances.

Detector

A receiver converts an optical signal into a usable electrical signal. Photodetectors perform the opposite function of light emitters (1,3). The most common detector is the semiconductor photodiode, which produces current in response to incident light. Detectors operate based on the principle of the *pn*-junction. An incident photon striking the diode gives an electron in the valence band sufficient energy to move to the conduction band, which creates a free electron and a hole. If the creation of these carriers occurs in a depleted region, the carriers will quickly separate and create a current. As they reach the edge of the depleted area, the electrical forces diminish and current ceases. Because the *pn*-diodes are insufficient detectors for fiber optic systems, to improve the efficiency of the photodetector, both *p-i-n*(PIN) photodiodes and avalanche photodiode (APDs) are designed to compensate for the drawbacks of the *pn*-diode. Once the detector converts optical signals back into electrical currents proportional to the incident optical power, the currents are then amplified to a usable level and fed to the decision circuit of the digital communication system that estimates the data bits from the electrical currents received. This process depends on the modulation schemes used at the transmitters.

Modulation

The process of imposing data on the optical signal is called modulation. The most widely used modulation scheme for optical communication is called on-off keying (OOK), where the light signal is turned on or off, depending on whether the data bit is 1 or 0. Other forms of modulation include return-to-zero (RZ) modulation, phase shift keying (PSK) modulation, frequency shift keying modulation, and so on. OOK modulation can be realized in two ways: (*1*) by direct modulation of a semiconductor laser or an LED, where the drive current to the light source is turned on or off based on whether the data bit is 1 or 0; and (*2*) by using an external modulator. Direct modulation is simple; its application is, however, limited to certain types of lasers. Direction modulation will result in a phenomenon wherein the carrier frequency of the transmitted pulse varies with time, which causes a broadening of the transmitted spectrum. In external modulation, an OOK external modulator is placed in front of a light source and turns the light signal on or off based on the data to be transmitted, whereas the light source itself is continuously operated. External modulators become essential in transmitters for communication systems using other forms of modulation, e.g., RZ modulation. Two types of external modulators are widely used today: lithium niobate modulators and semiconductor electroabsorption (EA) modulators. A summary of reported modulator results is provided in Ref. (4), which takes into account design considerations and applications. In Ref. (5), a tandem electroabsorption modulator with an integrated semiconductor optical amplifier is developed that is capable of both non-return-to-zero and return-to-zero data transmission at 40 Gb/s.

Optical Fiber Amplifier

One milestone in the evolution of optical fiber communication systems was the development of erbium-doped fiber amplifiers (EDFAs) in the late 1980s and early 1990s, which are capable of amplifying signals at many wavelengths in the 1550-nm window simultaneously in the optical domain and therefore reduces the cost of long-distance fiber systems by eliminating the need for optical-electro-optical regenerators. This technology enables the transmission capacity of optical communication systems to be significantly increased by using multiple wavelength channels simultaneously through wavelength division multiplexing (WDM). Under WDM, the optical transmission spectrum is carved up into several nonoverlapping wavelength bands, with each wavelength supporting a single communication channel. WDM provides the ability to turn on capacity quickly by lighting up new wavelengths in fibers already deployed. WDM systems with EDFAs are widely deployed today and are achieving capacities over 1Tb/s on a single fiber. Other types of optical amplifiers exist, each suitable for a spectral range; e.g., thulium-doped fiber amplifiers can be used for amplification in the *S* band around 1460–1530 nm, praseodymium-doped fiber amplifiers are for the 1.3-μm window, neodymium and ytterbium fiber amplifiers are for 1-μm laser sources, and Raman amplifiers can potentially generate gain in very different large wavelength regions.

Other Components

Various other optic components have been developed to control the flow of light signals. Optical filters are devices for selecting wavelengths from optical signals. Optical filters are essential components to construct multiplexers and demultiplexers used in WDM terminals wherein multiplexers combine wavelength channels into a single optical signal before transmission and demultiplexers extract individual wavelength channels from the optical signal after reception. Multiplexers and demultiplexers can be cascaded to realize static wavelength crossconnects (WXCs). The device routes signals from an input port to an output port based on the wavelength. Dynamic WXCs can be constructed by combining optical switches with multiplexers and demultiplexers such that wavelengths of signals from an input port can be dynamically selected and routed to an output port. Many different technologies are available to realize optical switches (1): (*1*) mechanical switches, e.g., using a mirror arrangement whereby the switching state is controlled by moving a mirror in and out of the optical path; (*2*) two-/three-dimensional (2-D/3-D) micro-electro-mechanical system (MEMS) switches (6,7); (*3*) bubble-based waveguide switches; (*4*) liquid crystal switches; (*5*) electro-optic switches; and (*6*) thermo-optic switches. Among the various technologies, the 3-D-MEMS beam steering mirror technology offers the best potential for building large-scale optical switches.

Optical switches are used inside WXCs to reconfigure them to provision lightpath, a circuit-switched end-to-end optical channel. Multiplexers and demultiplexers are components for constructing wavelength add/drop multiplexers (WADMs), devices used in WDM systems for mixing and routing different channels of light into or out of a single mode fiber. “Add” and “drop” refer to the capability of the device to add one or more new wavelength channels to an existing multiwavelength WDM signal, or to remove one or more channels, routing those signals to another network path. A wavelength converter is a device that converts data from one incoming wavelength to another outgoing wavelength. Wavelength converters can be used to improve the utilization of the available wavelengths on the network links.

OPTICAL FIBER COMMUNICATION

Evolution of Optical Fiber Transmission System

The construction of a communication network, which has and will continue to bring forth an extensive social innovation from the past decades to this new millennium, owes very much to the recent progress in the communication technologies. Above all, the development of optical fiber communication technology, which allows transmitting a large quantity of information over longer distance at reduced cost, has intensively promoted the progress. The evolution of an optical communication system has gone through several generations (1,8) (Fig. 3).

Early systems of the late 1970s through the early 1980s used LEDs or MLM laser transmitters in the 0.8- and 1.3-μm wavelength bands and multimode fibers, which enables the signal to be transmitted for a reasonable distance before the signal needs to be regenerated every few kilometers (e.g., 10 km) through an optical-electro-optical regeneration process. During regeneration, the receiver converts the incoming optical signal to an electrical signal. The signal is amplified, reshaped by sending it through a logic gate, and retimed. Retiming is a bit-rate-specific function. This signal is then modulated and retransmitted using a light source. This regeneration with reshaping and retiming completely resets the effects of nonlinearities, fiber dispersion, and amplifier noise; moreover, it does not introduce additional noise. Regenerators were expensive devices and continue to be expensive today. The distance between regenerators is limited because of attenuation and modal dispersion. These early systems operated at bit rates ranging from 32 to 140 Mb/s. Such systems are still used for low-cost computer interconnection at a few hundred megabits per second over a few kilometers, e.g., fiber channels for connecting computer servers to shared storage devices and for interconnecting storage controllers and drives.

The next generation of systems deployed starting around 1984 used MLM lasers in the 1.3-μm wavelength band and single-mode fiber that eliminates modal dispersion, therefore dramatically increasing the bit rates and distances possible between regenerators. Typically, the regenerator spacing in these systems is about 40 km or higher, limited primarily by fiber attenuation, and a few hundred megabits per second of bit rates are achieved. With improved optical fiber—single-mode fiber and laser as the transmitters—in the late 1980s, a new generation of systems in the 1.55-μm lower loss wavelength window was deployed. This generation increased the span between regenerators further. However, the data bit rates were limited by chromatic dispersion, which did not exist in the 1.3-μm band. This effect was offset by the development and deployment of dispersion-shifted fibers as well as by using SLM lasers that transmit pulses with significantly

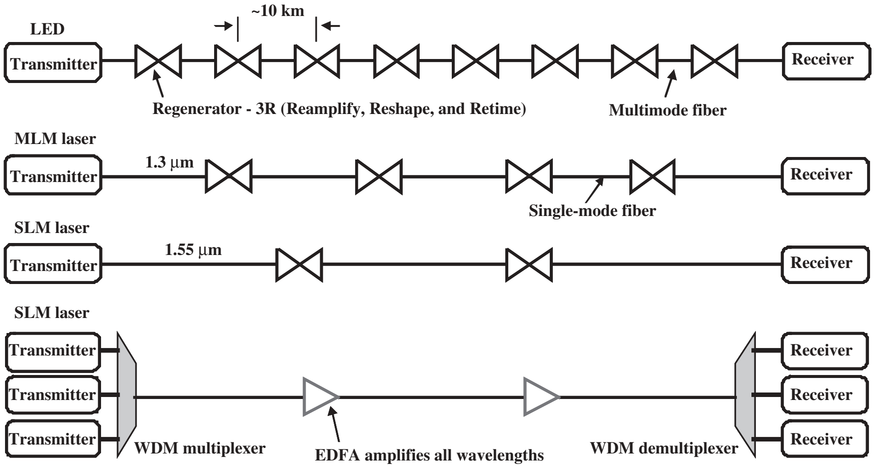


Figure 3. Evolution of optical fiber transmission systems.

reduced spectrum width. As a result, data bit rates were increased to more than 1 Gb/s.

The invention of EDFA optical amplifiers made it feasible to deploy a new generation of WDM systems, taking advantage of EDFA amplification of signals at many wavelengths simultaneously in the optical domain. The capacity increase of optical communication systems experienced a quantum leap. Instead of increasing the bit rate alone, using WDM, more than one wavelength can be used for transmission concurrently while the bit rate on individual wavelength channel can remain unchanged or be further increased. WDM techniques can thus be used to bridge the mismatch between electronic speeds and that of fiber equipment. Fewer regenerators are needed because, at a regenerator location, a single optical amplifier can replace an entire array of expensive regenerators, one per fiber. Another advantage of optical amplification is protocol transparency. WDM systems allow incremental capacity upgrade on demand. Starting in the mid-1990s, WDM systems with EDFAs were deployed. Today, almost all long-haul carriers have widely deployed amplified WDM systems. Transmission bit rates on a single channel have risen to 10–40 Gb/s. High-capacity amplified terabits/second WDM systems have hundreds of channels at 10 Gb/s with distances between electrical regenerators extending to a few thousand kilometers. Nowadays, achievable transmission capacity continues to grow while the cost per bit transmitted per kilometer continues to get lower to a point where it has become practical for carriers to price circuits independently of the distance.

Optical Fiber Networks

Early deployment of optical data communication networks included metropolitan area networks, such as the 100-Mb/s fiber distributed data interface (FDDI), and networks to interconnect mainframe computers, such as the enterprise serial connection (ESCON). At the same time, synchronous optical network (SONET), a standard for connecting fiber optic transmission systems, was proposed by Bellcore in the middle 1980s. SONET defines a hierarchy of interface rates that allow data streams at different rates to be multiplexed. SONET establishes optical carrier (OC) levels from 51.8 Mbps (OC-1) to 9.95 Gbps (OC-192). Prior rate standards used by different countries specified rates that were not compatible for multiplexing. With the implementation of SONET, communication carriers throughout the world can interconnect their existing digital carrier and fiber optic systems.

Today, a public network can be divided into a long-haul network and a metropolitan or metro network. The metro network spans a large campus or a region, typically reaching tens to a few hundred kilometers. The long-haul network interconnects different regional networks and can be as large as a few thousand kilometers. The metro network consists of a metro access network and a metro interoffice network. The access network extends from a central office out to individual businesses or homes as far as a few kilometers away, mostly collecting traffic from customer locations into the carrier network. The metro interoffice network interconnects central offices within a region. Optical fiber pairs and WDM technology have been used as the links in both the long-haul and the metro networks. Ring topologies have been widely deployed because of their simplicity and low cost and their ability to offer an alternative path to reroute traffic in case of failure. Metro access networks are almost exclusively ring based. In long-haul networks, mesh topologies are getting more attention recently because they offer more alternative paths and can be more resource efficient if properly managed. In many cases, a mesh network is actually implemented in the form of interconnected ring networks.

Point-to-Point WDM Systems

Driven by the increasing demands on communication bandwidth, WDM technology has been widely deployed for point-to-point communications in the Internet infrastructure. When the bandwidth demand exceeds the capacity in existing fibers, WDM can be more cost-effective than laying more fibers, especially over a long distance because more wavelength channels can be lit up as necessary. The trade-off is between the cost of installation/burial of additional fibers and the cost of additional line terminating equipment.

Broadcast-and-Select Local Area Optical WDM Networks

A broadcast-and-select local area WDM optical network consists of nodes connected by two-way fibers via a passive star coupler. Nodes are equipped with fixed tuned or tunable transceivers. A node's transmission over an available wavelength is received by the star coupler, which combines the received transmission with signals from other sources. The combined signal power is equally split and forwarded to all of the nodes on their receive fibers. A node's receiver then tunes to a wavelength agreed upon between the transmitter and the receiver using a distributed protocol. Two types of network architecture are possible (9): (*1*) single-hop architecture where a transmitter and a receiver communicate directly via the coupler; and (*2*) multi-hop architecture where information is forwarded via nodes in the network.

Metro Optical Ring Networks

Much of today's optical ring networks are built around SONET rings. A pair of fibers is used in unidirectional path-switched ring (UPSR) where one fiber is used as the working fiber and the other as the protection fiber. Traffic from node *A* to node *B* is sent simultaneously on the working fiber in the clockwise direction and on the protection fiber in the counterclockwise direction. As a result, if a link fails on one fiber, node *B* will be able to receive from the other fiber. The bidirectional line-switched ring (BLSR) connects adjacent nodes through one or two pairs of optical fibers, which corresponds to BLSR/2 and BLSR/4, respectively. BLSRs are much more sophisticated than UPSRs by incorporating additional protection mechanisms. Unlike a UPSR, working traffic in a BLSR can be carried on different fibers in both directions and is routed along the shortest path in the ring. Half of the capacity of each fiber is reserved for carrying the protection traffic in BLSR/2. In the event of a link failure, the traffic on the failed link is rerouted along the other part of the ring using the protection capacity available in the two fibers. A BLSR with four fibers (i.e., BLSR/4) uses a pair of fibers for protection and employs a span switching protection mechanism first. If a transmitter or receiver on a working fiber fails, the traffic is routed on the protection fibers between the two nodes on the same span. BLSRs provide spatial reuse capabilities by allowing protection capacity to be shared between spatially separated connections. BLSRs are significantly more complex to implement than UPSRs because of the extensive signaling required between the nodes. The WDM technology has provided the ability to support multiple SONET rings on a single fiber pair by using wavelength add/drop multiplixers (WADMs) to separate the multiple SONET rings. This tremendously increases the capacity as well as the flexibility of the optical ring networks. However, additional electronic multiplexing equipment is needed, which dominates the cost component and needs to be minimized via traffic grooming (10).

Wavelength-Routed Optical Networks

The massive increase in network bandwidth from WDM has heightened the need for faster switching at the core of the network (i.e., long-haul networks) to move from point-to-point WDM transmission systems to an all-optical backbone network that eliminates the need for per-hop packet forwarding. Wavelength-routed networks have been a major focus area since early 1990s. Wavelength-routed networks are considered to be an ideal candidate for wide area backbone networks.

A wavelength-routed network physically consists of several optical cross-connects (OXCs) or wavelength routers, taking an arbitrary topology. Each wavelength router takes in a signal at each wavelength at an input port and routes it to a particular output port, independent of the other wavelengths. The wavelength routers may also be equipped with wavelength converters that allow the optical signal on an incoming wavelength of an input fiber to be switched to some other wavelength on an output fiber link. The basic mechanism of communication in a wavelength-routed network is a *lightpath*. A lightpath is an all-optical communication channel that may span more than one fiber link between two nodes in the network. The intermediate nodes in the physical fiber path route the lightpath in the optical domain using the wavelength routers. If no wavelength converters are used, a lightpath must use the same wavelength on each hop of its physical fiber link, which is known as the *wavelength continuity constraint*. However, if converters are available, a different wavelength on each fiber link may be used to create a lightpath. A fundamental requirement of a wavelength-routed optical network is that two or more lightpaths traversing the same fiber link must use different wavelengths so that they do not interfere with each other. The end-nodes of the lightpath access the lightpath with transmitters and receivers that are tuned to the wavelength used by the lightpath.

Because of limitations on the number of wavelengths that can be used, and hardware constraints at the network nodes, it is not possible to set up a lightpath between every pair of source and destination nodes. The particular set of lightpaths that are established on a physical network constitutes the virtual topology or logical topology. Careful design of virtual topologies over a WDM network is to combine the best features of optics and electronics. The trade-off is between bandwidth flexibility and electronic processing overhead. The traffic on the lightpath does not have to undergo optoelectronic conversion at intermediate nodes. Traffic delay can be reduced through the use of virtual topologies and appropriate routing. However, because lightpaths are circuit-switched, forming lightpaths locks up bandwidth in the corresponding links on the assigned wavelength. A good virtual topology trades some ample bandwidth inherent in the fiber to obtain a solution that is the best of both worlds. Different virtual topologies can be set up on the same physical topology, which allows operators to choose or reconfigure a virtual topology that achieves the best network performance given network conditions such as average traffic between network nodes.

Passive Optical Networks

The access network or first-mile network, once called the last mile, connects the service provider central offices to business and residential subscribers. Currently, a variety of technologies and services are in use: dial-up service, DSL technology, cable modem, point-to-point microwave radio, and metro wireless access networks, which suffer from several drawbacks, e.g., limited reachable distances, limited data rates (shared in the case of cable TV network), and so on. Subscribers demand first-mile access solutions that are broadband and offer low-cost media-rich services. Fiber-to-the-home solution is still costly in most cases. To alleviate the problems and minimize fiber deployment cost, passive optical networks (PONs) (11,12) deploy a remote switch close to the subscribers' neighborhood and use a point-to-multipoint optical network with no active elements in the signals' path from source to destination. PONs allow for a long reach (over 20 km) between central offices and customer premises and much higher bandwidth per customer. All transmissions in a PON are performed between an optical line terminal (OLT) and optical network units (ONUs). The OLT resides in the central office, connecting the optical access network to the metro network. The ONUs are located at the customer location. From OLT to ONUs, a PON is a point-to-multipoint broadcast network (Fig. 4), and in the reverse direction, it is a multipoint-to-point network (Fig. 5) where bandwidths are shared by, for instance, TDM.

Ethernet is an inexpensive technology that is ubiquitous and interoperable with a variety of legacy equipment, and it is a logical choice for an IP data-optimized access network. The IEEE P802.3ah Ethernet in the First Mile Task Force completed its work with the approval of IEEE Std 802.3ah-2004 on Ethernet PON (EPON). An Ethernet PON (EPON) is a PON that carries all data encapsulated in Ethernet frames. PONs use a single wavelength in each of the two directions, and the wavelengths are multiplexed on the same fiber through coarse WDM. For example, the Ethernet PON (EPON) uses the 1490-nm wavelength for OLT to ONUs (downstream) traffic and the 1310-nm wavelength for ONUs to OLT (upstream) traffic. An enhancement of the PON supports an additional downstream wavelength, which may be used to carry video and cable TV services separately. PONs are in the initial stages of deployment in many parts of the world to support converged IP video, voice, and data services. In the near future, WDM-PONs in which multiple wavelengths may be supported in either or both upstream and downstream directions may become commercialized as technologies and markets mature (12).

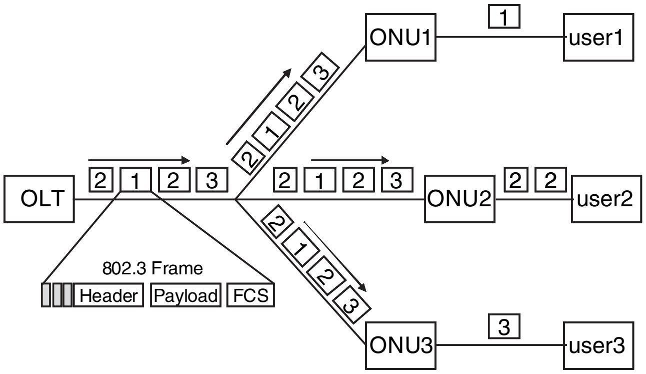


Figure 4. Downstream traffic of Ethernet over passive optical networks.

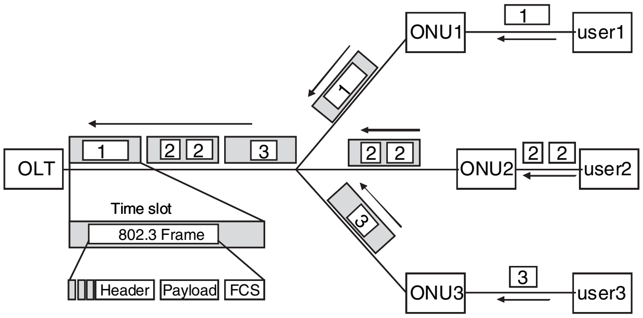


Figure 5. Upstream traffic of Ethernet over passive optical networks.

FREE-SPACE OPTICAL COMMUNICATION

Invented in the 1970s, free-space optics (FSO) (13), fiber-optic communication without the fiber, is an optical communication technology that uses low-power light propagating in free space to transmit two-way data between two points at gigabit-per-second rates (Fig. 6). Small-scale FSO systems have already been installed around the world. The reinvigoration of FSO is from the demand of advanced bandwidth-intensive services and applications as well as the inability of traditional copper wires and coaxial cables to keep up with the gigabits-per-second capacity needed in the first mile. Traditional fiber-oriented access networks incur high installation costs. Commercially available FSO equipment provides data rates much greater than those of digital subscriber lines or coaxial cables—from 10 Mb/s to 1.25 Gb/s. In addition, FSO systems can cost one third to one tenth the price of conventional underground fiber optic installations. Moreover, an FSO link can be up and running in a matter of days, whereas it could take 6 to 12 months to lay optic cables.

The operational principle of FSO is essentially the same as that of fiber optical communication. Similarly, FSO can also support multiple channels using WDM. The narrow transmitted infrared light beam suffers from beam disper-

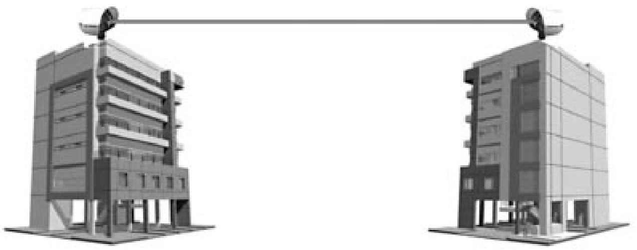


Figure 6. Free-space optical communication link connecting two office buildings.

sion where the light beam diverges over distance to form a cone with a fairly large breadth, rapidly reducing the amount of energy collectable by the receiver and energy received decreases inversely with the square of the distance. The lasers' limited power restricts their range to up to a few kilometers. Another critical issue is to align the light transmitter and receiver. As the light beam is narrow, alignment is affected easily by building sway and the thermal expansion and contraction of materials. Therefore, automatic active tracking systems are necessary that use movable mechanical platforms with feedback controls for regular adjustment to keep the transmitter and receiver on target in both directions, which adds complexity and cost to the FSO systems.

When used in vacuum, for example, for inter-space craft communication, FSO may provide similar performance to that of fiber optic systems. However, for terrestrial applications, the distance and data rate of FSO connections are highly dependent on atmospheric conditions. FSO links are prone to frequent failures caused by atmospheric absorption. Fog is vapor composed of water droplets, which are only a few hundred microns in diameter but can modify light characteristics or completely hinder the passage of light through a combination of absorption, scattering, and reflection. Fog causes significant loss of received optical power with 10–100 dB/km, which considerably limits the maximum range of an FSO link. This optical attenuation factor scales exponentially with distance. Rain and snow have little effect on FSO technology. Scintillation and pollution (smog) also cause varying degrees of attenuation. These factors result in an attenuated receiver signal and lead to higher bit error rates (BERs). Physical obstructions such as flying birds or construction cranes can temporarily block a single-beam FSO system, but this tends to cause only short interruptions, and transmissions are easily and automatically resumed.

To overcome these issues, solutions such as multibeam or multipath architectures are devised, which use more than one transmitter and more than one receiver. Each optical transceiver node can be set up to communicate with several nearby nodes in a network arrangement. Some state-of-the-art devices also have larger fade margin (e.g., extra power, reserved for rain, smog, and fog). Specifically, to increase the link range/reliability and network availability, FSO systems can be designed with limited link lengths as part of an interconnected optical mesh topology that connects FSO nodes. Each FSO node is equipped with multiple transceivers. These transceivers allow the nodes to communicate with nearby neighbors. Traffic generated by the clients of these nodes is ultimately relayed by the multihop optical mesh to the wired access infrastructure, e.g., a fiber ring add/drop multiplexer or an end-office switch. In addition to requiring a few optical transceivers, each repeater station in a mesh system must contain an electronic switch to combine (multiplex) the traffic from the local clients with that beamed from other nearby FSO nodes and to route traffic between the wired access infrastructure and each client served in the network. One approach to the reliable operation of a low-cost, free-space optical mesh is adequate density of switching nodes. If the density is sufficiently high, the length of each optical link will be sufficiently small such that fog attenuation can be negligible and mechanical tolerances can be loose. The FSO link expenses associated with tight link margins (e.g., pointing accuracy, optical beam-width, focusing, high-power lasers, and sensitive photo-receivers) can be eliminated or significantly reduced. The mesh topology can also be connected to several different locations of the wired access infrastructure, thereby providing greater overall capacity of the network. Intelligent routing and network management, e.g., multipath routing, can be implemented to choose a path for each FSO node's traffic through the mesh that passes through one of the system's outlets to the wired access infrastructure. Should a link fail, traffic would be redirected along an alternative path, making use of redundant routes and thereby facilitating rapid recovery from equipment failures. By reserving some unallocated capacity on each optical link, the network designer can ensure that sufficient capacity exists to reroute and recover from single- or multiple-link failures that might occur. The network reliability and availability can be further boosted by combining 60-GHz microwave radio with FSO because severe rain (which might cause a radio link failure) and dense fog (which might cause an FSO link failure) do not exist simultaneously, and microwave radio has some advantages, e.g., a longer distance and less attenuation by fog. Linking these two technologies, high access capacity can be economically and reliably delivered over a wide service area.

The advantages of FSO are many, including quick link setup, license-free operation, high transmission security, high bit rates, protocol transparency, and no interference. FSO is useful where physically connecting transmit and receive locations is difficult or economically prohibitive. The applications of FSO networks are abundant. FSO can be used for constructing community wireless networks where the network should largely self-configure and have robust connectivity, scalable network capacity, and low capital cost. Optical wireless networks are emerging as a viable, cost-effective technology for rapidly deployable broadband sensor communication infrastructures. The use of directional, narrow beam, optical wireless links provides great promise for secure, extremely high data rate communication between fixed or mobile nodes, which is very suitable for sensor networks in civil and military contexts. FSO can also be used for communications between spacecrafts, including elements of a satellite constellation.

ADVANCED TOPICS

Many areas of optical communication and networking are under intensive research and development. For example, integrated optics is to develop miniaturized optical devices of high functionality on a common substrate. The state-of-the-art of integrated optics is still far behind its electronic counterpart. Today, only a few basic functions are commercially feasible. However, a growing interest exists in the development of more and more complex integrated optical devices. New types of fibers are still being developed. The emerging field of photonic crystals led to the development of photonic crystal fiber, which guides light by means of diffraction from a periodic structure, rather than total internal reflection. The first photonic crystal fibers became commercially available in 1996. Photonic crystal fibers can be designed to carry a higher power than conventional fiber, and their wavelength-dependent properties can be manipulated to improve their performance in certain applications. The Internet Engineering Task Force (IETF) is investigating the use of Generalized Multi-Protocol Label Switching (GMPLS) and related signaling protocols (14) to set up and tear down lighpaths as well as for traffic engineering. GMPLS is an extension of Multi-Protocol Label Switching (MPLS) that supports multiple types of switching, including switching based on wavelength (a.k.a. Multi-Protocol Lambda Switching). With GMPLS, the OXC backbone network and the IP/MPLS subnetworks will share common functionality in the control plane, which makes it possible to seamlessly integrate all-optical networks within the overall Internet infrastructure. Various protection and restoration schemes and protocols for increasing the survivability and availability of WDM optical networks have been developed (1). Traffic grooming (10,15), dynamic service provisioning, and support for multicast services in optical networks are also active areas of research in both academia and industry. Finally, new network architectures such as optical burst switching (OBS) and optical packet switching (OPS) are currently under active research, experimentation, and evaluation.

FURTHER READING

W. Grover, *Mesh-Based Survivable Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking*. Upper Saddle River, NJ: Prentice-Hall PTR, 2003.

U. Black, *Optical Networks: third Generation Transport Systems*. Upper Saddle River, NJ: Prentice Hall, 2002.

S. Dixit (ed.), *IP over WDM: Building the Next Generation Optical Internet*. Hoboken, NJ: Wiley, 2003.

A. Somani, *Survivability and Traffic Grooming in WDM Optical Networks*. Cambridge, UK: Cambridge University Press, 2006.

C. Ye, *Tunable External Cavity Diode Lasers*. London: World Scientific, 2004.

BIBLIOGRAPHY

1. R. Ramaswami and K. N. Sivarajan, *Optical Networks: A Practical Perspective*, 2nd ed.San Francisco, CA: Morgan Kaufmann, 2002.
2. B. Mason, G. A. Fish, S. P. DenBaars, and L. A. Coldren, Widely tunable sampled grating DBR laser with integrated electroabsorption modulator, *IEEE Photonics Technol. Lett.*, **11** (6): 638–640, 1999.
3. G. E. Keiser, *Optical fiber Communications*, 3rd ed., Boston, MA: McGraw-Hill, 2000.
4. R. C. Alferness, Waveguide electrooptic modulators, *IEEE Trans. Microwave Theory Techniques*, **30** (8): 1121–1137, 1982.
5. B. Mason, A. Ougazzaden, et al, 40-Gb/s tandem electroabsorption modulator, *IEEE Photonics Technol. Lett.*, **14** (1): 27–29, 2002.
6. L. Lin, E. L. Goldstein, R. W. Tkach, On the expandability of free-space micromachined optical cross connects, *J. Lightwave Technol.*, **18** (4): 482–489, 2000.
7. T. Yamanoto, J. Yamaguchi, N. Takeuchi, A. Shimizu, E. Higurashi, R. Sawada, and Y. Uenishi, A three-dimensional MEMS optical switching module having 100 input and 100 output ports, *IEEE Photonics Technol. Lett.*, **15** (10): 1360–1362, 2003.
8. B. Mukherjee, WDM optical communication networks: Progress and challenges, *IEEE J. Selected Areas Commun.*, **18** (10): 1810–1824, 2000.
9. B. Mukherjee, *Optical Communication Networks*. New York: McGraw-Hill, 1997.
10. E. Modiano, Traffic grooming in WDM networks, *IEEE Commun. Mag.*, **39** (7): 124–129, 2001.
11. IEEE 802.3ah Ethernet in the First Mile Task Force. Home page. http://www.ieee802.org/3/efm/public/index.html, August 2006.
12. A. Banerjee, Y. Park, F. Clarke, H. Song, S. Yang, G. Kramer, K. Kim and B. Mukherjee, Wavelength-division-multiplexed passive optical networks (WDM-PON) technologies for broadband access: A review, *OSA J. Optical Networking*, **4** (11): 737–758, 2005.
13. Free Space Optics Technology. Home page. http://www.freespaceoptics.org/freespaceoptics/default.cfm, August 2006.
14. IETF Common Control and Measurement Plane Working Group. Home page. http://www.ietf.org/html.charters/ccamp-charter.html, August 2006.
15. K. Zhu, B. Mukherjee, A review of traffic grooming in WDM optical networks: Architecture and challenges, *Optical Networks Mag.*, **4** (2), 2003.

BIN WANG
Wright State University
Dayton, Ohio

PARALLEL AND VECTOR PROGRAMMING LANGUAGES

A parallel programming language is a formal notation for expressing algorithms. The meaning of this notation can be defined by appealing to a parallel computational model.

Parallel programming languages have more complicated data and control models than sequential programming languages. The data model in sequential languages is that of the random access machine (RAM) model in which there is a single address space of memory locations that can be read and written by the processor. The analog in parallel languages is the *shared-memory* model in which all memory locations reside in a single address space and are accessible to all the processors (the word *processors* in this article always refers to the logical processors in the underlying parallel execution model of the language and not to hardware processors). A more decoupled data model is provided by the *distributed-memory* model in which each processor has its own address space of memory locations inaccessible to other processors. The choice of the data model determines how processors communicate with each other—in a shared-memory model, they communicate by reading and writing shared locations, but in a distributed-memory model, they communicate by sending and receiving messages.

The control model in a parallel programming language determines how processors are coordinated. The simplest parallel control model is *lock-step* (vector) *synchronization.* At every step of program execution, each processor is either turned off or is required to perform the same operation as all other processors. The active processors at each step work on different data items, so this control model is also called the *single-instruction–multiple-data* (SIMD) model. SIMD-style parallel execution can be exploited in performing vector operations like adding or multiplying the elements of a set of vectors. *Bulk synchronization* is a more decoupled control model in which processors synchronize occasionally by using a *barrier* instruction. No processor is allowed to execute a statement past a barrier until all processors have arrived at that barrier. Between the execution of successive barrier statements, processors are autonomous and may execute different operations. Bulk synchronization can be used to exploit the *data parallelism* that arises when a function *f* is applied to each of the elements of a data structure such as a vector. All evaluations of *f* can be performed in parallel, so processors synchronize only at the beginning and end of this computation. Since *f* may have conditionals inside it, the processors may end up performing different computations, which is permitted in the bulk synchronous model. The most decoupled form of synchronization is *fine-grain synchronization* in which two or more processors can synchronize on their own whenever they need to, without involving other processors. This form of parallel execution is sometimes called *multiple-instruction–multiple-data* (MIMD) parallelism. The MIMD model is appropriate for exploiting *task* parallelism, which arises when autonomous computations (tasks) can execute concurrently, synchronizing only for exclusive access to resources or for coordinating access to data that is being produced and consumed concurrently by different tasks.

Figure 1 classifies the languages discussed in this article according to their control and data models. A survey of parallel programming languages can be found in Ref. 1.

LOCK-STEP SYNCHRONOUS PARALLEL LANGUAGES

Lock-step (SIMD) parallel languages are used mainly to program vector and array processors for performing scientific computations in which matrices are the primary data structures. Not surprisingly, most of these languages are extensions of FORTRAN. Programs in these languages contain a combination of scalar and vector operations. On array processors such as the Connection Machine CM-2 (Thinking Machines) (2) the scalar operations are usually performed by a front-end high-performance workstation, while the vector operations are performed on the array processor. Vector processors such as the CRAY processor (3) can execute both scalar and vector instructions. Therefore, the key problem in designing a SIMD language is to design constructs that expose as many vector operations as possible to the compiler.

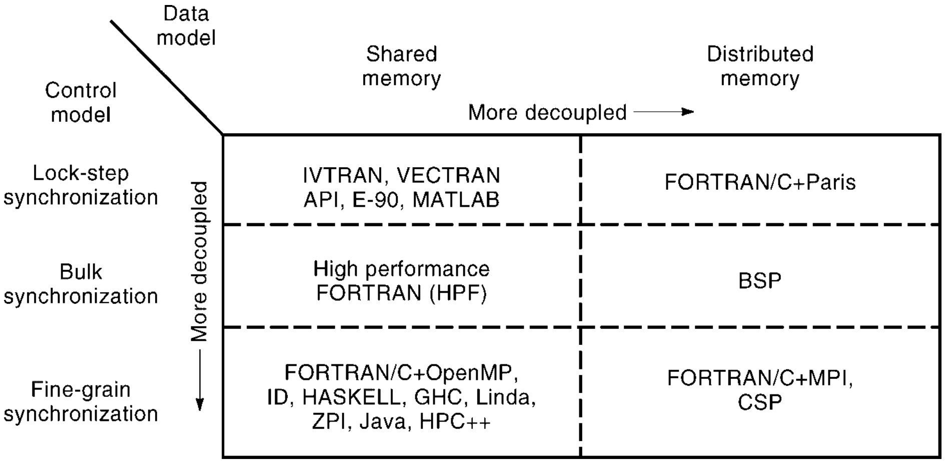
Shared-Memory SIMD Languages

The simplest vector operations involve the application of an arithmetic or boolean function to each element of an array (or arrays), thus computing the sum of two arrays by elements. These operations can be expressed quite simply by overloading arithmetic and boolean operators. For example, the FORTRAN-90 (4) statement `C = A + B` specifies that the sum by elements of arrays `A` and `B` is to be stored into array `C`.

In many applications, however, vector operations must be performed on some but not all of the elements of an array. For example, in solving partial differential equations, it may be necessary to apply one operator to points in the interior of the domain and a different one to points at the boundaries. Operator overloading is not sufficient to permit the expression of conditional vector operations, so a variety of constructs for describing sparse index sets have been invented.

Many SIMD languages provide the programmer with constructs for specifying the array section on which the vector operation is to be performed. One approach is to use *control vectors,* first introduced in the Burroughs Illiac IV FORTRAN language (5)—a value of *true* in a control vector indicates that the vector operation should be performed for the corresponding data element. An asterisk indicates a control vector of arbitrary length in which all elements are true. The following code shows the use of control vectors in this language. The first array statement adds the elements of the `A` and `C` arrays pointwise and assigns the results to `A`.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.



Control model \ Data model	Shared memory	Distributed memory
Lock-step synchronization	IVTRAN, VECTRAN API, E-90, MATLAB	FORTRAN/C+Paris
Bulk synchronization	High performance FORTRAN (HPF)	BSP
Fine-grain synchronization	FORTRAN/C+OpenMP, ID, HASKELL, GHC, Linda, ZPI, Java, HPC++	FORTRAN/C+MPI, CSP

Figure 1. A classification of parallel programming languages.

Because only the odd elements of B are true, the second array assignment adds only the odd elements of A and C and assigns the results to odd elements of A.

```
do 10 i = 1, 100,2
  B(i) = .true.
  B(i + 1) = .false.
10 continue
  A(*) = A(*) + C(*)
  A(B(*)) = A(B(*)) + C(B(*))
```

An important special case of conditional vector operations is *constant-stride* vector operations in which the elementary operations are applied to every *k*th element of the vector(s) for some integer *k*. On many vector computers, it is difficult to generate efficient code for these operations if control vectors are used. The operands and results of vector operations are usually stored in memory, so it is usually not worth performing an arithmetic operation in vector mode unless the loads and stores can also be done in vector mode. However, some computers [such as the CRAY-1 and CRAY-2 (3)] permit only constant-stride loads and stores from memory. Unless the compiler can determine that the *true* entries in a control vector occur with a fixed stride, it is forced to generate scalar loads and stores.

The IBM VECTRAN language (6) addressed this problem by introducing *array triplets,* which can describe many constant-stride array sections. An array triplet consists of three expressions separated by colons and specifies the start, end, and stride of the range of execution of a statement. If the stride is 1, the last expression and its preceding colon can be omitted. There is an obvious similarity between triplets and the specification of DO loop index sets in FORTRAN. The following code shows a use of triplets. After the last statement is executed, A(2) contains 1, A(4) contains 2, etc. Multidimensional arrays can be handled by using a triplet for each dimension of the array. Array triplet notation is also used in other array languages like MATLAB (7) and FORTRAN-90 (4).

```
  do 10 i = 1, 10
10 A (i) = i
  A(2:10:2) = A(1:5)
```

Although triplet notation is powerful, it is not a replacement for control vectors since it cannot describe arbitrary index sets. Therefore, VECTRAN supplemented the triplet notation with where statements, an example of which is given below.

```
where (A(1:100) .LT. 0)
   A(1:100) = − A(1:100)
otherwise
   A(1:100) = 0.0
endwhere
```

The where statement first evaluates a logical array expression. Statements in the body of where are executed for each index for which the logical array expression is true, while statements in the otherwise clause are executed for indices for which the logical expression is false. The clauses can contain only assignment statements where statements are in FORTRAN-90 as well.

The approaches described so far for expressing conditional vector operations are data-oriented in the sense they require the programmer to specify the array section on which the vector operation must be performed. A complementary approach is to embellish the control constructs in the language. One such construct, which was introduced in the IVTRAN language (8), is the forall statement in which the sparse index set is specified in terms of the control variables of the loop. The following code shows an example of its use. The loop has a two-dimensional index space in which all iterations can be performed in parallel, and in each iteration (i, j), the assignment is performed if A(i, j) is less than zero. Note that the forall construct permits assignment to constant-stride array sections such as diagonals, which cannot be described using triplet notation.

```
forall (i = 1:100:2, j = 1:100, A(i, j) .LT.
  0)
  A(i, j) = B(i, j)
```

Although reduction operations such as adding all the elements of a vector can also be done in vector mode, shared-memory SIMD languages have traditionally not had constructs to support these operations. However, most of them provide library routines that can be invoked by the programmer to perform reduction operations in vector mode.

Other shared-memory vector languages are LRLTRAN (9) from Lawrence Livermore Laboratories, BSP FORTRAN (10) from Burroughs, and Cedar FORTRAN (11) from the University of Illinois, Urbana. Cedar FORTRAN permitted the expression of both SIMD and MIMD parallelism. None of these languages, other than FORTRAN-90 and MATLAB, is in use.

Distributed-Memory SIMD Languages

The CM-2 machine (2) from Thinking Machines was a distributed-memory array processor and its assembly language, called Paris (parallel instruction set) (12), had FORTRAN, C, and Lisp interfaces that permit programmers to write high-level language programs with Paris commands embedded in them. The resulting languages were called FORTRAN/Paris, C/Paris, and Lisp/Paris, and they are examples of distributed-memory SIMD languages.

The programming model of Paris has an unbounded number of virtual processors (VPs) that can be configured into Cartesian grids of various sizes. Each VP has *local memory* for storing data, a *context flag* that controls instruction execution, and a unique *address* that can be used by other VPs to send it messages. Each VP can perform the usual arithmetic and logical operations, taking operands from its local memory and storing the result back there (one of the operands can be an immediate constant that is broadcast from the front-end processor). The execution of these operations can be made conditional on the context flag. Interprocessor communication is performed by executing the `send` instruction. Since processors operate in lock step, a separate instruction for receiving messages is not required; rather, the execution of the `send` instruction results in data transfer from the source VP to the destination VP. Therefore, the `send` instruction has to specify the address of the receiving processor and the memory addresses of the source and destination locations of the message. A given VP may receive messages from several other VPs during a `send` operation. If so, the data in these messages can be combined using a reduction operator specified in the `send` instruction. A noteworthy feature of Paris is that it was the first SIMD language to include a rich set of instructions for performing global reductions and parallel prefix operations on data stored in the VPs.

BULK SYNCHRONOUS PARALLEL LANGUAGES

Lock-step synchronization provides a simple programming model but it can be inefficient for programs with many data-dependent conditionals. Since processors operate in lock step, every processor must participate in the execution of *both* clauses of a conditional statement even though it performs computations in only *one* of the clauses. Bulk synchronization is a more relaxed synchronization model in which processors execute instructions autonomously but must rendezvous at intervals by executing a barrier instruction. No processor can execute an instruction past a barrier until *all* processors have arrived at that barrier. The interval between two successive barriers is called a *superstep*.

The requirement that all processors rendezvous at all barriers means that the most natural approach to programming in this model is to require all processors to execute the same program even though they can take different paths through that program to arrive at the same sequence of barrier instructions. This approach is sometimes called single-program–multiple-data (SPMD) parallelism, but this term has been abused sufficiently that we will not use it any further in this article.

Bulk synchronization is appropriate for exploiting data parallelism in programs. The simplest kind of data parallelism arises when a function is applied to each element of a data structure (like *mapcar* in LISP). A more subtle form of data parallelism arises when an associative operation such as addition or multiplication is used to combine all the elements of a data structure together. There is a well-known parallel algorithm ("tree reduction") for performing this operation in time proportional to the logarithm of the number of elements in the data structure (13). Data parallelism is also present in the computation of parallel prefix operations.

Shared-Memory Bulk Synchronous Languages

We use High-Performance FORTRAN (HPF) (14) as our example. HPF is somewhat unique among parallel languages in that it was designed by a group of no less than 50 researchers. It has two parallel loop constructs called the `FORALL` loop and the `INDEPENDENT` directive for expressing bulk synchronous parallelism. The body of the `FORALL` must consist of a sequence of assignments without conditionals or invocations of general procedures, although side-effect functions, declared to be `PURE` functions, can be invoked in a `FORALL`. These functions can contain conditionals. There is an implicit barrier at the end of every statement in a `FORALL`. The semantics of the `FORALL` is that all iterations of the first statement can be executed concurrently, and when these are completed, all iterations of the second statement can be executed concurrently, and so on. The right hand side of each statement is fully evaluated before the assignment is performed.

The `INDEPENDENT` directive before a DO loop tells the compiler that the iterations of the loop can be done in parallel since they do not effect each other. There is an implicit barrier at the end of the loop but loop iterations do not have to be synchronized in any way. This directive is often used to expose opportunities for parallel execution to the compiler, as shown in the following code. The `NEW` clause asserts that `J` is local to the outer loop. Iterations of the outer loop can be executed concurrently if the values of `IBLACK(I)` are distinct from the values of `IRED(J)` and if the `IBLACK` array does not have repeated values. This information cannot be deduced by a compiler, so the `INDEPENDENT` directive is useful for conveying this information.

```
!HPF$ INDEPENDENT, NEW(J)
  DO I = 1, N
    DO J = IBEGIN(I), IEND(J)
      X(IBLACK(I)) = X(IBLACK(I)) + X(IRED(J))
  END DO
  END DO
```

In HPF, the assignment of computational work to processors is not directly under the control of the programmer. Instead, it relies on a combination of data-distribution directives and compiler technology to produce code with good locality, as described in Refs. 15 and 16. The two basic distributions are *block* and *cyclic* distributions. Block distributing an array gives each processor a set of contiguous elements of that array; if there are p processors and n array elements, each processor gets a contiguous block of n/p elements. In a cyclic distribution, successive array elements are mapped to successive processors in a round-robin manner; therefore, element i is mapped to processor i mod p. HPF also supports a block–cyclic distribution in which blocks of elements are dealt to processors in a round-robin manner. The compiler can exploit data distributions in assigning work by assigning an iteration to a processor if that processor has most of the data required by that iteration. Alternative strategies like the owner-computes rule (16) are also popular.

An HPF program for computing π is shown below. It approximates the definite integral $\int_0^1 4/(1 + x^2)\ dx$ by using the rectangle rule, computing the value of $(1/n)$ $\Sigma_{i=1}^{n} 4/\{1 + [(i - 0.5)/n]2\}$. In this program, n is chosen to be 1000. SUM is a built-in function for computing the sum of the elements of a distributed array.

```
  PURE REAL FUNCTION F(X)
  REAL, INTENT(IN) :: X
  F = 4.DO/(1.DO + X*X)
  END FUNCTION F
  PROGRAM COMPUTE_PI
  REAL TEMP(1000)
!HPF$ DISTRIBUTE TEMP(BLOCK)
  WIDTH = 1.DO/1000
  FORALL (I = 1:1000)
  TEMP(I) = WIDTH * F((I - 0.5DO)*WIDTH)
  END FORALL
  T = SUM(TEMP)
  END
```

A second version of HPF called HPF-2 with support for irregular computations and task parallelism has been defined. IBM, PGI, DEC (now Compaq), and other companies have HPF compilers targeted to distributed-memory computers like the IBM SP-2 computer. However, the quality of the compiler-generated code is relatively poor in comparison to handwritten parallel code, and source-level performance prediction has proved to be difficult since performance depends greatly on decisions about interprocessor communication made by the compiler (17). For these reasons, interest in HPF is on the wane.

Distributed-Memory Bulk Synchronous Languages

The first theoretical study of bulk synchronous models was done by Valiant, who proposed the bulk synchronous parallel (BSP) model (18) as a bridging model between parallel hardware and software. A parallel machine in the BSP model has some number of processors with local memories, interconnected by a routing network. The computation consists of a sequence of supersteps; in each superstep, a processor receives data sent by other processors in the previous superstep, performs local computations, and sends data out to other processors that receive these data in the following superstep. A processor may send and receive any number of messages in each superstep. Consecutive supersteps are separated by barrier synchronization of all processors. Communication is therefore separated from synchronization.

Although BSP is a model and not a language, a number of libraries that implement this model on a variety of parallel platforms have been written (19,20). In this article, we describe the BSP Green library (19), which provides the following functions:

1. `void bspSendPkt(int pid, const bspPkt *pktPtr)`: Send a packet to the process whose address is `pid`; the data to be sent are at address `pktPtr`.
2. `bspPkt *bspGetPkt()`: Receive a packet sent in the previous superstep; returns NULL if all such packets have already been received.
3. `void bspSynch()`: Barrier synchronization of all processors.
4. `int bspGetPid()`: Return the process ID.
5. `int bspGetNumProcs()`: Return the number of processes.
6. `bspGetNumPkts()`: Return the number of packets sent in the previous superstep to this process that have not yet been received.
7. `bspGetNumStep()`: Return the number of the current superstep.

The first three functions are called *fundamental* functions since they implement the core functionality of the BSP model, and the last four are called *supplemental* functions. This set of functions is somewhat limited, and a more user-friendly library would provide other supplemental functions such as one to perform reductions, while remaining true to the BSP spirit. For example, the BSPLib project (http://www.BSP-Worldwide.org/) includes support for one-sided communication and high-performance unbuffered communication.

The following program (from Ref. 19) uses the BSP Green library functions to perform a trivial computation with three processors connected logically in a ring. Each processor sends the value of a local variable A to its neighbor in the ring and then performs a local computation with the value it receives. This takes two two supersteps. Note that some of the code (such as the calls to memcpy) is at a fairly low level of abstraction. The philosophy behind the decision to expose such details to the programmer is that all expensive operations should be evident when reading the program text.

```
void program(void)
{ int pid, numProcs, A,B,C;
  bspPkt pkt, *pktPtr;
  pktPtr = &pkt;
  pid = bspGetPid();    //get process ID
  numProcs = bspGetNumProcs();  /get number of
    processes
  if (pid==0){ A=3; B=12;} //initialize A and B
  if (pid == 1) { A = 1; B = 18;}
  if (pid == 2) { A = 5; B = 7;}
  memcpy((void*)pktPtr, (void*)&A, 4); //Store
    A into packet buffer
  bspSendPkt((pid+1)%numProcs, pktPtr); //send
    data to neighbor in ring
  bspSynch();    //superstep synchronization
  pktPtr = bspGetPkt();  //receive packet
  memcpy((void*)&C, (void*)pktPtr, 4);  //store
    data in C
    C = C + B;
  fprintf(stdout, ``Process %d, C = %d\n'', pid,
    C);
  bspSynch();  // superstep synchronization
}
```

One of the goals in the design of BSP is to permit accurate performance prediction of parallel programs. Performance prediction of BSP programs is made using a model with three parameters: (1) the number of processors p, (2) the gap g, which reflects the network bandwidth available to each processor, and (3) the latency L, which is the time required to send a packet through the network and perform a barrier synchronization. If a BSP program consists of S supersteps, the execution time for superstep i is $w_i + gh_i + L$, where w_i is the longest computation time required by any processor in that superstep and h_i is the largest number of packets sent or received by any processors in that superstep. This performance model assumes that communication and computation are not overlapped. The execution time for the program is $W + gH + LS$, where $W = \Sigma w_i$ and $H = \Sigma h_i$.

A major contribution of BSP has been to highlight what can be accomplished with its minimalist approach to communication and synchronization. However, the exchange of a single message between just two processors requires the cooperation of all processors in the machine! The BSP counterargument is that worrying about optimizing individual messages makes parallel programming too difficult and that the focus should be on getting the large-scale structure of the parallel program right.

FINE-GRAIN SYNCHRONOUS PARALLEL LANGUAGES

The most relaxed form of synchronization is *fine-grain* synchronization in which two or more processors can synchronize whenever they need to without the involvement of other processors. This style of programming is usually called multiple-instruction–multiple-data (MIMD) programming. Fine-grain synchronization is appropriate for exploiting *task parallelism* in which autonomous computations (tasks) need to synchronize either to obtain exclusive access to shared resources or because they are organized as a pipeline in which data structures are produced and consumed concurrently.

Shared-Memory MIMD Programming

We discuss FORTRAN/OpenMP (21), which is a new industry standard API (Applications Programmer Interface) for shared-memory parallel programming and contrast it with the more "expression-oriented" approach of Multilisp (22).

OpenMP. OpenMP is a set of compiler directives and run-time library routines that can be used to extend FORTRAN and C to express shared-memory parallelism. It is an evolution of earlier efforts like pthreads and the now-moribund ANSI X3H5 effort. An OpenMP FORTRAN program for computing π is shown below. A single thread of control is created at the start, and this thread executes all statements till the PARALLEL directive is reached. The PARALLEL directive and its corresponding END PARALLEL directive delimit a *parallel* section. At the top of the parallel section, a certain number of slave threads are created that cooperate with the master to perform the work in the parallel section and then die at the bottom of the parallel section. In our example, the only computation in the parallel section is the do loop. Furthermore, the DO directive asserts that the iterations of the loop can be performed in parallel. Optional clauses in this directive permit the programmer to specify how iterations should be assigned to threads. For example, the SCHEDULE(DYNAMIC,5) clause specifies that iterations are assigned to threads in blocks of five iterations; when a thread completes its iterations, it returns to ask for more work, and so on. There is an implicit barrier synchronization at the bottom of the parallel DO loop, as well as at the end of the parallel region. The barrier synchronization may be avoided by specifying the clause NOWAIT at these points. Once the parallel region is done, all threads except the master die. The master completes the execution of the rest of the program.

By default, all variables in a parallel region are shared by all the threads. Declaring a variable to be PRIVATE gives each thread its own copy of that variable. By default, loop control variables like i in our example are PRIVATE. Note that all the threads in our program write to the sum variable. Declaring sum to be a REDUCTION variable permits the compiler to generate code for updating this variable atomically. The compiler may also generate more elaborate code such as performing the reduction in a tree of processors.

```
  program compute_pi
  integer n,i
  double precision w,x,sum,pi,f,a
  f(a) = 4.d0/(1.d0 + a*a)
  print *, 'Enter the number of intervals'
  read *,n
  w = 1.0d0/n
  sum = 0.0d0
!$OMP PARALLEL
!$OMP DO SCHEDULE(DYNAMIC,5), PRIVATE(x),
  REDUCTION(+: SUM)
  do i = 1, n
    x = w * (i - 0.5d0)
    sum = sum + f(x)
    enddo
!$OMP END PARALLEL
  pi = w*sum
  print *,
  computed pi = ‘, pi
  stop
  end
```

Fine-grain synchronization in OpenMP is accomplished by the use of critical sections. The `CRITICAL` and `END CRITICAL` directives restrict access to the enclosed region to one thread at a time. For example, instead of declaring `SUM` to be a reduction variable as before, we can use a critical section to update it atomically as shown below.

```
!$OMP PARALLEL
!$OMP DO SCHEDULE(DYNAMIC,5), PRIVATE(x,temp)
  do i = 1, n
  x = w * (i - 0.5d0)
  temp = f(x)
!$OMP CRITICAL
  sum = sum + temp
!$OMP END CRITICAL
  enddo
!$OMP END PARALLEL
```

OpenMP also has a *parallel section* directive. Each section contains computations that can be performed in parallel with the computations in the other sections of this construct. OpenMP is being supported by SGI, KAI (Silicon Graphics Inc., Kuck and Associates Inc.), International Business Machines, and other companies.

Multilisp. It is instructive to contrast OpenMP with Multilisp (22), which is also a shared-memory MIMD parallel language, but one in which synchronization between producers and consumers of data can often be folded quite elegantly into the data accesses themselves. Multilisp is a parallel extension of Scheme, which was intended for writing parallel programs for the MIT Concert multiprocessor. There are two parallel constructs, one for evaluating the arguments to a function in parallel (`pcall`), and another for computing a value in parallel with executing code that will eventually use that value (`future`).

The expression (`pcall F A`) is equivalent to the Scheme procedure call (`F A`) except that the expressions `F` and `A` are evaluated in parallel. The function that expression `F` evaluates to is invoked after that evaluation of the argument `A` is complete. The `pcall` construct can be nested; for example, the expressions `F` and `A` may themselves contain `pcall` constructs.

The `future` construct can be used to fork off a computation that is performed in parallel with execution of code that may ultimately need the value of that computation. For example, the expression (`pcall cons A B`) evaluates `A` and `B` in parallel, and builds the `cons` cell when the evaluations are complete. The construction of the data structure need not wait for the termination of the computations of `A` and `B` since these computations can immediately return “place holders” for the ultimate values, replacing these place holders with the actual values when those become available. This can be accomplished by the invocation (`pcall cons (future A) (future B)`). While the computation of `A` and `B` is taking place, the `cons` cell can be used to build other data structures or be passed to other procedure invocations. An operation such as addition that tries to use the value of `A` or `B` before that value is available is blocked until the corresponding place holder is replaced with the value; when that value becomes available, that computation is allowed to continue. This is a form of fine-grain data-flow synchronization at the level of data structure elements.

The following program shows a Multilisp version of Quicksort (taken from Ref. 22). The `partition` procedure uses the first element `elt` of list `l` to divide the rest of `l` into two lists, one containing only elements less than `elt` and the other containing elements greater than or equal to `elt`. These lists are themselves sorted in parallel recursively, and the resulting lists, together with `elt`, are appended together to form the output. To reduce the overhead of explicitly appending lists, `qs` takes an additional argument `rest` that is the list of elements that should appear after the elements of `l` in the sorted list.

```
(defun qsort (l) (qs l nil))
(defun qs (l rest)
  (if (null l) rest
  (let ((parts (partition (car l) (cdr l)))) ;
  sort the two partitions in parallel
    recursively
  (qs (left-part parts)
    (future (cons (car l) (qs (right-part
    parts) rest)))))))
(defun partition (elt lst)
  (if (null lst)
  (bundle-parts nil nil)
  (let ((cdrparts (future partition elt (cdr
    lst))))
    (if (> elt (car lst))
      (bundle-parts (cons (car lst)
        (future (left-part cdrparts)))
      (future (right-part cdrparts)))
    (bundle-parts (future (left-part
      cdrparts))
       (cons (car lst)
        (future (right-part cdrparts))))))))
(defun bundle-parts (x y) (cons x y))
(defun left-part (p) (car p))
(defun right-part (p) (cdr p))
```

It can be seen that this Multilisp program is a functional program to which `future`’s have been added. The problem of deciding where it is safe and profitable to insert `future`’s in a general Multilisp program is a nontrivial one since Multilisp is an imperative language in which expression evaluation can have side effects. The suggested programming style is to write mostly functional code and look for opportunities to evaluate data structure elements as well as function arguments in parallel.

As in Scheme, the linked list is the key data structure in Multilisp. The role of lists in parallel programming is somewhat controversial because unlike arrays, lists are sequential access data structures and this sequentiality can limit acceleration in some programs. For example, consider applying a function *f* in parallel to each data item in a list. The list must traversed sequentially to spawn the parallel tasks, so parallel speed-up will be limited especially if the time for each function evaluation is small. If an array is used instead, the time required for the entire computation may be as small as the maximum of the times required for the individual function evaluations. Although linked lists are not used very often in parallel programming, note the future construct and its associated dataflow synchronization can be used in the context of other data structures.

The Linda language (23) also folds synchronization into data accesses, although in the case of Linda, synchronization is done during associative access of a shared tuple space.

Object-Oriented MIMD Languages

We describe HPC++ (24) and Java (25). There are both shared-memory languages.

HPC++. HPC++ (24) is a C++ library and language extension framework. For exploiting loop level parallelism, HPC++ has compiler directives called pragmas which are similar to the OpenMP directives. For example, parallel loops are exposed to the compiler by the `HPC_INDEPENDENT` directive, used in the following code to compute the ComputePi function.

```
double ComputePi(int n) {
  double w = 1.0/n;
  double sum = 0.0;
  #pragma HPC_INDEPENDENT, PRIVATE x
  for (int i = 1; i < n; i++) {
    double x = w * (i − 0.5);
    #pragma HPC_REDUCE
    sum += f(x);}
  return sum;}
double f(double a) {
  return 4.0/(1.0 + a*a);
}
```

One of the innovative aspects of HPC++ is its extension of the standard template library (STL) to support data parallelism. The STL in C++ provides (1) *containers* that define aggregate data structures like vector, lists, and queues, (2) *iterators* for enumerating over the contents of containers, and (3) *algorithms* that allow operations by element to be applied to containers. HPC++ has a parallel standard template library (PSTL) that provides parallel versions of these.

The most important container class in PSTL is the `Array` container (STL does not have multidimensional arrays that are crucial for scientific programming). By default, array containers are block-distributed but the programmer can specify a custom distribution by providing a *distribution object* containing a function that maps array indices to processors. The `par_for_each` iterator in PSTL is the parallel analog of the `for_each` iterator in STL. HPC++ also has a number of parallel algorithms such as `par_apply` for applying a function to each element of a container, and `par_reduction`, which is a parallel apply followed by a reduction with an associative binary operation. The following code shows HPC++ code for summing all the positive elements of a vector. The vector `v` is block distributed. The parameters to the `par reduction` algorithm are the associative combining operation, the function to be applied to each element of the container, and the starting and ending parallel iterators for the reduction.

```
BlockDistribution d(100, 100/numcontexts());
distributed_vector⟨double⟩ v(100, &d);
class GreaterThanZero{
  public:
    double operator() (double x){
      if (x > 0) return x;
      else return 0;
    }
};
...
double total = par_reduction(plus⟨double⟩(),
  GreaterThanZero(),
v.parbegin(),
    v.parend());
...
```

HPC++ is under active development. Planned enhancements to the existing implementation include a library for distributed active objects and an interface to CORBA via the IDL mapping. Another approach to extending C++ for parallel computing is the Charm++ effort (26).

Java

Java is a new object-oriented programming language that has a library of classes that support programming with threads. The thread library is intended primarily for writing multithreaded uniprocessor programs such as GUI managers. A parallel Java program consists of a number of threads executing in a single global object namespace. These threads are instances of user-defined classes that are usually subtypes of the `Thread` class in the Java library that override the run method of the `Thread` class to define what threads must do once they are created. Threads are first-class objects that can be named, passed as parameters to methods, returned from methods, etc. In addition, methods inherited from the `Thread` class permit a thread to be suspended, resumed, put to sleep for specified intervals of time, etc. Java also supports the notion of *thread groups.* Threads in a group can be suspended and resumed collectively.

Synchronization in Java is implemented using *monitors.* A monitor is associated with every object that contains a method declared to be *synchronized.* Whenever control enters a synchronized method in an object, the thread that invoked that method acquires the monitor for that object until the method returns. Other threads cannot call a synchronized method in that object until the monitor is released.

Java was not intended to be a language for parallel scientific computation. For example, it does not support multidimensional arrays nor are there any constructs for performing collective communication operations like reductions. However, there are efforts under way to use Java as a coordination language for multiplatform computational science applications (27).

Distributed-Memory MIMD Languages

One of the earliest distributed-memory MIMD languages is communicating sequential processes (CSP) (28) which spurred a lot of work on the theory and practice of message-passing language constructs. More recent languages in this area have taken a message-passing library like PVM (Parallel Virtual Machine) (29) or MPI (Message Passing Interface) (30) and grafted it onto a sequential language to obtain a distributed-memory parallel programming language. We will use FORTRAN/MPI to discuss this class of languages. In this programming model, a certain number of processes are assumed to exist, each having a unique name (usually a non-negative integer) and its own address space. Processes communicate by sending and receiving messages. A process can send data to another process by executing a SEND command, specifying the data to be transferred and the name of the recipient. The receiving process gets the data by executing a RECEIVE command, specifying the name of the sending process and the variable into which the data should be stored.

There are a number of variations on this basic SEND–RECEIVE theme. *Blocking* SEND–RECEIVE constructs requires the two processes to rendezvous before the data transfer takes place, which allows data to be transferred from one process to another without buffering in the operating system. However, if one process gets to the rendezvous considerably in advance of the other one, it cannot do useful work till the other process catches up with it. This problem led to the development of *nonblocking* SEND and RECEIVE constructs. A nonblocking SEND permits the sending process to continue execution as soon as the data has been shipped out to the receiving process even if the receiving process has not executed a RECEIVE command; the nonblocking RECEIVE construct is like a probe that permits the receiving process to check for availability of data without getting stuck if the data has not yet been received.

In addition to these SEND/RECEIVE commands, MPI has a number of collective *communication calls* that are useful for doing reductions, broadcasts, etc., collectively among *process groups*. These collective operations can be implemented using send and receive commands, but it is often possible to exploit the topology of the interconnection network to implement them more efficiently. MPI permits processes to belong to any number of process groups (called *communicators* in MPI terminology). All processes are members of the universal group `MPI_COMM_WORLD`.

The following code computes the value of π. The invocations of `MPI_COMM_SIZE` and `MPI_COMM_RANK` permit a process to determine the number of processes in the system and its own ID. The broadcast of the value of `n` is performed by invoking `MPI_BCAST`. The parameters to this call are the (1) the starting address of the data to be broadcast, (2) the number of values to be broadcast, (3) the type of the data, (4) the ID of process initiating the broadcast, (5) the process group to which the broadcast is performed, and (6) an error flag. Global reductions may be performed with a similar invocation.

```
program compute_pi
include ‘mpif.h’
double precision mypi,pi,w,sum,s,f,a
integer n, myid, numprocs,i,rc
f(a) = 4.d0/(1.d0 + a*a)
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrid, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs,
  ierr)
if (myid .eq. 0) then
  print *, ‘Enter number of intervals’
  read *, n
endif
call
  MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_
    WORLD,ierr)
w = 1.0d0/n
sum = 0.0d0
do i = myid+1,n,numprocs
  x = w*(i - 0.5d0)
  sum = sum + f(x)
enddo
mypi = w*sum
call MPI_REDUCE(mypi,pi,1,MPI_DOUBLE_
  PRECISION,
  MPI_SUM,0,MPI_COMM_WORLD,ierr)
if (myid .eq. 0) then
  print *, ‘computed pi =’ , pi
endif
call MPI_FINALIZE(rc)
stop
end
```

IMPLICITLY PARALLEL PROGRAMMING LANGUAGES

Many of the parallel programming languages described previously are in active use, but none of them is particularly *abstract* since they are all close to particular implementation models of parallel computing. It is likely that as better compiler and run-time systems technology becomes available, languages for programming parallel machines will become more abstract. This evolution would then parallel the evolution of sequential programming language that started out being very close to the hardware on which programs ran, but have since evolved to higher levels of abstraction. For example, early sequential languages like FORTRAN had GOTO statements which were manifestations of jump instructions in the underlying hardware, but GOTO statements have since been replaced by more abstract structured programming constructs. Similarly, variables in FORTRAN were names for fixed-memory locations and existed for the duration of the program just like memory addresses in the machine model, but the data models of modern programming languages are built on abstract notions like type, scope, and lifetime.

Although existing compiler and run-time systems technology is inadequate to permit efficient parallel programming in high-level abstract programming languages, many such languages have been proposed. In this section, we describe ZPL (Z-level Programming Language), an imperative array language that relies on parallelizing compiler technology to find opportunities for parallel execution, the functional languages Id and Haskell, and the logic programming languages Concurrent Prolog and PARLOG. An even more ambitious approach is taken by Unity (31), which attempts to derive parallel programs from high-level specifications written in a variation of temporal logic.

ZPL

In a FORTRAN or C program, statements that read and update disjoint memory locations can be executed concurrently. Therefore, it is possible in principle to use a sequential programming language like FORTRAN to program a parallel machine if one has a parallelizing compiler that can extract opportunities for parallel execution from sequential programs. An early compiler of this sort was PARAFRASE (32), which took FORTRAN programs and attempted to find parallel DO loops through program analysis. However, automatic parallelization has proved to be difficult in general, although there has been noteworthy success in some problem domains like numerical linear algebra.

ZPL (33) is an imperative array language without explicitly parallel constructs that relies on compiler technology to identify opportunities for parallel execution. A novel feature of this language is its *region* construct, an alternative to the *triplet* notation for describing the constant-stride index sets that was presented earlier. A disadvantage of the triplet notation is that it must be repeated for every subarray reference with this index sets (as in[1 : n] = B[1 : n] + C[1 : n] .) ZPL permits a more compact expression of such statements by providing the region construct that permits the definition and naming of index sets. The declaration region R = [1..n] can be viewed as defining a template of virtual processors of the appropriate size. Regions can be used with both data declarations and blocks of statements, as shown in the following code. An integer Intval is allocated on each virtual processor of the region; similarly, the statements in the block are executed by each virtual processor. Index1 is a keyword that permits each virtual processor to determine its index.

```
program Compute_pi;     – Program to approx. pi
  config var n : integer = 100;     – Changeable
    on Cmd Line
  region R = 1..n;     – Problem space
procedure f(a : double) : double; – Fcn for
  rectangle rule
    return 4 / (1 +  2);
procedure Compute_pi();     – Entry point
var Intval : [R] double;     – A vector of rect.
  pt.s
    pi :    double;    – Scalar result
[R] begin
  Intval := (Index1 − 0.5) / n; – Figure
    interval pts
  pi    := +< f(Intval) / n; – Approximate,
    sum, div
writeln(‘‘Computed pi = ’’ , pi);– Output to
  standard out
end;
```

Regions in ZPL may also be defined by applying operations like shifts to previously defined regions. Although ZPL compilers have been written for a variety of parallel platforms, it remains to be seen if the performance of the compiled code is sufficient to persuade programmers to move away from writing explicitly parallel programs in a language like FORTRAN with MPI.

Functional Languages

One approach to addressing the difficulty of determining noninterference of statements in languages like FORTRAN or C is to use functional language. These languages are based on the notion of mathematical functions that take values as inputs and produces values as outputs. When executing a functional language program, all functions whose inputs are available can be evaluated in parallel without fear of interference. This *data-driven* parallel execution model is the foundation of a number of functional languages like VAL (34), ID (35), and SISAL (36). An alternative execution model called *lazy evaluation* evaluates a function only if its inputs are available and it has been determined that the result of the function is required to produce the output of the program. Lazy evaluation permits the programmer to define and use infinite data objects such as infinite arrays or infinite lists (as long as only a finite portion of these infinite objects is required to produce the output), a feature that has been recommended for promoting modularity. Miranda (37) and Haskell (38) are languages that are based on the lazy evaluation model. Neither language is intended for parallel programming, but there is interest in defining a parallel verison of Haskell.

Operations like I/O do not fit naturally into the functional model since they are effects and not functions. Haskell uses *monads* to integrate I/O into a purely functional setting. A monad provides the illusion of an object with updatable state on which all actions are sequenced in a well-defined manner, which is sufficient for performing I/O. Monads permit the introduction of a limited form of side effects into functional language in a controlled manner, but these side effects are limited since monads cannot be used to define objects that can be updated concurrently.

Two problems have limited the impact of functional languages on the parallel programming community. The first is *aggregate update problem,* which refers to the difficulty of manipulating data structures like large arrays efficiently. Data structures are treated as values in functional languages, so they cannot be updated in place. The effect of storing a value v into element i of array A must be obtained by defining a new array B that is identical to A

except in the *i*th position where it has the value *v*. A naive implementation that makes a copy of *A* will be very inefficient. A variety of compiler optimizations (39) and language constructs like [*I structures* in Id (40)] have been proposed to address this problem but it is not clear to what extent these address the problem. A second problem is locality. In principle, an interpreter for a functional language can keep a work list of expressions whose inputs are available and evaluate these expressions in any order. Unless this is done carefully, it will have an adverse effect on locality, making it difficult to exploit caches and memory hierarchies. One solution is to remove caches from the implementation model and rely on *multithreaded processors* like dataflow processors that are latency tolerant. A complementary solution is to use compiler techniques to extract long sequential threads of computation from functional programs, and exploit locality in the execution of these threads. However, there appears to be little commercial interest in building multithreaded processors at this time; furthermore, the problem of sequentializing functional programs does not appear to be any easier than the problem of parallelizing imperative language programs.

Logic Programming Languages

Although functional languages eliminate the notion of sequential control from the programming model, they still retain the notion of *directionality* in the sense that the inputs of a function are distinct from its output. Logic programming languages provide an even higher level of abstraction by eliminating directionality through the use of relations (predicates) instead of functions. A logic program consists of a set of clauses that describe relations either explicitly by enumerating the tuples in the relation or implicitly in terms of other relations. Clauses that describe a relation explicitly are called *facts,* while those that describe relations implicitly are called *rules.* The first three facts in the program shown below specify that the `father` relation contains the tuples ⟨`Adam,Abel`⟩, ⟨`Adam,Cain`⟩, and ⟨`Abel,Bill`⟩. The parent relation is described by a rule: for all `X` and `Y`, the tuple ⟨`X,Y`⟩ is contained in the `parent` relation if it is contained in the `mother` relation (informally, `X` is the parent of `Y` if `X` is the mother of `Y`). The `grandfather` clause is defined implicitly as well: for all `X`, `Y` and `Z`, the tuple ⟨`X,Y`⟩ belongs to the `grandparent` relation if ⟨`X,Z`⟩ and ⟨`Z,Y`⟩ belong to the `parent` relation.

```
father(Adam, Abel).
father(Adam, Cain).
father(Abel, Bill).
mother(Eve, Abel).
mother(Eve, Cain).
parent(X,Y) :− mother(X,Y).
parent(X,Y) :− father(X,Y).
grandparent(X,Y)  :−  parent(X,Z),parent(Z,Y).
:− grandparent(Adam,W).
```

In terms of formal logic, the symbol :− stands for logical implication, and the symbol, on the right-hand side of clauses stands for conjunction. Variables like `X` and `Y` are universally quantified over the clause in which they appear. Each clause is therefore a *Horn clause,* and the program is a conjunction of Horn clauses.

Given the relations, it is possible to make a variety of queries such as asking if a given tuple occurs in a relation. *Bottom-up query evaluation* starts from the facts and uses the rules repeatedly to compute the tuples in the relations of the program, terminating when enough information has been obtained to answer the query. This kind of data-driven evaluation obviously exposes a lot of parallelism but it can lead to an unbounded amount of useless computation in general. *Top-down* query evaluation generates subproblems from the original query and solves them recursively to answer the query. The query `grandfather(Adam,W)` can be answered if we can find a `Z` and `W` such that `parent(Adam,Z)`*and* `parent(Z,W)`. The first subproblem can be solved in two ways: either by solving `mother (Adam,Z)`*or* by solving `father(Adam,Z)`. These explorations can be described compactly by an AND-OR tree, shown in Fig. 2.

Parallelism in top-down query evaluation comes in two flavors called *and-parallelism* and *or-parallelism.* In and-

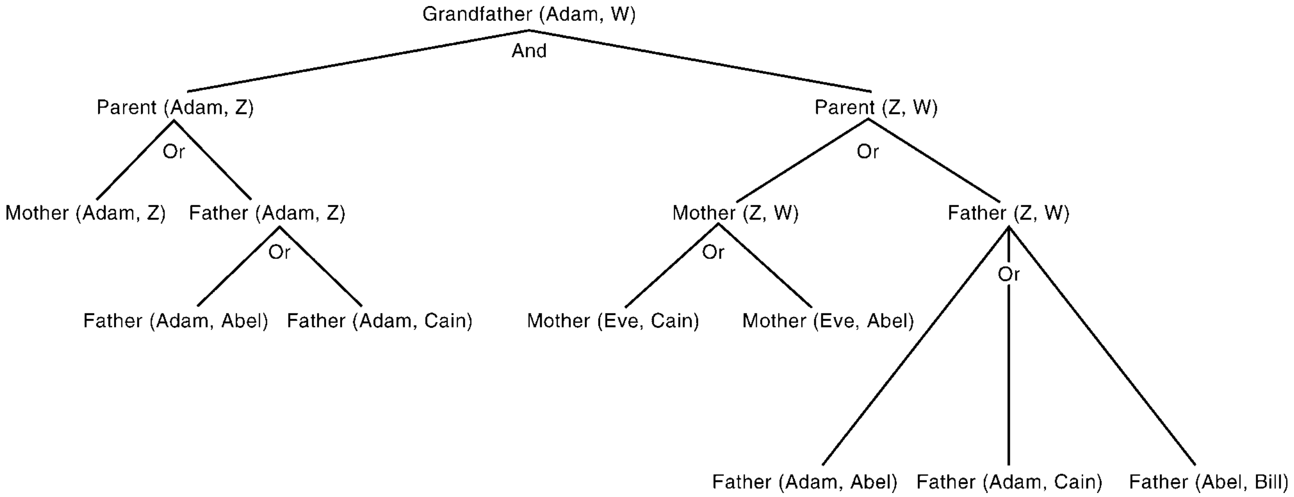


Figure 2. And-or tree.

parallelism, conjunctive subgoals such as `parent(Adam,Z)` and `parent(Z,W)` in our example are solved concurrently. The first subgoal produces possible solutions for `Z`, the second subgoal produces possible solutions for `Z` and `W` and the natural join of these solution sets produces the answers to the original query. Similarly in or-parallelism, disjunctive subgoals are solved in parallel and the results are unioned together.

The idealized model of parallel logic programming described here is difficult to implement efficiently, so researchers have proposed adding constructs to give programmers some control of parallel execution. To avoid having to compute the natural join of solutions from conjunctive subgoals solved in parallel, *mode declarations* can be used to specify that some subgoals will produce solutions that will be consumed by other subgoals. For example, Concurrent Prolog (41) has *read-only* annotations (?) using which we can write the `grandfather` clause as follows:

```
grandparent(X,Y) :— parent(X,Z),parent(Z?,Y).
```

This requires the first subgoal to produce `Z` and the second subgoal to read it. Similarly, PARLOG (42) has *mode declarations* on variables in the left-hand side of clauses. A limited form of or-parallelism called *committed choice or-parallelism* that uses Dijkstra's guards has been proposed in Guarded Horn Clauses (43) and PARLOG.

Logic programming ideas continue to be used in areas such as artificial intelligence, but there is little mainstream interest at this point. The early enthusiasm for separating the logic of algorithms from their control did not last very long, and logic programming found themselves introducing extralogical constructs like guards and modalities to improve program efficiency. In addition, a real programming language has to have arithmetic functions like addition and multiplication, but interpreted functions have always existed somewhat uneasily in the relational model. Some of these concerns are being addressed by Concurrent Constraint Programming languages like OZ (44).

CONCLUSION

Parallel programming today is done in languages that are very close to particular parallel implementation models. Thus, efficiency comes at the cost of portability. It is likely that parallel programming languages will become more abstract when the necessary compiler and runtime systems technology becomes available.

BIBLIOGRAPHY

1. D. Skillicorn and D. Talia, *Programming Languages for Parallel Processing*, New York, NY: IEEE, 1994.
2. Thinking Machines Corporation, *Connection Machine CM-200 Technical Summary*, June 1991.
3. CRAY Research Inc., *CRAY-1 Computer System Hardware Reference Manual*, 1978, Bloomington, MN.
4. W. Brainerd, C. Goldberg, and J. Adams, *Programmer's Guide to FORTRAN 90*, New York: Springer, 1996.
5. R. Millstein and C. Muntz, The Illiac IV FORTRAN compiler, *ACM Sigplan Notices*, **10** (3): 1–8, 1975.
6. G. Paul and M. Wilson, An introduction to VECTRAN and its use in scientific computing, *Proc. 1978 LASL Workshop Vector Parallel Process.*, 1978, pp. 176–204.
7. MathWorks Inc., *MATLAB Programmer's Manual*, 1996, Natick, MA.
8. R. Millstein and C. Muntz, The Illiac IV Fortran compiler, *ACM Sigplan Notices*, **10** (3), 1975.
9. R. G. Zwakenberg, Vector extensions to LRLTRAN, *ACM Sigplan Notices*, **10** (3): 77–86, 1975.
10. Burroughs Corporation, *Burroughs Scientific Processor Vector Fortran Specification*, 1978, Paoli, PA.
11. M. Guzzi et al., Cedar FORTRAN and other vector parallel FORTRAN dialects, *J. Supercomput.*, **3**: 37–62, 1990.
12. Thinking Machines Corporation, *Paris Reference Manual*, 1991, Cambridge, MA.
13. F. Leighton, *Introduction to Parallel Algorithms and Architectures*, San Francisco: Morgan Kaufmann, 1992.
14. C. Koelbel et al., *The High Performance Fortran Handbook*, Cambridge, MA: MIT Press, 1994.
15. D. Callahan and K. Kennedy, Compiling programs for distributed memory multiprocessors, *J. Supercomput.*, **2** (2), 151–169, 1988.
16. A. Rogers and K. Pingali, Process decomposition through locality of reference, *Proc. ACM Symp. Program. Lang. Design Implement.*, Portland, OR, 1989.
17. P. Hansen, An evaluation of high performance FORTRAN, *ACM Press Sigplan Notices*, **33** (3): 57–64, 1998.
18. L. Valiant, A bridging model for parallel computation, *Commun. ACM*, **33** (8): 103–111, 1990.
19. M. Goudreau et al., Towards efficiency and portability: Programming with the BSP model, *Proc. 8th Annu. ACM Symp. Parallel Algorithms Architect.*, Padua, Italy, June, 1996, pp. 1–12.
20. R. Miller, A library for bulk synchronous parallel programming, *Proc. BCS Parallel Process. Specialist Group Workshop Gen. Purp. Parallel Comput.*, London, England, December, 1993, pp. 100–108.
21. OpenMP Organization, OpenMP: A proposed industry standard API for shared memory programming. Available http://www.openmp.org
22. R. Halstead, Multilisp: A language for concurrent symbolic computation, *ACM Trans. Programming Lang. Syst.*, **7** (4): 31–56, October 1985.
23. D. Gelernter et al., Parallel programming in Linda, *Proc. Int. Conf. Parallel Programming*, Chicago, IL, August 1985, pp. 255–263.
24. E. Johnson and D. Gannon, HPC++: Experiments with the Parallel Standard Templates Library, Technical Report TR-96-51, Indiana University, 1996.
25. J. Gosling, W. Joy, and G. Steele, *The Java Language Specification*, New York: Addison-Wesley, 1996.
26. L. Kale and S. Krishnan, Charm++: A portable concurrent object-oriented system based on C++, *Proc. Conf. Object-Oriented Programming Syst., Lang. Appl.*, Washington, D.C., September 1993.

27. K. Dincer and G. Fox, Using Java and JavaScript in the Virtual Programming Laboratory: A web-based parallel programming environment. Technical report, Syracuse University, 1997.

28. C. Hoare, Communicating sequential processes, *Commun. ACM*, **21** (8): 666–677, 1978.

29. A. Beguelin et al., A user's guide to PVM: Parallel virtual machine. Technical Report TM-11826, Oak Ridge National Laboratories, 1991.

30. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI.* M.I.T. Press, 1994.

31. K. Chandy and J. Misra, *Parallel Program Design: A Foundation*, New York: Addison-Wesley, 1988.

32. D. Kuck, et al., The effects of program restructuring, algorithm change and architectural choice on program performance, *Int. Conf. Parallel Programming*, Chicago, IL, 1984, pp. 129–138.

33. W. Griswold, et al., Scalable abstractions for parallel programming, *Proc. 5th Distributed Memory Comput. Conf.*, Seattle, WA, 1990, pp. 1008–1016.

34. W. Ackerman and J. Dennis, VAL—A value-oriented language, Technical Report LCS/TR-218, MIT, 1979.

35. R. Nikhil, K. Pingali, and Arvind, Id Nouveau, Technical Report CSG Memo 265, M.I.T. Laboratory for Computer Science, 1986.

36. J. McGraw et al., Sisal: Streams and iterations in a single-assignment language, Technical Report M-146, Lawrence Livermore National Laboratories, 1985.

37. I. Holyer, *Functional Programming with Miranda*, London, England, UCL Press, 1992.

38. J. Peterson et al., *Haskell: A purely functional language* online, 1997. Available www: http://www.haskell.org

39. D. Cann, Compilation techniques for high performance applicative computation, Ph.D. thesis, Fort Collins, Colorado State University, 1989.

40. Arvind, R. Nikhil, and K. Pingali, I-structures: Data structures for parallel computing, *ACM Trans. Programm. Lang. Syst.*, **11**, 598–632, October 1989.

41. E. Shapiro, *Concurrent Prolog: collected papers*, volume 1, chapter A subset of Concurrent Prolog and its interpreter. Cambridge, MA: M.I.T. Press, 1987.

42. K. Clark and S. Gregory, *Concurrent Prolog: Collected Papers, Vol. 1*, Chapter PARLOG: Parallel programming in logic, Cambridge, MA: MIT Press, 1987.

43. K. Ueda, *Concurrent Prolog: Collected papers*, Volume 1, Chapter Guarded Horn Clauses. Cambridge, MA: M.I.T. Press, 1987.

44. G. Smolka, Problem solving with constraints and programming, *ACM Computing Surveys*, **28** (4), 1996.

KESHAV PINGALI
Cornell University
Ithaca, New York

PARALLEL ARCHITECTURES

INTRODUCTION

The need for solving increasingly complex problems has led to the design of fast computers capable of performing several things at once. Although it is difficult to give a single, precise definition that would describe all parallel architectures, one can think of these machines in terms of their parallel computational capabilities to speed up the execution of real applications. With large, compute-intensive applications, more operations can potentially be performed in parallel. To realize the speed up desired in executing an application, three components must work together: solution algorithms involving many independent operations, explicit or implicit parallel programming languages that identify parallel operations used to implement the algorithms, and the architecture of an underlying computer that can execute multiple operations simultaneously (1). In an attempt to define and distinguish between various types of parallelism that may be implemented in parallel architectures, Flynn (2) has characterized architectures based on the presence of single or multiple instruction streams and data streams. Single instruction stream (SI) combined with single data stream (SD) leads to *SISD* computers (single instruction stream, single data stream), which are the traditional sequential machines (also known as von Neumann). Single instruction stream combined with multiple data (MD) are *SIMD* or vector computers. In these machines, multiple processing elements (PEs) simultaneously execute the same instruction on different data. For example, an SIMD computer with 64 PEs can add the elements of two vectors A and B with 64 elements each in one single instruction $A + B$. This instruction is the equivalent of 64 addition operations on a sequential machine. However, everything else being equal, it is performed in roughly 1/64 of the time it would take in the sequential version. Multiple instruction streams combined with multiple data, *MIMD*, are *multiprocessor* architectures. Multiprocessors consist of several autonomous processors capable of executing independent sequential programs concurrently or cooperatively execute a single parallel program. The first results in increased system throughput but individual programs do not run faster, whereas the second approach leads to executing individual applications fast, which is the primary purpose of using these powerful computing machines. MIMD computers are capable of thread-level parallelism that is more generally applicable than data-level parallelism of SIMD computers. Multiprocessors are further distinguished by the way in which memory is accessed by processors. If all processors can access all system memory locations, then the multiprocessor is characterized as shared memory. If each processor has access to only its own memory, then it is characterized as distributed memory. In shared memory MIMD computers, parallel processors communicate with each other by writing and reading shared memory locations, whereas in distributed MIMD machines, processors must communicate through sending and receiving messages to and from each other. Other variations of memory access that result in multiprocessor hybrids not classified by Flynn's taxonomy will be described in a later section. The last combination of multiple data stream and single data stream is not very practical, although there are a few research machines that it may fit. Implied in all parallel types of architectures described here as SIMD or MIMD is the existence of some form of network to provide connectivity between their components (processor to processor or processor to memory) to facilitate communication and cooperation among parallel units (3). Parallel computer systems are at times categorized even more by the number of physical parallel units (processors) they provide. A massively parallel system (MPP) is often the term used to refer to parallel systems with hundreds and thousands of processors. The interconnection networks in these large-scale parallel systems must be highly concurrent and capable of delivering many simultaneous messages very fast.

A DEEPER LOOK INTO PARALLELISM IN COMPUTER ARCHITECTURES

In general, two main approaches to building faster computers are a faster clock rate, driven by the advances in the technology, and concurrency in operations, driven by architectural design. Sequential computer designers have been exploiting successfully both of these techniques to build very fast SISD computers. The next step in achieving a higher speed is parallel architectures. To gain insight into the variety of parallelism and to distinguish the concurrency within SISD and parallel computers, the focus here will be on architectural parallelism introduced at various levels of computer design instead of clock rates. The two main approaches to introduce operational concurrency are *overlap* and *replication*. In the original von Neumann architecture, the major components consisted of a control unit (CU) and an arithmetic logic unit (ALU) together forming the central processing unit (CPU) and the main memory (M). Today, these still form the major components, but much has been done to improve performance of this early computer (Fig. 1). When a computer is started, it repeatedly executes a hardware loop known as *fetch/execute* cycle. Initially the program instructions are stored in the main memory. To execute the program, the CPU must fetch instructions from the memory. The program counter register (PC) in the CPU always holds the address of the next instruction in the memory to be fetched and executed. To make sure the following terminologies are not new to the reader, without going into details, let us assume a machine instruction of the form $c = a \text{ op } b$, where "op" refers to one of the machine operations implemented in hardware, such as addition,

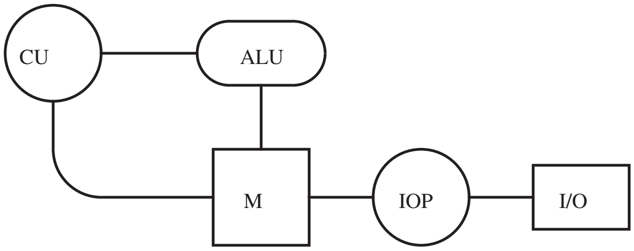


Figure 1. von Neumann architecture with I/O processor for I/O and CPU overlap. (Reprinted from Ref. 1 with Permission from Pearson Education.)

subtraction, and multiplication. a, b, and c are addresses of operands of this instruction in memory, and from the instruction, their addresses can be calculated in the CPU. A typical fetch/execute cycle for a von Neumann computer consisted of the following sequential steps:

1. IF: Fetch instruction from M at address pointed to by PC; bring it into the CPU.
2. ID: Decode instruction and increment PC to point to the next instruction.
3. Effective operand address calculation.
4. Operand fetch: Fetch operands from M at above address(es) and bring them into the CPU.
5. Execute: Perform the operation indicated by the instruction.
6. Store result: Store at the result operand address in M.

Improvements on this base machine started with adding features for faster input/output (I/O) processing where the concepts of overlap and parallelism were first introduced as are currently used and applied. The next major architectural advance toward concurrency in SISD architectures, also applied to many parallel computers, is a powerful technique known as *pipelining*. Going back to the sequential steps of the fetch/execute cycle in the von Neumann, it is clear that once an instruction is fetched from memory, no new instruction may be fetched until the current one has completed its execution. Using additional hardware, the steps of fetch/execute cycle can be overlapped as shown in Fig. 2. In this *instruction pipeline*, instructions follow each other through the pipeline stages that correspond to the fetch/execute cycle described above. Instructions are being executed sequentially and enter the pipeline in order, but once the pipeline is full (*start-up time*), five different instructions are being processed at different pipeline stages simultaneously and one instruction completes at every pipeline step rate. Theoretically, this design can be five times faster than the non-pipelined processor. However, complexities are associated with pipelining that make this peak performance unachievable although still by far faster than the non-pipeline design. The major issues here have to do with branches (*control hazard*) and dependencies between instructions (*data hazard*) in the pipeline. From the fetch/execute cycle, it can be seen that the next instruction being fetched is always coming from the next address relative to the current instruction being executed. However, when the instruction is for example a conditional branch, depending on the true/false outcome of the test, the next instruction may be coming from a different address called the target of the branch. In such an instruction, the content of PC is changed to that of the target. In a pipelined processor, the outcome of the branch is not known at the time the next instruction is to be fetched; therefore, the pipeline must be emptied. Performance degradation can also occur when an instruction in the pipeline needs the result of another instruction in the pipeline as its operand, which are known as data hazards. Solutions to deal with these types of dependencies (data and control hazards) may introduce bubbles or null operations into the pipeline. Various methods to deal with branch instructions and data hazards and to optimize the pipeline performance have been developed (4). Multiple arithmetic units in SISD computers introduced a different means for parallelism (replication). In this case, the CPU design is modified so that it consists of several arithmetic units capable of executing one operation each, but they can perform simultaneously. In this type of computer, potentially as many instructions as there are arithmetic units can be executing concurrently. To increase the potential for parallel execution, special hardware to prefetch several instructions from memory (*look-ahead buffer*), test for resource and data conflicts, and issue instructions out of program order (*score boarding*) is necessary. The two types of parallelism described here, instruction pipeline and multiple arithmetic (functional) units, for SISD computers are known as low-level parallelism and because of technological advances are included in most desktop computers of today. *Instruction-level parallelism (ILP), very long instruction word (VLIW)* architectures, and *superscalar* machines

Instruction fetch	I1	I2	I3	Jump	—	—	—	—
Instruction decode	—	I1	I2	I3	Jump	—	—	—
Operand fetch	—	—	I1	I2	I3	Jump	—	—
Execute	—	—	—	I1	I2	I3	Jump	—
Store operand	—	—	—	—	I1	I2	I3	Jump
Time	1	2	3	4	5	6	7	8

Figure 2. Instruction pipeline for an SISD architecture.

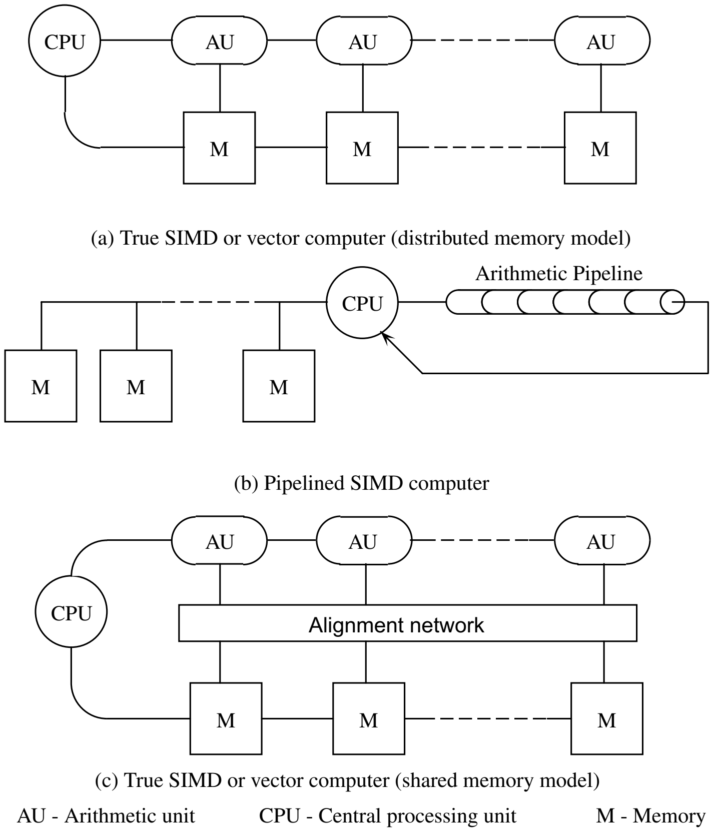
are also advances based on these basic parallel techniques. These machines are still classified as SISD even though they incorporate much concurrency in their design.

PARALLEL ARCHITECTURES

Technological advances combined with sophisticated architectural design resulted in uniprocessor performance growth throughout the 1986–2002 period. During this time, a large number of diverse, innovative, and expensive parallel computer architectures (*supercomputers*) have been designed with varying success mostly for the scientific community with compute-intensive applications. Networking available microprocessors to build affordable multiprocessors (*commodity multiprocessors*) made the field of parallel processing available to the larger community. The uniprocessor performance growth is reaching its limit because of high clock speed, resulting in problems with power consumption and heat dissipation, and a limited amount of ILP that can be exploited from sequential programs. The increasing capacity of a single chip has enabled placement of multiple processors on a single die resulting in *multicore* architectures. These machines can run at a lower clock speed to reduce heat dissipation and power consumption, they allow exploitation of a higher degree of concurrency from parallel programs instead of sequential programs, and they provide greater system density. Multicore architectures, which contain multiple logical processors in a single package available in almost all computers today, have renewed the interest in parallel computers as they are being mass produced. Different forms of multiprocessors with SIMD parallelism on a single cell chip have also been designed. Effective use of these computers through parallel algorithms, programming languages, compilers, and operating systems will greatly improve the overall system and application performance. Parallel computer architectures are mainly defined based on providing an explicit and coherent framework for high-level parallel solutions to application problems. This definition distinguishes the type of parallelism these machines must provide from those of SISD described earlier.

SIMD Computers

SIMD architectures provide hardware to execute the same instruction on many data items. SIMD computers incorporate this parallelism either through several *arithmetic pipelines (pipelined SIMD),* which is the most common (5), or through replicating PEs (*true SIMD*), (Fig. 3). These computers have a control unit that is capable of fetching and decoding instructions. SIMD computers have a single program counter in the control unit and perform the concurrent operations in locked steps using a global clock. The control unit sends the vector instructions either to the complete replicated arithmetic units in the true SIMD or issues them into arithmetic pipelines to be processed in an assembly fashion in the pipelined SIMD. SIMD Computers implement special vector and communication instructions for data routing in addition to the typical SISD machine



(a) True SIMD or vector computer (distributed memory model)

(b) Pipelined SIMD computer

(c) True SIMD or vector computer (shared memory model)

Figure 3. SIMD architectures. (Reprinted from Ref. 1 with Permission from Pearson Education.)

instructions. The true SIMD computers can further be organized as a distributed memory where each PE has access to its own memory [Fig. 3(a)]. In this model, an interconnection network provides communication between the PEs. Alternatively, SIMD computers may be organized as shared memory SIMD where an interconnection network allows for data routing between PEs and memory modules, [Fig. 3(b)]. Pipelining keeps the amount of parallel activity high while reducing the hardware requirement, [Fig. 3(c)]. Pipelined SIMD computers consist of pipelined arithmetic units that are different from instruction pipelining described for the SISD computers. Pipelining of arithmetic operations divides each operation, for example floating point addition, into several smaller ones and executes the subfunctions in parallel on different data as shown in Fig. 4.

SIMD Issues. Partitioning and data layout in memory for parallel access and interconnection networks for routing data are two key performance issues in SIMD machines. In a true distributed SIMD computer with 64 PEs, the addition of two 64-element vectors such as $C = A + B$ can be thought of as storing the corresponding vector elements (a_i, b_i, c_i) in the ith memory for the ith PE. Upon issuing the add instruction, the 64 PEs in the distributed model will simultaneously fetch their corresponding a and b operands from their memories, perform the addition, and store the result in the corresponding c locations. The key to obtaining good performance in this machine is to store the data to be accessed for parallel operation in different memory modules for parallel access. The interconnection network between the PEs must provide enough concurrency for PEs to exchange data. If the array elements are stored in the same memory module, then the array elements will have to be accessed from the memory sequentially and sent to the correct PE degrading performance to that of a sequential machine. The prime memory system is a technique for avoiding multiple references to the same memory module with regular access patterns (6). Although data layout in memory is the responsibility of the programmer, programming language, or compiler, the machine must provide an interconnection network to allow for routing the data to the correct PE needing it. In the shared memory model, the machine architecture must provide high concurrency interconnection networks between the memory modules and the PEs to allow data from different memories to be routed to the PEs in parallel. In pipelined SIMD, elements of A and B vectors are streamed into a floating point add pipeline as shown in Fig. 4. Although the vector components are not accessed simultaneously, successive references must still be made to different memory modules to attain full memory bandwidth and to match the pipeline speed. Most pipelined SIMD architectures provide fast vector registers where the vector operands are fetched into from memory and results are stored in before the final store in memory, *vector register pipelined SIMD*. Pipeline chaining, where the result of one arithmetic pipeline is fed as input into another arithmetic pipeline, is used to improve performance by reducing the number of memory accesses. An example for a chaining operation is $D = A(B + C)$, where the result from $B + C$ pipeline is fed into the multiple pipeline with elements of vector A synchronized with the first result from the adder. Some pipelined SIMD machines do not provide vector registers. In these machines, vector operands are pipelined from memory in a stream fashion to the arithmetic pipeline and results are stored similarly into memory, *memory-to-memory pipelined SIMD*. A larger memory bandwidth is needed to supply the pipelines with data at the pipeline speed. In this type of machine, the best performance is achieved for very long vector operations. As with instruction pipeline, a startup cost is associated with filling the pipelines. But once the pipeline is full, one result is produced at every minor pipeline cycle.

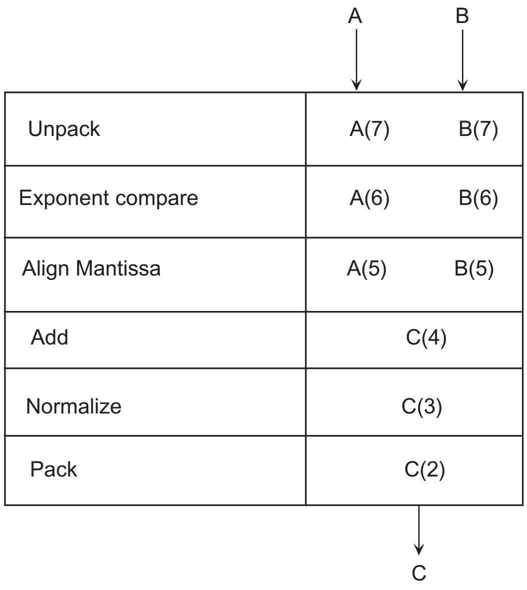
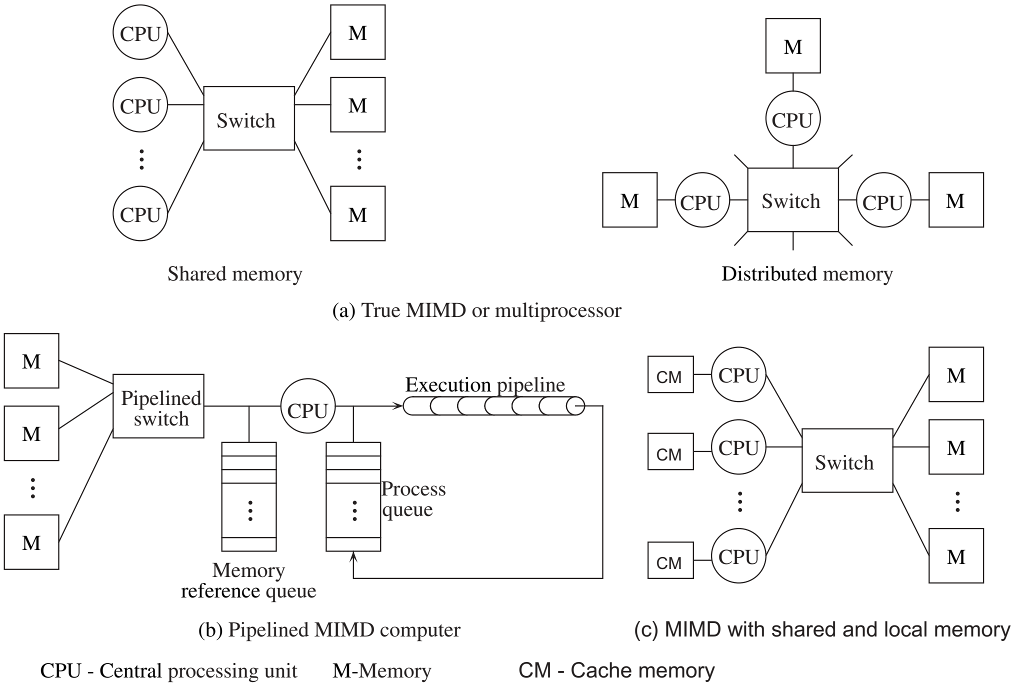


Figure 4. Floating point add pipeline. (Reprinted from Ref. 1 with Permission from Pearson Education.)

The parallelism provided by SIMD architectures is at the instruction level and is well suited to applications needing regular-patterned parallel operations. Today, SIMD processors most commonly are organized within an MIMD configuration to provide higher degrees of parallelism.

MIMD Architectures

A multiprocessor is a computer system that consists of multiple processors capable of executing independent instruction streams and one integrated system for moving data among the processors, memory, and I/O devices. MIMD computers can support higher levels of parallelism such as subprograms and tasks in comparison with SIMD-type parallelism. The parallelism in these machines can be exploited by numerous types of parallel operations that may be identified in the application programs. Many configurations of multiprocessors have been realized. What distinguishes the various configurations is the way in which results produced by one processor are made available to the others. Unlike SIMD, little difference exists between the programmer's view of one processor of an MIMD and the single processor of an SISD computer. The two basic types of MIMD computers, shared memory and distributed MIMD, are shown in Fig. 5.



(a) True MIMD or multiprocessor

(b) Pipelined MIMD computer

(c) MIMD with shared and local memory

Figure 5. MIMD architectures. (Reprinted from Ref. 1 with Permission from Pearson Education.)

Shared-Memory MIMD. The general interconnection network (switch) between the processors and system memory of Fig. 5(a) indicates that any processor can access any memory location. The communication and cooperation among processors that execute a parallel program takes place through reading and writing of shared memory locations. Synchronization operations must be provided to control access to shared data and to control the rate of progress of cooperating processes. Similar to SIMD machines, good performance depends on the interconnection network providing enough concurrency and bandwidth for fast and parallel memory accesses by processors. Shared memory MIMD computers may be designed as pipelined processors as in the SISD case instead of multiple complete processors [Fig. 5(b)]. However, unlike the SISD pipeline where the instructions issued into the pipeline come from a single process, instructions issued into the MIMD pipeline come from different instruction streams (processes). Therefore, the number of bubbles inserted into the MIMD pipeline caused by instruction dependencies is reduced significantly in comparison with SISD pipelining. The pipelined MIMD architectures are also known as *multithreaded* computers (7–10).

MIMD computers with various configurations, mainly due to the type of memory access they are organized to provide, have been designed. For example, each processor of the shared memory model in Fig. 5(a), may have some local (private) memory [Fig. 5(c)]. The private memories may be cache memories if they are controlled by hardware.

The issue in this type of architecture is how the local memories are used. The simplest is when they are used for read-only data and program stacks in shared memory MIMD. When the local cache memories in a shared memory MIMD are used for shared variables that may be both read and written, then a *cache coherence* protocol is needed to ensure the information in the main memory and the cache memories will remain consistent during program execution. Several approaches to providing coherent caches in these types of MIMD architectures have been implemented (11,12). A shared memory MIMD computer is referred to as *uniform memory access (UMA)* if it is configured such that any memory location can uniformly be accessed in the same amount of time. If the machine is organized so that access to some locations in the shared memory takes longer than others, then it is called a *non-uniform memory access (NUMA)*. A *cluster* is formed by connecting several shared memory multiprocessors through a communication network that they can use to send and receive instructions. In this case, the shared memory of each component of the cluster is considered private with respect to the other components. The recent multicore computers take the place of a cluster node where each multicore provides a shared memory MIMD configuration.

Distributed-Memory MIMD. Each node of this architecture consists of an autonomous processor and its local memory. The communication and cooperation among processors executing a parallel program takes place by processors explicitly send and receipt messages through the interconnection network. In these architectures, the synchronization is tied to the send and receipt of messages. Distributed memory architectures are distinguished from each other by the topology of the interconnection network through which their processors are connected. The network topologies will directly impact the way messages are routed from one processor to another, the number of messages that can concurrently be exchanged, the latency of message

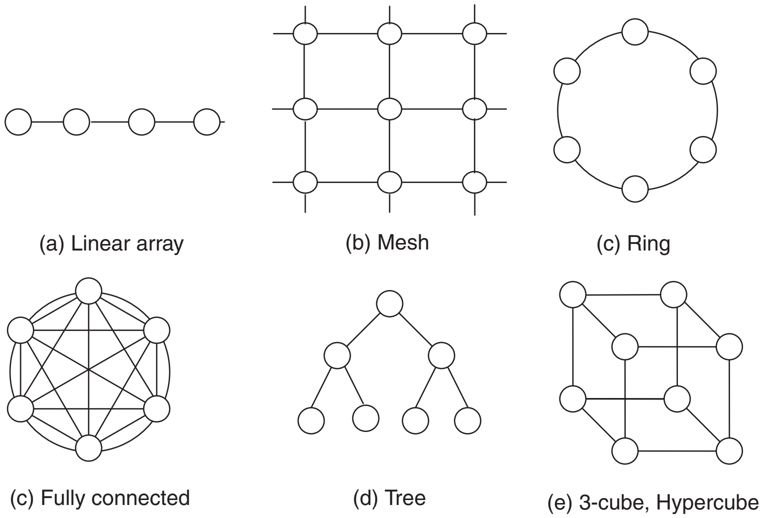


Figure 6. Common distributed memory MIMD toplogies. (Reprinted from Ref. 1 with Permission from Pearson Education.)

delivery, and the performance of executing parallel programs on the distributed memory architecture. Some of common topologies are shown in Fig. 6. The *ring* topology has commonly been used to interconnect a number of computers. In a unidirectional ring connecting N processors, each node is connected to one source and one destination processor. A message may have to travel through N-1 nodes (*hops*) to arrive at its destination. This longest path between any two nodes is called the *diameter* of the network. A bidirectional ring will improve the network diameter so that the longest path a message travels is $N/2$. The ring topologies have simple logic and can be used effectively with few processors, but several architectures have been implemented using more complex extensions of the ring such as multilevel hierarchy of unidirectional rings (13). Mesh topologies have been used extensively in designing distributed memory MIMD machines. Many topologies may be listed under this topology from a simple *linear array* to high-dimensional meshes. Two-dimensional mesh topologies are the most common. They are distinguished by the way the boundary nodes are connected to their neighboring processors. For example, *Wrap-around* connections reduce the network diameter. In a k- dimensional network with N^k nodes on each dimension, the diameter is $k(N^k-1)$. Hypercube topologies arrange $N = 2^n$ processors in an n-dimensional cube. Each node of this machine is directly connected to $n = \log_2 N$ other processors through a bidirectional link. Tree topologies with parent–child-type connections support divide-and-conquer problem-solving approaches. For two processors to communicate in this architecture, a path ascending from the two to a common parent is used. In these machines, the links closer to the root have a high traffic rate than the leaf nodes resulting in a bottleneck. Fat trees, where the number of links connecting parent–child processors increases as we get closer to the root, have been used to alleviate the problem. Full connectivity, although desirable, is not practical to implement for multiprocessors with large number of processors, N, as N^2 connections will be needed.

It may be noteworthy to consider that by replacing the single processor at each node with a shared memory MIMD, a cluster architecture can be configured. In general, hardware and software techniques may be devised to implement a distributed memory MIMD computer as a shared memory multiprocessor that results in a *distributed shared memory machine (DSM)*, which is also referred to as a shared address space multiprocessor.

Issues in MIMD. The performance of a parallel architecture is impacted significantly by inter-related factors such as the algorithm design, programming languages, operating systems, processor design, interconnection networks, memory hierarchy, cache and memory management, and latency tolerance mechanism. The performance capabilities of computers are often reported with measures such as Hertz ratings or peak-floating-point-operation-per-second (FLOPS) ratings. Careful interpretation of these processor-centered measures is needed as they do not reveal enough information regarding the architecture's overall performance. The data transfer capacity of the machine is a more accurate way to express performance and is measured by bandwidth and latency. The basis for this performance measure is that data have to arrive at the processor before they can be operated on. The layers of memory hierarchy and interconnection networks that may separate data from the processor will influence the amount of delay associated with getting data to the processor. For example, in shared memory MIMD, a processor may be slowed down when it has to wait for large shared memory access latency, for either data transmission or cache coherence information. In a distributed memory MIMD, the movement of messages containing intermediate results is the principal reason for not obtaining a peak operation rate in the processor. Discussions regarding the scalability of parallel architectures mostly favor distributed MIMD computers. However, regardless of the memory organization, a scalable computer system is defined as one in which data transport depends on the bandwidth and not on the latency, so that as the system size is increased, bandwidth increases but not the latency. Latency in computer architectures is mainly dealt with in two ways: (*1*) latency reduction as in cache memories or cut-through routing in interconnection networks, and (*2*) latency tolerance mechanisms as in overlapping operations in pipelining and multiprogramming. Thus, a computer architecture that can reduce its performance dependence on latency through one or a combination of these techniques will be the most scalable. In general, the optimal balance occurs when bandwidths above and below a given level differ by the amount of data reuse at that level, which is a difficult goal to achieve. When traffic is bursty, latency to satisfy a request for the next higher level can prevent the bandwidth from being fully utilized. Cluster computer organizations have become popular and are used commonly today. The cluster architectures are also highly vulnerable to most of these issues. They need to address issues of latency, synchronization, fine-grain parallelism, memory management, deep memory hierarchy, data movement, application types, form and degree of parallelism within applications in order to achieve high performance.

Other Parallel Architectures

Dataflow Architectures. Parallelism in computation is normally stated by specifying which operations can be executed in parallel, allocating storage to data, and scheduling those operations on parallel units. *Dataflow* is a concept that allows a computation to be represented without specifying any control flow or other dependence constraints on the order of operations except those of flow dependence among data. The concepts of dataflow have been central in the field of parallel processing and were originated by Dennis in 1973 (14). Computations can be represented accurately with *dataflow graphs*, which consist of directed edges, nodes (or actors), and tokens. The nodes or actors represent an operation corresponding to a program instruction and are connected by directed edges. The tokens are data values that move over directed edges; they are operated on at the node and are transformed into result tokens. Program instructions are executed (or *fired*) when their needed data tokens arrive at the instruction node. In this way the traditional control flow in other programming paradigms is replaced by dataflow. Dataflow architectures that are capable of executing dataflow graph concepts have processing elements that receive input data tokens (operands), perform the specified operation upon receipt of the operands, form new result tokens, and send them to the destination actor (instruction) in the dataflow graph. This capability is a major departure from the conventional von Neumann model that uses a program counter in the processor to address and fetch instructions. In this model, instructions are stationary, but data flows to the instruction that needs them as operands. The architecture of dataflow machines is impacted by the type of dataflow representations it must implement. In a static model, only one instance of a data token value is allowed per input edge of a node at a time. A typical static dataflow architecture is shown in Fig. 7. In a dynamic model, multiple instances of data tokens may exist on an input edge at a given time. Tokens belonging to different instances must be distinguished using matching tags. Dynamic dataflow is more flexible, can achieve higher parallelism, and results in a more complex architecture. Storage is a major issue that dataflow computers need to address. The ideal dataflow representation requires replication of data for every use or modification as no allocation of a memory location to data (variables) exists. Efficient scheduling of all ready operations for parallel execution is also another challenging issue that needs to be carefully addressed in dataflow architectures.

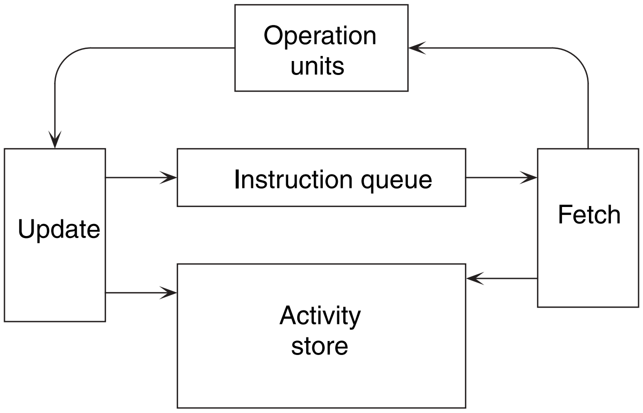


Figure 7. A typical dataflow architecture. (Reprinted from Ref. 1 with permission from Pearson Education.)

Systolic Arrays. Systolic arrays are special-purpose architectures consisting of simple computing cells with regular design patterns that are constructed in a modular layout well suited for VLSI implementations (15,16). Data are pumped through computing arrays in a pipeline fashion. In most cases, the computing cells in an array are identical and the design of the array is geometrically regular. For example, two-dimensional systolic arrays can be designed to perform fast matrix multiplication operations where each cell of the array can perform a multiply–add operation on its input operands as the appropriate elements of the two matrices flow through the cells of the array. Systolic arrays represent data dependences in the cell interconnections and can result in very efficient implementation of special-purpose algorithms (17).

BIBLIOGRAPHY

1. H. F. Jordan and G. Alaghband, *Fundamentals of Parallel Processing*, Englewood Cliffs, NJ: Prentice Hall, 2003.
2. M. J. Flynn, Some computer organizations and their effectiveness, *IEEE Trans. Computers*, **21**(9): 948–960, 1972.
3. T. Y. Feng, A survey of interconnection networks, *IEEE Computer*, **14**(12): 12–27, 1981.
4. J. P. Shen and M. Lipasti *"Modern Processor Design: Fundamentals of Superscalar Processors"*, McGraw-Hill, 2005.
5. P. M. Kogge, *The Architecture of Pipelined Computers*, New York: McGraw-Hill, 1981.
6. D. H. Lawrie and C. R. Vora, The prime memory system for array access, *IEEE Trans. Computers*, **31**(5): 1982.

7. R. Saavedra-Barrera, D. Culler, and T. vonEicken, Analysis of multithreaded architectures for parallel computing, *Proceedings of 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, 1990.
8. R. S. Nikhil, Tutorial notes on multithreaded architectures, *Proceedings of 19th Annual Symposium on Computer Architecture*, 1992.
9. R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Poterfield, and B. Smith, The Tera computer system, *Proceedings of the International Conference on Supercomputing*, Amsterdam, 1990.
10. D. E. Lenoski and W-D. Weber, *Scalable Shared-Memory Multiprocessing*, San Fransisco, CA: Morgan-Kaufman Publishers, 1995.
11. S. Adve and K. Gharachorloo, Shared memory consistency models: A tutorial, *IEEE Computer,* **29**(12): pp. 66–76, 1996.
12. D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture*, San Francisco, CA: Morgan Kaufmann Publishers, 1999.
13. Kendall Square Research Corporation, *KSR-1 Principles of Operation*, 1991.
14. J.B. Dennis, 'Data flow supercomputers,' *Computer*, **13**: 48–56, 1980.
15. H. T. Kung and C. E. Leiserson, Systolic arrays (for VLSI), in Duff and Stewart, eds., *Sparse Matrix Proceedings*, Knoxville, TN: SIAM, 1978.
16. H. T. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice-Hall, 1988.
17. D. I. Moldovan, *Parallel Processing from Applications to Systems*, San Mateo, CA: Morgan Kaufmann Publishers, 1993.

CROSS REFERENCES

Clusters. See Clusters and Grids.

Shared memory MIMD. See Shared-Memory Multiprocessors.

Interconnection networks. See Interconnection Networks.

Pipelined processors. See Computer Architecture.

Dataflow architectures. See Dataflow Computers.

GITA ALAGHBAND
University of Colorado
Denver, Colorado

PARALLEL DATABASE MANAGEMENT SYSTEMS

A *database* is a collection of data. A *database management system,* also called a DBMS, allows users to create a new database by specifying the logic structure of the data. For instance, the world of interest is represented as a collection of tables in relational DBMSs. This simple model is useful for many applications, and it is the model on which the major commercial DBMSs are based today. After a database has been created, the users are allowed to insert new data and query and modify existing data. The DBMS provides the users with the ability to access the data simultaneously, without allowing actions of one user to interfere with those of other users. The DBMS ensures that no simultaneous accesses can corrupt the data accidentally. In this article, we discuss how parallel processing technology is used to effectively address the performance bottleneck in DBMSs. After a brief discussion of the various parallel computer architectures suitable for DBMSs, we present the techniques for organizing data in such machines and the strategies for processing these data using multiple processors. Finally, we discuss some future directions and research problems.

Modern DBMSs are designed to support the client–server computing paradigm. In this paradigm, applications running on client computers or workstations are allowed to store and access data from a remote database server. This configuration makes best use of both hardware and software resources. Both the client and the database server can be dedicated to the tasks for which they are best suited. This architecture also provides an opportunity for both horizontal (i.e., more servers) and vertical (i.e., larger servers) scaling of resources to perform the task.

Today's database servers are generally general-purpose computers running database management software, typically a relational DBMS. These servers employ essentially the same hardware technology used for the client workstations. This approach offers the most cost-effective computing environment for a wide range of applications by leveraging the advances in commodity hardware. A potential pitfall of this approach is that the many equally powerful workstations may saturate the server. The situation is aggravated for applications that involve very large databases and complex queries. To address this problem, designers have relied on parallel processing technologies to build the more powerful database servers (1–4). This solution enables servers to be configured in a variety of ways to support various needs.

PARALLEL DATABASE SERVER ARCHITECTURES

The disk input/output (I/O) limitation problem has long been the obstacle for database applications. The disk I/O bottleneck sets a hard limitation on the performance of a database server. To address this problem, all parallel database approaches distribute the data across a large number of disks to take advantage of their aggregate disk bandwidth. The different types of parallel database servers are characterized by the way their processors are allowed to share the storage devices. Existing systems employ one of the three basic parallel architectures (5): shared everything (SE), shared disk (SD), and shared nothing (SN). None emerges as the undisputed winner. Each has its advantages as well as its disadvantages.

Shared Everything Architecture

The processors share all disks and memory modules [see Fig. 1(a)]. Examples of this architecture include IBM mainframes, HP T500, SGI Challenge, and the symmetric-multiprocessor (SMP) systems available from PC manufacturers. A major advantage of this approach is that interprocessor communication is fast because the processors can cooperate via the shared memory. This system architecture, however, does not scale well for very large databases. For an SE system with more than 32 processors, the shared memory would have to be a physically distributed memory to accommodate the aggregate demand on the shared memory from the large number of processors. An interconnection network (e.g., multistage network) is needed, in this case, to allow the processors to access the different memory modules simultaneously. As the number of the processors increases, the size of the interconnection network grows accordingly, which renders longer memory access latency. The performance of microprocessors is very sensitive to this factor. If the memory-access latency exceeds one instruction time, the processor may idle until the storage cycle completes. A popular solution to this problem is to have cache memory with each processor. However, the use of caches requires a mechanism to ensure cache coherency (i.e., ensure that all cached copies of the same data item have the same value). As we increase the number of processors, the number of messages caused by cache coherency control (i.e., cross interrogation) increases. Unless this problem can be solved, scaling an SE database server into the range of 64 or more processors will be impractical. Commercial DBMSs designed for this architecture include Informix Online Dynamic Server, Oracle Parallel Query Option, and IBM DB2/MVS.

Shared Disk Architecture

To address the memory-access-latency problem encountered in SE systems, each processor is coupled with its private memory in an SD system [see Fig. 1(b)]. The disks are still shared by all processors as in SE. Intel Paragon, nCUBE/2, and Tandem's ServerNet-based machines typify this design. As each processor may cache data pages in its private memory, SD also suffers the high cost of cache coherency control. In fact the interference among processors is even more severe than in SE. As an example, let us

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

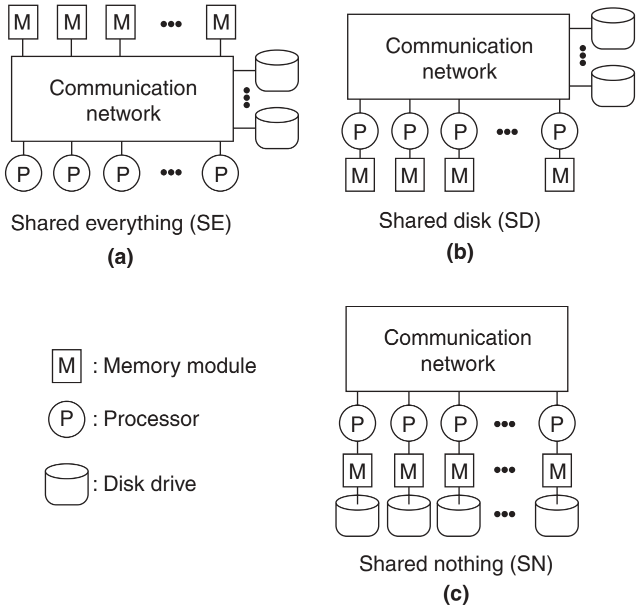


Figure 1. Three basic architectures for parallel database servers. Both disks and memory modules are shared by all processors in SE. Only disks are shared in SD. Neither disks nor memory modules are shared by the processors in SN.

consider a disk page containing 32 cache lines of data. No interference occurs in an SE system as long as the processors update different cache lines of this page. In contrast, an update to any of these cache lines in an SD system will interfere with all processors currently having a copy of this page even when they are actually using different cache lines of the page. Commercial DBMSs designed for this architecture include IBM IMS/VS Data Sharing Product, DEC VAX DBMS and Rdb products, and Oracle on DEC's VAXcluster and Ncube Computers.

Shared Nothing Architecture

To improve scalability, SN systems are designed to overcome the drawbacks of SE and SD systems [see Fig. 1(c)]. In this configuration, a message-passing network is used to interconnect a large number of processing nodes (PNs). Each PN is an autonomous computer consisting of a processor, a local private memory, and dedicated disk drives. Memory access latency is no longer a problem. Furthermore, as each processor is only allowed to read and write its local partition of the database, cache coherency is much easier to maintain. However, SN is not a performance panacea. Message passing is significantly more expensive than data sharing through the centralized shared memory as in SE systems. Some examples of this architecture are Teradata's DBC, Tandem NonStopSQL, Intel's Paragon, and IBM 6000 SP. Commercial DBMSs designed for this architecture include Teradata's DBC, Tandem NonStopSQL, and IBM DB2 Parallel Edition.

To combine the advantages of the previously discussed architectures and to compensate for their respective disadvantages, new parallel database servers are converging toward a hybrid architecture (6). In this architecture, SE clusters are interconnected through a communication network to form an SN structure at the intercluster level (see Fig. 2). The motivation is to minimize the communication overhead associated with the SN structure, yet keep each cluster size small within the limitation of the local memory and I/O bandwidth. Examples of this architecture include new Sequent computers, IBM RS/6000 SP, NCR 5100M, and Bull PowerCluster. Some commercial DBMSs designed for this structure are the Teradata Database System for the NCR WorldMark 5100 computer, Sybase MPP, and Informix-Online Extended Parallel Server.

DATA PARTITIONING TECHNIQUES

Traditional use of parallel computers is to speed up the complex computation of scientific and engineering applications. In contrast, database applications use parallelism primarily to increase the disk-I/O bandwidth. The level of achievable I/O concurrency determines the degree of parallelism that can be attained. If each relation (i.e., dataset) is divided into partitions, each stored on a distinct disk, a database operator can often be decomposed into many independent operators, each working on one partition. To maximize parallelism, several data partitioning techniques have been used (7).

Round-Robin Partitioning

The tuples (i.e., data records) of a relation are distributed among the disks in a round-robin fashion. The advantages of this approach are simplicity and the balanced data load among the disks. The drawback of this scheme is that it does not support associative search (i.e., search for tuples with the desired attribute values). Any search operations would require searching all disks in the system. Typically, local indices must be created for each data partition to speed up the local search operations.

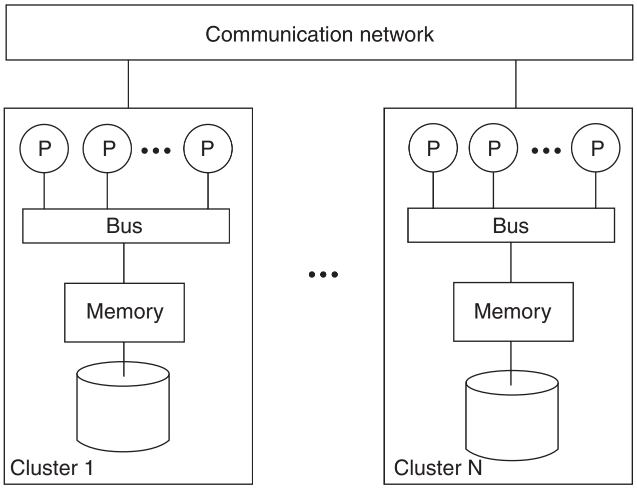


Figure 2. A hybrid architecture for parallel database servers. SE clusters are interconnected to form an SN structure at the intercluster level.

Hash Partitioning

A randomizing hash function is applied to the partitioning attribute (i.e., key field) of each tuple to determine the disk to store the tuple. Like round-robin partitioning, hash partitioning usually provides an even distribution of data across the disks. However, unlike round-robin partitioning, the same hash function can be employed at run time to support associative searches. A drawback of hash partitioning is its inability to support range queries. A range query retrieves tuples that have the value of the specified attribute falling within a given range. This type of query is common in many applications.

Range Partitioning

This approach maps contiguous key ranges of a relation to various disks. This strategy is useful for range queries because it helps to identify data partitions relevant to the query, skipping all uninvolved partitions. The disadvantage of this scheme is that data processing can be concentrated on a few disks, which leaves most computing resources underused. This phenomenon is also known as *access skew*. To minimize this effect, the relation can be divided into a large number of fragments using very small ranges. These fragments are distributed among the disks in a round-robin fashion.

Multidimensional Partitioning

Range partitioning cannot support range queries expressed on nonpartitioning attributes. To address this problem, multidimensional partitioning declusters a relation based on multiple attributes. As an example, let us consider the case of partitioning a relation using two attributes, say age and salary (see Fig. 3). Each data fragment is characterized by a unique combination of the age and salary ranges. For instance, tuples in the fragment [8,7] in Fig. 3 have the age values in range 8 and the salary values in salary range 7. These data fragments can be assigned to the disks in various ways (8–11). As an example, the following function can be used to assign a fragment $[X_1, X_2, \ldots, X_n]$ to a disk:

$$\mathit{DISK\_ID}(X_1, X_2, \ldots, X_n) = \left[\sum_{i=2}^{d} \left[\frac{X_i \cdot \mathit{GCD}_i}{N} \right] + \sum_{i=1}^{d} (X_i \cdot \mathit{Shf\_dist}_i) \right] \bmod N \qquad (1)$$

where N is the number of disks and d is the number of partitioning attributes;

$\mathit{Shf\_dist}_i = [\sqrt{N}]^{i-1}$ and $\mathit{GCD}_i = \gcd(\mathit{Shf\_dist}_i, N)$. A data placement example using this mapping function is illustrated in Fig. 3. Visually, the data fragments represented by the two-dimensional grid are assigned to the nine disks as follows:

1. Compute the shift distance *Shf_dist*. For this example, $\mathit{Shf\_dist} = [\sqrt{N}] = 3$.
2. Mark the top-most row as the check row.
3. Disks 0, 1, . . ., 8 are assigned to the nine fragments in this row from left to right. Make the next row the current row.
4. The allocation pattern for the current row is determined by circularly left-shifting the pattern of the row above it by three (i.e., *Shf_dist*) positions.
5. If the allocation pattern of the current row is identical to that of the check row, we perform a circular left-shift on the current row one more position and mark the current row as the new check row.
6. If there are more rows to consider, make the next row the current row and repeat steps 4, 5, and 6.

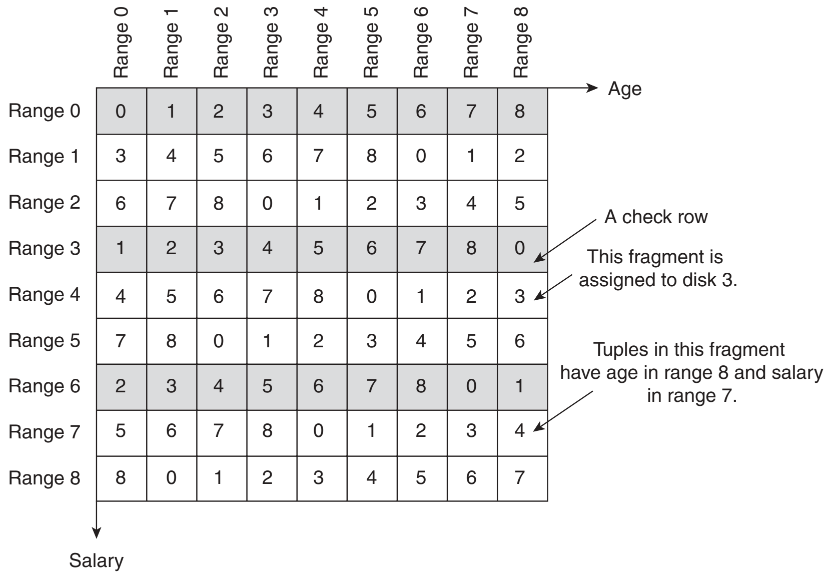
Assuming that nine had been determined to be the optimal degree of I/O-parallelism for the given relation, this data placement scheme allows as many types of range queries to take full advantage of the I/O concurrency as possible. Range queries expressed on either age or salary or both can be supported effectively. The optimal degree of I/O parallelism is known as the degree of declustering (DoD), which defines the number of partitions a relation should have. For clarity, we assume in this example that the number of intervals on each dimension is the same as the DoD. The mapping function Eq. (1), however, can be used without this restriction.

Many studies have observed that linear speed-up for smaller numbers of processors could not always be extrapolated to larger numbers of processors. Although increasing the DoD improves the performance of a system, excessive declustering will reduce throughput caused by overhead associated with parallel execution (12). Full declustering should not be used for very large parallel systems. The DoDs should be carefully determined to maximize the system throughput. A good approach is to evenly divide the disks into several groups and to assign relations that are frequently used together as operands of database operators (e.g., join) to the same disk group. Having different DoDs for various relations is not a good approach because the set of disks used by each relation would usually overlap with many sets of disks used for other relations. Under the circumstances, scheduling one operator for execution will cause most other concurrent queries to wait because of disk contention. This approach generally results in very poor system utilization.

PARALLEL EXECUTION

Today, essentially all parallel database servers support the relational data model and its standard query language: SQL (structured query language). SQL applications written for uniprocessor systems can be executed in these parallel servers without needing to modify the code. In a multiuser environment, queries submitted to the server are queued up and are processed in two steps:

- During compile time, each query is translated into a query tree that specifies the optimized order for executing the necessary database operators.
- During execution time, the operators on these query trees are scheduled to execute in such a way to maximize system throughput while ensuring good response times.



	Range 0	Range 1	Range 2	Range 3	Range 4	Range 5	Range 6	Range 7	Range 8
Range 0	0	1	2	3	4	5	6	7	8
Range 1	3	4	5	6	7	8	0	1	2
Range 2	6	7	8	0	1	2	3	4	5
Range 3	1	2	3	4	5	6	7	8	0
Range 4	4	5	6	7	8	0	1	2	3
Range 5	7	8	0	1	2	3	4	5	6
Range 6	2	3	4	5	6	7	8	0	1
Range 7	5	6	7	8	0	1	2	3	4
Range 8	8	0	1	2	3	4	5	6	7

A check row

This fragment is assigned to disk 3.

Tuples in this fragment have age in range 8 and salary in range 7.

Figure 3. Two-dimensional data partitioning based on age and salary. The 9 × 9 data fragments are assigned to nine processing nodes. Range queries based on age, salary, or both can be supported effectively.

Three types of parallelism can be exploited: intra-operator parallelism, intraquery parallelism, and interquery parallelism.

Intra-operator parallelism is achieved by executing a single database operator using several processors. This result is possible if the operand relations are already partitioned and distributed across multiple disks. For instance, a scan process can be precreated in each processor at system start-up time. To use a set of processors to scan a relation in parallel, we need only to request the scan processes residing in these processors to carry out the local scans in parallel. To effectively support various types of queries, it is desirable to create at least one process in each processor for each type of primitive database operator. These processes are referred to as *operator servers*. They behave as a logical server specializing in a particular database operation. Once an operator server completes its work for a query, the logical server is returned to the free pool awaiting another service request to come from some pending query. By having queries share the operator servers, this approach avoids the overhead associated with process creation. Intraquery parallelism is realized by arranging query operators in a query tree to allow several database operators to run concurrently without changing the query result. On the other hand, interquery parallelism is realized by scheduling database operators from different queries for concurrent execution. Two scheduling approaches have been used as follows.

Competition-Based Scheduling

In this scheme, a set of coordinator processes is precreated at system start-up time. They are assigned to the queries by a dispatcher process according to some queuing discipline, say, first come first serve. The coordinator that is assigned the query becomes responsible for scheduling the operators in the corresponding query tree. For each operator in the tree, the coordinator competes with other coordinators for the required operator servers. When the coordinator has successfully acquired all operator servers needed for the task, the coordinator coordinates these servers to execute the operation in parallel. An obvious advantage of this approach is its simplicity. It assumes that the number of coordinators has been optimally set by the system administrator and deals only with ways to reduce service times. The scheduling strategy is fair in the sense that each query is given the same opportunity to compete for the computing resources.

Planning-Based Scheduling

In this approach, all active queries share a single scheduler. As this scheduler knows the resource requirements of all active queries, it can schedule the operators of these queries based on how well their requirements match the current condition of the parallel system. For instance, a best-fit strategy can be used to select from the pending operators the one that can make the maximum use of currently available operator servers to execute first. The motivation is to maximize the resource utilization. This approach, however, is not as fair as the competition-based technique. Queries that involve very small or very large relations can experience starvation. The scheduler can also become a bottleneck. To ameliorate the latter problem, a parallel search algorithm can be used to determine the best fit.

We note that the scheduling techniques discussed previously do not preclude the possibility of executing two or more operators of the same query simultaneously (intraquery parallelism). Both scheduling techniques try to maximize the system performance by strategically mixing all three forms of parallelism discussed herein.

LOAD BALANCING

As each PN in an SN system processes the portion of the database on its local disks, the degree of parallelism is dictated by the placement of the data across the PNs. When the distribution is seriously skewed, balancing the load on these PNs is essential to good system performance

(12,13). Although SE systems allow the collaborating processors to share the workload more easily, load balancing is still needed in such systems to maximize processor utilization (14). More specifically, the load balancing task should equalize the load on each disk, in addition to evenly dividing the data-processing tasks among the processors. As an example, let us consider an extreme scenario in which a large portion of the data that needs to be processed happens to reside on a single disk. As little I/O parallelism can be exploited in this case, the storage subsystem cannot deliver a level of I/O performance commensurate with the computational capabilities of the SE system. Although the data-processing tasks can still be perfectly balanced among the processors by sharing the workload stored on that one disk, the overall performance of the system is deteriorated because of poor utilization of the available I/O bandwidth. Similarly, balancing the data load among the disks is essential to the performance of SD systems. In summary, no architecture is immune to the skew effect. We shall see shortly that similar techniques can be used to address this problem in all three types of systems.

SE and SD systems, however, do have the advantage under the following circumstances. Let us consider a transaction-processing environment in which frequently accessed data are localized to only a few disks. Furthermore, the system memory is large enough to keep these frequently used data in the memory buffer most of the time. In this case, it is very easy for the processors of an SE or SD system to share the workload because each processor is allowed to access the shared disks. In contrast, when an SN system is faced with this situation, only a couple of the PNs that own the disks with the frequently used data are overly busy. The remaining PNs are idle most of the time. This phenomenon, however, is most likely from bad data placement and usually can be rectified by redistributing the tuples.

Many load-balancing techniques have been developed for parallel database systems. Let us first examine techniques designed for SN systems. Several parallel join algorithms have been proposed. Among them, hash-based algorithms are particularly suitable for SN systems. In these strategies, the operand relations are partitioned into buckets in the hashing phase by applying the same randomizing hash function to the join key value, e.g., the join key value modulo the desired number of buckets. The buckets of the two relations, which correspond to the same hash value, are assigned to the same PN. These matching bucket pairs are evenly distributed among the PNs. Once the buckets have been assigned, each processor joins its local matching bucket pairs independently of the other PNs in the joining phase. This strategy is very effective unless there is a skew in the tuple distribution; i.e., some buckets are substantially larger than the remaining buckets. When severe fluctuations occur among the bucket sizes, some processors are assigned significantly more tuples on which to perform the local join operation. As the computation time of the join operation is determined by the slowest PN, skew in the tuple distribution seriously affects the overall performance of the system.

To minimize the skew effect, the buckets can be redistributed among the PNs as follows. At the end of the hashing phase, each PN keeps as many of the larger local buckets as possible; however, the total number of tuples retained should not exceed the ideal size each PN would have if the load were uniformly distributed. The excessive buckets are made available for redistribution among the PNs, using some bin-packing technique (e.g., largest processing time first), so as to balance the workload. This strategy is referred to as *partition tuning* (12). It handles severe skew conditions very well. However, when the skew condition is mild, the overhead associated with load balancing outweighs its benefits, which causes this technique to perform slightly worse than methods that do not perform load balancing at all, because this load balancing scheme scans the entire operand relations to determine the redistribution strategy. To reduce this overhead, the distribution of the tuples among the buckets can be estimated in the early stage of the bucket formation process as follows (15):

- *Sampling Phase:* Each PN independently takes a sample of both operand relations from its disk. The size of the sample is chosen such that the entire sample can fit in the memory capacity. As the sampling tuples are brought into memory, they are declustered into several in-memory buckets by hashing on the join attributes.
- *Partition Tuning Phase:* A predetermined coordinating processor computes the sizes of the sampling buckets by adding up the sizes of the corresponding local buckets. It then determines how the sampling buckets should be assigned among the PNs, using some bin-packing technique, so as to evenly distribute the sampling tuples among the PNs.
- *Split Phase:* Each processor collects the assigned local sampling buckets to form the corresponding sampling join buckets on its disk. When all sampling tuples have been stored to disks, each PN continues to load the remaining tuples from the relations and redistribute them among the same buckets on disks. We note that tuples are not written to disk one at a time. Instead, each processor maintains a page buffer for each hash value. Tuples having the same hash values piggyback to the same page buffer, and the buffer is sent to its disk destination when it is full.
- *Join Phase:* Each PN performs the local joins of respectively matching buckets.

The sampling-based load balancing technique has the following advantages. First, the sampling and load balancing processes are blended with the normal join operation. As a result, the sampling phase incurs essentially no overhead. Second, as the sample is a byproduct of the normal join operation and therefore is free, the system can afford to use a large sample whose size is limited only by the memory capacity. Although the technique must rely on page-level sampling to keep the I/O cost low, studies show that a sample size as small as 5% of the size of the two operand relations is sufficient to accurately estimate the tuple distribution under practical conditions. With the capacity of today's memory technology, this scheme is effective for a wide range of database applications.

We note that although we focus our discussion on the join operation, the same technique can also be used for other relational operators. For instance, load balancing for the union operation can be implemented as follows. First, each PN hashes its portion of each operand relation (using an attribute with a large number of distinct values) into local buckets and stores them back on the local disks. A predetermined coordinating PN then assigns the respectively matching bucket-pairs to the PNs using the partition tuning technique. Once the distribution of the bucket pairs has been completed, each PN independently processes its local bucket pairs as follows. For each bucket pair, one bucket is first loaded to build an in-memory hash table. The tuples of the other bucket are then brought into memory to probe the hash table. When a match is found for a given tuple, it is discarded; otherwise, it is inserted into the hash table. At the end of this process, the hash tables located across the PNs contain the results of the union operation. Obviously, the sampling-based technique can also be adapted for this and other relational operators.

Partition tuning can also be used to balance workload in SE and SD systems. Let us consider an SE system, in which the operand relations are evenly distributed among n disks. A parallel join algorithm which uses n processors is given below.

- *Sampling Phase:* Each processor is associated with a distinct disk. Each processor independently takes a local sample of both operand relations from its disk. The size of the local samples is chosen such that the entire sample can fit in the available memory. As the sampling tuples are brought into memory, they are declustered into several in-memory local buckets by hashing on the join attributes. Each processor also counts the number of tuples in each of its local buckets.
- *Partition Tuning Phase:* A predetermined coordinating processor computes the sizes of the sampling buckets by adding up the sizes of the corresponding local buckets. It then determines how the sampling buckets should be assigned among the disks, using some bin-packing technique, so as to distribute the sampling tuples evenly among the disks.
- *Split Phase:* Each processor collects the assigned local sampling buckets to form the corresponding sampling join buckets on its disk. When all sampling tuples have been collected to disks, each PN continues to load from its disk the remaining tuples of the two relations and redistribute them among the same buckets.
- *Join Phase:* Each PN joins the matching buckets located on its disk independently of the other PNs.

We observe in this algorithm that each disk performs the same number of read-and-write operations assuming the operand relations were evenly distributed across the disks. Furthermore, each processor processes the same number of tuples. The workload is perfectly balanced among the computing resources. An important advantage of associating a processor with a distinct disk unit is to avoid contention and to allow sequential access of the local partitions. Alternatively, the load can be evenly distributed by spreading each bucket across all disks. This approach, however, requires each disk to serve all processors at once during the join phase, causing the read head to move in an anarchic way. On another issue, each processor using its local buckets and page buffers during the sampling phase and split phase, respectively, also avoids contention. If the processors were allowed to write to a set of shared buckets as determined by the hash values, some mechanism would have been necessary to synchronize the write conflicts. This approach is not good because the contention for some buckets would be very severe under a skew condition.

FUTURE DIRECTIONS AND RESEARCH PROBLEMS

Traditional parallel computers were designed to support computation-intensive scientific and engineering applications. As the processing power of inexpensive workstations has doubled every two years over the past decade, it has become feasible to run many of these applications on workstations. As a result, the market for parallel scientific and engineering applications has shrunk rapidly over the same period. A few major parallel computer manufacturers having financial difficulties in recent years are evidence of this phenomenon. Fortunately, a new and much stronger market has emerged for those manufacturers that could make the transition to adapt their machines to database applications. This time, business is much more profitable for the following reasons. First, the database market is much larger than that of scientific and engineering applications. In fact, significantly more than half of the computing resources in the world today are used for data-processing-related tasks. Second, advances in microprocessor technology do not make workstations more suitable for handling database management tasks, which are known to be I/O intensive. It would be impractical to pack a workstation with a very large number of disks. Third, managing a large amount of multimedia data has become a necessity for many business sectors. Only parallel database servers can have the scalable bandwidth to support such applications.

As parallel database systems displaced scientific and engineering applications as the primary applications for parallel computers, manufacturers put a great deal of attention in improving the I/O capabilities of their machines. With the emergence of multimedia applications, however, a new hurdle, the network-I/O bottleneck (16–18), has developed for the database community. Essentially all of today's parallel database servers are designed for conventional database applications. They are not suitable for applications that involve multimedia data. For conventional database applications, the server requires a lot of storage-I/O bandwidth to support query processing. On the other hand, the demand on the network-I/O bandwidth is minimal because the results returned to the clients are typically a very small fraction of the data examined by the query. In contrast, the database server must deliver very large multimedia objects as query results to the clients in a multimedia application. As an example, the network-I/O bottleneck is encountered in Time Warner Cable's *Full Service Network* project in Orlando. Although each SGI Challenge server used in this project can sustain thousands of storage-I/O streams, the network-I/O bottleneck limits its performance to less than 120 MPEG-1 video streams. This poor performance is reminiscent of a large crowd funneling out of the gates after a football match. To address this bottleneck, eight servers had to be used at Time Warner Cable to serve the 4000 homes, which significantly increased the hardware cost and the costs of hiring additional system administrators. It is essential that future-generation servers have sufficient network-I/O bandwidth to make their storage bandwidth available to clients for retrieving large multimedia data.

Today's parallel database systems use only sequential algorithms to perform query optimization despite the large number of processors available in the system. Under time constraints, no optimizer can consider all parallel algorithms for each operator and all possible query tree organizations. A parallel query optimizer is highly desirable because it would have the leeway to examine many more possibilities. A potential solution is to divide the possible plans among several optimizer instances running on different processors. The costs of various plans can be estimated in parallel. At the end, a coordinating optimizer compares the best candidates nominated by the participating optimizers and selects the best plan. With the additional resources, it also becomes feasible to optimize multiple queries together to allow sharing of intermediate results. Considering the fact that most applications access 20% of their data 80% of the time, this approach could be a major improvement. More work is needed in this area.

Parallel database systems offer parallelism within the database system. On the other hand, existing parallel programming languages are not designed to take advantage of parallel database systems. A mismatch occurs between the two technologies. To address this issue, two strategies can be considered. One approach is to introduce new constructs in the parallel programming language to allow computer programs to be structured in a way to exploit database parallelism. Alternatively, one can consider implementing a persistent parallel programming language by extending SQL with general-purpose parallel programming functionality. Several companies have extended SQL with procedural programming constructs such as sequencing, conditionals, and loops. However, no parallel processing constructs have been proposed. Such a language is critical to applications that are both I/O intensive and computationally intensive.

As the object-oriented paradigm becomes a new standard for software development, SQL has been extended with object functionality. The ability to process rules is also being incorporated to support a wider range of applications. How to enhance existing parallel database server technology to support the extended data model is a great challenge facing the database community. For instance, SQL3 supports sequence and graph structures. We need new data placement techniques and parallel algorithms for these nonrelational data objects. Perhaps, techniques developed in the parallel programming language community can be adapted for this purpose.

BIBLIOGRAPHY

1. H. Boraket al., Prototyping bubba, a highly parallel database system, *IEEE Trans. Knowl. Data Eng.*, **2**: 4–24, 1990.
2. D. DeWittet al., The gamma database machine project, *IEEE Trans. Knowl. Data Eng.*, **2**: 44–62, 1990.
3. K. A. Hua and H. Young, Designing a highly parallel database server using off-the-shelf components, *Proc. Int. Comp. Symp.*, 1990, pp. 17–19.
4. M. Kitsuregawa, H. Tanaka, and T. Moto-oka, Application of hash to data base machine and its architecture, *New Gen. Comp.*, **1** (1): 63–74, 1983.
5. M. Stonebraker, The case for shared nothing, *Database Eng.*, **9** (1): 1986.
6. K. A. Hua, C. Lee, and J. Peir, Interconnecting shared-nothing systems for efficient parallel query processing, *Proc. Int. Conf. Parallel Distrib. Info. Sys.*, 1991, pp. 262–270.
7. D. DeWitt and J. Gray, Parallel database systems: The future of high performance database systems, *Commun. ACM*, **35** (6): 85–98, 1992.
8. L. Chen and D. Rotem, Declustering objects for visualization, *Proc. Int. Conf. Very Large Data Bases*, 1993, pp. 85–96.
9. H. C. Du and J. S. Sobolewski, Disk allocation for Cartesian product files on multiple disk systems, *ACM Trans. Database Sys.*, **7** (1): 82–101, 1982.
10. C. Fabursos and P. Bhagwat, Declustering using fracals, *Proc. Int. Conf. Parallel Distrib. Inf. Sys.*, 1993, pp. 18–25.
11. K. A. Hua and C. Lee, An adaptive data placement scheme for parallel database computer systems, *Proc. Int. Conf. Very Large Data Bases*, 1990, pp. 493–506.
12. K. A. Hua and C. Lee, Handling data skew in multicomputer database systems using partitioning tuning, *Proc. Int. Conf. Very Large Data Bases*, 1991, pp. 525–535.
13. J. Wolf, D. Dias, and P. Yu, An effective algorithm for parallelizing hash joins in the presence of data skew, *Proc. Int. Conf. Data Eng.*, 1991, pp. 200–209.
14. E. Omiecinski, Performance analysis of a load balancing hash-join algorithm for shared memory multiprocessor, *Proc. Int. Conf. Very Large Data Bases*, 1991, pp. 375–385.
15. K. A. Hua, W. Tavanapong, and Y. Lo, Performance of load balancing techniques for join operations in shared-nothing database management systems, *J. Parallel Distributed Comput*. **56**: 17–46, 1999.
16. K. Hua and S. Sheu, Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems, *Proc. ACM SIGCOMM'97 Conf.*, 1997.
17. S. Sheu, K. Hua, and W. Tavanapong, Chaining: A generalized batching technique for video-on-demand systems, *Proc. IEEE Int. Conf. Multimedia Com. Sys.*, 1997.
18. K. A. Hua, M. Tantaoui, and W. Tavanapong, Video delivery technologies for large-scale deployment of multimedia applications. *Proc. IEEE on Evaluation of Internet Technologies towards the Business Environment*, 2004.

KIEN A. HUA
University of Central Florida
Orlando, Florida
WALLAPAK TAVANAPONG
Iowa State University
Ames, Iowa

PEER-TO-PEER COMMUNICATION

INTRODUCTION

A *peer-to-peer (P2P) system* is a type of distributed system constructed at the application level and running at the edge of the Internet, usually on personal computers such as desktops and laptops of millions of users. Each end point in a P2P system is called a *peer*. Peers communicate through *peer-to-peer protocols,* which are on top of the Transmission Control Protocol and Internet Protocol. *Peer-to-peer communication* mainly refers to the communication protocols of peer-to-peer systems.

Peer-to-peer communication is different from a traditional *client–server model*. In peer-to-peer communication, peers in the system are symmetric: Each peer is both a *client* that requests information and services and a *server* that produces and/or provides information and services. A peer in peer-to-peer systems is also known as a *servent,* abbreviated from the combination of the words "*serv*er" and "cli*ent*." P2P systems aim to use the information and resources among end users of the Internet, which complements existing client–server systems.

A peer-to-peer system is an autonomous system in which peers are self-organized into an *overlay network*. Thus, peer-to-peer systems are often called *peer-to-peer networks*. No strict central control exists over all peers in the system (although there may be some kinds of centralized coordination mechanisms), and peers are free to come and go at any time. That is, P2P systems are highly transient. A major and important peer-to-peer application is file sharing among peers. The representative P2P file-sharing systems are Napster (1), Gnutella (2), KaZaa (3), eDonkey/eMule/Overnet (4), and BitTorrent (5).

The Basic Facilities of P2P Systems

The main facilities that P2P systems provide to peers are the content location, file downloading facilities, and incentive for service contributions. By organizing the index of the content that is shared by peers in the system into a uniform structure (centralized or decentralized), a P2P system can provide a hash table-like interface, where the content location and the content ID map are one-to-one. By performing a content search at each peer locally in parallel, a P2P system can support advanced search facilities such as a keyword search and a full-text search. By using some advanced techniques such as *Latent Semantic Indexing*, a hash table-like interface can also support a keyword search in P2P networks. Some P2P systems have no search facility and rely on users employing Web-based search engines to search the desired content manually.

Some P2P systems use the HTTP protocol for file downloading, such as Napster, Gnutella, and KaZaa. BitTorrent and eDonkey/eMule/Overnet use their own file downloading protocols so that the server can control the data sending rate. BitTorrent and eDonkey/eMule/Overnet support parallel downloading, in which each peer can download different parts of the file from multiple peers simultaneously.

KaZaa also supports parallel downloading by using the range request in the HTTP protocol. The advantage of parallel downloading is that it can reduce downloading time on the client side, and thus, it improves the user experience significantly. Another advantage is that peers do not need to wait for the complete downloading of the file to serve for other peers. Once the peer has downloaded a chunk of the file completely, it can serve other peers and continue to download other chunks simultaneously.

Early P2P systems such as Napster and Gnutella have no incentive mechanism for peers to contribute their service. Instead, they rely on the altruism of peers to support the file-sharing service. As a result, *free riding* is very common in such systems (6); many peers called *free riders* receive services without making any contribution. KaZaa provides a credit-based system to encourage peers to contribute, but it is difficult to prevent collusion on the contribution each peer makes. BitTorrent uses a "tit-for-tat" mechanism to restrain free riding and to prevent collusion effectively.

The Classification of P2P Systems

In general, peer-to-peer file-sharing systems can be classified as *centralized*, in which a central server hosts the indices of the content shared by peers in the system or *decentralized*, in which the indices of the content are distributed among peers in the system. Decentralized peer-to-peer systems can be further classified into *unstructured* and *structured* systems, based on the mechanisms of overlay organization and index search. A structured peer-to-peer system has global coordination on the overlay structure and the datasets in the system, whereas an unstructured system does not. Furthermore, according to the methodology of file sharing, peer-to-peer systems can be classified as *exchange-based*, where peers exchange different files with each other by their interests, or *swarming-based*, where peers download the same content by exchanging small chunks of a large file.

In addition to being used for file sharing, peer-to-peer systems have also been used for Internet telephony, also known as voice over IP (VoIP), and live media streaming on the Internet, also known as IPTV. Skype (7,8) is a peer-to-peer VoIP system, in which peers are used for both searching clients and relaying voice packets. PPLive (9,10) is a peer-to-peer streaming video system, which uses peer-to-peer collaboration to distribute online and live media among users.

CENTRALIZED P2P SYSTEM

The first generation of peer-to-peer file-sharing systems is centralized and index-based, such as Napster. In *centralized peer-to-peer systems*, a central index server maintains

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

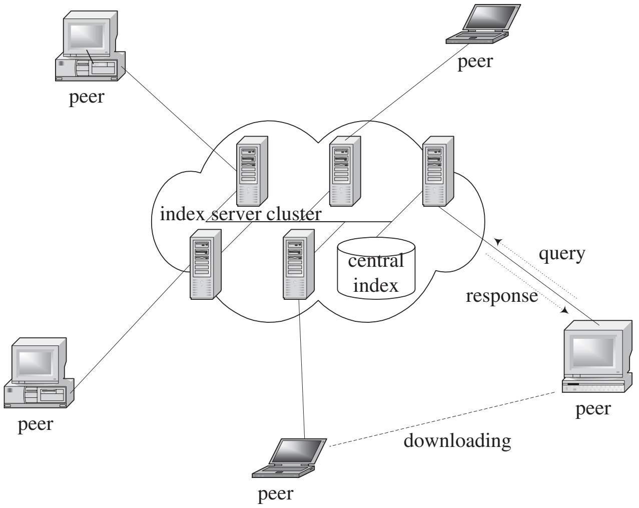


Figure 1. Index-based P2P system.

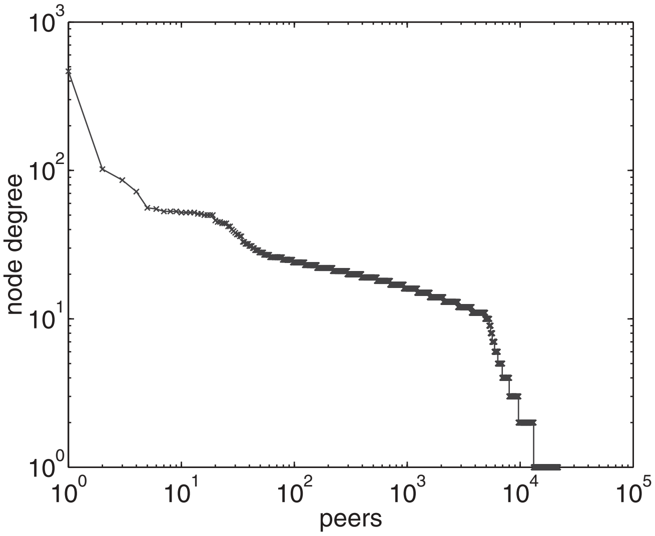


Figure 2. The connectivity degree of an unstructured P2P network.

the directory of files that all peers are sharing, and peers send queries to the server to search the content they want. In Napster, a large cluster of dedicated central servers is used to maintain the indices of the shared files. Each peer connects to one of these servers when it joins the system, uploads the index of its shared files to the server, and sends queries to the server. The server responds with a list of matched files with the location of these files to the peer. Upon receiving the response, the peer selects one file from the return list and initiates a file downloading. The server also monitors the state of peers through the connection and sends related information within the response message. The file transmission is between the clients without passing through the server. Fig. 1 shows the architecture of a centralized, index-based P2P system.

Napster was closed in 2002 due to legal issues. Open-Napster (OpenNap) (11) is an open source project that extends the Napster protocol for file sharing.

Index-based peer-to-peer systems are not scalable and are prone to a single point of failure, which can be overloaded due to flash crowd or attacked by malicious users.

DECENTRALIZED AND UNSTRUCTURED P2P SYSTEMS

To circumvent the limitations of centralized peer-to-peer systems, the P2P community has developed decentralized peer-to-peer systems. Instead of maintaining a huge index in a central server or server cluster for the search service, a decentralized system distributes searching and locating loads across the participating peers. In such systems, peers self-organize into an overlay network to communicate with each other.

The *overlay network* of a peer-to-peer system is a logical network on top of the Internet. Each peer selects a number of peers as its neighbors to connect to in order that the departure of a single peer cannot disconnect the peer from the overlay network. A *host-cache site*, which maintains a list of active peers in the system, works as the *bootstrap site* of the P2P system, which provides an entry for new peers to join the system. When a peer wants to join the system, it connects to the host-cache site to get a list of peers and randomly selects a number of peers to connect to. A peer may try to connect to other peers when some of its neighbors leave or may accept connection requests from other peers. We call such a P2P overlay an unstructured P2P network because the connections of peers are random and do not follow any rules. However, in practice the connectivity of peers in the overlay is not randomly distributed. Instead, the node degree of the overlay topology graph is heavily skewed due to the heterogeneity of the lifetime and computing capacity of peers. Research has shown that the node connectivity of many unstructured P2P networks follows a two-phrase Zipf-like distribution, as shown in Fig. 2. This kind of overlay is highly resilient to random node breakdowns but is vulnerable to attacks that target those highly connected nodes.

Each peer in a P2P overlay network is not only a servent but also a *router* that forwards messages it receives to its neighboring peers. In this way, a message can travel the network to reach the destination peer whose location is unknown to the sender. Message forwarding enables content search over the P2P overlay network. We briefly introduce three well-known peer-to-peer search algorithms for unstructured P2P networks, namely, flooding, super node, and random walk.

Flooding is a broadcast mechanism for the P2P system, such as Gnutella. In the flooding algorithm, a peer sends a message to its neighbors, which in turn forward the message to all their neighbors except the message sender. Each message has a unique message ID. A message received by a peer that has the same message ID as the one received previously is considered a redundant message and will be discarded. Flooding is conducted in a hop by hop fashion counted by Time-to-Live (TTL). A message starts off with its initial TTL, which is decremented by one when it travels

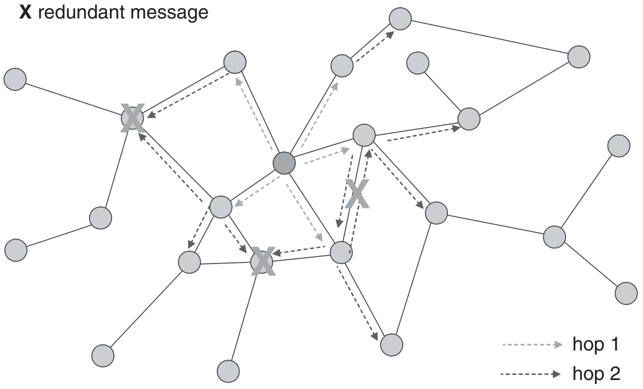


Figure 3. A two-hop flooding in an unstructured P2P network.

across one hop. A message comes to its end either because it becomes a redundant message or because its TTL is decremented to 0. The default initial TTL value is 7 since 7-hop flooding can cover more than 90% of the nodes in the P2P network. Fig. 3 shows the message flooding on the P2P overlay network.

Flooding is very simple and very effective. However, it may cause a great amount of redundant or unnecessary traffic. It has been estimated that routing traffic for the Gnutella network was about 1.7% of the total traffic in the U.S. Internet backbones in December 2000. To reduce the flooding traffic in a P2P overlay, many P2P systems, such as KaZaA, Morpheus, and current Gnutella, adopt a *super node* architecture. A super node is a proxy and index server of a number of leaf nodes. A peer connects to one or several super nodes to join the overlay network. The super node maintains the indices of its leaf nodes, and queries are only flooded in the super node network, in order to limit the flooding scope. Although a super node may leave the system at any time, a peer can still maintain the connection to the overlay network by connecting to several super nodes simultaneously. Fig. 4 shows the super node of the P2P overlay network.

Random walk is another approach to reduce search traffic. The content distribution in the system is heavily skewed, and popular objects have more copies in the system than unpopular ones. Most queries in P2P systems are for popular objects, which are distributed redundantly in the system. Thus, it is unnecessary to travel every node in the overlay to find the information a peer needs. In the random walk search approach, several walkers randomly travel the network in parallel and forward the query initiated by the sender along the travel path. An improved random walk algorithm is the *biased random walk*, in which each peer maintains the indices of its one-hop neighbors, and in which the query is routed to the node with higher connectivity randomly. Thus, the query is routed to the nodes with highest connectivity quickly, and the indices in these highly connective nodes can satisfy the query with high probability. Although random walk has the least communication traffic for message routing, it may result in a long response time. As a result, it has not been practically implemented so far. By constructing a content abundant cluster (CAC) on top of the entire P2P network, which consists of those peers with more objects than other peers, CAC can also reduce the search scope without increasing the average query response time (12).

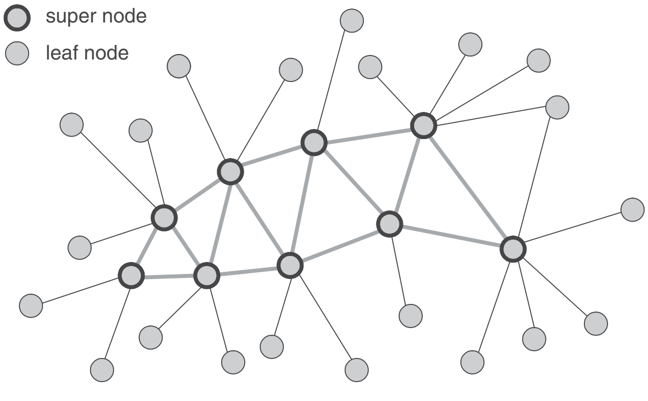


Figure 4. The super node architecture for an unstructured P2P network.

DECENTRALIZED STRUCTURED P2P SYSTEMS

The lack of global data management in unstructured P2P systems makes content locating inefficient and expensive. Decentralized structured P2P systems organize peers in the system into a *distributed hash table* (DHT), which supports hash table-like operations in the overlay network. In structured P2P systems, each node maintains a routing table that is determined by the overlay structure of the distributed hash table. Each object is placed in a unique location in the system based on its key value and can be reached by routing query between nodes, according to the DHT routing rules. The object and its key can be maintained by different peers. The key idea is that the key space is organized as a hierarchical structure so that the key search can be conducted efficiently. A one-dimensional distributed hash table, such as Chord, Pastry, and Tapestry, uses skiplist-like routing or tree-like routing to pass the query to the destination node. Such distributed hash tables can provide $O(\log n)$ lookup with each node maintaining $O(\log n)$ routing table entries. The content addressable network (CAN) uses a multi dimensional mapping mechanism that can provide $O(dN^{1/d})$ lookup with each peer maintaining $O(d)$ routing table entries, where d is the dimension of the system coordination space and N is the number of nodes in the system.

Fig. 5 shows a two-dimensional CAN structure. Keys are mapped into a two-dimensional Cartesian space, with each rectangle of the coordination zone representing a fraction of the entire key space. Each object is mapped into a key in the two-dimensional Cartesian space one-to-one. Each rectanglular zone in the key space is assigned to a peer, which maintains the objects mapped to the keys in this rectangle. The figure shows how a message is routed from coordinate (0.4,0.1) to (0.9,0.7) in the overlay network. Each node maintains a routing table, where each entry corresponds to the coordination zones of one of its neighbors. Intuitively, routing in a CAN overlay is performed by following a

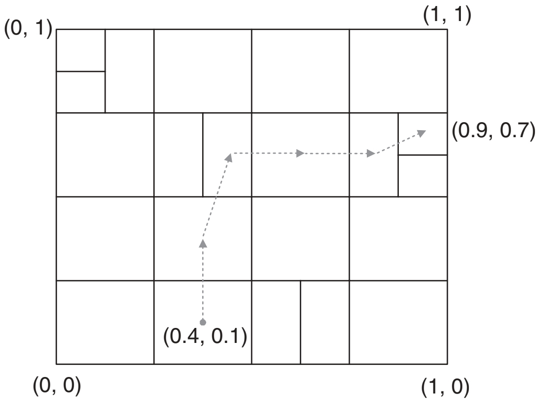


Figure 5. The content addressable network.

straight line in the vertical dimension and then a straight line in the horizontal dimension from the source node to destination node in a decentralized way.

The typical popular distributed hash tables are Tapestry (13), Pastry (14), CAN (15), and Chord (16). Distributed hash tables are often used as an infrastructure to construct large-scale distributed file systems or storage systems. Although early P2P file-sharing systems were usually unstructured, recently distributed hash tables have also been used in P2P file sharing; for example, Overnet uses Kademlia (17) for content search.

BITTORRENT: A SWARMING-BASED P2P SYSTEM

Early P2P file-sharing systems, for example, Napster, Gnutella, KaZza, and eDonkey/eMule/Overnet are basically exchange-based. In exchange-based P2P systems, peers share and exchange different files with each other.

BitTorrent is a swarming-based of P2P system that has become very popular recently. As reported by CacheLogic, BitTorrent traffic represented 53% of all P2P traffic on the Internet in June 2004 (18). Unlike traditional P2P systems such as Napster (1), Gnutella (2), and KaZaa (3), which use various search protocols to find a target file, BitTorrent organizes peers that share the same file into a P2P network and focuses on an efficient replication mechanism to distribute the file among them.

BitTorrent uses parallel downloading techniques to speed up content distribution. By dividing a file into small chunks, a peer can download multiple parts of the file in parallel, which enhances the efficiency of file distribution. Once a peer completes downloading, it becomes a *seed* of the system. BitTorrent uses a "tit-for-tat" incentive mechanism, which enables peers with high uploading bandwidth to have correspondingly high downloading bandwidth. The incentive mechanism of the BitTorrent system effectively prevents free riding, which is the most important difference between BitTorrent systems and other systems. In practice, BitTorrent-like systems scale fairly well during flash crowds and are now widely used for various purposes, such as for distributing large software packages (19).

In a BitTorrent system, a content provider creates a *meta file* (with the torrent suffix name) for the *data file* it wants to share and publishes the meta file on a website. Then the content provider starts a BitTorrent client with a full copy of the torrent file as the original *seed*. For each data file, a *tracker site* is used to help peers find each other to exchange the file chunks. A user starts a BitTorrent client as a *downloader* at the beginning in order to download file chunks from other peers or seeds in parallel. A peer that has downloaded the file completely also becomes a seed that could, in turn provide a downloading service to other peers. All peers in the system, including both downloaders and seeds, self-organize into a P2P network, known as a *torrent* (or a swarm). The initial seed can leave the torrent when other seeds are available; the content availability and system performance in the future depend on the arrival and departure of downloaders and seeds. With the decrease of content popularity over time, the downloading speed of the file may become poor or even unavailable because of the decrease of the number of peers sharing the file (20).

SUMMARY

Peer-to-peer communications are applications that use a self-organized protocol to connect a large number of end machines to share resources among them, where each peer is both a client and a server. The applications of peer-to-peer communications have expanded from initial music file exchanges to large software distribution, live streaming video, and VoIP. Traditional media and movie industries are also considering using peer-to-peer techniques to distribute their content with the protection of copyrights. We believe that peer-to-peer communication represents a common and cost-effective trend on the Internet.

BIBLIOGRAPHY

1. http://www.napster.com/.
2. http://www.gnutelliums.com/.
3. http://www.kazaa.com/.
4. http://www.edonkey2000.com/.
5. http://bittorrent.com/.
6. E. Adar and B. Huberman, Free riding on Gnutella, Technical report, Xerox PARC, August 2000.
7. Skype–Internet calls, http://www.skype.com/.
8. S. Ren, L. Guo, and X. Zhang, ASAP: An AS-aware peer-relay protocol for high quality VoIP with low overhead, *Proc. 26th International Conference on Distributed Computing Systems*, July 2006.
9. PPlive–the largest world wide Internet tv network home. http://www.pplive.com/en/index.html.
10. X. Hei, C. Liang, I. Liang, Y. Liu, and K. W. Ross, Insight into PPlive: Measurement study of a large scale P2P IPTV SYSTEM, *PROC. WWW 2006 WORKSHOP OF IPTV SERVICES OVER WORLD WIDE WEB*, 2006.
11. OpenNap: Open source Napster server. http://opennap. sourceforge.net/.
12. L. Guo, S. Jiang, L. Xiao, and X. Zhang, Exploiting content localities for efficient search in P2P systems, *Proc. 18th International Symposium on Distributed Computing*, October 2004, pp. 349–364.

13. B. Zhao, J. Kubiatowicz, and A. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing, *in Report No. UCB/CSD-01-1141*, April 2001.

14. A. Rowstron and P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, *Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001, pp. 329–350.

15. S. Ratnasamy, P. Francis, M. Handley, and R. Karp, A scalable content-addressable network, *Proc. ACM SIGCOMM 2001*, San Diego, CA, August 2001, pp. 161–172.

16. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for Internet applications, *Proc. ACM SIGCOMM 2001*, San Deigo, CA, August 2001, pp. 149–160.

17. P. Maymounkov and D. Mazieres, Kademlia: A peer-to-peer information system based on the XOR metric. *Proc. 1st International Workshop on Peer-to-Peer Systems*, March 2002.

18. A. Parker, The true picture of peer-to-peer file sharing. Available http://www.cachelogic.com, 2004.

19. M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garc'es-Erice, Dissecting BitTorrent: Five months in a torrent's lifetime, *Proc. 5th Annual Passive & Active Measurement Workshop*, April 2004.

20. L. Guo, S. Chen, Z. Xiao, E. Fan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of BitTorrent-like systems, *Proc. Internet Measurement Conference 2005*, October 2005, pp. 35–48.

Lei Guo
Xiaodong Zhang
The Ohio State University
Columbus, Ohio

PROGRAMMING MODELS: CLIENT–SERVER, PROCESS GROUPS, AND PEER-TO-PEER

INTRODUCTION

Programming a distributed application is often significantly different from programming an application intended for a single machine. Accessing a remote object may require locating the object first. The object may not currently be available, and even if it is, access latency may be high or unpredictable in case of slow or overloaded network connections. The rate of access to an object may grow unreasonably high as the number of clients in the network grows, overrunning the physical resources that implement the object. Security may be an additional issue, as the network is easily accessed by unrelated third parties. As a result, programming a distributed application is often significantly harder than programming a centralized one, and thus, much thought has been given to how to make distributed programming easier.

The most popular approach to achieving this simplicity is to make distributed programming similar to centralized programming, which is also known as transparency. For the execution of any kind of program, there are two basic ingredients: memory and processing. The memory contains the data structures that are used by the program, whereas the processing implements the program's algorithms. In this essay we focus on memory and the complications of distributing this memory to a set of machines connected only by a network.

Perhaps the most obvious approach is to implement a shared virtual memory address space that can be used by all processes involved in the distributed application. Indeed, this approach exists and is known both as shared virtual memory and distributed shared memory (1). It is typically implemented using hardware page protection bits and intercepting memory locking operations. The approach is valid for homogeneous sets of mutually trusting processes and thus best applied in parallel applications running on CPU clusters.

Instead, we will be focusing on the more general case of sharing arbitrary storage objects among a distributed set of processes. In our terminology, an object has state and a set of operations or methods that operate on the state. Distributed shared memory then is a special case, in which the object is a memory page and the methods are read, write, lock, and unlock operations on the page.

An object is implemented by one or more servers and accessed by one or more clients. Clients send requests for operations to servers, and servers return results to the clients, which is called the client–server model. Typically, servers run on different machines than clients, but this is not a requirement. However, remote access is sometimes necessitated by geographic, reliability, and/or security considerations.

The client–server model is ubiquitous in today's distributed systems. Examples include shared file systems, shared database systems, e-mail, domain name resolution, and of course the World Wide Web. In each of these cases, the specific network protocols are different, but they all involve client–server interactions and allow a set of clients to share and access storage resources.

THE CLIENT–SERVER PARADIGM

The most prominent implementation of the client–server model today is the one where individual server processes implement services. Each server may have many clients. Perhaps the best known example is the Web server, but other examples include FTP, TELNET, SMTP, and shared databases. The server maintains one or more resources or objects. Clients send requests to access or modify the objects or even to have the server do some processing on behalf of the clients.

A server may be also be a client to another server. For example, a Web server may need to access a remote database in order to retrieve data for one of its Web clients. In the case of e-mail, a client sends a request to post an e-mail message to an SMTP server. The server, in turn, has to send a request to the recipient's POP server to deliver the message. In the case of domain name resolution, one domain name service (DNS) server may need to ask another server for the information it needs to satisfy the request of a client, and this can continue recursively.

In Fig. 1, we show a typical example of client–server interactions. A Web browser first looks up a DNS name by sending a request to a DNS server. The DNS server then recursively may use another DNS server. After receiving a response containing an address, the Web browser sends a request to the corresponding Web server. The Web server may in turn invoke several application servers, which may in turn interact with a set of databases. In all of these cases, a specific protocol has been developed for the interactions between a client and a server.

In 1984, Andrew Birrell and Bruce Nelson developed a paradigm called remote procedure call (RPC) (2), which makes these interactions look like normal procedure calls from a client's code into the server's code. To the client, the only difference is that a new exception may be raised such as that the server is not available. To the server, each client's invocation typically looks like a separate thread that is spawned within the server's address space.

The way this process works is quite simple. On the client end, a stub procedure is created for each of the procedures that the server exports. The stub collects all arguments into a request message and sends this message to the server. The stub procedure now blocks the client process while awaiting the reply. At the server, the message is unpacked and the procedure is invoked. The result is placed into a reply message that is sent back to the client. On receipt, the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

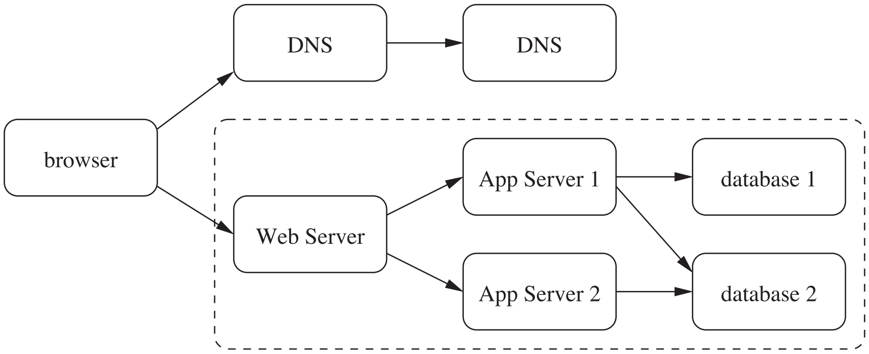


Figure 1. Typical client–server interactions when accessing the WWW. Arrows point from clients to servers.

stub retrieves the result from the message and resumes the client process.

Despite its simplicity, often some issues are tricky to resolve. For example, how does the client's stub procedure determine to which server to send the request? What if the client wants to operate on two different objects stored at two different servers? What if the client specifies as its argument a reference to a very large object within its address space? How are unreferenced objects released (distributed garbage collection)?

Many RPC platforms have been developed over the years. Currently, the best known are Web Services, Java Remote Method Invocation (RMI), and CORBA. Web Services builds on top of the HTTP protocol used to access Web servers and uses XML for data representation within the messages. RMI uses Java's automatic serialization of Java objects to create request and response messages. CORBA (3) is a language-independent framework that uses an Interface Description Language (IDL) to describe a server's methods and its parameters. An Object Request Broker (ORB) automatically locates objects and activates servers if necessary.

Besides RPC, another common form of communication is the one in which a server communicates a stream of events to its clients. Rather than polling a server for updates to objects, a client can be notified at the time such updates occur. A popular form of such event notification is known as publish/subscribe and is treated in more detail below.

PROCESS GROUPS

Although the client–server paradigm simplifies the structuring of distributed applications, it does little to deal with the realities of scale. Servers are centralized processes that can easily become bottlenecks and are also single points of failures. Many clients may not get the service they require if a server becomes overloaded or crashes. Both problems can be addressed by using a collection of machines to implement a service.

A process group is a set of processes cooperating on a common task, such as the maintenance of one or more shared objects. The processes are commonly called the members of the group. The notion of process group was first introduced in the V system (4). The V system supported an unreliable multicast mechanism by which one member could send a message to all members of a group. More recent process group implementations support various forms of reliable communication.

Process groups can be created for a variety of purposes. For example, a service could be made fault tolerant by creating multiple copies of the server. By grouping the servers into a process group, the reliable communication properties can be exploited to keep the servers in sync with one another. For dealing with high load, a service can be partitioned so that each server is responsible for a subset of objects. Process groups can also be used for clients. An example is an event notification service, in which the multicast capabilities of a process group can be used to disseminate events to a group of clients. Below we look at two popular incarnations of process groups, namely Virtual Synchrony and Publish/Subscribe.

Virtual Synchrony

Virtual Synchrony (VSync) was introduced in the Isis system (5). VSync presents a model of a distributed computation in which an execution of a program is divided into self-contained failure-free epochs. At the beginning of an epoch, each process is notified about the current membership or view of the epoch, that is, the set of processes that participate in the computation and are alive and reachable during the epoch. The only messages that may be delivered during an epoch are messages that were sent in that epoch by the initial members of that epoch. Processes may only fail by crashing at the end of an epoch; in which case, they will not participate in the next epoch. Finally, no message loss occurs in an epoch, and all messages sent in an epoch must be delivered before the end of the epoch.

In asynchronous distributed environments, messages may be arbitrarily delayed, lost, reordered, or duplicated; processes may crash or become arbitrarily slow or otherwise unavailable at any given moment; and the network may even partition. Clearly, the pure model as stated above cannot be implemented in these environments. However, it is possible to implement a non-blocking emulation, in which a process cannot tell the difference between the observed execution and an execution in which no failures occur.

In Fig. 2, we show an example of an epoch with five processes A through E. Time is from left to right. The actual

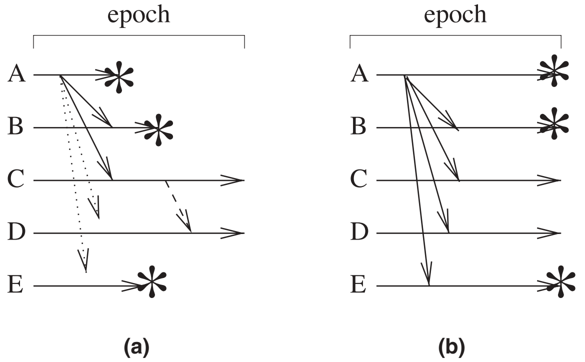


Figure 2. (a) Actual execution. (b) VSync execution. Arrows indicate messages. "*" indicates a process crash. Dotted arrows indicate lost messages, whereas the dashed arrow indicates a message retransmission.

execution is shown in Fig. 2(a). Process A sends a message. Only processes B and C receive the message. Processes A, B, and E crash soon afterward. The VSync protocol will detect the crashes and end the epoch but not before sending a copy of the message from C to D. As a result, C and D cannot distinguish the execution from that of Fig. 2(b), in which no failures occurred. This behavior allows considerably simplified algorithms of several important distributed paradigms, such as replication, leader election, and voting.

One of the most prevalent uses of VSync is replicating objects, in which the state of an object is copied among a set of servers. To keep the replicas consistent with one another, they have to start in the same state and execute updates in the same order (6). Most VSync implementations therefore support state transfer and totally ordered multicast. State transfer mechanisms allow newly joined members to receive a copy of the state at the beginning of an epoch. Totally ordered multicasts allow the members of the group to apply all updates in the same order.

Although simple, VSync has significant scalability problems, as both the rate of failures and the recovery time grow with the size of the membership of the group.

Publish/Subscribe

Another popular paradigm for building distributed applications is publish/subscribe (Pub-sub) (7). In this paradigm, publishers post messages, whereas subscribers specify what messages they are interested in. The pubsub system is responsible for routing the publisher's messages to the corresponding subscribers. Pubsub can be broadly classified into two types. In topic-based pubsub (TPS), subscribers specify what topics they are interested in. Topics are typically indicated by a string name. A publisher specifies a topic for each message that it sends. TPS is essentially a multicast mechanism, in which the multicast address is the topic.

In content-based pubsub (CPS), each message contains a set of attributes. By specifying a predicate over such attributes, a subscriber indicates which messages it is interested in. Note that TPS is easily implemented over CPS by having each message contain an attribute that indicates the topic. CPS is capable of much more precise addressing than TPS but often at much increased routing overhead.

Pubsub has become the backbone of many datacenters. It has gained its popularity because little structure is imposed on the applications. Subscribers can come and go, and the publisher does not need to be aware of the set of subscribers or where they reside. Similarly, publishers can migrate from one machine to another without having to notify the subscribers. A publisher simply posts messages and subsequently can forget about them. This lack of structure makes it very easy to glue together various applications within a datacenter and to scale the system up to virtually any size.

For these reasons, datacenters use pubsub not only as a multicast communication mechanism to keep distributed data such as caches and configuration files up to date but also for point-to-point communication between services. Besides the flexibility afforded by not specifying explicit network addresses, pubsub makes it easy to listen in on point-to-point communication for the purposes of monitoring and debugging.

Essentially, pubsub is a shared object paradigm, in which an object is some implicit state that is shared among a set of processes. Either this state is updated by publishing a request message that specifies the update to be performed on the state or the result of an update is notified after the fact by publishing an update notification message. Subscribers specify which state changes they are interested in, either by noting the topic (TPS) or by using a predicate (CPS).

Herein lie some problems with the pubsub paradigm. As the publisher does not know the set of receivers for any particular message, it cannot guarantee that the message is delivered to all receivers. Even it it did, the publisher could crash before it can ensure that the message is received by all subscribers. For the same reasons, no ordering can be guaranteed among the set of subscribers, and so the shared state may be observed differently by different subscribers. Such problems can lead to rare, unexpected executions that are hard to track down and debug.

PEER-TO-PEER (P2P)

The proliferation of home computers and home Internet connections has made an enormous number of connected storage and computing resources available. Peer-to-peer techniques aim to exploit this availability by providing mechanisms to harness all these resources. P2P intends to provide global services to any client anywhere at any time. This goal is complicated because the resources provided by home computers and home Internet connections are often highly unreliable, highly heterogeneous, and highly susceptible to malicious exploitation. Building a large reliable system out of such components is a significant challenge, but one in which significant progress has been made.

In pure P2P systems, hosts that traditionally are exclusively clients also become servers. Together, they provide a large storage and/or execution facility. Again, we will focus on storage, but projects that focus on computing exist as well, such as SETI@HOME, which harvests unused home computer cycles to search for signs of extraterrestial intelligence in radio telescope data. P2P techniques have become highly popular for sharing music and video, and this has driven much of the initial P2P protocol development. Other drivers include censorship concerns and anonymity. Today, one of the most popular P2P applications is Skype, which provides Internet telephony.

The first widely used P2P protocols, such as Gnutella, are simple. As a distinction between clients and servers no longer exists, we will call the processes that execute the P2P protocol agents. Each agent connects to a small, more or less random set of other agents, and thus a graph of agents emerges. To search for a file, a request is flooded from one agent up to a certain depth in this graph. Any successful matches are returned to the originating agent. By caching results, the most popular files will be found easily. But the flooding protocol is inefficient both in terms of resources used and in how long it can take to obtain a result, and rare files can be very hard to find.

The first more structured approaches were actually proposed before the development of Gnutella. The most famous result is that of Plaxton et al. (8), which presents a technique for finding nearby replicas of objects. The basic idea is to create a user-level routing framework for messages. Messages are addressed to objects' identifiers. Each agent is set up with a routing table that allows it to forward incoming messages to other agents if it does not store the requested object itself.

Peer-to-peer protocols come in various shapes. They can be roughly classified as

- Resource sharing, location, and search
- Application-level routing
- Monitoring and aggregation

SETI@HOME is a resource-sharing protocol in which clients contribute their CPU resources to a public cause. Various large-scale P2P storage services have been proposed as well, in which clients offer capacity on their disks that can be used for cheap backup or caching of public data. Such services could also be used to make censorship-sensitive material available in an anonymous and reliable way. Other P2P services, like Gnutella and BitTorrent, make clients' resources available for public access without offering public storage. Such services focus on location and search, and many audio and video sharing facilities are good examples.

Application-level routing is another interesting area for P2P protocols. Although the Internet essentially supports only unicast routing between hosts, P2P protocols can provide both unicast and multicast routing with a rich set of addressing and routing options not found in the Internet. Many such protocols have been developed, specializing in such features as location-independent routing, optimizing latency or bandwidth, fault-tolerance, security, and anonymity.

Although multicast and pubsub protocols disseminate data from a sender to a set of receivers, it is often required to retrieve information from a set of processes or objects and to return the result to a single process. For example, in a sensor network, it may be necessary to calculate the average temperature in a particular geographical area. For such applications, P2P protocols have been developed that allow clients to query a set of objects and aggregate the results. Such systems often support standing queries that report updates of aggregates.

Programmers can use the sharing, location, routing, and aggregation paradigms to build various distributed, collaborative services. Unfortunately, so far only preliminary proposals have been made toward standardizing the interfaces to these P2P paradigms, which complicates widespread adoption.

CONCLUSION

Whether you use client–server, process groups, or P2P techniques, distributed programming revolves around the maintenance of shared objects. An object can be low level such as a CPU, memory, or any hardware device or high level such as a spreadsheet, a Web page, or an entire running application. The objects have a state that can be centralized in one location, copied in various locations (replication), or partitioned across various locations. The state in turn is manipulated through procedure calls.

Depending on your performance and reliability requirements, either RPC, process groups, or P2P, or some combination, may be the best approach to implementing your application. Client–server style request–response interactions such as RPC are routinely used, for example, in the implementation of the World Wide Web and for the resolution of domain names using DNS.

Process groups are used within datacenters for the management of replicated services and in the implementation of publish/subscribe. P2P techniques are popular among home Internet users for file sharing and Internet telephony. A developer of distributed applications needs a thorough understanding of each of these techniques.

BIBLIOGRAPHY

1. K. Li and P. Hudak, Memory coherence in shared virtual memory systems. *Proc. Fifth ACM Symp. on Principles of Distributed Computing*, Calgary, Alberta, Canada, Aug. 1986, pp. 229–239.
2. A. D. Birrell and B. J. Nelson, Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, **2**(1): 39–59, 1984.
3. S. Vinoski, CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communi. Mag.*, **35**(2): 46–55, 1997.
4. D. Cheriton and W. Zwaenepoel, Distributed process groups in the V kernel. *ACM Trans. Comput. Syst.*, **3**(2): 77–107, 1985.

5. K. P. Birman and T. A. Joseph, Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, **5**(1): 47–76, 1987.

6. F. B. Schneider, Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, **22**(4): 299–319, 1990.

7. B. Oki, M. Pfluegl, A. Siegel, and B. Skeen, The information bus—an architecture for extensible distributed systems. *Proc. Fourteenth ACM Symp. on Operating Systems Principles*, Asheville, NC, Dec. 1993, pp. 58–68.

8. C. G. Plaxton, R. Rajaraman, and A. W. Richa, Accessing nearby copies of replicated objects in a distributed environment. *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 1997, pp. 311–320.

ROBBERT VAN RENESSE
Cornell University
Ithaca, New York

QUEUEING THEORY

INTRODUCTION

Waiting is a common phenomenon in our daily lives. People wait in a post office, at an elevator, at traffic lights, and so on. Airplanes wait to take off. Parts wait to be assembled. Data wait to be transmitted. Taxis wait for passengers, and passengers wait for taxis.

A system in which waiting phenomena exist is called a queueing system. (The word "queueing" is probably the only American English word that contains five vowels in a row). Queueing theory is an academic discipline that studies queueing phenomena in a systematic way.

A queueing system is composed of a set of customers, a set of server(s), and a service discipline (also called an order of service). A doctor is a server, and patients are customers. Failed machines are customers, and a repairman is a server. Data packets are customers, and the router is a server.

The first systematic treatment of a queueing system was carried out by A. K. Erlang (1878–1929) in early 1900s when he studied the congestion phenomena in a telephone exchange (1-3). See also Ref. (4).

In computer science, queueing theory is at the heart of analyzing the performance of several systems such as a router that routes data packets in a computer network (5), a communication switch that switches data or voice packets in a computer or telecom network, a data storage system (such as a hard disk) that serves requests for data access, a file server that serves user requests for access to files, a web server that serves requests for web content, and so on.

Queueing theory is critical to analyze these and other computer systems to ensure efficient resource usage. The arrival pattern of requests for service (be it any service) is rarely deterministic; it is mostly a random process. Eliminating wait time (when there will be no need to use queueing theory to analyze the performance) will in most cases require provisioning the resource for peak usage. However, in such a design, the system will be idle most of the time (except during peak times) and, hence, it will lead to a waste of resources. In several systems, introducing even a small amount of wait period (using buffers), can significantly bring down the amount of resources needed to serve the user requests. This resource efficiency is one primary reason for the prevalence of queueing theory in analyzing the performance of computer systems.

Another reason for the prevalence of queueing theory in computer system analysis is the need to make quality-of-service (QoS) guarantees (or to provide guidance on expected performance) to users. QoS can include a guarantee on the response time, a guarantee on the minimum throughput, and a guarantee on the minimum availability, among others. Making any of these guarantees for a given design typically involves analyzing the system for its queueing behavior.

In summary, queueing theory is an important analytical tool in computer science that has a wide variety of applications. In this article, we provide an introduction to queueing theory. We discuss key concepts and major results for the M/M/1, M/M/1/K, M/M/c, M/M/c/c, M/G/1, closed queueing networks, and open queueing networks. We conclude with a brief discussion of current research interests in this area.

A QUEUEING SYSTEM

A system is an organization in which entities interact with each other. A queueing system is a system in which customers arrive, wait, receive service, and leave. Customers and servers interact with each other in such a way that if the service time of a customer is delayed, other customers end up waiting longer, and if the server momentarily stops serving customers, the number of waiting customers may increase.

Characteristics of a Queueing System

The distinguishing characteristics of a queueing system are as follows:

Customer Classification. In many cases, customers form a single class. But, if there are multiple types of services, or if the customers have priorities, then customers may be classified according to their types of service and/or priorities.

Population Types. Customers can be from a finite population or an infinite population. As an example of a finite population, consider a factory where there are three machines. As soon as the machines break down, they are sent to the repair shop. Then, the repair shop becomes a queueing system and the broken machines are customers. In this system, only three machines can be customers.

No definite distinction is made between finite and infinite populations. The customers to a hardware store in a town are from a finite population to be exact. But if the population size is large, they can be thought of as the customers from an infinite population. In general, a queueing system with a finite population is more difficult to analyze.

The Arrival Process. When the interarrival periods are independent and identically distributed (i.e., a renewal process), the arrival process is represented by the interarrival time distribution. The arrival rate is defined as the average number of arriving customers per unit time. This rate is the reciprocal of the average interarrival time. In some special cases, arrival rates may vary depending on the number of customers in the system. This rate is called the state-dependent arrival rate (or load-dependent arrival rate). Arrival rate can vary as a function of time, as well. This arrival process is called nonstationary. Restaurants are a good example.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

In some cases, customers arrive in groups (batch arrival).

All arriving customers may not enter the system, and some may leave without receiving service. The average number of customers that enter the system (i.e., receive service) per unit time is called the input rate or effective arrival rate. If all arriving customers enter the system, the arrival rate is equal to the input rate. One rich application area of queueing theory is computer networks. In many cases, arrival processes in the communication systems are not renewal processes; i.e., interarrival times are neither independent nor identically distributed.

The Service Process. In many cases, service times of customers are assumed to be independent and identically distributed. In this case, as soon as a customer arrives, a random sample is generated from the service time distribution and the sampled value is assigned to the customer as its service time. This service time is independent of his waiting time or the number of customers in the system.

The service rate is defined as the average number of customers that can be served per unit time, which is the reciprocal of the average service time. The service rate can vary as a function of the number of customers in the system (state-dependent service rate or load-dependent service rate).

The service times of customers may be determined by the customers or by the servers. From the modeling point of view, we do not need to differentiate between these two. The service times may not be identically distributed for all customers. In this case, the complexity of analysis increases.

In many systems, customers depart the system as soon as they have received service. But, in some systems, the serviced customers may return to the queue (are fed back) for another round of service. A good example is the inspection system in which defects are returned for reprocessing.

Customers may be served one at a time (single-unit service) or in batches (batch service, bulk service).

Usually, one server is dedicated to one customer or one customer group. But, there are some cases where more than one server is assigned to serve a customer.

Number of Servers. There are cases of a single server, multiple servers, and an infinite number of servers. In many multiple-server systems, the servers are identical. In a queueing system with an infinite number of servers, customers are served immediately on arrival.

Service Discipline. A service discipline is a rule that stipulates how the next customer is selected at a service completion or in the middle of a service. Typical service disciplines are FCFS (first come, first served), LCFS (last come, first served), RSS (random selection for service), and PR (priority). Other service disciplines are SJF (shortest job first), LJF (longest job first), SRPT (shortest remaining processing time), and so on.

Customer Behavior. Customers may not enter the system for some reason (balk). They may even renege from the system (i.e., leave without receiving service). In the case of forced balking, we say that the customers are *blocked* or *lost*. If multiple waiting lines exist, customers may move from one to another (*jockeying*). Sometimes, blocked customers may retry to enter the system. For example, when a telephone line is busy, customers may hang up and try again.

Queue Structure. In most queueing systems, customers form one waiting line (single-queue system). However, multi-queue systems (i.e., one line for each server) do exist, as in some fast food restaurants. Also, situations exist where a single server serves multiple queues, in turn. A single-lane bridge is a good example. This bridge is analogous to the token-ring system.

Sometimes, the input/output of several queueing systems is interconnected to form a *queueing network*. If the queueing systems are connected in series, the network is called a *tandem queue*.

Performance Measures. Some important performance measures are as follows:

Queue length (system size, queue size). This is the number of customers in the system, including the one(s) in service, if any.

Waiting Time. The waiting time of a customer is the time from its arrival until it begins to be served. To obtain the waiting time distribution, we need to keep track of all the possibilities that the *test customer* (an arbitrarily selected customer) experiences.

Sojourn Time. Sojourn time is the time from the arrival until the customer departs the system. Usually, this time is the sum of waiting time and the service time. It is sometimes also called the *response time*.

Busy period. This period is the time when the server begins to be busy until it becomes idle.

KENDALL'S NOTATION

Until the 1950s, the authors had to list all the characteristics to explain a queueing system and a lot of confusion often ensued. Kendall suggested that a queueing system be specified by the notation (6):

arrival process/service process/number of servers

But the inconsistent notation among authors still caused a lot of problems. Several technical societies whose journals contained such articles recognized the problem and held a joint conference at Northeastern University in May 1971 to discuss the standardization of terminology and notation. The Operations Research Society of America recommended that the queueing systems be specified in the following notation:

(***arrival process/service process/number of servers***)

When necessary an appendage may be added of the form

(/***system capacity/size of calling population/service discipline***)

When the appendage is omitted, the infinite system capacity, the infinite size of the calling population, and the service discipline FCFS is assumed. The system capacity refers to the maximum number of customers that can exist in the system (usually including the one(s) in service). Calling population refers to the group of potential customers. The following notation is recommended.

M	Markovian. It denotes Poisson arrivals, exponential interarrival time, or exponential service time
E_k	Erlang distribution of order k
PH	Phase-type distribution
D	Deterministic (constant)
G	General (sometimes *GI* (General and Independent) is used instead)
Geo	Geometric
B	Bernoulli
MAP	Markovian arrival process
MMPP	Markov modulated poisson process
FCFS	First-come–first-served
LCFS	Last-come–first-served
RSS	Random selection for service

Sometimes FIFO (first in–first out) is used in place of FCFS, LIFO (last in–first out) in place of LCFS, and SIRO (selection in random order) in place of RSS.

Batch arrivals and batch services are denoted by superscripts.

Many queueing systems exist that cannot be described by the above notation. In these cases, authors invent their own notation.

As examples of the above notation, M/M/1 is a queueing system in which customers arrive according to the Poisson process, the service times are exponential, and there is one server. System capacity (∞), the size of the calling population (∞), and FCFS service discipline are omitted. For another example, M/G/1 is a queueing system in which customers arrive according to the Poisson process, the service times are general (i.e., not necessarily exponential), and one server exists. $M^X/G/1$ is an M/G/1 queueing system in which customers arrive in groups. (X denotes a group size. For example, customers arrive in taxis where the taxis arrive according to a Poisson process).

LITTLE'S FORMULA

Consider some "system" in which customers arrive, get served, and depart. Let $A(t)$ be the number of entering customers during t and $D(t)$ be the number of departing customers during t. The ith customer enters the system at a_i and departs the system at d_i. Let $A_a(t)$ and $D_a(t)$ be the average arrival and departure processes. If we assume that all entering customers depart the system only after their service is finished, we have a situation as shown in Fig. 1, where W is the mean time a customer stays in the system and L is the mean number of customers in the system at an arbitrary time. Then, we have

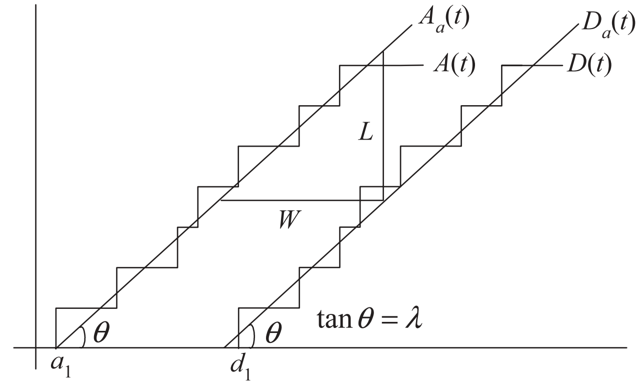


Figure 1. Little's formula $L = \lambda W$.

$$L = \lambda W \tag{1}$$

Equation (1) is called *Little's formula* in the queueing literature (7). We note that in Equation (1), λ is the *entering rate*, not the arrival rate (in some queueing systems, not all arriving customers enter the system).

THE EXPONENTIAL DISTRIBUTION

Let X be a random variable with the following distribution function (DF)

$$F(x) = Pr(X \le x) = 1 - e^{-\lambda x} \tag{2}$$

Then, X is said to be *exponentially* distributed. Its probability density function (pdf) becomes

$$f(x) = \frac{d}{dx}F(x) = \lambda e^{-\lambda x}, (x \ge 0) \tag{3}$$

The nth moment of X is given by

$$E(X^n) = \int_0^\infty x^n \lambda e^{-\lambda x} dx = \frac{n!}{\lambda^n} \tag{4}$$

Thus, the mean and variance become

$$E(X) = \frac{1}{\lambda} \tag{5}$$

and

$$\mathrm{Var}(X) = E(X^2) - E^2(x) = \frac{1}{\lambda^2} \tag{6}$$

The Momoryless Property

Let us assume that the lifetime of a light bulb follows exponential distribution. Let the bulb begin to operate at time 0. Knowing that the bulb has not failed until time t, the probability that it will not fail during the next s time units is given by

$$Pr(X > t + s|X > t) = \frac{Pr(X > t + s, X > t)}{Pr(X > t)}$$
$$= \frac{Pr(X > t + s}{Pr(X > t)} = \frac{e^{-\lambda(t+s)}}{e^{-\lambda t}} = e^{-\lambda s} \qquad (7)$$

From Equation (2), we know that $e^{-\lambda s} = Pr(X > s)$ is the probability that the bulb does not fail during (0, s). Thus, the fact that the bulb has not failed until t does not affect the future life of the bulb. In other words, as long as the bulb is working, it is just like the new one at every moment. This property is called the *memoryless property* of exponential distribution. It can be proved (mathematically) that the exponential distribution is the unique continuous probability distribution with the memoryless property.

THE POISSON PROCESS

If the interarrival times of the customers follow independent and identically distributed (iid) exponential distribution with mean $1/\lambda$, then we say that the customers arrive according to the Poisson process with rate λ. If we define $N(t)$ to be the number of such customers arriving in t time units, then we can show that

$$Pr[N(t) = n] = \frac{e^{-\lambda t}(\lambda t)^n}{n!}, \ (n = 0, 1, 2, \ldots) \qquad (8)$$

The Poisson arrival process is a completely random process, which means that customers arrive in a random way. This is similar to the memoryless property of the exponential distribution.

PASTA (POISSON ARRIVALS SEE TIME AVERAGES)

Suppose that customers arrive in a queueing system according to a Poisson process. Consider an arbitrarily arriving customer A. Also consider an outsider B who passes by the queueing system at an arbitrary time. PASTA (8) says that what is observed by A (arriving customer's distribution) is *stochastically equivalent* to what is observed by B (outsider's distribution). Thus, if we denote P_n as the probability that B observes n customers (this is equal to the probability that there are n customers at an arbitrary point of time) and $\bar{\pi}_n$ as the probability that A observes n customers just before its arrival, then,

$$\bar{\pi}_n = P_n \qquad (9)$$

Equation (9) is not guaranteed if arrivals do not follow a Poisson process.

For a more detailed discussion of the PASTA property, we refer the readers to Ref. (9).

M/M/1 QUEUEING SYSTEM

In an M/M/1 queueing system, customers arrive according to a Poisson process and the service times follow an exponential distribution. If we define λ to be the arrival rate, $\mu = 1/E(S)$ to be the service rate, and $X(t)$ to be the number of customers in the queueing system at time t, the stochastic process $\{X(t), t \geq 0\}$ becomes the birth–death process and we obtain a rate-flow diagram as shown in Fig. 2.

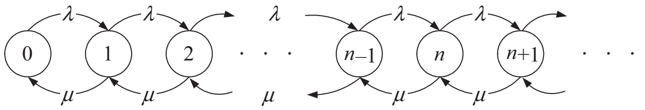


Figure 2. The rate-flow diagram of the queue length process of M/M/1.

Defining $P_n(t) = Pr[X(t) = n]$ to be the probability that there are n customers in the system at time t, we obtain the following system equations:

$$\frac{d}{dt}P_0(t) = -\lambda P_0(t) + \mu P_1(t)$$
$$\frac{d}{dt}P_n(t) = \lambda P_{n-1}(t) - (\lambda + \mu)P_n(t) + \mu P_{n+1}(t), (n = 1, 2, \ldots)$$
$$(10)$$

In the steady state, $\frac{d}{dt}P_n(t) \to 0$ and $P_n(t) \to P_n$. Thus, the steady-state system equations are

$$\begin{aligned} 0 &= -\lambda P_0 + \mu P_1, \\ 0 &= \lambda P_{n-1} - (\lambda + \mu)P_n + \mu P_{n+1}, (n = 1, 2, \ldots). \end{aligned} \qquad (11)$$

Interpretations of the system equations in Equation (11) are as follows:

First equation: λP_0 is the rate out of state 0 and μP_1 is the rate into state 0. These two rates should be equal in steady state.

Second equation: $(\lambda + \mu)P_n$ is the rate out of state n. $\lambda P_{n-1} + \mu P_{n+1}$ is the rate into state n. These two rates should be equal in steady state.

The solution to Equation (11) becomes

$$P_n = (1 - \rho)\rho^n, (n = 0, 1, \ldots), (\rho = \lambda/\mu) \qquad (12)$$

It is observed from Equation (12) that for the system to be stable, it is necessary to have

$$\rho < 1 \qquad (13)$$

Interpretation of Equation (13) is obvious if we note that $\rho = \lambda/\mu = \lambda E(S)$ is the average amount of work that is brought into the system per unit time and that 1 is the maximum amount of load that can be reduced per unit time by the server. We note that ρ also is the probability that the server is busy in steady state.

Performance Measures

Some performance measures of the M/M/1 queueing system are as follows:

Mean Queue Length.

$$L = \sum_{n=0}^{\infty} nP_n = \frac{\rho}{1-\rho} \tag{14}$$

Mean Number of Waiting Customers.

$$\begin{aligned} L_q &= \sum_{n=1}^{\infty}(n-1)P_n = \frac{\rho^2}{1-\rho} = \frac{\rho}{1-\rho} - \rho \\ &= L - L_{\text{in service}} \end{aligned} \tag{15}$$

In Equation (15), $L_{\text{in service}}$ is the mean number of customers being served at an arbitrary time; i.e.,

$$L_{\text{in service}} = (1)P_{\text{busy}} + (0)P_{\text{idle}} = P_{\text{busy}} = \rho \tag{16}$$

Mean System Sojourn Time.

$$W = \frac{L}{\lambda} = \frac{1}{\mu - \lambda} \quad \text{(Little's law)} \tag{17}$$

Mean Waiting Time.

$$W_q = \frac{L_q}{\lambda} = \frac{\rho}{\mu - \lambda} \quad \text{(Little's law)} \tag{18}$$

Alternatively,

$$W_q = W - E(S) = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda} \tag{19}$$

Behavior of the Mean Waiting Time

Figure 3 shows the mean waiting time as ρ varies (for simplicity, μ is fixed at 1). Notice that the mean waiting time increases sharply as ρ approaches 1, which means that when ρ is close to 1 (this function is called the *heavy traffic*), even a small increase in the arriving traffic results in an enormous amount of increased waiting time. From Little's law, we can make a similar conclusion regarding the mean number of waiting customers, the mean queue length, and the mean system sojourn time. This type of queueing behavior is typical of queueing systems and can be observed in many real-world systems.

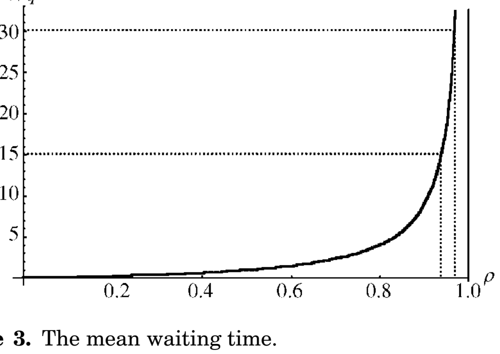


Figure 3. The mean waiting time.

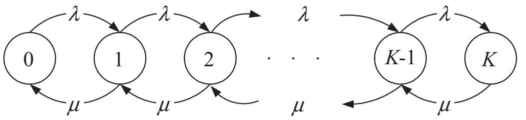


Figure 4. The rate-flow diagram of M/M/1/K.

For a more elementary introduction of the M/M-type queueing systems, we refer the readers to Refs. (10–12).

M/M/1/K QUEUEING SYSTEM

The M/M/1/K queueing system is the same as M/M/1 except that the maximum number of customers that can exist in the system is limited by K (including the one in service). Thus the customers who arrive when K customers are in the system are lost. The rate-flow diagram is as shown in Fig. 4.

In this system, the steady-state exists even when $\rho > 1$. Arriving customers cannot enter the system with probability P_k according to PASTA. Thus the entrance rate (effective arrival rate) becomes

$$\lambda_e = \lambda(1 - P_K) \tag{20}$$

Thus, the Little's formula in this case becomes

$$L = \lambda_e W = \lambda(1 - P_K)W \tag{21}$$

From the rate-flow diagram and using the approach of Equation (11), we can set up the steady-state system of equations as follows:

$$\begin{aligned} 0 &= -\lambda P_0 + \mu P \\ 0 &= \lambda P_{n-1} - (\lambda + \mu)P_n + \mu P_{n+1}, (n = 1, 2, \ldots, K-1) \\ 0 &= \lambda P_{K-1} - \mu P_K \end{aligned} \tag{22}$$

By solving Equation (22), we obtain

$$P_n = \begin{cases} \dfrac{(1-\rho)\rho^n}{1-\rho^{K+1}}, (\rho \neq 1) \\ \dfrac{1}{K+1}, (\rho = 1) \end{cases} \tag{23}$$

Mean Performance Measures

From Equation (23), we can derive the performance measures as follows:

Mean Queue Length.

$$L = \sum_{n=0}^{K} nP_n = \begin{cases} \dfrac{\rho}{1-\rho} - \dfrac{(K+1)\rho^{K+1}}{1-\rho^{K+1}}, (\rho \neq 1) \\ \dfrac{K}{2}, (\rho = 1). \end{cases} \tag{24}$$

Mean System Sojourn Time.

$$W = \frac{L}{\lambda(1-P_K)} \quad \text{(Little's law)} \tag{25}$$

Mean Number of Customers in Service.

$$L_{\text{in service}} = (0)P_0 + (1)(1-P_0) = \frac{(1-\rho^K)\rho}{1-\rho^{K+1}} \tag{26}$$

Mean Number of Waiting Customers.

$$L_q = L - L_{\text{in service}} \tag{27}$$

Mean Waiting Time.

$$W_q = \frac{L_q}{\lambda(1-P_K)} \quad \text{(Little's law)} \tag{28}$$

M/M/c QUEUEING SYSTEM

Customers arrive according to a Poisson process with rate λ. c identical servers exist, and the service times follow the exponential distribution with mean $E(S) = 1/\mu$. All customers wait in a single line. If an arriving customer observes multiple idle servers, it "randomly" chooses one server with equal probability. It is known that the departure process from the steady-state M/M/c queueing system is again a Poisson process with rate λ.

Figure 5 shows the rate-flow diagram for the queue length process of the M/M/c queueing system.

The system equations can be set up as in Equations (10) and (11). The steady-state queue length probability becomes

$$P_n = \begin{cases} \dfrac{\lambda^n}{n!\mu^n}P_0, (1 \le n \le c-1) \\ \dfrac{\lambda^n}{c^{n-c}c!\mu^n}P_0, (n \ge c) \end{cases} \tag{29}$$

where

$$P_0 = \left[\sum_{k=0}^{c-1}\frac{(\lambda/\mu)^k}{k!} + \frac{(\lambda/\mu)^c}{c!(1-\rho)}\right]^{-1}, \left(\rho = \frac{\lambda}{c\mu}\right) \tag{30}$$

The stability condition becomes

$$\rho = \frac{\lambda}{c\mu} < 1 \tag{31}$$

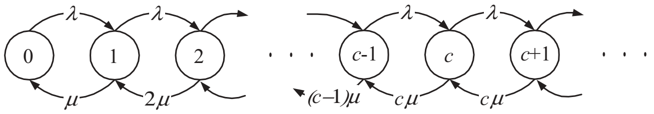


Figure 5. The rate-flow of M/M/c.

It is to be noted that ρ is the probability that server-i is busy at an arbitrary time for all i.

Erlang Delay Formula (Erlang C Formula)

From Equation (29) and from PASTA, the probability that an arbitrary arriving customer waits becomes

$$C(c,a) = \sum_{j=c}^{\infty}P_j = \frac{\dfrac{a^c}{c!(1-\frac{a}{c})}}{\displaystyle\sum_{k=0}^{c-1}\frac{a^k}{k!} + \frac{a^c}{c!(1-\frac{a}{c})}} \tag{32}$$

where

$$a = \lambda E(S) = \lambda/\mu \tag{33}$$

is called the *offered load*, which is the average amount of work offered to the servers per unit time. $C(c,a)$ is determined by the offered load. $C(c,a)$ is called the *Erlang delay formula* or *Erlang C formula*.

Mean Performance Measures

From Equation (29), we can derive the performance measures as follows:

Mean Number of Waiting Customers.

$$L_q = \sum_{n=c}^{\infty}(n-c)P_n = \frac{a^c}{c!(1-\rho^2)}P_0 \tag{34}$$

Probability that an Arbitrary Server is Busy.

$$P_{\text{busy}} = \frac{a}{c} = \rho \tag{35}$$

Mean Number of Customers in Service.

$$L_{\text{in service}} = P_{\text{busy}}c = \lambda E(S) = \lambda/\mu = a \tag{36}$$

Mean Queue Length.

$$L = L_q + L_{\text{in service}} \tag{37}$$

Mean Waiting Time.

$$W_q = \frac{L_q}{\lambda} \text{ (Little's law)} \tag{38}$$

Mean Sojourn Time.

$$W = \frac{L}{\lambda} \quad \text{(Little's law)} \tag{39}$$

Or

$$W = W_q + E(S) = W_q + \frac{1}{\mu} \tag{40}$$

M/M/c/c QUEUEING SYSTEM

In M/M/c/c queueing systems, arriving customers cannot enter the system if all servers are busy. A typical example is the telephone exchange system. If all lines are busy, arriving calls are blocked.

The steady-state system equations can be set up as in M/M/1. The queue length probabilities in this case are as follows:

$$P_0 = \left[\sum_{k=0}^{c} \frac{a^k}{k!}\right]^{-1} \quad (a = \lambda E(S)) \tag{41}$$

$$P_n = \frac{a^n/n!}{\sum_{k=0}^{c} a^k/k!} \quad (n = 0, 1, 2, \ldots c) \tag{42}$$

Erlang Loss Formula (Erlang B formula)

The probability that an arriving customer is blocked is given by

$$B(c, a) = P_c = \frac{a^c/c!}{\sum_{k=0}^{c} a^k/k!} \tag{43}$$

B(c,a) is called the *Erlang loss formula* or *Erlang B formula*.

Some Useful Relations

(i) $$B(c, a) = \frac{aB(c-1, a)}{c + aB(c-1, a)}, \quad (B(0, a) = 1) \tag{44}$$

Note that Equation (44) can be used to compute the loss probability recursively.

(ii) $$C(c, a) = \frac{cB(c, a)}{c - a[1 - B(c, a)]} \tag{45}$$

(iii) $$B(c, a) < C(c, a). \tag{46}$$

(iv) $$C(c, a) = \frac{1}{1 + \frac{c-a}{aB(c-1, a)}}, \quad (c > a, B(0, a) = 1) \tag{47}$$

(v) $$C(c, a) = \frac{1}{1 + \frac{c-a}{a} \frac{c - 1 - aC(c-1, a)}{(c - 1 - a)C(c-1, a)}}, \ (c > a + 1) \tag{48}$$

Note that Equation (48) can be used to compute the delay probability recursively.

An Application of the Erlang B Formula

Consider a small telephone exchange system with c = 10 lines. The current QoS is 99%; i.e., 1% of arriving calls are blocked. It is expected that the number of arriving calls will double next year. To maintain the same QoS, how many more lines are needed?

(i) From $B(10, a) = P_{10} = \frac{a^{10}/10!}{\sum_{k=0}^{10} a^k/k!} = 0.01$, we derive the offered load $a = \lambda E(S) = 4.461$.

(ii) If arrival rate doubles, the offered load doubles to 8.922.

(iii) So, we need to determine a value for c such that
$$B(c, 8.922) = P_c = \frac{8.922^c/c!}{\sum_{k=0}^{c} 8.922^k/k!} \leq 0.01.$$

(iv) Because $B(16, 8.922) = 0.0104$ and $B(17, 8.922) = 0.0054$, we conclude that at least 17 lines are needed to maintain 99% QoS.

Notice that even though the arrival rate doubles, only 70% of additional lines are needed to maintain the same QoS.

M/G/1 QUEUEING SYSTEM

So far we dealt with queueing systems in which the service times follow iid exponential distribution. Now, let us consider a queueing system in which customers arrive according to a Poisson process but the service times are not necessarily exponentially distributed. If we let $X(t)$ to be the queue length at time t, the stochastic process $\{X(t), t \geq 0\}$ is no longer a Markov process because the service time does not possess the memoryless property. Thus, the system equations as in the M/M/1 queueing system are not possible for the M/G/1 queueing system, which means that we need a completely new method to analyze the M/G/1 queueing system. This goal can be accomplished by analyzing the queue length just after service completions (i.e., customer departures).

Let us define the probabilities as follows:

P_n probability that there are n customers at an arbitrary time,

$\overline{\pi}_n$ probability that an arbitrary arriving customer observes n customers,

π_n probability that an arbitrary departing customer leaves n customers behind in the system.

Then, for an M/G/1 queueing system, we have

$$\overline{\pi}_n = P_n = \pi_n \tag{49}$$

Equation (49) comes from PASTA. $\overline{\pi}_n = \pi_n$ comes from the fact that in an M/G/1 queueing system, the number of customers increases by one and decreases by one. Equation (50) implies that the mean queue length L at an arbitrary time is equal to the mean queue length L^+ just after an arbitrary departure and the mean queue length L^- just before an arbitrary arrival

$$L = L^- = L^+ \tag{50}$$

Thus, we will derive L^+ instead of L.

Let us define the random variables as follows:

N_n^+ the number of customers just after the departure of the nth customer,

A_{n+1} the number of customers that arrive during the service of the $(n+1)$st customer.

Then, we obtain

$$N^+_{n+1} = \begin{cases} N^+_n - 1 + A_{n+1}, & (N^+_n \geq 1) \\ A_{n+1}, & (N^+_n = 0) \end{cases} \tag{51}$$

We note that the service times are iid and that customer arrivals are independent of what is happening in the system. Thus, the number A_n is independent of n, and without loss of generality we can use the generic notation A.

Let us define $U(N^+_n)$ as

$$U(N^+_n) = \begin{cases} 1, (N^+_n \geq 1) \\ 0, (N^+_n = 0) \end{cases} \tag{52}$$

Using Equation (52), we can express Equation (51) as follows:

$$N^+_{n+1} = N^+_n - U(N^+_n) + A_{n+1} \tag{53}$$

In steady state, we obtain

$$\lim_{n\to\infty} E(N^+_{n+1}) = \lim_{n\to\infty} E(N^+_n) = L^+ \tag{54}$$

If we take an expectation and let $n \to \infty$ on Equation (53), we obtain

$$L^+ = L^+ - \lim_{n\to\infty} E[U(N^+_{n+1})] + \lim_{n\to\infty} E(A_{n+1}) \tag{55}$$

which implies that

$$\lim_{n\to\infty} E[U(N^+_{n+1})] = \lim_{n\to\infty} E(A_{n+1}). \tag{56}$$

$E(A_{n+1})$ can then be computed as follows:

$$\begin{aligned} E(A_{n+1}) = E(A) &= \int_0^\infty E(A|S = x)s(x)dx \\ &= \int_0^\infty \lambda x \cdot s(x)dx = \lambda E(S) = \rho \end{aligned} \tag{57}$$

where $s(x)$ is the pdf of the service time.

Squaring Equation (53), we get

$$\begin{aligned} (N^+_{n+1})^2 = {} & (N^+_n)^2 + [U(N^+_n)]^2 + (A_{n+1})^2 \\ & - 2N^+_n U(N^+_n) - 2A_{n+1}U(N^+_n) + 2N^+_n A_{n+1} \end{aligned} \tag{58}$$

The following identities can be shown to hold:

$$\lim_{n\to\infty} E[(N^+_{n+1})^2] = \lim_{n\to\infty} E[(N^+_n)^2] \tag{59}$$

$$\lim_{n\to\infty} E[(U(N^+_{n+1}))^2] = \lim_{n\to\infty} E[(U(N^+_{n+1})] = \rho \tag{60}$$

$$\lim_{n\to\infty} E[N^+_n \cdot U(N^+_n)] = \lim_{n\to\infty} E(N^+_n) = L^+ \tag{61}$$

Taking expectation, letting $n \to \infty$ on both sides of Equation (58), and using Equation (57) together with Equations (59), (60), and (61), we obtain

$$L^+ = \frac{\rho - 2\rho^2 + E(A^2)}{2(1-\rho)} \tag{62}$$

$E(A^2)$ can be obtained as follows:

$$E(A^2) = \mathrm{Var}(A) + E^2(A) = \mathrm{Var}(A) + \rho^2 \tag{63}$$

where, by using the mean and variance of the Poisson random variable,

$$\begin{aligned} \mathrm{Var}(A) &= E[\mathrm{Var}(A|S)] + \mathrm{Var}[E(A|S)] \\ &= E(\lambda S) + \mathrm{Var}(\lambda S) = \rho + \lambda^2 \mathrm{Var}(S) \end{aligned} \tag{64}$$

Using Equations (63) and (64) in Equation (62) yields

$$L^+ = L^- = L = \rho + \frac{\lambda^2 E(S^2)}{2(1-\rho)} \tag{65}$$

From Little's law, the mean sojourn time becomes

$$W = \frac{L}{\lambda} = E(S) + \frac{\lambda E(S^2)}{2(1-\rho)} \tag{66}$$

Since the sojourn time is the sum of the waiting time and the service time, it can be shown using Equation (66) that the mean waiting time becomes

$$W_q = \frac{\lambda E(S^2)}{2(1-\rho)} \tag{67}$$

Then, from Little's law,

$$L_q = \lambda W_q = \frac{\lambda^2 E(S^2)}{2(1-\rho)} = \frac{\rho^2}{2(1-\rho)} + \frac{\lambda^2 \mathrm{Var}(S)}{2(1-\rho)} \tag{68}$$

which can alternatively be derived from

$$L_q = L - \rho \tag{69}$$

Significance of Variance of the Service Time

As Equation (68), the variance of the service time affects the system performance significantly. Table 1 shows the mean queue length of five different M/G/1 systems with the same arrival rate and mean service times, but with different variances of the service times (M/D/1 is a queueing system with deterministic (constant) service time).

Notice the differences in the mean queue lengths ranging from 0.25 to ∞ (the Pareto distribution is an example of probability distributions with finite mean but with infinite variance). This distribution is possible because the mean queue length depends heavily on the variance of the service time [see Equation (68)]. Note that the mean queue length of system 2 is 33 times larger than that of the M/D/1 queue. The difference is more spectacular in system 3.

Table 1. Mean queue lengths for different variances of the service times

	M/D/1	M/M/1	System 1	System 2	System 3
λ	2	2	2	2	2
$E(S)$	0.25	0.25	0.25	0.25	0.25
Var(S)	0	1/16	1	2	∞
L_q	0.25	0.5	4.25	8.25	∞

But, it should be noted that the server is idle 50% of its time in all five systems since we have the same $\rho = \lambda E(S) = 0.5$. It is hard to imagine a queueing system in which an average of 10,000 customers are waiting, but the server is idle 50% of its time.

It is important to appreciate, from this example, the effect of the variance on the system performances. In general, if one wants to reduce the mean queue length and the mean waiting time, the variance is the first thing to check.

For an analysis of the variants of the M/G/1 queueing system, including vacation systems and server controls, we refer the readers to Ref. (9).

QUEUEING NETWORKS

A queueing network is composed of several queueing systems connected to each other. Many computer systems, communication systems, and production systems can be modeled by a queueing network.

Classification by Network Structure

Each queueing system that comprises a queueing network is called a node. Multiple servers can exist in a node.

A queueing network can be classified as an open network, a closed network, or a mixed network. One might add a tandem queue as a special case of an open network.

OQN (Open Queueing Network). In an open queueing network, customers can enter the system and leave the system. An arrival to a node can be from either outside (external arrival) or inside (internal arrival). Figure 6 shows an OQN in which the finished customers at node-1 leave the system with probability p_1 and join node-i with probability $p_i, (i = 2, \ldots m)$.

CQN (Closed Queueing Network). In a CQN, a fixed number of customers circulate (existing customers cannot leave the system, and new customers cannot enter the system). Figure 7 shows a CQN model in which there are three nodes and N customers are circulating in the system.

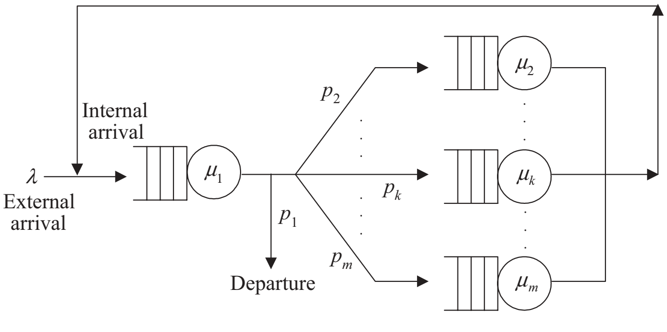


Figure 6. An open queueing network.

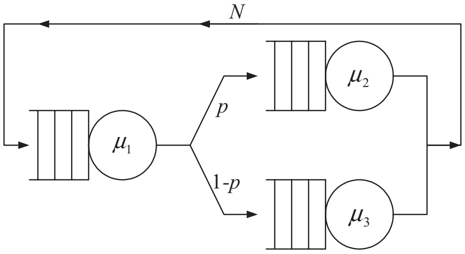


Figure 7. A closed queueing network.

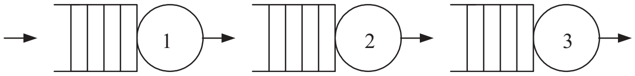


Figure 8. A tandem queue.

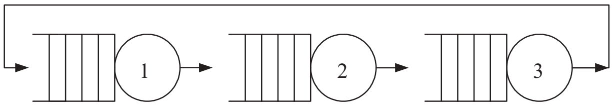


Figure 9. A cyclic queue.

Mixed-Type Queueing Network. This network is a mixture of OQN and CQN. Some classes of customers are free to leave or enter the system, where as others are not allowed to do so.

Tandem Queue (Series Queue). In a tandem queue, multiple nodes are connected in a series (Fig. 8). If it is closed, it is called a cyclic queue (Fig. 9).

MARKOVIAN OPEN QUEUEING NETWORKS (JACKSON NETWORKS)

A Jackson network is an open Markovian queueing network in which external customers arrive according to Poisson processes and service times follow exponential distributions. This network was first studied by Jackson (13,14) and is named after him.

The Jackson network is an open queueing network with the following specifications (13). Let K be the number of nodes.

(i) External arrivals to node-i occur according to a Poisson process with rate λ_i. External arrival processes are independent.

(ii) The service time at node-i is load-dependent exponential: When n customers are at the node, the service rate is $\mu_i(n)$. The case of identical multiple servers can be viewed as a single-server node with load-dependent service rate.

(iii) The customer whose service is completed at node-i goes to node-j with a routing probability of γ_{ij} or departs the system with a probability of $\gamma_{i0} = 1 - \sum_{j=1}^{K} \gamma_{ij}$.

(iv) The buffer size at each node is infinity.

System Equations

Consider a simple case of $K = 2$ nodes with a single server at each node (Fig. 10). More complex systems can be analyzed in an analogous way.

Let us define $P(n_1, n_2)$ as the probability that there are n_1 customers at node-1 and n_2 customers at node-2. Then, the steady-state system equations can be written as follows:

$$\begin{aligned}(\mu_1 + \mu_2 + \lambda_1 + \lambda_2)P(n_1, n_2) \\ = \lambda_1 P(n_1 - 1, n_2) + \lambda_2 P(n_1, n_2 - 1) \\ + \mu_1 \gamma_{12} P(n_1 + 1, n_2 - 1) + \mu_2 \gamma_{21} P(n_1 - 1, n_2 + 1) \\ + \mu_1 \gamma_{10} P(n_1 + 1, n_2) + \mu_2 \gamma_{20} P(n_1, n_2 + 1)\end{aligned} \tag{70}$$

The left-hand side of the above equation is the rate out of state (n_1, n_2), and the right-hand side is the rate into state (n_1, n_2). In Ref. (13), Jackson showed that the following "product-form" solution satisfies Equation (70).

$$P(n_1, n_2) = (1 - \rho_1)\rho_1^{n_1} \cdot (1 - \rho_2)\rho_2^{n_2} \tag{71}$$

where

$$\rho_i = \frac{\Lambda_i}{\mu_i} \tag{72}$$

In Equation (72), Λ_1 is the aggregate arrival rate into note-1 and is given by

$$\Lambda_1 = \lambda_1 + \sum_{j=1}^{2} \Lambda_j \gamma_{j1} \tag{73}$$

Note that λ_1 is the external input rate into node-1 and $\sum_{j=1}^{2} \Lambda_j \gamma_{j1}$ is the internal input rate into node-1. Likewise, we get

$$\Lambda_2 = \lambda_2 + \sum_{j=1}^{2} \Lambda_j \gamma_{j2} \tag{74}$$

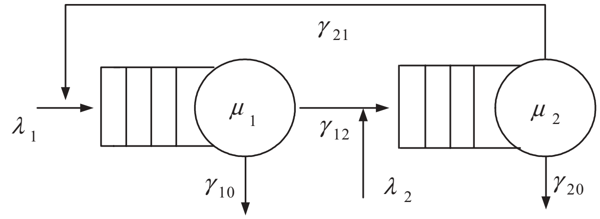


Figure 10. A Jackson network with two nodes (each node has a single server).

From Equation (71), we see that the marginal queue length distribution at node-1 is $P(n_1) = (1 - \rho_1)\rho_1^{n_1}$ and $P(n_2) = (1 - \rho_2)\rho_2^{n_2}$ at node-2. Observe that each node behaves as if it were an M/M/1 queue with arrival rate Λ_i and service rate μ_i. But these queues are not actually M/M/1 queues because the aggregate arrival process (i.e., superposition of the internal and external arrival processes) at each node is not a Poisson process. The aggregate arrival process is not even a renewal process. In general, the aggregate arrival process, into a node at which customers visit more than once is not a Poisson process.

If there exists more than one identical server at a node, we can use the M/M/c results from Equations (41) and (42). That is, the probability $P(n_1, n_2, \ldots n_K)$ that there are n_1 customers at node-1, n_2 customers at node-2,... and n_K customers at node-K is given by the product-form probability as follows:

$$P(n_1, n_2, \ldots n_K) = \prod_{j=1}^{K} P_j(n_j) \tag{75}$$

where

$$P_i(k) = \begin{cases} \dfrac{\Lambda_i^k}{k! \mu_i^k} P_i(0), (1 \le k \le c_i - 1) \\ \dfrac{\Lambda_i^k}{c_i^{k-c_i} c_i! \mu_i^k} P_i(0), (n \ge c_i) \end{cases} \tag{76}$$

and

$$P_i(0) = \left[\sum_{k=0}^{c_i - 1} \frac{(\Lambda_i / \mu_i)^k}{k!} + \frac{(\Lambda_i / \mu_i)^{c_i}}{c_i! (1 - \rho_i)} \right]^{-1}, (\rho_i = \frac{\Lambda_i}{c_i \mu_i}) \tag{77}$$

Λ_i can be computed using

$$\Lambda_i = \lambda_i + \sum_{j=1}^{K} \Lambda_j \gamma_{ji} \tag{78}$$

Equation (78) is called the *traffic equation*.

Mean Performance Measures

From Equations (75), (76) and (77), we can derive the following performance measures.

Mean Number of Waiting Customers at Node-i.

$$L_{q_i} = \frac{(\Lambda_i / \mu_i)^{c_i}}{c_i! (1 - \rho_i^2)} P_i(0) \tag{79}$$

Mean Queue Length at Node-i.

$$L_i = L_{q_i} + \frac{\Lambda_i}{\mu_i} \tag{80}$$

One-Time Mean Waiting Time at Node-i.

$$W_{q_i} = \frac{L_{q_i}}{\Lambda_i} \text{ (Little's law)} \tag{81}$$

One-Time Mean Sojourn Time at Node-i.

$$W_i = \frac{L_i}{\Lambda_i} \text{ (Little's law)} \tag{82}$$

Total Mean Time a Customer Spends in the Network. Let W_T be the total mean time a customer spends in the network. Let R_i be the mean time an arriving customer (internal or external) to node-i spends in the system until it departs the network. The customer has to spend W_i time at node-i first. Then, it goes to node-j with probability γ_{ij} and spend R_j time starting all over again from node-j until it departs the network. Thus, we obtain

$$R_i = W_i + \sum_{j=1}^{K} R_j \gamma_{ji} \tag{83}$$

An arbitrary external customer enters the network through node-i with probability

$$\alpha_i = \frac{\lambda_i}{\sum_{j=1}^{K} \lambda_j} \tag{84}$$

Thus, we get

$$W_T = \sum_{j=1}^{K} \alpha_i R_i \tag{85}$$

Total Mean Number of Customers in the Network at an Arbitrary Time. From Little's law, the total mean number L_T of the customers in the network can be obtained using

$$L_T = \left(\sum_{j=i}^{K} \lambda_j \right) W_T \tag{86}$$

which is equal to

$$L_T = \sum_{j=1}^{K} L_i \tag{87}$$

Total Mean Service Time Received. Let v_i be the mean number of times an external customer visits node-i before it departs the network. Then, v_i can be computed using

$$v_i = \alpha_i + \sum_{j=1}^{K} v_j \gamma_{ji} \tag{88}$$

The total mean service time received at node-i is

$$D_i = \frac{v_i}{\mu_i} \tag{89}$$

The total mean service time received by a customer in the network is

$$D_T = \sum_{j=1}^{K} D_j \tag{90}$$

MARKOVIAN CQN (GORDON-NEWELL NETWORK)

The Gordon-Newell network is a Markovian CQN (15). In a CQN, a fixed number of customers circulate the system. We use an example for illustration; see Fig. 11.

Suppose a fixed number of N customers circulate in the network of Fig. 11. Let $P(k, N-k)$ be the probability that k customers at node-1 exist and $N-k$ customers exist at node-2. In steady-state, we obtain the following set of system equations,

$$\begin{aligned}(\mu_1 + \mu_2)P(k, N-k) = \mu_1 P(k+1, N-k-1) + \mu_2 P \\ \times (k-1, N-k+1), \\ \times (k \neq 0, k \neq N)\end{aligned} \tag{91}$$

$$\mu_1 P(1, N-1) = \mu_2 P(0, N) \text{ and} \tag{92}$$

$$\mu_1 P(N, 0) = \mu_2 P(N-1, 1) \tag{93}$$

In steady-state, if a customers enter node-1 per unit time, the same number of customers enter node-2 per unit time. Let us define $\rho_1 = \frac{a}{\mu_1}$ and $\rho_2 = \frac{a}{\mu_2}$, where a is an arbitrary constant. Thus, ρ_1 and ρ_2 are not the probability that the servers are busy. The reason why ρ_1 and ρ_2 are not unique will be discussed subsequently. If we use $\mu_1 = \frac{a}{\rho_1}$ and $\mu_2 = \frac{a}{\rho_2}$ in the above equations, we get the product-form probability as follows:

$$P(k, N-k) = \frac{1}{C(2, N)} \rho_1^k \rho_2^{(N-k)} \tag{94}$$

Note that $C(2, N)$ is a normalization constant that is determined using

$$\sum_{k=0}^{N} P(k, N-k) = 1 \tag{95}$$

Also, note that $C(2, N)$ changes if a changes (and thus ρ_1 and ρ_2 change). But the effect of a is absorbed into $C(2, N)$ and therefore $P(k, N-k)$ does not change. For convenience, if we let $a = \mu_1$, we get $\rho_1 = 1, \rho_2 = \mu_1/\mu_2$, and

$$P(k, N-k) = \frac{1}{C(2, N)} \rho_2^{(N-k)} \tag{96}$$

Using $\sum_{k=0}^{N} P(k, N-k) = 1$, we get

$$C(2, N) = \begin{cases} \dfrac{1 - \rho_2^{N+1}}{1 - \rho_2}, (\mu_1 \neq \mu_2) \\ N + 1, (\mu_1 = \mu_2) \end{cases} \tag{97}$$

where $\rho_2 = \mu_1/\mu_2$.

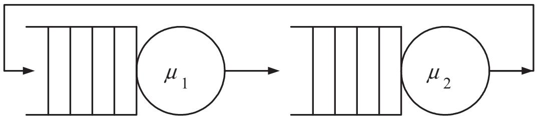


Figure 11. A closed queueing network with feedback.

In general, because the entrance and exit to and from the network is impossible, the traffic equation becomes

$$\Lambda_i = \sum_{j=1}^{K} \Lambda_j \gamma_{ji} \tag{98}$$

This equation does not have a unique solution because

$$\sum_{j=1}^{K} \gamma_{ij} = 1 \tag{99}$$

Thus, $\{\Lambda_i, (i = 1, 2, \ldots, K)\}$ obtained from the traffic equation are not unique. If we let $\{e_i, (i = 1, 2, \ldots, K)\}$ be a solution to Equation (98), then e_i is proportional to Λ_i. One simple way is to let $e_1 = 1$, which means that a customer visits note-1 once while he visits node-i for e_i times.

For the general CQN with K nodes and N circulating customers, according to Gordon and Newell (15)

$$P(n_1, n_2, \ldots n_K) = \frac{1}{C(K,N)} \prod_{i=1}^{K} f_i(n_i) \tag{100}$$

where

$$f_i(n_i) = \frac{e_i^{n_i}}{\prod_{j=1}^{n_i} \mu_i(j)} \tag{101}$$

$$f_i(0) = 0 \tag{102}$$

$C(K,N)$ is a normalization constant that is be determined using

$$C(K,N) = \sum_{\text{for all system states}} \prod_{i=1}^{K} f_i(n_i) \tag{103}$$

As in the preceding example, the biggest problem with a CQN is how to determine the normalization constant. In the preceding example, we had only two nodes and it was not a problem. But we have $\binom{N+K-1}{K-1}$ different system states in general. If we have $K = 8$ nodes and $N = 20$ customers circulating in the network, the number of different system states becomes $\binom{N+K-1}{K-1} = 888,030$. Computing that many probabilities to determine the normalization constant $C(K,N)$ is almost impossible even for the closed network of moderate size. Buzen (16) presents an efficient algorithm to compute the normalization constant.

CURRENT RESEARCH ISSUES

The research area of queueing theory is mature by now. However, several research issues are still being investigated actively. One such area is that of self-similar traffic models. Traditionally, the traffic models assumed in queueing theory were assumed to be Markovian, which was true of telephone network traffic. Subsequently, as queueing theory began seeing applications in other disciplines, other models of traffic were added. In the early 1990s research revealed that the traffic in computer networks did not follow the traffic models assumed in the literature. Rather, the traffic model exhibited a self-similarity behavior where the traffic pattern observed at different time scales had the same bursty pattern. This observation led to a spur in the research activity on the impact of this traffic model in the design of computer systems. This research area continues to be active today. For an overview of traffic patterns and optimal scheduling, we refer the reader to Refs. (17), (18), and (21).

Another rich research area is the discrete-time queueing system. In the discrete-time queueing systems, the time is expressed in multiples of slots and services can start only at slot boundaries. Because information in communication systems is transmitted by means of discrete units of cells, discrete-time models are believed to be more suitable to represent the modern telecommunication systems. Readers are referred to Refs. (19) and (20).

BIBLIOGRAPHY

1. A.K. Erlang, The theory of probabilities and telephone conversations, *Nyt Tidsskrift Matematik, B.* **20**: 33–39,1909. Reproduced in Brockmeyer et al. (4), pp. 131–137.
2. A.K. Erlang, Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges, *Electroteknikeren* **13**: 5–13, 1917. Reproduced in Brockmeyer et al. (4), pp. 138–155.
3. A.K. Erlang, Telephone waiting times, *Matematisk Tidsskrift, B*, **31**: 1920. Reproduced in Brockmeyer et al. (4), pp. 156–171.
4. E. Brockmeyer, H.L., Halstrom and A., Jensen, *The Life and Works of A.K. Erlang*, Copenhagan: Copenhagen Telephone Co., 1948.
5. D. Berksekas and R. Gallager, *Data Networks*, Englewood Cliffs, NJ: Prentice Hall, 1992.
6. D.G. Kendall, Stochastic processes occurring in the theory of queues and their analysis by the method of imbedded Markov chains, *Ann. Mathemat. Stat.* **24**: 338–354, 1953.
7. J.D.C. Little, A proof for the queueing formula: $L = \lambda W$, *Oper. Res.*, **9** (3): 383–387, 1961.
8. R.W. Wolff, Poisson arrivals see time averages, *Oper. Res.*, **30** (2): 223–231, 1982.
9. H. Takagi, *Queueing Analysis, Vol I: Vacation and Priority Systems, Part 1*, Amsterdam: North-Holland, 1991.
10. W.C. Giffin, *QUEUING: Basic Theory and Applications*, Columbus, OH: Grid Inc., 1978.
11. D. Gross, and C.M. Harris, *Fundamentals of Queueing Theory*, 2nd ed., New York: John Wiley & Sons, 1985.
12. R.B. Cooper, *Introduction to Queueing Theory*, 2nd ed., New York: Elsevier North Holland, 1981.
13. J.R. Jackson, Networks of waiting lines, *Oper. Res.*, **5**: 518–527, 1957.
14. J.R. Jackson, Jobshop-like queueing systems, *Manage. Sci.*, **10** (1): 131–142, 1963.

15. W.J. Gordon, and G.F., Newell, Closed queueing systems with exponential servers, *Oper. Res.*, **15**: 254–265, 1967.

16. J.P. Buzen, Computational algorithms for closed queueing networks with exponential servers, *Comm. of ACM*, **16**: 527–531, 1973.

17. T.G. Robertazzi, *Computer Networks And Systems: Queueing Theory and Performance Evaluation*, 3rd ed., New York: Springer, 2000.

18. W. Stallings, *High Speed Networks: TCP/IP and ATM Design Principles*, Englewood Cliffs, NJ: Prentice Hall, 1998.

19. H. Bruneel, and B.G. Kim, *Discrete-Time Models for Communication Systems including ATM*, Dordrecht, the Netherlands: Kluwer Academic Publishers, 1993.

20. H. Takagi, *Queueing Analysis, Vol 3: Discrete-Time Systems*, Amsterdam: North-Holland, 1993.

21. M. Harchol-Balter, *http://www.cs.cmu.edu/~harchol/homepage.html.*

HO WOO LEE
Sungkyunkwan University
Suwan, Korea
SANTOSH KUMAR
University of Memphis
Memphis, Tennessee

SERVICE-ORIENTED ARCHITECTURE AND WEB SERVICES

INTRODUCTION

Service-oriented architecture (SOA) (1,2) is a software paradigm that enables large applications to be created in an ad hoc, loosely coupled manner from smaller modules called services. SOA defines a methodology for the reuse and interoperability of software components and business processes within and between enterprises over the Internet and, thus, promises to achieve flexibility, agility, and cost savings for enterprises.

In the past, enterprises integrated their silo systems using a point-to-point or enterprise application integration (EAI) approach for each project. That approach resulted in systems that are complex, difficult-to-modify, and expensive-to-maintain. Today, an enterprise must be able to do business with many other enterprises, and it must be able to respond rapidly to changes and challenges, such as competitive pricing and offshoring to compete in the global economy.

SOA enables enterprises to develop and deploy applications more rapidly. It supports modularity of design, facilitates software reuse, promotes standardization, and supports interoperability across diverse hardware platforms, operating systems, and programming languages. Software reuse spreads the costs of software development over many customers. Interoperability allows software services to be accessed without having to know their underlying implementations or the computing platforms on which they run.

SOA offers the promise of reduced time and costs in software development, and reduced application and infrastructure complexity. It aims to promote business between enterprises, increase profits, improve product quality, increase customer satisfaction and achieve operation agility.

Web services (3) are the most common implementation of the service-oriented architecture. However, some SOA implementations do not use Web services but provide similar benefits.

SERVICE-ORIENTED ARCHITECTURE

SOA (1,2) is an architecture definition and process that enables large applications to be created in an ad hoc, loosely coupled manner from modular services. A service is a unit of work performed by a service provider to achieve desired end results for a service consumer. SOA has the following requirements:

- Interoperability of services regardless of the hardware platforms and operating systems on which they are deployed or the programming languages in which they are written.
- Description of services in a clear and unambiguous manner that allows a potential consumer to find and use a service offered by a provider.
- Access to services by means of a standard communication protocol and a common format for messages and the data that they contain, so that a consumer can access and use a service offered by a provider.

SOA consists of multiple layers and components, as shown in Fig. 1. The top layer provides the Services Interface, which allows the clients to invoke the services, and the bottom layer contains the Application Services. The middle layer contains the Services Coordinator, which controls the flow of messages from the Services Interface to the Application Services. SOA may also contain application-neutral Service Management components and Quality of Service components, as shown in Fig. 1.

The essence of SOA is independent services that can be called in a standard way to perform their tasks, without a service needing to know about a client and without a client needing to know how a service actually performs its tasks. Each service interaction is self-contained, and different service interactions are coupled loosely, so that each service interaction is independent of other service interactions. SOA supports communication between services using standard protocols and enables one application to perform a service on behalf of another service.

In SOA, a service represents a larger unit of functionality than a traditional function or class. Unlike a function, the software providing a service must not interact internally with the software of other services. In SOA, services are coupled loosely, in contrast to the functions that a linker binds together to form an executable or a dynamically linked library.

Underlying SOA is metadata that are used to describe not only the characteristics of the services but also the data that the services exchange. The metadata must be in a form that system designers can understand and in a form that software systems can use dynamically and automatically.

In SOA, services work together based on a formal definition or contract (e.g., WSDL description) that is independent of the underlying hardware platform and operating system on which the services are deployed and the programming language in which they are written.

As shown in Fig. 2, a service can assume one or more of three roles: service provider, service broker, or service requester.

A service provider creates a service and defines an interface for invoking that service. The service provider also creates a service description for the service and makes the service available to potential consumers through a service broker by publishing the service description in a service registry.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

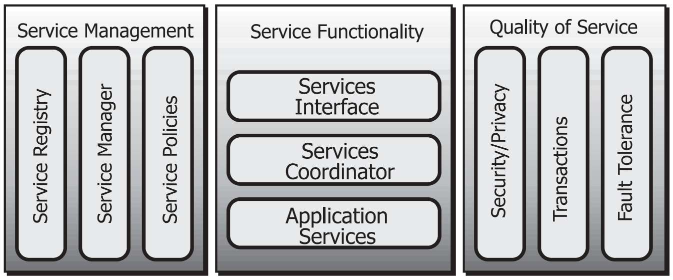


Figure 1. Service-Oriented Architecture. SOA consists of multiple layers and components and promotes modularity of design and software reuse.

A service broker uses the information in the service description to catalog the service and to search for the service when it receives a request for information about the service. The service broker provides a service, the address, and the interface are known a priori to the service requester.

A service requester that is trying to find a service queries the service broker. The service broker replies with a service description that indicates where to find the service and how to invoke it. The service requester can then bind to the service provider by invoking the service.

The basic service concept is extended by orchestration of fine-grained services into more coarse-grained business services, which in turn can be incorporated into business processes and workflows.

SOA can be implemented using a variety of technologies, including Web Services, RPC, Java RMI, CORBA, and DCOM. Typically, the services run in Java Enterprise Edition or .NET environments that manage memory allocation and deallocation, allow ad hoc and late bindings and perform type checking. Services written in Java for Java Enterprise Edition environments and services written in C# for .NET environments can be used by a client, and can use each other. Legacy systems, written in COBOL, can be wrapped and presented as services.

WEB SERVICES

SOA is typically implemented using Web services (3), although that is not required. The World Wide Web Consortium (W3C) defines a Web service as:

> A software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web Service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols.

This definition states explicitly that a Web service is based on XML. It emphasizes that a Web service must be capable of being defined, described, and discovered, to enable creation of client software that binds to, and interacts with, the Web service using the defined interfaces.

XML

The eXtensible Markup Language (XML) (4) provides a common syntax for Web services documents, so that the information in those documents is self-describing. It defines the rules that a document must follow to be well-formed. XML aims to achieve interoperability, portability, and automatic processing with data independence for different programming languages, middleware systems, and database management systems.

Like the HyperText Markup Language (HTML), XML has elements, attributes, values, and tags. XML elements and attributes provide type and structure information for the data values. XML element tags describe the data values that they enclose. For example:

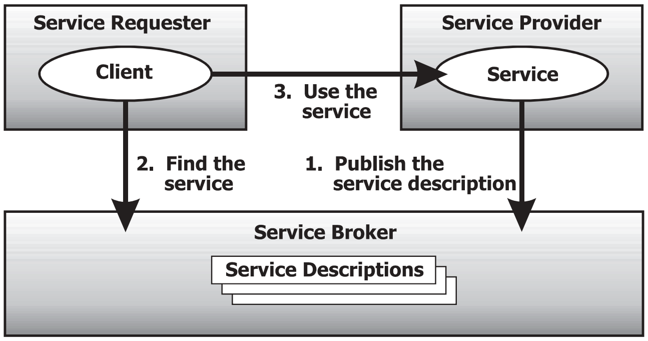


Figure 2. Service-oriented architecture. The service provider creates a service and publishes a service description at the service broker. The service consumer finds the service description at the service broker and then uses the service provided by the service provider.

```
 <Service provider>
    <Name>Enterprise</Name>
    <PhysicalAddress>3210 State Street, Santa Barbara,
     CA 93101<\PhysicalAddress>
    <PhoneNumber>805-569-6222<\PhoneNumber>
     <URL>www.enterprise.com<\URL>
<\ServiceProvider>
```

XML provides a standard way to define the structure of documents so that they are suitable for automatic processing. An XML parser can determine that a document contains a certain element and can extract the content associated with that element.

XML schemas and document type definitions are used to specify document types and to state that a document is of a certain document type. An XML document verifier can be used to check whether the structure and content of a document are consistent with the prescribed type. Currently, XML schemas and document type definitions do not provide semantic information about the document or the elements contained within the document.

More precise tagging instructions have been defined for various vertical business sectors. In particular, the Organization for the Advancement of Structured Information Standards (OASIS) and the United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT) have developed the electronic business XML (ebXML) standard (5) as a successor to the Electronic Data Interchange (EDI) standard. The ebXML standard defines an architecture and a specification that are designed to automate business process interactions among trading partners.

Web services are classified into two categories: Representational (REST) Web services or Simple Object Access Protocol (SOAP) Web services. Both kinds of Web services use XML for formatting the data so that they are self-describing. These two categories of Web services are discussed below.

REST Web Services

REST Web services (6,7) are based on the concept of a resource, which has a Uniform Resource Identifier (URI). A resource may have zero or more representations. If no representations for the resource exist, the resource is said not to exist. A REST Web service has the following additional requirements:

- Interfaces are limited to HTTP, namely:
  - GET is used for obtaining a representation of a resource. A consumer uses it to retrieve a representation from a URI.
  - DELETE is used for removing a representation of a resource.
  - POST is used for updating or creating a representation of a resource.
  - PUT is used for creating a representation of a resource.
- Messages are in XML, which are confined by a schema written in a schema language such as RELAX NG (8) or XML Schema (9).
- Messages can be encoded with URL encoding.
- Services and service providers must be resources, whereas a consumer may be a resource but is not required to be a resource.

REST Web services require little infrastructure support other than standard HTTP and XML. They are simple and effective, because HTTP is available widely and works for most applications.

As interest in REST Web services has grown, so has the scope and size of business applications that it supports. The input parameters to the HTTP methods have grown in size and number. Structured response values have also grown in complexity, ranging from customer XML namespaces to JavaScript Object Notation (JSON) (10). These trends have made descriptors a natural addition to REST Web Services.

The newly proposed Web Application Description Language (WADL) (11) provides standard descriptors for REST Web services just as the Web Services Description Language (WSDL) provides standard descriptors for SOAP Web services (see the section on WSDL below). A WADL descriptor not only describes the service, including the grammars, resources, and methods of the service, but also aids in the creation of stubs that are used to build service clients. Currently, tools for WADL that create stubs from descriptors are available only for Java environments.

IBM has initiated a project, named Project Zero (12), for REST Web services that aims to extend the Service Oriented Architecture for enterprises to the Web Oriented Architecture. The Zero platform is a Java runtime environment that is optimized to run script and REST Web services. It uses PHP and Groovy for producing REST Web Services and Ajax for building interactive clients. It allows enterprise services to be transformed and exposed as RSS/Atom feeds, which are easy to consume using feed readers.

REST Web services are growing in popularity, perhaps because they are lighter weight than SOAP Web services and because they can achieve a higher level of interoperability more easily.

SOAP Web Services

Today, most people think of Web services as SOAP Web services, rather than as REST Web services. (See the section on SOAP below.) SOAP Web services are based on the following core specifications and standards:

- SOAP (13), which is an XML-based protocol for accessing a Web service using HTTP or SMTP.
- WSDL (14), which is an XML-based language for describing Web services and the means to access them.
- Universal Description Discovery and Integration (UDDI) (15,16), which is used by the service broker (registry).

Some industry organizations, such as the Web Services Interoperability (WS-I) organization (17), mandate the use of both SOAP and WSDL in their definition of a Web service.

SOAP Web services are classified into two categories: SOAP Document-Centric Web services and SOAP Remote Procedure Call (RPC) Web services. These two categories of SOAP Web services are discussed below.

SOAP Document-Centric Web Services

In SOAP document-centric Web services, the service requester and service provider exchange XML documents. They must agree on the structures of the documents to be exchanged. The documents are transported between them in SOAP messages.

As an example of a SOAP document-centric Web service, consider a service for reserving a rental car from a car rental agency.

The service requester creates a rental car reservation document, which contains the kind of car requested, city, and date. The service requester then sends the rental car reservation document to the car rental agency in a SOAP message. The body of the SOAP message contains the rental car reservation document, and the header includes information that identifies the service requester to the car rental agency, as shown at the left in Fig. 3.

The service provider (the car rental agency) creates a reservation confirmation document, which contains the cost of the car rental and the reservation Id for the service requester, and then sends it to the service requester in a SOAP message. The body of the SOAP message contains the reservation confirmation document, and the header includes information that identifies the car rental agency, as shown at the right in Fig. 3.

SOAP RPC Web Services

In SOAP RPC Web services, the service requester encapsulates a remote procedure Call (RPC) in a SOAP message and sends the request message to the service provider. The body of the SOAP request message contains the procedure call, including the name of the procedure being invoked and the input parameters. The service provider processes the RPC and returns the results and output parameters in a SOAP response message to the service requester. The body of the SOAP response message contains the result and the output parameters of the RPC. The service requester and service provider must agree on the RPC signature rather than on the document structures.

As an example of a SOAP RPC Web service, again consider a service for reserving a rental car from a car rental agency.

The service requester creates a SOAP message and sends it to the car rental agency as a request. The body of the SOAP message contains the procedure name reserveRentalCar, as well as the city, date, and kind of car parameters, as shown at the left in Fig. 4.

The service provider (the car rental agency) processes the RPC and returns a SOAP message to the service requester as a response. The body of the SOAP message contains the reservation Id and the cost of the rental car as the return values of the RPC, as shown at the right in Fig. 4.

Any additional properties associated with the RPC are included in the header of the SOAP message. For example, for a transactional RPC, the request header includes the transaction context, which enables the receiver to process the request as a transaction.

The SOAP RPC Web service tunnels application-specific RPC interfaces through the generic SOAP interface. Effectively, it prescribes both system behavior and application semantics, and is imperative rather than descriptive, which is contrary to the spirit of SOA. Procedural interfaces require more complete and rigorous specification, and greater prior agreement, than do document interfaces. Consequently, some people consider applications created with SOAP RPC Web services not as interoperable as SOAP document-centric Web services. Both the WS-I Basic Profile and the SOAP 1.2 Specification consider support for SOAP RPC optional.

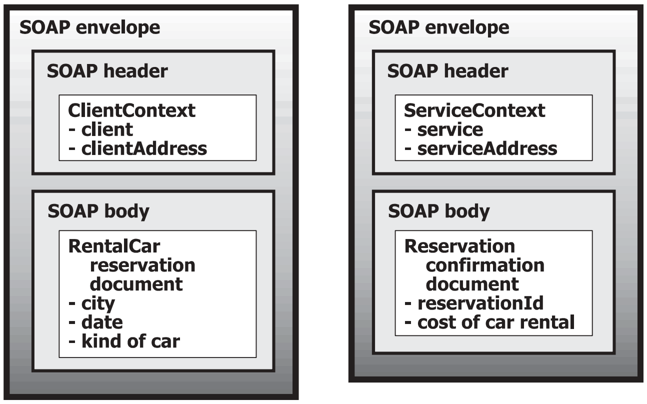


Figure 3. SOAP document-centric Web service. The SOAP message communicated by the service requester contains a reservation request document, and the SOAP message communicated by the service provider contains a reservation confirmation document.

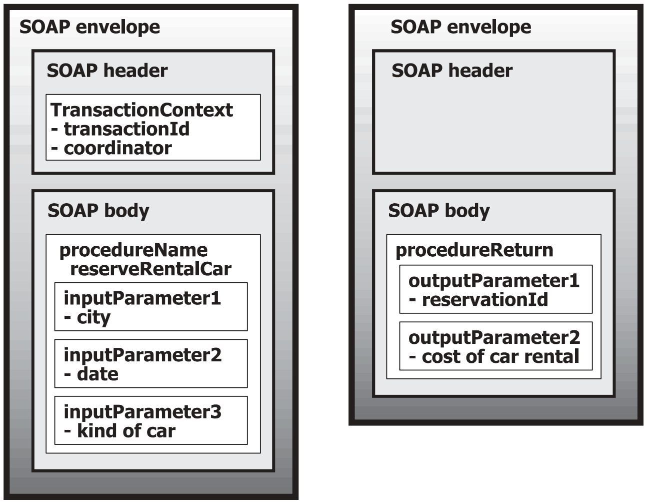


Figure 4. SOAP RPC Web service. The SOAP message from the service requester contains a RPC, and the SOAP message from the service provider contains the results from execution of the RPC.

SOAP

SOAP (13) defines how to organize information and messages using XML, so that the information can be exchanged among service requesters, service providers, and service brokers.

SOAP is an application-layer protocol that operates on top of other protocols, most commonly the HyperText Transfer Protocol (HTTP) but also the Simple Mail Transfer Protocol (SMTP).

SOAP supports applications that interact via one-way asynchronous messages as for SOAP document-centric Web services and also applications that interact via two-way synchronous request-response messages as for SOAP RPC Web services.

SOAP messages are used as envelopes that enclose the data that the application wants to send. An envelope consists of two parts: a header and a body. The header is optional, and the body is mandatory. Both the header and the body can have multiple subparts in the form of header blocks and body blocks, as shown in Figs. 3 and 4 for SOAP Web services.

A SOAP message has a sender, a receiver, and an arbitrary number of intermediaries (nodes) that process the message and route it to the receiver. The information that the sender wants to transmit to the receiver is in the message body. Additional information needed for intermediate processing or for value-added services (such as security or transactions) is included in the message header.

SOAP incurs processing overhead for parsing and serializing the XML messages, and communication overhead for the extra XML tags, but it promotes interoperability among the interacting service requesters, service providers, and service brokers.

WSDL

WSDL (14) provides descriptors for SOAP Web Services in a standard way. A WSDL descriptor specifies how to interact with the Web service, the data that are to be sent, the operations that are involved, the protocol that is to be used to invoke the service, and the data that can be expected in return. A WSDL descriptor can be used as input to a tool that generates stubs and can be used to capture information that allows reasoning about semantics. Thus, it is similar in purpose to the Interface Definition Language (IDL) of other middleware platforms, such as the Common Object Request Broker Architecture (CORBA).

A WSDL descriptor consists of an abstract part and a concrete part, as shown in Fig. 5. The abstract part is analogous to conventional IDL and uses type, message, operation, and port type constructs. These four constructs are called abstract because they do not have a concrete binding, a specific encoding, or a definition of a service that implements them.

Types allow the exchanged data to be interpreted correctly at both endpoints of the communication. By default, WSDL uses the same basic and structured types as XML schemas.

Messages are typed documents that are divided into parts, each of which has a name and a type. For example, a message for a procedure call with integer and string parameters has a part that contains the integer and a part that contains the string.

Operations are classified as one-way, notification, request-response, and solicit-response. One-way and notification operations involve a single message, whereas request-response and solicit-response operations involve two messages. One-way and request-response operations are initiated by the client, whereas notification and solicit-response operations are initiated by the service.

A port type in WSDL is analogous to an interface in IDL. A port type consists of a set of related operations.

The concrete part of a WSDL description defines an instance of a service and uses interface bindings, ports, and service constructs.

An interface binding specifies the message encoding and protocol bindings for all operations and messages defined in a port type. In particular, it specifies the encoding rules to be used in serializing the parts of a message into XML. It can also be used to specify that an operation is either SOAP document-centric or SOAP RPC style, or that the messages of the operation must be communicated using SOAP with HTTP or SMTP bindings.

A port combines the interface binding information with the network address, specified as URIs, where the implementation of the port type can be accessed.

A service is a logical grouping of ports, which are typically related ports at the same address.

UDDI

The UDDI specification (15,16) defines a mechanism for clients to find Web services dynamically. UDDI is based on the notion of a business registry (essentially, a naming or directory service). UDDI defines data structures and application programming interfaces for publishing service descriptions in the business registry and for querying the registry to look for published descriptions.

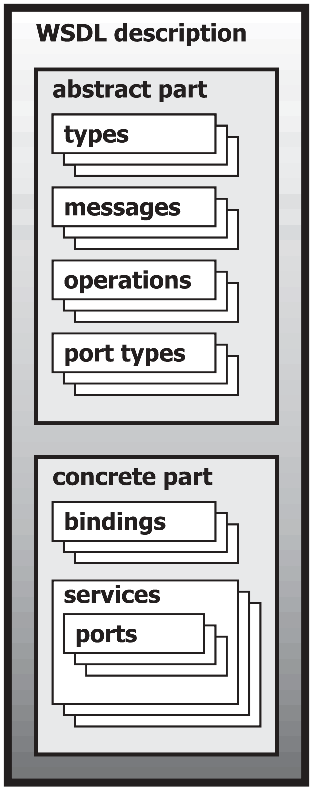


Figure 5. Web Service Description Language. The WSDL descriptor not only describes a service but also allows client stubs to be created from the descriptor.

UDDI registries have three types of users to which they expose their application programming interfaces: service providers that want to publish a service (and its usage interfaces), service requesters that want to obtain services of a certain kind and bind programmatically to those services, and other registries (service brokers) that need to exchange information. Interaction with a UDDI registry takes place as a sequence of exchanges of XML documents, typically using SOAP.

UDDI supports application developers in finding information about Web services, so that they know how to write clients that can interact with those services. It also enables dynamic binding by allowing clients to query the registry and obtain references to services in which they are interested. In addition, it supports the idea of a Universal Business Registry (UBR), where anyone can publish service descriptions and can query the registry for services of interest.

The information within a UDDI registry can be categorized as follows:

- Listings of organizations, contact information, and services that those organizations provide.
- Classifications of companies and Web services according to taxonomies that are either standardized or user-defined.
- Descriptions of how to invoke Web services, by means of pointers to service description documents, stored outside the registry, for example, at a service provider's site.

As yet, few applications use a UDDI registry to discover a Web service and then invoke that Web service dynamically using its WSDL interface. Rather, in current practice, client applications are designed explicitly to invoke Web services with known WSDL interfaces and Uniform Resource Identifiers (URIs). Currently, some Web services can be found on publicly available Web sites, such as the XMethods Website (18), but it is also possible to find Web Services using Google or Amazon as described in Ref. 19.

In addition to the core specifications (XML, SOAP, WSDL, UDDI), many other SOAP Web Services specifications, exist which are general in referred to as WS-*. These specifications include:

- WS-AtomicTransaction, WS-Coordination, WS-BusinessActivity–Specifications that define mechanisms and interfaces for transactions in a distributed environment (20).
- WS-ReliableMessaging–A protocol, issued by IBM, BEA, Microsoft, TIBCO, and currently being standardized by OASIS, for reliable messaging between two Web Services (21).
- WS-Reliability–An OASIS protocol for reliable messaging between two Web services (22).
- WS-Security–A specification that defines how to use XML encryption and XML signatures in SOAP messages to secure message exchange as an alternative or extension to HTTPS (23).

The growing numbers of these SOAP Web services specifications increase the complexity of the systems being built from SOAP Web services and increase the difficulty of achieving interoperability.

CHALLENGES FOR SERVICE-ORIENTED ARCHITECTURE AND WEB SERVICES

One of the primary challenges for SOA and Web Services is achieving interoperability across diverse hardware platforms, operating systems, programming languages, and data representations. The Web Services Interoperability (WS-I) organization (24) has developed the Basic Profile and the Basic Security Profile to improve interoperability. A profile is a set of core specifications (SOAP, WSDL, etc.) in a specific version (SOAP 1.1, UDDI 2, etc.) with additional requirements to restrict the use of the core specifications. The WS-I has also published use cases and tools to assess whether a Web service is conformant with the WS-I profile guidelines.

Related to the challenge of interoperability is ensuring that the XML data are interpreted in the same way by both the service requester and the service provider. Each well-defined data item, such as a date, can be represented in multiple ways. Many data items used in business interactions are much less well defined. Web services, and the

Web in general, currently focus on syntactic aspects of representing and communicating information. In contrast, semantic Web services (25) aim for automation of Web services by standardizing the representation and handling of semantic metadata to describe the services and how to use them. Semantics are difficult even for humans, and initial use of semantic Web services will be restricted to narrow, well-defined domains. The ability to locate an appropriate Web service, based on semantics, and to choose among alternative Web services, requires major advances in automated analysis of semantic information.

Another major challenge for SOA and Web services is the orchestration of services into complex applications and the management of how they interact. Metalanguages, such as the Business Process Execution Language (BPEL) (5), and specifications, such as the Web Services Choreography Description Language (WS-CDL) (26), provide a means of orchestrating fine-grained services into more coarse-grained business services, which in turn can be incorporated into business processes and workflows. As more enterprises use Web services for business interactions, more coordination between service requesters and service providers will be required. Such coordination will, in turn, require more coordination between business partners, rather than simply interfaces between service requesters and service providers.

Also a major challenge for SOA and Web services is the management of change. New services are introduced, and old services are discontinued. Data items acquire new attributes and even new meanings. Operation with multiple interfaces or multiple data representations to achieve interoperability for legacy relationships adds considerable programming complexity and a high rate of errors. SOA and Web services communities have yet to address the topic of change management.

SOA supports services on both sides of the firewall and, thus, opens up the most critical business processes and data to security and privacy risks. WS-Security (23) offers a solution for SOAP Web services, but REST Web services need to use SSL or define their own security protocols. Security appliances that parse XML messages, ensuring that known business partners originated them, can help to address the security and privacy issues, as can HTTPS and IP filtering.

Reliability is also a challenge for SOA and Web services. No guarantee exists that SOAP messages sent over HTTP or SMTP will be delivered reliably to the applications exactly once and in the correct order. The competing WS-ReliableMessaging (21) and WS-Reliability (22) specifications address this issue for SOAP Web services. However, without agreement on a single standard, reliable messaging will be implemented in an ad hoc manner, which unnecessarily complicates interoperability, portability, and extensibility.

Critics of SOA and Web services claim that they result in additional XML layers with applications running slower and consuming more processing power as a consequence. Performance is an issue because XML documents are text-based (rather than binary-based), self-describing, and interpreted. Consequently, they consume more network bandwidth, memory space, and processing cycles. However, new XML parsing and indexing technologies are available, such as VTD-XML (17), that promise to improve the performance of SOA and Web services significantly. Moreover, SOA can be implemented using technologies, such as Java Business Integration (JBI) (27), that do not depend on XML or RPC.

CONCLUSION

SOA aims to promote modularity and reuse of software components. It also aims to maintain interoperability between software applications within a single enterprise and between enterprises that operate across diverse computing platforms over the Internet. SOA can be implemented using Web services, although that is not required.

Web services depend on the use of XML to structure and tag information so that it is self-describing. Web services can be categorized as REST Web services and SOAP Web services. REST Web services currently rely only on XML and HTTP, but soon they might also depend on WADL for describing Web services. SOAP Web services use SOAP to convey XML documents and RPCs in messages between service requesters and service providers, WSDL to describe Web services so that they can be easily accessed, and UDDI to publish and discover Web services.

The potential widespread use and benefits of SOA and Web services are compelling. By supporting modularity of design and maintaining interoperability, they enable enterprises to streamline and automate their business processes and allow diverse computer systems applications to be coupled together. They offer the promise of reduced application development time and cost, increased business agility, and increased business profits.

BIBLIOGRAPHY

1. OASIS, Reference Model for the Service-Oriented Architecture (SOA). Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.
2. Open Group, Service-Oriented Architecture (SOA). Available: http://opengroup.org/projects/soa/doc.tpl?gdid=10632.
3. W3C, Web Services Architecture, 2004. Available: http://www.w3.org/TR/ws-arch.
4. W3C, eXtensible Markup Language (XML), 2004. Available: http://www.w3.org/XML/.
5. OASIS and UN/CEFACT, Extensible Business Using eXtensible Markup Language (ebXML), 2004. Available: http://www.ebxml.org/geninfo.htm.
6. P. Prescod, Second Generation Web Services, February 2002. Available: http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html.
7. P. Prescod, REST and the Real World, February 2002. Available: http://www.xml.com/pub/a/ws/2002/02/20/rest.html.
8. OASIS, Relax NG, September 2003. Available: http://www.relaxng.org/.

9. W3C, XML Schema, October 2007. Available: http://www.w3.org/XML/Schema.
10. Java Service Object Notation (JSON), 2007. Available: http://www.json.org.
11. D. Rubio, WADL: The REST Answer to WSDL, July 2007. Available:http://searchwebservices.techtarget.com/tip/0,289483,sid26_gci1265367,00.html.
12. OASIS, Reference Model for the Service-Oriented Architecture (SOA). Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.
13. W3C, Simple Object Access Protocol (SOAP), April 2007. Available: http://www.w3.org/TR/soap/.
14. W3C, Web Services Description Language (WSDL), June 2007. Available: http://www.w3.org/TR/wsdl/.
15. OASIS, Universal Description, Discovery and Integration Specifications (UDDI), 2004. Available: http://www.uddi.org/specification.html.
16. UDDI Consortium, UDDI Executive White Paper, 2001. Available: http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf.
17. VTD-XML, 2007. Available: http://vtd-xml.sourceforge.net/.
18. XMethods. Available: http://xmethods.org/ve2/index.po.
19. Y. Li, Y. Liu, L. Zhang, G. Li, B. Xie, and J. Sun, An exploratory study of Web Services on the Internet, *Proc. IEEE Int. Conf. web services*, Salt Lake City, UT, 2007, pp. 380–387.
20. IBM, Web Services Transactions Specifications (WS-AtomicTransaction, WS-BusinessActivity, WS-Coordination), 2004. Available: http://www-106.ibm.com/developerworks/library/specification/ws-tx.
21. IBM, BEA, Microsoft, and TIBCO, Web Services ReliableMessaging, 2004. Available: http://www-128.ibm.com/developerworks/webservices/library/ws-rm/.
22. OASIS, Web Services Reliability Specification (WS-Reliability), 2004. Available: http://oasis-open.org/committees/tc_home.php?wg_abbrev=wsrm.
23. OASIS, Web Services Security Specification (WS-Security), 2004. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.
24. Web Services Interoperability Organization (WS-I). Available: http://wwww.ws-i.org.
25. S. McIlraith, T. C. Son, and H. Zeng, Semantic web services, *IEEE Intelligent Sys.*, **16** (2): 46–53, 2002.
26. W3C, Web Service Choreography Description Language (WS-CDL), 2007. Available: http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/.
27. Java Community Process, Java Business Integration (JBI), 2007. Available: http://jcp.org/en/jsr/detail?id=208.

FURTHER READING

G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services Concepts: Architectures and Applications*, Berlin: Springer-Verlag, 2004.

S. Chatterjee and J. Webber, *Developing Enterprise Web Services: An Architect's Guide*, Upper Saddle River, NJ: Prentice Hall, 2003.

T. Erl, *Service-Oriented Architecture: Concepts, Technology and Design*, Upper Saddle River, NJ: Prentice Hall, 2005.

M. Fisher, The Java Web Services Tutorial 1.0., 2002. Available: http://www.java.sun.com/webservices/docs/1.0/tutorial.

E. Newcomer and G. Lomow, *Understanding SOA with Web Services*, Boston, MA: Addison Wesley, 2005.

O. Zimmerman, M. R. Tomlinson, and S. Peuser, *Perspectives on Web Services: Applying SOAP, WSDL and UDDI to Real-World Projects*, Berlin: Springer-Verlag, 2003.

LOUISE E. MOSER,
P. M. MELLIAR-SMITH
University of California
Santa Barbara, California

SHARED MEMORY MULTIPROCESSORS

INTRODUCTION

Shared memory multiprocessors are multiprocessor systems that logically implement a single global address space. The model for parallel programming based on such systems, the shared address space model, is straightforward and frees programmers from the tedious and sometimes complicated task of orchestrating all communication and synchronization through explicit message passing to access remote data. As a result, this class of multiprocessor systems has received much commercial as well as research interest. The effectiveness of shared memory systems as a cost-effective option for high-performance parallel and distributed computing is quantified by four key characteristics: *simplicity, portability, efficiency*, and *scalability*.

Simplicity: Shared memory systems provide a uniform and easy-to-use model for accessing all shared data, whether local or remote. Beyond such uniformity and ease of use, shared memory systems should provide simple programming interfaces that allow them to be platform and language independent.

Portability: The portability of the shared memory programming environment across a wide range of platforms is important as it obviates the labor of rewriting codes for large complex applications. In addition to being able to be portable across "space," good shared memory systems should also be portable across "time," i.e., be able to run on future systems, to enable system stability.

Efficiency: For shared memory systems to achieve widespread acceptance, they should be capable of providing high efficiency over a wide range of applications without requiring much programming effort, especially applications with irregular and/or unpredictable communication patterns.

Scalability: To support high-performance computing, shared memory systems should be able to run efficiently on systems with hundreds (or potentially thousands) of processors. Scalability offers end users yet another form of stability—knowing that applications running on small-to-medium systems could run unchanged and still deliver good performance on large systems.

Most existing shared memory multiprocessor systems represent a practical balance of these properties. The shared memory abstraction in existing systems is supported either in hardware or in software or using a hybrid approach. Figure 1 illustrates the spectrum of shared memory multiprocessor systems. Based on the underlying architectural approach, the current systems can be broadly grouped into two categories: (*1*) physically shared memory (PSM) multiprocessors and (*2*) distributed shared memory (DSM) multiprocessors. However, irrespective of their architectures, shared memory systems must address two critical issues, cache coherence and memory consistency.

Cache Coherence

Although the shared memory abstraction enables global accesses to remote data in a straightforward manner, the difference in access time between local and remote memory accesses in some of these architectures is significant (access times may differ by a factor of 10 or higher). Local caches can be used to hide long remote memory access times. However, ensuring coherence of cached data across the multiprocessor system with (possibly remote) memory is a challenging problem (Fig. 2). Two key approaches have been used to maintain cache coherence.

Snoopy Cache Coherence Protocols. Small shared memory multiprocessor systems that are based on a shared bus implement *snoopy protocols* to maintain cache coherence. In this approach, all caches snoop on the shared "snoopy" bus. When a processor writes into a shared cache block, the write request is transmitted on the bus. All caches snooping on the bus read the address associated with every read/write request and check whether they are currently caching that address. If a cache contains the address, the corresponding entry in the cache is invalidated in case of a write request, or it is used to satisfy the read request. For write-through caches, where data are simultaneously written to main memory and cache, the snoopy protocol is only an incremental addition to the normal cache protocol and the memory is always up-to-date. Write-back caches, which copy modified data to the source memory only when a cache block is replaced, require extra work to implement the protocol. In this type of cache, the most recently modified copy of the data may be in some processor's cache, and on a read miss, the coherence protocol has to retrieve these data by snooping all the caches.

Snoopy protocols require all read and write requests to be broadcast on the bus. As the bus processes only one request at a time, concurrent writes to the same cache are automatically serialized. This serialization of requests by the bus imposes an ordering on all writes, which is critical to maintaining coherence. For small multiprocessor systems (up to 64 processors), snoopy cache coherence protocols work well. The use of caches reduces bandwidth requirements for the bus and main memory. Furthermore, as the caches are kept functionally transparent, the shared-memory programming model is preserved. For larger systems, however, the bus becomes a communication bottleneck.

Directory-Based Cache Coherence. Directory-based cache coherence protocols use a directory to keep track of the caches that share the same cache line. The individual caches are inserted into and deleted from the directory to reflect the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

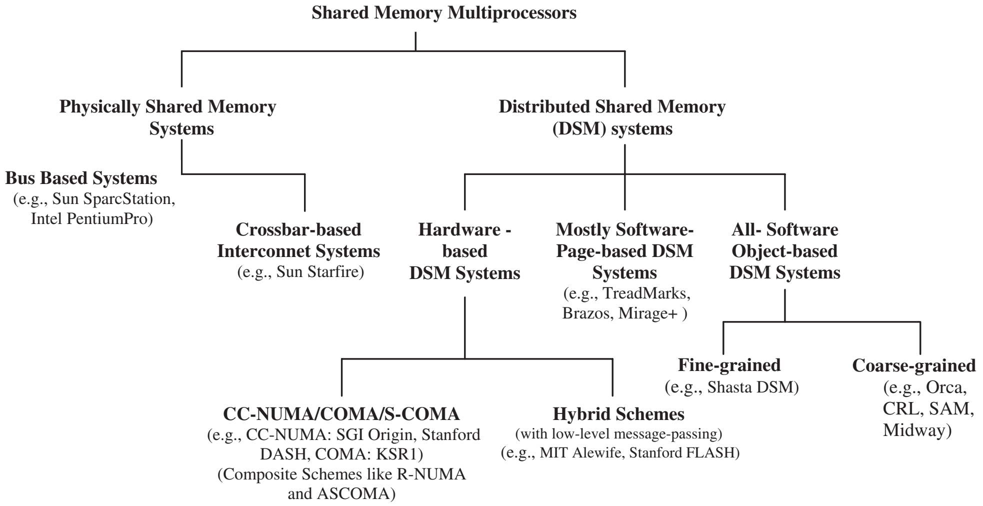

Shared Memory Multiprocessors
Physically Shared Memory Systems
Distributed Shared Memory (DSM) systems
Bus Based Systems (e.g., Sun SparcStation, Intel PentiumPro)
Crossbar-based Interconnet Systems (e.g., Sun Starfire)
Hardware - based DSM Systems
Mostly Software- Page-based DSM Systems (e.g., TreadMarks, Brazos, Mirage+)
All- Software Object-based DSM Systems
Fine-grained (e.g., Shasta DSM)
Coarse-grained (e.g., Orca, CRL, SAM, Midway)
CC-NUMA/COMA/S-COMA (e.g., CC-NUMA: SGI Origin, Stanford DASH, COMA: KSR1) (Composite Schemes like R-NUMA and ASCOMA)
Hybrid Schemes (with low-level message-passing) (e.g., MIT Alewife, Stanford FLASH)

Figure 1. Taxonomy of shared memory multiprocessors.

use or rollout of shared cache lines. This directory is also used to purge (invalidate) a cached line because of a remote write to that line.

The directory can either be centralized or distributed among nodes of the shared memory multiprocessor system. Generally, a centralized directory is implemented as a bit map of the individual caches, where each bit set represents a shared copy of a particular cache line. The advantage of this type of implementation is that the entire sharing list can be found by simply examining the appropriate bitmap. However, each potential reader and writer has to access the centralized directory, which becomes a bottleneck. Additionally, the reliability of the scheme is a serious issue as a fault in the bit map would result in an incorrect sharing list.

The bottleneck and single point of failure resulting from a centralized directory is alleviated by distributing the directory. The distributed directory scheme (also called the distributed pointer protocol) implements the sharing list as a distributed linked list. In this implementation, each directory entry (corresponding to a cache line) points to the next member of the sharing list. Cache lines are inserted into and deleted from the linked list as necessary.

Shared Memory Consistency Models

In addition to the use of caches, scalable-shared memory multiprocessor systems migrate or replicate data to local processors. Most scalable systems choose to replicate (rather than migrate) data as this gives the best performance for a wide range of application parameters. With replicated data, maintaining *memory consistency* becomes an important issue. The shared memory scheme (hardware or software) must control replication in a manner that preserves the abstraction of a single address-space shared memory.

The shared memory consistency model refers to how local updates to shared memory are communicated to the processors in the system. The most intuitive model is that a read should always return the last value written. However, the idea of "the last value written" is not well defined in multiprocessor environments, and its different interpretations have given rise to a variety of memory consistency models such as sequential consistency (1), processor consistency, release consistency (2), entry consistency (3), scope consistency (4), and variations of these.

"Sequential consistency" requires that the result of any execution is the same as if the operations of all processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program. This definition implies (*1*) maintaining program order among operations from individual processors and (*2*) maintaining a single sequential order among operations from all processors. The second aspect makes it appear as if a memory operation executes atomically or instantaneously with respect to other memory operations (5). The simplicity of this model, however, exacts a high price because sequentially consistent memory systems preclude many optimizations, such as reordering, batching, or coalescing, which are feasible in uniprocessor system. These optimizations reduce the performance

Time	Processor P1	Processor P2
↓	x = 0	y = 0
	x = a	y = b
	y = c	
		x = d

Figure 2. Coherence problem when shared data are cached by multiple processors. Suppose initially x = y = 0 and both P1 and P2 have cached copies of x and y. If coherence is not maintained, P1 does not get the changed value of y and P2 does not get the changed value of x.

Table 1. Relaxations in Program Order Allowed by Different Memory Consistency Models

Model	Program Order Relaxation
Sequential consistency	None
Processor consistency	Write->Read
Weak ordering	Data->Data
Release consistency*	Data->Data, Data->Acquire, Release->Data, Release->Acquire

*Release consistency categorizes synchronization operations into acquires and releases.

Note: Only the program order between memory operations to different locations is considered.

impact of having distributed memories and have led to a class of weakly consistent models.

A weaker memory consistency model offers fewer guarantees about memory consistency, but it ensures that a "well-behaved" program executes as though it were running on a sequentially consistent memory system. Once again the definition of "well behaved" varies according to the model. Relaxed memory consistency models can be categorized based on two key characteristics: (*1*) how is the program order requirement relaxed and (*2*) how is the write atomicity requirement relaxed. Program order relaxations include relaxing the order from a write to a following read, between two writes, and from a read to a following read or write. Atomicity relaxations differ in whether they allow a read to return the value of another processor's write before the write is made visible to all other processors. Table 1 summarizes hardware-centric relaxations in program order allowed by the various memory consistency models commonly supported in commercial systems.

Some researchers have proposed DSM systems that support a family of consistency protocols or application-specific protocols, and programmers are allowed to choose any one of them for each memory object (or page) or each stage of an application. Although this scheme might be able to achieve optimal performance, it does impose undue burden on the programmer. Another suggestion is to implement adaptive schemes that automatically choose the appropriate memory consistency protocol to trade off between performance and programming ease.

As many recent high-performance computing platforms have been built by loosely connecting a constellation of clusters, each of them being made of a set of tightly connected nodes, hierarchy-aware consistency protocols have been proposed. At first such protocols have focused on improving locality in data management by caching remote data within clusters. Later studies have addressed locality in synchronization management, which is also a major source of inefficiency.

Programs with good behavior do not assume a stronger consistency guarantee from the memory system than is actually provided. For each model, the definition of "good behavior" places demands on the programmer to ensure that a program's access to the shared data conforms to that model's consistency rules. These rules add an additional dimension of complexity to the already difficult task of writing new parallel programs and porting old ones. But the additional programming complexity provides greater control over communication and may result in higher performance. For example, with entry consistency, communication between processors occurs only when a processor acquires a synchronization object.

PHYSICALLY SHARED MEMORY (PSM) MULTIPROCESSORS

The structure of a physically shared memory multiprocessor system is illustrated in Fig. 3. A small number of microprocessors (typically less than 64) is integrated with a common memory using a shared bus or a crossbar interconnect, that allows all processors to have roughly equal access time to the centralized main memory, i.e., uniform memory access (UMA). Physically shared memory multiprocessors are also termed as *symmetric multiprocessors* (SMPs) or *centralized shared memory processors*. PSM multiprocessors using a shared bus interconnect are called *bus-based symmetric multiprocessors*.

The primary strengths of PSM systems include uniform memory access and a single address space offering the ease of programmability. These systems do not need explicit data placement, as memory is equally accessible by all processors. The PSM design approach is used in commercially successful machines such as the Compaq PentiumPro and Sun UltraEnterprise.

Bus-Based Systems

Early PSM machines, including many desktop PCs and workstations, used a shared serial bus with an address cycle for every transaction. This tied the bus during each

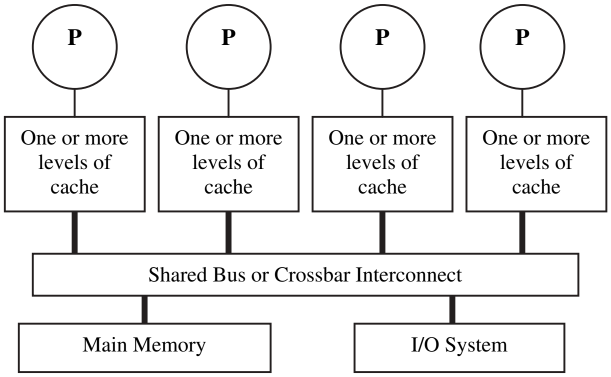

P
P
P
P
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Shared Bus or Crossbar Interconnect
Main Memory
I/O System

Figure 3. Physically shared memory multiprocessors (P: Processor, I/O: Input and Output).

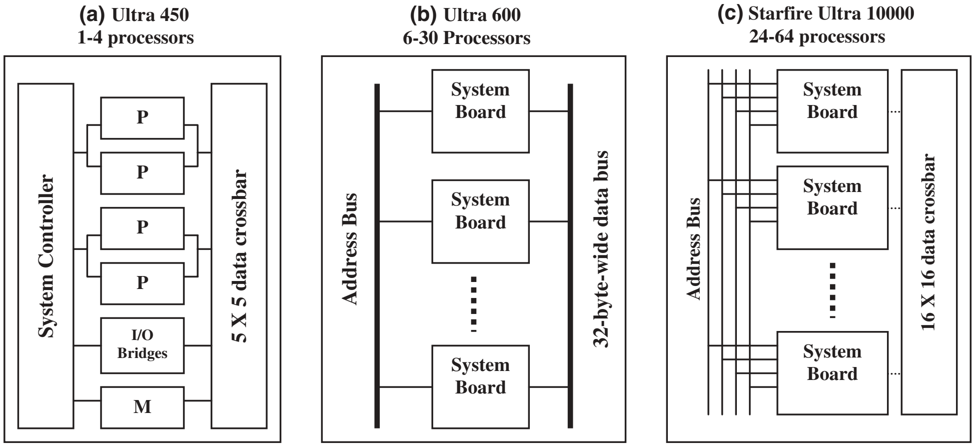


Figure 4. Three Ultra Port Architecture Implementations: (a) small system consisting of a single board with four processors, I/O interfaces, and memory; (b) a medium-sized system with one address bus and a wide data bus between boards; and (c) a large system with four address buses and a data crossbar between boards. [Source: A. Charlesworth (6).]

access for the needed data to arrive. An example is the Sun Microsystems Mbus (6) used in SparcStations. This shared bus, in addition to allowing access to the common memory, is used as a broadcast medium to implement the *snoopy cache coherence protocol*.

Subsequent PSM designs used the *Split-Transaction Bus* (7), with separate buses for address and data. It allows an address cycle to overlap with a data transfer cycle. Split-Transaction buses were used in Sun Microsystems' original Gigaplane (7), in the Ultra-Enterprise 3000-6000. The split-transaction bus also allowed overlapping of snooping and data transfer activities, thereby increasing bandwidth. However, this overlapping needed handshaking during data transfer. The *Pipelined Bus*, used as the PentiumPro System Bus, is a special case of split-transaction bus wherein the address and data cycles are pipelined and devices can respond only in specific cycles, obviating the need for the data handshake. This scheme, however, requires the data cycle to correspond to the slowest device.

Crossbar-Based PSM Systems

In crossbar-based systems, the data bus is replaced with a crossbar switch to provide high-performance UMA. The address bus is also replicated by a factor of four. Point-to-point routers and an active center-plane with four address routers are the key components of the larger UMA symmetric multiprocessors such as the Sun Ultra Enterprise series (6).

Although physically shared memory multiprocessor architectures are used in most commercially successful machines, these systems have relatively high minimum memory access latency as compared with high-performance uniprocessor systems. Furthermore, the inherent memory contention in these systems limits their scalability.

Example System—Sun Ultra-Port Architecture

Figure 4 illustrates the family of SUN Ultra Port Architectures (6) used in their workstations. These systems use a combination of bus and crossbar to implement shared memory. In the smaller Ultra 450 system (1–4 processors), illustrated in Fig. 4(a), a centralized coherency controller and a crossbar is used to connect the processors directly to the shared memory. This system is a relatively low-cost single-board configuration. The intermediate-sized Ultra 6000 system has a Gigaplane bus that interconnects multiple system boards and is designed to provide a broad range of expandability with the lowest possible memory latency, typically (216 ns for a load miss). This scheme supports systems with 6 to 30 processors and is shown in Fig. 4(b). For large systems with 24 to 64 processors, the address bus is replicated by a factor of four. The scheme is illustrated in Fig. 4(c). These four address buses are interleaved so that memory addresses are statically divided among the four buses; i.e., each address bus covers one quarter of the physical address space. A 16 × 16 crossbar is chosen to match the quadrupled snoop rate. To avoid failures on one system board from affecting other boards, and to electrically isolate the boards, point-to-point router application-specific integrated circuits (ASICs) are used for the entire interconnect, i.e., for the data crossbar, the arbitration interconnect, and the four address buses. The ASICs are mounted on a centraplane, which is physically and electrically in the middle of the system.

PHYSICALLY DISTRIBUTED MEMORY ARCHITECTURES

The structure of a typical distributed memory multiprocessor system is shown in Fig. 5. This architecture enables scalability by distributing the memory throughout the machine and by using a scalable interconnect to enable processors to communicate with the memory modules. Based on the communication mechanism provided, these architectures are classified as multicomputer/message passing architectures and DSM architectures. The multicomputers use a software *Message Passing* layer to communicate among themselves, and they are called message passing architectures. In these systems, programmers are required to explicitly send messages to request/send remote data. As these systems connect multiple computing nodes sharing only the scalable interconnect, they are also referred to as multicomputers.

DSM machines logically implement a single global address space although the memory is physically distributed. The memory access times in these systems depended

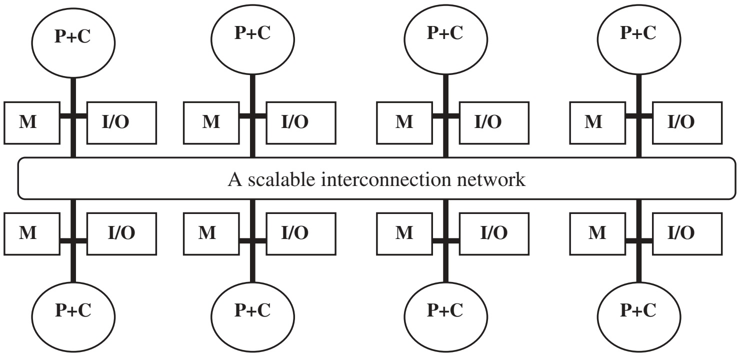


Figure 5. Distributed memory multiprocessors (P+C: Processor + Cache, M: Memory). Both message-passing systems and DSM systems have the same basic organization. The key distinction is that the DSMs implement a single shared address space.

on the physical location of the processors and are no longer uniform. As a result, these systems are also termed as nonuniform memory access (NUMA) systems.

Classification of Distributed Shared Memory (DSM) Systems

Providing DSM functionality on physically distributed memory requires the implementation of three basic mechanisms.

Processor side hit/miss check: This operation, on the processor side, is used to determine whether a particular data request is satisfied in the processor's local cache. A "hit" is a data request satisfied in the local cache, whereas a "miss" requires the data to be fetched from the main memory or the cache of another processor.

Processor side request send: This operation is used on processor side in response to a "miss," to send a request to another processor or the main memory for the latest copy of a data item and wait for a response.

Memory side operations: These operations enable the memory to receive a request from a processor, perform any necessary coherence actions, and send its response typically in the form of the requested data.

Based on how these mechanisms are implemented in hardware/software, various DSM systems can be classified as list in Table 2.

Almost all DSM models employ a directory-based cache coherence mechanism implemented either in hardware or software, which makes these systems highly scalable. DSM systems have demonstrated the potential to meet the objectives of scalability, programmability, and cost-effectiveness (8, 9). In general, hardware DSM systems provide excellent performance without sacrificing programmability. Software DSM systems typically provide a similar level of programmability while trading some performance for reduced hardware complexity and cost.

Hardware-Based DSM Systems

Hardware-based DSM systems implement the coherence and consistency mechanisms in hardware, which makes these systems faster but more complex. Clusters of symmetric multiprocessors, or SMPs, with hardware support for shared memory, have emerged as a promising approach to building large-scale DSM parallel machines. Each node in these systems is an SMP with multiple processors. The relatively high volumes of these small-scale parallel servers make them extremely cost-effective as building blocks.

Hardware-Based DSM System Classification. In hardware-based DSM systems, software compatibility is preserved using a directory-based cache coherence protocol. This protocol supports a shared-memory abstraction despite having memory physically distributed across the nodes. Several cache coherence protocols have been proposed for these systems. These protocols include (*1*) cache-coherent nonuniform memory access (CC-NUMA), (*2*) cache-only memory access (COMA), (*3*) simple cache-only memory access (S-COMA), (*4*) reactive-NUMA, and (*5*) adaptive S-COMA. Figure 6 illustrates the processor memory hierarchies for CC-NUMA, COMA, and S-COMA architectures.

Cache Coherent Nonuniform Memory Access (CC-NUMA). Figure 6(a) shows the processor memory hierarchy in a

Table 2. DSM Systems Classification

System Type	Hardware-implemented	Software-implemented	Sample Systems
Hardware-based DSM	All processor side mechanism	Some part of memory side support	SGI Origin (8), HP/Convex Exemplar (9), IBM RP3 (10), MIT Alewife (11), and Stanford FLASH (12)
Mostly software-based DSM	Hit/miss check based on virtual memory protection mechanism	All other support Coherence unit is virtual memory page	TreadMarks (2), Brazos (4), and Mirage+ (13)
Software-based DSM	None	All three mechanisms mentioned above	Orca (1), SAM (14), CRL (15), Midway (3), and Shasta (16)

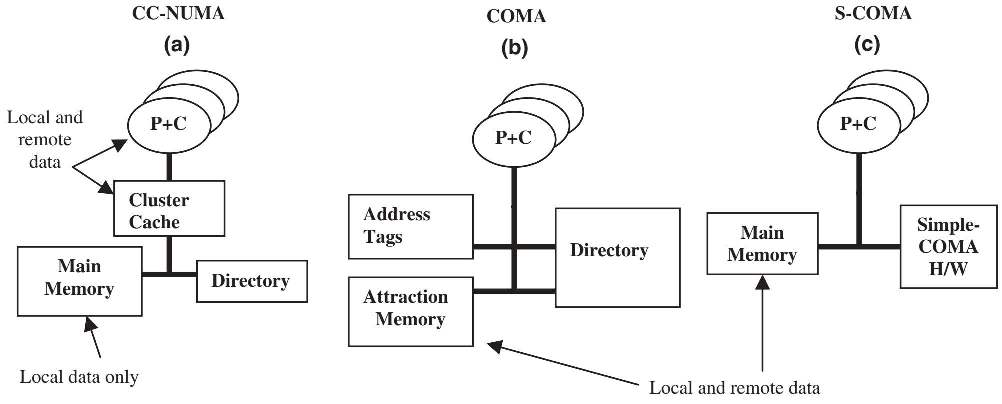


Figure 6. Processor memory hierarchies in CC-NUMA, COMA, and S-COMA (P–C: Processor – Cache, H/W: Hardware).

CC-NUMA system. In this system, a per-node cluster cache lies next to the processor cache in the hierarchy. Remote data may be cached in a processor's cache or the per-node cluster cache. Memory references not satisfied by these hardware caches must be sent to the referenced page's home node to obtain the requested data and perform necessary coherence actions. The first processor to access a remote page within each node results in a software page-fault. The operating system's page fault handler maps the page to a CC-NUMA global physical address and updates the node's page table. The Stanford DASH (17) and SGI Origin (8) systems implement the CC-NUMA protocol.

Cache-Only Memory Access (COMA). The key idea behind the COMA architecture is to use the memory within each node of the multiprocessor as a giant cache (also termed as attraction memory), which this is shown in Fig. 6(b). Data migration and replication is the same as in regular caches. The advantage of this scheme is the ability to capture remote capacity misses as hits in local memory; i.e., if a data item is initially allocated in a remote memory and is frequently used by a processor, it can be replicated in the local memory of the node where it is being frequently referenced. The attraction memory maintains both the address tags as well as the state of data. The COMA implementation requires a customized hardware and hence has not become a popular design choice. The Kendall Square Research KSR1 (18) machine implemented the COMA architecture.

Simple Cache Only Memory Access (S-COMA). A S-COMA system [shown in Fig. 6(c)] uses the same coherence protocol as CC-NUMA, but it allocates part of the local node's main memory to act as a large cache for remote pages. S-COMA systems are simpler and much cheaper to implement than COMA, as they can be built with off-the-shelf hardware building blocks. They also use standard address translation hardware. On the first reference to a remote page from any node, a software page fault occurs, which is handled by the operating system. It initializes the page table and maps the page in the part of main memory being used as cache. The essential extra hardware required in S-COMA is a set of fine-grain access control bits (1 or 2 per block) and an auxiliary translation table. The S-COMA page cache, being part of main memory, is much larger than the CC-NUMA cluster cache. As a result, S-COMA can outperform CC-NUMA for many applications. However, S-COMA incurs substantial page overhead as it invokes the operating system for local address translation. Additionally, programs with large sparse data sets suffer from severe internal fragmentation resulting in frequent mapping and replacement (or swapping) of the S-COMA page caches, which is a phenomenon called thrashing. In such applications, CC-NUMA may perform better. As S-COMA requires only incrementally more hardware than CC-NUMA, some systems have proposed providing support for both protocols. For example, the S3.mp (19) project at Sun Microsystems supports both S-COMA and CC-NUMA protocols.

Hybrid Schemes—Reactive-NUMA and ADAPTIVE-SCOMA. Given the diversity of application requirements, hybrid schemes such as reactive-NUMA (R-NUMA) (20) and adaptive-SCOMA (ASCOMA) (21) have been proposed. These techniques combine CC-NUMA and S-COMA to get the best of both with incrementally more hardware. These schemes have not yet been implemented in commercial systems.

Example Systems. Table 3 presents several research/commercial hardware-based DSM systems.

Recent Advances. Sequential consistency imposes more restrictions than simply preserving data and control dependences at each processor. It can restrict several common hardware and compiler optimizations used in uniprocessors. Relaxed consistency models allow optimization to some extent by permitting relaxations of some program ordering. As it is sufficient to only appear as if the ordering rules of the consistency model are obeyed (22), some researchers have proposed deploying features, such as out-of-order scheduling, non-blocking loads, speculation, and prefetching, into recent processors to improve the performance of consistency models. Three such hardware techniques are described below.

Hardware Prefetching: The instruction window is used to maintain several decoded memory instructions. In existing hardware-based DSM implementations,

Table 3. Hardware-Based DSM Systems

System Name	System Features
SGI Origin (8) (Fig. 7)	The Origin adopts the directory-based cache coherence protocol. Its primary design goal is to minimize the latency difference between remote and local memory, and it includes hardware and software support to ensure that most memory references are local. It primarily supports the shared-memory programming model.
HP/CONVEX Exemplar (Fig. 8) (9)	The Exemplar adopts the two-tiered directory-based cache-coherence protocol. Its primary design goal is to combine the parallel scalability of message-passing architectures with hardware support for distributed shared memory, global synchronization, and cache-based latency management. It supports shared-memory and message-passing programming models.
IBM RP3 (10) (Fig. 9)	The RP3 adopts the directory-based cache coherence protocol. Its primary design goal is to evenly distribute the global address space across all modules to balance access requests across the modules. It supports the shared-memory programming model.
The MIT Alewife Machine (11) (Fig. 10)	The Alewife machine adopts a software-extended cache coherence scheme called LimitLESS (23,24), which implements a full-map directory protocol. Its primary design goal is to combine several mechanisms, including software-extended coherent shared memory, integrated message passing, support for fine-grain computation, and latency tolerance, to enable parallel systems to be both scalable and programmable. It supports shared-memory and message-passing programming models.
The Stanford FLASH Multiprocessor (12) (Fig. 11)	FLASH adopts the directory-based cache coherence protocol. Its primary design goal is to efficiently integrate cache coherent shared memory and low overhead user-level message passing. It supports shared-memory and message-passing programming models.

these instructions may not be issued to the memory because of consistency constraints. With hardware prefetching, the processor can issue nonbinding prefetches for these instructions without violating the consistency model, thus hiding some memory latency.

Speculative Load Execution: This technique speculatively consumes the value of loads brought into the cache, regardless of consistency constraints. In case consistency is violated, the processor rolls back its execution to the incorrect load.

Cross-Window Prefetching: Instructions currently not in instruction window but expected to be executed in the future can also be prefetched. This technique alleviates the limitations imposed by a small instruction window size.

At the processor level, the above techniques narrow the performance gap between consistency models. Other design decisions below the processor level, such as cache write policy and cache coherence protocol, can also affect the performance of the consistency model.

Software-Based DSM Systems

These systems use software to, either partially or completely, implement shared memory. This alternative approach has been used by several DSM systems. Based on their design, these DSM systems can be classified as *mostly software-based systems* and *all software systems*.

Mostly DSM systems are page-based systems. They make use of the virtual memory hardware in the underlying system to implement shared memory consistency

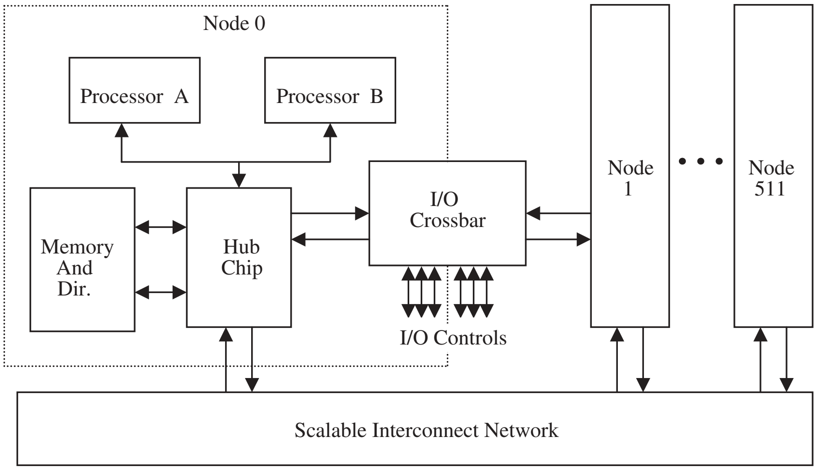


Figure 7. Origin block diagram. [Source: J. Laudon et. al. (8).]

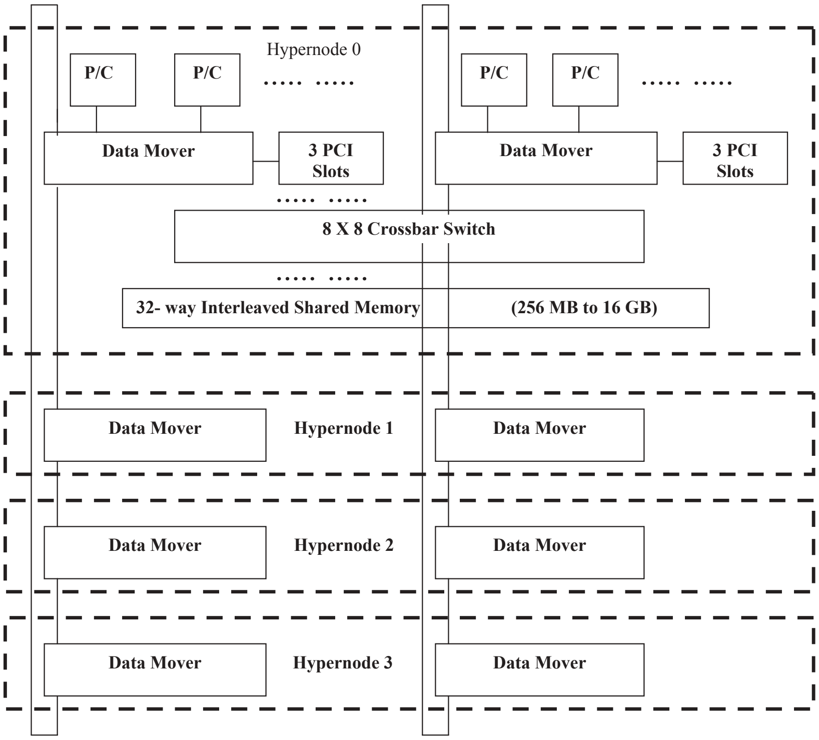


Figure 8. Architecture of the HP/Convex Exemplar X-Class SPP. [P/C: Processor/ Cache, CTI: Coherent Toroidal Interconnect, PCI: Peripheral component interconnect. Source: T. Brewer et. al. (9).]

models in software and to resolve conflicting memory accesses (memory accesses to the same location by different processors, at least one of which is a write access). Examples of mostly software page-based DSM systems include TreadMarks (2), Brazos (4) and Mirage+ (13).

The advantage of page-based DSM systems is that they eliminate the shared-memory hardware requirement, which makes them inexpensive and readily implementable. These systems have been found to work well for certain applications classes, e.g., dense matrix codes (2). As the coherence policy is implemented in software, it can be optimized to make use of the operating system to implement coherence mechanisms. The use of the operating system, however, makes it slow as compared with hardware coherence mechanisms. Additionally, the coarse sharing granularity (i.e., large page size) results in false sharing and relatively higher communication time per page. One solution is to have multigrain systems, e.g., using fine-grain shared memory within an SMP and page-based distributed-shared memory across SMPs.

All-software DSM systems are typically object-based systems. The virtual view of a shared address space is implemented entirely in software in these systems. Examples for DSM systems in this category include Orca (1), SAM (14), Midway (3), CRL (15) and Shasta (16).

Write-Update and Write-Invalidate Protocols. A key issue in software-based DSM systems is the *write protocol.* Two approaches maintain the memory coherence requirement for a write operation. One approach is to ensure that a processor has an exclusive access to a data item before it writes to it, which is the *write invalidate* protocol because it invalidates all other copies on a write. It is by far the most common protocol. The other alternative is to update all the cached copies of a data item when it is written, which is the *write update* protocol.

Single- and Multiple-Writer Protocols. Most DSM systems (and hardware caches) use single-writer protocols.

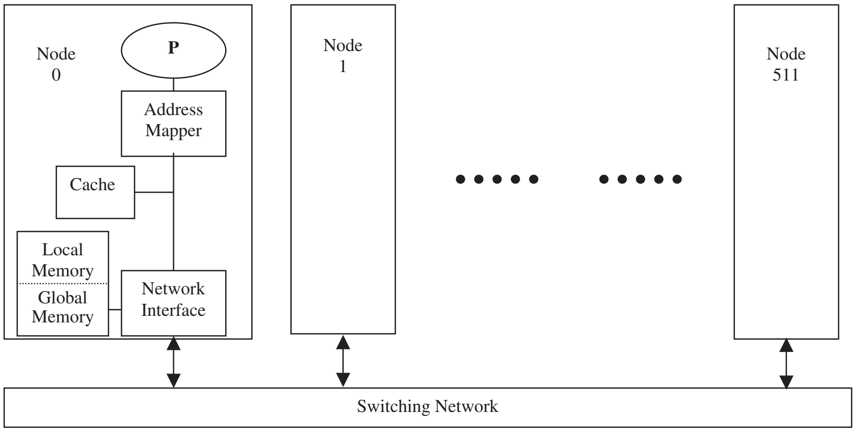


Figure 9. IBM RP3 block diagram.

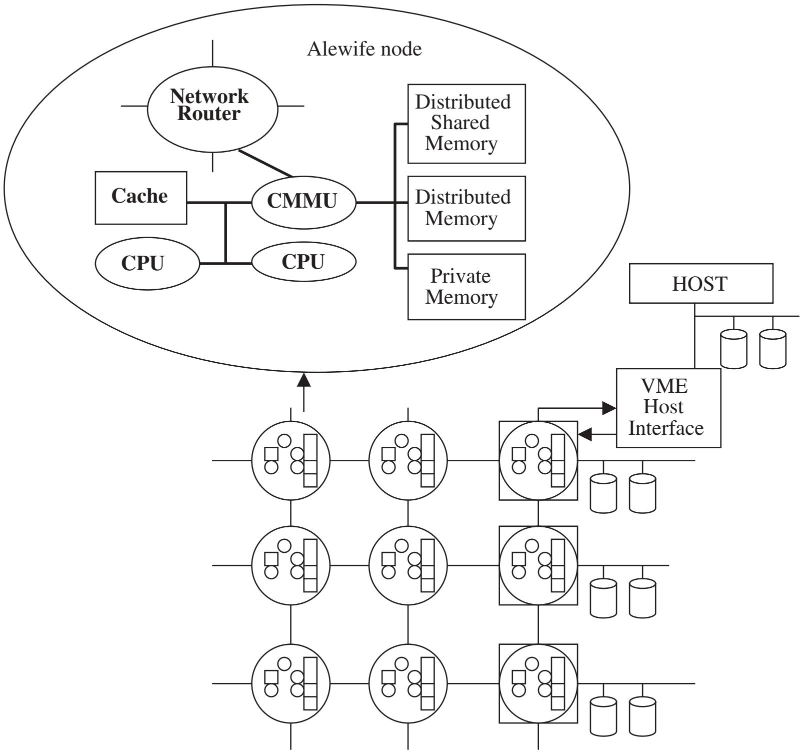


Figure 10. The Alewife architecture (CMMU: Communication and Memory Management Unit, FPU: Floating-point Unit).

These protocols allow multiple readers to access a given page simultaneously, but a writer is required to have exclusive access to a page before making any modifications. Single-writer protocols are easy to implement because all copies of a given page are always identical, and page-fault can always be satisfied by retrieving a valid copy of the page. This simplicity often comes at the expense of high message traffic. Before a page can be written, all other copies must be invalidated. These invalidations can then cause subsequent access misses, if the processors whose pages have been invalidated are still accessing the page's data. *False sharing* occurs when two or more unrelated data objects are located in the same shared page and are written concurrently by separate processors. As the consistency unit (usually a virtual memory page) is large in size, *false sharing* is a potentially serious problem. It causes the performance of the single-writer protocol to further deteriorate because of interference between unrelated accesses. Multiple-writer protocols allow multiple processors to simultaneously modify their local copy of a shared page. The modifications are then merged at certain points of execution.

Example Systems. Table 4 presents several software-based DSM systems.

EMERGING ENABLING TECHNOLOGIES FOR SHARED MEMORY SYSTEM

Recent years have seen the emergence of hardware devices customized to support certain types of shared memory system implementations. Furthermore, standards and technologies have emerged that have the potential to facilitate shared memory system implementations in a broader way.

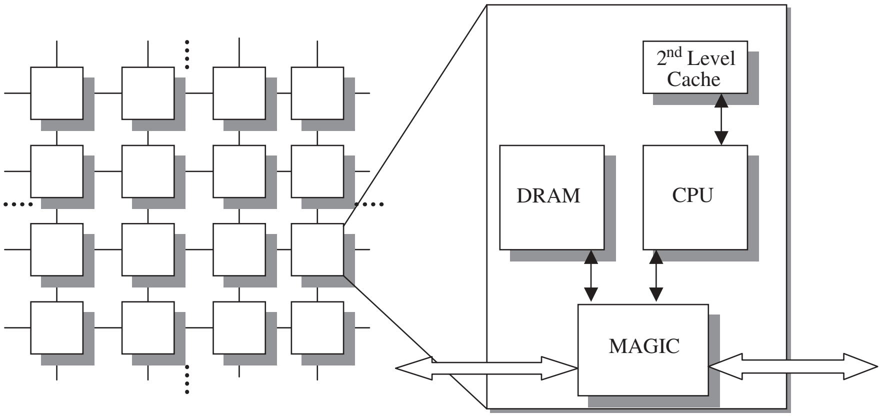


Figure 11. FLASH system architecture. [Source: J. Kuskin et al. (12).]

Table 4. Software-Based DSM Systems

System Name	System Features
	Page-based DSM systems
TreadMarks (2)	TreadMarks is a mostly software-page-based DSM system. It uses lazy release consistency as its memory consistency protocol and adopts the multiple-writer protocol to reduce false-sharing effect. TreadMarks is implemented on a network of workstations.
Brazos (4)	Brazos is a mostly software page-based DSM system. The Brazos implements a scope consistency model, which is a bridge between the release consistency and entry consistency models. Brazos is implemented on network of workstations.
Mirage+ (13)	Mirage+ is a mostly-software page-based DSM system. It extends the strict coherence protocol of the IVY system (25). It also allocates a time window during which nodes possess a page, which provides some degree of control over processor locality. Mirage+ is implemented on a network of personal computers.
	Object-based DSM systems
Orca (1)	Orca is an all-software object-based DSM system. It implements sequential consistency and adopts the write-update coherence protocol with function shipping and totally ordered group communication to achieve competitive performance.
SAM (14)	SAM is an all-software object-based DSM system. Its design ties synchronization with data access and avoids the need for coherence communication. It is implemented as a portable C library and supports user-defined data types.
Midway (3)	Midway is an all-software object-based DSM system. It supports multiple consistency models within a single parallel program and requires a small amount of compile time support to implement its consistency protocols.
CRL (15)	CRL is an all-software DSM system. It employs a fixed-home, directory-based write-invalidate protocol and provides memory coherence through entry or release consistency. It is implemented as a library.
Shasta DSM (16)	Shasta is a fine-grained all-software DSM system. It supports coherence at fine-granularity, and coherence is maintained using a directory-based invalidation protocol. A key design goal of Shasta is to overcome both the false sharing and the unnecessary data transmission.
DSM Using. NET (26)	This is an all-software object-based DSM system. It follows a Multiple Readers Multiple Writers (MRMW) memory model. Its implementation is based on the Microsoft .NET framework adding facilities for object sharing and replication and relies on the availability of IPv4 or IPv6 (unreliable) multicast.
Orion (27)	Orion is an all-software DSM system. It implements the home-based eager release consistency model. Adaptive schemes for the home-based model are also proposed to provide good performance with minimal user intervention. A POSIX-thread-like interface is provided.
DSZOOM-WF (28)	DSZOOM-WF is an all-software DSM system. It implements the sequential consistency model. It assumes basic low-level primitives provided by the cluster interconnect and the operating system bypass functionality to avoid the overhead caused by interrupt- and/or poll-based asynchronous protocol processing, which affects the performance of most software-based DSM systems.

SCI: Scalable Coherent Interface

Scalable Coherent Interface (SCI) is an ANSI/IEEE 1596-1992 standard that defines a point-to-point interface and a set of packet protocols. The SCI protocols use packets with a 16-byte header and 16, 64, or 256 bytes of data. Each packet is protected by a 16-bit CRC code. The standard defines 1-Gbit/second serial fiber-optic links and 1-Gbyte/second parallel copper links. SCI has two unidirectional links that operate concurrently. The SCI protocols support shared memory by encapsulating bus requests and responses into SCI request and response packets. Packet-based handshake protocols guarantee reliable data delivery. A set of cache coherence protocols is defined to maintain cache coherence in a shared memory system. SCI technology has been used to implement DSM systems, e.g., the hardware-based DSM system HP/CONVEX Exemplar. Recently it has also been adopted to build software-based DSM systems, e.g., a cluster of PCs interconnected by a SCI network providing a memory-mapped file abstraction (29).

Active Memory Techniques for CCNUMA Multiprocessors

Active memory systems provide a promising approach to overcome the memory wall (30) for applications with irregular access patterns that are not amenable to techniques like prefetching or improvements in the cache hierarchy. The central idea in this approach is to perform data-parallel computations or scatter/gather operations, via address remapping techniques in the memory system, to either offload computation directly or to reduce the number of processor cache misses. This technique is expanded to multinode hardware DSM systems (31) using the same active memory controller with an integrated commodity network interface and without any hardware modifications, by designing appropriate extensions to the DSM cache coherence protocol.

APPLICATIONS OF SHARED MEMORY MULTIPROCESSOR SYSTEM

Shared memory multiprocessor systems are traditionally used to provide an intuitive programming model for parallel programs based on shared memory. Memory sharing technology is also viewed as a building block for constructing a Single System Image (SSI). It can also be used for code coupling or for realizing shared data repositories.

Single System Image (SSI)

The computing trend is moving from clustering high-end mainframes to clustering desktop computers, triggered by widespread use of PCs, workstations, gigabyte networks, and middleware support for clustering (32). Future clusters will offer increased SSI support with better transparency, for which a single memory space is a fundamental building block.

Code Coupling/Shared Data Repository

Mome (33), a user-level DSM, is designed to provide a shared segment space for parallel programs running on distributed memory computers or clusters. Besides supporting high-performance SPMD applications, the system also targets coupling of parallel applications using an MIMD model. The Mome DSM allows heterogeneous processes running on distributed memory architectures and clusters to share data by mapping the shared memory segments into their address space. A persistent data repository for parallel applications is enabled by allowing programs to dynamically connect to the DSM, map existing segments on their memory, read and modify the data, and leave this data in the repository for further use by other programs.

CONCLUDING REMARKS

Shared-memory machines built with symmetric multiprocessors and clusters of distributed multiprocessors are becoming widespread, both commercially and in academia (1,3,4,6,8,9,11–13,15,19,20,34). Shared memory multiprocessors provide ease of programming while exploiting the scalability of distributed-memory architectures and the cost-effectiveness of SMPs. They provide a shared memory abstraction even though memory is physically distributed across nodes. Key issues in the design of the shared memory multiprocessors are cache coherence protocols and shared memory consistency models, as discussed in this article. Symmetric multiprocessors (SMPs) typically use snoopy cache coherence protocols, whereas, the DSM systems are converging toward directory-based cache coherence. More popular consistency models include sequential consistency, release consistency, and scope consistency.

High-level optimizations in the programming model, such as single global address space and low latency access to remote data, are critical to the usability of shared memory multiprocessors. However, these optimizations directly trade off with system scalability and operating system performance. Current shared memory multiprocessors are built to achieve very high memory performance in bandwidth and latency (6,8). An important issue that needs to be addressed is the input/output behavior of these machines. The performance of distributed input/outputs and the distributed file system on a shared memory abstraction need to be addressed in the future designs.

BIBLIOGRAPHY

1. H. E. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, K. Langendoen, and T. Ruhl, Performance evaluation of the Orca shared object system, *ACM Trans. Comput. Syste.*, 1998.
2. C. Amza, A. Cox, S. Dwarakadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, TreadMarks: Shared memory computing on networks of workstations, *IEEE Comput.*, 1996.
3. B. Bershad, M. Zekauskas, and W. Swadon, The midway distributed shared memory system, *Proc. IEEE International Computer Conference (COMPCON)*, 1993.
4. E. Speight and J. K. Bennett, Brazos: A third generation DSM system, *Proc. 1997 USENIX.*
5. S. V. Adve and K. Gharachorloo, Share memory consistency models: A tutorial, WRL Research Report 95/7, September 1995.
6. A. Charlesworth, STARFIRE: Extending the SMP envelope, *Proc. IEEE MICRO*, January/February 1998.
7. Sun Enterprise X000 Server Family: Architecture and Implementation. Available: http://www.sun.com/servers/whitepapers/arch.html.
8. J. Laudon and D. Lenoski, The SGI Origin: A ccNUMA Highly Scalable Server. Available: http://www-europe.sgi.com/origin/tech_info.html.
9. T. Brewer and G. Astfalk, The evolution of HP/Convex Exemplar, *Proc. IEEE Computer Conference (COMPCON)*, Spring, February 1997.
10. G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, A. Norton, and J. Weiss, The IBM research parallel processor prototype (RP3): Introduction and architecture, *Proc. International Conference on Parallel Processing*, August 1985.
11. A. Agarwal, R. Bianchini, D. Chaiken, K. L. Johnson, D. Krauz, J. Kubiatowicz, B. Lim, K. Mackenzie, and D. Yeung. The MIT Alewife machine: Architecture and performance, *Proc. 22nd International Symposium on Computer Architecture (ISCA)*, June 1995.
12. J. D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Roseblum, and J. Henessy, The Stanford FLASH multiprocessor. *Proc. 21st International Symposium on Computer Architecture*, April 1994.
13. B. D. Fleisch, R. L. Hyde, and N. Christian, Mirage+: A kernel implementation of distributed shared memory for a network of personal computers, *Softw. Pract. Exper.*, 1994.
14. D. J. Scales and M. S. Lam, The design and evaluation of a shared object system for distributed memory machines, *Proc. First Symposium on Operating Systems Design and Implementation*, November 1994.
15. K. L. Johnson, M. Kaashoek, and D. Wallach, CRL: high-performance all-software distributed shared memory, *Proc. 15th ACM Symposium on Operating Systems Principles* (SOSP '95), 1995.
16. D. J. Scales, K. Gharachorloo, and A. Aggarwal, Fine-grain software distributed shared memory on SMP clusters, Research Report 97/3, February 1997.
17. D. Lenoski, J. Laudon, K. Garachorloo, W.-D. Weber, A. Gupta, J. Henessy, M. Horowitz, and M. S. Lam, The Stanford dash multiprocessor, *IEEE Comput.* **25** (3): 63–79, 1992.
18. H. Burkhardt III, S. Frank, B. Knobe, and J. Rothnie, Overview of the KSR1 computer system, Tech. Rep KSR-TR-9202001, Kendall Square Research, Boston, MA, February 1992.
19. A. Saulsbury and A. Nowatzyk, Simple COMA on S3.MP, *Proc. 1995 International Symposium on Computer Architecture Shared Memory Workshop, Portofino, Italy*, June 1995.

20. B. Falsafi and D. A. Wood, Reactive NUMA: A design for unifying S-COMA and CC-NUMA, *Proc. 24th International Symposium on Computer Architecture (ISCA)*, 1997.
21. C. Kuo, J. Carter, R. Kumarkote, and M. Swanson, ASCOMA: An adaptive hybrid shared memory architecture, *Proc. International Conference on Parallel Processing (ICPP'98)*, August 1998.
22. S. Adve, V. Pai, and P. Ranganathan, Recent advances in memory consistency models for hardware shared-memory systems, *Proc. IEEE*, 1999.
23. D. Chaiken, J. Kubiatowicz, and A. Agarwal, LimitLESS directories: A scalable cache coherence scheme, *Proc. 4th International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.
24. D. Chaiken and A. Agarwal, Software-extended coherent shared memory: Performance and cost, *Proc. 21st Annual International Symposium on Computer Architecture*, April 1994.
25. IVY system. Available http://cne.gmu.edu/modules/dsm/red/ivy.html.
26. T. Seidmann, Distributed shared memory using the NET framework, *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03).*
27. M. C. Ng and W. F. Wong. Orion: An adaptive home-based software distributed shared memory system, *Proc. Seventh International Conference on Parallel and Distributed Systems (ICPADS'00)*, Iwate, Japan, July 4–7, 2000.
28. Z. Radovic and E. Hagersten, Removing the overhead from software-based shared memory, *Proc. 2001 ACM/IEEE Conference on Supercomputing*, Denver, CO.
29. A. Meyer and E. Cecchet, Stingray: Cone tracing using a software DSM for SCI clusters, *Proc. 2001 IEEE International Conference on Cluster Computing (CLUSTER'01).*
30. W. A. Wulf and S. A. McKee, Hitting the memory wall: Implications of the obvious, *Comput. Architecture News*, **23** (1): 20–24, 1995.
31. D. Kim, M. Chaudhuri, and M. Heinrich, Active memory techniques for ccNUME multiprocessors, *Proc. 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, April 22–26, 2003.
32. K. Hwang, H. Jin, E. Chow, C. Wang, and Z. Xu, Design SSI clusters with hierarchical checkpointing and single I/O space, *IEEE Concurrency*, 60–69, 1999.
33. Y. Jegou, Implementation of page management in Mome, a user-level DSM, *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)*, 2003
34. B. Verghese, S. Devine, A. Gupta, and M. Rosenblum, Operating system support for improving data locality on CC-NUMA computer servers, *Proc. 7th Symposium on Architectural Support for Programming Languages and Operating Systems (ASPOLS VII)*, 1996.

FURTHER READING

Message Passing Interface Forum, MPI: A Message Passing Interface Standard, May 1994.

Li Zhang
Manish Parashar
Rutgers, The State University of New Jersey
Piscataway, New Jersey

SOFTWARE ENGINEERING FOR TELECOMMUNICATIONS SYSTEMS

INTRODUCTION

Since the construction of the worldwide telephone network started more than a century ago, advances in communication systems and their widespread availability have been a source of profound change in societies and are an important part of what we call the "information society." Today, communication systems allow people to talk, see, and exchange data with each other almost independently of their physical location in the world. In the so-called developed countries, telephones are in virtually every household, cell phones are omnipresent, and half a billion computers are part of the Internet—together, these technologies form a gigantic network that allows anyone easy access to an enormous amount of information and to communicate easily with each other.

According to a broad definition, any system that makes communication over long distances (tele = distant) possible is a telecommunication system. Historically however, the term refers primarily to telephony networks for fixed and mobile communication. For the Internet and other interconnections of computers, the term computer network is used. The term communication system can refer to both computer networks and telecommunication systems. Because of the convergence of telecommunication and Internet technology, the once sharp line between computer networks and telecommunication systems has, however, been blurred in recent years. End users engaged in distant communication are indifferent as to whether their voice is transported via traditional telecommunication networks or over the Internet—as long as the service preserves high-quality demands expected from telecommunication systems: lost calls, too much delay in voice transmission, echoes, and so on are not tolerated.

Although from today's point of view many commonalities seem to exist between the two, the telephone system and the Internet were historically created with different aims and design philosophies in mind and are based on different technologies. The telephone system was designed primarily for voice communication between humans; even data services like transmission of facsimiles (fax) used tone modulation techniques over the voice channel. The telephone system uses circuit and packet switching to establish a dedicated connection with guaranteed quality of service (QoS) for the duration of a call. The Internet, on the other hand, was designed for flexible data exchange between computers with the capability to compensate single points of failure—it was intended as a network for military purposes in the first place. The Internet uses packet switching for efficient communications with best effort QoS.

Many definitions for software engineering exist in the literature. One definition that we feel is particularly suitable in the context of telecommunication systems is "the application of engineering to software" (1). In fact, (electrical) engineers built the first telecommunication systems. The key challenges for software engineering telecommunication systems developed from several aspects that make telecommunication systems special. These aspects relate to the system in general and the software components in particular and can be subdivided into industry practices, general technical challenges, and quality demands. Important industry practices in the telecommunications domain include the definitions of standards and protocols, the layering of systems, and intensive testing. General technical challenges develop mainly from the distribution aspect of telecommunication systems and the large amount of communication that these systems need to handle simultaneously in real time. For example, modern switching systems can handle several ten thousands of calls simultaneously. Thereby, high-quality demands must be fulfilled.

From a user's perspective, telecommunication systems must provide a high quality of service, namely the fulfillment of real-time and lossless requirements. In addition to that, they must satisfy high availability, reliability, and robustness demands. For example, delays greater than a tenth of a second or lost words in a telephone conversation are unacceptable; the expectation of users is that the telephone system "always works" and that especially emergency calls always go through, no matter the amount of traffic. From the perspective of software engineers, telecommunication systems must be scalable, extensible, and portable. Scalability means that the code basis can be used for different traffic demands. For example, it is desirable that the same switching software can be used in a system that handles an average of 1000 simultaneous calls as well as in a system that handles 100,000 calls simultaneously. Scalability is especially important as the demand for telecommunication services is steadily increasing. Extensibility is important because the services that must be provided by telecommunication systems are constantly subject to enhancements. Portability has to do with the long lifetime of telecommunication software and the rapid advances in hardware platforms; one cannot afford to throw away the software developed for two decades just because of a switch to new hardware.

The next section provides an overview of relevant historic developments and crucial design decisions that led to today's telecommunication systems; important background information for understanding what makes software engineering for telecommunication systems special is given. The section entitled System Design in the Large describes fundamental telecommunication concepts that are encountered when designing systems in the large. The basic notions of distribution and communication, as well as layering, planes and resource control are discussed. Systems Design in the Small is the topic of the section that follows. The last section surveys a selection of the literature on modeling telecommunication systems and introduces

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

the Real-Time Object-Oriented Modeling (ROOM) language as an example of a modeling language for telecommunication systems.

BACKGROUND AND RATIONALE

Modern telecommunication systems were developed in the early 1960s. The new technology of computer control, called stored program control (SPC), started to substitute electromechanical systems (2). One of the main advantages introducing SPC was flexible systems, in which additions and changes could be introduced primarily through program modifications rather than through changes in the hardware (3). However, by the late 1960s, it was time for a review. At Ericsson, one had learned that the current generation of SPC, as it existed in the late 1960s, was expensive and way too complex, with hindsight, for widespread use, except, to some extent, in the American Bell companies. The disadvantages were above all in the high costs of handling—design, testing, modification, fault-correction, production, installation, and operation and maintenance (4). What was needed was a new approach to structure and organize these complex systems. With the engineering techniques available at that time—“Structured Programming” is in the air (5), the principle of functional modularity was a promising approach. Within Ericsson, it was Ivar Jacobson who made the important contribution of the “block concept” in 1967 (6), which included the structuring of the system into self-contained functional modules (blocks), with all interworking between blocks performed by software signals (7). The development of Ericsson’s AXE switching system was based on these principles; it went into trial service late in 1976 and became and still is one of the most successful switching systems worldwide (4).

Hand in hand with this development, the study of new languages was initiated. The industry was in need of languages highly adapted to the demands of programming and designing telecommunication systems. The outcome of these efforts were Specification and Description Language (8) (SDL), Message Sequence Chart (9) (MSC), CHILL CCITT High Level Language (CHILL) (10), and Man-Machine Language (MML) (11). All three languages have been standardized by Consultatif International de Télégraphique et Téléphonique) (CCITT) and are still in use today. In the early 1980s, SDL and MSCs were intended for system specification and design, CHILL for detailed design, coding and testing, MML primarily for operation and maintenance. Especially for coding, many companies developed their own variant of a programming language. For example, Ericsson developed the Programming Language for EXchanges (PLEX) (7), and Northern Telecom developed the Procedure Oriented Type Enforcing Language (PROTEL) (12); both languages are block structured. More recently, new languages and paradigms have become part of the toolset of software engineers in the telecommunication domain. An example of a modern programming language for telecommunication systems is Erlang (13); Erlang can be classified as a functional programming language. It was developed at the Ericsson Computer Science Laboratory in the late 1980s and was released as open source in 1998. It has been used in industrial projects for the production of highly reliable and fault-tolerant telecommunication systems. For example, Ericsson’s AXD301 switching system handles 30–40 million calls per week and node, and its reliability is measured at 31 milliseconds downtime per year. It contains 1.7 million lines of Erlang code (14).

In 1994, the ROOM language appeared. ROOM blends object-oriented and real-time concepts and techniques and is thus particularly well suited for modeling telecommunication systems. Elements of ROOM were added to the Unified Modeling Language (UML) (15,16) version 2.0 that was released in 2004. In 2006, the Object Management Group (OMG) released the specification for the Systems Modeling Language (SysML) (17), which is a modeling language for systems engineering that seems to be a promising addition to the toolset of software engineers in the telecommunications domain. A good chance exists that telecommunication systems engineering might benefit from the recent research and commercial interest in generative (18) and model-driven development (19). Domain-specific notations have been used by telecommunication engineers for a long time, and new technologies might enable the generation of systems based on descriptions using these notations.

The complexity of switching systems by sheer size of code is impressive. Already around 1980, several hundred programmers had produced over one million lines of code over a five-year period for the DMS-100 switching system family of Northern Telecom. The source code was organized into 15,000 procedures in 1500 modules (12). The systems of today are even more complex. A code base of several million lines of code is not unusual. Still, these systems fulfill high-quality demands on availability, reliability, fault tolerance, and so on. Such systems can be upgraded and maintained while being in operation! A downtime of some few minutes per year is already perceived as “bad quality.”

Considering their complexity, it may come as no surprise that *architecture* is and always has been an important issue in telecommunication systems design. Architecture is and was a means to deal with complexity. Of course, the term “architecture” was not defined clearly, but it is absolutely in line with the design paradigm of the 1970s: The modularization of a system is regarded as its architecture. Architectures were not modeled, as we tend to say today, but rather described either informally, usually in some sort of box-line diagrams, or formally with SDL. It is interesting to read which design conceptions were identified for new software architectures in the 1980s: independent subsystems for call control (features), signaling, and hardware control; data abstractions partitioned for each subsystem; formal communication protocols; concurrent and asynchronous operation of each subsystem; terminal-oriented control; layered virtual machines; finite state machine specifications; application programs; and systems programs (20)—the topicality of the list is astonishing.

Before “software engineering” was an established field, telecommunication engineers had already established a discipline of engineering highly reliable, scalable, and robust real-time systems, which are open and standardized—and it included software. When the telecommunication engineers included programmable devices into their systems, they integrated these devices into a hardware-driven environment. Thus, they applied a lot of their hardware principles to software. In effect, they made it transparent to the system, whether an entity is realized in hardware or software. Simply speaking, the software was and still is developed as seriously and effortful like hardware. Because failures and downtimes of telecommunication components are not an option, much energy is put into the design and architecture of those systems. The engineering aspect of the software part of telecommunication systems resembles many qualities of systems engineering: standardized interfaces, message-orientation, cascading, and composition as main design principles; exhaustive testing routines including load and stress testing, configuration management, process driven development—to name just a few—are best practices in the telecommunication domain.

The concepts, techniques, and requirements we describe in the historic overview above are still relevant to software engineering for telecommunication systems today. It is the way telecommunication engineers design their systems in the large and in the small that is special. That is why we put our focus on these two topics in the following sections. Other software engineering issues like requirements engineering and traceability, configuration and product management, software product lines and families, testing, project management and so on do not differ that much from software development practices in other domains such as large enterprise information systems.

Regarding software engineering for telecommunication systems, no established body of literature exists yet, which reflects a commonly agreed viewpoint on how telecommunication systems are (to be) designed in the large and in the small. However, if you spend some years in the telecommunication industry among systems designers and software developers and study existing publications, then you will notice that they somehow speak "one language" and design their software in similar ways. This article is an attempt to uncover the elements of design of telecommunication system engineers to provide valuable input for the interested reader. A more elaborated version of the systematics presented here can be found in Ref. 21.

SYSTEMS DESIGN IN THE LARGE

In telecommunications, systems design in the large must deal with the notion of distribution, layering, planes, and resource control. We will discuss each issue in turn.

Distribution

A telecommunication system is made up of entities like switching systems, radio base stations, and mobile phones. These entities are physically distributed in space; they are either located in a fixed place (like switching systems) or are mobile (like mobile phones). These entities collaborate with each other to provide a service to end users. Thus, the most obvious characteristic of a communication system is its aspect of distribution. If two or more devices, processes, users or—more abstractly—entities are physically spread in space but want to collaborate, they somehow have to bridge spatial distribution and establish communication. We will give rather informal definitions of the concepts related to distribution in the following sections. Formal definitions of these concepts can be found in Ref. 22.

Communication. "It is all about communication"—this slogan characterizes concisely the motto of telecommunications. We can classify three types of communication used in telecommunications. The classification scheme bases on the question "Who controls whom?" We can distinguish three basic combinations of the exertion of control between two communicating parties: (*1*) no side exerts control, that is no side has a state model of the other side to influence the other side's behavior in a controlled way, which we call *data-oriented* communication; (*2*) only one side exerts control, which we call *control-oriented* communication; (*3*) both sides exert control, which we call *protocol-oriented* communication.

Any communication type can be realized in a connection-oriented mode, a connectionless mode, or even other kinds of communications styles. We will come back to this in the discussion of communication services. Note that communication in telecommunications is message-oriented and that communication relations are strictly specified in form of protocols.

Decomposition and Remote Communication. What is distribution? With the eyes of software engineers, we tackle the notion of distribution in two steps: First, distribution is an issue of logical *decomposition*. Second, we need to consider the effects of *remote communication*.

We can view a telecommunication system as a logical entity that encapsulates some functionality and offers interfaces (often called "ports") for message-based communication with the environment. We assume that the behavior of the system is given, meaning that we know the set of allowed messages per interface, their format, how the system reacts on messages delivered to the interfaces and which messages it emits to the environment. The behavior of a telecommunication system is often said to provide services to its environment, usually its end users.

A first step toward distribution is that the entity under consideration can be logically split up ("decomposed") into separate parts, each part representing a new entity. The parts also communicate to each other via messages through their interfaces. The interfaces are connected via so-called channels, which are sometimes also called connectors. A channel is an idealized communication medium, which transfers messages faultless and in an instant. In other words, a logical entity gets refined by a network of separated but cooperating parts. From an outer perspective, the conglomerate of parts preserves the behavior that can be experienced at the interfaces of the single entity. The decomposition process is recursive.

The second step is to take into account that the communication over a channel is not ideal but suffers from the real-world effects of remote communication. When the decomposed parts get spread over, say, hosts or physical nodes, they require some sort of communication means to bridge the spatial separation. The interaction of the decomposed parts in a distribution network is *not* fault-free per se; it is sensitive to disturbances on the communication medium and dependent on the properties of the connection. We condense the whole communication medium in a model of a nonideal channel, which we call complex connector. The complex connector is a component that represents the properties of the communication channel and its effects on the transmission of messages. These properties are called QoS attributes and include all relevant characteristics, such as reliability, throughput, jitter, and delay (21,22).

To summarize: A telecommunication system is a distributed system. To its end users, a telecommunication system appears as a single, coherent, service-provisioning system. As a matter of fact, the system is decomposed into a number of physically separated but interacting parts, called nodes, which constitute a communication network. The effects of remote communication are captured by the notion of a complex connector.

Network Topology and Communication Services. Readers might be familiar with the fact that communication systems are composed of a stack of layers. We will come back to layering in a subsequent section entitled Layering. In this section we view each layer in a communication system as a self-contained unit without any dependencies to other layers. Each layer unit consists of distributed entities communicating remotely to each other via a network that interconnects the entities. In short, we treat a layer as a distributed network in its own right.

Here, we are concerned with what kind of communication services and communication resources the distributed entities use to bridge their spatial distance; for the time being we are not interested in how it is achieved via a lower layer. That means our understanding of a network is an abstract model of distribution, which includes a *network topology* (who is permitted to communicate with whom) and the used *communication services*. A communication service can offer connection-oriented or connectionless communication means.

Connection-Oriented Communication Services. With the help of the complex connector, we can describe *static* configurations of distant connection-oriented communication. The complex connector concentrates all impacts that the transmission may have on the messages to be conveyed. In reality, connections are rarely static; they are rather a form of a long-lasting, dynamically created connection. Normally, connections are set-up and released on demand. Thus, we need something; we can ask for inserting and removing a complex connector between any two ports at some point in time. The connection-oriented communication service fulfills this role. In a telecommunication system, circuit switching is a connection-oriented communication services.

Connectionless Communication Services. A style of communication exists that requires no connection. Instead, messages include the address of the receiver. The sender hands the message over to a connectionless communication service, which distributes the message according to the address to a destination. If the sender wants to get a response on a delivered message from the receiver, then the sender has to include its source address in the message as well. In a telecommunication system, packet switching is a connectionless communication service.

Addressing. Addressing is crucial for communication systems in general and telecommunication systems in particular. In the following, we focus on addressing in the context of telecommunication systems. Generally speaking, an *address* denotes a concept to identify and locate objects in a defined scope. The scope is the so-called *address space* (or name space), which is an assembly of addresses with each address being unique in the assembly. An *address association* relates two addresses to each other; the association is directed pointing from one address (the source address) to another address (the destination address). Source and destination address may or may not belong to the same address spaces; we must make a difference between external address associations and internal address associations. External address associations relate addresses of different address spaces, internal address associations relate addresses of the same address space.

For example, the difference between connection-oriented communication and connectionless communication is basically different ways of working with address spaces. In connection-oriented communication, communication interfaces are associated with a fixed (better: temporarily fixed) communication partner. Information that is pushed to the interface is conveyed to the communication partner; reversely, information the communication partner wants us to notice, pops up at the interface. In that sense, the interface is a sort of representation of the other party, and the interface identifier is an internal address denoting the other party. So, to talk to another party it is necessary to either use another interface (that is bound to the other party) or to newly bind the interface with the other communication party.

For connectionless communication, the general addressing structure looks different. The arrangement of associations is so that two communication partners do not maintain direct relations between their address spaces. Instead, local addresses are associated to a third party, which is an external address space. Consequently, users who communicate connectionless need to have an internal representation of the address space outside their locally addressable scope. They need to specify the destination of their messages. Users who communicate connection-oriented do not have to do that.

Remarks. Definitions on distribution to be found in literature suffer preciseness on the one hand and generality on the other hand. We think that at an abstract-level distribution is primarily a logical conception and that it is adequate to give a formal definition based on a proper model. Secondary, distribution has a technical dimension. A formal definition of distribution is given in Ref. 22.

No notion exists of a complex connector in Open Systems Interconnection (OSI), but in practice, telecommunication engineers work with this concept. As an evidence for that statement, have a look at channel substructures in SDL (23, p.121), which basically captures the same intention as complex connectors.

Addressing is a delicate issue in modeling and an neglected issue in software engineering.

Layering

Layering is one of the oldest techniques in software engineering to structure a system. Possibly the first, who made systematically use of layering was Dijkstra, he used layering for the design of the THE operating system (24). Layering is also a key structuring principle in the design of communications systems, be they telecommunications or computer networks. In the previous section, we intentionally left out the issue of layering. We simply said that one can look at each layer of a distributed communication system individually. Now it is time to explain, how several layers of communication networks are interconnected and make up a layered system.

In the next section, we will briefly outline the seven layer reference model (RM) of open systems interconnection(25) (OSI). We will then distill the key idea that underlies layering. This progression will naturally lead us to two viewpoints one can have on a communication network: a network-centric or a node-centric perspective. Finally, we discuss the concept of planes.

The OSI Reference Model. Layering is a means to stepwise provide higher-level services to a user or the next "upper" layer, and to separate levels of services by precisely defined interfaces. This overall design principle is reflected by the use of protocol stacks. The OSI RM is the most prominent framework for a layered communication architecture. We do not repeat OSI RM to the full extent, we just would like to remind the reader of the basic outlook, see Fig. 1: Several network layers are stacked on each other, each layer realizing a complete network of its own. Higher-layer network services rely on lower layer services until a physical layer is reached. Additional introductory information can be either retrieved from the X-Series of the ITU-T recommendations or from textbooks. Almost any textbook on computer networks and/or data communications gives an introduction into OSI RM, for example Ref. 26.

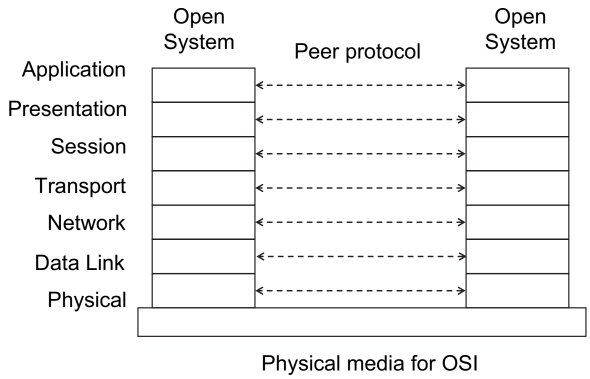


Figure 1. OSI seven layer reference model; see Ref. 25, p. 31.

We wish to highlight one important point. OSI RM clearly distinguishes two communication relations: layer-to-layer ("vertical") communication from peer-to-peer ("horizontal") communication. "Vertical" communication refers to the exchange of information between layers (that is levels of services usually within the same physical entity) in the form of *Service Data Units* (SDU). "Horizontal" communication refers to the exchange of information between remote peers. Remote peers are physically distributed and communicate with each other according to a protocol in the form of protocol messages, also called *Protocol Data Units* (PDU), thereby sharing the same level of protocol conventions. PDUs are the vehicles for SDUs. A single SDU may be packaged into one or more PDUs. Such PDUs are also called *data* PDUs. Nondata PDUs are called *control* PDUs. In a multilayer communication architecture, a service provisioning layer becomes the service user of the next lower layer.

Communication Refinement. To understand how two different networks of service levels are connected through layering, one has to know that Fig. 1 unveils only half the truth. The dotted lines labeled with "Peer protocol" do not represent protocol relations only. Each double-headed arrow "hides" a complete infrastructure of a communication service for this specific layer. The communication service per layer is an abstract model of the style of communication (connection-oriented, connectionless), properties (delay, reliability, etc.), topology, and addressing schema. This abstract model can be refined into a more concrete model, which is in effect the next lower layer of the protocol stack. The next lower layer includes the communicating entities (the boxes next to the double-headed arrow) and the communication service of that layer. In essence, layering is the result of refining communication services; we call this *communication refinement*.

Communication refinement leads to two different viewpoints on distributed layered communication systems. Both viewpoints are important for systems modeling in software.

Network-Centric Viewpoint. If we regard the communication service as an abstract model of the means of communications, suppressing all the details of lower layers, then we just observe a network of communicating entities of one layer using the communication service. This view is the network-centric viewpoint on communication systems. This view allows one to look onto a network as a distributed system ignoring layering. We made use of this technique in the section about distribution.

Node-Centric Viewpoint. If all communication services are resolved by a refinement, which represents the next lower layer, we end up with a situation similar to Fig. 1: We have a communication service at the very bottom, a physical media, which cannot be resolved more. The pile of boxes labeled "Open System" on the left and on the right represent the entities that, together, make up the software or hardware, which resides on a physical node in a network. This view is the node-centric viewpoint on communication systems.

Planes

One concept that has turned out to be extremely useful is the concept of *planes*. The concept was introduced in Integrated Services Digital Network (ISDN) (27), taken over in Global System for Mobile communication (28), and currently shapes the network architecture of Universal Mobile Telecommunication System (UMTS) (29). The distinction is usually in three planes, namely the control plane, the user plane, and the management plane.

A plane encapsulates service functionality and may have internally a layered (protocol) structure. Planes are an organizational means on top of layering and communication refinement, respectively. In telecommunications, the user plane provides for user information flow transfer (data PDUs), along with associated controls (e.g., flow control, recovery from errors); the control plane performs call and connection control functions (control PDUs), dealing with the necessary signaling to set up, supervise, and release calls and connections; the management plane takes care of (*1*) plane management functions related to the system as a whole including plane coordination and (*2*) functions related to resources and parameters residing in the layers of the control and/or user plane (30).

OSI RM is not prepared to handle planes (nor is the Internet architecture), which is also one of its major deficiencies. The control and user plane are not separated. In software engineering, the organization of a system in planes is almost unknown. On a case-by-case basis, designers had and still have to invent individual solutions to handle planes in their models. For example, in ISDN the engineers introduced a synchronization and coordination function (SCF) as a major component of the management plane. The SCF is connected to the highest layer of the user plane and to the highest layer of the control plane to coordinate and synchronize the required collaboration of planes (27).

Resource Control

A field that is largely ignored in computer networks but is of importance in telecommunications is the issue of resource control—most popular textbooks on computer networks and distributed systems do not touch on the subject at all. The term *resource* does not only include physical resources such as adaptors, switchboards, echo cancellers, codec converters, and so on, but also resources implemented in software. On a software level, resources can be combined, added by some functionality, and offer value added services that make a user believe to access a “new” kind of resource that is more than the sum of its physical components. Take for example an alarm clock and a radio, add a composing layer, and you will get a clock radio. The new feature, that the radio turns on at a certain alarm time, is more than any of the resources could provide in isolation.

We recognize a need to pay some special attention to resource control. As was mentioned previously, telecommunication systems are sliced in a control and a user plane; basically, it is the control plane that controls the user plane. In most cases this control relationship breaks down to resource control. The control plane controls resources of the user plane. Although the control plane and the user plane may operate as largely independent networks, the combining spots are locations of resource control. Usually, the node hosting the resource brings together the control and the user plane. Traditionally, the aspect of resource control has been a local, internal issue. Often, inside such a node, the border between controlling and controlled behavior is blurred and not fully separable. At best, the designers define a proprietary application programming interface (API) for the resource.

One of the intentions of UMTS has been to clearly separate the control and the user plane and to avoid the blur of the control/user plane inside nodes hosting resources; this idea is the so-called *architectural split* introduced with UMTS. As a result of that, the telecommunication sector of the International Telecommunication Union (ITU-T) defined a protocol, a control-oriented protocol in our terminology, that describes how a user can control a switching center. This protocol is called media gateway control protocol, it is specified in H.248 (31) and has been taken over as a standard by IETF as well, see RFC 3015 (32). With the definition of a protocol and the separation in a resource user and a resource provider all prerequisites are given to aim for physical separation of both roles. In the UMTS architecture, these two roles are logically fulfilled by the media gateway controller and the media gateway. It is up to a manufacturer to produce two individual nodes or a single combined node. Important is that the distinction has been made logically.

Remarks

We mentioned the OSI RM. The reference model of the Internet Architecture, is related loosely to OSI RM, see Ref. 26. Other frameworks exist that address the topic of distributed communication system and propose a terminology, a set of conceptions, and a system architecture organization. The most important frameworks to mention are the Reference Model for Open Distributed Processing (33,34) (RM-ODP), the Telecommunications Information Networking Architecture (35) (TINA), and the object management architecture (36,37), (OMA), which is the basis for the Common Object Request Broker Architecture (38,39) (CORBA). Basically, all three frameworks specify an environment to develop, install, and maintain distributed applications.

SYSTEMS DESIGN IN THE SMALL

When it comes to systems design in the small, the most apparent issue a software engineer is confronted with is that telecommunication systems are real-time systems. An understanding of real-time systems and a suitable approach for designing such systems is required. The use and the understanding of the term “real-time system” is not consistent in the literature. It is a mixture of characterizing attributes and structural properties of a system. For example: On one hand, it is said that a real-time system fulfills timing constraints, (i.e., a real-time system has to react to a stimulus in a certain time frame); in this example the guaranteed response time is an attribute, which characterizes a real-time system. On the other hand, real-time systems are often classified as “embedded systems.” An embedded system can be seen as a specific part of a larger system, which is a structural aspect. Besides this lack of clarity in terminology, there is not even common agreement on the word “real-time.” The following paragraphs summarize findings from studying the literature.

What is a Real-Time System?

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time in which the results are produced (40). After more than a decade this definition still seems to be the greatest common denominator. Here, “real-time” is an attribute to “system.” Because of their specific field of application, additional attributes are usually associated with real-time systems. Included in this category are e.g., reliability, fault tolerance, adaptability, and speed (41).

Hard versus Soft Real-Time

The most popular classification is the distinction in *hard* and *soft* real-time systems. Hard real-time systems are under deadline constraints. Passing a deadline is considered unacceptable. A soft real-time system retains some tasks that are still valuable for execution even if they miss their deadlines (41). Telephony systems belong to the class of soft real-time systems (42): Passing of deadlines is accepted as long as the number of failures is below a defined threshold. Although this categorization might be true in general, some components in telecommunication networks, have to fulfill hard real-time constraints. For example, the time delay perceived as acceptable for voice transmission in a speech conversation places tough time limitations on a mobile phone for speech encoding- and decoding including cyphering and channel coding.

Rough Structure

A very rudimentary structure of the basic elements of a real-time system is given by Ref. 42: It consists of hardware, *sensors* and *effectors*, the environment, and software. The sensors and effectors interact with the environment; the software controls the actions of the hardware via a hardware interface. A similar description using different terminology can be found in Ref. 43: A real-time system consists of a controlling and a controlled system. The controlling system interacts with its environment using information about the environment available from various sensors and activating elements in the environment through “actuators.” The controlled system can be viewed as the environment with which the computer interacts.

A loose reasoning describes why timing aspects and structural issues of a real-time system are related: Timing correctness requirements arise because of the physical impact of the controlling systems’ activities on its environment. That means that the environment needs to be monitored periodically and sensed information needs to be processed in time (41). This finding implies that we have to distinguish the environment from a controlling part, and detecting and acting devices are needed.

What is an Embedded System?

The definition of an embedded system is vague; it mainly describes a structural aspect. In its most general form, an embedded system is simply a computer system hidden in a technical product (44). A more concrete definition is that most embedded systems consist of a small microcontroller, and limited software situated within (e.g., an automobile or a video recorder) (45). Three issues seem to be important here: (*1*) size matters, (*2*) an embedded system is part of a technical system, and (*3*) it serves the purpose of the technical system and not vice versa. Issue (*3*) especially helps delimitate nonembedded systems from embedded systems. A Personal Computer (PC) for instance is a general purpose computing machine, the software and the central processing unit (CPU) are an integral part of it. This eliminates a PC from being an embedded system. A counterexample might be a mobile phone. The digital signal processing chip and its software serve a single purpose: to offer phone functionality. Embedded systems may or may not have real-time constraints (43), but many real-time systems are embedded systems (45).

To summarize: The special character of systems, that have a physical impact on the “real” world by means of reactiveness is most significantly described by the requirement on the timing constraints to be met by the system. Such systems are called *real-time systems*. Additional properties, which reflect other aspects of the physical impact character, include reliability, fault tolerance, stability, safety and so on. As yet no commonly agreed list of properties exists that constitutes a real-time system. Moreover, the physical impact nature of such systems implies a rough structure: a controlling part interacting with the environment (the controlled part) through sensors and effectors. The hardware mediates between the sensors/effectors and the software of the system. Many real-time systems are embedded systems, which means they serve a specific purpose in a technical system, which is actually the case for all nodes in a telecommunication system.

Despite these various aspects of real-time systems and partly confusing definitions from the literature, designing real-time systems is a well-established domain. When designing telecommunication systems in the small, it becomes obvious that just a few key design concepts are required, such as active objects for modeling threads and message-orientation for modeling asynchronous communication. Interestingly, these concepts can also be used for systems design in the large. This statement means that one can use the same language for both systems design in the small and in the large. In the section on Real-Time Object-Oriented Modeling, we describe a language that can be used for both tasks.

MODELING TELECOMMUNICATION SYSTEMS

In this section, we describe different modeling approaches by surveying the available literature. We then go on to describe ROOM, which is a language in widespread use in the telecommunications domain. ROOM can be used for both systems design in the large and in the small.

Modeling Approaches

Since the UML has been standardized by the Object Management Group (OMG) and published in many books, *modeling* is on everybody's lips. Also, the importance of the architecture level in software systems is more and more respected, see for example OMG's initiative on Model Driven Architecture (46). However, when it comes to modeling telecommunication systems the fundus of literature is even smaller.

In some books, the object-oriented paradigm has been used to model communication systems. One example is "Object-Oriented Networks: Models for Architecture, Operations, and Management" (47). The book uses not only conventional object-oriented modeling concepts but also advanced concepts from specialization theory. The syntax used to capture the semantics of models is the Abstract Syntax Notation One (see Ref. 48). The author develops a classification scheme adapted to the needs of communication networks that enables a designer to develop understandable and meaningful object and class diagrams. The approach is descriptive and the techniques presented seem to be suited for modeling product architectures. The risk is that given "facts" are just schematically modeled (it is relatively easy to note down an object diagram for almost anything) without any reflection about the actual functioning and the actual meaning for the architecture.

Another example is "Object-Oriented Network Protocols" (49). The book's intention is to provide a foundation for the object-oriented design and implementation of network communication protocols. Although modeling of communication systems is not the topic of the book, it is worth to have a look at the modular communication system framework developed by the author. It gives an insight how protocols could be modeled and that object-orientation is a practical approach in protocol design.

A completely different approach is taken by "Modeling Telecom Networks and Systems Architecture: Conceptual Tools and Formal Methods" (50). This book condenses more than 20 years of experience gained on the subject within Ericsson. It presents a method and a language for modeling telecommunication system and is based on the processing system paradigm (51). The whole field of communication systems is covered, and a stringent methodology and classification scheme is discussed. The interested reader might also look at Ref. 52.

Real-Time Object-Oriented Modeling

Subsequently, we briefly present the ROOM language to give the reader a notion of what kind of concepts software engineers in the telecommunication system domain work with. Even though the publication of ROOM dates back to 1994, it is still modern and a rare example of a well-documented design language, see Ref. 42. Many features of the ROOM language have been incorporated into the UML (15,16). Nonetheless, we have chosen to describe ROOM, because it represents a coherent set of features required for designing (embedded) real-time systems in the telecommunication domain; the UML is just a rich set of modeling concepts a designer can choose from. In this section we will briefly discuss structural elements of ROOM, behavioral elements and mention model execution.

Structural Elements. ***Actor, Port, Message, Protocol.*** The ROOM language is built on the notion of an *actor*. An actor represents a physical device or a software unit; it is a sort of active object that clearly separates its internals from the environment. Everything inside the actor, meaning the actor's structure and behavior, is not visible to the environment. Only at distinct points of interaction, so-called *ports*, the actor interfaces the environment. A port is somewhat comparable to an interface as known for example, in the UML but the comparison blurs two important facts. First, ports in ROOM are not method interfaces but message interfaces. A *message* consists of a message name, priority, and data. Messages may be incoming and/or outgoing at a port (the direction is always defined from the viewpoint of the actor). So, ports are message exchange points between the actor and its environment. Secondly, a port is not only an interface that tells the environment how to use the actor but also is a definition of the actor's expectations on the environment. Therefore, a *protocol* is always associated with a port, which defines the set of incoming and outgoing messages that may pass the port. An actor is specified by means of an *actor class*. An actor class is symbolized by a rectangular box with a thick black border. A port is figured by a small squared box that appears on the border of an actor class symbol. An example is shown in Fig. 2.

Actor References. An actor can be composed of other actors. In ROOM, references describe compositions. That means, an actor class specification may reference zero or more other actor class specifications. Such a reference is called *actor reference*; it is a way to include other actors into the name space and life-time context of an actor. Per actor reference, a *replication factor* determines the maximum number of valid actors of the referenced actor class that can be put in context. By default, the replication factor is set to one. The following types of references can be distinguished: an actor reference may be fixed, optional, imported or substitutable. These types specify run-time relations. For a *fixed* actor reference, actors of the referenced actor class are incarnated along with the incarnation of the composing actor. If the actor reference is declared as *optional*, then actors of the referenced actor class can be

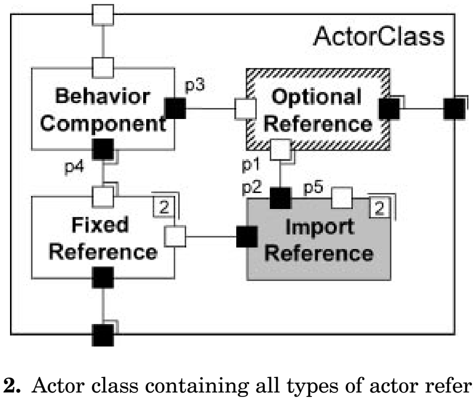


Figure 2. Actor class containing all types of actor references.

dynamically created and destroyed during the life time of the composing actor. The maximum number of allowed actors (given by the replication factor of the actor reference) may not be exceeded. If declared as *imported*, then an actor that already exists in another context of another composing actor is plugged-in at incarnation of the composing actor. That means a single actor instance may act in two or more contexts of a composing actor: in the context of the "original" composing actor that created the actor and owns the permission to destroy it and in the context of one or more other composing actors which imported that specific actor. Imported actor references are a powerful tool to define different roles for different contexts of an actor and thereby to define patterns of collaboration. A *substitutable* actor reference means that any actor instance of that actor reference can be replaced by another actor, provided that the other actor's class specification is compatible with the referenced actor class of the actor reference. Here, compatibility means that the other class specification supports at least the same set of ports (with the same message schema).

Binding, Contract. To build up complete structures of actor references, some means to interconnect their ports must exist. This connection is done by so-called *bindings*, sometimes also referred to as *connectors*. A binding connects a port of an actor reference either with the port of another actor reference or with a port of the composing actor class. Bindings define communication relationships on class level. The auxiliary concept of a *contract* consists of a binding and the two interface components (ports) that the binding connects.

Example. An example of an actor class specification that encompasses all the discussed modifications of an actor reference is shown in Fig. 2. Actor references are symbolized by a rectangular box with a thinner black border and can only appear "inside" the context (also called *decomposition frame*) of an actor class specification. Names for actor references begin with a small letter. Names for bindings begin with a small letter by convention. Sometimes, to avoid visual clutter, the names of bindings and ports are not displayed in the diagram. The replication factor of a replicated actor reference is displayed inside a box in the upper right-hand corner. Optionality is indicated by stripes. If imported, the actor reference is colored grey. Substitutability is indicated by a "+" symbol in the upper left-hand corner.

The Behavior Component. A component specifies the actor class' behavior. In fact, the behavior component is invisible; the behavior component's border is colored in grey just for demonstration purposes. Thus, all ports of an actor class specification that are not connected somewhere else are actually connected to the actor's behavior component; they are called *end ports*. Otherwise, they are called *relay ports*. All other ports (p3, p4) that "hang around" are also implicitly connected to the behavior component. *Reference ports* (the name for ports of actor references) that are not involved in a contract are actually not in use, see p5.

Layer Connection, Service Provision Point, Service Access Point. The notion of layers is a built-in concept in ROOM. Layering is a form of abstraction that is used to define "islands" of self-contained functionality that provide services to another "island" of functionality. In contrast to the *horizontal* structure of peer-to-peer communication between ports, layers represent a *vertical* organization of a system. The terms "horizontal" and "vertical" are apparently vague and indicate the difficulty for giving a precise definition of layers. Actually, the sort of interfaces used to describe layers are very close to ports. The interface of an actor that provides (layer) *services* towards another actor is called service provision point (SPP). The SPP may be replicated; the number of replications is given by a replication factor. Its counterpart, the interface that accesses services of an SPP is called service access point (SAP). SAPs can be replicated as well but they rarely need to. The SPP and the SAP each have a protocol associated with it that determines the interface type. Similar to a binding, a SPP and a SAP are connected to each other by a *layer connection*.

Behavioral Elements. *ROOMcharts, Scheduler.* We already mentioned the behavior component of an actor. In ROOM, behavior is specified in form of state machines, so-called *ROOMcharts*, a variant of Harel's statechart formalism (53). Actors in ROOM are reactive objects with their own thread of execution, which is a typical characteristic for real-time systems. All incoming messages at the behavior component are *events* that may trigger a *transition* to leave a *state*, to perform some *action* and enter the same or another state. For a state, entry and exit actions can be specified. Actions are specified in a detail level language such as C, C++, or Java. A *guard* (a boolean condition) can be attached to a transition, which prevents the transition from firing if the condition evaluates to false. The concept of *composite states* enables the modeler to nest states within states. Once all actions have been executed (ROOM follows the "run-to-completion" processing model), the actor "falls asleep" waiting for additional events to process. Because incoming events are queued, the actor may immediately become busy again until the event queue is empty. Events can also be deferred (i.e., the processing is postponed). Message priorities change the order of event processing usually to "the more important, the more up front in the event queue." In principle, the scheduling semantics of the *scheduler* can be adapted to any other scheme. ROOM is flexible in that respect to cover a wide range of real-time applications. For example, time-based scheduling ("the more urgent, the more up front in the queue") may be an alternative.

Data Classes. Complex data structures can be modeled using the concept of *data classes*. Data classes correspond to traditional classes: they define data and methods that operate on them. In contrast to actors, data objects do not have their own thread of control; they are extended state variables that are encapsulated within the actor and are accessible by the behavior component. Typically, data classes are based on classes provided by the detail-level programming language. That means within an actor the modeler can use and stick to a traditional object-oriented design paradigm. In addition to their role as variables, data classes are used to define the data carried in messages. Remember that a message consists of a name, a priority, and data, or more precisely, a data object. This single data object is an instance of a predefined or user defined data class. The basic requirement put on data objects is that they must be serializable for message transfer by the ROOM virtual machine.

Model Execution. In principle, two possible methods exist to execute ROOM models: (*1*) the model is accompanied by an interpreter called the *ROOM virtual machine*, which is a hypothetical platform implemented in software that interprets ROOM models; and (*2*) the elements of the model are mapped to their functional equivalents in the target environment, which usually is a real-time operating system.

BIBLIOGRAPHY

1. IEEE Standard Glossary of Software Engineering Terminology. Standard 610. 12-1990, Piscataway, NJ: IEEE Standards, 1990.
2. J. Meurling and R. Jeans, *A Switch in Time—An Engineer's Tale*, Chicago, IL, Telephony Publishing Corp., 1985.
3. F. S. Viglinate, Fundamentals of stored program control of telephone switching systems. *Proceedings of the 1964 19th ACM National Conference*, 1964, pp. 142.201–142.206.
4. J. Meurling and R. Jeans, *The Ericsson Chronicle: 125 Years in Telecommunications*, Stockholm, Sweden: Införmationsförlaget Heimdahls, 2000.
5. O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming*, New York: Academic Press, 1972.
6. I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object-Oriented Software Engineering*, Reading, MA: Addison-Wesley, 1992.
7. D. Herzberg, UML-RT as a candidate for modeling embedded real-time systems in the telecommunication domain, in R. France and B. Rumpe, (eds.), *UML '99 —The Unified Modeling Language: Beyond the Standard; Second International Conference, Fort Collins, CO*, 1999, LNCS 1723, Springer, 1999, pp. 330–338.
8. Specification and Description Language (SDL), ITU-T Recommendation Z.100, International Telecommunication Union, November 1999.
9. Message Sequence Chart (MSC), ITU-T Recommendation Z.120, International Telecommunication Union, November 1999.
10. CCITT High Level Programming Language (CHILL), ITU-T Recommendation Z.200, International Telecommunication Union, October 1996.
11. Introduction to the CCITT Man-Machine Language, ITU-T Recommendation Z.301, International Telecommunication Union, November 1988.
12. B. K. Penny and J. W. J. Williams, The software architecture for a large telephone switch, *IEEE Trans. Communicat. Communicat. Software*, COM-**30**(6): 105–114, 1982.
13. J. Armstrong, *Programming Erlang: Software for a Concurrent World*, Raleigh, NC: The Pragmatic Programmers, 2007.
14. J. Armstrong, Concurrency oriented programming in erlang, *Proc. of the German Unix User Group's Frühjahrsfachgespräch (FFG)*, 2003.
15. Unified Modeling Language: Superstructure, Version 2.1.1, Technical Specification, Object Management Group (OMG), February 2007.
16. Unified Modeling Language: Infrastructure, Version 2.1.1, Technical Specification, Object Management Group (OMG), February 2007.
17. OMG Systems Modeling Language (OMG SysML) Version 1.0, Technical Specification, Object Management Group (OMG), September 2007.
18. K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, New York: ACM Press/ Addison-Wesley Publishing Co., 2000.
19. T. Stahl and M. Völter, *Model-Driven Software Development*, London: John Wiley & Sons, 2006.
20. D. A. Lawson, A new software architecture for switching systems, *IEEE Trans. Commun. Communication Software*, COM-**30**(6): 17–25, 1982.
21. D. Herzberg, Modeling telecommunication systems: From standards to system architectures, PhD thesis, Aachen University of Technology, Department of Computer Science III, 2003.
22. D. Herzberg and M. Broy, Modeling layered distributed communication systems, *Formal Aspects Comput.*, **17**(1): 1–18, 2005.
23. J. Ellsberger, D. Hogrefe, and A. Sarma, *SDL—Formal Object-oriented Language for Communicating Systems*, London: Prentice Hall, 1997.
24. E. W. Dijkstra, The structure of the "THE"-multiprogramming system. *Commun. ACM*, **11**(5): 341–346, 1968.
25. Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model, ITU-T Recommendation X.200, International Telecommunication Union, July 1994.
26. A. S. Tanenbaum, *Computer Networks*, 4th edition, Upper Saddle River, NJ: Prentice Hall PTR, 2003.
27. ISDN Protocol Reference Model, ITU-T Recommendation I.320, International Telecommunication Union, November 1993.
28. J. Eberspacher and H.-J. Vögel, *GSM—Switching, Services and Protocols*, New York: Wiley, 1998.
29. B. Walke, M. P. Althoff, and P. Seidenberg, *UMTS—Ein Kurs*, J. Schlembach Fachverlag, 2001.
30. B-ISDN Protocol Reference Model and its Application, ITU-T Recommendation I.321, International Telecommunication Union, April 1991.
31. Gateway Control Protocol, ITU-T Recommendation H.248, International Telecommunication Union, June 2000.
32. F. Cuervo, N. Greene, C. Huitema, A. Rayhan, B. Rosen, and J. Segers, Megaco Protocol Version 1.0. Standard RFC 3015, Internet Engineering Task Force, November 2000.
33. Information Technology—Open Distributed Processing—Reference model: Overview. ITU-T Recommendation X.901, International Telecommunication Union, 1997.
34. J. R. Putman, *Architecting with RM-ODP*, Englewood Cliffs, NJ: Prentice Hall, 2001.
35. M. Chapman and S. Montesi, Overall Concepts and Principles of TINA—Version 1.0, Tina baseline, TINA-C, February 1995.
36. R. M. Soley and C. M. Stone, Object Management Architecture Guide—Revision 3.0. Document ab/97-05-05, Object Management Group (OMG), June 1995.

37. R. M. Soley and C. M. Stone, *Object Management Architecture Guide*, 3rd edition, New York: Wiley, 1995.

38. T. J. Mowbray and W. A. Ruh, *Inside CORBA: Distributed Object Standards and Applications*, Reading, MA: Addison-Wesley, 1997.

39. Common Object Request Broker Architecture: Core Specification—Version 3.0. Specification formal/2002-11-03, Object Management Group (OMG), November 2002.

40. J. Stankovic, Misconceptions about real-time computing: A serious problem for next generation systems, *IEEE Comput.*, **21**(10): 10–19, 1988.

41. A. B. Tucker, Real-time and embedded systems, in *The Computer Science and Engineering Handbook*, Boca Raton, FL: CRC Press, 1997, pp. 1709–1724.

42. B. Selic, G. Gullekson, and P. T. Ward, *Real-Time Object-Oriented Modeling*, New York: John Wiley & Sons, Inc., 1994.

43. J. A. Stankovic, Real-time and embedded systems, *ACM Comput. Surv.*, **28**(1): 205–208, 1996.

44. D. E. Simon, *An Embedded Software Primer*, Reading, MA: Addison-Wesley, 1999.

45. J. A. Stankovic et al. Strategic directions in real-time and embedded systems. *ACM Comput. Surv.*, **28**(4): 751–763, 1996.

46. J. Miller and J. Mukerji, Model driven architecture (MDA). Technical Description ormsc/2001-07-01, Object Management Group (OMG), 2001.

47. S. Bapat, *Object-Oriented Networks—Models for Architecture, Operations, and Management*. Englewood Cliffs, NJ: Prentice Hall, 1994.

48. J. Larmouth, *ASN.1 Complete*, San Francisco, CA: Morgan Kaufmann, 1999.

49. S. Boecking, *Object-Oriented Network Protocols.*, Reading, MA: Addison-Wesley, 2000.

50. T. Muth, *Modeling Telecom Networks and Systems Architecture: Conceptual Tools and Formal Methods*, Berlin: Springer, 2001.

51. T. Muth, D. Herzberg, and J. Larsen, A fresh view on model-based systems engineering: The processing system paradigm, in *Proc. of the 11th Annual International Symposium of The International Council on Systems Engineering (INCOSE 2001)*; Melbourne, Australia, 2001.

52. T. Muth, *Functional Structures in Networks: AMLn—A Language for Model Driven Development of Telecom Systems*, Berlin: Springer, 2005.

53. D. Harel, Statecharts: A visual formalism for complex systems, *Sci. Comp. Prog.*, **8**(3): 231–274, 1987.

DOMINIKUS HERZBERG
TIM REICHERT
Heilbronn University
Heilbronn, Germany

T

TIME AND STATE IN ASYNCHRONOUS DISTRIBUTED SYSTEMS

INTRODUCTION

A distributed system is characterized by multiple processes that are spatially separated and are running independently. As processes run, they change their *states* by executing *events*. Processes communicate with each other by exchanging messages over a set of communication channels. However, message delays are arbitrary and may be unbounded.

Two inherent limitations of distributed systems are as follows: *lack of global clock* and *lack of shared memory*. Two important implications exist. First, due to the absence of any system-wide clock that is equally accessible to all processes, the notion of common time does not exist in a distributed system, and different processes may have different notions of time. As a result, it is not always possible to determine the order in which two events on different processes were executed. Second, since processes in a distributed system do not share common memory, it is not possible for an individual process to obtain an up-to-date state of the entire system. In addition, because of the absence of a global clock, obtaining a meaningful state of the system, in which states of different processes are consistent with each other, is difficult.

We describe different schemes that implement an abstract notion of time and can be used to order events in a distributed system. We also discuss ways to obtain a consistent state of the system possibly satisfying certain desirable property.

CLOCKS AND ORDERING OF EVENTS

For many distributed applications such as distributed scheduling and distributed mutual exclusion, it is important to determine the order in which various events were executed. If the system has a shared global clock, then time-stamping each event with the global clock would be sufficient to determine the order. However, if such a clock is not available, then it becomes impossible to determine the actual execution order of events. A natural question to ask is as follows: What kind of ordering information can be ascertained in the absence of a global clock?

Each process in the system generates a sequence of events. Therefore it is clear how to order events within a single process. If event *e* occurred before *f* on a process, then *e* is ordered before *f*. But, how do we order events across processes? If *e* is the send event of a message and *f* is the receive event of the same message, then *e* is ordered before *f*. Combining these two ideas, we obtain the following definition:

Definition 1 (Happened-Before Relation). *The happened-before relation, denoted by* $\rightarrow$, *is the smallest transitive relation that satisfies the following:*

(1) *If e occurred before f on the same process, then* $e \rightarrow f$.

(2) *If e is the send event of a message and f is the receive event of the same message, then* $e \rightarrow f$.

As an example, consider a distributed computation involving three processes, namely P_1, P_2, and P_3, shown in Fig. 1. In the figure, time progresses from left to right. Moreover, circles denote events and arrows between processes denote messages. Clearly, $e_2 \rightarrow e_4$, $e_3 \rightarrow f_3$, and $e_1 \rightarrow g_4$.. Also, events e_2 and f_2 are not related by happened-before relation and therefore could have been executed in any order.

The concept of happened-before relation was proposed by Lamport (1). The happened-before relation imposes a *partial order* on the set of events. Any extension of the happened-before relation to a total order gives a possible ordering in which events could have been executed.

The happened-before relationship also captures the causality between events. If an event *e* happened-before an event *f*, then *e* could have caused *f*. In other words, if *e* had not occured, then *f* may not have occurred as well. Events *e* and *f* are said to be *causally related*.

For some distributed applications such as distributed mutual exclusion, it is sufficient to know *some total order* in which events *could have* been executed. The total order may or may not correspond to the actual order of execution of events. However, all processes must agree on the same total order. Furthermore, the total order must respect the happened-before relation. We next describe a mechanism to determine such an ordering at runtime.

Ordering Events Totally: Logical Clocks

A logical clock time-stamps each event with an integer value such that the resulting order of events is consistent with the happened-before relation (Fig. 2). Formally,

Definition 2 (Logical Clock). *A* logical clock *C is a map from the set of events E to the set of natural numbers N with the following constraint:*

$$\forall e, f \in E : e \rightarrow f \Rightarrow C(e) < C(f)$$

The implementation of logical clock, first proposed by Lamport (1), uses an integer variable to simulate local clock on a process. On sending a message, the value of the local clock is incremented and then sent with the message. On receiving a message, a process takes the maximum of its own clock value and the value is received with the message. After

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

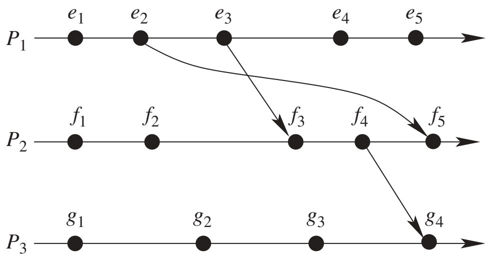


Figure 1. An example of a distributed computation.

taking the maximum, the process increments the clock value. On executing an internal event, a process simply increments its clock. The algorithm can be used even when message communication is unreliable and unordered.

A logical clock has been used to devise efficient distributed algorithms for solving many problems in distributed computing such as mutual exclusion, causal message ordering, and termination detection. For example, in many mutual exclusion algorithms, a logical clock is used to time-stamp requests for critical section. Requests will smaller time-stamps are given priority over requests with larger time-stamps.

Ordering Events Partially: Vector Clocks

A logical clock establishes a total order on all events, even when two events are incomparable with respect to the happened-before relation. For many problems such as distributed debugging and distributed checkpointing and recovery, it is important to determine whether two given events are ordered using the happened-before relation or are incomparable.

The set of events E are partially ordered with respect to $\rightarrow$, but the domain of logical clock values, which is the set of natural numbers, is a total order with respect to $<$. Thus, logical clocks do not provide complete information about the *happened-before* relation. We describe a mechanism called a *vector clock* that allows us to infer the happened-before relation completely.

Definition 3 (Vector Clock). *A* vector clock V *is a map from the set of events E to* N^N *(vectors of natural numbers) with the following constraint:*

$$\forall e, f \in E : e \rightarrow f \Leftrightarrow V(e) < V(f)$$

```
Process $P_i$::
var
    $c$: integer initially 0;

send event :
    $c := c + 1$;
    send $c$ along with the message;

receive event with $d$ as the received timestamp:
    $c := max(c,d) + 1$;

internal event :
    $c := c + 1$;
```

Figure 2. A logical clock algorithm.

```
Process $P_i$:
1   var
2       $\upsilon$ :array[1..$N$] of integer
            initially$(\forall j : j \neq i : \upsilon[j] = 0)$;

3   send event :
4       $\upsilon[i] := \upsilon[i] + 1$;
5       send $\upsilon$ along with the message;

6   receive event of a message tagged with vector $u$:
7       for $j := 1$ to $N$ do
8           $\upsilon[j] := \max(\upsilon[j], u[j])$;
9       $\upsilon[i] := \upsilon[i] + 1$;

10  internal event :
11      $\upsilon[i] := \upsilon[i] + 1$;
```

Figure 3. A vector clock algorithm.

Because $\rightarrow$ is a partial order, it is clear that the time-stamping mechanism should also result in a partial order. Thus, the range of the time-stamping function cannot be a total order like the set of natural numbers used for logical clocks. Instead, we use vectors of natural numbers. Given two vectors x and y of dimension N, we compare them as follows:

$$\begin{aligned} x \leq y &= (\forall k : 1 \leq k \leq N : x[k] \leq y[k]) \\ x < y &= (x \leq y) \vee (x \neq y) \end{aligned}$$

For example, [1,2,1] < [2,2,3] but [2,3,0] and [0,4,1] are incomparable. The vector clock mechanism was proposed independently by Fidge (2) and Mattern (3). Figure 3 shows an implementation of vector clock using vectors of size N, where N is the number of processes in the system.

The algorithm presented in Fig. 3 is described by the initial conditions and by the actions taken for each event type. A process increments its own component of the vector clock after each event (lines 4, 9, and 11). Furthermore, it includes a copy of its vector clock in every outgoing message (line 5). On receiving a message, it updates its vector clock by taking a component-wise maximum with the vector clock included in the message (lines 7 and 8). It is not required that message communication be ordered or reliable. A sample execution of the algorithm is given in Fig. 4.

A vector clock is useful when it is important to determine the *exact relationship* between two events. For example, when *debugging a distributed program*, the programmer may want to find out whether two events are causally related. A potential race condition exists if there is no causal relationship between two events. (This relationship can be detected by comparing the vector time-stamps of the two events.) Likewise, the knowledge that an event could not have caused another event can be used to locate a bug more efficiently. Another application of vector clock arises when *simulating a system in a distributed manner*. In a distributed simulation system, a process needs to know the clock of other processes in order to safely advance its own clock. The notion of a vector clock applies naturally in such a system. The jth entry of a vector clock at process P_i can be interpreted as P_i's knowledge about the (virtual) time at process P_j.

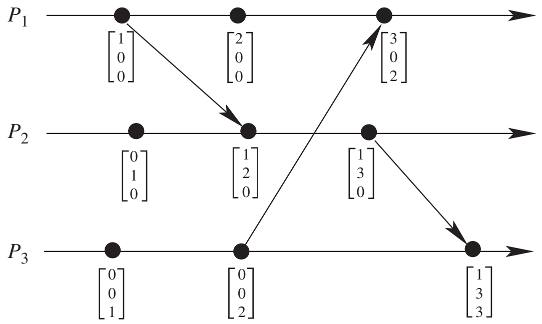

P_1
P_2
P_3
$\begin{bmatrix}1\\0\\0\end{bmatrix}$
$\begin{bmatrix}2\\0\\0\end{bmatrix}$
$\begin{bmatrix}3\\0\\2\end{bmatrix}$
$\begin{bmatrix}0\\1\\0\end{bmatrix}$
$\begin{bmatrix}1\\2\\0\end{bmatrix}$
$\begin{bmatrix}1\\3\\0\end{bmatrix}$
$\begin{bmatrix}0\\0\\1\end{bmatrix}$
$\begin{bmatrix}0\\0\\2\end{bmatrix}$
$\begin{bmatrix}1\\3\\3\end{bmatrix}$

Figure 4. A sample execution of the vector clock algorithm.

As it can be observed, when capturing the happened-before relationship between events, we use a vector containing N entries, where N is the number of processes in the system. Each process has to maintain a vector of size N and each message has to carry a vector of size N, which is expensive when N is large. A question that arises is as follows: Is it possible to capture the happened-before using a vector of smaller size? The answer is in general "no". It can be shown that there are distributed computations for which a vector of size at least N is required to faithfully capture the happened-before relationship (4).

Higher Dimensional Clocks

It is natural to ask whether two or more dimensional clocks can give processes additional knowledge. The answer is "yes." For an event e, let $e.v$ denote the value of the local clock immediately after executing e. A vector clock can be viewed as a knowledge vector. In this interpretation, for an event e on process P_k, $e.v[i]$ denotes what process P_k knows about process P_i after executing event e. In some application, it may be important for the process to have even more *fine-grained* knowledge about its causal past. The value $e.v[i, j]$ could represent what process P_k knows about what process P_i knows about process P_j. For example, if $e.v[i,k] \geq m$ for all i, then process P_k can conclude that everybody knows that it has executed at least m events.

Physical Clocks

Until now, we assumed that message delays are arbitrary and unbounded. However, if message delays are bounded (but still arbitrary), then another way to time-stamp events is to equip each process with a physical clock. Due to limitations in technology, it is possible for physical clocks on different processes to drift apart from each other. Therefore, different physical clocks have to be synchronized with each other at a regular interval. Clocks are synchronized in a manner such that a sufficiently small constant ε exists satisfying the following:

$$\forall i, j : |C_i(t) - C_j(t)| < \varepsilon \tag{1}$$

where $C_i(t)$ denotes the value of the physical clock on process P_i at time t. Let $dC_i(t)/dt$ denote the rate at which the clock on process P_i is running at time t. Clearly, if C_i is an ideal clock, then $dC_i(t)/dt = 1$. Even if C_i is not an ideal clock, we assume that its rate of drift is bounded. Specifically, let κ be the maximum rate at which a clock can drift away from the actual time. Therefore for all i:

$$1 - \kappa < dC_i(t)/dt < 1 + \kappa \tag{2}$$

Clearly, to avoid anomalous behavior, the time-stamp of a receive event should be greater than the time-stamp of the corresponding send event. Therefore, for all i, j, and t:

$$C_i(t + \mu) > C_j(t) \tag{3}$$

where μ is the transmission time. To achieve synchronization of physical clocks that satisfies Equations (1), (2), and (3), the following algorithm proposed by Lamport (1) can be used:

1. Each process sends a synchronization message to all its neighboring processes after every τ units of time. A process includes its value of local physical clock along with the message.
2. A process, on receiving a synchronization message with time-stamp T_m, sets its physical clock value to the maximum of its current value and $T_m + \mu_m$, where μ_m is the minimum amount of time required for message transmission.

GLOBAL STATE

To solve many problems in distributed systems such as termination detection, we need to examine the state of the entire system, which is also referred to as *global state* or *global snapshot*. (In contrast, state of a process is referred to as *local state* or *local snapshot.*) A simple collection of local states, one from each process, may not correspond to a meaningful system state. To appreciate this, consider a distributed database for a banking application. Assume for simplicity that only two sites keep the accounts for a customer. Also assume that the customer has \$500 at the first site and \$300 at the second site. In the absence of any communication between these sites, the total money of the customer can be easily computed to be \$800. However, if there is a transfer of \$200 from site A to site B, and a simple procedure is used to add up the accounts, we may falsely report that the customer has a total of \$1000 in his or her accounts (to the chagrin of the bank). This happens when the value at the first site is used before the transfer and the value at the second site after the transfer. Clearly, the two values are not consistent with each other. Note that \$1000 cannot be justified even by the messages in transit (or that "the check is in the mail"). We now describe what it means for a global state to be meaningful or consistent.

Consistent Global State

Intuitively, a global state captures the set of events that have been executed so far. For a global state G to be consistent, it should satisfy the following condition:

$$\forall e, f : (e \rightarrow f) \wedge (f \in G) \Rightarrow e \in G$$

Sometimes, it is more convenient to describe a global state in terms of local states instead of events. For a local state s, let $s.p$ denote the process to which s belongs. We can extend the definition of the happened-before relation, which was defined on events, to local states as follows: $s \rightarrow t$ if $s.p$ executed an event e after s and $t.p$ executed an event f before t such that either $e = f$ or $e \rightarrow f$. Two local states s and t are *concurrent*, which is denoted by $s \| t$, if $s \nrightarrow t$ and $t \nrightarrow s$. For a global state G, let $G[i]$ refer to the local state of process P_i in G. We now define what it means for a global state to be consistent, when the global state is expressed as a collection of local states.

Definition 4 (Consistent Global State). *A global state G is consistent if it satisfies*

$$\forall i, j : G[i] \| G[j]$$

In general, a global state can be used to deduce meaningful conclusions about the state of the system only if it is consistent.

Finding a Consistent Global State

We discuss how to obtain a consistent view of the entire system. The algorithm, which was proposed by Chandy and Lamport (5), assumes that all channels satisfy the first-in–first-out (FIFO) property. Moreover, it also records the state of all communication channels, which is given by the set of messages in transit. The computation of the snapshot is initiated by one or more processes. We associate with each process a variable called *color* that is either white or red. All processes are initially white and turn red eventually. Intuitively, the computed global snapshot corresponds to the state of the system just before processes turn red. After recording its local state, a process turns red. Thus, the local snapshot of a process is simply the state just before it turned red. The algorithm relies on a special message called a marker. The consistent global snapshot algorithm is given by the following rules:

(1) **(Turning Red Rule):** When a process records its local state, it turns from white to red. On turning red, it sends out a marker on every outgoing channel before sending any application message on that channel. It also starts recording messages on all incoming channels.

(2) **(Marker Receiving Rule):** On receiving a marker, a white process turns red. The process also stops recording messages along that channel.

A process has finished its local snapshot when it has received a marker on each of its incoming channel. The algorithm requires that a marker be sent along all channels. Thus, it has an overhead of one message per channel in the system. We have not discussed how to combine local snapshots into a global snapshot. A simple method would be for all processes to send their local snapshots to a predetermined process. This color-based description of Chandy and Lamport's algorithm for recording a consistent global state of the system was proposed by Dijkstra.

One advantage of Chandy and Lamport's snapshot algorithm is that it is not necessary to "freeze" the computation when recording a global state. However, it is possible that the global state recorded by the algorithm is such that the system never actually passes through the (recorded) state during its execution. But, Chandy and Lamport show that it is possible to reorder the events (while still respecting the happened-before relation) in such a way that the system indeed passes through the recorded global state (5).

Finding a Consistent Global State Satisfying the Given Property

Sometimes it is not sufficient to find just any consistent global state. Rather, we may want to find a consistent global state that satisfies certain global property (6–8). If the global property is stable, that is, it stays true once it becomes true, then repeated invocations of the Chandy and Lamport's algorithm for taking a consistent global snapshot can be used to find the required global state.

We discuss an algorithm that can be used to find a consistent global state satisfying an unstable property. We will assume that the given global property, say B, is constructed from local predicates using Boolean connectives. We first show that B can be detected using an algorithm that can detect q, where q is a pure conjunction of local predicates. The predicate B can be rewritten in its disjunctive normal form as

$$B = q_1 \vee \ldots \vee q_k \qquad k \geq 1$$

where each q_i is a pure conjunction of local predicates. Next, observe that a global state satisfies B if and only if it satisfies at least one of the q_i's. Thus, the problem of detecting B is reduced to solving k problems of detecting q, where q is a pure conjunction of local predicates.

Formally, we define a *weak conjunctive predicate (WCP)* to be true for a given computation if and only if a consistent global state exists in the computation for which all conjuncts are true (7). Intuitively, detecting a WCP is useful generally when one is interested in detecting a combination of states that is unsafe. For example, violation of mutual exclusion for a two-process system can be written as "P_1 is in the critical section and P_2 is in the critical section." To detect a weak conjunctive predicate, it is necessary and sufficient to find a set of concurrent local states, one on each process, in which all local predicates are true. We now present an algorithm to do so.

In this algorithm, one process serves as a checker. All other processes involved in detecting the WCP are referred to as *application processes*. Each application process maintains a vector clock. It also checks for the respective local

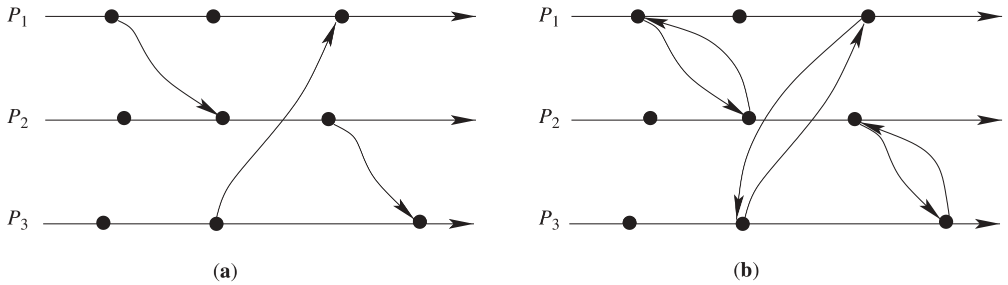


Figure 5. (a) A distributed computation and (b) its slice with respect to the property "all channels are empty."

predicate. Whenever the local predicate of a process becomes true for the *first* time since the most recently sent message (or the beginning of the trace), it generates a debug message containing its local time-stamp vector and sends it to the checker process.

Note that a process is not required to send its vector clock every time the local predicate is detected. If two local states, say s and t, on the same process are separated only by internal events, then they are indistinguishable to other processes so far as consistency is concerned; that is, if u is a local state on some other process, then $s \| u$ if and only if $t \| u$. Thus, it is sufficient to consider at most one local state between two external events and the vector clock need not be sent if no message activity has occurred since the last time the vector clock was sent.

The checker process is responsible for searching for a consistent global state that satisfies the WCP by considering a sequence of candidate global states. If the candidate global state either is not consistent or does not satisfy some term of the WCP, the checker can efficiently eliminate one of the local states in the global state. The eliminated state can never be part of a consistent global state that satisfies the WCP. The checker can then advance the global state by considering the successor to one of the eliminated states. If the checker finds a global state for which no state can be eliminated, then that global state satisfies the WCP and the detection algorithm halts.

Finding All Consistent Global States Satisfying the Given Property

In debugging applications, it is sometimes useful to record all consistent global states that satisfy the given property. A computation slice is a concise representation of all such global states. A slice of a distributed computation with respect to a given property B is a concise representation of all the global states that satisfy B(8).

To understand the principle behind slicing, one needs to note that a computation (an acyclic directed graph on set of events) can be viewed as a generator of all consistent global states. A subset of vertices H of a directed graph is a *consistent global state* if it satisfies the following condition: If H contains a vertex v and *(u, v)* is an edge in the graph, then H also contains u. Given a computation, if one adds additional edges to the computation, the number of consistent possible global state can only decrease. The goal of slicing is to determine the maximum set of edges to add to the graph such that the resulting graph continues to contain all consistent global states of the computation that satisfy the given property. Note that when an edge is added to the original graph, the resulting graph may not be acyclic anymore.

As an example, consider the distributed computation shown in Fig. 5(a). Its slice with respect to the global property "all channels are empty" is depicted in Fig. 5(b).

Three main motivations for computing all the global states satisfy a given property. First, for debugging applications, the programmer may not know the exact condition under which a bug occurs, but only that whenever the bug occurs B is true. Therefore we have to record all global states that satisfy B. Based on slicing, one can provide a "fast-forward" utility in debuggers where the system only goes through global states satisfying B. The second motivation comes from detecting predicates of the form $B_1 \wedge B_2$ in which the programmer knows an efficient detection algorithm for B_1 but not B_2. Instead of searching the set of all global states for a global state that satisfies $B_1 \wedge B_2$, slicing allows the programmer to restrict the search to only those global states that satisfy B_1. This set of global states may be exponentially smaller than the original set of global states.

The reader is referred to Ref. 9 for a more detailed description of slicing and associated algorithms.

BIBLIOGRAPHY

1. L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Commun. ACM*, **21**(7): 558–565, 1978.
2. C. Fidge, Logical time in distributed computing systems, *IEEE Computer*, **24**(8): 28–33, 1991.
3. F. Mattern, Virtual time and global states of distributed systems, *Parallel and Distributed Algorithms: Proceedings of the Workshop on Distributed Algorithms (WDAG)*, 1989, pp. 215–226.

4. B. Charron-Bost, Concerning the size of logical clocks in distributed systems, *Informat. Process. Lett.*, **39**: 11–16, 1991.
5. K. M. Chandy and L. Lamport, Distributed snapshots: Determining global states of distributed systems, *ACM Trans. Comp. Syst.*, **3**(1): 63–75, 1985.
6. R. Cooper and K. Marzullo, Consistent detection of global predicates, *Proc. of the ACM/ONR Workshop on Parallel and Distributed Debugging, Santa Cruz, California, 1991*, 163–173
7. V. K. Garg and B. Waldecker, Detection of weak unstable predicates in distributed programs, *IEEE Trans. Parallel Distributed Sys.*, **5**(3): 299–307, 1994.
8. S. Alagar and S. Venkatesan, Techniques to tackle state explosion in global predicate detection, *IEEE Trans. Softw. Engineer.*, **27**(8): 704–714, 2001.
9. V. K. Garg, *Elements of Distributed Computing*, New York: John Wiley and Sons, Inc., 2002.
10. N. Mittal and V. K. Garg, Computation slicing: Techniques and theory, *Proc. of the Symposium on Distributed Computing (DISC)*, 2001, pages 78–92.

VIJAY K. GARG[1]
The University of Texas
at Austin
Austin, Texas
NEERAJ MITTAL
The University of at Dallas
Richardson, Texas

[1] Supported in part by the NSF Grant CNS-0509024, Texas Education Board Grant 781, SRC Grant 2006-TJ-1426, and Cullen Trust for Higher Education Endowed Professorship.

TRANSPORT LAYER

INTRODUCTION

The Internet has evolved into an extremely large complex system and has changed many important aspects of our lives. Like any complex engineering system, the design of the Internet is carried out in a modular way, where each main functional module is called a "layer." One of the layering structures often used is the five-layer model consisting of the physical layer, the link layer, the network layer, the transport layer, and the application layer.[1] See Fig. 1 for a simple illustration.

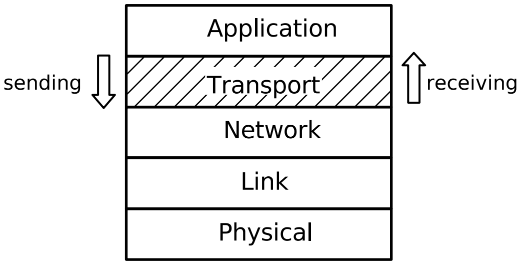


Figure 1. Internet protocol stack.

The sending and receiving computers each run analogous stacks, with data being passed down the stack from the sending application, and then up the receiver's stack to the receiving application. The physical layer is the part that actually moves information from one place to another, such as by varying voltages on wires or generating electromagnetic waves. The application layer is the part with which users interact, such as the hypertext transport protocol (HTTP) used to browse the Web, or the simple mail transfer protocol (SMTP) used to send e-mail.

Each layer consists of *protocols* to specify such things as the data format, the procedure for exchanging data, the allocation of resources, and the actions that need to be taken in different circumstances. This protocol can be implemented in either software or hardware or both. This article concerns transport layer protocols and their associated algorithms, mainly focusing on the wireline Internet but also discussing some other types of networks such as wireless ones.

The transport layer manages the end-to-end transportation of packets across a network. Its role is to connect application processes running on end hosts as seamlessly as possible, as if the two end applications were connected by a reliable dedicated link, thus making the network "invisible." To do this, it must manage several nonidealities of real networks: shared links, data loss and duplication, contention for resources, and variability of delay. By examining these functionalities in turn, we will provide a brief introduction to this important layer, including its functions and implementation, with an emphasis on the underlying ideas and fundamentals. We will also discuss possible directions for the future evolution of the transport layer and suggest some further reading.

MULTIPLEXING: MANAGING LINK SHARING

One basic function of the transport layer is *multiplexing* and *demultiplexing*. Usually there are multiple application processes running on one host. For example, a computer may be sending several files generated by filling in web forms, while at the same time sending e-mails. The network layer only cares about sending a stream of data out of the computer. Therefore, the transport layer needs to aggregate data from different applications into a single stream before passing it to the network layer. This is called multiplexing. Similarly, when the computer receives data from the outside, the transport layer is again responsible for distributing that data to different applications—such as a web browser or e-mail client—in a process called demultiplexing. Figure 1 also shows the data directions for multiplexing (sending) and demultiplexing (receiving).

Multiplexing is achieved by dividing flows of data from the application into (one or more) short *packets*, also called *segments*. Packets from different flows can then be interleaved as they are sent to the network layer. Demultiplexing is achieved by allocating each communication flow a unique identifier. The sender marks each packet with its flow's identifier, and the receiver separates incoming packets into flows based on their identifiers. In the Internet, these identifiers consist of transport layer *port* numbers, and additionally the network layer addresses of the sender and receiver and a number identifying the transport layer protocol being used. Even before a flow is started, a "well-known" port number can be used to identify which process on the receiver the sender is attempting to contact; for example, Web servers are "well known" to use port 80.

Multiplexing and demultiplexing is one of the most fundamental tasks of the transport layer. Other functions are required by some applications but not by all, and so different transport layer protocols provide different subsets of the possible services. However, essentially all transport layer protocols perform at least multiplexing and demultiplexing.

There are two dominant types of transport layer protocol used in the Internet. One is UDP (User Datagram Protocol), and the other is TCP (Transmission Control Protocol). The former provides unreliable and connectionless service to the upper layers, whereas the latter generates reliable and connection-based service.

[1] There are also some other ways of defining these layers; e.g., the standard OSI (open systems interconnection) reference model defines seven layers with the session layer and the presentation layer added.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

UDP: Multiplexing Only

UDP is a very simple protocol. It is called connectionless because all UDP packets are treated independently by the transport layer, rather than being part of an ongoing flow.[2] Besides minor error checking, UDP essentially only does multiplexing and demultiplexing. It does not provide any guarantee that packets will be received in the order they are sent, or even that they will be received at all. It also does not control its transmission rate. In fact, the rationale behind the design of UDP is to let applications have more control over the data sending process and to reduce the delay associated with setting up a connection. These features are desirable for certain delay-sensitive applications such as streaming video and Internet telephony. In general, applications that can tolerate certain data loss/corruption but are sensitive to delay would choose to use UDP.

TCP: Reliable Connection-Oriented Transport

In contrast to UDP, TCP provides a connection-oriented service, which means that it sends data as a stream of related packets, making concepts such as the order of packets meaningful. In particular, TCP provides reliable service to upper layer applications, ensuring that the packets are correctly received and in the order in which they are sent.

At the start of a connection, TCP uses a *three-way handshake* to establish a connection between sender and receiver, in which they agree on what protocol parameters to use. This process takes 1.5 round-trip times (one side sends a SYNchronize packet, the other replies with a SYN and an ACKnowledge packet, and the first confirms with an ACK), which is an overhead avoided by UDP.

TCP receives data from the application as a single stream, e.g., a large file, and segments it into a sequence of packets. It tries to use large packets to minimize overhead, but there is a maximum size that the network can carry efficiently, called the MTU (maximum transfer unit). TCP is responsible for choosing the correct size, in a process called *path MTU discovery*. In contrast, UDP is given data already segmented into packets, and so it is the application's responsibility to observe MTU restrictions.

TCP is the most common transport protocol in the Internet; measurements show that it accounds for about 80% of the traffic (1). Applications such as file transmission, Web browsing, and e-mail use it partly because of its ability to transfer continuous streams of data reliably, and partly because many firewalls do not correctly pass other protocols. The following two sections, on *reliable transmission* and *congestion control*, describe in greater detail the main features of TCP.

RELIABLE TRANSMISSION: MANAGING LOSS, DUPLICATION, AND REORDERING

When the underlying network layer does not guarantee to deliver all packets, achieving reliable transmission on top of this unreliable service becomes an important task. Reasons for packet loss include transient routing loops, congestion of a resource, or physical errors that were not successfully corrected by the physical or link layer.

This problem is very similar to that faced by the link layer. The difference is that the link layer operates over a single unreliable physical link to make it appear reliable to the network layer, whereas the transport layer operates over an entire unreliable network to make it appear reliable to the application. For this reason, the algorithms employed at the transport layer are very similar to those employed at the link layer, and they will be reviewed briefly here.

ARQ (Automatic Repeat-reQuest) is the basic mechanism to deal with data corruption. When the receiver receives a correct data packet, it will send out a positive acknowledgment (ACK); when it detects an error, it will send out a negative acknowledgment (NAK).

Because physically corrupted packets are usually discarded by the link layer, the transport layer will not directly observe that a packet has been lost, and hence, many transport layer protocols do not explicitly implement NAKs. A notable exception is multicast transport protocols. Multicast protocols allow the sender to send a single packet, which is replicated inside the network to reach multiple receivers, possibly numbering in the thousands. If each sent an ACK for every packet, the receiver would be flooded. If instead receivers send NAKs only when they believe packets are missing, the network load is greatly reduced. TCP implicitly regards ACKs for three or more out-of-order packets (resulting in "duplicate ACKs") as forming a NAK.

An interesting problem immediately develops. If the sender only sends the next packet when it is sure the first one is received correctly, the system will be very inefficient. In that case, the sender would only be able to send one packet every round-trip time (the time it takes for a packet to reach the destination and the ACK to return), whereas the time taken to send a packet is usually much smaller than the round-trip time. This is especially true for today's high-speed networks. For example, consider a 1 Gbit/s (1,000,000,000 bits per second) connection between two hosts that are separated by 1500 km and therefore have a round-trip distance of 3000 km. Sending a packet of size 10 kbit takes only 10 μs (ten microseconds), whereas the round-trip time cannot be smaller than the physical distance divided by the light of speed (300,000 km per second), which in this case is 10 ms. In other words, in this case, the utilization of the sender is about 0.1%, which is clearly not acceptable. This is the motivation for the general sliding window algorithm that will be discussed in the following subsection.

Sliding Window Transmission Control

The basic idea here is again simple. The sender sends more packets into the network while it is waiting for the acknowledgment of the first packet. This certainly increases the utilization. On the other hand, it cannot send too much before receiving ACKs as that may heavily congest the network or overflow the receiver. In general, the sender is allowed to send no more than W packets into the network before receiving an acknowledgment. Here W is called the *window* size. If $W = 1$, it reverts to the inefficient transmission previously discussed. Since the sender can send W

[2] It is common for the application layer to implement flows on top of UDP, but that is not provided by UDP itself.

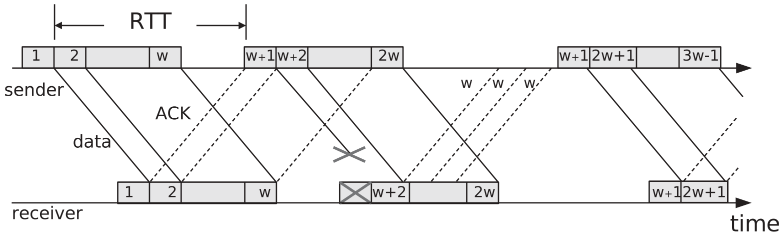


Figure 2. Sliding window flow control, with packet $W + 1$ being lost.

packets every round-trip time, the utilization is increased by a factor of W if the network and receiver can handle the packets. If not, then there will be congestion and the value of W must be reduced, which is the topic of the next section. The left of Fig. 2 shows the case when packets are successfully received. It shows that using $W > 1$ allows more than one packet to be sent per round-trip time (RTT), increasing the throughput of the connection.

However, things become a little more complex when $W > 1$ as now the sender needs to keep track of acknowledgments from more than one packet in one round-trip time. This is done by giving every packet a *sequence number*. When a packet is received, the receiver sends back an ACK carrying the appropriate sequence number.

When packets are received out of order, the receiver has three options, depending on the protocol. It may simply discard packets that are received out of order, it may forward them immediately to the application, or it may store them so that it can deliver them in the correct order once the missing packet has been received, possibly after being resent. Finally, if the sender has not received an ACK for a particular packet for a certain amount of time, a *timeout* event occurs. After that, the sender resends all packets that are sent out but have not yet been acknowledged.

Realization

The following example presents a simplified example of how the above ideas are realized by TCP, which is currently the most widespread example of sliding window transmission control.

Example 1. Rather than acknowledging each packet, TCP ACKs cumulatively acknowledge all data up until the specified packet. This increases the robustness to the loss of ACKs. Figure. 2 shows the operation of TCP when packet $W + 1$ is lost. Initially, the window spans from 1 to W, allowing the first W packets to be sent. The sender then waits for the first packet to be acknowledged, causing the window to slide to span packets 2 to $W + 1$, allowing the $W + 1$st packet to be sent. This continues until the second window is sent, and after sending the $2W$th packet, the sender again must pause for an ACK to slide the window along. However, this time, the $W + 1$st packet was lost, and so no ACK is received. When packet $W + 2$ is received, the receiver cannot acknowledge it, since that would implicitly acknowledge packet $W + 1$, which has not yet arrived. Instead, it sends another ACK for the most recently received packet, W. This is repeated for all subsequent arrivals, until $W + 1$ is received. When the sender receives the third duplicate ACK for W, it assumes that $W + 1$ was lost, and retransmits it. It then continues transmitting from where it left off, with packet $2W + 1$.

The precise response to packet loss of current TCP is more complex than in this example, because packet loss is treated as a signal that a link in the network is overloaded, which triggers a congestion control response, as described in the following section.

CONGESTION CONTROL: MANAGING RESOURCE CONTENTION

When transport layer protocols were first designed, they were intended to operate as fast as the receiver could process the data. The transport layer provided “flow control” to slow the sender down when the receiver could not keep up. However, in the 1980s, the Internet suffered from several famous *congestion collapses*, in which the sliding window mechanism was resending so many packets that the network itself because overloaded to the point of inoperability, even when the receivers were not overloaded.

Recall from the previous section that senders use $W > 1$ to increase the utilization of the network. Congestion occurred because flows sought to use more than 100% of the network capacity. As a result, a set of rules were proposed (2) for how senders should set their windows to limit their aggregate sending rate while maintaining an approximately fair allocation of rates.

Congestion control considers two important topics: what rates would we ideally like to allocate to each flow in a given network, and how can we achieve that in practice using only distributed control. The latter is made difficult because of the decentralized nature of the Internet: senders do not know the capacity of the links they are using, how many other flows share them, or how long those flows will last; links do not know what other links are being used by the flows they are carrying; and nobody knows when a new flow will arrive. Figure 3 shows an example in which two flows each use three links, of which they share one.

Let us now consider the current solutions to the problem of implementing congestion control in a scalable way, and then examine the other problem of deciding what rate allocation is more desirable.

Existing Algorithms

There are two main phases of a congestion control algorithm: *slow start* and *congestion avoidance*, punctuated by

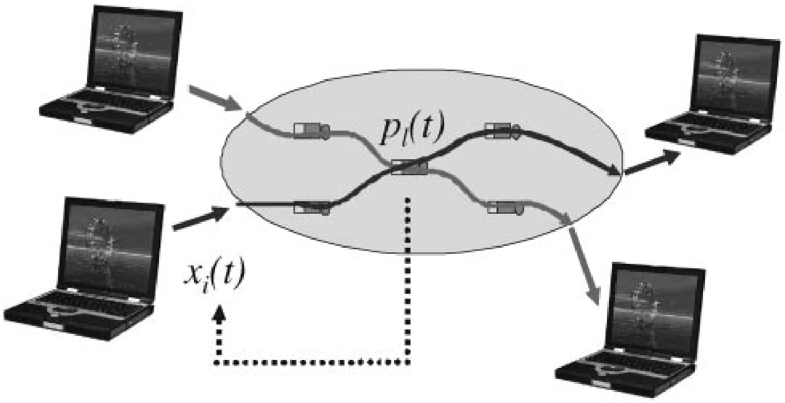


Figure 3. Two flows sharing a link, and also using nonshared links.

short periods of retransmission and loss recovery. We now introduce both using the standard TCP congestion control algorithm, commonly called TCP Reno.[3]

When a TCP connection begins, it starts in the slow start phase with an initial window size of two packets. This results in a slow initial transmission, giving rise to the name. It then rapidly increases its sending rate. It doubles its window every round-trip time until it observes a packet loss, or the window reaches a threshold called the "slow start threshold." If a loss occurs, the window is then halved, and in either case, the system enters the congestion avoidance phase. Note that the sender increases its transmission rate exponentially during the slow start.

In the congestion avoidance phase, the sender does what is known as Additive Increase Multiplicative Decrease (AIMD) adjustment. This was first proposed by Chiu and Jain (3) as a means to obtain fair allocation, and implemented in the Internet by Jacobson (2). Every round-trip time, if all packets are successfully received, the window is increased by one packet. However, when there is a loss event, then the sender will halve its window. Because large windows are reduced by more than small windows, AIMD tends to equalize the size of windows of flows sharing a congested link (3). Finally, if a timeout occurs, the sender will start from slow start again. Figure 4 shows how the window evolves along time in TCP Reno. Importantly, TCP Reno uses packet loss as congestion indication.

In summary, the basic engineering intuition behind most congestion control protocols is to start probing the network with a low transmission rate, quickly ramp up initially, then slow down the pace of increase, until an indicator of congestion occurs and transmission rate is reduced. Often packet loss or queueing delay (4) are used as congestion indicators, and packet loss events are in turn inferred from local measurements such as three duplicated acknowledgments or timeout. These design choices are clearly influenced by the views of wireline packet-switched networks, in which congestion is the dominant cause of packet loss. The choice of the ramp-up speed and congestion indicators have mostly been based on engineering intuition until recent developments in predictive models of congestion control have helped with a more systematic design and tuning of the protocols.

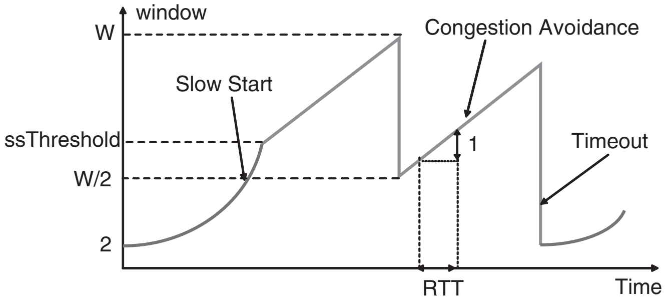


Figure 4. TCP Reno Window Trajectory.

This window adaptation algorithm is combined with the sliding window transmission control, to form the whole window-based congestion control mechanism, as illustrated in Fig. 5. The transmission control takes two inputs, the window size and the acknowledgments from the network. The window size is controlled by the congestion control algorithm such as TCP Reno, which updates the window based on the estimated congestion level in the network. In summary, with window-based algorithms, each sender controls its window size—an upper bound on the number of packets that have been sent but not acknowledged. As pointed out by Jacobson (2), the actual rate of transmission is controlled or "clocked" by the stream of received acknowledgments (ACKs). A new packet is

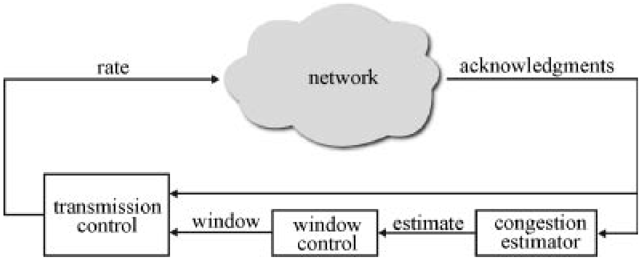


Figure 5. Window-based congestion control.

[3] Most systems actually implement a variant of Reno, typically NewReno, since Reno performs poorly when two packets are lost in a single round trip. However, the differences do not affect the descriptions in this section, and so we use the term "Reno."

transmitted only when an ACK is received, thereby ideally keeping the number of outstanding packets constant and equal to the window size.

Theoretical Foundation

As mentioned, congestion control is essentially a resource allocation scheme that allocates the capacities of links to TCP flows. It is desirable to be able to calculate the share of the capacity and discuss its properties, such as the fairness of the allocation.

Many valuable models of TCP have been proposed. Since Kelly's work in the late 1990s, generic congestion control protocols have been modeled as distributed algorithms maximizing the total benefit obtained by the applications (5–8). This "reverse-engineering" approach shows that existing TCP congestion control mechanisms are implicitly solving an underlying global optimization, with an interpretation of link-price-based balancing of bandwidth demand by the end users. Following economic terminology, the user objective being maximized is called the *utility*, and the utility that the ith flow obtains by sending at a rate x_i is denoted $U_i(x_i)$. If each flow i uses a set of links $L(i)$ and link $l \in L(i)$ has capacity c_l, then the problem of maximizing the utility can be expressed as follows:

$$\max_{x \geq 0} \sum_i U_i(x_i)$$

$$\text{subject to} \sum_{i:l \in L(i)} x_i \leq c_l$$

This is a convex optimization problem provided that the utility functions follow the usual "law of diminishing returns," that is, the utility increases as the rate received increases, but the incremental benefit becomes smaller. Such problems have a very rich mathematical structure.

The theory of Lagrange duality for convex optimization allows the problem to be decomposed into subproblems in which each flow *independently* chooses its rate based on congestion signals from the links, such as packet loss or queueing delay, which are computed based only on local information. Again following economic terminology, these congestion signals are sometimes referred to as *prices*.

The strict convexity structure also implies that the optimal rates are unique, and that those rates are independent of many properties of the links, such as their buffer sizes. In particular, as long as the congestion signal is zero when the sum of the rates through the link is less than its capacity, it does not matter how the congestion signals are calculated; the equilibrium rates will depend only on the utility functions, which are in turn determined by the TCP algorithm at the sender.

The choice of utility function determines the notion of fairness implemented by the network (9). If the utility function is almost linear, it reflects only slightly diminishing returns as the transmission rate is increased, and the network will seek to maximize the sum of the rates of all flows, with no regard to fairness. At the opposite extreme, if the incremental benefit decreases rapidly, the utility function will be very concave and *max–min* sharing is achieved. The max–min rate allocation is the one in which no flow can increase its rate, except by reducing the rate of a flow that already has a lower rate. This is often seen as the fairest way to allocate rates.

A logarithmic utility function results in a compromise between fairness and throughput known as proportional fairness. Similarly, to a first approximation, the utility of the AIMD algorithm used by TCP Reno is as follows:

$$U_i(x_i) = -\frac{1}{x_i \tau_i^2}$$

where τ_i is the round-trip time of the flow i. This is similar to proportional fairness, but it tends slightly toward improving fairness at the expense of throughput, as will be seen in the following example.

Example 2. Consider a network with two congested links, each with a capacity of c. One flow uses both links, and each link also carries a single-link flow, as shown in Fig. 6.

The maximum sum of rates is achieved when $x_1 = x_2 = c$ and $x_3 = 0$, which maximizes the sum of utilities if $u_i(x_i)$ is approximately proportional to x_i. This is clearly unfair since the two-link flow cannot transmit at all. In contrast, the max–min rates are $x_1 = x_2 = x_3 = c/2$, which maximizes the sum of utilities if $u_i(x_i)$ rises very sharply for $x_i < c/2$, and rises only very slightly for $x_i > c/2$. This is completely fair in that all flows receive the same rate, but it is unfair in the sense that the long flow causes twice as much congestion but still achieves the same rate. In this case, the total rate has reduced from $c + c = 2c$ to $c/2 + c/2 + c/2 = 1.5c$.

The proportional-fair rates, which maximize logarithmic utilities, are $x_1 = x_2 = 2c/3$ and $x_3 = c/3$. These rates are in the ratio 2:1 because the resources consumed are in the ratio 1:2, and they give a total rate of around 1.67c. If all flows have equal round-trip times τ_i, TCP Reno will give average rates in ratio $1 : \sqrt{2}$, namely $x_1 = x_2 = \sqrt{2}c/(1 + \sqrt{2})$ and $x_3 = c/(1 + \sqrt{2})$, with a total throughput of 1.59c. The fact that the rates are more similar for Reno than for proportional fairness, but the sum of rates is lower, supports the statement that Reno is a compromise between proportional fairness and max–min fairness.

In concluding this subsection, we mention that there has been a lot of recent research on both reverse-engineering and forward-engineering congestion control protocols, based on the above mathematical model and its variants. Some of the ongoing research issues will be briefly presented toward the end of this entry.

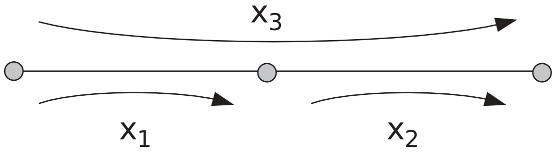


Figure 6. A two-link network shared by three flows.

TIMING RESTORATION: MANAGING DELAY VARIATION

In most cases, it is desirable for the transport layer to pass data to the receiving application as soon as possible. The notable exception to this is streaming audio and video. For these applications, the temporal spacing between packets is important; if audio packets are processed too early, the sound becomes distorted. However, the spacing between packets gets modified when packets encounter network queueing, which fluctuates in time. In its role of hiding lower layer imperfections from the upper layers, it is up to the transport layer to reestablish the timing relations between packets before sending them to the application.

Specialist transport protocols such as the Real Time Protocol (RTP) are used by flows requiring such timing information. RTP operates on top of traditional transport layer protocols such as UDP and provides each packet with a timestamp. At the receiver, packets are inserted as soon as they arrive into a special buffer known as a jitter buffer, or playout buffer. They are then extracted from the jitter buffer in the order in which they were sent and at intervals exactly equal to the interval between their timestamps. Jitter buffers can only add delay to packets, not remove delay; if a packet is received with excessive delay, it must simply be discarded by the jitter buffer. The size of the jitter buffer determines the tradeoff between the delay and packet loss experienced by the application.

TCP can itself cause delay fluctuation, both through ACK-clocking and the fluctuation in rate induced by Reno-like congestion control. When transmitting video and other streaming data, it is sometimes desirable to have packets sent with more uniform spacing. The burstiness caused to ACK-clocking can be avoided by *paced* TCP. Rather than sending packets exactly when acknowledgments are received, paced TCP sends one window's worth of packets uniformly spaced throughout a round-trip time. Many congestion control algorithms have been proposed that dispense with Reno's AIMD, reducing burstiness on longer timescales; notable examples include TCP Vegas and TCP Friendly Rate Control (TFRC).

RECENT AND FUTURE EVOLUTION

With the Internet expanding to global scale and becoming ubiquitous, it is encountering more and more new environments. On the one hand, the TCP/IP "hourglass model"[4] has been very successful at separating applications from the underlying physical networks and enabling the Internet's rapid growth. On the other hand, some basic assumptions are becoming inaccurate or totally invalid, which therefore imposes new challenges. This section describes some of the hot issues in both the Internet Engineering Task Force (IETF, the primary Internet standards body) and the broad research community. Many topics touch on both implementation issues and fundamental questions. We will start with the most implementation related ones and then progress to the more theoretical ones. It is certainly clear that the list below cannot be exhaustive and instead reflects the authors' taste and expertise. For example, many more variants of TCP congestion control are proposed in the last few years than can be surveyed within an encycopedia. In addition to the rest of this section, there are many other exciting developments in the theory and practice of transport layer design for future networks.

Protocol Enhancement

1) Datagram Congestion Control Protocol: Although TCP provides reliable in-order data transfer and congestion control, UDP provides neither. However, applications such as video transmission should implement congestion control, but they do not need guaranteed transmission. Moreover, they cannot tolerate the delay caused by retransmission and in-order delivery. Consequently, the IETF has developed a new protocol called DCCP (Datagram Congestion Control Protocol), which can be viewed either as UDP with congestion control or as TCP without the reliability guarantees. Because many firewalls block unknown protocols, DCCP has not yet been widely used, although it is implemented in many operating systems.

2) Multiple indicators of congestion: The current TCP NewReno relies primarily on detection of packet loss to determine its window size. Other proposals have been made that rely primarily on estimates of the queueing delay. The utility maximization theory applies to networks in which all flows are of the same "family." For example, all flows in the network may respond solely to loss; different flows may respond differently to loss, provided that loss is the only congestion signal they use. However, when a single network carries flows from both the "loss" and "delay" families, or flows responding to other "price" signals such as explicit congestion signals, the standard theory fails to predict how the network behaves.

Unlike networks carrying a single family of algorithms, the equilibrium rates now depend on router parameters, such as buffer sizes, and flow arrival patterns. The equilibrium can be nonunique, inefficient, and unfair. The situation is even more complicated when some individual flows respond to multiple congestion signals, such as adjusting AIMD parameters based on estimates of queueing delay. This has motivated recent efforts to construct a more general framework, which includes as a special case the theory for networks using congestion signals from a single family (10).

Applications

1) Delay-tolerant networks: Sliding window protocols rely on feedback from the receiver to the sender. When communicating with spacecraft, the delay between sending and receiving may be minutes or hours, rather than milliseconds, and sliding windows become infeasible. This has lead to research into "interplanetary TCP." Technology called DTN (Delay-Tolerant Networking) is being developed for this, and also for more mundane situations in which messages suffer long delays. One example is in vehicular networks, in which messages are exchanged over short-range links as vehicles pass one another, and

[4] It is called an hourglass because a small number of simple network and transport layer protocols connect a large variety of complex application layer protocols above with a large variety of link and physical layer protocols below.

are physically carried by the motion of the vehicles around a city. In such networks, reliability must typically be achieved by combinations of error correcting codes and multipath delivery (e.g., through flooding).

2) Large bandwidth delay product networks: In the late 1990s, it became clear that TCP NewReno had problems in high speed transcontinental networks, commonly called "large bandwidth-delay product" or "large BDP" networks. The problem is especially severe when a large BDP link carries only a few flows, such as those connecting supercomputer clusters. In these networks, an individual flow must have a window *W* of many thousands of packets. Because AIMD increases the window by a single packet per round trip, the sending rate on a transatlantic link will increase by around 100 kbit/s. It would thus take almost three hours for a single connection to start to use fully a 1 Gbit/s link.

Many solutions have been proposed, typically involving increasing the rate at which the window is increased, or decreasing the amount by which it is decreased. However, these both make the algorithm more "aggressive," which could lead to allocating too much rate to flows using these solutions and not enough to flows using the existing TCP algorithm. As a result, most solutions try to detect whether the network actually is a "large BDP" network, and adjust their aggressiveness accordingly. Another possibility is to avoid dealing with packet loss in large BDP networks. Researchers have developed various congestion control algorithms that use congestion signals other than packet loss, e.g., queueing delay. Many proposals also seek to combine timing information with loss detection. This leads to the complications of multiple indicators of congestion described previously.

An alternative that is often proposed is for the routers on the congested links to send explicit messages indicating the level of congestion. This was an important part of the available bit-rate (ABR) service of asynchronous transport mode (ATM) networks. It may allow more rapid and precise control of rate allocation, such as the elimination of TCP's time-consuming slow start phase. However, it presents significant difficulties for incremental deployment in the current Internet.

3) Wireless networks: Wireless links are less ideal than wired links in many ways. Most importantly, they corrupt packets because of fading and interference, either causing long delays as lower layers try to recover the packets, or causing packets to be lost. The first of these results in unnecessary timeouts, forcing TCP to undergo slow start, where as the latter is mistaken for congestion and causes TCP NewReno to reduce its window. Again, many solutions have been proposed. Some mask the existence of loss, where as others attempt to distinguish wireless loss from congestion loss based on estimates of queueing delay or explicit congestion indication.

The fundamental task of resource allocation is also more challenging in wireless networks, partly because resources are more scarce and users may move, but more importantly because of the interaction between nearby wireless links. Because the capacity of a wireless link depends on the strength of its signal and that of interfering links, it is possible to optimize resource allocation over multiple layers in the protocol stack. This cross-layer optimization generalizes the utility maximization. It provides challenges as well as opportunities to achieve even greater performance, which requires a careful balance between reducing complexity and seeking optimality.

Research Challenges

1) Impact of network topology: Transport layer congestion control and rate allocation algorithms are often studied in very simple settings. Two common test networks are *dumbbell* networks in which many flows share a single congested link, and *parking lot* networks, consisting of several congested links in series with one flow traversing all links, and each link also being the sole congested link for another short flow. Figure 6 shows a two-link parking lot network. These are used partly because they occur frequently in the Internet (such as when a flow is bottlenecked at the ingress and egress access links), and partly because there are intuitive notions of how algorithms "should" behave in these settings. However, these simple topologies often give a misleading sense of confidence in our intuition. For example, in parking lot topologies, algorithms that give a high rate to the single link flows at the expense of the multilink flow achieve higher total throughput, and thus it is widely believed that there is a universal tradeoff between fairness and efficiency. However, networks exist in which increasing the fairness actually increases the efficiency (11). This and other interesting and counter-intuitive phenomena develop only in a network setting where sources interact through shared links in intricate and surprising ways.

2) Stochastic network dynamics: The number of flows sharing a network is continually changing, as new application sessions start, and others finish. Furthermore, packet accumulations at each router is shaped by events in all upstream routers and links, and packet arrivals in each session are shaped by the application layer protocols, including those in emerging multimedia and content distribution protocols. Although it is easy to study the effects of this variation by measuring either real or simulated networks, it is much harder to capture these effects in theoretical models. Although the deterministic models studied to date have been very fruitful in providing fundamental understanding of issues such as fairness, there is an increasing interest in extending the theoretical models to capture the stochastic dynamics occurring in real networks.

As an example of one type of these dynamics, consider a simple case of one long flow using the entire capacity of a given link, and another short flow that starts up using the same link. If the short flow finishes before the long flow does, then the finish time of the long flow will be delayed by the size of the short flow divided by the link capacity, *independent* of the rate allocated to the short flow, provided that the sum of their rates is always the link capacity. In this case, it would be optimal to process the flows in "shortest remaining processing time first" (SRPT) order; that is, to allocate all rate to the short flow and meanwhile totally suspend the long flow. However, as the network does not know in advance that the short flow will finish first, it will instead seek to allocate rates fairly between the two flows. This can cause the number of simultaneous flows to be much larger than the minimum possible, resulting in each flow getting a lower average rate than necessary. The fundamental difficulty is that the optimal strategy is no longer to allocate instantaneous rates fairly based on the existing flows.

FURTHER READING

The transport layer is a main topic in many textbooks on computer networks, which is now a standard course in most universities. This article only seeks to provide a basic understanding of the transport layer. For those who are interested in digging into details and working in related areas, the following references are a useful starting point. For a complete introduction to computer networks including the transport layer, see any of the major networking textbooks such as Ref. 12. The Internet's main transport layer protocol, TCP, is described in detail in Ref. 13, although several details have evolved since that was written. For a general mathematical approach to understanding network layering, see a recent survey (14). Samples of early TCP congestion control analysis include Refs. 15–18. A survey on the mathematical treatment of Internet congestion control can be found in Ref. 19. Enduring issues are also well described in Ref. 20.

BIBLIOGRAPHY

1. M. Fomenkov, K. Keys, D. Moore, and K. Claffy, Longitudinal study of Internet traffic in 1998-2003, *WISICT '04: Proc. Winter Int. Symp. Info. Commun. Technol*, 2004.
2. V. Jacobson, Congestion avoidance and control, *Proc. ACM SIGCOMM*, 1988.
3. D. M. Chiu and R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, *Computer Networks ISDN Sys.*, **17**(1): 1–14, 1989.
4. L. Brakmo and L. Peterson, TCP Vegas: End-to-end congestion avoidance on a global Internet, *IEEE J. Selected Areas Communi.***13**(8): 1465–80, 1995.
5. F. Kelly, A. Maoulloo, and D. Tan, Rate control for communication networks: Shadow prices, proportional fairness and stability. *J. Operational Research Society*, **49**: 237–252, 1998.
6. S. Kunniyur and R. Srikant, End-to-end congestion control schemes: Utility functions, random losses and ECN marks, *IEEE/ACM Trans. Networking*, **11**(5): 689–702, 2003.
7. S. Low, A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. Networking*, **11**(4): 525–536, 2003.
8. S. Low and D. Lapsley, Optimization flow control—I: Basic algorithm and convergence, *IEEE/ACM Trans. Networking*, **7**(6): 861–874, 1999.
9. J. Mo and J. Walrand, Fair end-to-end window-based congestion control, *IEEE/ACM Trans. Networking*, **8**(5): 556–567, 2000.
10. A. Tang, J. Wang, S. Low and M. Chiang, Equilibrium of heterogeneous congestion control: Existence and uniqueness, *IEEE/ACM Trans. Networking*, **15**(4): 824–837, 2007.
11. A. Tang, J. Wang and S. H. Low, Counter intuitive throughput behaviors in networks under end-to-end control, *IEEE/ACM Trans. Networking*, **14**(2): 355–368, 2006.
12. J. Kurose and K. Ross, *Computer Networking*. Fourth edition, Addison Wesley, 2007.
13. W. R. Stevens, *TCP/IP Illustrated, Volume 1, The Protocols*. Upper Saddle River, NJ: Addison-Wesley, 1994.
14. M. Chiang, S. Low, A. Calderbank and J. Doyle, Layering as optimization decomposition: A mathematical theory of network architectures, *Proc. of the IEEE*, **95**(1): 255–312, 2007.
15. T. Lakshman and U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, *IEEE/ACM Trans. on Networking*, **5**(3): 336–350, 1997.
16. M. Mathis, J. Semke, J. Mahdavi and T. Ott, The macroscopic behavior of the TCP congestion avoidance algorithm, *ACM Computer Communication Review*, **27**(3): 67–82, 1997.
17. J. Padhye, V. Firoiu, D. Towsley and J. Kurose, Modeling TCP throughput: A simple model and its empirical validation, *ACM Computer Communication Review*, **28**(4): 303–314, 1998.
18. K. Ramakrishnan and R. Jain, A Binary feedback scheme for congestion avoidance in computer networks with connectionless network layer, *Proc. ACM SIGCOMM*, 1988.
19. R. Srikant, *The Mathematics of Internet Congestion Control*, Cambridge, MA: Birkhauser, 2003.
20. J. Walrand and P. Varaiya, *High Performance Communication Networks*, San Francisco: CA, Morgan Kaufmann, 2000.

Ao Tang
Cornell University
Ithaca New York

Lachlan L. H. Andrew
California Institute of Technology
Pasadena, California

Mung Chiang
Princeton University
Princeton New Jersey

Steven H. Low
California Institute of Technology
Pasadena, California

VIDEO CONFERENCING AND IP TELEPHONY

INTRODUCTION

In the early 1990s, computer processing power and networking connectivity had advanced enough to allow for the digitizing, compression, and transmission of audio and video. Communicating audio and video over traditional packet-switched networks, however, is harder than traditional data communications for several reasons. First, the amount of data required to transmit video and audio can be significantly higher than their traditional data counterparts. Second, the stream needs to be continuous over time, requiring that the resources be sufficiently allocated to allow for the continuity of the media being delivered. Third, for video conferencing and IP telephony, the end-to-end latency needs to be minimized. This latency includes the capture and compression of the audio and video, compression, transmission, and display on the remote side. Finally, because the data are being streamed, the variation in delay (i.e., jitter) needs to be minimized as well.

Through the 1990s, several efforts focused on standardizing the storage and transmission of digital media emerged. These standards covered a broad range of applications and network assumptions. For example, MPEG-1 was designed for the storage of VHS quality video onto a CD-ROM, whereas MPEG-2 was designed for high-definition digital video applications (1). Other standards such as H.261 and H.263 were designed to enable digital media over telephony-based networks (2). From an internetworking perspective, standards such as Session Initiation Protocol (SIP), H.320, and H.323 were defined to specify how connections for video conferencing and IP telephony were managed.

In the rest of this article, we will provide an overview of digital audio and video formats as well as a discussion of compression algorithms. We will then provide an overview of both video conferencing and IP telephony. Finally, we will summarize where these fields are moving to in the future.

DIGITAL MEDIA BACKGROUND

Sound

Sound is a variation in air pressure that the human ear can detect. The physical parameters of a sound wave involve its frequency and amplitude. The ability to detect such sound depends on the physiology of the ear. For example, humans can typically hear frequencies between 15 Hz and 20 kHz. Cats and dogs, on the other hand, can typically hear frequencies up to 40 or 60 kHz. Sound can be represented digitally through a sampled signal stream. The stream is determined by two primary factors: the sample depth (or the bits required to represent each sample) and the sampling frequency (samples per second). The goal of digitizing sound is to take samples at a rate high enough to capture the highest frequency needing to be represented and to use a large enough sampling depth in order to avoid significant sample distortion. According to Nyquist, the sampling rate needs to be twice the maximum frequency required. As humans are capable of hearing up to about 20 kHz, capturing all audio, a human can hear requires a sampling frequency of at least 40 kHz. For this reason, CD-audio uses a 44.1 kHz with a 16-bit per channel sample.

For computer and telephony applications that require audio, several standards can be used to represent the sound, including the International Telecommunications Union (ITU) G.711 standard and MPEG-audio (from the MPEG-1 audio and video codec). For IP telephony, the primary representation is the G.711 format. G.711 is an international ITU standard for representing sound for a 64-kbps channel. It is a pulse code modulation scheme that uses 8 bits per sample with a sampling frequency of 8 kHz. Thus, the speech signal is limited to a 4-kHz band. Two methods are used. A-law and μ-law differ slightly in their nonlinear transform used to encode the data into 8-bit samples. Both encoding mechanisms use a nonlinear, logarithmic, transform of the input sample space. As a result, the samples are spaced uniformly on a perceptual scale to represent the amplitude.

The compression of audio signals can take several forms. In the G.711 standard, the compression ratio from its samples is fixed to approximately 1.7 to 1. Additional compression algorithms have been developed for telephony applications. These applications include algorithms that perform silence suppression or take advantage of the limitation of human hearing by removing perceptually undetectable sound. In particular, the MPEG audio algorithms (e.g., MPEG audio layer 3, or MP3) remove perceptually undetectable sound and are applicable to a wider range of audio streams, including music.

Video

Digital video consists of a sequence of images, called frames. Digital video can be described by its (*1*) frame rate—the number of frames captured per second and (*2*) the resolution of the images in pixels. Unfortunately, high-quality video requires significant resources. For example, a VHS quality video stream of 352 × 240 pixels at 30 frames per second requires approximately 60 megabits per second to transmit over a network in uncompressed form. Digital video compression algorithms aim to reduce the required bit rate by a factor of 50 to 100.

Digital video compression algorithms take advantage of the redundancy within a frame and between frames of the video. Since the early 1990s, many different video compression algorithms have been developed such as H.261, H.263, Motion JPEG, MPEG-1, and MPEG-2. The ITU and the International Standards Organization (ISO) have standardized the H.26* and the MPEG formats, respectively. In addition, there are proprietary formats such as the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

RealVideo suite from Real Networks, Quicktime from Apple, and the Windows Media Encoding algorithm from Microsoft. For the rest of this article, we will focus on video compression techniques that have been primarily developed and used in video conferencing systems.

Video Compression Standards. Two primary groups are responsible for the development of standardized video compression formats: the ITU and the Motion Pictures Experts Group (MPEG). The ITU is responsible for many of the encoders and decoders (codecs) that are used in the H.320 and H.323 umbrella standards for video conferencing and IP telephony. The ITU group is responsible for the H.261, H.263, and H.264 standards, which we provide brief overviews of here.

- H.261 is a video coding standard for audio and video over multiples of 64 kilobit per second (kbps) channels. The standard, which was intended specifically for interactive video conferencing applications, supports two main resolutions. The Common Interchange Format (CIF) is defined as 352 × 288 pixel video, and quarter CIF (QCIF) is defined for 176 × 144 pixel video. H.261 is intended for communication channels that are multiples of 64 kilobits per second and is sometimes called *p*x64 where *p* runs from 1 to 30. The compression algorithm uses the discrete cosine transform (DCT) as its main compression algorithm. The DCT algorithm transforms small blocks (8 × 8 pixel in size) of the video into the frequency domain, allowing for greater compression efficiency. In H.261, there are two main types of pictures. (*1*) intracoded frames, which are independently coded; and (*2*) predictive coded frames, which are predicted from a previous frame. Finally, block-based motion compensation is used to reduce the bit rate for predictive coded frames.
- H.263 is a video coding format for audio and video that is considered the successor to the H.261 standard. It is similar in format to H.261 but provides better picture quality for the same bandwidth. It was originally intended for bandwidth as low as 20 kbps to 40 kbps but has been applied to larger bandwidth scenarios. It improves the image quality for a given bit rate through half-pixel motion compensation. It also supports additional pixel resolutions, including Sub-Quarter CIF (SQCIF) at 128 × 96 pixel video, 4CIF at 704 × 576 pixels, and 16CIF at 1408 × 1152 pixel resolution. Finally, it provides bidirectionally coded frames, called PB frames, which are similar to MPEG-style P and B frames that we will describe in the compression section.
- H.264 is a newer video compression algorithm from the ITU and MPEG groups. It provides even higher compression efficiency than the H.263 standard through several refinements. Many of these refinements are beyond the scope of this overview.

The MPEG group is a working group of the ISO and the International Electro-Technical Commission (IEC). They are responsible for the MPEG-1, MPEG-2, and MPEG-4 standards. An overview of each is described below.

- MPEG-1 is one of the first standardized video compression formats. In 1988, the Motion Pictures Expert Group gathered a group of companies to standardize the compression of VHS quality video for storage to CD-ROM. The standard, released in 1992, specified the compression of CIF quality video and audio into a 1.5-Mbps stream. As in the ITU video coding standards, the core MPEG algorithms are DCT-based. MPEG has three types of frames (*1*) I-frames that are independently coded frames using a technique similar to the JPEG compression algorithm; (*2*) P-frames that are predictive coded to a previous frame; and (*3*) B-frames that are coded with respect to both a previous and a future reference frame. Compression ratios in the range of 100:1 are possible using MPEG. As an aside, the popular MP3 format is the MPEG-1 Audio Layer-3 compression algorithm.
- MPEG-2 is intended for the compression of TV signals and other applications capable of 4 Mbps and higher data rates, which result in a very high-quality video stream. MPEG-2 is the algorithm that is typically used for DVD format video disks. The underlying algorithms between MPEG-1 and MPEG-2 are very similar. MPEG-2 provides several refinements to deal with the interlaced video signals found in television signals.
- MPEG-4 was originally intended for low-bit-rate applications. One such application is the streaming of video over wireless channels. Through its development, it became a compression format intended for video in general. It has numerous refinements over the previous MPEG formats. It also adds several new features such as primitive media objects that allow for the specification of virtually arbitrary objects, both natural and synthetic.

Fortunately, all of the above compression algorithms from the ISO and the ITU are DCT-based and are fairly similar in their basic structure. In the next section, we will describe a generic DCT-based video compression algorithm. Readers interested in the low-level details of a particular encoding algorithm are referred to the list of references at the end of the article.

A Generic Video Compression Algorithm. In this section, we will describe a basic DCT-based video compression algorithm. The purpose of this discussion is to give an overview of DCT-based video so that we can better describe the issues involved in delivering video over the Internet. We will describe a video compression algorithm that is most similar to the MPEG-1 video standard as it is the most "generic" of the standards above.

The two main areas that compression algorithms can take advantage of are redundancy within a single frame and the redundancy between nearby video frames. I-frames are independently coded video frames. They result in the largest size when compressed but are independently decodable. P-frames are predictive coded from a previous

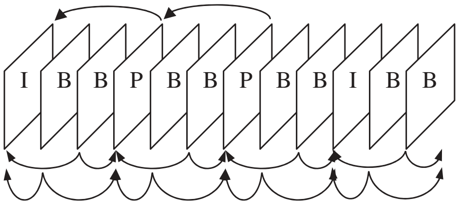


Figure 1. This figure shows the frame dependence and frame pattern that can be found in an MPEG-1 video stream.

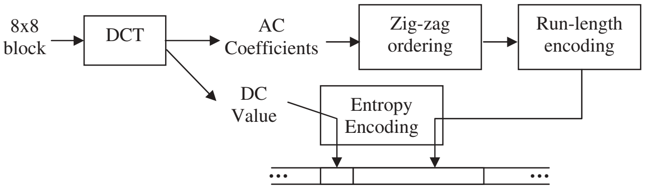


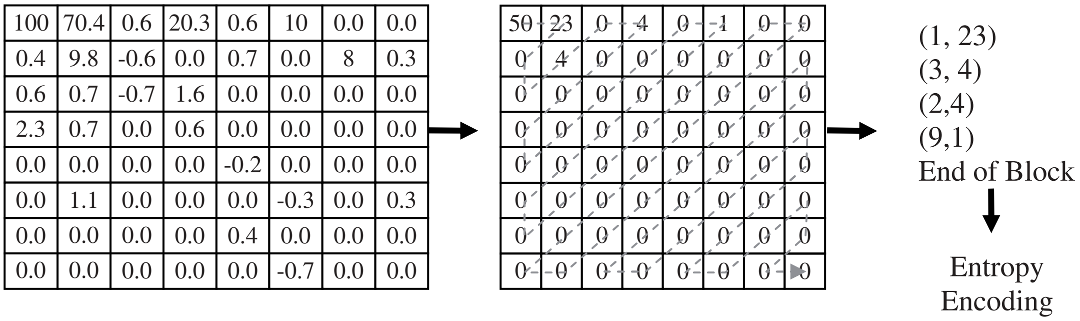
Figure 2. This figure shows the basic steps involve in the coding of each block within a single frame of video.

reference frame. This results in a frame that is considerably smaller than the I-frames but also requires a reference frame to be present in order for it to be decodable. Finally, B-frames are bidirectionally interpolated between two reference frames, one in the past and one in the future. This results in the smallest compressed frames but requires the most computation in order to decode it. The actual ordering of frames depends on the application. For MPEG-1, virtually any ordering of frame types is possible; however, repeated patterns are typically chosen. An example sequence, along with the frame dependence, is shown in Fig. 1.

Within a frame, the data are compressed in several steps. First, the pixels encoded in the red, green, blue (RGB) color space are first converted into the YUV color space, which represents the luminosity channel (grayscale) and two chrominance channels that add color. Next, the frame is split into 16 × 16 pixel regions called macroblocks. Each macroblock is then further subdivided into 8 × 8 blocks. The purpose of this is that the U and V channels are typically further subsampled because the human eye cannot discern small differences in the chrominance channels. In general, each 16 × 16 pixel U and V block is represented by one 8 × 8 pixel subsampled block, respectively. Once the frame is divided into its relevant blocks, the blocks are then compressed.

For each block within a macroblock, several additional steps are taken. An overview of the basic steps is shown in Fig. 2. Each block is transformed into the frequency domain through a DCT. The unique property of this transform is that areas of relatively constant color can be represented by only a few coefficients, rather than the 64 unique pixel values in the spatial domain. After the DCT transform, the DC value (or average value) for the entire block is in the upper left-hand corner. The rest of the coefficients are called the AC coefficients. If all coefficients are 0, then this means the entire block can be represented by a solid 8 × 8 block of a single value. The coefficients are then quantized. Quantization accomplishes two main functions. First, it converts the floating point values back into integers. Second, it reduces the number of nonzero coefficients that need to be represented by dividing each coefficient by a predefined table look-up and a user-defined quantization value. Finally, the coefficients are zigzag ordered, run-length encoded, and then entropy encoded (typically Huffman encoding). The steps for the last part of the compression are shown in Fig. 3.

For coding P- and B-frames, each macroblock has an additional block-based motion compensation algorithm applied to it. The goal of the motion compensation algorithm is to find an area within the reference frame that is the closest match to it. Although a pixel by pixel comparison within the reference frames might be computationally prohibitive, several heuristics have been proposed and put to use that make finding reasonably close matches fairly quick. These heuristics include performing sampled searches and limiting the area that the reference frame is searched for a match. For the P-frames, the previous reference frame (either an I- or P-frame) is searched for the match. The closest match is then used as a prediction for the blocks to be encoded. The goal is to have a prediction that requires very little correction, which results in many coefficients in the transform to be close to 0. For the B-frames, both the previous reference frame and a future reference frame are used to find a match. Furthermore, the B-frame allows for the forward and reverse matches to be interpolated in order to predict the block to be encoded. Clearly, B-frames require the buffering of several frames in order



100	70.4	0.6	20.3	0.6	10	0.0	0.0
0.4	9.8	-0.6	0.0	0.7	0.0	8	0.3
0.6	0.7	-0.7	1.6	0.0	0.0	0.0	0.0
2.3	0.7	0.0	0.6	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	-0.2	0.0	0.0	0.0
0.0	1.1	0.0	0.0	0.0	-0.3	0.0	0.3
0.0	0.0	0.0	0.0	0.4	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	-0.7	0.0	0.0

50	23	0	4	0	-1	0	0
0	4	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(1, 23)
(3, 4)
(2,4)
(9,1)
End of Block

Entropy Encoding

Figure 3. This figure shows the basic process of quantization and run-length encoding. Coefficients in the upper left are quantized with larger values. For the run-length encoding, the run represents the number of zeros until the next coefficient.

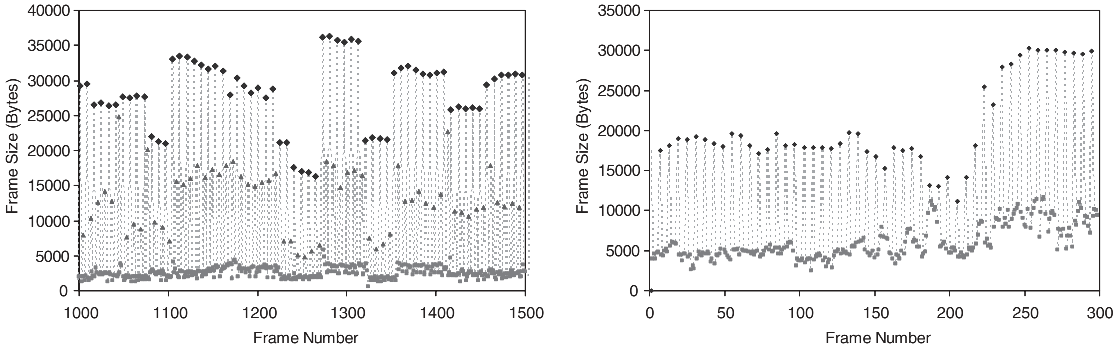


Figure 4. This figure shows the result of compression of video into MPEG and H.263. On the left, in MPEG, there are three distinct frame types (I-frames are diamonds, P-frames are triangles, B-frames are squares). On the right is an H.263 sequence with I-and P-frames (P being smaller in size).

for the forward reference frame to appear at the encoder. This additional delay may not be acceptable for some low-latency applications.

Because the compressed stream is heavily dependent on the data, the actual compressed frame sizes tend to vary considerably over time. As a result, this variability can cause strain on the network that needs to deliver each *frame* of video with low latency as well as fairly constant delay jitter. As an example, we have graphed the result of applying MPEG compression to a sequence of frames. They are shown in Fig. 4 for a constant quality compressed video stream. We will describe the impact of the video requirements later in the chapter.

Basic Multimedia Transport

There are several ways in which video conferencing and IP telephony data can be transmitted between two points. Transmitting data over a telecommunications channel (e.g., ISDN or the plain analog telephone network) is relatively simple, requiring that the application allocate as many channels as necessary. As the network is circuit switched, transmitting data over such networks is relatively easy and has guaranteed service. The main disadvantage of using the telephony network is the high cost associated with using such a service. The alternative to this is to use a data network such as the Internet to transfer the session.

The primary transport mechanisms that are in use for the Internet are the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) over the Internet Protocol (IP). TCP provides a congestion-controlled (network-friendly) delivery service that is reliable. Thus, nearly all data traffic such as Web traffic or file transfers occur over the TCP protocol. There are two main disadvantages of using TCP for video conferencing and telephony networks. First, because TCP attempts to fairly share network bandwidth while maximizing throughput, the bandwidth from the application perspective is bursty over time. Second, TCP is a reliable transport protocol. When the network drops a packet because of congestion, an application-layer delay will be induced because TCP will retransmit the lost data. UDP is a lighter weight protocol that simply transmits packets. Whether the packet arrives at the receiving side is left up to the application layer to manage. For video conferencing and IP telephony, this has several implications. First, lost data may impact the ability to display or play-back the data being transmitted. Upon the loss of a packet within a compressed video stream, the application will not be able to display the frame or frames that were in the packet. Furthermore, all other packets will need to be discarded until the application can find an area within the stream to resynchronize itself with the video stream (e.g., the start of new frame). Second, UDP is not sensitive to the load within the network. As a result, it may overrun the network with data. For IP telephony applications, this may not be that large a concern as the data rate for IP telephony is relatively small. For video conferencing, this becomes a larger concern.

For managing the real-time nature of audio and video, the Real-time Transport Protocol (RTP) and the Real-time Transport Control Protocol (RTCP) can be used. Typically, these protocols are used in tandem to deliver streaming data over the best-effort Internet. RTCP is the control part of the protocol that provides feedback information to the applications. For example, it provides feedback on the quality of the data delivered such as packet loss or network jitter. In addition, it provides for intrastream synchronization. RTP is the transport mechanism for real-time data that is typically built on top of the UDP/IP protocols. It provides primitive network information to the application such as sequencing of packets and time-stamps for media synchronization.

MULTIMEDIA CONFERENCING AND TELEPHONY SESSION MANAGEMENT

For interactive conferencing and telephony, there are two primary protocols that are in use for data networks: H.323 and the SIP.

H.323

H.323 is an ITU standard for packet-based multimedia communications that was released in 1996. It is, perhaps, the most widely deployed protocol for video conferencing and IP telephony. The H.323 protocol is used for several interactive applications, including the popular Polycom and Microsoft NetMeeting products. H.323 encompasses several standards, some of which are mandatory in H.323 implementations and others that are optional. For video conferencing and telephony, H.323 must implement H.261 and G.711, for video and audio, respectively. Other standards such as H.263 are optional. H.323 defines several entities that can participate in interactive conferencing and telephony. They are as follows:

- Terminals—Terminals are the end devices that the users use. These devices include telephones, video phones, PCs running video conferencing software, and voice mail systems.
- Multipoint Control Units—Multipoint control units (MCUs) are used to manage multiway video and audio conferences. For video conferencing applications, MCUs take the individual incoming videos from the participants and mix the streams together to create a mosaic of the videos. As a result, MCUs add delay to the video conference and are expensive because of the hardware cost necessary to mix video in real time. MCUs are, however, necessary for low-latency multiway video conferencing.
- Gateways—Gateways are used to allow H.323-compliant systems to interact with other systems. For example, a gateway can be used to cross between H.323- and SIP-based communications. Additionally, they can be used to bridge between an H.323-based network and the regular voice telephony network.
- Gatekeepers—Gatekeepers, although not necessary to use H.323, can act as a manager for H.323 sessions. They can provide address translation from local addresses to IP addresses. Gatekeepers can also perform bandwidth management, authentication, and billing.

For the actual transmission of data, H.323 specifies several standards for the encoding of audio, video, and data. As mentioned, H.323 requires the support of H.261 streams for video and G.711 for audio. In addition, there are many optional components such as having H.263 as a video codec. In more recent versions of H.323, support for H.264 video streams has been added.

Session Initiation Protocol

The SIP is a protocol standardized by the Internet Engineering Task Force (IETF) for the transmission of teleconferencing and multimedia communications over the Internet. SIP was introduced in 1999 in IETF RFC 2543 and later updated in 2002 in IETF RFC 3261 (3). SIP, like H.323, is an umbrella protocol that provides the signaling necessary to bring video, audio, and data communications together for interactive applications (4). SIP is more open in that it does not require any particular media compression format to be implemented. As a result, its use may include other interactive applications beyond audio and video. Its main functions include the negotiation and initiation of sessions between two endpoints as well as connection maintenance and termination. SIP is a text-based protocol allowing for simple debugging and easier interoperability.

SIP is a peer-to-peer architecture, where the endpoints are called user agents. The endpoints can be SIP-enabled telephones or PCs. Gateways can also be used to provide translation between various entities (e.g., format translation or between different types of networks).

MANAGING THE DATA IN VIDEO CONFERENCING AND TELEPHONY

In the rest of this article, we briefly describe some issues with managing the actual compressed data within video conferencing and telephony applications.

Voice Over IP

Although voice can be represented with relatively few bytes when compared with video, it is still possible to reduce the amount of data required to transmit voice over IP further. The Algebraic Code Excited Linear Prediction (ACELP) algorithm can be used to further compress the audio. This has been specified in the G.723.1 standard. Other techniques involve silence suppression, which has been applied in the regular telephony network.

Sending voice over IP requires the management of two key parameters: end-to-end delay and delay jitter. Both of these parameters can impact the ability of two users to interactively carry on a conversation. End-to-end delay is the amount of delay required for the actual transmission of bits across the network (including all queuing delay within the routers of the network). Typically, the delay is correlated to the number of routers that the packets must go through. Overcoming network delay that causes unacceptable application-layer performance requires dedicated network lines, which is typically an expensive operation. Fortunately, the delay within the Internet is typically not that large.

Delay jitter, or the variation in end-to-end delay, is more problematic to handle in general. Buffering can be used to mitigate delay jitter. Unfortunately, the variation in delay continues to vary over time. Tuning the buffer delay to handle the maximum delay jitter will cause unnecessary delay at the other times. Tuning the buffer delay to something too small, however, will cause excessive packet loss and drops in the audio. Techniques like queue management can be used to actively adapt the amount of buffering at the client to mitigate the effects of delay jitter for audio applications (5).

Video Conferencing Over IP

As mentioned, delivering compressed video over packet-switched networks is even more complicated than delivering voice over IP because of (*1*) the variability in frame sizes from the compression algorithms, (*2*) the larger size of the video relative to the audio channel, and (*3*) the variability in both network delay and delay jitter. In the remainder of this section, we will briefly highlight some mechanisms that one can use to deliver high-quality video over the Internet.

Depending on the choice of frame types that are used, a small amount of buffering can be used to smooth the video stream a little so that the extra bandwidth required to deliver I-frames can be amortized across several smaller predictive coded frames. Unfortunately, such smoothing is very sensitive to delay as each frame that is buffered requires an additional one thirtieth of a second. This is partially why there is a noticeable delay in most video conferencing applications.

In addition to buffering of data for video, one can employ techniques that actively manage the video data itself by adapting the video to the underlying network resources. Adaptation can happen either at encode time, where the video codec estimates the available network bandwidth and codes for it, or can be at transmission time, where the sender of the video can drop some data in order to make it fit within the available network resources. For the former, the network bandwidth needs to be actively monitored in order to provide feedback to the encoder. The encoder, in turn, can adjust the quantization value, which forces more coefficients to zero, making the video stream smaller. The net effect, however, is that the quality of the video will be lower. For senders that code the video and drop data in order make it fit within the available network bandwidth, typically layered encoders are used. Standards such as MPEG-2, MPEG-4, and H.264 have been designed to allow for fine-grain scalability on-the-fly. Layered encoders work by encoding the video stream into multiple layers consisting of a “base-layer” and “enhancement layers.” Sending a higher priority “base-layer” that encodes a basic quality video stream first allows a minimum quality of video delivered to the client. The delivery of each enhancement layer after that will gradually continue to raise the quality of the video. Typically, most encoders use no more than four layers.

To support layered transmission, encoders use one of two mechanisms. First, the encoder can use a lower pixel resolution as a base layer and as an enhancement layer, which provides more details that raise the quality of the video. Second, the encoder can split the coefficients between the various layers. For example, the encoder can encode the lowered numbered coefficients in the zigzag ordering in one layer and have the enhancement layer with the remaining coefficients. Thus, the enhancement layer adds the higher frequency details to the image. Finally, the encoder can encode all the higher order bits of the coefficients into the base layer. Each enhancement layer can then add more of the lower order bits in succession. Obviously, using enhancement layers will reduce the coding efficiency of the compression algorithm but, nevertheless, make them more flexible for network adaptation.

Even with buffering and layered coding, it is entirely plausible that packets will be dropped within the network. Removing data from a compressed stream can cause significant artifacts within the display, particularly if data are lost in a reference frame on which other frames will depend. In such an event, error correction techniques can be applied. Several such techniques can be used. First, the frame that has any data lost can just not be displayed. Second, for macroblocks that are lost, the macroblocks from the previous frame can be reused. Third, one can use the previous motion vector to offset a previous macroblock into the new frame. Finally, one could interpolate the data from nearby regions within the current frame. Error recovery techniques, however, are not a replacement for streaming and adaptation algorithms.

BIBLIOGRAPHY

1. D. Le Gall, “MPEG: A Video Compression Standard for Multimedia Applications”, *Communications of the ACM*, Vol. **34**, No. 4, pp. 46–58, April 1991.
2. Ming Liou, “Overview of the px64 kbit/s Video Coding Standard”, *Communications of the ACM*, Vol. **34**, No. 4, pp. 59–63, April 1991.
3. Internet Engineering Task Force (IETF) Request for Comments (RFC) 3261, “SIP: Session Initiation Protocol”, June 2002.
4. Josef Glasmann, W. Kellerer, “Service Architectures in H.323 and SIP – A Comparison”, White Paper, Munich University of Technology (TUM), Siemens AG, Germany.
5. D. L. Stone and K. Jeffay, “Queue Monitoring: A Delay Jitter Management Policy”, in *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 149–160, November 1993.

Wu-chi Feng
Portland State University
Portland, Oregon

WIDE-AREA NETWORKS

INTRODUCTION

Wide-area networks (WANs) can be defined roughly as the computer networks that cover a broad geographical area. These networks may be considered as the opposite of local-area networks (LANs) and metropolitan-area networks (MANs). LANs refers to the networks within a single office, building, or campus, wheres MANs refers to the networks within a single metropolitan area.

Although no specific boundary definition distinguishes MANs from WANs, in the real world a WAN usually covers multiple locations in a country or, in many cases, in multiple countries. WANs are usually used to connect LANs and MANs in different locations. The most typical WAN is the Internet. Another example would be an intranet for a large enterprise that has multiple geographically distributed branches.

The size difference between WANs and LANs/MANs necessitates some fundamental differences in both the network hardware and software. Communications over a WAN travel much longer distances, and, although it is not necessarily true, usually many more hosts and pieces of network equipment comprise WANs. WANs need to use switches and, more importantly, routers to connect different LANs and other networks. In contrast, LANs usually do not need routers. Furthermore, the networks and networking equipment in a WAN can all belong to different organizations; thus, stricter policies may be applied in a WAN.

WANs, like other networks, are usually organized in a layered architecture so that technology changes in one layer will not affect functionalities of other layers. For example, a transport protocol, such as TCP, can work on top of many different physical networks.

In the rest of this article, we will describe WANs in detail for each layer in the five-layer TCP/IP reference model. Brief background knowledge is introduced first.

BACKGROUND

TCP/IP Model

The TCP/IP network reference model, also known as the Internet reference model, is a depiction of the layered model used in the current Internet. Another well-known model is the OSI reference model, which is also a layered architecture, but the OSI model has two additional layers that do not exist in the current Internet.

The TCP/IP model has five layers (Fig. 1). From bottom to top they are the physical layer, data-link layer, network layer, transport layer, and application layer. In network terms, they are also called Layer 1 to Layer 5, respectively.

The physical layer takes care of the encoding and decoding of signals (bits) that represent the data passed down from the upper layers. This layer is responsible for moving the data along the physical network wires and equipment.

The data-link layer transmits packets, or a group of bits, between two hosts. Data are packed by using different schemes, such as Ethernet (1) or ATM (asynchronous transfer mode) (2). Switches work in this layer so that packets can be transferred to different network segments.

The first two layers are often mixed together. Examples of hardware/protocols working in these two layers include Ethernet, SONET (synchronous optical networking), Wi-Fi (3), PPP (point-to-point protocol) (4), and so on. Ethernet and SONET are usually used for high-speed networks, WiFi is used for wireless networks, and PPP is used for home dial-up networks or DSL (digital subscriber line).

The network layer addresses two issues: addressing and routing. In other words, hosts and certain pieces of network equipment are assigned a unique network address; then packets can be routed to a certain destination. Routing allows the network to be extended to a much broader area. Protocols in this layer include communication protocols such as IP (Internet protocol) (5); routing protocols such as OSPF (open shortest path first) (6), IS–IS (intermediate system to intermediate system) (7), and BGP (border gateway protocol) (8); and address-resolution protocols such as ARP (address resolution protocol) (9).

The transport layer is responsible for transferring application data on an end-to-end basis. It provides direct support for application-data transfer. Congestion control is one of the most important schemes in the layers. It allows all connections to coexist and to realize their fair and efficient share of the network bandwidth. The two core transport protocols are TCP (transmission control protocol) (10) and UDP (user datagram protocol) (11); over 90% of the Internet traffic is carried by TCP.

Finally, applications such as worldwide web (WWW), file transfer protocol (FTP), and e-mail work in the application layer. Applications define their own specific protocols and send data using the transport protocols.

At the sender side, applications pass their data down to each lower layer, which adds their own packet header in front of each packet. The data eventually is transmitted to the receiver side via the physical link. At the receiver side, the data packet is passed up, and each layer interprets and removes the proper packet headers.

End-to-End Principle

Another principle that plays a significant role in the Internet design is the end-to-end (E2E) principle (12). It has been applied to the Internet design since the early 1980s. The E2E principle argued that systems tend to require end-to-end processing to operate correctly, in addition to any processing in the intermediate system. It pointed out that most features in the lowest level of a communications system have costs for all higher-layer clients, even if those clients do not need the features, and are redundant if the clients have to reimplement the features on an end-to-end

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

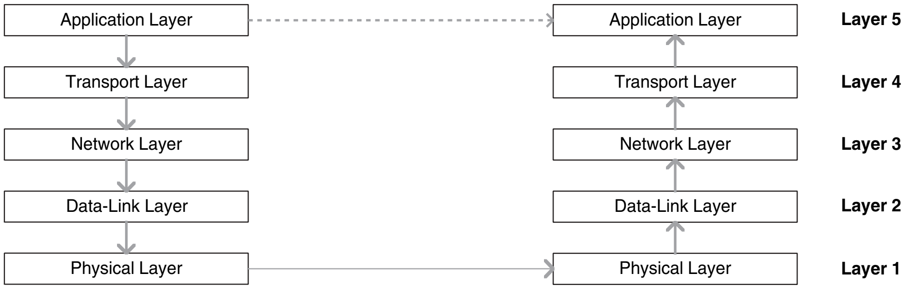


Figure 1. TCP/IP-layered model.

basis. As a result, the Internet can be described as a "dumb, minimal" network with smart terminals.

For example, IP is a very simple protocol that transmits packets only with best-effort delivery, whereas TCP adds reliability control, flow control, and congestion control to the data transfer. All of these schemes are done at the end host.

In addition to the layered architecture and E2E principle, more Internet design guidelines can be found in Ref. 13.

Standardization

For network equipment and software to communicate with each other, the network hardware and software must follow the same standard, usually called a protocol in network terms. Most protocols are formed by two organizations. IEEE (Institute of Electrical and Electronics Engineers) focuses on lower layers (physical and data-link layer), whereas IETF (Internet Engineering Task Force) focuses on network, transport, and application layers. IETF publishes Internet standards as RFCs (Request for Comment). Other standardization committees include ITU (International Telecommunication Union) and ANSI (American National Standards Institute).

INFRASTRUCTURE (LAYER 1/2)

The bottom two layers set up the infrastructure of a computer network. In the real world, they are often treated together as one single layer.

Today, WANs use optical fiber as the major physical signal transfer media. Fiber is less expensive and provides much faster speed (compared with traditional media such as copper). The Internet2 (a WAN that connects most educational institutes in United States) just realized 100 gigabits per second (Gb/s) backbone transfer speed.

Fiber/WDM

Optical fiber is the major physical media used in WANs. The light signal is reflected by the cladding layer outside the fiber core until it reaches the other end (Fig. 2).

The reflection of light within the fiber core is affected by the diameter of the core. The larger the core is, the more reflections are needed for the light to pass through. Fiber with a significant larger core (compared with wavelength) is called multimode fiber; the light may have many different paths because of the input angle.

However, if the core is thin enough (the diameter is less than 10 times that of the wavelength), the above geometric analysis cannot be used, but electromagnetic analysis is in effect. In this case, the light can traverse only one mode in the fiber. Such fibers are called single-mode fibers.

Single-mode fiber is better in conserving the light power and thus can pass the light through a greater length. However, it is also more expensive to make. Overall, fibers are cheaper than traditional media, such as copper, and can transmit signals much farther without the use of relay equipment. For example, 1 Gb/s (1 gigabit = 1,000,000,000 bits) Ethernet copper cable can only extend to 100 meters, whereas 1 Gb/s Ethernet single-mode fiber can extend to 6000 meters.

Signals transmitted in the fiber can be differentiated by the different wavelengths of the light. Therefore, a single fiber can be used to carry multiple signals at the same time. This use is called wavelength-division multiplexing (WDM). WDM significantly increases the capacity of a single fiber. Greater capacity increasing can be done by simply upgrading the WDM equipment at the end of the fiber.

Dense wavelength-division multiplexing, or DWDM, is common today. DWDM can support wavelengths that are closer to each other, thus it supports more data channels. For example, some DWDM equipment can support up to 160 channels (which means up to 160 separate connections can be supported by a single fiber).

Note that computers and switches do not recognize optical signals directly yet. (All-optic switches are in experimental stage.) Light signals have to be converted to electronic signals for processing, which limits the network speed. Currently 100 Gb/s is being deployed, whereas 10 Gb/s is becoming common in WANs.

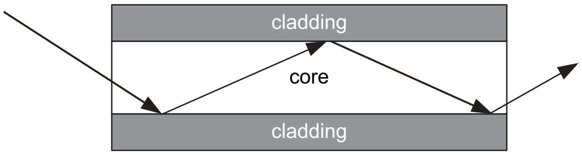


Figure 2. Light passing through optical fiber.

Tiers of Network

A natural question is: Who owns the fiber around the world? Although fiber is a relatively inexpensive solution for a WAN, it is still beyond the financial ability of most single organizations (business or government) to deploy fibers over a wide area. Because WANs usually span international borders, not only economical but also political issues are involved.

In the real world, the hardware infrastructure of WANs is actually maintained in a "tier" hierarchical structure (Fig. 3). The most powerful stakeholders use each other's network by a reciprocal agreement. More precisely, this agreement is a settlement-free peering agreement. The networks owned by such organizations are called Tier 1 networks. Traffic from one Tier 1 network can pass through another Tier 1 network without paying any fee. In return, it allows traffic from other Tier 1 networks to pass through for free. Obviously, not many Tier 1 networks exist in the world. Other smaller and less-powerful networks that own a segment of the network but have to pay some Tier 1 networks to pass their traffic are called Tier 2 networks. Both Tier 1 and Tier 2 networks are usually by large national or international networking companies. In fact, Tier 2 networks may purchase services from multiple Tier 1 networks, and it is possible that some Tier 2 networks are larger than some Tier 1 networks.

Similarly, if a network is operated by solely purchasing services from either Tier 1 or Tier 2 networks, without any settlement-free peering, then it is called a Tier 3 network. Typical Tier 3 networks are those local ISPs (Internet service providers, e.g., DSL, cable, or campus network) that cover a small community.

In network terms, each region in Fig. 3 is also called an autonomous system (AS), defined as a collection of IP networks and routers under the control of one entity.

Switch and VLAN

Above the physical fiber, switches work at Layer 2, and they forward packets to destinations according to the physical MAC (media access control) address.

Note that sometimes the term "switch" is used to name interconnection devices in other layers, too. However, such devices working at Layer 1 are usually called hubs and Layer 3 switches are usually called routers to differentiate them from Layer 2 switches. Hubs, switches, and routers have different functionalities as they work at different layers. In this article, we use the term "switch" only for Layer 2 interconnecting devices.

A switch learns the MAC addresses of those switches and hosts that connect to it and forwards the packets to the destination with the specified MAC address. This method is not a scalable because as the MAC-address table of a switch is limited in size. For the same reason, switches are usually only used in LANs.

In technical terms, a Layer 2 switch covers a broadcast domain. Each switch is capable of broadcasting a packet to every connected switch or host. When the packet reaches a switch, it checks the hardware address (MAC) and either forwards the packet or drops it (if no MAC address is associated with the switch).

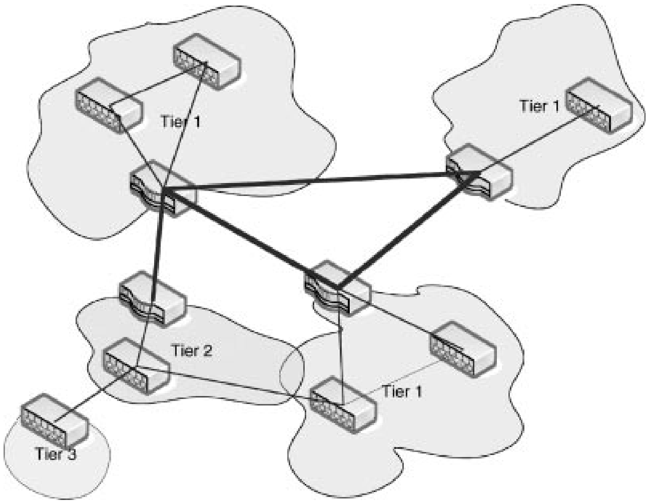


Figure 3. Tier hierarchy of the Internet.

However, the forwarding scheme described above can cause an infinite loop if the switches are connected in a loop. The problem is solved using the spanning tree protocol (STP) (14). STP forms a logical tree structure among all connected switches to avoid loops in the forward path.

An important technology called virtual LAN, or VLAN, allows networks to expand to wide area and to form a WAN by using only switches. VLAN allows multiple network segments to be grouped together even if they are physically located in different places and belong to different LANs.

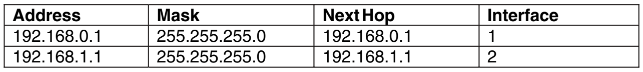
Originally VLANs were created to separate a physically connected network into several independent logical networks to help management and security. For example, the LAN in a campus can be separated into VLANs for different departments (even though offices of the same department may not be physically located together). Without a VLAN, this separation has to be done by routers, with additional cost and performance overhead. This purpose remains the major purpose of the VLAN today.

Although it is possible to use VLAN to construct a WAN, such WANs are small ones and usually only used as an Intranet for a single organization. Wide-area VLANs are usually used inside a single organization to extend the logical network to their branches located in different places. For general public WANs, routers are used for much greater flexibility and scalability.

IP AND ROUTING (LAYER 3)

Routers are usually used to interconnect network segments to construct a WAN, such as the Internet. Whereas Layer 2 switches use a physical address (MAC) to match the destination by broadcasting the message to all adjacent switches or hosts, routers, which are sometimes called Layer 3 switches, use an IP address to locate a single path to the destination by looking up a routing table that is updated by routing protocols.

IP addresses are assigned hierarchically to each organization and its subdepartments; thus, the router knows



Address	Mask	Next Hop	Interface
192.168.0.1	255.255.255.0	192.168.0.1	1
192.168.1.1	255.255.255.0	192.168.1.1	2

Figure 4. Routing table.

that all packets with destinations within a specific IP range can be forwarded to the same next hop. Otherwise the router would need to maintain too many entries for it to handle. Currently, version 4 of IP uses 32-bit addresses, which are usually represented by four numbers between 0 and 255 (*a.b.c.d*). For example, a university uses IP address 192.168.X.X, where X can be any number between 0 and 255. The math department uses 192.168.0.X, the English department uses 192.168.1.X, and so on. For more information about IP addressing, see Ref. 15.

To route a packet, each router maintains a routing table that tells the next hop of a packet which destination falls into a specific IP range. When a packet comes in, it checks the destination address against each entry in the routing table (Fig. 4).

For example, in Fig. 4, all packets sent to the math department 192.168.0.X (192.168.0.1 masked by 255.255.255.0, results which in 192.168.0.X) will be sent to the router in math department 192.168.0.1 via the physical router interface 1.

The algorithm used to update the routing table is called a routing algorithm. Two kinds routing algorithms occur: distance-vector protocol and linked-state protocol. In distance-vector algorithms, each link/hop is assigned a cost. The algorithm computes a route of hops with the smallest sum of cost based on the Bellman–Ford algorithm (16). Each router periodically sends its routing cost information to all of its neighbors; thus, its neighbors can update the routing table according to the new cost information.

In link-state algorithms, each router broadcasts to the whole network its own connectivity information. Thus each node will obtain the map information (a graph) of the whole network. The routing table can be computed from the graph by finding the shortest path, for example, using Dijkstra's algorithm (17).

Because the Internet is organized through many autonomous systems (AS) (Fig. 3), intra-AS and inter-AS routing have different requirements and limitations. Interior gateway protocols (IGPs) are used inside an autonomous system, whereas exterior gateway protocols (EGPs) are used between autonomous systems. Roughly speaking, IGP seeks to optimize a complete autonomous system; thus, its coverage is limited because of the limitation of computation power on the router and the number of messages generated by the protocol. In contrast, EGPs work on only representative nodes from each autonomous system.

Today, the most common routing algorithms are OSPF, IS–IS, and BDP. Both OSPF and IS–IS are link-state algorithms, and both belong to IGP. In contrast, BGP uses a distance-vector algorithm, and it belongs to EGP.

TRANSPORT PROTOCOL (LAYER 4)

Transport protocols provide end-to-end data-transfer service for the applications. They provide data-transfer functionality to applications without knowing the intermediate routers and switches, which are taken care of by Layer 3 and Layer 2, respectively. Transport protocols can be described by the following characteristics:

Connection-oriented or connection-less. Certain transport protocols need to set up a virtual circuit for data transfer between two end hosts. The existence of the connection allows easier control of reliability, security, and traffic. In contrast, some other protocols do not set up a connection, and an end-host entity can send a message to any destination at any time.

Streaming or messaging. Transport protocols also differ at the semantics of data delivery. Some protocols treat the application data as an infinite stream, and the receiver side can read at any length. The other class of protocols, namely messaging, conserves the boundary of each application buffer, and the receiver side will be able to retrieve the original buffer one at a time.

Reliability. Data can be delivered either completely intact or simply in a best-effort manner. Some protocols allows user-defined partial reliability. In this mode, only the data that can be delivered within a specified time are guaranteed full reliability; otherwise, it will be discarded. Note that when data is not delivered reliably, the application messages can be delivered out of order.

Congestion control. Transport protocols usually work on shared networks; therefore, they need to adjust the data sending rate to realize fairness and efficiency of the network bandwidth usage. Different protocols may use different congestion-control algorithms, whereas some protocols do not have any congestion control at all. Recall that transport protocols work in an end-to-end manner, and no single point in the network coordinates traffic from different flows (i.e., flows do not know the existence of each other). This agreements makes congestion control probably the most difficult part in developing transport protocols.

TCP and UDP

TCP and user datagram protocol (UDP) are the two core protocols of the transport layer. Over 90% of Internet traffic is carried by TCP because it is reliable and because it has a congestion-control scheme to use the network efficiently and stably. UDP is used in a certain area only when reliability is not required or when applications have their own reliability control.

TCP is the *de facto* transport protocol on the Internet. It is a connection-oriented, reliable, streaming protocol. TCP is used for WWW/HTTP, e-mail, FTP, and many other Internet applications.

For TCP, a connection needs to be established between two nodes before the data transfer. The user buffers are packed into segments (packets). The receiver sends back acknowledgments for received packets. A loss report is sent back to the sender if packet loss is detected, thus the sender will retransmit the lost packets. In addition, a retransmission timer is used in case no acknowledgments or loss reports are received. In this way, TCP guarantees data reliability. However, TCP does not keep the boundaries of the application buffers (i.e., TCP works in data-streaming mode).

TCP has two important schemes to maintain the stability of the network. Flow control is used to avoid overwhelming packets to overrun the receiver. The receiver acknowledges the flow window size (the available space for incoming data) so that the sender will never send more data than the flow window size.

Whereas flow control is used to maintain the stability of end hosts, congestion control is used to maintain the stability of the complete network. Basically, TCP starts sending data at a low initial rate and slowly increases the sending rate if no packet loss is detected. However, once a packet loss happens, TCP will halve the sending rate and increases again. This scheme is the AIMD (additive increasing multiplicative decreasing) scheme used in TCP congestion control.

Figure 5 depicts the change in sending rate of a single TCP flow. The algorithm had been working fine until recently when the inexpensive optical fiber greatly increased the network bandwidth. The recovery time after a packet loss (during which time the sending rate increases from the lowest point to the highest) depends on the bandwidth-delay product (BDP), which can be very large in WANs (e.g., 10Gb/s network between NYC and Tokyo with more than 200ms round-trip delay). Thus, TCP does not work well on such networks. New TCP algorithms are being investigated, including HighSpeed TCP (18), CUBIC TCP (19), and Compound TCP (20), just to name a few.

UDP is a connectionless, unreliable, messaging protocol. Applications can send data to any destination without setting up a connection. This feature allows UDP to send data simultaneously to multiple destinations, which is described by the term “multicast,” in contrast to the “unicast” mode in which data has a single determined destination. However, multicast in WANs is often limited or blocked by routers to avoid data flooding.

Although data transferred over UDP is unreliable, sometimes it is a desirable feature because reliability requires additional mechanisms such as acknowledging, which leads to additional delay in data transfer. Some applications, like streaming media, can tolerate limited packet loss, but cannot tolerate long delays. Finally, messages transferred over UDP can be conserved so that the receiving side knows the boundaries between messages. Applications will be able to read a whole message each time, which is a convenient feature.

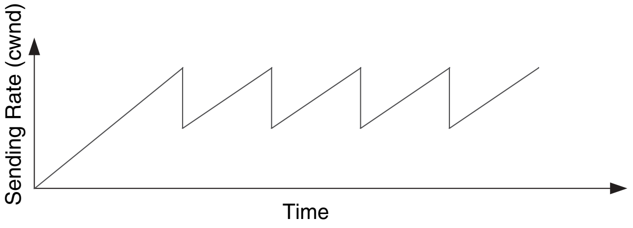


Figure 5. AIMD.

New Transport Protocols

The core transport protocols were designed for general purposes, and both TCP and UDP have been working for several decades. The rapid growth of the Internet has enabled a wide variety of applications (although, it can also be argued that these emerging applications stimulate the Internet expansion), and it is not shocking that neither TCP nor UDP may fit the requirements of some new applications. Therefore, recently several new transport protocols have been proposed.

The datagram congestion control protocol (DCCP) is a message-oriented protocol (21). DCCP can be regarded as a somewhat-enhanced version of UDP. In contrast to UDP, DCCP applies congestion control on the unreliable messaging delivery. Various congestion-control schemes are deployed in DCCP. However, the new feature comes with a cost; DCCP requires connection setup, and hence multicast is not supported, which means that it cannot replace the connection-less UDP. DCCP is designed for certain applications such as streaming media and Internet telephony, which require timed delivery of each message.

Whereas DCCP is somewhat related to UDP, SCTP (stream control transmission protocol) (22) can be compared with TCP. SCTP provides reliable message streaming, instead of the byte streaming of TCP. That is, SCTP is a messaging protocol. In addition, SCTP also supports parallel streams in one single connection. These streams can deliver message independently so that packet loss in one stream will not block others. (In TCP, when packet loss occurs, subsequent data cannot be delivered until the packet loss is recovered.) Furthermore, SCTP also supports multi-homing, and one SCTP socket can be bound to multiple IP addresses of the end which hosts, which enables transparent failover between redundant network paths. Finally, transport protocols can also be built at the application layer on top of UDP. UDT (UDP-based data transfer protocol) (23) is such an example. UDT is a high-speed data-transfer protocol that targets a fair and efficient usage of high-speed long-distance links, a situation that applies to many WANs today. UDT solves TCP’s efficiency problems described in Fig. 5. In addition, because it is at the application level, it does not require changes in operating systems to get deployed.

APPLICATION (LAYER 5)

The application layer hosts most of the Internet applications, including WWW, e-mail, FTP, Telnet, SSH, streaming multimedia, and so forth.

Applications define their own protocols, pack the application data, and pass it into lower layers. Each layer below adds their own packet header and eventually delivers the data at the physical layer (see Fig. 1). At the other end, the data is passed from the lower layers to the upper layers, and each layer's own packet header is removed. The application will receive the data and interpret it according to the application-specific protocol.

Applications on WANs often differ from those usually running in LANs. In addition, WAN applications are usually more difficult to write because WANs are less reliable, are less predictable, and have less available bandwidth.

For example, it is fairly easy to move a 1 Gb file between two hosts in the same LAN, but it may take days to move the same file between Asia and America. Actually, certain applications address such file-transfer problems in WANs. Content distribution network (CDN) (24) is an example.

For similar reasons, other applications that work well in LANs, such as database systems (DBMS), may not have good performance over WANs.

WWW

We describe the most popular application, WWW, as an example of Internet applications. A user usually types a URL into the web browser to go to a website. The URL is translated into IP address by another application-layer service called domain name system or DNS (25). DNS is a hierarchical service that translates domain names (URL) into IP addresses (recall that IP addresses are also assigned to organizations in a hierarchical manner).

After the web browser obtains the IP address of the web server, it sets up a TCP connection to the web server listening on the IP address. The subsequent process is interpreted by HTTP (hypertext transfer protocol) (26).

The web browser first will send out an HTTP request message to the server for a specific page. The server then responds to the request by either sending the requested page back or sending error information (e.g., 404 Page Not Found). Web pages are special text documents written in HTML, or hypertext markup language (27).

After successfully receiving the web page, the browser will parse the HTML web page and render the page within the browser display. The HTML page may contain non-HTML components, such as a flash movie. The web browser will locate the proper local programs to run the components.

Web pages also contain forms and allow uses to submit/upload information. The web browser will send a "PUT" request to the server to upload the user's input. The web server will then process the input and return the result, if necessary.

WWW has evolved considerably since its birth in the early 1980s. Today it has grown far beyond the simple static web page of text. It now serves as a platform for a broad range of complicated applications such as e-commerce, social networking, search engine, online video, and so forth. In recent years, an informal term, Web 2.0, is being used to represent a set of emerging Internet and WWW-based applications that aim to facilitate creativity, collaboration, and sharing between users. Examples of Web 2.0 applications include blogs, wikis, and online communities.

CONCLUDING REMARKS

We have briefly described wide-area networks from the perspective of each layer in the TCP/IP model. In the last four decades or so, computer networks have expanded from a small network with just several nodes connected by 2.4 Kb/s to today's Internet with billions of nodes and backbone speed now at a 100 Gb/s level.

WANs will continue to evolve rapidly, and new transport media, switches, network protocols, and, in particular, applications, will emerge.

BIBLIOGRAPHY

1. Ethernet/IEEE 802.3 group. Available: http://www.ieee802.org/3/.
2. A. E. Joel, Jr., *Asynchronous Transfer Mode*. IEEE Press, 1993.
3. WiFi/IEEE 802.11 group Available: http://www.ieee802.org/11/.
4. W. Simpson, (ed.), *The Point-to-Point Protocol (PPP)*. RFC 1661, July 1994.
5. J. Postel, (ed.), *Internet Protocol*. RFC 791, Sep. 1981.
6. J. Moy, *OSPF Version 2*. RFC 2338, Internet Engineering Task Force, April 1998.
7. D. Oran, (ed.), *OSI IS-IS Intra-domain Routing Protocol*. RFC 1142, February 1990.
8. Y. Rekhter, T. Li, and S. Hares, (eds.), *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271, January 2006.
9. D. C. Plummer, *An Ethernet Address Resolution Protocol*. RFC 826, November 1982.
10. J. Postel, (ed.), *Transmission Control Protocol*. RFC 793, Sep. 1981.
11. J. Postel (ed.) *User Datagram Protocol*, RFC 768, Aug. 1980.
12. J. H. Saltzer, D. P. Reed, and D. D. Clark, End-to-end arguments in system design, *ACM Transactions on Computer Systems 2*, **4**; 277–288, 1984.
13. R. Bush and D. Meyer, *Some Internet Architectural Guidelines and Philosophy*. RFC 3439, December 2002.
14. R. Perlman, An algorithm for distributed computation of a spanning tree in an extended LAN, ACM SIGCOMM Computer Communication Review **15** (4): 44–53 1985.
15. Y. Rekhter and T. Li, *An Architecture for IP Address Allocation with CIDR*. RFC 1518, September 1993.
16. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001, pp. 588-592.
17. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001, pp. 595–601.
18. S. Floyd, *HighSpeed TCP for Large Congestion Windows*. RFC 3649, Experimental, December 2003.
19. I. Rhee and L. Xu, CUBIC: A new TCP-friendly high-speed TCP variant, *PFLDnet*, Lyon, France, 2005.
20. K. Tan, J. Song, Q. Zhang, and M. Sridharan, A compound TCP approach for high-speed and long distance networks, *Proc.*

INFOCOM 2006 / 25th IEEE International Conference on Computer Communications, April 2006, pp. 1–12.

21. E. Kohler, M. Handley, and S. Floyd, Designing DCCP: Congestion control without reliability, *Proc. ACM SIGCOMM*, 2006.
22. L. Ong and J. Yoakum, *An Introduction to the Stream Control Transmission Protocol (SCTP)*. RFC 3286, May 2002.
23. Y. Gu and R. L. Grossman, UDT: UDP-based data transfer for high-speed wide area networks, *Computer Networks*, **51** (7), 2007.
24. J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, Globally distributed Content Delivery, *IEEE Internet Computing*, September/October 2002, pp. 50–58.
25. P. Mockapetris, *Domain Names - Implementation and Specification*. RFC1035. Nov. 1987.
26. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *Hypertext Transfer Protocol - HTTP/ 1.1*. RFC 2616, June 1999.
27. T. Berners-Lee and D. Connolly, *Hypertext Markup Language - 2.0*. RFC 1886, November 1995.

FURTHER READING

S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460, December 1998.

J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, 3rd ed. Addison Wesley, 2004.

W. Stallings, *Data and Computer Communications*, 8th ed. Prentice Hall, 2006.

W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*.

W. R. Stevens and G. R. Wright, *TCP/IP Illustrated, Vol. 2: The Implementation*.

A. S. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall PTR, 2002.

YUNHONG GU
University of Illinois at Chicago
Chicago, Illinois

WIMAX NETWORKS

IEEE Std 802.16-2004 or Worldwide Interoperability for Microwave Access (WiMAX), is a broadband wireless system that offers packet-switched services for fixed, nomadic, portable, and mobile accesses. WiMAX uses orthogonal frequency division multiplexing (OFDM) and many other advanced technologies in the physical (PHY) and the medium access control (MAC) layers to provide higher spectrum efficiency than a code division multiple access (CDMA) system. Moreover, WiMAX supports scalable channel bandwidths and can be operated over different frequency bands so that operators have the flexibility to deploy a WiMAX network over various radio spectrums. With these important features, WiMAX has become one of the most important technologies for broadband wireless access (BWA) in both fixed and mobile environments.

IEEE Std 802.16-2004 is initially designed as an access technology for a wireless metropolitan area network (WMAN). The first specification ratified by the IEEE in 2004, i.e., IEEE Std 802.16-2004, targets on fixed and nomadic accesses in both line-of-sight (LOS) and non-line-of-sight (NLOS) environments. In the IEEE 802.16e-2005 amendment, the IEEE 802.16e system (also called Mobile WiMAX) further provides handover, sleep-mode, idle-mode, and roaming functions to facilitate mobile accesses. The system also uses scalable orthogonal frequency division multiplexing access (SOFDMA), which is optimized for accessing dynamic mobile radio channels. Besides the PHY and MAC layer specifications, IEEE working groups and technical forums have also defined management and networking protocols for WiMAX. For example, IEEE Std 802.16g standardizes the management plane for both fixed and mobile devices and networks. IEEE Std 802.16f and IEEE Std 802.16i facilitate cross-vendor interoperability for IEEE 802.16 and IEEE 802.16e devices and networks, respectively. To address the requirements for network and service deployment, the WiMAX Forum was thus formed in 2001 to promote and certify WiMAX products. The WiMAX Forum also specifies management plane procedures, an end-to-end network architecture, application and service operations, and conformance test cases for both fixed and mobile WiMAX. With these efforts, WiMAX becomes a complete solution for broadband wireless access beyond 3G.

This article provides an overview to WiMAX from an end-to-end perspective. The next section describes the architecture and entities of a WiMAX network. Then the design of fixed/mobile WiMAX PHY and MAC layers is presented. Then in the subsequent sections, protocols and procedures for the network entry, connection management, mobility management, sleep-mode and idle-mode operations, and security management are introduced.

WIMAX NETWORK ARCHITECTURE

Based on IEEE Std 802.16 and IEEE Std 802.16e, the network group (NWG) under the WiMAX Forum develops network architecture, entities, and protocols for a WiMAX network and defines reference points between the entities. These network entities are logical components and may be integrated in a physical network node. A reference point is a conceptual point between network entities, which associates with a number of protocols. When logical entities colocate in a network node, reference points between the entities are implicit. Figure 1 illustrates the WiMAX network architecture consisting of three major parts: subscriber stations/mobile stations [SSs/MSs; Fig. 1(1)], network access providers [NAPs; Fig. 1(2)], and network service providers [NSPs; Fig. 1(3,4)]. An SS/MS is customer premise equipment (CPE) that is a mobile or a personal device for individual usage or a residential terminal that is shared by a group of users. Subscription, authentication, authorization, and accounting (AAA) of WiMAX services can be applied to either devices or both devices and subscribers. In this architecture, interfaces R1–R8 between network entities are specified. The R1 interface between SSs/MSs and BSs implements control and data planes conformed to IEEE Std 802.16-2004 and IEEE Std 802.16e-2005 specifications, and other management plane standards. R2 logical connection between an SS/MS and the home AAA server is established for authentication and authorization purposes.

An NAP establishes, operates, and maintains several access service networks [ASNs; Fig. 1(8)] deployed in different geographical locations. An ASN consists of base stations [BSs; Fig. 1(5)] controlled by one or more ASN-gateways [ASN-GWs; Fig. 1(6)]. An ASN-GW inter-works an ASN with a connectivity service network [CSN; Fig. 1(7)] operated by a network service provider (NSP). The ASN-GW transmits packets between SSs/MSs and CSNs, handles ASN-anchored mobility, implements a mobile IP foreign agent, and security functions such as authenticator and key distributor. The ASN-GW also manages radio resources of the BSs in an ASN.

The functional partition between BS and ASN-GW is an implementation issue not defined by either IEEE 802.16 or WiMAX Forum. Generally speaking, a BS implements most PHY and MAC functions. On the other hand, an ASN-GW implements data plane functions such as packet classification, and control plane functions such as handover decisions, radio resource control, an address allocation relay, and an AAA proxy. In decentralized ASN implementation, certain functions such as handover decisions and radio resource management are moved from ASN-GW to BS. This approach increases the scalability of an ASN. R4, R6, R7, and R8 reference points are defined in an ASN. R4 is the interface between ASN-GWs. This interface defines control plane for mobility management and data

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

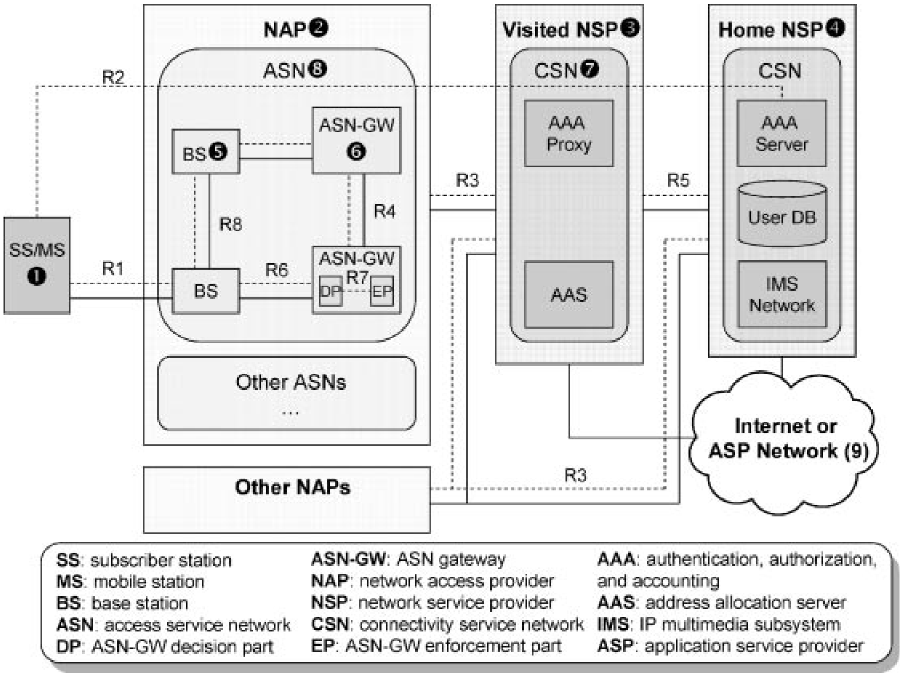

NAP❷
ASN❽
Visited NSP❸
CSN❼
Home NSP❹
CSN
R2
AAA Proxy
AAA Server
BS❺
ASN-GW ❻
R3
R5
R8
R4
User DB
SS/MS ❶
R1
R6
ASN-GW
R7
DP
EP
BS
AAS
IMS Network
Other ASNs
...
Internet or ASP Network (9)
Other NAPs
R3
SS: subscriber station
MS: mobile station
BS: base station
ASN: access service network
DP: ASN-GW decision part
ASN-GW: ASN gateway
NAP: network access provider
NSP: network service provider
CSN: connectivity service network
EP: ASN-GW enforcement part
AAA: authentication, authorization, and accounting
AAS: address allocation server
IMS: IP multimedia subsystem
ASP: application service provider

Figure 1. WiMAX network architecture.

packet forwarding between ASN-GWs during handover. R6 reference point defines control and data plane packet delivery between BSs and an ASN-GW. R8 is the interface for transferring control plane packets and optionally data packets between BSs. This interface facilitates fast and seamless handover.

An NSP operates a CSN, and the CSN manages subscriber information such as service policies, AAA records, and so on. To provide services to SSs/MSs, an NSP can either establish its own service networks such as IP multimedia core network subsystem (IMS) in a CSN or forwards SSs/MSs' requests to other application service providers [ASPs; Fig. 1(9)]. A user initially subscribes to the services through a contract agreement with an NSP. The NSP then establishes contact agreements with one or more NAPs that offer WiMAX access services. Also, the NSP may have roaming agreements with other NSPs so that a roaming SS/MS can attach to its home NSP [Fig. 1(4)] via visited NSPs [Fig. 1(3)]. In such a case, the SS/MS first associates with an NAP, which only has a contact agreement with a visited NSP. Then, the visited NSP relays authentication messages to the SS/MS's home NSP, and finally the home NSP authenticates and authorizes the SS/MS. To further access Internet or services provided by ASP networks, IP addresses should be assigned to SSs/MSs. An ASN-GW implements DHCP relay functions and forwards SSs/MSs' IP acquisition requests to either visited NSPs or home NSPs to obtain IP addresses. In a CSN, R3 (between an NAP and an NSP) and R5 (between NSPs) are defined. The R3 reference point implements control plane protocols such as AAA, policy enforcement, and mobility management. Data plane packets are tunneled and transferred between an ASN and a CSN over the R3 interface. The R5 reference point consists of a set of control and data plane protocols for interconnecting home NSP with visited NSP.

PHY AND MAC LAYERS

Figure 2 illustrates the control plane and data plane protocols for WiMAX. IEEE Std 802.16 and IEEE Std 802.16e specify control plane messages for network entry, connection management, mobility management, security management, and so on. These messages are carried by either basic, primary management or secondary management connection identifiers (CIDs) [(1) in Fig. 2(a)] and then they are transferred between SSs/MSs and BSs through the MAC layer [(3) in Fig. 2(a)] and the PHY layer [(4) in Fig. 2(a)]. In IEEE Std 802.16, a connection that is numbered by a unique CID in a cell is a unidirectional mapping between BS and MS MAC peers for transferring a service flow's traffic. The WiMAX Forum further defines control protocols [(2) in Fig. 2(a)] between BSs and ASN-GWs over UDP/IP in order to support the control plane procedures in an ASN network.

IEEE Std 802.16 and IEEE Std 802.16e also define the data plane protocols for data packet delivery between SSs/MSs and BSs. The convergence sublayer [CS; (6) in Fig. 2(b)] performs packet classification, header suppression, and converts packets between upper layer and the MAC layer. Currently, two CSs, i.e., the asynchronous transfer mode (ATM) CS and packet CS, are supported [(7) in Fig. 2(b)]. The MAC layer receives service data units (SDUs) from the CS, which may fragment and pack the SDUs, encrypts the packets, generates the MAC protocol data units (PDUs), and then sends the PDUs to the PHY layer [(3) in Fig. 2(a)]. The PHY layer performs the baseband processing on MAC PDUs and transmits the information over the air by using OFDM/OFDMA technologies [(4) in Fig. 2(a)]. A BS or an SS/MS receives the signals and then passes data to the MAC layer after the baseband processing. The receiver MAC needs to reassemble the PDUs,

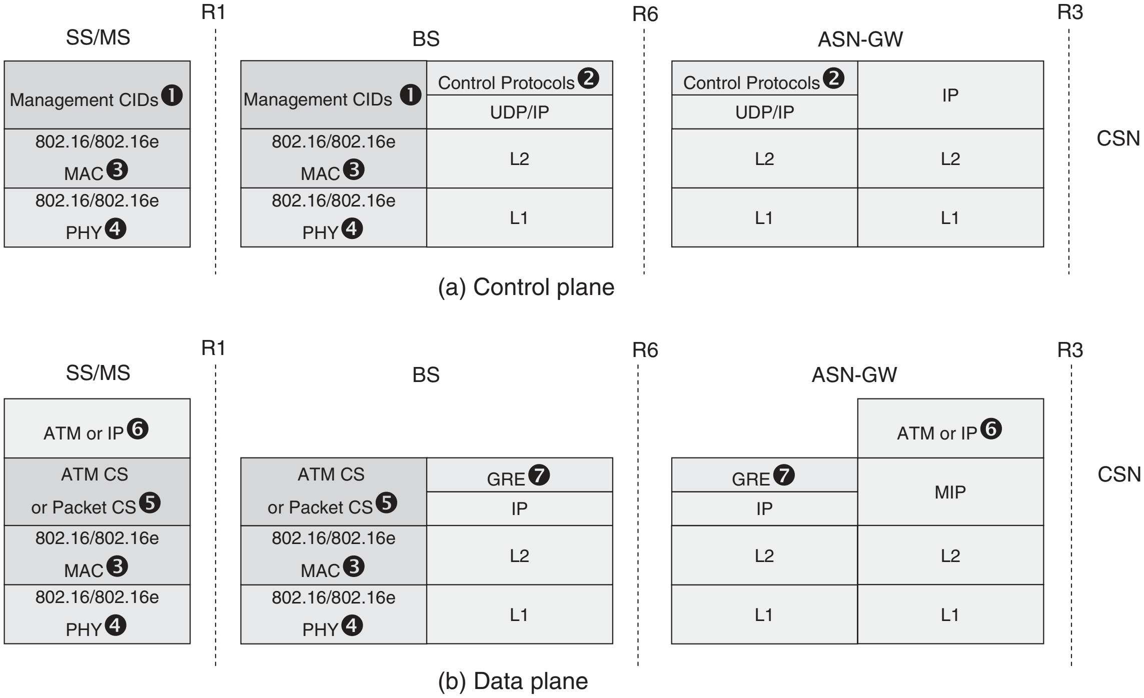


Figure 2. Overview of WiMAX protocol stack.

performs retransmission if necessary, decrypts the packets, and finally forwards the packets to upper layer protocols via the service-specific CSs. To deliver packets between BSs and ASN-GWs, the WiMAX Forum reuses Generic Routing Encapsulation [GRE; (8) in Fig. 2(b)], which is a tunnel protocol over an IP transport infrastructure defined by the Internet Engineering Task Force (IETF).

Figure 3 shows the details of data packet processes for IEEE Std 802.16 and IEEE Std 802.16e. A network-layer connection such as an IP connection has to be mapped to a service flow, which has its own service flow identifier (SFID) in a WiMAX network. The service flow is defined as a unidirectional flow of MAC SDUs and has its own quality-of-service (QoS) requirements. A service flow is a logical entity. During transmission, the service flow must associate with a link-layer connection, i.e., an IEEE 802.16 connection with a CID. One of CS major tasks performs the CID classification while it receives upper layer SDUs such as ATM cells or IP packets [Fig. 3(1)]. The classification for the ATM CS can be done by mapping ATM virtual circuit or virtual path to a specific CID. On the other hand, the packet CS may have to check the IP or TCP/UDP header of the SDU to determine the CID. Besides the CID mapping, the CS may perform the optional payload header suppression (PHS) to eliminate the redundant parts of the SDUs during the transmission over the air interface [Fig. 3(2)]. For example, if the header information of an IP packet is not used during transmission and routing in a WiMAX network, the IP header can be removed by the sender and reconstructed by the receiver to save radio resources. An SS/MS and a BS that activate the PHS function should first negotiate header suppression parameters. For example, the PHS parameters are composed of a classification rule for identifying the packets that should be processed by the header suppression, a payload header suppression mask (PHSM) that indicates the parts of a header should be removed, and a payload header suppression field (PHSF) that tells the receiver the original parts of headers for reconstruction. These PHS-related information are described in a data structure, indexed, and stored on the BS and the corresponding SS/MS. When a BS or an SS/MS sends a packet, the CS matches the PHS rules, finds the PHS index (PHSI), masks the packet using the PHSM, generates the new PDU with the PHSF, and sends the packet to the receiver. The receiver checks the PHSI in the PDU, searches the PHS information, and rebuilds the original packet using the PHSM and PHSF. The PHS is applied to a connection, and each connection may associate with more than one PHS rule and PHS setting.

SDUs are sent to the MAC layer after they are processed by the CS. The MAC layer may perform the block processing of the automatic repeat request (ARQ) on MAC SDUs if the ARQ is enabled for this connection [Fig. 3(3)]. The ARQ mechanisms used for retransmitting lost packets are optional in IEEE Std 802.16 but are mandatory for IEEE Std 802.16e. WiMAX and Mobile WiMAX support several ARQ mechanisms, and their parameters should be negotiated by a BS and an SS/MS. When the ARQ is enabled, SDUs are first segmented into fixed-size ARQ blocks, which are the basic retransmission units defined in the ARQ mechanism. When any ARQ block is lost, the sender needs to retransmit the ARQ block. As an ARQ block is the basic retransmission unit, the following MAC processes such as packet fragmentation and packing must

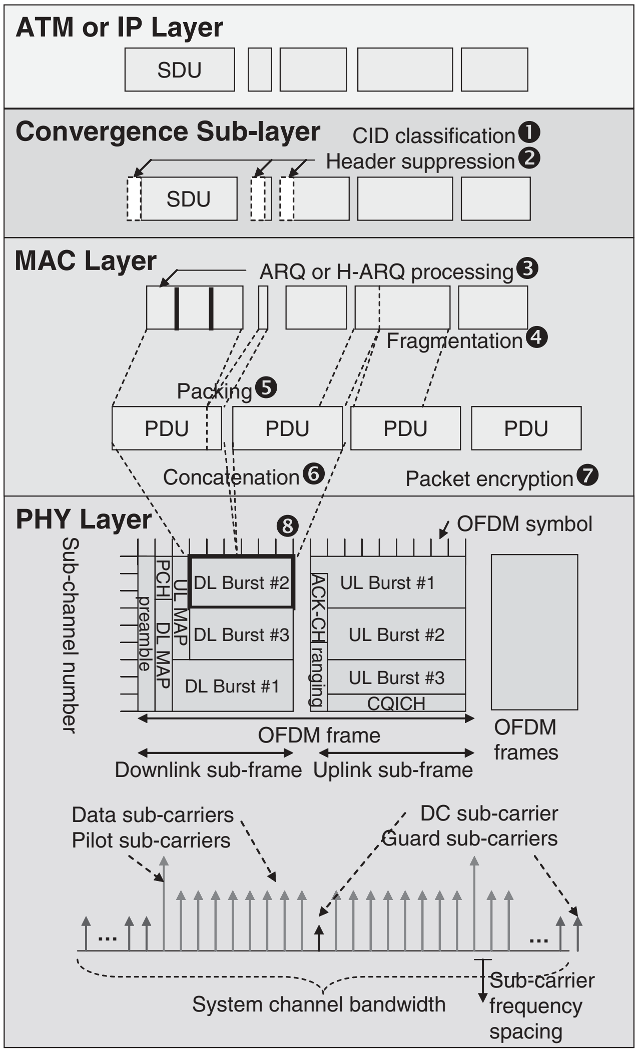


Figure 3. Overview of packet processing in IEEE Std 802.16 and IEEE Std 802.16e.

align with the boundary of an ARQ block. The MAC fragmentation divides an MAC SDU into one or more small PDUs [Fig. 3(4)], and the MAC packing packs multiple MAC SDUs into a single MAC PDU [Fig. 3(6)]. The MAC concatenates multiple MAC PDUs into a single transmission [Fig. 3(7)]. The MAC fragmentation, packing, and concatenation mechanisms are designed for efficient use of the available radio resources to meet the QoS requirements. The MAC layer also encrypts and decrypts MAC PDUs to prevent packet sniffing and modifications [Fig. 3(9)]. To perform packet encryption and decryption, a security association (SA) for a connection contains the security information and settings such as encryption keys. The SA information is negotiated by a BS and an SS/MS during the connection establishment phase. The MAC layer in the sender then encrypts MAC PDUs, and the receiver can decrypt these PDUs according to the information in the SA.

One of the most critical tasks for the MAC layer is the PDU scheduling and radio resource management. IEEE Std 802.16 and IEEE Std 802.16e reuse the data over cable system interface specifications (DOCSIS) MAC, which is a deterministic access method with a limited use of contention for bandwidth requests. All radio resources for downlink (DL) and uplink (UL) accesses are controlled by a BS. An SS/MS receives DL bursts that contain several PDUs to the SS/MS, and sends packets via the UL transmission opportunities, called UL bursts, which are also scheduled by a BS. In WiMAX, each service flow has its own QoS, and a BS uses these QoS information of these service flows to schedule DL/UL bursts. For example, a BS can schedule DL resources to SSs/MSs according to the QoSs associated with service flows. Also, a BS schedules UL resources based on the QoS of UL service flows and the bandwidth requests from SSs/MSs. All DL/UL schedules are decided by a BS, and the scheduling results are embedded in the DL-MAP and UL-MAP in every OFDM frame. SSs/MSs should listen to the DL-MAP and UL-MAP and receive and transmit packets according to the schedule.

For the IEEE 802.16 and IEEE 802.16e PHY layer, a system channel bandwidth must be first allocated. WiMAX supports both frequency division duplex (FDD), which requires two separated spectrums for DL and UL accesses, and time division duplex (TDD) where DL/UL accesses share the same spectrum. FDD may suffer from inefficient channel utilization due to unbalanced UL/DL traffics. TDD, on the other hand, can dynamically change the allocation of the UL and DL resources in each OFDM frame and is more flexible than FDD in terms of radio resource management. Figure 3 also shows a frame structure of a TDD-based OFDMA system. A system channel bandwidth is divided into several subcarriers. The frequencies of subcarriers are all orthogonal. These subcarriers can be categorized into pilot subcarriers that are used for pilot, a DC subcarrier that indicates the center subcarrier, guard subcarriers that serve as the guard band, and data subcarriers that are used to carry data packets. In OFDMA, subcarriers are further divided into groups, and one subcarrier from each group forms a subchannel. Subchannels are the basic unit to schedule DL/UL accesses. As shown in Figure 3, DL/UL bursts are scheduled and transmitted by several subchannels and for several OFDM symbols. A DL/UL burst that may contain several MAC PDUs for the same SS/MS are the basic schedule unit.

An OFDM frame is fixed with lengths such as 2 ms, 5 ms, and 10 ms, and each frame is composed of several OFDM symbols [Fig. 3(8)]. Two consecutive OFDM frames are guarded by a Receive Transition Gap (RTG). In an OFDM frame, a BS further divides a frame into a DL subframe and a UL subframe. An OFDM frame begins with a DL subframe, and a DL subframe has a preamble to identify the start of an OFDM frame. Followed by the preamble, a frame control header (FCH) contains the DL frame prefix and specifies the burst profile and the length of a DL-MAP. After FCH, the first DL burst is a broadcast burst containing many important information such as DL-MAP, UL-MAP, downlink channel describer (DCD), and uplink channel describer (UCD). The DL-MAP indicates the DL burst allocations, and the DCD describes the coding and modulation scheme that each burst uses. On the other hand, UL-MAP and UCD inform SSs/MSs how UL bursts are arranged and how UL bursts should be coded and modulated. OFDM/OFDMA support adaptive modulation and coding (AMC), and each burst can apply different modulation and coding schemes depending on the channel condition between a BS and an SS/MS. In UL subframes, there are several important bursts. The contention ranging period is a period that an SS/MS uses for the initial ranging. The channel quality information channel (CQICH) is a channel for SSs/MSs to report its channel conditions, and it can be used for the AMC. The details will be further elaborated in the next section.

NETWORK ENTRY

An SS/MS has to complete network entry procedures before it can access the Internet. Network entry for an SS/MS begins with a cell selection procedure [Fig. 4(1)]. An SS/MS first searches the cells that it associated before. If the last associated cells cannot be detected, the SS/MS performs a complete search of the spectrum. To locate the boundary of an OFDM frame, an SS/MS seeks for the preambles situated in the beginning of every OFDM/OFDMA frame. Once OFDM frames are synchronized, the SS/MS decodes FCH and the first DL burst containing the broadcast information from the BS [Fig. 4(2)]. The broadcast information composes of a DL-MAP, UL-MAP, DCD, and UCD, which indicate to all SSs/MSs how a DL subframe and UL subframe are organized. Based on the information, an SS/MS locates the contention period for the initial raging [Fig. 4(3)]. The initial ranging synchronizes the time and frequency between a BS and an SS/MS and adjusts the transmission power. The initial ranging is a contention-based ranging, which means that all SSs/MSs

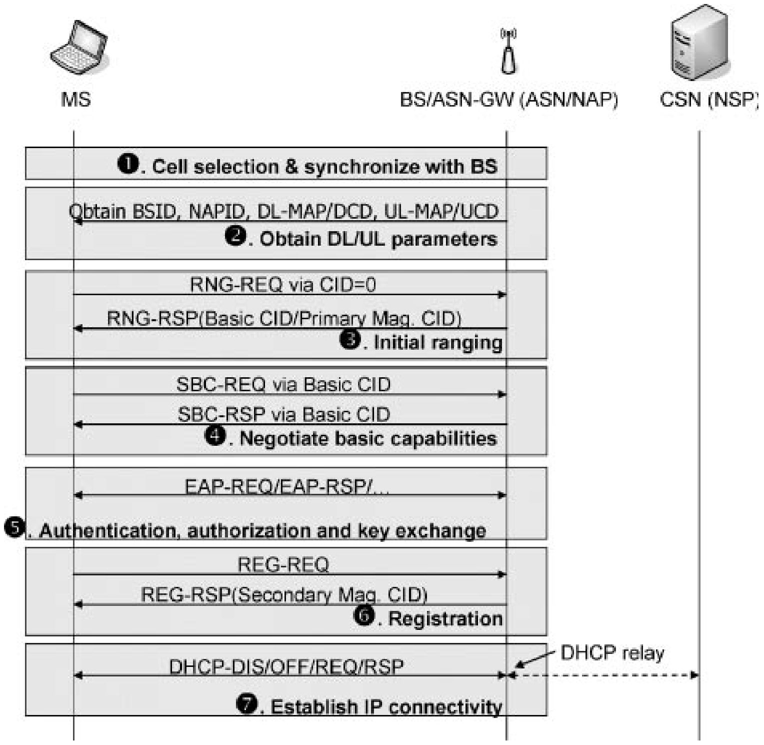

MS
BS/ASN-GW (ASN/NAP)
CSN (NSP)
❶. Cell selection & synchronize with BS
Obtain BSID, NAPID, DL-MAP/DCD, UL-MAP/UCD
❷. Obtain DL/UL parameters
RNG-REQ via CID=0
RNG-RSP(Basic CID/Primary Mag. CID)
❸. Initial ranging
SBC-REQ via Basic CID
SBC-RSP via Basic CID
❹. Negotiate basic capabilities
EAP-REQ/EAP-RSP/...
❺. Authentication, authorization and key exchange
REG-REQ
REG-RSP(Secondary Mag. CID)
❻. Registration
DHCP relay
DHCP-DIS/OFF/REQ/RSP
❼. Establish IP connectivity

Figure 4. An example of network entry.

send ranging requests in the same period. If an SS/MS does not receive ranging response from the BS, the SS/MS should increase the transmission power and retransmit the ranging requests in the subsequent contention-based ranging periods with random back-offs. After successfully receiving a ranging response, the initial ranging is complete. The CID of an initial ranging message (RNG-REQ) is zero. When a BS replies to the request, a ranging response message (RNG-RSP) informs the SS/MS of the basic CID and the primary management CID, which are used to carry important management messages between the BS and the SS/MS.

After ranging procedures, an SS/MS negotiates basic capacities of the PHY/MAC layer such as ARQ supports with a BS through SBC-REQ/SBC-RSP messages [Fig. 4(4)]. Following the basic capacity exchanges, authentication and authorization procedures are performed [Fig. 4(6)]. For an SS shared by several users, devices and subscribers might be authenticated and authorized separately. Security management related functions will be discussed below. Once an SS/MS has been authenticated and authorized, an SS/MS sends a registration request message (REG-REQ) to register to a WiMAX network [Fig. 4(7)]. In the registration response message (REG-RSP), the BS provides the SS/MS a new CID called the secondary management CID. The secondary management CID carries management messages forwarded to the network nodes behind a BS/ASN-GW. To access Internet, the SS/MS further acquires an IP address [Fig. 4(8)] either allocated by the visiting NSP or issued by the home NSP.

CONNECTION MANAGEMENT AND QOS

After an SS/MS has successfully attached to a WiMAX network, the home NSP downloads the user's QoS profile and the associated policy rules to the service flow management (SFM) and service flow authorization (SFA) [Fig. 5(1)]. Both SFA/SFM are logical entities implemented in an ASN/NAP. SFM is responsible for the admission control and management such as creation and deletion of service flows. SFA is responsible for evaluating service requests against the user's QoS profile. The establishment of a new service flow is either initiated by the network or by an SS/MS. Figure 5 shows an example where an SS/MS sends a service flow creation message (DSA-REQ) to a BS to initiate a service flow [Fig. 5(2)]. A service flow creation message from an SS/MS contains a service flow identifier (SFID) and may specify the PHS and other MAC parameters. When a BS receives the message, it first checks the integrity of the message and sends an acknowledgment message (DSA-RVD) to the SS/MS. Then the BS determines whether the service flow is accepted according to the QoS profile and available resources of a BS. If so, the BS replies to the SS/MS with a response message (DSA-RSP), and then the service flow is established [Fig. 5(3)]. When a BS or an SS/MS starts to transmit packets, the service flow needs to be activated and associated with a link-layer connection with a unique CID.

A connection for a service flow is associated with a schedule data service that is an unsolicited grant service (UGS), enhanced real-time polling service (ertPS), real-time polling service (rtPS), non-real-time polling service (nrtPS), or best effort service (BE). These data service scheduling are defined by IEEE Std 802.16. IEEE 802.16e further defines ertPS. Characteristics of data connections for these scheduling services are described below.

- **UGS**: For a UGS connection, a BS guarantees a fixed amount of DL or UL transmission bandwidths. UGS is

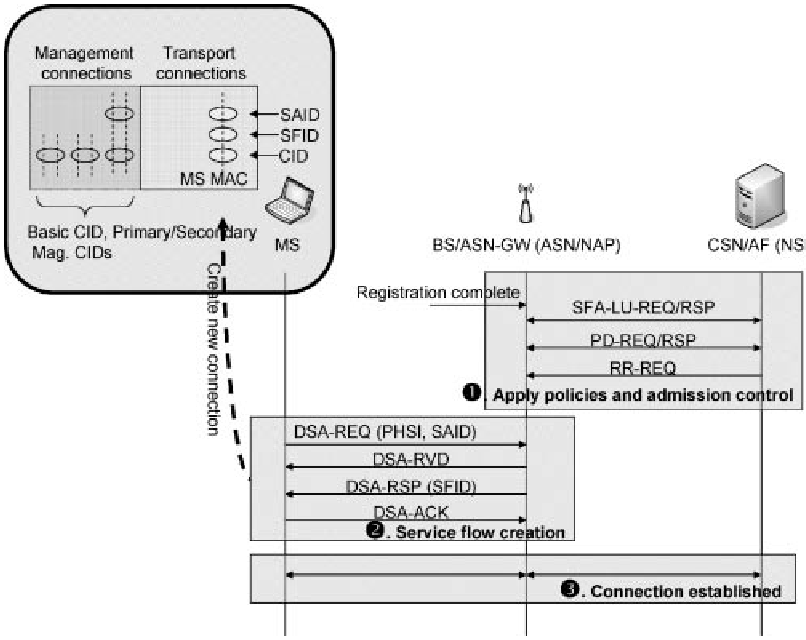


Figure 5. An example of service flow establishment.

suitable for the constant bit rate (CBR) traffic such as voice over IP (VoIP) without silence suppression.

- **ertPS**: Different from a UGS service, ertPS supports VoIP with silence suppression or variable bit rate (VBR) real-time services. In ertPS, a BS not only allocates fixed amount of UL or DL resources to an MS, but also allocates the bandwidth requests in the UL bursts to an MS so that the MS can use the bandwidth requests to change UL allocations. This mechanism allows a BS to save the radio resources if it does not have packets to transmit during silence periods.
- **rtPS**: To support real-time service flows such as video streaming, a BS allocates periodical bandwidth requests in UL bursts to an SS/MS and polls the SS/MS if there is any UL burst need. If an SS/MS has packets to transmit, it can simply use the reserved bandwidth request slots to request UL bursts. Since the requests of UL bursts are done by a periodical polling basis, the response time for a UL packet is fast and the rtPS can support real-time applications.
- **nrtPS**: For non-real-time traffic, such as Web access and Telnet, an nrtPS connection is allocated regular bandwidth request resources to an SS/MS, and an SS/MS that has packets to transmit should use the bandwidth requests to request UL bursts. Since the bandwidth request is not sent periodically, the bandwidth request might not be received by the BS immediately and the delay for UL burst allocations cannot be guaranteed.
- **BE**: A BS allocates resources to BE connections in a best effort manner. Therefore, this type of connection cannot guarantee any QoS.

A BS has to schedule DL and UL resources and guarantees the QoSs of the service flows. It also has to refer to the channel qualities between a BS and each SS/MS to schedule DL/UL bursts, which associate with different modulation and coding schemes in order to maximize the radio utilization.

MOBILITY MANAGEMENT

WiMAX mobility functions can be categorized into ASN-anchored and CSN-anchored mobility management. ASN-anchored handover, also called micro mobility, implies that an MS moves from one BS to another BS without updating its care-of address (CoA). CSN-anchored handover, on the other hand, defines macro mobility where an MS changes its serving ASN-GW/FA and its CoA. In general, the handover procedure includes the following steps. First, an MS performs a cell (re)-selection, which comprises scanning and association procedures to locate candidate BSs to handover. Second, the MS is informed or decides to handover to the target BS. Finally, the MS completes network (re)-entry procedures and performs network-layer handover procedures if necessary.

The scan measures the signal qualities of the neighboring BSs for an MS, and the measurement reports are used for either MSs or BSs to select the target BS during handovers [Fig. 6(1)]. Initially, the serving BS may indicate MSs for the scanning trigger-conditions in DCD and/or neighbor advertisement messages (MOB_NBR-ADV). The MOB_NBR-ADV broadcasting message contains a list of suggested BSs for scanning and the DCD, UCD, and other parameters of the BSs. Therefore, an MS can synchronize with the neighbor BSs. After receiving DCD or MOB_NBR-ADV messages, an MS should measure the signal qualities of the serving BS and other BSs and check whether the measurement results satisfy the trigger criteria. If the scan procedure is triggered, an MS sends a MOB_SCN-REQ message to the serving BS with the MS's preferred scanning and interleaving intervals. Also, the MOB_SCAN-REQ message contains a list of BSs that are selected from the neighbor BSs in the MOB_NBR-ADV message or other BSs, which are not in the neighbor BS list. The serving BS then replies a scan response message (MOB_SCN-RSP), which contains the final list of BSs to scan, the start frame of the scan, the length of a scanning and interleaving interval, and the scan iteration. The start frame of the scan indicates the exact frame for the MS to perform scan, and the scan and interleaving interval are used to determine the length of a scan and normal operation period. The scanning and interleaving intervals are scheduled in a round-robin basis, and the scan iteration controls the number of iterating scanning intervals.

An MS may perform associations with neighbor BSs during scanning intervals. Association helps an MS to establish basic relationships such as ranging for these BSs, which may become potential target BSs for the MS. By conducting associations before handovers, MSs can reduce the time to synchronize and register with the target BS. The scanning type in a MOB_SCN-RSP message indicates whether an MS should perform an association with a neighbor BS, and what association type an MS and a BS should establish. Several scanning types are defined.

- **Without Association**: The MS does not have to perform associations during scanning intervals.
- **Association Level 0** (scan/association without coordination): The MS should perform an association during scanning intervals, but the neighbor BSs do not allocate dedicated ranging regions for the MS. Therefore, the MS must perform ranging procedures such as an initial ranging on a contention basis.
- **Association Level 1** (association with coordination): The serving BS coordinates ranging parameters of the neighbor BSs for the MS. The serving BS sends an association request over the backbone to notify the neighbor BSs, and the neighbor BSs allocate ranging opportunities for the MS and inform the serving BS. Then the serving BS sends the MS the association parameters such as the reserved ranging slots via a MOB_SCN-RSP message. The association parameters assist the MS to send ranging requests to the neighbor BSs in the reserved ranging slots. That reserved-based ranging is faster than the contention-based ranging.
- **Association Level 2** (network assisted association reporting): The MS is not required to wait for ranging

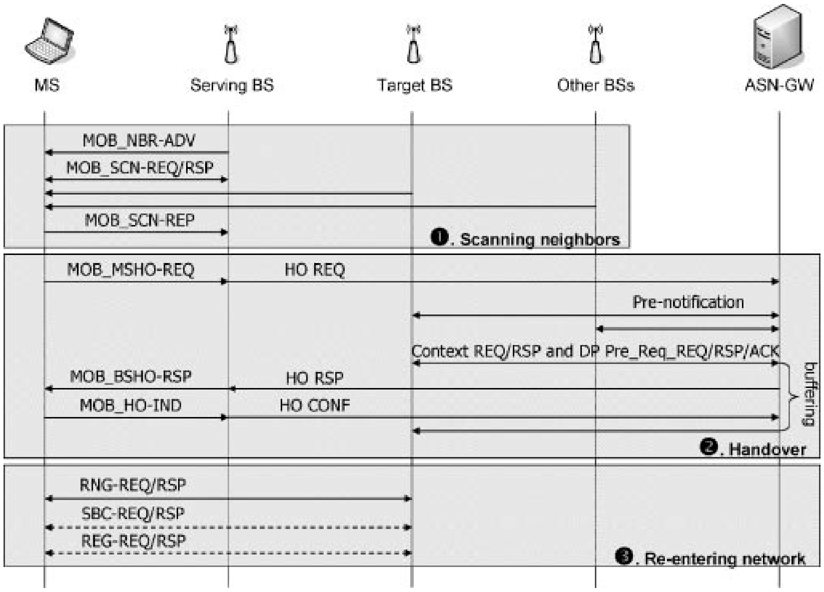


Figure 6. An example of an ASN-anchored handover.

response messages replied by the neighbor BSs after sending ranging requests. The ranging response messages are forwarded to the serving BS over the backbone network and are sent by the serving BS to the MS.

A handover followed by scanning and association procedures can be initiated by an MS or the network. Figure 6 gives an example of an ASN-anchored handover initiated by an MS. After the cell selection [Fig. 6(1)], an MS sends a handover request message (MOB_MSHO_REQ) to the serving BS [Fig. 6(2)]. The handover request message contains a list of candidate BSs and a measurement report of the BSs. Based on this report and some other information on the serving BS, the serving BS sends a handover request message (HO request) to one or several neighbor BSs over the backbone network to identify the possible target BSs. Once the neighbor BSs receive handover requests from the serving BS, the BSs may send a context request to the context server to collect information such as the QoSs of current connections of the MS and check whether they have sufficient resources to support this handover. After the context transfer and data path pre-registration, the neighbor BSs send handover response messages (HO response) to the serving BS. The serving BS summarizes the results from the neighbor BSs and finally decides a new list of recommended BSs and replies a MOB_BSHO-RSP message to the MS. Meanwhile, buffering schemes for queueing incoming packets to the MS should be performed on an ASN-GW and/or BSs to the MS to prevent packet loss.

After receiving a handover response message (MOB_BSHO-RSP), an MS should send a handover indication message (MOB_HO-IND) to confirm or terminate the handover process. In the MOB_HO-IND message, an MS explicitly notifies the target BS of the MS. Finally, an MS disconnects from the serving BS and synchronizes with the target BS. An MS can either perform ranging procedures or directly accesses the target BS if the association has been already established during the scanning phase. After the ranging procedure, an MS needs to perform network (re)-entry procedures [Fig. 6(3)]. To accelerate network (re)-entry, the target BS can obtain the configurations and settings such as service flows, state machines, and service information of an MS from the serving BS via the context server without the MS's involvement.

During handover, an MS may have to disconnect from the serving BS and then attaches to the network again via the target BS. Packets may be lost, and services may be disrupted during handover. To reduce the handover delay and minimize packet loss during handover, two advanced handover mechanisms, i.e., fast BS switching (FBSS) and macro diversity handover (MDHO), are proposed in the IEEE 802.16e-2005 specification. In FBSS and MDHO, an MS maintains a diversity set and an anchor BS. The diversity set is a list of target candidate BSs to handover for an MS. An anchor BS is the serving BS that transmits/receives packets to/from the MS over the air interface for FBSS. For MDHO, an MS receives the same data packets from all BSs in the diversity set, and only monitors the control information from the anchored BS, which may be any BS in the set. An MS must associate with the BSs in the diversity set before handover and should perform a diversity set update to include new neighbor BSs or remove BSs with poor signal qualities from the list. The ASN-GW should multicast incoming packets for an MS to all BSs in the diversity set, and therefore, the BSs in the diversity set are always ready to serve the MS for FBSS and MDHO. For the packet transmission over the air interface, an MS transmits/receives packets to/from the anchored BS only for FBSS. Since packets are ready in the BSs in the diversity set, the packet transmission can be resumed quickly after an MS performs an anchor BS update to change the serving BS. The packet loss and handover delay are reduced by employing the FBSS. On the other hand, in MDHO the BSs in the diversity set transmit the same data packets to the MS simultaneously. In this case, an MS can still receive packets from several BSs during handovers, and the MDHO approach further minimizes the packet loss and handover delay.

CSN-anchored mobility management involves MSs moving from the current FA to another FA. This type of handover requires MSs to change its CoA. Mobile WiMAX supports network-layer handover for both IPv4 and IPv6 networks. For IPv4, client mobile IP (CMIP) and proxy mobile IP (PMIP) are supported. For IPv6, only client mobile IPv6 (CMIPv6) is defined because each MS has its own IP address in an IPv6 network. CMIP integrates the conventional mobile IP (MIP) mechanisms with the designs for an MS and a Mobile WiMAX network to handle network-layer handover. On the other hand, to minimize the development efforts on MSs and to reduce MIP message exchanges over the air interface, PMIP suggests running a PMIP client on the ASN-GW or a dedicated node in the ASN. The PMIP client serves an agent to handle network-layer handover for MSs, and thus, network-layer handover is transparent to the MS.

SLEEP AND IDLE MODE MANAGEMENT

Power consumption might not be a problem for Fixed WiMAX, but it is a critical issue for Mobile WiMAX, which targets on portable devices. IEEE Std 802.16e, therefore, defines sleep-mode operations for MSs that have data connections but does not have a packet to send or receive. Three power-saving classes for sleep-mode operations are defined to accommodate network connections with different characteristics. Each connection on an MS can be associated with a power-saving class, and connections with a common demand property can be grouped into one power-saving class. If an MS establishes multiple connections with different demand properties, the periods that an MS can sleep are determined by the sleep-mode behaviors associated with all connections. The parameters of a power-saving class, i.e., the time to sleep and listen, the length of a sleep period and a listen period are negotiated by a BS and an MS. Then, an MS can sleep during the sleep periods, and can wake up to listen to the incoming packets during listen periods. Once an MS receives DL-MAP, which indicates packets to receive, the MS must return to the normal mode to receive the packets. Three power-saving classes are defined as follows:

- The type-one power-saving class specifies that an MS sleeps for a period and wakes up to listen for incoming packets. If there is no packet to send or receive during a listen period, the MS doubles the period for the next sleep. This power-saving class is suitable for Web browsing or data access services.
- The type-two power-saving class requires an MS to repeat the sleep and listen with fixed periods. This sleep mode is appropriate for real-time connections such as VoIP and video streaming services with periodic packet delivery. In this class, an MS only needs to wake up for packet delivery in those listen periods without violating the QoSs of the real-time connections.
- The type-three power-saving class defines the length of a sleep period, and an MS sleeps for that period and then returns to the normal operation.

On the other hand, if an MS does not have any connection for a period, an MS might want to switch to a deeper sleep state, called the idle mode, to conserve the energy. Mobile WiMAX defines its own idle-mode operations and paging network architecture. Four logical entities, i.e., paging controller (PC), paging group (PG), paging agent (PA), and location register (LR), for idle-mode and paging operations are defined. A PG that comprises one or several PAs in the same NAP is controlled by a PC, and a PC can manage one or more PGs. A PC can access an LR that contains information such as paging parameters for idle-mode MSs, and administers the activities of all idle-mode MSs situated in the PG managed by the PC. A PC can function as an anchor PC that is in charge of the paging and idle-mode management, and/or a relay PC that only forwards paging-related messages between PAs and an anchor PC. A PC could either colocate with a BS or a PC can be implemented on a network node such as an ASN-GW to communicate with its PAs through the R6 interface. PAs that are implemented on BSs interact with the PC to perform paging functions.

Figure 7 illustrates an example for an MS to enter the idle mode, update its location, and to be paged by the network. This example assumes that an LR and PC colocate on an ASN-GW and PAs are implemented on BSs. When an MS decides to switch to the idle mode [Fig. 7(1)], it first sends a de-registration message (DREG-REQ) to the ASN-GW [Fig. 7(2)]. The serving BS/PA and ASN-GW/PC release the resources such as the data path occupied by the MS and update the information of the MS to the LR. Meanwhile, the PA and PC negotiate, configure, and inform the paging parameters such as paging cycle, paging offset, paging interval length, anchor PC identifier, and paging group identifier for the MS. Based on the paging cycle (*PAGING_CYCLE*), paging offset (*PAGING_OFFSET*), and paging interval length, the MS derives the BS paging listening interval. A BS paging listening interval begins from the *PAGING_OFFSET* frame in every paging cycle, and each paging listening interval lasts for paging interval length. The MS has to stay awake during the entire BS paging listening interval in order to receive BS broadcasting paging messages (MOV_PAG-ADV).

The MS should perform a location update (LU) upon LU evaluation conditions [Fig. 7(3)]. For example, the MS performs an LU while the MS detects change of the paging group or when idle-mode timer expires. After a BS receives LU messages, the BS/PA updates the MS information to the PC/LR. When receiving an incoming packet sent to an idle MS, the ASN-GW/FA first obtains the information of the MS from the LR and informs the PC to page the MS. Then, the PC generates a paging announcement message and sends it to the relay PCs or PAs [Fig. 7(4)]. Based on the paging parameters of the MS, PAs/BSs send BS broadcasting paging messages (MOV_PAG-ADV) to the MS.

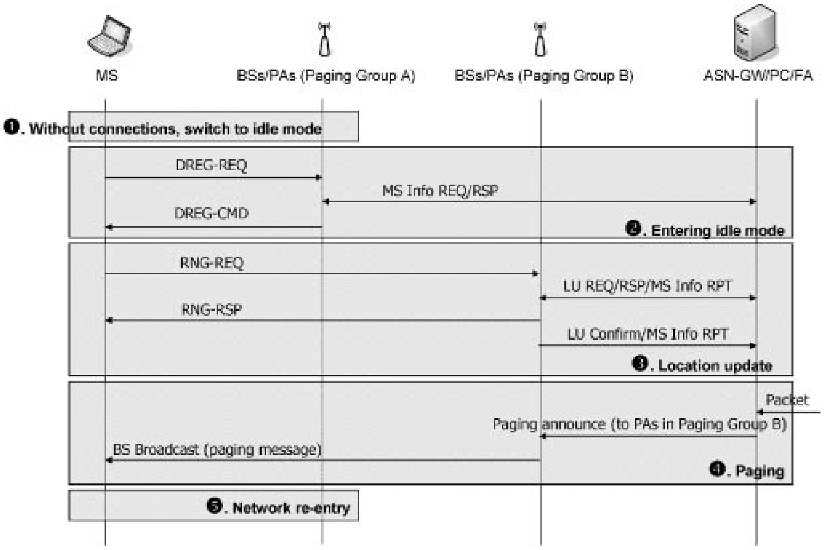


Figure 7. An example of idle-mode operation.

After an MS is paged, the MS shall exit idle mode, perform ranging with the serving BS, and complete the network (re)-entry procedures [Fig. 7(6)].

SECURITY MANAGEMENT

Security management in WiMAX includes authentication, authorization, key management, and encryption functions. When an SS/MS attaches to a WiMAX network, it is requested to perform the authentication and authorization based on X.509 protocol before it can register to a network. In IEEE Std 802.16e, authentication and authorization are enhanced by adopting IEEE Std 802.1X. In IEEE Std 802.16, the authentication and authorization can be applied to both a device and subscribers if an SS serves as a gateway, and it is shared by several users. Figure 8 shows an example for authentication and authorization using IEEE Std 802.1X. The authenticator first sends an identifier request to an SS/MS based on the EAP protocol after the SS/MS finishes the services capacity exchange with a BS during a network entry [Fig. 8(1)]. Depending on the authentication method negotiated by the authenticator and the subscriber, i.e., SS/MS, the message exchange between the authenticator and subscriber may be different [Fig. 8(2)]. After the authentication and authorization procedures, the SS/MS can register to a WiMAX network.

IEEE Std 802.16 uses privacy key management protocol version 1 (PKMv1) to support packet encryption and decryption. IEEE Std 802.16e further enhances the features by supporting PKMv2. In PKMv2, the master session key (MSK) is first established between the AAA server in the home NSP and the SS/MS. The MSK is transferred to the authenticator in the ASN, e.g., ASN-GW, which generates a pairwise master key (PMK) based on the MSK and other information. After the PMK is established, an SS/MS and authenticator further establish the authentication key (AK). The AK is then transferred from an ASN-GW to the

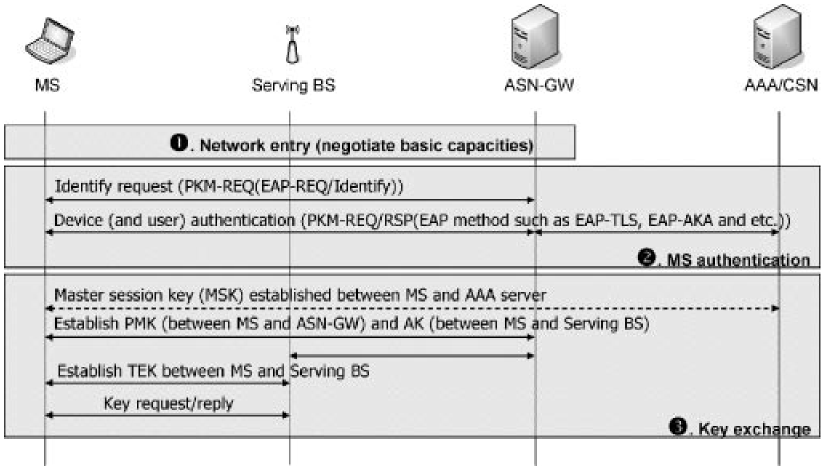

MS
Serving BS
ASN-GW
AAA/CSN
❶. Network entry (negotiate basic capacities)
Identify request (PKM-REQ(EAP-REQ/Identify))
Device (and user) authentication (PKM-REQ/RSP(EAP method such as EAP-TLS, EAP-AKA and etc.))
❷. MS authentication
Master session key (MSK) established between MS and AAA server
Establish PMK (between MS and ASN-GW) and AK (between MS and Serving BS)
Establish TEK between MS and Serving BS
Key request/reply
❸. Key exchange

Figure 8. An example of the authentication and key exchange.

serving BS. Finally the serving BS and SS/MS derive the traffic encryption key (TEK) based on IEEE 802.16 and IEEE 802.16e specifications [Fig. 8(2)]. With TEK, the data packets are encrypted and decryption based on specific algorithms such as the Advanced Encryption Standard (AES). Data encryption and decryption are applied to all data connections and the secondary management connection. Each connection must associate with a security association (SA), which is identified by an SA identifier (SAID). An SA is a data structure shared by a BS and an SS/MS. It is constructed during the connection establishment phase and describes the security information such as keys or other parameters for the connection.

SUMMARY

WiMAX and Mobile WiMAX have become a complete network solution for a broadband wireless and mobile communication system. With a total packet-switched design, the existing all-IP service network, e.g., IP multimedia subsystem (IMS), can be easily integrated with a WiMAX network to offer mobile data services. Although the basic functions and protocols of WiMAX have been established, several challenging issues need to be further investigated. Adaptive Antenna Systems (AAS) and Multiple-Input Multiple-Output (MIMO) are considered as important technologies for an OFDM-based system. They significantly influence MAC and radio resource management (RRM). Cross-layer approaches that consider not only physical behaviors, MAC designs, and upper layer transport and application protocols are important and should be further studied. MAC/RRM scheduling algorithms should be developed to improve the throughput and radio utilization, guarantee QoS, and minimize the power consumption for mobile devices. Mobility management mechanisms such as FBSS and MDHO for Mobile WiMAX and the integration of IEEE Std 802.16e and IEEE Std 802.21 can optimize and support seamless handovers within a Mobile WiMAX network and between WiMAX and other wireless access technologies. Technologies such as Mobile Multi-hop Relay (MMR), i.e., IEEE Std 802.16j, and Advanced IEEE 802.16, i.e., IEEE Std 802.16m, also bring new challenges for MAC, RRM, mobility management, and network architecture designs.

BIBLIOGRAPHY

1. IEEE Standard 802.16-2004, Air Interface for Broadband Wireless Access Systems, 2004.
2. IEEE Standard 802.16e-2005, Air Interface for Fixed and Mobile Broadband Wireless Access Systems; Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands, 2005.
3. H. Yaghoobi, Scalable OFDMA physical layer in IEEE 802.16 WirelessMAN, *Intel Tech. J.*, **8** (3): 201–212, 2004.
4. Understanding WiMAX and 3G for Portable/Mobile Broadband Wireless, *Intel Tech. White Paper*, 2004.
5. WiMAX End-to-End Network Systems Architecture (Stage 3: Detailed Protocols and Procedures), *WiMAX Forum Draft Document*, Aug. 2006.
6. Fixed, nomadic, portable and mobile applications for 802.16-2004 and 802.16e WiMAX networks, *WiMAX Forum Technical Document*, Nov. 2005.
7. Mobile WiMAX – Part I: A Technical Overview and Performance Evaluation, *WiMAX Forum Technical Document*, March 2006.
8. WiMAX End-to-End Network Systems Architecture (Stage 2: Architecture Tenets, Reference Model and Reference Points, *WiMAX Forum Draft Document*, Aug. 2006.
9. G. Hair, J. Chou, T. Madejski, K. Perycz, D. Putzolu, and J. Sydir, IEEE 802.16 medium access control and service provisioning, *Intel Tech. J.*, **8**(3): 213–228, 2004.
10. A. Ghosh, D. R. Wolter, J. G. Andrews, and R. Chen, Broadband Wireless Access with WiMAX/802.16: Current performance benchmarks and future potential, *IEEE Comm. Mag.*, 2005.
11. WiMAX Forum, http://www.wimaxforum.org/home/.
12. IEEE 802.16 Work Groups, http://www.ieee802.org/16/.

SHIAO-LI TSAO
YI-BING LIN
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

ABSTRACT DATA TYPES

INTRODUCTION TO DATA ABSTRACTION

The tasks involved in the design, development, and maintenance of large software systems are extremely complex because the details that need to be mastered far exceed most programmers' comprehensive ability. Abstraction is a fundamental technique for dealing with comprehension. In addition, abstraction supports reuse of code and flexibility in choosing different implementations.

Abstraction means that one focuses on the essential properties of the software system while ignoring most of the details of the implementation. Well-designed procedures and data types that hide the unnecessary implementation details are the key to abstraction and are discussed in Refs. 1 and 2.

Data abstraction focuses on the types of data and the operations on that data and ignores the complex details of the code. Abstract data types (ADTs) have been devised to support data abstraction.

An initial step toward the notion of using abstract data types (ADTs) was introduced through classes in the programming language Simula (3) in the 1960s. Parnas first discussed the concept of information hiding through modularization (4,5), which formed a basis for ADTs. Subsequent research introduced the notion of programming with ADTs (6). Since then, procedural languages such as ADA (7), CLU (8) and object-oriented languages such as Smalltalk (9), C++ (10), and Java (11) have allowed programmers to implement ADTs.

Early work formalizing ADTs also includes the introduction of procedural abstraction (12) and the use of algebraic methods in semantics (13,14,15). This research evolved into topics such as algebraic specification (16, 17, 18) and type theories (19). Hoare (20) provided an early theoretical treatment of ADTs within procedural languages. Many systems for the formal description of programs such as VDM (21) and Z (22) naturally support ADTs because they allow the description of intentions rather than detailed implementation. One use of VDM in this context can be found in Refs. 23 and 24.

The remainder of this article introduces ADTs for the data types stack, queue, set, and bag by describing their semantics or meaning using a form of mathematical description called algebraic semantics. The article also contains a detailed description of a stack implementation in the procedural programming language C (25). In this way, the reader can observe the details of how a practical ADT implementation might be structured.

INTRODUCTION TO THE CONCEPT OF ABSTRACT DATA TYPES

The term "abstract data type" can be understood based on four related phases; "type," "data type," "data structure," and "abstract." These terms are amplified in the next few paragraphs.

A type is a classification term related to collections of entities with common properties. For example, a type could represent all the people who live in a specific city or all the engineers in a country or state.

A data type is a type in which the classification relates to data values, such as integers, characters, or strings, or structures of data values, such as (string, integer) pairs, which could represent persons and their corresponding ages. A specific data type, such as all the real numbers or all the integers, also has related operations, such as addition and multiplication. A data structure is a specific implementation of a data type. The term abstract is defined later in this section after a brief discussion of some specific data types.

A data type in programming languages such as Pascal or Java is usually defined by:

(i) a collection of data values with similar characteristics (data type),
(ii) the operations on the data type, and
(iii) the implementation of the type and its corresponding operations (data structure).

Many different data types exist. Simple ones in programming languages are integers, reals (real numbers), characters, or strings of characters. Operations on integers or reals can include addition, subtraction, multiplication, or equals, whereas operations on strings could be concatenation, equals, or substring. More complex types can include arrays, tuples (sometimes called records), stacks, and queues.

Arrays are ordered finite sequences of the same data type, so we can have an array of reals. Elements of an array are usually accessed by their position in the sequence, and operations on an array include those to extract an element from a given position or place an element in a given position.

Tuples are ordered finite sequences of different data types and can also be extracted or placed by their position or by the name of the position. A tuple could consist of a name, gender, and age, where the name is a string of alphabetic characters, the age is a non-negative integer, and the gender is a single character, namely, "F" or "M."

Stacks consist of elements of the same data type and operate on the last-in-first-out principle similar to the stack of plates at a buffet table in a restaurant. Stacks have three basic operations, namely, *push*, which puts an item on top of the stack; *popOff*, which removes an item from the top of the stack but does not copy its value; and *top*, which copies the value of an element from the top of the stack but does not remove it. There is also a function *isEmpty* that determines if the stack is empty, because top and popOff can not be applied to an empty stack. From a programming perspective, it is also convenient to have an operation *pop*, which combines the effect of top and popOff.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Queues also consist of elements of the same data type and operate on a first-in-first-out principle in which elements are added at one end of the queue and removed from the other end. A queue is similar to a line waiting at a bus stop. Typical operations on a queue are *add*, which puts an element on a queue; *front*, which provides access to the value of the earliest added item placed on a queue; and *remove*, which removes the earliest added item placed on a queue. A queue also has a function *isEmpty* because front and remove cannot operate on an empty queue. Queues can also have an operation *delete*, which combines the operations front and remove.

Each data type can be implemented in many different ways. For example, integers can use a representation that allows for plus (+) and minus (−) signs or uses a modular or wrap-around approach. Complex data types such as stacks and queues can be implemented as either a fixed-size sequence (often called an array) of memory cells that are contiguous or separate memory cells that are linked together where each cell contains pointers (or the address) to its immediate successor or predecessor. This second implementation is called a linked list and allows a stack or queue or similar data type to grow to the size of available memory rather than be constrained to a fixed size as in an array.

An abstract data type (ADT) is a data type and its related operations, independent of its underlying implementation. Therefore, a change in this implementation should not affect the program that is using abstract data types because it only uses the operations and the data types involved with those operations, thus, following the hiding principle (4). Usually the operations on an ADT are defined by an interface, which cannot change when the ADT implementation changes.

A definition of abstract data type provided by the National Institute of Standards and Technology (NIST) (26) states that an ADT is a set of data values and associated operations that are precisely specified independent of any particular implementation.

To use a data type as an ADT, two specific details must be known:

(i) The meaning of the operations or functions acting on the ADT or operation semantics, that is, the properties of the data objects that the operations require and produce, and

(ii) The form of the operations on an ADT or operations syntax, that is, the program interface to the ADT.

The next two sub-sections contain a general description of how the syntax and semantics can be represented.

Operation Meaning or Semantics

The semantics of the operations or functions acting on an ADT can be described in natural language, mathematics, or programming languages. By using mathematics, it becomes possible to be precise about the semantics of each operation and thus be able to reason about results, such as the effects of an operation, the relationships to other operations and properties of data objects related to other ADTs, and whether a program implements an ADT correctly.

The descriptions of operation semantics in this article will use algebraic semantics. Other mathematical notations such as logic could be used, but algebraic semantics seems to be the easiest to comprehend. In this case, algebraic semantics is simply the definition of the behavior of an ADT using an algebra. An algebra consists of one or more sets of values (called domains or sorts) and a set of operations on these domains. Simply stated:

$$\text{algebra} = <\text{domains, operations}>$$

The algebraic approach uses a set of statements (axioms) in a mathematical format to describe the properties of the operations. Operations in an algebra are usually described in terms of each other. Note that the data representation and implementation are not part of the algebraic specification.

In the description that follows, the algebraic statements that define each operation in terms of the domain will be accompanied by a verbal description to assist the reader in understanding the meaning of the statement.

Operation Form or Syntax

The operation form or syntax describes how the ADT is used, that is, what types of values or parameters are required for the operations and what type of values the operation provides or returns. Programming languages and mathematical descriptions use different approaches to specify the operations or interface to an ADT. We choose the programming language C to illustrate a way of specifying the interface independent of the implementation and an algebraic method to provide a mathematical description.

Now that we have presented a general description of the syntax and semantics of ADTs, in the next part of this article, we will describe the algebraic semantic operations and the syntax of a stack in detail so that the principles involved are clear. The algebraic semantics of the stack are defined first because this specifies the operations needed. Once the stack is thoroughly described, the algebraic semantics for other ADTs, including queue, set, and bag, are presented. The reader should be able to infer the program syntax from these semantic descriptions of operations.

THE STACK ADT

Informal Definition

A stack is a collection of items of the same data type in which only the most recently added item may be examined or removed. For example, the data type could be integers, real numbers, or something more complex, such as a tuple. If the most recently added item is removed, then the second most recently added item becomes the most recently added item and may be examined or removed. The stack is also known as a last-in-first-out data type or LIFO.

The subsequent discussion uses elements from the domain of natural numbers {1, 2, 3, . . .} as the data type to be placed on a stack. The symbol "N" is used to represent this domain. Using only natural numbers as the data type simplifies the discussion without affecting its generality. Changing to a different data type is straightforward as it only requires choosing a new domain such as $X = \{x_1, x_2, x_3, \ldots\}$. For example, if the data type is a tuple that consists of a name and an age, then the domain X would be the domain that consists of all name, age pairs.

Algebraic Semantics of Operations

Operations on a Stack. The normal definition of a stack S has the following basic operations:

- push(v, S) puts item v from N on stack S and returns a stack. Thus, the operation push operates on two domains, the domain N and the domain of all stacks of N, which is designated as S_t. The operation push returns a member of the domain S_t.
- popOff(S) removes the item from N most recently placed on the stack S and returns a stack. The function popOff only operates on the domain S_t returning a member of S_t.
- top(S) provides access to the item v from N, the item most recently placed on the stack S, and returns that value. The function top operates on the domain S_t and returns a value in the domain N.
- isEmpty(S) returns a value true (T) if stack S is empty and false (F) in all other cases. The function isEmpty is necessary to ensure that popOff and top do not try to manipulate an empty stack. The function isEmpty introduces the Boolean domain designated by B, which contains two members true (T) and false (F). The function isEmpty operates on the domain S_t and returns a value in the domain B.
- *new*() returns a new stack. The function new operates on the domain S_t and returns a specific member of S_t, namely, the empty stack.

Related Domains. Based on the previous discussion, it can be seen that the operations on a stack involve three domains:

N—the domain of natural numbers

S_t—the domain of all stacks of natural numbers including the empty stack

B—the domain of Boolean values (true, false)

Elements of the domain N are pushed onto a stack in S_t, and produce a stack in S_t. Elements of the domain N are produced when the top of the stack in S_t is examined, and elements of the domain S_t are produced when the top element in S_t is removed. Thus, elements of the domains N and S_t act as both input and output for the operations push, popOff, and top. Finally, the operation isEmpty acts on an S in S_t to produce elements of the domain B. Elements of B are only outputs of operations.

A domain is often called a sort, and the domain and the operations on the domain are called the signature of the algebra. A many-sorted algebra has operations based on more than one domain. Because an ADT uses values from several different sorts or domains, an ADT can be defined by a many-sorted algebra.

Algebraic Semantics of a Stack. Once the operations for a stack are understood, a set of precise mathematical identities or axioms that describe the behavior of the stack in terms of the operations must be created. These axioms are developed and explained progressively. Note that these identities use the "$\equiv$" sign to indicate that the two sides of the expression are identical.

Axiom 1—popOff(push(v, S)) $\equiv$ S

This axiom says that when an item v from the domain N is pushed onto S and then removed, the stack is the same. For example, if elements 1, 2, and 3 are in stack S with 3[1] at the top of the stack, then popOff(push(300, S)) produces S = (1, 2, 3) because 300 is pushed onto S by the push operation to produce S = (1, 2, 3, 300), and then the element 300 is removed by the popOff operation.

Axiom 2—top(push(v, S)) $\equiv$ v

Here, when an item v is pushed onto S from the domain N, the function top provides access to that item. For example, if a new stack is produced by new(), then top(push(5, push(2, new())) will produce 5 and leave the stack as (2, 5).

Axiom 3 for isEmpty (S):

Axiom 3a—isEmpty(new()) $\equiv$ true

The stack generated by new() is the empty stack. Testing the empty stack with the operation isEmpty produces true.

Axiom 3b—isEmpty(push(v, S)) $\equiv$ false

The push operation puts one element on the stack, so even if S is empty, the stack generated by push is not. Therefore, isEmpty returns false.

These three axioms define the complete behavior of a stack. However, an additional operation can be introduced, namely, the operation pop, because pop is used later in the syntactic description of the stack as a matter of programming convenience. The operation pop is defined in terms of top and popOff. The operation pop returns the value of the top item on the stack, removes the item most recently placed on the stack S, and returns a stack. Thus, pop returns a tuple (v, S) where v belongs to N and S belongs to S_t. Thus, a new domain T has been introduced, namely, all pairs of values from N and S_t. The operation of combining two domains to make tuple is called the Cartesian product, in this case, the Cartesian Product of N and S_t.

Axiom 4—pop(push(v, S))$\equiv$(top(push(v,S)), popOff(push (v,S))) $\equiv$ (v, S)

This axiom defines the pop operation. An item v is placed on the stack with push; top provides access to the data item v, and popOff removes v leaving the stack S. Thus, the result of the pop operation is a tuple that consists of the top element and the remaining stack. In a stack (1, 4 ,8), the operation pop(push(5, (1, 4, 8)) creates a stack (1, 4, 8, 5), and pop(1, 4, 8, 5) produces (5, (1,4,8)) which provides access to 5, removes 5, and leaves the stack (1,4, 8).

[1] Note that stacks are written from left to right and the last added element is the rightmost element in the sequence.

The axioms popOff and top have assumed that the stack always has an entry. We could define a new domain E containing one element, namely, the value "error," and two corresponding axioms for popOff and top acting on an empty stack.

Axiom 5—popOff(new()) ≡ error

This axiom states that removing the top element of an empty stack produces an error.

Axiom 6—top(new()) ≡ error

This axiom states that examining the top element of an empty stack produces an error.

The equations just presented define equivalences between syntactic elements; they specify the transformations that are used to translate from one syntactic form to another. This mathematical description of an axiom for an operation on an ADT is known as a rewrite rule, which means which that any axiom "X ≡ Y" is a statement that any occurrence of an X can be replaced with a Y. For example, whenever the expression or sequence of symbols of the form

popOff(push(v, S))

is encountered, it can be replaced by the symbol S. If

popOff(push(w, T))

is encountered, then, similarly, it can be replaced by the symbol T.

Syntax of Operations

Interface. In this section, several programs for a stack that is written in the programming language C and uses the information-hiding principle are described to show how to produce an ADT in practice. The presentation is based on a calculator program adapted from Ref. 25. Hopefully, the commentary that accompanies the programs is self-explanatory. However, if the reader needs more details about the language C, then Ref. 25 is an excellent reference.

The calculator program, which is shown next, uses reverse polish (also called postfix) notation instead of infix because it makes a simple demonstration of the stack. Some

```
/* Stack example using an array */
#include <stdio.h>
#include <stdlib.h>      /* for atof() */
#define MAXOP     100   /* max size of operand or operator */
#define NUMBER    '0'   /* signal that a number was found */

#define MAXVAL     100   /* max depth of val stack */
int sp;                  /* next free stack position */
double stack[MAXVAL];    /* value stack as an array */
int getop(char[]);
void new();
void push (double);
double pop (void);

/* reverse Polish calculator */

main()
{
          int type;
         double op2;
         char s[MAXOP];
         new();
         while ((type = getop(s)) != EOF)
         {
              switch (type)
              {
        case NUMBER:
              push(atof(s));
              break;
         case'+':
              push(pop() + pop());
              break;
        case '*':
              push(pop() * pop());
              break;
         case'-':
              op2 = pop();
              push(pop() - op2);
              break;
         case'/':
              op2 = pop();
              if(op2!=0.0)
                   push(pop() / op2);
              else
                   printf("error: zero divisor\n");
              break;
         case '\n':
              printf("\t%.8g\n", pop());
              break;
         default:
             printf("error: unknown command %s\n", s);
                   break;
              }
         }
         return 0;
}
 /* new; initialize the stack */
void new()
{
          sp = 0;
}
/* is Full: is stack full? */
int isFull()
{ if(sp<MAXVAL)
              return 0;
          else
              return 1;
}
/* isEmpty: is stack empty? */
int isEmpty()
{
if(sp>0)
             return 0;
          else
             return 1;
}
/* push: push f onto value stack */
void push (double f)
{
    if (!isFull())
         stack[sp++] = f;
     else
         printf("error: stack full can not push %g\n", f);
}
/* top: return top value from the stack */
double top()
{
           return stack[sp-1];
}
/* popOff: remove top elementof stack */
void popOff()
{
           --sp;
}
/* pop: pop and return top value from stack */
double pop()
{
           double top_value;
           if(!isEmpty())
           {
                 top_value = top();
                 popOff();
                 return top_value;
            }
            else
            {
                 printf("error: stack empty\n");
                 return 0.0;
            }
```

```
#include <ctype.h>
int getch(void);
void ungetch(int);

/* getop: get next operator or numeric operand */
int getop(char s[])
{
          int i, c;
          while ((s[0] = c = getch()) == '' || c == '\t')
                ;
          s[1] = '\0';
          if (!isdigit(c)&& c !='.')
                    return c;           /* not a number */
          i = 0;
          if(isdigit (c))      /* collect integer part */
                   while (isdigit (s[++i] = c = getch ()))
                         ;
          if(c == '.')         /* collect fraction part */
                   while (isdigit (s[++i] = c = getch ()))
                         ;
          s[i] = '\0';
          if(c!=EOF)
                 ungetch (c);
          return NUMBER;
}

#define BUFSIZE 100
char buf[BUFSIZE];    /* buffer for ungetch */
int bufp = 0;         /* next free position in buf */

int getch(void)     /*get a (possibly pushed back) character*/
{
        return (bufp > 0) ? buf[--bufp] : getchar();
}
void ungetch(int c)       /* push character back on input */
{
        if (bufp >= BUFSIZE)
              printf("ungetch: too many characters\n");
        else
            buf[bufp++] = c;
}
```

calculators and programming languages, such as Postscript (27), use reverse polish notation. In reverse polish notation each operator follows its operands; so the familiar infix expression

$$(3+2)*(6-2) \text{ is written as } 3\ 2 + 6\ 2 - *$$

Parentheses are not needed in postfix notation, and it is unambiguous, as long as we know how many operands go with each operator. In this example, each operator expects two operands.

The implementation of the calculator is simple. When each operand is encountered, as the expression is scanned from left to right, the operand is pushed on the stack. When an operator occurs, the stack is popped the correct number of times (in this case, always two), the operator is applied to the operands, and the result is pushed back on the stack.

In the example just described, 3 and 2 would be pushed on the stack, the "+" operator would cause the stack to be popped twice, and the result 5 would be computed and be pushed on the stack. Next, the 6 and 2 would be pushed on the stack, and then the "−" operator would cause the stack to be popped twice with the result 4 being pushed back on the stack. Finally, the operator "∗" would cause the stack to be popped twice (remember 5 and 4 are on the stack), which would produce the final result of 20. Note that the equal sign (=) is used for assignment, whereas the identity operator and its inverse are coded as "==" and "!=", respectively.

Note that the operations described for a stack have been used in this example so that the syntactic form and algebraic semantic form look similar. An actual operational program could be simplified by omitting top, popOff, and ifEmpty and coding them inline.

The algebraic semantics approach always assumes that elements in the domain S_t can grow without bound; in other words, no bound exist on the size of memory available for a stack. Of course, programs run in large, but finite, memory, and a limit to memory size and thus a limit to the size of a stack does exist. Therefore, the function *isFull* is introduced to indicate when no more memory is available so appropriate action can be taken.

The part of the program labeled main{} is the calculator and consists of a loop that "reads" operands (decimal numbers) and operators (+, *, -, and /). Inside the loop is a case statement that pushes numbers on the stack or pops numbers, applies the operator, and pushes the result back on the stack. One or two functions need explaining. The function getop uses other functions to collect the characters comprising an operand or operator from the input, the function atof converts the collection of characters constituting an operand (digits and a decimal point) into a decimal or floating point number.

Notice that the program uses an array for the stack that is defined by the statements:

```
#define MAXVAL 100         /* max depth of val stack */
int sp;                    /* next free stack position */
double stack[MAXVAL];      /* value stack */
```

The stack is an array of 100 elements of floating point numbers (double). To demonstrate data abstraction, the statements that define an array will be replaced by statements that define a linked list, namely:

```
struct stack_element {
        double data;                     /* data value */
        struct stack_element * next;     /* pointer to next
                                            data element */
};
struct stack_element * stack_top;        /* top of stack */
```

The statement "struct stack_element" defines a new data type that contains a number and a pointer to the next stack element. The asterisk (*) indicates that next is a pointer to a data type called stack_element. The definition in this case is recursive. The functions new, isFull, isEmpty, push, top, popOff and pop are replaced by new functions with the same name where the differences between each function is indicated by boldface type.

```
/* new; initialize the stack */
void new()
{
                stack_top = NULL;
}
/* is Full: is stack full? */
int isFull(struct stack_element * s)
{
if (s !=NULL)
                    return 0;
          else
                    return 1;
}
/* isEmpty: is stack empty? */
int isEmpty()
{
if(stack_top !=NULL)
                return 0;
           else
```

```
                    return 1;
/* push: push f onto value stack */
void push (double f)
{
      struct stack_element * p;     /* new stack element */
      p = malloc(sizeof(struct stack_element));
      if(!isFull(p))
      {
          (*p).data = f;
          (*p).next = stack_top;
          stack_top = p;
      }
      else
          printf("error: stack full can not push %g\n", f);
}
/* top: return top value from the stack */
double top()
{
     return (*stack_top).data;
}
/* popOff: remove top elementof stack */
void popOff()
{
     struct stack_element * temp;
     temp = stack_top;
     stack_top = (*stack_top).next;
     free(temp);
}
/* pop: pop and return top value from stack */
double pop(void)
{
     double top_value;
     if(!isEmpty())
     {
           top_value = top();
           popOff();
           return top_value;
     }
     else
     {
           printf("error: stack empty\n");
           return 0.0;
     }
}
```

Notice that substitution of a linked-list implementation for an array implementation does not require a change in the program main{}, that is, the program using these operations. This is what makes this data type abstract; the use of its operations is independent of the underlying implementation, but not of the interface.

The program can be generalized by allowing the push and pop operations to have a stack as an argument, which, thus, allows them to be used in more general situations. A version of the program main{} that uses an array for the stack and the functions new, isFull, isEmpty, push, top, popOff and pop with arguments follows. Programs written in the language C follow the call-by-value convention in which they always pass the values of a parameter. C uses an “asterisk” sign (*) to indicate that a variable is a pointer and the “and” sign (&) to indicate the value associated with a pointer. The function *malloc* creates a block of memory the same size as an element of the data type. Note that the functions for getop are not included because they do not change from implementation to implementation, which is another example of data abstraction.

```
/* Stack example using an array with arguments */
#include <stdio.h>
#include <stdlib.h>    /* for atof() */
#define MAXOP   100    /* max size of operand or operator */
#define NUMBER ’0’    /* signal that a number was found */
#define MAXVAL 100    /* max depth of val stack */
struct stack_type {
    int sp;                  /* next free stack position */
    double array [MAXVAL];  /* value stack */
};
struct stack_type * stack;
int getop(char[]);
void new(struct stack_type **);
void push (double, struct stack_type *);
double pop (struct stack_type *);

/* reverse Polish calculator */
main()
{
      int type;
      double op2;
      char s[MAXOP];
      new(&stack);
      while ((type = getop(s)) != EOF)
{
      switch (type)
      {
      case NUMBER:
            push(atof(s), stack);
            break;
      case’+’:
            push(pop(stack) + pop(stack), stack);
            break;
      case ’*’:
            push(pop(stack) * pop(stack), stack);
            break;
      case’-’:
            op2 = pop(stack);
            push(pop(stack) - op2, stack);
            break;
      case ’/’:
            op2 = pop(stack);
            if(op2!=0.0)
                  push(pop(stack) / op2, stack);
      else
                  printf("error: zero divisor\n");
            break;
      case ’\n’:
            printf("\t%.8g\n", pop(stack));
            break;
      default:
            printf("error: unknown command %s\n", s);
            break;
      }
  }
  return 0;
}
/* new; initialize the stack */
void new(struct stack_type ** s)
{
        *s = malloc(sizeof(struct stack_type));
        (**s).sp = 0;
}
/* is Full: is stack full? */
int isFull(struct stack_type * s)
{
if((*s).sp<MAXVAL)
                return 0;
          else
                return 1;
}
/* isEmpty: is stack empty? */
int isEmpty (struct stack_type * s)
{
if((*s).sp>0)
                return 0;
          else
                return 1;
}
/* push: push f onto value stack */
void push (double f, struct stack_type * s)
{
     if(!isFull(s))
          (*s).array[(*s).sp++] = f;
     else
          printf("error: stack full can not push %g\n", f);
}
/* top: return top value from the stack */
double top(struct stack_type * s)
{
             return (*s).array[(*s).sp-1];
}
 /* popOff: remove top elementof stack */
void popOff( struct stack_type * s)
{
           --(*s).sp;
}
/* pop: pop and return top value from stack */
double pop(struct stack_type * s)
{
          double top_value;
          if (!isEmpty(s))
          {
                 top_value = top(s);
                 popOff(s);
                 return top_value;
          }
          else
          {
                 printf("error: stack empty\n");
                 return 0.0;
          }
}
```

Again, the statements defining the stack as an array, namely,

```
struct stack_type {
     int sp;                /* next free stack position */
     double array[MAXVAL]; /* value stack */
};
struct stack_type * stack;
```

are replaced by the statements defining a list:

```
struct stack_element {
   double data;                  /* data value */
   struct stack_element * next;   /* pointer to next data
element */
};
struct stack_element * stack; /* top of stack */
```

Further more, the functions new, isFull, isEmpty, push, top, popOff and pop are replaced by new functions with the same name where the differences between each function is indicated by boldface type.

```
/* new; initialize the stack */
void new(struct stack_element ** s)
{
                *s = NULL;
}
 /* is Full: is stack full? */
int isFull(struct stack_element * s)
{
if (s !=NULL)
             return 0;
     else
             return 1;
}
/* isEmpty: is stack empty? */
int isEmpty (struct stack_element * s)
{
 if (stack != NULL)
             return 0;
       else
             return 1;
}
/* push: push f onto value stack */
void push (double f, struct stack_element * s)
{
     struct stack_element * p;        /* new stack element */
     p = malloc(sizeof(struct stack_element));
     if(!isFull(p))
     {
           (*p).data = f;
           (*p).next = stack;
           stack = p;
      }
     else
         printf("error: stack full can not push %g\n", f);
}
 /* top: return top value from the stack */
double top(struct stack_element * s)
{
        return (*stack).data;
}
/* popOff: remove top elementof stack */
void popOff( struct stack_element * s)
{
         struct stack_element * temp;

         temp = stack;
         stack = (* stack).next;
         free(temp);

/* pop: pop and return top value from stack */
double pop(struct stack_element * s)
{
         double top_value;
         if (!isEmpty(s))
         {
                 top_value = top(s);
                 popOff(s);
                 return top_value;
          }
          else
          {
                printf("error: stack empty\n");
                return 0.0;
           }
}
```

This section is intended to illustrate how a specific ADT, namely, a stack, may be implemented in practice. Similar techniques apply to all ADTs. The remainder of the article focuses on the definition for several ADTs using algebraic semantics. Included are the abstract data types (ADTs) for queue, set, and bag.

THE QUEUE ADT

The queue is the next ADT to be examined because it is similar to the stack in operation. However, the axioms for the queue involve recursion.

Informal Definition

A queue is a collection of items in which only the earliest added item may be accessed. The queue, just like an orderly line to board a bus or airplane, has a head and a tail. Items at the head are the next to board the bus or plane or to be accessed. Items joining the queue join at the tail. If no other item is added to the tail, then this item would be the last to be accessed. The queue is also known as a first-in-first-out data type or FIFO. Similar to the stack, this presentation about queues only uses elements from the domain of natural numbers as the data type to be placed on a queue. All possible queues of natural numbers themselves form a domain that is named Q_u.

Operations on a Queue

The queue Q has the following basic operations:

- add(v,Q) puts item v from the domain N on a queue Q and returns a new queue Q. The function combines elements from two domains N and the domain of all queues of N that is designated by Q_u, and produces an element from the domain Q_u.
- front(Q) provides access to the value of the earliest added item placed on the queue Q. The operation front

takes an element from the domain Q_u and produces an element from the domain N.

- remove(Q) removes the item placed first on the queue Q. The function or operation remove takes an element from the domain Q_u and produces an element in the domain Q_u.
- isEmpty(Q) returns a value true (T) if Q is empty and false (F) in all other cases. Thus, isEmpty maps an element from the domain $Q_u$to the domain B (Boolean).
- new() returns a new queue. The function new operates on the domain Q_u and returns a specific member of Q_u, namely, the empty queue.

Related Domains

Based on the previous discussion, it can be seen that the operations on a stack involve three domains:

N—the domain of natural numbers

Q_u—the domain of all queues of natural numbers

B—the domain of Boolean values

Note that we could introduce the domain containing "error," but we do not include it in this and subsequent examples.

Algebraic Semantics of Operations

Once the operations are defined for a queue, the axioms that describe their behavior are produced. These axioms are developed and explained progressively. Note that axioms 1 and 2 each have two parts because the axioms are recursive. The first part of the axiom is the result for a queue of length 1, whereas the second part of the axiom deals with the case where the queue has more than one element. An analysis of the recursion shows the operation moving recursively from the tail to the head of the queue to locate the element to be returned and to be removed. In other words, the recursive definition reduces a queue of length greater than 1 one step at a time until the resulting queue is of length 1 and the front element of the queue can be accessed or removed.

Axiom 1 for front(Q):

Axiom 1a—front(add(v, (new())) ≡ v

This axiom states that the front operation on a queue of length 1 returns the single element in the queue. In other words, when an item v is added to an empty queue, then it is the front item on the queue.

Axiom 1b—front(add(v, add(w, Q))) ≡ front(add(w, Q))

This axiom defines the general recursive definition of the front operation. It states that the front of a queue with elements v and w added to a queue Q is the same as the front of the Q with only the element w added. The operation front is applied repeatedly to reduce the length of the queue to a queue of length 1 where Axiom 1a can be applied. For example, if Q has elements 5, 7 and the front is 7, then the front of (v, w, 5, 7) is the same as the front of (w, 5, 7). This queue then is processed by the same expression, namely, Axiom 1b to produce (5, 7) and finally (7). The queue (7) then is processed by Axiom 1a to produce 7 as the front.

Axiom 2 for remove(Q):

Axiom 2a—remove(add(v, new())) ≡ new()

This axiom states that removing a single item from a queue of length 1 produces the empty queue.

Axiom 2b—remove(add(v, add(w, Q))) ≡ add(v, remove (add(w, Q)))

This axiom provides the recursive definition of the remove operation. It states that removing the head of the queue (v, w, Q) is the same as removing the head of the queue (w, Q). Again, the axiom is applied recursively to reduce the queue to a length of 1 where Axiom 2a is applied. If Q has elements 5, 7 and the front is 7, then remove(add(v, add(w, 5, 7))) is the same as add(v, remove (add(w, 5, 7))), which is the same as add(v, add(w, remove(add(5, 7)))), which becomes add(v, add(w, add(5, remove(7, new()))))). Of course, remove(7, new()) is new(), which is the empty stack. Thus, the queue is (v, w, 5) with the 7 removed.

Axiom 3 for isEmpty():

Axiom 3a—isEmpty(new())) ≡ true

A new queue is empty, and therefore isEmpty returns the value true (T).

Axiom 3b—isEmpty(add(v, Q)) ≡ false

For a queue that has at least one element, isEmpty returns the value false (F).

These three axioms define the complete behavior of a queue. However, an additional operation delete could be introduced as a matter of programming convenience. The operation delete is defined in terms of front and remove. The operation delete returns the value of the front item of a queue and removes this front item. Thus, delete returns a tuple (v, Q) where v belongs to N and Q belongs to Q_u. Thus, a new domain T has been introduced, namely, the domain of all pairs of values from N and Q_u.

Axiom 4—delete(add(v, add(w, Q))) ≡ (front(add(v, add(w, Q))), remove(add(v, add(w, Q)))

This axiom states that deleting the item at the head of the queue Q produces a tuple of two elements, namely, the item at the front of the queue and the queue itself. For example, delete operating on the queue (v, w, 3) produces the tuple containing the element 3 and the queue (v, w).

THE SET ADT

Informal Definition

A set is an unordered collection of elements where each element occurs at most once. A set has three properties: (1) All elements belong to a universe or domain, (2) either each element is a member of the set or it is not, and (3) the elements are unordered. The statement "all elements belong to a universe" needs some explanation. A universe encompasses all elements that have at least one common property. For example, the universe of all black chairs encompasses all chairs that are black. They could be made of wood, leather, or other materials. Another example is the universe or domain of all grades in a course where the numeric grades are integers from 0 to 100. A set could be some of these grade values. Because an element of a set can occur only once, if two students receive a grade of 72, the grade 72 only appears once in the set.

Operations on a Set

A set S has the following basic operations:

- add(v, S) adds an element v to a set S if v is not already in the set and returns a set S. The function add operates on the domain N of natural numbers and the domain of all sets of N, which is designated by S_e, and produces a set in the domain S_e.
- isIn(v, S) determines if a set S contains the element v and returns either true (T) or false (F). Thus isIn operates on the domain of all sets of natural numbers S_e and returns a value in the Boolean domain B.
- remove(v, S) removes an element v from the set S if v is in the set. The operation remove operates on the domain N of natural numbers and the domain S_e and produces a set in the domain S_e.
- isEmpty(S) determines if the set has no elements and returns either true or false. Thus isEmpty operates on the domain of all stacks of natural numbers S_e and returns a value in the Boolean domain B.
- new() returns an empty set. The function new operates on the domain S_e and returns a specific member of S_e.

Related Domains

Based on the previous discussion, it can be seen that the operations on a stack involve three domains:

N—the domain of natural numbers

S_e—the domain of all sets of natural numbers

B—the domain of Boolean values

Semantics of Operations

Now that the operations have been defined, we provide their definition through a set of axioms. Axioms for isIn, add, remove, and isEmpty exist where isIn, add, and remove have three parts and isEmpty has two parts. Multiple axioms are required for each of these operations to cover all situations. Note that the operations add, isIn, and remove are recursive.

Axiom 1 for add(v,S):

Axiom 1a—add(v, add(v, S)) ≡ add(v, S)

A set has only one occurrence of an element. Adding an element twice is the same as adding it once.

Axiom 1b—add(v, add(u, S)) ≡ add(u, add(v, S)) v ≠ u

This axiom states that elements that are not the same can be added to a set in either order.

Axiom 2 for isIn(v, S):

This operation has the value true (T) or false (F).

Axiom 2a—isIn(v, new()) ≡ false

An empty set does not contain any elements, so the element v is not in the set. The operation is false.

Axiom 2b—isIn(v, add(v, S)) ≡ true

A set that just has had v added must contain v. If v was already in the set then the new v will not be added. Thus, v is in the set whether it is added by the add operation or not, and so the isIn operation is true.

Axiom 2c—isIn(v, add(u, S)) ≡ isIn(v , S) if v ≠ u

This axiom provides a recursive definition of the isIn operation. The operation states that v is not the most recently added element to S, but it may be in the remainder of S. For example, the elements 5, 4, 3, and 7 can be added to S using the operations add(7, add(3, add(4, add(5, new())))). Then, isIn(4, add(7, add(3, add(4, add(5, new()))))) could be replaced by isIn(4, add(3, add(4, add(5, new())))), which could be replaced by isIn(4, add(4, add(5, new()))), which is true.

Axiom 3 for remove(v, S):

Axiom 3a—remove(v, new()) ≡ new()

Removing the element v from the empty set produces the empty set.

Axiom 3b—remove(v, add(v, S)) ≡ remove(v, S)

The first step in this axiom adds the element v to the set S. If v is already in S, then adding v produces S, which gives the right-hand side of the identity. If v is not in S, then we get S, which can also be written as remove(v, S).

Axiom 3c—remove(v, add(u, S)) ≡ add(u, remove(v, S)) if v ≠ u

This axiom states that removing v from the set that results when u is added to S (v ≠ u) is the same as add u to the set that results when v is removed from S.

Axiom 4 for isEmpty(S):

Axiom 4a—isEmpty(new())) ≡ true

A new set is empty, and therefore isEmpty returns the value true (T).

Axiom 4b—isEmpty(add(v, S)) ≡ false

For a set that has at least one element, isEmpty returns the value false (F).

THE BAG ADT

Informal Definition

A bag is an unordered collection of elements where each element can occur more than once. A bag has three properties: (*1*) All elements belong to a universe or domain, (*2*) either each element of the universe or domain is a member of the bag or it is not, and (*3*) the elements are unordered. The statement "all elements belong to a universe" has already been explained under the discussion of the set ADT. The example of all grades in a course shows how a set and bag differ. For example, if two students receive a grade of 72, then the grade 72 could only appear once in the set of grades but twice in a bag of grades.

Operations on a Bag

A bag S (we use S for sack, a synonym for a bag, as B is already taken as the symbol for the Boolean domain) has the following basic operations:

- add(v, S), which adds an element to the bag.
- isIn(v, S), which tells whether an element is in the bag.

- numberIn(v, S), which tells how many times an element is in the bag.
- remove(v, S), which removes an element from the bag.
- isEmpty(S), which returns true if the bag is empty and false otherwise.
- new(), which returns a new empty bag from the domain of all bags.

Related Domains

Based on the previous discussion, it can be seen that the operations on a stack involve three domains:

N – the domain of natural numbers
S_b – the domain of all bags of natural numbers
B – the domain of Boolean values

Axiom 1—add(v, add(u, S)) $\equiv$ add(u, add(v, S))

This axiom states that the elements can be added in either order and achieve the same result because all elements are added to the bag.

Axiom 2 for isIn(v, S):

Axiom 2a—isIn(v, new()) $\equiv$ false

The element is not in the empty bag, and so the operation is false.

Axiom 2b—isIn(v, add(v, S)) $\equiv$ true

The element is in the bag because it has just been added, and so the operation is true.

Axiom 2c—isIn(v, add(u, S)) $\equiv$ isIn(v, S) if v $\neq$ u

The element v is not in the bag produced by adding u to S, but it could be in S itself.

Axiom 3 for numberIn(v, S):

Axiom 3a—numberIn(v, new()) $\equiv$ 0

The number of times the element v is in the empty bag is zero.

Axiom 3b—numberIn(v, add(v, S)) $\equiv$ 1 + numberIn(v, S)

If the element v is added to S, then the count of v is one more than the number of times v is in S.

Axiom 3c—numberIn(v, add(u, S)) $\equiv$ numberIn(v, S) if v $\neq$ u

If the element u is added to S where v and u are not the same, then the count of v is the same as the number of times v is in S.

Axiom 4 for remove(v, S):

Axiom 4a—remove(v, new()) $\equiv$ new()

Removing v from the empty bag produces the empty bag.

Axiom 4b—remove(v, add(v, S)) $\equiv$ S

Removing v after it has been added to S produces S.

Axiom 4c—remove(v, add(u, S)) $\equiv$ add(u, remove(v, S)) if v $\neq$ u

Removing v from S augmented by u is the same as removing v from S when v and u are not the same.

The axiom isEmpty is not defined because it is similar to the ones for data types previously discussed.

SUMMARY

This article provides a comprehensive introduction to methods for describing and implementing abstract data types (ADTs). Many other ADTs could be described, including maps, which can be used to define dictionaries and priority queues. More information can be found in the references. A very complete formal description of ADTs is contained in Ref. 1, although the mathematical symbolism requires some translation into the notation used in this article.

BIBLIOGRAPHY

1. H. A. Partsch, *Specification and Transformation of Programs: A Formal Approach to Software Development*. Springer-Verlag, 1990.
2. Available: http://www.cs.uiowa.edu/~slonnegr/plf/Book/Chapter12.pdf.
3. O.-J. Dahl, K. Nygaard, *SIMULA: An ALGOL-based simulation language, Communications of the ACM*, **9**(9): 671–678, 1966.
4. D. L. Parnas, On the criteria to be used in decomposing systems into modules, *Communications of the ACM*, **15**(12): 1053–1058, 1972.
5. D. L. Parnas, A technique for software module specification, *Communications of the ACM*, **15**: 330–336, 1972.
6. B. Liskov, S. Zilles, Programming with abstract data types, *SIGPLAN Notices*, April 1974.
7. USA Department of Defense, *Reference Manual for the Ada Programming Language*, ANSI/MIL-STD-1815 A, 1983.
8. B. Liskov, A. Snyder, R. Atkinson and C. Schaffert, Abstraction mechanisms in CLU, *Communications of the ACM*, **22**: 564–576, 1977.
9. A. Goldberg and D. Robson, Smalltalk-80: *The Language and Its Implementation*. Addison-Wesley, 1983.
10. B. Stroustrup, *The C++ Programming Language*, 3rd ed., special ed. Addison-Wesley, 2000.
11. K. Arnold, J. Gosling, and D. Holmes. *Java Programming Language*, 4th ed., Sun Microsystems.
12. S. Zilles, Procedural encapsulation: A linguistic protection mechanism, *SIGPLAN Notices*, **8**(9): 142–146, 1973.
13. J. Guttag, Abstract data types and the development of data structures, *Communications of the ACM*, June 1977.
14. J. Guttag, E. Horowitz and D. Musser, The design of data type specifications, *International Conference on Software Engineering (ICSE)*, 1976, pp. 414–420.
15. J. Goguen, J. Thatcher, E. Wagner and J. Wright, Initial algebra semantics and continuous algebras, *J. ACM*, **24**(1): 68–95, January 1977.
16. J. Guttag and J. Horning, *An Introduction to the Larch Shared Language*, IFIP, 1983.
17. E. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification: Equations and Initial Semantics*, Springer-Verlag, 1985.
18. J. Goguen and J. Meseguer, Eqlog: Equality, types, and generic modules for logic programming, in *Functional and Logic Programming*. Prentice-Hall, 1986, pp. 295–263.
19. J. Mitchell and G. Plotkin, Abstract types have existential type, *Proc. ACM Symposium on Principles of Programming Language, 1986*.
20. C. A. R. Hoare, Proof of the correctness of data representations, *Acta Informatica*, **1**: 271–281, 1972.
21. C. B. Jones, *Systematic Software Development Using VDM*. Prentice Hall, 1990.

22. J. M. Spivey, *Z Reference Manual*. Prentice-Hall, 1989.

23. D. D. Cowan and C. J. P. Lucena, Abstract data views: An interface specification concept to enhance design for reuse, *IEEE Trans. on Software Engineering*, **21**(3): 229–243, 1995.

24. P. S. C. Alencar, D. D. Cowan and C. J. P. Lucena, A logical theory of interfaces and objects, *IEEE Trans. on Software Engineering*, **28**(6): 548–575, 2002.

25. B. W. Kernighan, and D. M. Ritchie, *The C Programming Language*, 2nd ed., Prentice Hall Software Series, 1988.

26. Available: http://www.nist.gov/dads/HTML/abstractDataType.html NIST.

27. G. C. Reid, *Thinking in PostScript*. Addison-Wesley, 1990.

DONALD COWAN
P.S.C. ALENCAR
University of Waterloo
Waterloo, Ontario, Canada

AGENT-ORIENTED SOFTWARE ENGINEERING

INTRODUCTION

Agent-oriented software engineering (AOSE) is an approach to construct software systems based on the *agent* paradigm.[1] An agent can be thought of as an *autonomous* and *social* entity, which can communicate, coordinate, and cooperate with other agents to achieve goals. This notion of agent offers a higher-level abstraction than the notion of object used in object oriented (OO) software engineering.

AOSE approaches were originally proposed to engineer software agents. However, as soon as heterogeneous, networked software systems began to appear, machine-oriented views of computing appeared to be inappropriate to understand the complexity of these systems and to design them. AOSE is being recognized as a promising approach, offering human-oriented abstractions when designing these software systems (1). Along this view, agents are not only building blocks of a software systems, but also are proposed as design abstractions.

More generally, it can be said that AOSE provides computational abstractions, models, and tools with which to conceptualize and implement distributed software systems, whether they are realized as software agents or not. In the following article we will call them agent-oriented (AO) systems.

To help understanding the complexity of these systems and the type of problems we encounter building them, it can be useful to consider some applications of software agents and of current distributed systems, along with their properties.

Systems that operate in high-risk situations that are unsuitable or impossible for humans, like control systems on board robotic spacecraft for deep space exploration, were initially considered "killer" applications for software agents. For instance, the NASA space exploration mission *Deep Space 1* in 1999 experimented with agent-based software to manage exploration tasks in a flexible way. This software has been named *Remote agent*.[2] The *Remote agent* software could plan and execute many activities on board the spacecraft, being given only general mission goals from ground controllers on the Earth. An example goal might have been to "take asteroid pictures for navigation every 2 days for 2 hours" or "turn off the camera once you are done using it." The *Remote agent* monitored the execution of a plan that had been generated to accomplish a goal, to assess unforeseen failure conditions, and to decide to change a plan accordingly. For instance, it could recognize false information sent by a failed sensor and correctly ignore it. A main feature *of Remote agent* is that of behaving autonomously and in a goal-directed way.

In a domestic scenario, Robot vacuum cleaners are available today that behave autonomously (once switched on), having the ability to perceive their environment and adjusting their actions to achieve their design objectives, such as cleaning a room.

The Internet and the availability of distributed computational resources and services offer tremendous technological challenges and opportunities for AO systems. When planning for a journey or a vacation, we can access a virtual travel agency on the Internet to get an offer for a travel package that consists of flights, hotel rooms, and car rental, taking into account our date constraints and preferences, and we may eventually buy it, paying by credit card, in a secure way.

Current e-commerce applications provide product discovery and packaging (e.g., flight, hotel, car rental). Moreover, they perform simple bidding actions on behalf of a user. Research is progressing toward developing systems that act as brokers, engaging negotiation activities in the context of multiple auctions, taking into account customer preferences and constraints.[3]

The main feature of this type of agent-based system is the ability to interact with other systems following high level protocols, such as electronic auctions, and to decide how to perform a bid according to a specific policy that fit the customer's objectives.

Web 2.0, that is, the second generation of web-based communities and hosted services (such as social-networking sites, wikis, and folksonomies), which aim to facilitate collaboration and sharing between users, is also presenting tough software engineering problems. Complexity here derives from the heterogeneity of the platforms and networks these systems operate on; the diversity of their users, with different needs and preferences that evolve continuously; and the dynamicity of their operating environment.

Intervehicle communications for improving traffic safety and efficiency are also worth mentioning as a challenging application scenario for AO systems. A car can communicate with neighboring vehicles with the aim of coordinating at critical points (e.g., blind crossing, highway entries) or in case of critical events (e.g., accident, fog). These types of functions are particularly useful on secondary roads, which cannot be equipped with an information and warning service infrastructure (such as highways have). Communication between heterogeneous systems is a primary issue to be addressed together with the need of making these systems aware of their local environment, which changes while moving.

Summarizing, AO systems are software systems that behave in a *goal-directed* manner, either recognizing and adopting users' goals or being driven by their own goals. They are *situated*, that is, they are aware of their operational context and are *autonomous,* being requested to

[1] Herein, see Intelligent Agent.

[2] http://asc.arc.nasa.gov/projects/remote-agent/faq.php.

[3] Yearly competitions are organized to verify research advancements and to further stimulate it, http://www.sics.se/tac/page.php?id=1.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

respond dynamically to changing circumstances while trying to achieve a goal. They *interact* with other systems using high-level communication protocols. They act on behalf of humans who may have different needs and preferences, which *evolve* continuously. These software systems can be implemented as software agents, using agent programming environments, or as software components deployed in a distributed system platform. The most challenging application areas, accordingly to Luck (2), are *ambient intelligence,* which foresees an environment of potentially thousands of embedded and mobile devices that interact to support user-centred goals and activities. *Grid computing* should enable the efficient use of distributed computational resources, and *Electronic Business* supports the automation of information gathering and purchase transactions over the Internet.

Having to deal with such complex application domains, how can we understand them properly to identify requirements and properties of suitable software systems? That is, how can we design software with properties of autonomy and social interaction? How can we implement and test them? In other words, which type of software engineering methods and tools are appropriate to build such a system?[4]

AOSE aims at addressing these questions. First, it proposes to adopt the notion of *agent* and its related notions as a conceptual paradigm to understand and specify properties of AO systems.

Second, AOSE offers methodologies for analyzing the requirements and the software architecture of these software systems. AOSE methodologies adopt a visual modeling language, based on the agent paradigm, define models to be built during requirements analysis and system-design phases. They often offer specialized techniques for the analysis of these models.

Third, AOSE can offer structured processes that guide the development of the software system from early requirements analysis down to the implementation in terms of agent code or software components of a distributed system platform.

Content

In the following, the agent paradigm and examples of modeling languages based on it will be described. AOSE methodologies and examples of tool-supported software development processes will be also illustrated. Research on challenging issues in the AOSE area is very active. Some of the most promising efforts and trends will be recalled briefly.

THE AGENT-ORIENTED PARADIGM

On Agent Definition

The concept of software agent has evolved from artificial intelligence (AI) research areas, and in particular from distributed AI work, which date back to 1980. Different definitions have been proposed since then (3,4), which show influence from a variety of disciplines, including economics, philosophy, logic, ecology, and social science. This fact also motivates the use of the agent paradigm along different perspectives.

Adopting the software engineering perspective, common properties referred by all the different definitions and that are considered making the agent paradigm disruptive with respect to previously adopted software engineering paradigms (2) are represented by the following:

- *Autonomy and Situatedness.* An agent is a computer program that encapsulates some state (also called mental state) that is not accessible to other agents, and it makes decisions about what to do, based on this state, without the direct intervention of others. An agent can perceive the environment in which it operates, through appropriate sensors, and respond in a timely fashion to changes that occur in it. An agent's environment may be the physical world, the Internet, a collection of other agents, or a combination of them. An agent can exhibit a goal-directed behavior and can select dynamically and autonomously which action to execute, according to its designed objectives.
- *Social Ability.* An agent can interact with other agents (artificial and human) through high-level protocols. It can coordinate and collaborate with other peers for achieving its designed objectives.

Quoting Luck (2), the agent paradigm "causes a re-evaluation of the very nature of computing, computation and computational systems, through concepts such as autonomy, coalitions and ecosystems, which make no sense to earlier paradigms." It can be useful to contrast the definition given above with other widely used abstractions in software engineering, that is, the *object* paradigm and the system *component* abstraction. Worth mentioning is also the difference between software agent and *agent in the world*, as used in requirements engineering approaches.

Agents versus Objects. Objects are defined as computational entities that encapsulate some states, perform actions or *methods* on this state, and communicate by message passing. Agents have goals and skills (services) to achieve them. Agents exploit a symbol-level communication.

Even if objects encapsulate state and behavior realization (method), they do not encapsulate behavior activation (action choice). An object's public method can be invoked by any object, and only once the method has been invoked, the corresponding actions are executed. Objects are passive, they can not refuse to execute them. Objects can be used to enable agent technology. Most currently used agent-programming languages and platforms are built on top of Java (5). More generally, object and methods abstractions are considered too low-level granularity for describing interactions. This motivated the development of more powerful abstraction mechanisms, such as system component.

[4] Herein, see: Software Engineering: Software Life-Cycle Activities and Software Engineering: Software Engineering Tools and Techniques

Agents versus System Components. A software component is a system element that offers a predefined service and communicates with other components. Components are considered to be a higher level of abstraction than objects, and as such they do not share state and communicate by exchanging messages that carry data. A main difference between components and agents is in the mechanism they use to communicate, which is usually referred as imperative versus declarative message passing, and in the purpose of communication. A component communicates with another one to force it to execute the body of a method without saying why. The sender is entirely responsible for such an execution (that is, it is responsible for guaranteeing that preconditions hold and for changes caused in the system). An agent communicates with another one in an attempt to transfer part of its mental state to the receiver. For instance, it can delegate one of its goals to the receiver. It is up to the receiver to accept this delegation. Moreover, the receiver is solely responsible for the outcome of its own actions.

The different communication models determine a major difference in their relationship to the outer world (the environment). Agents execute in the environment (that is the execution of their actions affects the environment); they can perceive it so that changes in the environment reflect into changes of the agent's mental state. Components use interfaces to enumerate what they can do and how clients can get in contact with them. Interfaces can specify postconditions, which define how the state of the component changed on executing an action, but no information on the environment is given (6).

Agents in Requirements Engineering. Requirements engineering provides methods and techniques for supporting the identification of users and stakeholders needs, and for analyzing them in terms of alternative solutions. Its ultimate goal is that of deriving software requirements that fit those needs. In requirements engineering, the agent paradigm is used to analyze the application domain, in which a new software has to be introduced, in terms of agent's intentionality and sociality. Agents here are humans, organizations, and artificial systems; each one has its own goals and mutual dependencies for goal achievement. Introducing a new software system means introducing a new agent in the domain, which will provide alternative ways to achieve the domain agents' goals. Requirements of this software are traced back to domain agents' goals, which provide a rationale for requirements and also a way for detecting reasons for possible conflicts among requirements. As pointed out by Yu (7) despite the fact that "agents-as-software" and "agents-in-the-world" may share conceptual features, important differences must be taken into account when using them. For instance, assigning greater autonomy to software agents means building a more powerful and complex system. When modeling the world instead, the analyst ascribes an increasing level of autonomy to the modeled agent when the implications of a greater level of uncertainty and variability need to be understood. As for sociality, agents in the world engage in complex relationships, which form an unbounded network, so when modeling them the purpose is to provide a means to acknowledge complexity in the world rather then to identify mechanisms to manage it.

Agent-Oriented Modeling Languages

Modeling languages allow one to represent the structure and the properties of a system in an abstract way. The most widely used in software engineering is the Unified Modeling Language (UML) (8), which is the standard language for object-oriented modeling. Besides a specific syntax and semantics, modeling languages usually provide a diagrammatic notation, which allows one to represent a system specification with a set of diagrams.

Visual modeling has been recognized as a powerful support for communication among the stakeholders involved in the development process and for the documentation of a project. It became a popular practice as soon as software tools that provide functions to create models and diagrams started to become available.

Modeling is a core activity in software development processes that follow the model-driven architecture (MDA) approach (9). Indeed, MDA conceives the software development as a modeling process. The basic artifacts in an MDA process are models that are used to specify the software to be built. Two types of models are usually created: (*1*) a model that corresponds to a software specification which is independent from the technology that will be used for its realization, called also platform-independent model; and (*2*) a model specified in a language, which allows it to represent basic construct of the target implementation platform (the platform-specific model). MDA proposes guidelines and standards to automate the mapping from a platform-independent model to a platform-specific model, provided that the syntax and semantics of the modeling languages used to build these models are given in terms of a meta-model. The term meta-model is used to indicate a model of the concepts that can be used to design and describe actual systems. These meta-models are usually specified as UML class diagrams. The models that describe a system contain instances of the meta-model classes. Figure 1(a) sketches the structure of an MDA process (9,10). Mapping between source and target models are obtained through an application of mapping rules defined between the elements of the meta-models of the source and the target modeling languages.

The ultimate goal of MDA is to improve the quality of software products and the development process, by allowing for the reuse of models and mappings between models. Nowadays, lot of effort is required to develop model interoperability standards, as well as model-to-model transformation concepts and techniques for their automation in MDA. The MDA initiative refers mainly to OO software development, but its ideas and standards influenced AO approaches also.

Agent-oriented modeling languages are usually given their own graphical notation and UML meta-models to express their syntax and semantics. In a broad sense, we may consider three main families of AO modeling languages: (*1*) languages that inspired directly from the AO

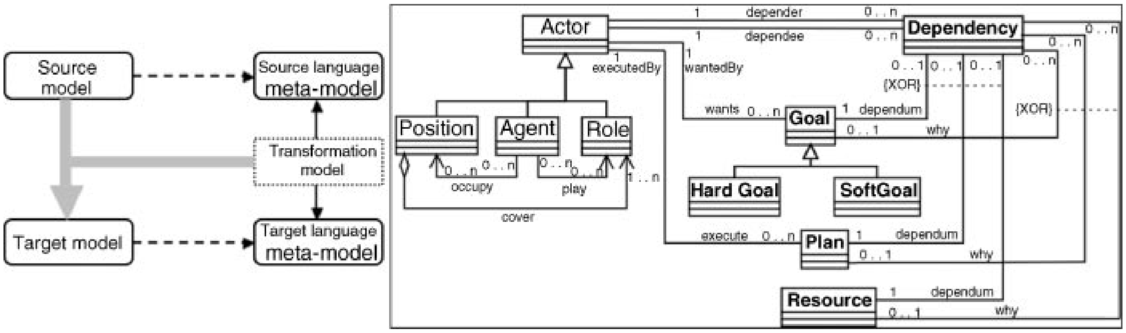


Figure 1. (a) Model to model transformations according to the MDA approach (9). (b) A portion of the meta-model specifying the semantics of the i^* based notation used in the *Tropos* methodology (10).

paradigm. Examples of languages belonging to this family are the agent-object-relationship modeling language (AORML) (11), the i^* framework (12), and Knowledge Acquisition in autOmated Specification (KAOS) (13), which have been initially proposed for modeling and analyzing requirements; (*2*) languages defined by abstracting from specific agent programming languages. Examples include the modeling language used by the *Prometheus* methodology; (*3*) languages that extend UML with agent paradigm notions. Examples are Agent UML (AUML) (14), Agent Modeling Language (AML) (15), and the modeling language used by the *ADELFE* methodology.

Figure 2 illustrates AO modeling languages' genealogy. In the rest of this section, for each family, an example of a modeling language is given in more detail. Some consider the situation at the time of writing analogous to the situation that preceded the agreement, which led to the definition of a unified language for object-orientation, namely UML.

Languages Inspired from the AO Paradigm

The i^* framework (12) proposes an agent-oriented approach to requirements engineering, which focuses on the *intentional* characteristics of agents, such as goals, beliefs, abilities, and commitments. The underlying idea is that agents in organizations depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished, which leads to a network of dependencies. That is, the framework rests on a concept of *distributed* intentionality, from which the name i^* derives.

Primitive concepts of the i^*modeling framework are: the concept of *actor*, which is an entity that has strategic goals and intentionality within the system or the organizational setting. An actor can represent a physical or a software *agent*, a *role* played by an agent in a specific context or a set of roles (*position*). The concept of *goal* represents a strategic interest of an actor. The language distinguishes hard goals from soft goals, which are typically used to model nonfunctional requirements. Task (*plan*) represents, at an abstract level, a way of doing something; *resource* represents a physical or an informational entity. Moreover, the framework supports several primitive relationships, such as strategic *dependencies* between actors where one actor wants something and another is willing and able to deliver it. Additional types of relationships are: goal AND/OR *decompositions, means-end* relationships between a plan (the means) and a goal (the end), as well as positive/negative *contributions* from goals/plans to soft goals.

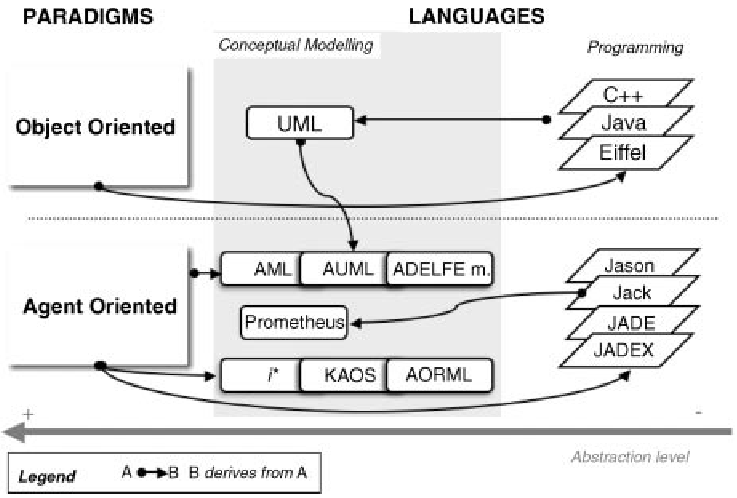


Figure 2. Genealogy of AO modeling languages. The abstraction level increases going from programming languages to paradigms. The UML language for object-oriented analysis and design was defined by abstracting from OO languages. Some AO modeling languages result from an analogous abstraction process from AO programming languages (e.g. Prometheus) whereas others inspire directly from the AO paradigm (e.g. i*, KAOS, AORML), or extend UML with AO concepts (e.g. AHL, AUML, ADELFE).

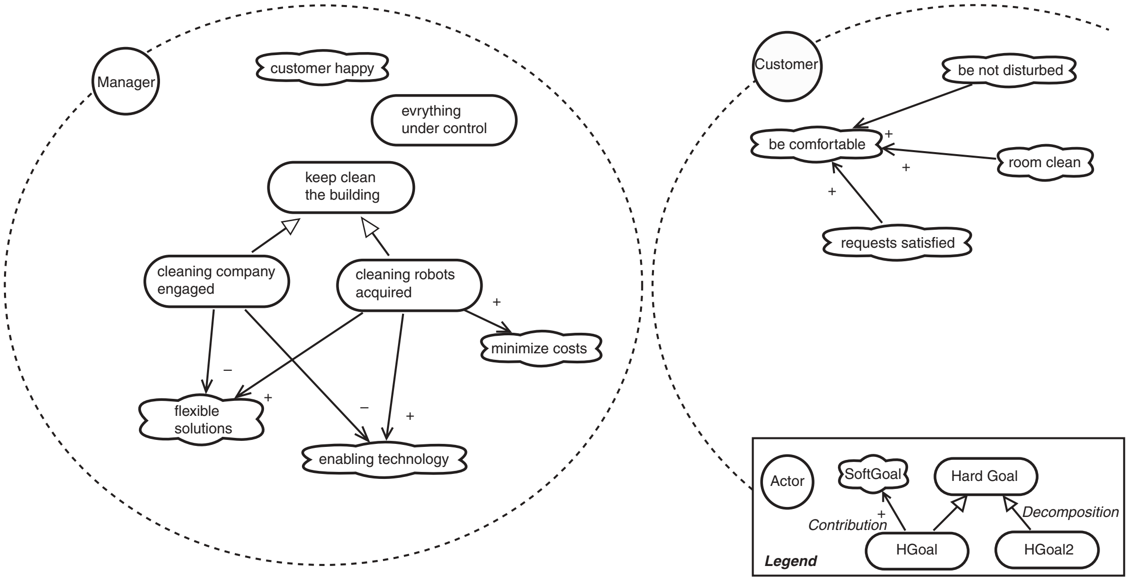


Figure 3. Goal diagrams of the Early Requirements *Tropos* model of the hotel cleaning domain.

These notions are more formally specified in the language meta-model, which is illustrated in Fig. 1(b). For instance, an actor *dependency* is a 4-ary relationship: The first and second arguments are of type actor (depender and dependee), and the third argument is of type goal or plan or resource (dependum). It is also possible to specify a reason for the dependency (labeled as why), which can be a goal or a plan, which is part of a goal/plan decomposition or a resource. A model is an instance of the meta-model and can have a graphical representation in terms of an *actor* diagram that represents strategic dependencies between networks of actors and a *goal* diagram, which depicts how actor goals have been decomposed into subgoals and possibly operationalized through means–end relationships.

An example of *goal* diagram is given in Fig. 3, which illustrates an excerpt from a model of an "hotel cleaning" domain.

> Two stakeholders, the hotel's manager, and the customer are represented as actors, the `Manager` and `Customer` actors respectively, with their goals. The point of view of the actor `Manager` with reference to main goals like `keep clean the building` is analyzed inside the balloon diagram. This top level goal is OR-decomposed into subgoals, which represents alternative ways to achieve it, namely engaging a cleaning company or acquiring a cleaning robot system.

Abstracting from Agent Programming Languages

As soon as agent programming environments were ready to be used outside the academic environment, modeling languages for specifying agent applications started to be proposed. The aim was mainly that of providing a modeling language for a practical methodology that should guide a software engineer during the use of a specific programming environment, which is eventually generalizable to others.

Along this line, we can mention the modeling language used by the *Prometheus* methodology (16). Among the basic concepts of this modeling language are those of *percepts* and *action*, which represent information that the environment provides to an agent and the actions the agent can perform to change the environment, respectively. Moreover, two types of agents can be specified: *proactive* agents, which can pursue goals, and *reactive* agents, which respond to significant occurrences (events). These events may be percepts from the environment, but they may also be messages from another agent or even internal occurrences. In case of proactive agents, the concepts of *belief* and *plan* are used to represent states and library of plans to achieve goals. To model social properties of agents concepts like *commitment, norm*, and *team* are also provided.

A set of diagrams to build views on a model are provided, including goal, scenario, role, system overview, agent overview diagrams. Figure 4, depicts a simple system overview diagram for a cleaning robot agent.

Extending UML

The UML language provides different mechanisms for its extensions, for instance, the stereotype mechanism that extends the UML vocabulary adding new model elements or the profile mechanism for tailoring the language to a specific problem domain. These features of UML were exploited in several AO modeling language proposals, which aimed at capitalizing on the wide use of UML in software engineering and on the availability of a variety of tools supporting UML, to favor the adoption of agent-based technology.

The AUML proponents made explicit their intention to provide minimal extension to UML for supporting the specification of properties peculiar to agents and for reusing UML diagrams as much as possible during AO systems design. The initial work on AUML focused on agent interaction, which is specified through sequence diagrams, that were extended, for instance with AND, OR, XOR operators (14).

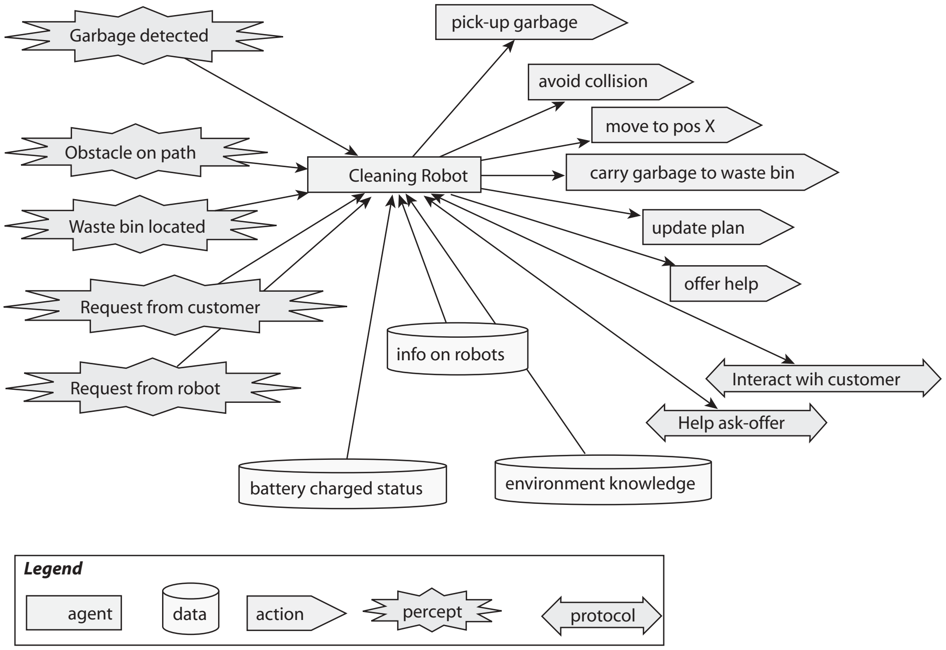


Figure 4. System overview diagram in the *Prometheus* methodology for the cleaning robot agent.

AUML sequence diagrams were adopted by FIPA[5] to express agent interaction protocols and became a reference notation for several methodologies to specify agent protocols. The example in Fig. 5, illustrates an interaction protocol in which an agent (a robot cleaner) requests peers about offering help for cleaning tasks. The protocol admits a set of communicative actions such as refusal to help, acceptance, and inform. Agent class diagrams as well as extensions to other UML elements like package, template, and activity diagrams are also provided. AUML definition is currently a task of the *Agent Platform Special Interest Group* of OMG,[6] whose final objective includes that of promoting a standard agent modeling language.

Based on UML extensions are the modeling language used by the *ADELFE* methodology (17) and the AML (15), which represents an industrial initiative.

AGENT-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

Software engineering methodologies define a structure for the development process in terms of phases, activities, and work products (artifacts).

Research on AOSE methodologies received a lot of attention in the last 10 years, and currently we may count tens of different proposals. Indeed, on one side, researchers try to analyze and compare them along different criteria, like coverage of the agent abstractions used in modeling, model's completeness, consistency, complexity, re-usability; development life-cycle; availability of tools; and in general usability of the methodology. On the other side, a reciprocal contamination and evolution of some previously proposed methodologies can be observed. Currently few of them provide tool-supported environments. Interesting is also an analysis made by Sellers et al. (18) that identified the roots of state-of-the art methodologies in OO frameworks like RUP and Fusion.

We focus here on AOSE methodologies that adopt a model-based approach. These methodologies define the models to be created, step by step, during the different phases in the development process, and adopt an AO modeling language for their specification.

To give a flavor of what AOSE methodologies offer today, in the following, basic features of the principal methodologies will be recalled along basic phases of the software development process, namely requirements analysis, system design, implementation and testing. Excerpts from the analysis and design of the "cleaner world" scenario, adapted to the problem of room cleaning in a hotel, will be used to exemplify.[7] In particular, room cleaning will be considered as a cooperative task performed by a team of autonomous agents, which have on-board sensors to help moving in the environment avoiding collision with objects or moving entities, and the ability to engage forms of collaborations with the other robots to ask/offer help for cleaning.

[5] The Foundation for Intelligent Physical Agent (FIPA) aims at promoting agent-based technology and the interoperability of its standards with other technologies. Since 2005, it is an IEEE Computer Society standards organization.

[6] The Object Management Group (OMG) is an international consortium aiming at promoting standards for a wide range of technologies. The effort on AUML definition has been first carried out inside FIPA, and since 2005 it is one of the OMG initiatives, http://agent.omg.org/.

[7] The "cleaner world" scenario has been largely used in AI research. A description is given by Firby (19).

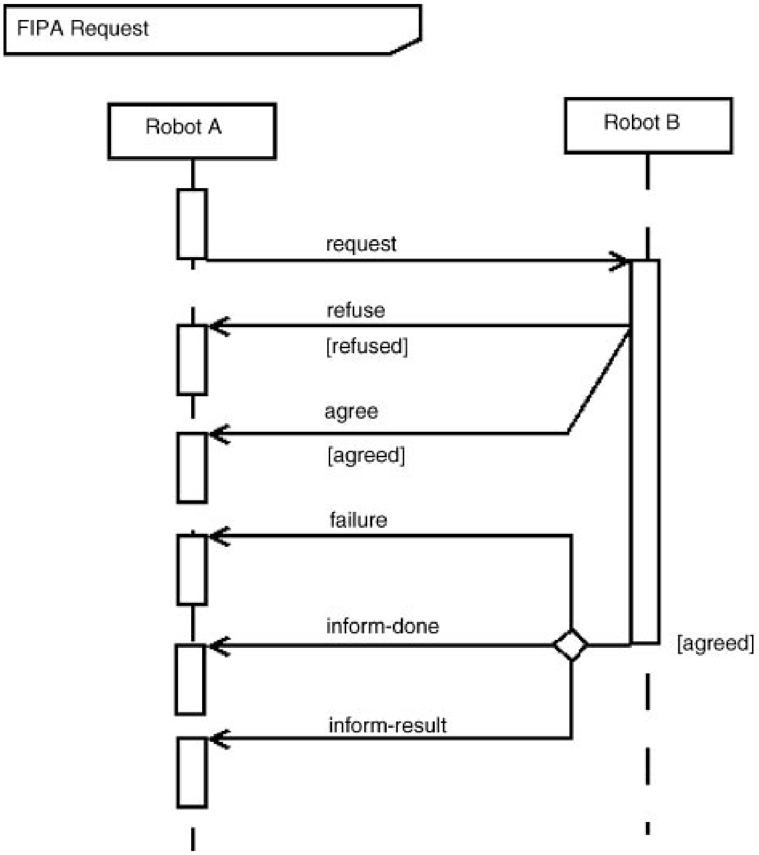


Figure 5. AUML specification of the ask/offer help interaction protocol between cleaning robot agents.

Analysis

The analysis phase aims at understanding the problem domain and the requirements of the software system to be built. These aspects are crucial in developing software because poor understanding of the problem domain and of customer needs is a major reason for software projects failures. This fact motivates the development of a specific discipline inside software engineering, named requirements engineering, which offers methods and techniques to discover and specify requirements, to analyze and verify the consistency of a requirements specification, and to manage requirements changes.

Focusing on AO approaches, the analysis phase will be guided by questions like the following ones. How will the system affect the current domain organization? Which goals will be the system responsible for? How will the system interact with the environment? The resulting models will specify aspects like: the role(s) of the system-to-be with respect to domain stakeholders; its assigned goals (responsibilities); reasoning capabilities, sensors/effectors the system should be provided with; human-system and system-system interaction protocols.

Table 1 recalls specific models to be built in this phase, according to some AOSE methodologies. In particular, the *Tropos*(20) methodology borrows ideas from the i^* framework for requirements engineering and proposes to model the application domain first, using actor and goal diagrams, as those depicted in Fig. 3 and in Fig. 6. In the *Early Requirements* model, the domain stakeholders are represented as *actors* together with their goals and their strategic dependencies for goal achievement. For example, the hotel cleaning scenario is modeled in terms of two main stakeholders: the hotel manager and the customer (Fig. 3).

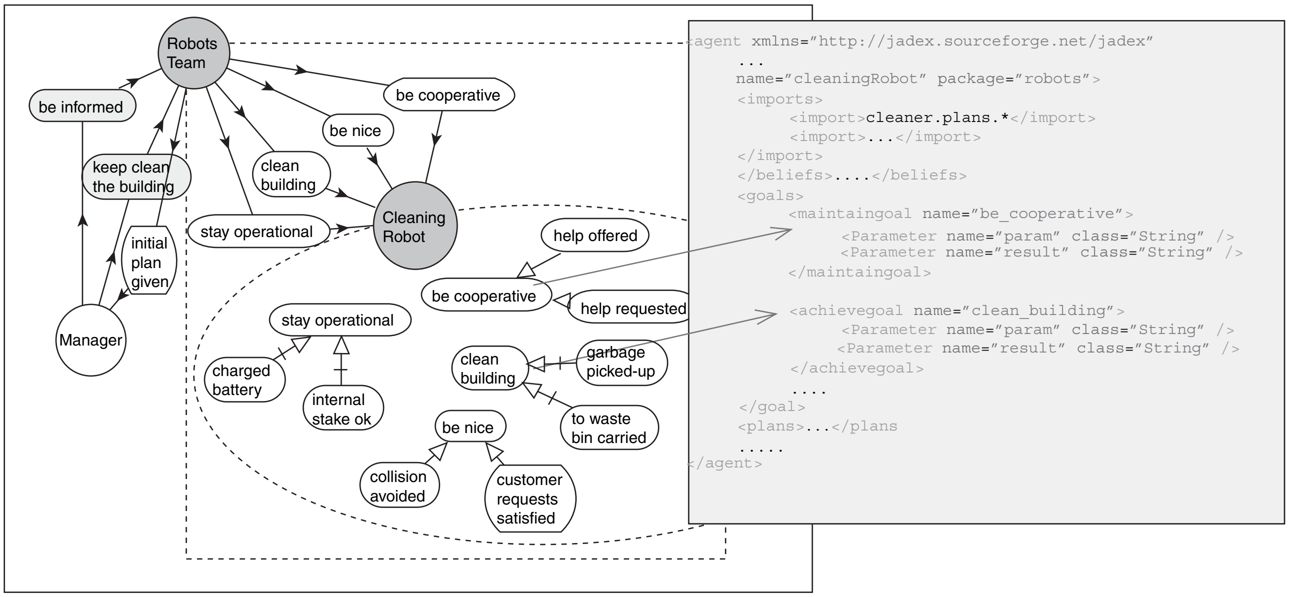
> The manager goal `keep clean the building` is analyzed along alternative ways to achieve it. Exploiting a cleaning robot system is represented by the subgoal `cleaning robots acquired`. This alternative is considered a more flexible solution with respect to engaging a cleaning company and contributes to reduce costs, this is represented in terms of positive contributions to the soft-goals `flexible solutions` and `minimize costs`.

A deep understanding of the domain is considered a crucial step to understand the role of the software system to be built and to identify the goals the new system should contribute to achieve. This is represented in the *Late Requirements* model, an excerpt from the model of the hotel cleaning scenario is depicted in Fig. 6.

> The `RobotTeam` actor represents the cleaning robot system. The overall objective of the system is that of daily cleaning the building (`clean building` goal). A predefined task allocation is given to the cleaning agents, but dynamic re-planning of the cleaning task is used to manage unforeseen situations like

Table 1. Artefacts by Process Steps, Supporting Tools and Development Process of Some AOSE Methodologies (6, 18).

Process Phase	*Tropos*	*MAS-CommonKADS*	*Gaia*	*o-MaSE*	*Prometheus*
	Early Req. goal actor-diagrams in i^*			*Domain model*	*Analysis Overview* model
Analysis (Tools)	*Late Req.* goal- actor-diagrams in i^* (TAOM4E modeler; T-Tool model checker)	*Agent, Organization, Expertise, Task, Communication, Coordination* models	*Role, Interaction* models - no specific notation-	*Goal, Organization, Role* models - OO derived notation; Role Description Document (aT^3 tool)	*Scenarios, System goals, System Interface* (actions, perceptions) models (PDT modeler)
Design (Tools)	*System actor* goal-actor-diagrams in i^*	*Design, Expertise* models	*Organization structure, Role, Interaction* models	*Agent class* and *Protocol* models	*System overview* (Agent-role grouping), *Protocol* (interaction diagrams), *Agent descriptors*
	Interaction, Capability and *Plan* diagrams in AUML (TAOM4E modeler)	*Agent, Organization* and *Reaction* models	*Agent, Service* and *acquaintance* models	*Plan, Capability* models (aT^3 tool)	*Process, Agent overview, Capability descriptors* (capability, event, data, plan descriptors) (PDT modeler)
Implementation (Tools)	MDA mapping to JADE/Jadex Agent skeleton (t2× tool - TAOM4E)				Automatic code generation in JACK
Testing (Tools)	Goal-oriented testing methodology (eCAT tool TAOM4E)				Interaction protocol debugger (petri nets)
Development Process	Iterative and incremental	Cyclic risk-driven process	Iterative within phase, sequential between phases	Iterative across all phases	Iterative across all phases



```
<agent xmlns="http://jadex.sourceforge.net/jadex"
     ...
     name="cleaningRobot" package="robots">
     <imports>
          <import>cleaner.plans.*</import>
          <import>...</import>
     </import>
     </beliefs>....</beliefs>
     <goals>
          <maintaingoal name="be_cooperative">
               <Parameter name="param" class="String" />
               <Parameter name="result" class="String" />
          </maintaingoal>

          <achievegoal name="clean_building">
               <Parameter name="param" class="String" />
               <Parameter name="result" class="String" />
          </achievegoal>
          ....
     </goal>
     <plans>...</plans
     .....
</agent>
```

Figure 6. Hotel cleaning scenario: excerpts of Late Requirements and Architectural diagrams in *Tropos*. A fragment of the Jadex code that has been automatically derived from the Architectural diagram is also shown.

> extra effort required to clean a room or robot malfunctions. Re-planning results from agents collaboration (be `cooperative` goal). Of course re-planning can still end up with a failure condition that will be communicated to the hotel manager.

Another methodology, *MAS-CommonKADS* (21) adapts conceptualization techniques, such as *Class-Responsibility-Collaboration* (CRC) cards together with user centred techniques and prescribes a set of models to be built in the analysis phase, including an organization model that describes the agent society, and a communication model that describes the human-software agent interactions.

In other AOSE methodologies the analysis phase focuses directly on the system-to-be and subsequently on the external actors, which interact with it. For instance, *Gaia* (22) assumes that requirements have been identified during a preliminary requirements elicitation activity and prescribes to build role and interaction models for the system. Note that *Gaia* does not commit to any specific notation for its models. Analogously in *o-MaSE* (23) first a goal model of the system is created and refined into an AND/OR goal tree, from which the organizational model is derived. The organizational model points out interfaces with external actors and it will be complemented with a specification of the interaction between those actors and specific roles in the system (Role model).

Other AOSE methodologies, like *Prometheus,* adapt Use Case analysis techniques used also in OO methodologies, which identify the external entities (referred to as actors) that will use or interact in some way with the system-to-be and the key scenarios around which the interaction will occur. The system interface is also modeled in terms of perceptions and actions the system can perform on the environment.

Design

The design phase aims at defining the system architecture accordingly to the functional and nonfunctional requirements analyzed in the previous phase. AOSE approaches design at two levels called the macro and the micro levels. The macro level concerns the specification of the system architecture in terms of components (e.g., agent types or multiagent systems) and of their dependencies. Here, available architectural styles (organizational patterns) can be used as a reference structure for the architecture. An example of system architecture model in *Tropos* is depicted in Fig. 6.

> The `Robots Team system` is defined in terms of `Cleaning Robot` agents with the following four individual goals: clean its environment by removing dirt whenever possible, that is pick-up any garbage and carry it to a near waste bin; keep itself operational by monitoring its internal state and in particular its battery charge state. Whenever the battery state is low, move to the charging station; be nice to other-people or objects that are close by, e.g. it should not collide with others. Moreover it should understand requests by an hotel customer like: "do not clean here now" or "do a quick cleaning now"; be collaborative with the other robot, that is answer to a request for help giving information about the percentage of assigned work done, the battery charge state and the location.

The corresponding agent model according to the *Prometheus* methodology is illustrated in Fig. 4.

The micro level concerns the design of the single agents. It specifies the agent's knowledge and reasoning capabilities, and the interaction protocols along with it will interact with the other agents. Interaction protocols are usually specified with diagrams as the one depicted in Fig. 5. Table 1 summarizes design model sets of current AOSE methodologies and supporting tools.

Implementation and Testing

Most currently available AOSE methodologies provide the developers a well-defined set of agent classes to implement and instantiate, according the specification that results from the design process. Guidelines or support for the implementation phase is usually considered out of the scope of the methodology.

The application of MDA principles to software development provides techniques to build automatic mappings from design specifications to code skeleton, in a chosen implementation platform. Research work on the application of these techniques to the case of AO systems is giving promising results (24).

Figure 6, depicts a fragment of agent code, in Jadex, which has been derived automatically from the agent's goal diagram of the *Architectural design* model in *Tropos*.

Agent programming environments, such as JADE and JACK, build on Java programming environments that provide their own supporting tools for programming and debugging (5). Current research addresses the need of debugging tools that can integrate the different programming paradigms and monitor the execution and communication of agents.

AO systems verification and testing is also mainly addressed in the research context.

Development Process and Tools

The development process of the AOSE methodologies involves a high degree of iteration within and or across the development phases. Process's features, from an overview on ten methodologies (18), are partially reported in Table 1.

Providing CASE tools at support of AOSE methodologies has been recognized a crucial step toward the adoption of *Agent-Oriented Software Engineering* methodologies by industry. Currently, specialized tools, which exploit model-checking or deductive reasoning techniques, are provided to support specific tasks such as the consistency verification of requirements (or design) specifications.

Visual modelers are provided by *Tropos* and *Prometheus*, whereas methodologies that adopt UML-based notation provide customization of UML modelers. The above-mentioned modelers provide also functions for automatic code generation toward specific Multi-Agent System platforms.

RESEARCH DIRECTIONS

Several challenges stimulate research in AOSE, as pointed out by Luck (2) and Zambonelli (25). Worth mentioning are efforts toward unification of AOSE methodologies and definitions of standards, as well as research on designing software with autonomic properties.

Given the great number of AOSE methodologies that can be found in literature, and the fact that each one defines its own concepts and system structure, interoperability becomes a main challenge. The Agentlink AOSE Technical Forum Group[8] addressed this issue by studying and comparing meta-models of existing methodologies to find commonalities and to give clear definitions of the core concepts used in AO systems development. The final objective was to propose a unified AO meta-model to become a reference for future agent-oriented modeling languages and development tools. This work is summarized by Bernon (26). The definition of standards for AOSE, including AO meta-models, is currently being investigated by other parallel initiative, such as IEEE/FIPA, OMG, and ISO.

Meta-models of the AO development process and of its products are at the core of *method engineering* approaches, which propose a project-specific methodology by composing coherent parts of existing methodologies, called method fragments. Method engineering has been successful applied in OO development (27). It requires a repository of method fragments specified accordingly to process and product meta-models. Construction guidelines support the use of the repository for deriving a personalized development methodology, which fits the needs of specific projects. Currently, a wide repository of method fragments coming from well-known AOSE methodologies, such as *Gaia*, and *Prometheus, Tropos*, has been included in the Open Process Framework (28). A similar approach is pursued by the FIPA Methodology Technical Committee. The method engineering approach allows, on one side, to overcome the need of a universally applicable methodology, and on the other side, it forces methods interoperability by means of meta-models.

The need of more solid tools to support agent systems developers is motivating studies on automating the generation of code from AO specifications and the verification of specifications and system testing. Adapting verification and testing techniques from concurrent and distributed systems may be not sufficient for the case of AO systems. Autonomous and deliberative behaviors of AO systems that operate in an open world may lead to unpredictable scenarios, and it may become difficult to say whether the AO systems are behaving "as wished." These problems become harder and harder as soon as the number of agents increases, the emergent behaviors are likely to be manifested. Studies on automating test cases generation and execution are promising toward managing scalability and evolvability issues. More generally, providing adequate development techniques and methods to support dependability characteristics, such as fault avoidance, fault tolerance, fault removal and fault forecasting, will make AO systems more robust, trustworthy, secure, and safe.

Research on the integration with other recently proposed computational paradigms, such as service-oriented architecture and autonomic computing, is also under way. According to the service-oriented architecture (29) model for distributed applications, new applications can be created at run-time by dynamically aggregating existing components, each one providing predefined computational services.

Autonomic computing (30) aims at creating computational systems, which can manage themselves. More specifically, these systems should automatically configure and reconfigure under varying (and unpredictable) conditions; they should self-optimize their operations by monitoring their components and fine tuning their workflow to achieve system goals; and they should automatically discover problems and recover from situations that might cause malfunctions. These properties are called in brief self-* properties. The autonomic computing vision takes inspiration from the autonomic function of the human central nervous system, which controls key functions without conscious awareness or involvement. This vision shares many goals with AO computing. AOSE seems to offer candidate solutions both at the design and the technology levels to manage the complexity of autonomic systems.

CROSS-REFERENCES

Intelligent Agent
Object-Oriented Analysis and Design

BIBLIOGRAPHY

1. N. R. Jennings, An agent-based approach for building complex software systems, *Commun. ACM*, **44** (4): 35–41, 2001.
2. M. Luck, P. McBurney, O. Shehory, and S. Willmott, *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*.Liverpool, UK: AgentLink, 2005.
3. J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1999.
4. M. Wooldridge, *An Introduction to MultiAgent Systems*. New York: John Wiley & Sons Ltd, 2002.
5. R. H. Bordini, L. Braubach, M. Dastani, A. E. F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci, A survey of programming languages and platforms for multi-agent systems. *Informatica*, **30** (1): 33–44, 2006.
6. F. Bergenti, M.-P. Gleizes, and F. Zambonelli, eds., *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. New York: Springer, 2004.
7. E. Yu, Agent orientation as a modelling paradigm, *Wirtschaftsinformatik*, **43** (2): 123–132, 2001.
8. G. Booch, J. Rambaugh, and J. Jacobson, *The Unified Modeling Language User Guide*. The Addison-Wesley Object Technology Series. Reading, MA: Addison-Wesley, 1999.
9. S. J. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled*. Boston, MA: Addison-Wesley, 2004.
10. A. Susi, A. Perini, P. Giorgini, and J. Mylopoulos, The Tropos metamodel and its use, *Informatica*, **29**: 401–408, 2005.

[8] http://www.agentling.org.

11. G. Wagner and K. Taveter, Towards radical agent-oriented software engineering processes based on AOR modeling. In Henderson-Sellers and Giorgini [18], 2005, pp. 277–315.
12. E. Yu, *Modelling Strategic Relationships for Process Reengineering*. PhD Thesis, Toronto, Canada: University of Toronto, Department of Computer Science, 1995.
13. A. van Lamsweerde and E. Letier, Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Software Eng.* **26** (10): 978–1005, 2000.
14. M.-P. Huget, J. Odell, and B. Bauer, The AUML approach. In Bergenti et al. [6], 2004, pp. 231–257.
15. I. Trencansky and R. Cervenka, Agent modeling language (AML): a comprehensive approach to modeling MAS. *Informatica*, **29** (4): 391–400, 2005.
16. L. Padgham and K. Taveter, *Prometheus: A Practical Agent-Oriented Methodology*. In Henderson-Sellers and Giorgini [18], 2005, pp. 107–135.
17. C. Bemon, V. Camps, M. -P. Gleizes, and G. Picard. Engineering Adaptive Multi-Agent Systems: the ADELFE Methodology. In Henderson-Sellers and Giorgini [18], 2005, pp. 172–202.
18. B. Henderson-Sellers and P. Giorgini, eds., *Agent-Oriented Methodologies.*Hershey, PA: Idea Group Inc., 2005.
19. J. Firby, An architecture for a synthetic vacuum cleaner. In *Proc. of the AAAI Fall Symp. Series Workshop on Instantiating Real-World Agents.* Raleigh, NC, 1993.
20. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, Tropos: an agent-oriented software development methodology. *Auton. Agents Multi-Agent Sys.*, **8** (3): 203–236, 2004.
21. C. A. Iglesias and M. Garijo. The Agent-Oriented Methodology MAS-CommonKADS. In Henderson-Sellers and Giorgini [18], 2005, pp. 46–78.
22. F. Zambonelli, N. R. Jennings, and M. Wooldridge, Developing multiagent systems: the gaia methodology, *ACM Trans. Softw. Eng. Methodol.* **12** (3): 317–370, 2003.
23. S. DeLoach. Engineering organization-based multiagent systems. In Software Engineering for Multi Agent Systems IV (SELMAS), Springer, LNCS 3914: 109–125, 2006.
24. L. Penscrini, A. Perini, A. Susi, and J. Mylopoulos, High variability design for software agents: extending tropos, *ACM Trans. Auton. Adapt. Syst.*, **2** (4): 2007.
25. F. Zambonelli and A. Omicini, Challenges and research directions in agent-oriented software engineering, *Auton. Agents Multi-Agent Syst.*, **9** (3): 253–283, 2004.
26. C. Bernon, M. Cossentino, and J. Pavón, Agent-oriented software engineering. *Knowl. Eng. Rev.*, **20** (2): 99–116, 2005.
27. S. Brinkkemper, Method engineering: engineering of information systems development methods and tools. *Informat. Soft. Technol.*, **38** (4): 275–280, 1996.
28. D. G. Firesmith and B. Henderson-Sellers, *The OPEN Process Framework: An Introduction*. Boston, MA: Addison-Wesley, 2002.
29. M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. KrÞmer, Service-oriented computing: a research roadmap. In *Service Oriented Computing*, volume 05462 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum f?r Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.
30. J. O. Kephart and D. M. Chess, The vision of autonomic computing, *IEEE Comput.* **36** (1): 41–50, 2003.

Reading List

F. Bergenti, M.-P. Gleizes, and F. Zambonelli, eds., *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook.* New York: Springer, 2004.

B. Henderson-Sellers and P. Giorgini, eds., *Agent-Oriented Methodologies.* Hershey, PA: Idea Group Inc., 29005.

ANNA PERINI
Fondazione Bruno Kessler—IRST
Trento, Italy

AGENT TECHNOLOGY

INTRODUCTION

Agent technology is a rapidly growing subdiscipline of computer science on the border of artificial intelligence and mainstream software engineering studying the construction of intelligent systems. It is centered around the concept of an (intelligent/rational/autonomous) *agent*. Generally, An agent is a software entity that displays some degree of autonomy; it performs actions in its environment on behalf of its user but in a relatively independent way. It takes initiatives to perform actions on its own by "deliberating" its options to achieve its goal(s).

Although there is no generally accepted precise definition of an agent, some consensus surrounds the (possible) properties of an agent (1,2): Agents are hardware or software-based computer systems that enjoy the properties of:

- Autonomy. The agent operates without the direct intervention of humans or other agents, and it has some control over its own actions and internal state.
- Situatedness. Agents are situated in an environment. They sense it and act in it.
- Reactivity. Agents perceive their environment and react to it in a timely fashion.
- Pro-activity. Agents take initiatives to perform actions and may set and pursue their own goals.
- Social ability. Agents interact with other agents (and humans) through communication; they may coordinate and cooperate while performing tasks.

Other properties that agents may have are *mobility* (the ability of an agent to move around in an electronic network and the Web in particular), *veracity* (the assumption that an agent will not knowingly communicate false information), *benevolence* (the assumption that an agent will always try to do what is asked of it), and *rationality* (the assumption that an agent will act to achieve its goals, and it will not act in such a way as to prevent its goals being achieved, its beliefs permitting), cf. Ref. 1.

Thus we observe that agents have both informational and motivational attitudes; they handle and act on certain types of information (such as knowledge, or rather beliefs) as well as motivations (such as goals). Many researchers adhere to a stronger notion of agency that is sometimes referred to as a "cognitive" agent, which is an agent that realizes the above properties by means of mentalistic attitudes that pertain some notion of a mental state. The mental state involves such notions as knowledge, beliefs, desires, intentions, goals, plans, commitments, and so on. The idea behind this notion is that through these mentalistic attitudes, the agent can achieve autonomous, situated, reactive, proactive, and social behavior in a way that is mimicking or at least inspired by the human way of thinking and acting. So, in a way we may regard agent technology as a modern incarnation of the old ideal of creating intelligent artifacts in artificial intelligence (AI). Indeed, some modern textbooks in AI (3,4) even identify AI with the study of agents!

However, we may also regard agent technology as a next step after the currently very popular *object* techno- logy in the form of object-oriented programming. Some researchers regard agents as a special kind of objects, and some use the term "active objects" for agents (cf. Ref. 2, p. 26,27). However, in principle, some fundamental differences exist between agents and objects. In object-oriented programming, objects are simply "used": For instance, a call to a method of an object, will provided everything works normally just be executed by that object. In some sense, objects are just passive entities that can be employed in a system. For agents, however, it is different. An agent cannot be controlled directly by some program outside that agent (such as another agent). The agent should be *requested* to perform a task or to provide a piece of information. This request may be denied for reasons that concern that agent: Perhaps it is too busy, the request is not in line with its own goals, or perhaps even the agent is not in the right "mood". (We will later see how far these "human-like" qualities of agents may go.) In short, one may say that the main difference between agents and objects is the lack of autonomy of the latter (no control over its internal state and actions), whereas for the concept of an agent, it is one of the defining properties.

It is also important to note that the property of situatedness that agents are supposed to possess puts them apart from the expert systems from the 1980s. Expert systems are not situated: They only get information (symptoms) as input (from a user) and yield information (a diagnosis) as output (to that user). They do not really sense an environment, and they do not act directly in such an environment.

Although the initial research on agents focused on models of individual agents, more recent developments in agent technology emphasize particularly models and architectures of multiagent systems, in which multiple agents share the same environment and interact with it and each other. Here, we see the emergence of an interesting amalgam of the areas of (distributed) artificial intelligence and distributed computing in mainstream computer science. As a consequence, one may discern a trend toward *distribution* (of resources, "intelligence", and reasoning) and *delegation* of tasks between agents. Below, we will sketch the main research issues. But first, we turn to the start of agent technology: Philosophical ideas about (human and artificial) agents that act autonomously in an environment. These agents act based on decisions involving their "mental states," in particular that pertain to knowledge and objectives.

PHILOSOPHICAL BACKGROUND

The field of agent technology emerged out of philosophical considerations on how to reason about courses of action and human action, in particular. An area of analytical

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

philosophy is occupied with what is called *practical reasoning*, in which one studies so-called practical syllogisms that constitute patterns of inference regarding actions. As an example, a practical syllogism may have the following form (5).

Would that I exercise.
Jogging is exercise.
Therefore, I shall go jogging.

Although this example has the form of a deductive syllogism in the familiar Aristotelian tradition of "theoretical reasoning", on closer inspection it seems that this syllogism does not express a purely logical deduction: The conclusion simply does not follow logically from the premises. It constitutes a representation of a *decision* of the agent (going to jog), where this decision is based on mental attitudes of the agent, which are his/her beliefs ("jogging is exercise") and his/her desires or goals ("would that I exercise"). So, practical reasoning is "reasoning directed toward action—the process of figuring out what to do", as Wooldridge (6) puts it. The process of reasoning about what to do next on the basis of mental states such as beliefs and desires is called *deliberation*. The philosopher Michael Bratman has argued that humans (and more generally, resource-bounded agents) also use the notion of an *intention* when deliberating their next action (7). An intention is a desire that the agent is committed to and will try to fulfill until it believes it has achieved it or has some other rational reason to abandon it. Thus, we could say that agents, given their beliefs and desires, choose some desire as their intention, and "go for it". As such, we can view Bratman's theory as an extension of the more general theory of *intentional stance* of Daniel Dennett (8). This stance views complex entities (including humans, animals, and computers) as if they were rational agents that deliberate their beliefs and desires for deciding their next actions. As an aside, we remark that Dennett's intentional stance is very much related to similar theories in psychology and biology, and as ethology, in particular, where humans, including children and animals, are supposedly endowed with a *theory of mind* about other such entities. Thus, humans can reason about the mental states of these entities (Ref. 9, p. 838–841). Bratman's philosophical theory has been formalized through several studies, in particular the work of Cohen and Levesque (10) Rao and Georgeff (11), and van der Hoek et al. (12), and it has led to the *Belief-Desire-Intention model (BDI)* of intelligent or rational agents 11. Since the beginning of the 1990s, researchers have turned to the problem of realizing artificial agents. We will discuss this development in some more detail below.

AGENT LOGICS

As mentioned above, to get a more precise grasp on the philosophical considerations of Bratman, AI researchers started to use formal tools in the form of especially devised logics for formalizing the main ideas. These approaches are based on *modal logics*, which are logics that concern modalities such as knowledge, belief, time, obligation, and action. Modal logics have been proposed in philosophy to analyze the properties of these modalities. Semantics of these logics are usually provided by "possible world semantics" or "Kripke-semantics" (13), in which modal operators are interpreted by means of an accessibility (or possibility) relation, relating possible worlds according to the modality at hand. So, for instance, for temporal logic, this accessibility relation expresses the flow of time, whereas for epistemic logic (the logic of knowledge) the accessibility relation points at possible alternative worlds that the agent deems possible on the basis of its (lack of) knowledge. For example, a knowledge operator then expresses truth in all epistemically alternative worlds designated by the accessibility relation (cf. Refs. 14 and 15).

Cohen and Levesque (10) used a (linear-time) temporal logic for the description of agent behavior. In their work, which has been very influential in subsequent agent research, they try to capture the notion of intention in terms of more primitive notions such as belief, goal, and action. So, the culmination of their paper is a definition of intention in terms of these notions. Actually they give two, because two natural notions of intention are suggested: intention to *do*/ *perform* an action and intention to *be* in a particular state or situation. So for instance, an example of the former is the intention to go on a journey, whereas an example of the latter is the intention to be in a particular city, for example, Paris. Of course, relations exist between these two kinds of intentions, but they are distinct concepts in principle.

Next, Rao and Georgeff came up with their famous BDI logic, which is sometimes also called BDI-CTL because it is based on the (branching–time) temporal logic CTL (computation tree logic). This logic is devised in mainstream computer science to reason about concurrent processes (16). Rao and Georgeff's approach to formalizing Bratman's philosophy is different from that of Cohen and Levesque. Apart from the fact that they use a branching–time logic (catering for possible choices of actions by the agent in a more direct fashion), they also do not build a definition of intention in terms of other notions. Rather they introduce in their logic three independent operators (*modalities*) for beliefs, desires (or goals), and intentions; then, they start to analyze possible relations between these three operators, which are captured by axioms. For instance, they propose to have an axiom $Intend(p) \rightarrow Desire(p)$, which indicates that an intention is a (special) kind of desire.

Finally, we mention that van der Hoek et al. (12) propose yet another approach to formalizing Bratman's theory of intentions. This approach, which is called KARO for Knowledge, Abilities, Results and Opportunities, comprising the core of the logic, is based not on temporal logic but on *dynamic logic*. Dynamic logic is a logic proposed in mainstream computer science to reason about programs (17). Here, it is used to reason about the actions of agents. Furthermore, several BDI-style operators are added such as knowledge, belief, desire, goal, and commitment to specify the behavior of agents.

AGENT ARCHITECTURES

Next, we turn to the issue of constructing agent-based systems. Since the philosophical and logical work on intelligent agents mentioned in the introduction was published, researchers have embarked on the enterprise of realizing agent-based systems. Actually, the first architecture for artificial agents was given by the philosopher Bratman himself together with two colleagues from artificial intelligence, David Israel and Martha Pollack. These authors devised their IRMA architecture (18). At around the same time, the influential BDI architecture (11) and its derivative Procedural Reasoning System (PRS) (19) were proposed, which in turn inspired the dMARS architecture (20). In brief, in the BDI architecture, the interpreter executes a *sense-reason-act* (or *deliberation*) cycle, repeating the following processes: getting (sense) input, leading to possible belief updates, in turn leading to generating new desires/goals, which are then filtered to intentions. These intentions cause the execution of actions usually by means of plans from a precompiled plan library. The BDI architecture and its derivatives are called *deliberative* agent architectures.

However, also other agent architectures exist. *Reactive* agent architectures perform no deliberation or any kind of reasoning; they simply react to an environment. A typical example of such a reactive agent architecture is the *subsumption architecture* proposed by Brooks, (21). In essence, this architecture consists of layered modules, that run in parallel when given input from the environment. However, the output of these layers may inhibit that of other layers, according to a hierarchy (called a subsumption hierarchy). In this way, a priority relation is given amongst several behaviors generated by the modules. For example, in robotic applications, collision avoidance takes priority over exploring and wandering around.

One may also combine reactive behavior and deliberation into one system, which generates *hybrid* agent architectures. Typically, these agents are also layered architectures in which the layers deal with reactive behaviors and deliberative behavior(s). *Horizontally* layered architectures resemble the subsumption architecture, where layers run in parallel after which possible conflicts in outcomes should be resolved in some way by means of a supervisory control framework. However, *vertically* layered architectures also exist, in which the input from the environment enters the top (or bottom) layer, and then input/output go through all layers successively, with final output from the bottom (or top, respectively) layer. The advantage of vertically layered architectures is that "conflicts" between layers are solved "on the fly", so to speak, whereas horizontally layered architectures typically will react faster (because of the layers running in parallel). An example of a horizontally layered architecture is the TouringMachines architecture, which was proposed by Ferguson (22). An example of a vertically layered architecture is Mller's InteRRaP (23). For more information about agent architectures, the reader is referred to Wooldridge (2).

AGENT-ORIENTED PROGRAMMING AND SOFTWARE ENGINEERING

Agent architectures are suitable for conveying the general ideas about the building blocks of agents. But of course, they are hard to use in building actual agents from scratch in some general-purpose programming language. To get a more systematic grip on the construction of agents, new developments came into existence. Some researchers took it onto themselves to provide methods and techniques (sometimes erroneously called a "methodology") for constructing agents in a principled way, similar to what has been done in the "traditional" ways of programming such as object-oriented programming. Typically, one discerns phases of the development of an agent system such as the (requirements) analysis phase, the design phase, and the actual implementation phase. This research area is generally referred to as *agent-oriented software engineering* (24,25). Several methods have been proposed in the literature so far. We mention here DESIRE (26), GAIA (27), AUML (28), TROPOS (29), OPERA (30). Most of these methods ultimately are meant to implement agent systems using generic, general-purpose programming languages such as JAVA or C++. However, some researchers feel that using agent concepts (such as those in the BDI model) in the analysis phase and design phase, but *not* in the implementation phase (since general-purpose languages do not contain agent notions), hampers a smooth and correct construction of these systems (e.g., Ref. 31).

These researchers devised dedicated *agent-oriented programming* languages to program agents directly in terms of mentalistic notions in the same spirit as the ones mentioned above. Thus, these languages typically contain beliefs, goals, commitments, and/or plans as built-in programming concepts (and nowadays mostly with a precise and formal semantics). The first researcher who proposed this approach was Yoav Shoham with the language AGENT0 (32). This language enables the programmer to program timed commitments of agents, based on their beliefs and messages (such as requests to perform actions) from other agents. Other languages include AgentSpeak(L) / Jason, (Concurrent) METATEM, CONGOLOG, JACK, JADEX, and 3APL/2APL (33–36). Although all these languages have in common that they employ several mentalistic notions as mentioned above, they still differ widely from each other. For instance, AgentSpeak(L) (and its JAVA-based interpreter JASON) can be viewed as a programming language based on (a simplified version of) the PRS architecture. It can deal with beliefs and goals, and it can generate plans on the basis of those by means of a plan library via an intricate mechanism using triggering events. METATEM is based on the idea of making logical specifications executable, which restricts, of course, the formulas one can use in the specification. These formulas are a certain subset of temporal logic mixed with other modalities such as knowledge. In a way, one can view this approach as "temporal logic programming". The (CON)GOLOG family of languages is based on the situation calculus, which is popular specification formalism to reason about action and change, proposed by AI pioneer John McCarthy (37). In essence, a CONGOLOG program, which is basically an imperative-style program, provides directions while doing planning to reach a goal ("sketchy planning"). JACK is an extension of JAVA with agent-oriented constructs, whereas JADEX is a BDI reasoning engine implemented in JAVA that provides an execution environment and an application platform interface. 3APL and its successor 2APL are rule-based languages proposed originally as simplifications of AgentSpeak(L), with rules for plan generation (given goals and beliefs) and plan revision (given beliefs, and—in the case of 2APL—to be applied only when current plans fail).

MULTIAGENT SYSTEMS AND AGENT SOCIETIES

Agent-based systems become truly interesting and useful if we have multiple agents at our disposal that share the same environment. Here, we have to deal with several of more or less autonomous agents interacting with each other. Such systems are called *multiagent systems* (MASs) (2) or sometimes also *agent societies*. The field of MASs as such can be viewed as generated from Distributed AI, but many computer scientists interested in distributed/parallel computing and concurrent programming, more in general, have been drawn to it. The advantages of a distributed way of solving problems and performing complex tasks are the following:

- Computations can be done in parallel (so, in principle, they can be done faster).
- Computations and reasoning can be done locally by the agents with limited information about the environment.
- The distributed nature of a MAS may improve robustness and reliability, because other agents may take over from failing agents.

However, a price must be paid: Performance may be lower (suboptimal) whereas centralized single-agent systems may perform optimally in the ideal case that all information and resources are available. On the other hand, these centralized systems typically are more brittle with respect to robustness and reliability, and they may take a very long time to come up with the optimal solution. Another possible price to be paid concerns the communication overhead in a MAS to let the agents cooperate and coordinate properly. (cf. Ref. 38).

AGENT INTERACTION: COORDINATION AND COMPETITION

When considering multiagent systems, especially when we want to design such systems, one needs to think about how the agents will mutually interact. Several kinds of interaction are possible. Generally, it is dependent on the *role* the agents play in the system. In fact, in most current agent-oriented software engineering methodologies, a multiagent system is designed by first considering the *organizational* or *social structure* of the system, in which roles of agents literally play a pivotal role. These roles determine all kinds of (social) properties of the agents that play a particular role, such as objectives, rights, and norms (30). Because of dependencies and power relations between roles, roles also ultimately determine how (role-playing) agents will/should interact and which *coordination type* they follow. In Ref. 39, three of these are distinguished: market, network, and hierarchy. In *markets*, agents are self-interested and competing with each other; in *networks*, there is mutual interest and cooperation, where *trust* in other agents is a crucial factor. In a *hierarchy*, there are dependency and power relations, and *delegation* of objectives and tasks takes place. Note that even in the case of cooperation, it is not a priori obvious how autonomous agents will react to requests from other agents: Because they ultimately have their own goals, it may be the case that they do not have the time to comply, or simply do not want to because they have incompatible objectives of their own.

Ultimately, when designing a MAS it should be specified how role-enacting agents should communicate with each other. This depends on the aims and characteristics of the application at hand, the way roles are related to each other, and how role objectives and norms are "passed" between related roles (30). To this end, communication protocols are employed, which specify who is to communicate with whom on what subject in what fashion. Communication protocols consist of communication actions that are taken mostly from standardized agent communication languages (see the next section). These protocols are application-dependent. For instance, agents in an auction scenario will use typical protocols for bidding. Several well-known protocols are defined in the auction theory literature, such as English, Dutch, and Vickrey auctions (40). Note that these protocols are not always fixed a priori: Depending on the setting (coordination type), these protocols may be subject to *negotiation* between agents ("*contract negotiation*" in Ref. 30).

In general, one can say that the designer of a multiagent system employs *mechanism design*, that is, the design of protocols that govern multiagent interactions such that desirable properties hold, such as guaranteed success (eventually agreement will be reached), certain efficiency and optimality criteria (such as *Pareto efficiency*), stability (agents that have an incentive to behave in a particular way, such as the *Nash equilibrium*, in which agents do not have any incentive to deviate from their behavior), and equity (are all agents treated fairly?) (cf. Ref. 40). Many of these criteria stem from the classic discipline of *game theory*, initiated by von Neumann and Morgenstern (42), which provides a rigorous, mathematical analysis of games.

In cooperative settings, one can consider coordination through *joint intentions*. As we have recognized the importance of the notion of intention (and BDI more generally) in the case of a single agent, we might also consider whether intentions (and other BDI notions) may be generalized to group notions. This method has resulted in notions such as *common knowledge*, *common/mutual belief*, and *joint intentions (*42). Work along these lines has yielded coordination models based on *teamwork*, in which teams are formed on the basis of joint intentions (goals) and recognized potential for cooperative action (43–45).

In more open forms of agent societies, such as networks and especially markets, room for *negotiation* between agents exists. Above, we have observed already that agents may negotiate their interaction contracts, including the protocols that will be used to interact with other agents in the system. However, other things may be negotiated as well, such as tasks and goods, depending on the application at hand (46). The area of agent negotiation has become a field of its own. A related area that is also getting more attention is that of agent *argumentation*. Argumentation is a classic field in philosophy and logic (47). It deals with trying to convince other agents of the truth or falsity of some state of affairs by putting forward reasons (arguments) for and against propositions, together with justifications for the acceptability of these arguments (2). It was already realized that argumentation can play a role in reasoning forms in AI, particularly the defeasible ones, where it is possible that preliminary conclusions have to be withdrawn when more information becomes available (48). Because traditional (game-theoretic) negotiation has severe limitations, especially in the setting of agents, namely that positions can neither be justified nor changed, researchers have turned to *argumentation-based negotiation*. Here we have negotiations in which argumentation is employed to reach an agreement (49,50). Argumentation takes place through a dialog, that is a series of arguments, communicated between agents. This topic brings us to the important subject of agent communication.

AGENT COMMUNICATION

As we have observed before, MASs will generally involve some kind of *communication* between agents. Agents may communicate by means of a communication primitive such as a *send* (agent, performative, content), which has as semantics to send to the agent specified the content specified with a certain illocutionary force, specified by the performative (e.g., inform or request). The area of agent communication (and agent communication languages) is a field of research in itself (51). It includes the languages that are used for communication [*agent communication languages* (ACLs)]. The most well-known ACLs are KQML (Knowledge Query and Manipulation Language) (52) and FIPA (53). These languages define performatives, such as inform, request, and confirm, without mandating any specific language for message content (cf. Refs. 2 and 38). These performatives stem from *speech act* theory in the philosophy of language, which was introduced by Austin (54) and further developed by Searle (55). Here it was observed that certain natural language utterances could be viewed as actions that change the state of the world by uttering them. These speech acts (performatives) are now used as building blocks for communication protocols for agents.

A related issue, particularly within heterogeneous agent societies, concerns the *(content) language* (*ontology*) agents use to reason about their beliefs and communicate with each other. Of course, if agents stem from different sources (designers) and have different tasks, generally they will employ different and distinct ontologies (concepts and their representations) for performing their tasks. When communicating, it generally is not efficacious to try to transmit their whole ontologies to each other or to translate everything into one giant "universal" ontology if it would exist anyway. Rather, it seems that a kind of "ontology negotiation" should take place to arrive at a kind of minimal solution (i.e., sharing of concepts) that will facilitate communication between those agents (56,57).

NORMS AND E-INSTITUTIONS

In general, one has to consider the issue of balancing the individual agent's autonomy with its behaviour in an agent society. Often this is regarded in a way similar to the way human societies operate: the behaviour of an agent in a society is constrained by *norms*. These are properties that agents should adhere to, either specified declaratively or procedurally by means of *protocols*. Typically the norms that an agent has to obey concern prohibitions and permissions to perform certain actions, which may be role-dependent. A MAS in which norms govern the behaviour of the agents is called a *normative system* (58,59, p. 276). The subsystem of a normative system that specifies (and enforces) the norms on agents in a MAS, is called an *electronic institution* (*e-institution*) (60).

APPLICATIONS

This brings us to an important question: What kind of applications are particularly suited for an agent-based solution? Although it is hard to say something about this topic in general, one may particularly think of applications in which (e.g., logistical or planning) tasks may be distributed in such as way that subtasks may be (or rather preferably so) performed by autonomous entities (agents). For instance, in situations where it is virtually impossible to perform the task centrally, either because of the complexity of this task or a large degree of dynamics (changes) in the environment in which the system has to operate. So, applications such as cooperative distributed problem solving agents, task and resource allocation in agent systems, distributed sensing agents, multiagent planning, and robotic cooperating teams, as well as workflow and business management and the management of industrial systems, fall into this category. These applications have become important subareas of agent research themselves. For instance, although planning is a classic subject within AI, particularly research in distributed planning has been taken up within the context of multiagent systems (61,62).

Also in the area of information retrieval and management, especially in case of large amounts of (complex) information, there are applications of agents of this kind: Information agents may, for example, function as brokering and matchmaking entities to connect providers and users of information and services. This method is essentially a distributed way of searching information. In these applications, one thus views agents more or less as a novel computing paradigm, related to grid, peer-to-peer and ubiquitous computing.

Of course, also applications also exist in which there is a natural notion of a "cognitive" agent, that is, an agent possessing mental attitudes. For instance, in virtual environments such as (entertaining or serious) gaming where virtual characters need to behave in a natural or '*believable*' way and have human-like features, agents seem to be the obvious choice for their realization. The cognitive attitudes of an agent may also be employed fruitfully in human–machine interaction applications, in interaction with advanced software systems, and in robotic applications.

Thus, this category includes synthetic, embodied, emotional, and believable agents; agent based simulation and modeling of cognitive and social behavior; human-machine interfaces; as well as humanoid and sociable robots.

We should not forget the common-sense meaning of agent, which is "someone representing you or who looks after your affairs on your behalf". So, this definition yields the application of personal software assistants, in which agents play the role of proactive assistants to users working with some application (2). Examples are personal information agents and web agents. In the same vein, we have applications in e-commerce/e-business (such as comparison shopping agents and auction bots in auctions and electronic markets) applications for the Web (where agents may act on behalf of a user), and *mobile* agents that travel to other platforms on behalf of their users.

ADDITIONAL DEVELOPMENTS

In this section, we will sketch a few more developments. First, as agent programming gets more mature, one realizes the need for the *formal verification* of agent programs. Because it is deemed imperative to check the correctness of complex agent-based systems for very costly and life-critical applications, one tries to develop formal verification techniques for agents such as model checkers (63). By no means is this a trivial matter. Having agent logics available as mentioned above does not mean that these logics can be used directly for verification. One of the problems known from the literature (e.g., Ref. 64) is that agent logics such as BDI-CTL are not grounded in computation. That is to say the notions used in those logics, such as beliefs, desires, and intentions, as well as semantical structures such as possible worlds and accessibility relations, are not related directly to computational notions. So, researchers are working to bridge the gap between agent programs and agent logics. The aim here is to obtain verification methods, preferably (semi-) automated ones, such as proof systems and model checkers.

On another front, research considers extensions to the deliberation process of agents, which includes notions that seem to be "irrational" at first sight, such as *emotions*. The idea here is that emotions may provide heuristics in choosing between (many) alternative options of goals and plans. Thus, they enhance the decision-making capabilities of agents. Also, it is to be expected that the agent will behave more "human-like", which is an aspect that is important in its own right in certain applications (such as believable virtual characters and advanced human–machine interfaces) (see e.g., Ref. 65). Another example of reconsidering agent deliberation is in the context of "hybrid" agent systems with "humans in the loop", where there is 'mixed initiative' (from both humans and artificial agents), which gives rise to the notion of *adjustable autonomy* of the agent. That is, in some cases the agent is more autonomous (has more initiative) than in other cases (66).

CONCLUSION

In this article, we have reviewed the area of agent technology. In particular, we have recognized how the idea of agent technology and agent-oriented programming evolved from philosophical considerations about human action to a way of programming intelligent artificial (computer-based) systems. We have also looked at the important and promising subfield of MAS, in particular the main issues of interest here and possible applications. Since (multi) agent programming is a way to construct complex intelligent systems in a structured and anthropomorphic way, it appears to be a technology that is widely applicable, and it may well become one of the main programming paradigms of the future.

ACKNOWLEDGMENTS

Thanks to the anonymous referees of this article for their valuable suggestions for improvement.

BIBLIOGRAPHY

1. M. J. Wooldridge and N. R. Jennings. (eds.), *Intelligent Agents, Lecture Notes in Artificial Intelligence* vol. 890, Berlin: Springer, 1995.
2. M. J. Wooldridge. *An Introduction to MultiAgent Systems*. Chichester, UK: John Wiley & Sons, 2002.
3. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Englewood Cliffs, NJ: Prentice Hall, 1995.
4. N. J. Nilsson. *Artificial Intelligence: A New Synthesis*, San Francisco, CA. Morgan Kaufmann, 1998.
5. R. Audi (ed.), *The Cambridge Dictionary of Philosophy*, Cambridge, UK: Cambridge University Press, 1999.
6. M. J. Wooldridge. *Reasoning about Rational Agents*. Cambridge, MA: The MIT Press, 2000.
7. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, 1987.
8. D. Dennett, *The Intentional Stance*. MA: Cambridge, Bradford Books / MIT Press, 1987.
9. R. A. Wilson and F. C. Keil. (eds.), *The MIT Encyclopedia of the Cognitive Sciences*. Cambridge, MA: Bradford Book / MIT Press, 1999.
10. P. R. Cohen and H. J. Levesque, Intention is choice with commitment, *Artif. Intell*. **42**(3): 213–261. 1990.
11. A. S. Rao and M. P. Georgeff, Modeling rational agents within a BDI-architecture, in J. Allen., R. Fikes., and E. Sandewall, (eds.), *Proc. 1991 Conf. On Knowledge Representation*. San Francisco, CA: Morgan Kaufmann, 1991, pp. 473–484.
12. W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer, An integrated modal approach to rational agents, in M. Wooldridge and A. Rao., (eds.), *Foundations of Rational Agency*. Dordrecht, The Netherlands: Kluwer, 1998, pp. 133–168.
13. S. Kripke, Semantical analysis of modal logic, *Zeitschrift for Mathematische Logik und Grundlagen der Mathematik* **9**: 67–96, 1963.
14. B. Chellas, *Modal Logic: an Introduction*, Cambridge, UK: Cambridge University Press, 1980.
15. J.-J. Ch. Meyer and W. van der Hoek, *Epistemic Logic for AI and Computer Science*. Cambridge, UK: Cambridge University Press, 1995.
16. E. A. Emerson, Temporal and modal logic, in: J. vanLeeuwen (ed.), *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics* New york: Elsevier, 1990, pp. 995–1072.

17. D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*, Cambridge, MA: The MIT Press, 2000.

18. M. E. Bratman, D. J. Israel, and M. L. Pollack, plans and resource-bounded practical reasoning, *Computat. Intell.* **4**: 349–355, 1988.

19. M. P. Georgeff and A. L. Lansky, Reactive reasoning and planning, *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, Seattle, WA, 1987, pp. 677–682.

20. M. d'Inverno et al., A formal specification of dMARS, in A. Rao, M. P. Singh, and M. J. Wooldridge, (eds.), *Intelligent Agents IV* 1365, Berlin: Springer, 1997, pp. 155–176.

21. R. A. Brooks, Intelligence without reason, *Proc. IJCAI-91*, Sydney, Australia, 1991, pp. 569–595.

22. I. A. Ferguson, TouringMachines: an architecture for dynamic, rational, mobile agents. PhD Thesis, Clare Hall, cambridge UK: University of Cambridge, 1992.

23. J. Müller, A cooperation model for autonomous agents, in J. P. Müller, M. Wooldridge., and N. R. Jennings. (eds.), *Intelligent Agents III*. Lecture Notes in Artificial Intelligence vol. 1193, Berlin: Springer, 1997, pp. 245–260.

24. P. Ciancarini and M. J. Wooldridge. (eds.), *Agent-Oriented Software Engineering*, Lecture Notes in Artificial Intelligence, Vol 1957, Berlin: Springer, 2001.

25. F. Bergenti., M.-P. Gleizes., and F. Zambonelli (eds.), *Methodologies and Software Engineering for Agent Systems – The Agent-Oriented Software Engineering Handbook*, Boston MA: Kluwer, 2004.

26. F. Brazier et al., Formal specification of multi-agent systems: a real-world case, *Proc. ICMAS-95*, San Francisco, CA, 1995, pp. 25–32.

27. M. J. Wooldridge, N. R. Jennings, and D. Kinny, A methodology for agent-oriented analysis and design, *Proc. Agents '99*, Seattle, WA, 1999, pp. 69–76.

28. J. Odell, H. Parunak, and B. Bauer, Representing agent interaction protocols in UML, in P. Ciancarini and M. J. Wooldridge (eds.), *Agent-Oriented Software Engineering*. Lecture Notes in Computer Science vol. 1957, Berlin: Springer, 2001, pp. 185–194.

29. J. Castro, M. Kolp, and J. Mylopoulis, Towards requirements-driven information systems engineering: the TROPOS project, *informat. Syst.*, **27**: 365–389, 2002.

30. V. Dignum, A Model for Organisational Interaction, Based on Agents, Founded in Logic, PhD thesis, Utrecht, The Netherland. University of Utrecht, 2004.

31. M. Dastani, J. Hulstijn, F. Dignum, and J.-J. Ch. Meyer, Issues in multiagent system development, in N. R. Jennings. C. Sierra., L. Sonenberg., and M. Tambe., (eds.), *Proceedings 3rd International Joint Conference On Autonomous Agents & MultiAgent Systems (AAMAS 2004)*New York: ACM, 2004.

32. Y. Shoham, Agent-oriented programming, *Artif. Intell.* **60**(1): 51–92, 1993.

33. M. Fisher, A survey of concurrent METATEM—the language and its applications, in D. M. Gabbay and H. J. Ohlbach (eds.), *Temporal Logic* Lecture Notes in Artificial Intelligence vol. 827, Berlin; Springer, 1994, pp. 480–505.

34. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer, Agent programming in 3APL, *Internat. J. Auton. Agents Multi-Agent Sys.* **2**(4): pp. 357–401, 1999.

35. G. deGiacomo, Y. Lespérance, and H. Levesque, ConGolog, a concurrent programming language based on the situation calculus, *Artif. Intell.* **121** (1,2): pp. 109–169, 2000.

36. R. H. Bordini. M. Dastani., J. Dix., and A. El Fallah Seghrouchni (eds.), *Multi-Agent Programming (Languages, Platforms and Applications)*, New York: Springer Science, 2005.

37. J. McCarthy and P. Hayes, Some philosophical problems form the standpoint of artificial intelligence, in B. Meltzer and D. Michie., (eds.) *Machine Intelligence 4*. Edinburgh, UK: Edinburgh Univ. Press, 1969.

38. G. Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: MIT Press, 1999.

39. V. Dignum, J.-J. Ch. Meyer, H. Weigand, and F. Dignum, An Organisational-Oriented Model for Agent Societies, in G. Lindemann., D. Moldt., M. Paolucci., B. Yu., (eds.), *Proc. International Workshop on Regulated Agent-Based Social Systems: Theory and Applications (RASTA'02)*Univ. Hamburg. FBI—HH-M-318/02, 31–50, 2002.

40. T. Sandholm, Distributed rational decision making, in G. Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: MIT Press, 1999, pp. 201–258.

41. J. vonNeumann and O. Morgenstern, *Theory of Games and Economic Behaviour*. Princeton, NJ: Princeton University Press, 1944.

42. P. R. Cohen and H. J. Levesque, Rational interaction as the basis for communication, in P. R. Cohen, J. Morgan., and M. E. Pollack (eds.), *Intentions in Communication*, Cambridge, MA: MIT Press, 1990, pp. 221–256.

43. P. R. Cohen and H. J. Levesque, Teamwork, *Nous*, **25**(4): 487–512, 1991.

44. N. R. Jennings, Controlling cooperative problem solving in industrial multi-agent systems using joint intentions, *Arti. Intelli.***75**(2): 195–240, 1995,

45. M. Tambe, Towards flexible teamwork, *J. AI Res.***7**: 83–124, 1997.

46. J. S. Rosenschein and G. Zlotkin, *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. Cambridge, MA: MIT Press, 1994.

47. D. N. Walton and E. C. W. Krabbe, *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. Albany, NY: SUNY Press, 1995.

48. H. Prakken and G. Vreeswijk, Logics for defeasible argumentation, in D. Gabbay and F. Guenthner (eds.), *Handbook of Philosophical Logic*, 2nd ed. Boston, MA: Kluwer, 2001.

49. K. P. Sycara, Multiagent compromise via negotiation, in L. Gasser and M. Huhns (eds.), *Distributed Artificial Intelligence*, Vol. II. London: Pitman/Morgan Kaufmann, 1989, pp. 119–138.

50. S. Parsons, C. A. Sierra and N. R. Jennings, Agents that reason and negotiate ny arguing, *J. Logic Computat.***8**(3): 261–292, 1998,

51. F. Dignum and M. Greaves (eds.), *Issues in Agent Communication*, Lecture Notes in Artificial Intelligence, Vol. 1906, Berlin: Springer, 2000.

52. J. Mayfield, Y. Labrou, and T. Finin, Evaluating KQML as an agent communication language, in M. Wooldridge., J. P. Müller,. and M. Tambe (eds.), *Intelligent Agents II*. Lecture Notes in Artificial Intelligence vol. 1037, Berlin: Springer, 1996, pp. 347–360.

53. FIPA, Specification Part 2 – Agent Communication Language, Technical Report, 1999.

54. J. L. Austin. *How To Do Things With Words*. Oxford UK: Oxford University Press, 1962.

55. J. R. Searle, *Speech Acts: an Essay in the Philosophy of Language*, Cambridge, UK: Cambridge University Press, 1969.

56. S. Bailin and W. Truszkowski, Ontology negotiation between intelligent information agents, *Knowl. Engineer. Rev.* **17**(1): 7–19, 2002.

57. J. vanDiggelen, R. J. Beun. F. Dignum, R. M. vanEijk, and J.-J. Ch. Meyer, ANEMONE: An Effective Minimal Ontology Negotiation Environment, in P. Stone and G. Weiss., (eds.), *Proc. Fifth Int. Joint Conf. On Autonomous Agents and Multiagent Systems (AAMAS'06)* Hakodate, Hokkaido, Japan: ACM Press, 2006, pp. 899–906.

58. J.-J. Ch. Meyer and R. J. Wieringa, *Deontic Logic in Computer Science: Normative System Specification*. Chichester, UK: Wiley, 1993.

59. A. Jones and M. Sergot, On the characterization of law and computer systems: the normative system perspective, in: J.-J. Ch. Meyer and R. J. Wieringa. (eds.), *Deontic Logic in Computer Science: Normative System Specification*. Chichester, UK: Wiley, 1993, pp. 275–307.

60. M. Esteva, J. Padget, and C. Sierra, Formalizing a language for institutions and norms, in: J.-J. Ch. Meyer and M. Tambe (eds.), *Intelligent Agents VIII*, Lecture Notes in Artificial Intelligence vol. 2333, Berlin: Springer, 2001, pp. 348–366.

61. E. Ephrati and J. S. Rosenschein, Multi-agent planning as a dynamic search for social consensus, *Proc. 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, 1993, Morgan Kaufmann, San Mateo, CA, 1993, pp. 423–429.

62. D. E. Wilkins and K. Myers, A multiagent planning architecture, *Proc. 4th Int. Conf. on Artificial Intelligence Planning Systems (AIPS-98)*, Menlo Park, CA: AAAI Press, pp. 154–162, 1998.

63. J. L. Rash. C. A. Rouff., W. Truszkowski., D. Gordon., and M. G. Hinchey (eds.), *Proceedings First Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS 2000)*, Lecture Notes in Artificial Intelligence vol. 1871, Berlin: Springer, 2001.

64. W. Van der Hoek and M. Wooldridge, Towards a logic of rational agency, *Logic J. IGPL*, **11**(2): 133–157, 2003.

65. M. Dastani and J.-J. Ch. Meyer, Programming emotional agents, in G. Brewka., S. Coradeschi., A. Perini and P. Traverso., (eds.), *Proc. ECAI 2006* Riva del Garda, Amsterdam: IOS Press, 2006, pp. 215–219.

66. G. Klein, D. D. Woods. J. M. Bradshaw. R. R. Hoffman. P. J. Feltovich, Ten challenges for making automation a "team player" in joint human-agent activity, *IEEE Intell. Sys.***19**(6): 91–95, 2004,

FURTHER READING

F. Dignum, Autonomous Agents with Norms, *Artif. Intell. and Law* **7**: 1999, pp. 69–79.

M. N. Huhns and M. P. Singh. *Readings in Agents*. Morgan Kaufmann, San Francisco, CA: 1998.

(worthwhile collection of the older, seminal papers)

Autonomous Agents & Multi Agent Systems, the yearly proceedings of the international JAAMAS conferences, ACM Press, since 2002. These are the continuation of separate conferences and workshops started in the 1990s until 2002, i.e., ICMAS (Int. Conf. on MultiAgent Systems), Autonomous Agents and ATAL (Agent Theories, Architectures and Languages). There is also a journal with the same name, originally published by Kluwer and currently by Springer.

John-Jules Ch. Meyer
Utrecht University
Utrecht, The Netherlands

AGILE SOFTWARE DEVELOPMENT

> *Plan-driven methods work best when developers can determine the requirements in advance ... and when the requirements remain relatively stable, with change rates on the order of one percent per month.*
>
> —Barry Boehm (1)

WHAT IS AGILITY IN SOFTWARE DEVELOPMENT?

In this section, we discuss the model underlying agile software development and typical characteristics of software development projects that might be prudently handled via an agile development methodology.

Agile Model

Agile methods (13–15) are a subset of iterative and evolutionary methods (10,11) and are based on iterative enhancement (8) and opportunistic development processes (16). Each iteration is a self-contained mini-project with activities that span requirements analysis, design, implementation, and testing (10). Each iteration leads to an iteration release (which may be only an internal release) that integrates all software across the team and is a growing and evolving subset of the final system. The purpose of having short iterations is so that feedback from iterations N and earlier, and any other new information, can lead to refinement and requirements adaptation for iteration $N + 1$. The customer adaptively specifies his or her requirements for the next release based on observation of the evolving product, rather than speculation at the start of the project (6). There is quantitative evidence that frequent deadlines reduce the variance of a software process and, thus, may increase its predictability and efficiency (17).

The pre-determined iteration length serves as a timebox for the team. Scope is chosen for each iteration to fill the iteration length. Rather than increase the iteration length to fit the chosen scope, the scope is reduced to fit the iteration length. A key difference between agile methods and past iterative methods is the length of each iteration. In the past, iterations might have been three-or six-months long. With agile methods, iteration lengths vary between one and four weeks, and intentionally do not exceed 30 days.

Research has shown that shorter iterations have lower complexity and risk, better feedback, and higher productivity and success rates (10).

An area of commonality among all agile methodologies is the importance of the people performing the roles and the recognition that, more so than any process or tool, people are the most influential factor in any software project. Brooks acknowledges the same in *The Mythical Man Month* (18),

> *The quality of the people on a project, and their organization and management, are more important factors in success than are the tools they use or the technical approaches they take.*

Unfortunately, there are commonalities among some agile methods that may be less than positive. One is that, unlike more classic iterative methods, explicit quantitative quality measurements and process modeling and metrics are de-emphasized. Possible justifications for this lack of modeling and metrics range from lack of time, to lack of skills, to intrusiveness, to social reasons. Additionally, only relatively small agile teams will be likely to be able to self-organize; self-organization is one of the agile principles) into something resembling a Software Engineering Institute (SEI) Capability Maturity Model (CMM) scale (19). Level 4 or 5 organization.

Agile Development and Principles

In February 2001, several software engineering consultants joined forces and began to classify a number of similar change-sensitive methodologies as *agile* [a term with a decade of use in flexible manufacturing practices (20,21) which began to be used for software development in the late 1990s (22)]. The term promoted the professed ability for rapid and flexible response to change of the methodologies. The consultants formed the Agile Alliance and wrote *The Manifesto for Agile Software Development* and the *Principles Behind the Agile Manifesto* (23,24). The methodologies originally embraced by the Agile Alliance were Adaptive Software Development (ASD) (25), Crystal (14,26), Dynamic Systems Development Method (DSDM) (27), Extreme Programming (XP) (28), Feature-Driven Development (FDD) (29,30) and Scrum (31,32).

Agile Software Development Values. The Agile Alliance documented its value statement (24) as follows:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions	*over*	*processes and tools*
Working software	*over*	*comprehensive documentation*
Customer collaboration	*over*	*contract negotiation*
Responding to change	*over*	*following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

The implication is that formalization of the software process hinders the human and practical component of software development, and thus reduces the chance for success. Although this statement is true when formalization is misused and misunderstood, one has to be very careful not to overemphasize and under-measure the items on the left-hand side, which can lead to the same problem, poor quality software. The key is appropriate balance (33).

The Principles. The Agile Alliance also documented the principles they follow that underlie their manifesto (24). As such, the agile methods are principle-based rather than

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

rule-based (10). Rather than have pre-defined rules regarding the roles, relationships, and activities, the team and manager are guided by these principles:

1. *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.*
4. *Business people and developers must work together daily through the project.*
5. *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
7. *Working software is the primary measure of progress.*
8. *Agile processes promote sustainable development.*
9. *The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
10. *Continuous attention to technical excellence and good design enhances agility.*
11. *Simplicity – the art of maximizing the amount of work not done – is essential.*
12. *The best architectures, requirements, and designs emerge from self-organizing teams.*
13. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

Characteristics of Projects Suitable for Agile Methods

Agile software development methodologies are not necessarily suitable for all projects. Boehm and Turner (33) share their view of the characteristics of projects that operate in agile "home grounds," where home ground is defined as the situation for which the approach has the greatest potential for success. The agile home ground is summarized in Table 1. It is important to note that some agile methodologists feel that their methodologies are suitable for most any project. Empirical and theoretical studies are needed to both support and refute the characteristics described by this home ground and the scope, effectiveness, and cost of the agile approach.

The general characteristics of agile and plan-driven methods led Boehm and Turner (33) to define five critical factors that can be used to describe a project environment and can be used to help determine the appropriate balance between agile and plan-driven methods:

1. Size (number of people on team).
2. Criticality (the impact of a software defect in terms of comfort, money, or lives).
3. Dynamism (the degree of requirements and technology change).
4. Personnel[1] (ratio of high and low skill level of team-to-team size).
5. Culture (whether the individuals on the team prefer predictability/order or change).

Table 1. Agile and Plan-driven Home Grounds (Adapted from Ref. 34)

Project Characteristics	Agile Home Ground
Primary goals	Rapid value, responding to change
Size	Smaller teams and projects
Environment	Turbulent, high change, project focused
Customer relations	Dedicated on-site customer, focused on prioritized product releases (increments)
Planning and control	Team has an understanding of plans and monitors to this plan
Communications	Passed from person to person (tacit, interpersonal)
Requirements	Prioritized, informal stories and/or features and more formal use cases. Requirements are likely to change in unpredictable ways
Development	Simple design, short increments
Test	Automated, executable test cases are used to further define the specifics of the requirements
Customers	Dedicated, co-located CRACK* performers
Developers	At least 30% experts or very experienced team members; no unexperienced personnel
Culture	Team enjoys being empowered and having freedom (thriving on chaos)

*CRACK = Collaborative, Representative, Authorized, Committed, and Knowledgeable

Boehm and Turner have created a polar chart as a means for visually displaying a team's values for each of these criticality factors. An example of such a polar chart is in Fig. 1. Each of the five factors has an axis. Each axis is labeled with carefully chosen values based on Boehm and Turner's experience. For each axis, the further from the graph's center the project lies, the more appropriate are plan-driven methods. Conversely, the more a point lies toward the center of the chart, the more a project may benefit from agile methods.

Consider the black polygon joining the points of a sample project in Fig.1. Starting at the top of the chart (Personnel), this team is comprised of a large number of novices and a small number of experts. Additionally, the requirements (Dynamism) are not expected to change much throughout the project. The team members have a fairly strong preference for order and predictability (Culture). There are

[1] Boehm and Turner adapt levels of Software Method Understanding and Use as defined by Cockburn (14). 1B personnel are team members who can perform procedural steps with training. 1A personnel are able to perform discretionary method steps. Level 2 personnel are able to tailor a method to fit a new situation. Level 3 personnel are able to revise a method to fit an unprecedented situation.

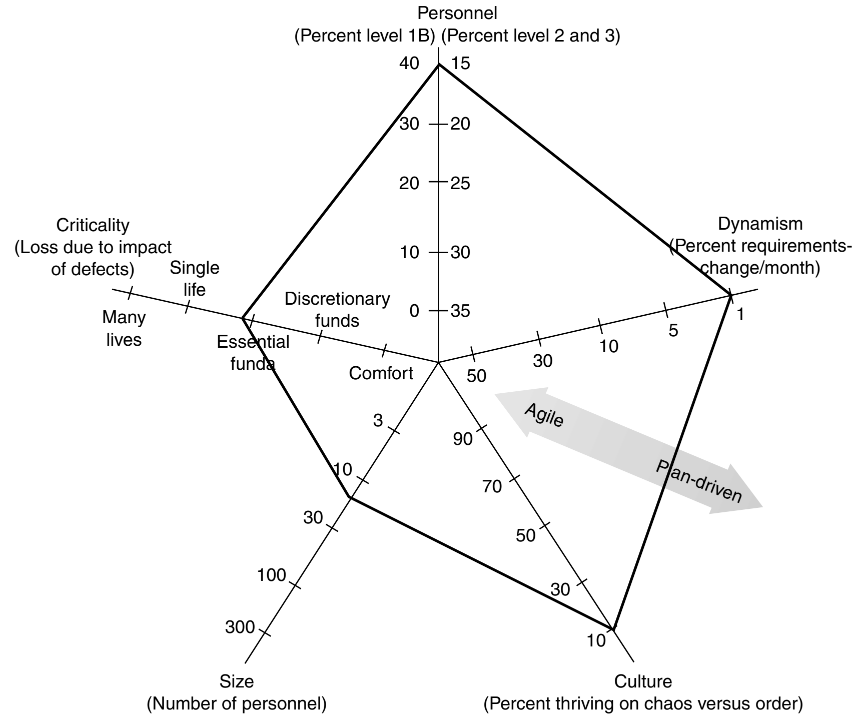


Figure 1. Example polar chart (adapted from Ref. 33).

about 15 people on the team (Size). The impact of a software defect could result in the loss of essential funds (Criticality) (i.e., the business could lose a large amount of money if the software fails). An example could be a retail point-of-sale application failure in which customers leave the store because the computers are not working. Based on the shape of the polar chart for this particular application, the project may not be suitable for an agile methodology. As a general rule, high-risk projects (34) may require more than agile methodologies may offer.

EXAMPLES OF AGILE SOFTWARE DEVELOPMENT METHODOLOGIES

This section provides a brief introduction to three agile methodologies. The three were chosen to demonstrate the range of applicability and specification of the agile methodologies. For each methodology, we provide an overview and discuss documents and artifacts produced by the development team, the roles the members of the development team assume, the process, and a discussion.

Extreme Programming (XP)

XP (28,35) originators aimed at developing a methodology suitable for "object-oriented projects using teams of a dozen or fewer programmers in one location" (36).

The methodology is based on five underlying values: communication, simplicity, feedback, courage, and respect.

1. **Communication.** XP has a culture of oral communication and its practices are designed to encourage interaction. The communication value is based on the observation that most project difficulties occur because someone *should have* spoken with someone else to clarify a question, collaborate, or obtain help. "Problems with projects can invariably be traced back to somebody not talking to somebody else about something important" (28).
2. **Simplicity.** Design the simplest product that meets the customer's needs. An important aspect of the value is to only design and code what is in the current requirements rather than to anticipate and plan for unstated requirements.
3. **Feedback.** The development team obtains feedback from the customers at the end of each iteration and external release. This feedback drives the next iteration. Additionally, there are very short design and implementation feedback loops built into the methodology via pair programming and test-driven development (37).
4. **Courage.** The other three values allow the team to have courage in its actions and decision making. For example, the development team might have the courage to resist pressure to make unrealistic commitments.
5. **Respect.** Team members need to care about each other and about the project.

Although one may be able to come up with reasonable quantitative metrics for the first three values, it is quite difficult to assess the other two, at least in advance, unless quantitative risk-based metrics and models are used (34,38).

Documents and Artifacts. In general, XP relies on "documentation" via oral communication, the code itself, and tacit knowledge transfer rather than written documents and artifacts. The following relatively informal artifacts are produced:

- **Story cards**, paper index cards that contain brief requirement descriptions. The user story cards are intentionally not a full requirement statement but are, instead, a commitment for further conversation between the developer and the customer. During this conversation, the two parties will come to an oral understanding of what is needed for the requirement to be fulfilled. Customer priority and developer resource estimate are added to the card. The resource estimate for a user story must not exceed the iteration duration.
- **Task list**, a listing of the tasks (typically one-half to three days in duration) for the user stories that are to be completed for an iteration. Tasks represent concrete aspects of a story. Programmers volunteer for tasks rather than being assigned to tasks.
- **CRC cards (39) (optional)**, paper index card on which one records the responsibilities and collaborators of classes that can serve as a basis for software design. The classes, responsibilities, and collaborators are identified during a design brainstorming/role-playing session involving multiple developers. CRC stands for **C**lass-**R**esponsibility-**C**ollaboration.
- **Customer acceptance tests**, textual descriptions and automated test cases that are developed by the customer. The development team demonstrates the completion of a user story and the validation of customer requirements by passing these test cases.
- **Visible wall graphs**, to foster communication and accountability, progress graphs are usually posted in a team work area. These progress graphs often involve how many stories are completed or how many acceptance test cases are passing.

Roles

- **Manager**, owns the team and its problems. He or she forms the team, obtains resources, manages people and problems, and interfaces with external groups.
- **Coach**, teaches team members about the XP process as necessary, intervenes in case of issues, and monitors whether the XP process is being followed. The coach is typically a programmer and not a manager.
- **Tracker**, regularly collects user story and acceptance test case progress from the developers to create the visible wall graphs. The tracker is a programmer, not a manager or customer.
- **Programmer**, writes, tests, and designs, code; refactors; and identifies and estimates tasks and stories (this person may also be a tester).
- **Tester**, helps customers write and develop tests (this person may also be a programmer).
- **Customer**, writes stories and acceptance tests; picks stories for a release and for an iteration. A common misconception is that the role of the customer must be played by one individual from the customer organization. Conversely, a group of customers can be involved or a customer representative can be chosen from within the development organization (but external to the development team).

Process. The initial version of the XP software methodology (28) published in 2000 had 12 programmer-centric, technical practices. These practices interact, counterbalance, and reinforce each other (13,28). However, in a survey (40) of project managers, chief executive officers, developers, and vice-presidents of engineering for 21 software projects, it was found that none of the companies adopted XP in a “pure” form wherein all 12 practices were used without adaptation. In 2005, XP was changed to include 13 primary practices and 11 corollary practices (35). The primary practices are intended to be (35) useful independent of each other and the other practices used, although the interactions between the practices may amplify their effect. The corollary practices are likely to be difficult without first mastering a core set of the primary practices.

Below, the 13 primary technical practices of XP are briefly described:

- **Sit together**, the whole team develops in one open space.
- **Whole team**, uses a cross-functional team of all those necessary for the product to succeed.
- **Informative workspace**, place visible wall graphs around the workspace so that team members (or other interested observers) can get a general idea of how the project is going.
- **Energized work**, XP teams do not work excessive overtime for long periods of time. The motivation behind this practice is to keep the code of high quality (tired programmers inject more defects) and the programmers happy (to reduce employee turnover). Tom DeMarco contends that, “extended overtime is a productivity-reducing technique” (41).
- **Pair programming (42)**, refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test.
- **Stories**, the team members write short statements of customer-visible functionality desired in the product. The developers estimate the story; the customer prioritizes the story.
- **Weekly cycle**, at the beginning of each week, a meeting is held to review progress to date, have the customer pick a week’s worth of stories to implement that week (based on developer estimates and their own priority), and to break the stories into tasks to be completed that week. By the end of the week, acceptance test cases for the chosen stories should be running for demonstration to the customer to drive the next weekly cycle.

- **Quarterly cycle**, the whole team should pick a theme or themes of stories for a quarter's worth of stories. Themes help the team reflect on the bigger picture (i.e., at the end of the quarter, deliver this business value).
- **Slack**, in every iteration, plan some lower-priority tasks that can be dropped if the team gets behind such that the customer will still be delivered their most important functionality.
- **Ten-minute build**, structure the project and its associated tests such that the whole system can be built and all the tests can be run in ten minutes so that the system will be built and the tests will be run often.
- **Test-first programming**, all stories have at least one acceptance test, preferably automated. When the acceptance test(s) for a user story all pass, the story is considered to be fulfilled. Additionally, automated unit tests are incrementally written using the test-driven development (TDD) (43) practice in which code and automated unit tests are alternately and incrementally written on a minute-by-minute basis.
- **Continuous integration**, programmers check into the code base completed code and its associated tests several times a day. Code may only be checked in if all its associated unit tests and all unit tests of the entire code base pass.
- **Incremental design**, rather than develop an anticipatory detailed design before implementation, invest in the design of the system every day in light of the experience of the past. The viability and prudence of anticipatory design has changed dramatically in our volatile business environment (13). Refactoring (44) to improve the design of previously written code is essential. Teams with robust unit tests can safely experiment with refactorings because a safety net is in place.

Below, the 11 corollary technical practices of XP are briefly described:

- **Real customer involvement**, the customer is available to clarify requirements questions, is a subject matter expert, and is empowered to make decisions about the requirements and their priority. Additionally, the customer writes the acceptance tests.
- **Incremental deployment**, gradually deploy functionality in a live environment to reduce the risk of a big deployment.
- **Team continuity**, keep effective teams together.
- **Shrinking team**, as a team grows in capacity (because of experience), keep their workload constant but gradually reduce the size of the team.
- **Root cause analysis**, examine the cause of a discovered defect by writing acceptance test(s) and unit test(s) to reveal the defect. Subsequently, examine why the defects was created but not caught in the development process.
- **Shared code**, once code and its associated tests are checked into the code base, the code can be altered by any team member. This collective code ownership provides each team member with the feeling of owning the whole code base and prevents bottlenecks that might have been caused if the "owner" of a component was not available to make a necessary change.
- **Code and tests**, maintain only the code and tests as permanent artifacts. Rely on social mechanisms to keep alive the important history of the project.
- **Daily deployment**, put new code into production every night.
- **Negotiated scope contract**, fix the time, cost, and required quality of a project but call for an ongoing negotiation of the scope of the project.
- **Pay-per-use**, charge the user every time the system is used to obtain their feedback by their usage patterns.

Discussion. The main advantages of XP relative to small, co-located teams have been demonstrated by several industrial case studies, including (45–49):

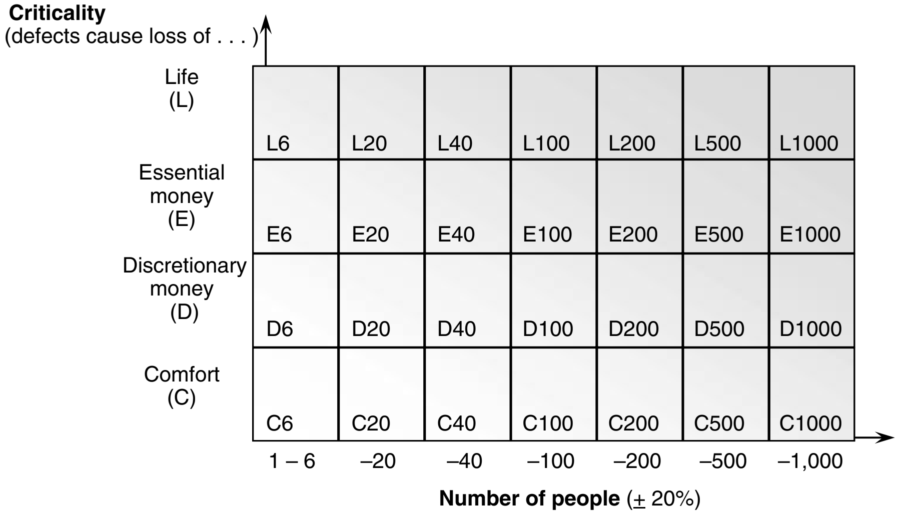
- Improved quality;
- Improved productivity (although the measures were relatively inexact);
- Improved team morale;
- Anecdotally, improved customer satisfaction.

The possible drawbacks of XP are as follows:

- XP may not be applicable for other than small, co-located teams developing noncritical software, although XP has been successfully used with mission-critical projects (50), distributed teams (51), and for scientific research (52).
- XP de-emphasizes documentation and relies on social mechanisms to keep alive the important history of the project. As a result, XP must be adapted for projects that require traceability and audit-ability.
- Some developers may not transition to pair programming easily; transitioning to the test-driven development practice may require technical training for some developers.
- The real customer involvement practice has shown to be very effective for communicating and clarifying requirements, but is a pressured, stressful, and time-consuming role (53).

Crystal

The RUP (4,5) is a customizable process framework. Depending on the project characteristics, such as team size and project size, the RUP can be tailored or extended to match the needs of an adopting organization. Similarly, the family of Crystal Methods (14) were developed to address the variability of the environment and the specific characteristics of the project. However, RUP generally starts with a plan-driven base methodology and tailors down for smaller, less-critical projects. Conversely, Crystal author Alistair Cockburn feels that the base methodology should be "barely sufficient." He contends, "You need one less notch of control than you expect, and less is better when it comes to delivering quickly," (13). Moreover, because the



Life (L)	L6	L20	L40	L100	L200	L500	L1000
Essential money (E)	E6	E20	E40	E100	E200	E500	E1000
Discretionary money (D)	D6	D20	D40	D100	D200	D500	D1000
Comfort (C)	C6	C20	C40	C100	C200	C500	C1000
	1 – 6	–20	–40	–100	–200	–500	–1,000

Figure 2. The family of Crystal Methods (adapted from Ref. 14).

project and the people evolve over time, the methodology so too must be tuned and evolved during the course of the project.

Crystal is a family of methods because Cockburn believes that there is no "one-size-fits-all" development process. As such, the different methods are assigned colors arranged in ascending opacity; the most agile version is Crystal Clear, followed by Crystal Yellow, Crystal Orange, and Crystal Red. The graph in Fig. 2 is used to aid the choice of a Crystal Method starting point (for later tailoring). Along the x-axis is the size of the team. As a team gets larger (moves to the right along the x-axis), the harder it is to manage the process via face-to-face communication and, thus, the greater the need for coordinating documentation, practices, and tools. The y-axis addresses the system's potential for causing damage. The lowest damage impact is loss of comfort, then loss of discretionary money, loss of essential money, and finally loss of life. Based on the team size and the criticality, the corresponding Crystal methodology is identified. Each methodology has a set of recommended practices, a core set of roles, work products, techniques, and notations.

All the Crystal Methods emphasize the importance of people in developing software. "[Crystal] focuses on people, interaction, community, skills, talents, and communication as first order effects on performance. Process remains important, but secondary" (13). There are only two absolute rules of the Crystal family of methodologies. First, incremental cycles must not exceed 4 months. Second, reflection workshops must be held after every delivery so that the methodology is self-adapting. Currently, only Crystal Clear and Crystal Orange have been defined. Summaries of these two methodologies are provided below.

Crystal Clear. Crystal Clear (14,26) is targeted at a D6 project and could be applied to a C6 or an E6 project and possibly to a D10 project (see Fig. 2). Crystal Clear is an optimization of Crystal that can be applied when the team consists of three to eight people sitting in the same room or adjoining offices. The property of close communication is strengthened to "osmotic" communication, meaning that people overhear each other discussing project priorities, status, requirements, and design on a daily basis. Crystal Clear's model elements are as follows:

- **Documents and artifacts**: release plan, schedule of reviews, informal/low-ceremony use cases, design sketches, running code, common object model, test cases, and user manual.
- **Roles**: project sponsor/customer, senior designer-programmer, designer-programmer, and user (part time at least).
- **Process**: incremental delivery, releases less than 2–3 months, some automated testing, direct user involvement, two user reviews per release, and methodology-tuning retrospectives. Progress is tracked by software delivered or major decisions reached, not by documents completed.

Crystal Orange. Crystal Orange is targeted at a D40 project. Crystal Orange is for 20–40 programmers working together in one building on a project in which defects could cause the loss of discretionary money (i.e., medium risk). The project duration is between 1 and 2 years and time-to-market is important. Crystal Orange's model elements are as follows:

- **Documents and artifacts**: requirements document, release plan, schedule, status reports, UI design document, inter-team specs, running code, common object model, test cases, migration code, and user manual.
- **Roles**: project sponsor, business expert, usage expert, technical facilitator, business analyst, project manager, architect, design mentor, lead designer-programmer, designer-programmer, UI designer, reuse point, writer, and tester.
- **Process**: incremental delivery, releases less than 3–4 months, some automated testing, direct user involvement, two user reviews per release, and methodology-tuning retrospectives.

Discussion. No empirical case studies of Crystal teams have been published. Anecdotally, the main advantages of Crystal methods are as follows:

- The family of methods accommodates teams of any size and criticality.
- The philosophy underlying the Crystal methods emphasizes simplicity, agility, and communication.

Possible drawbacks of Crystal are as follows:

- The flexibility of the methods may not provide enough prescriptive guidance to all teams on which software development practices to use.
- Not all of the Crystal methods have been defined.

Feature-driven Development (FDD)

FDD (29,30) authors Peter Coad and Jeff de Luca characterize the methodology as having "just enough process to ensure scalability and repeatability and encourage creativity and innovation all along the way" (13). Throughout, FDD emphasizes the importance of having good people and strong domain experts. FDD is build around eight best practices: domain object modeling; developing by feature; individual class ownership; feature teams; inspections; regular builds; configuration management; and reporting/visibility of results. UML models (54,55) are used extensively in FDD.

Documents and Artifacts

- **Feature lists**, consisting of a set of features whereby features are small, useful in the eyes of the client, a client-valued function that can be implemented in two weeks or less. If a feature would take more than two weeks to implement, it must be further decomposed.
- **Design packages**, which consist of sequence diagrams and class diagrams and method design information
- **Track by Feature**, a chart that enumerates the features that are to be built and the dates when each milestone has been completed.
- **"Burn Up" Chart**, a chart that has dates (time) on the x axis. On the y axis is an increasing number of features that have been completed. As features are completed, this chart indicates a positive slope over time.

Roles

- **Project manager**, the administrative lead of the project responsible for reporting progress, managing budgets, and fighting for and managing resources including people, equipment, and space.
- **Chief architect**, responsible for the overall design of the system including running workshop design sessions with the team.
- **Development manager**, responsible for leading the day-to-day development activities including the resolution of resource conflicts.
- **Chief programmer**, as outlined by Brooks' ideas on surgical teams (18), an experienced developer who acts as a team lead, mentor, and developer for a team of three to six developers. The chief programmer provides the breadth of knowledge about the skeletal model to a feature team, participates in high-level requirements analysis and design, and aids the team in low-level analysis, design, and development of new features.
- **Class owner**, responsible for designing, coding, testing, and documenting new features in the classes that he or she owns.
- **Domain experts**, users, clients, sponsors, business analysts, and so on who have deep knowledge of the business for which the product is being developed.
- **Feature teams**, temporary groups of developers formed around the classes with which the features will be implemented. A feature team dynamically forms to implement a feature and disbands when the feature has been implemented (two weeks or less).

Process. The FDD process has five incremental, iterative processes. Guidelines are given for the amount of time that should be spent in each of these steps, constraining the amount of time spent in overall planning and architecture and emphasizing the amount of time designing and building features. Processes 1 through 3 are done at the start of a project and then updated throughout the development cycle. Processes 4 and 5 are done incrementally on 2-week cycles. Each of these processes has specific entry and exit criteria, whereby the entry criterion of Process N is the exit criteria of Process N-1.

- **Process 1: Develop an overall model** (time: 10% initially, 4% ongoing).
  Domain and development team members work together to understand the scope of the system and its context. High-level object models/class diagrams are developed for each area of the problem domain. Model notes record information about the model's shape and why some alternatives were selected and others rejected.
- **Process 2: Build a features list** (time: 4% initially, 1% ongoing).
  Complete list of all the features in the project; functional decomposition that breaks down a "business activity" requested by the customer to the features that need to be implemented in the software.
- **Process 3: Plan by feature** (time: 2% initially, 2% ongoing).
  A planning team consisting of the project manager, development manager, and chief programmer plan the order in which features will be developed. Planning is based on dependencies, risk, complexity, workload balancing, client-required milestones, and checkpoints. Business activities are assigned month/year

completion dates. Every class is assigned to a specific developer. Features are bundled according to technical reasons rather than business reasons.

- **Process 4: Design by feature** (time: 34% ongoing in 2-week iterations).
  The chief programmer leads the development of design packages and refines object models with attributes. The sequence diagrams are often done as a group activity. The class diagrams and object models are done by the class owners. Domain experts interact with the team to refine the feature requirements. Designs are inspected.
- **Process 5: Build by feature** (time: 43% ongoing in 2-week iterations).
  The feature team implements the classes and methods outlined by the design. This code is inspected and unit tested. The code is promoted to the build.

Progress is tracked and made visible during the design-by-feature/build-by-feature phases. Each feature has six milestones, three from the design-by-feature phase (domain walkthrough, design, and design inspection) and three from the build-by-feature phase (code, code inspection, promote to build). When these milestones are complete, the date is placed on the track-by-feature chart, which is prominently displayed for the team. When a feature has completed all six milestones, this completion is reflected on the "Burn Up" chart. All features are scoped to be completed within a maximum of two weeks, including all six milestones.

Discussion. No empirical case studies of FDD teams have been published. Anecdotally, the main advantages of FDD are as follows:

- Teams that value and are accustomed to object-oriented analysis and design and associated documentation and inspections will transition to FDD more easily than some of the other agile methods.
- The documentation produced could lead to higher quality projects and enable traceability and auditability.

Possible drawbacks of FDD are as follows:

- The up-front design may not make FDD as agile as other methodologies.
- Teams must purchase and use UML design tools.

SUMMARY

In this article, we presented an overview of the agile software development model and the characteristics of the projects that may be suited for the use of this model. Additionally, we provided overviews of three representative methodologies: XP, Crystal, and FDD. A summary of the distinguishing factors of these three methodologies is presented in Table 2.

Table 2. Summary of XP, Crystal, and FDD Methodologiese

Agile Methodology	Distinguishing Factor
XP	• Intended for 10–12 co-located, object-oriented programmers • Five values • 13 primary and 11 corollary highly specified, disciplined development practices • Minimal archival documentation • Rapid customer and developer feedback loops
Crystal	• Customizable family of development methodologies for small to very large teams • Methodology dependent on size of team and criticality of project • Emphasis of face-to-face communication • Consider people, interaction, community, skills, talents, and communication as first-order effects • Start with minimal process and build up as absolutely necessary
FDD	• Scalable to larger teams • Highly specified development practices • Five subprocesses, each defined with entry and exit criteria • Development are architectural shape, object models, and sequence diagrams (UML models used throughout) • 2 week features

There are other defined agile software development methodologies as well, including ASD (25), Agile Modeling (56), DSDM (27), Lean Development (57), and Scrum (13,31). Additionally, teams can configure an agile RUP methodology. All agile methodologies consider software development to be an empirical process that necessitates short "inspect and adapt" feedback loops throughout the project. Furthermore, although we have not explicitly focused on such in this article, all of the agile methods do support either implied or explicit verification and validation processes. In some instances, it manifests as pair programming, in some as test-driven development, and in some as explicit unit, integration, system, and acceptance testing of the classic type. In general, there is also an underlying business model that often hinges on customer satisfaction, but in reality is very much driven by resource constraints and end-user expectations in terms of functionality and end-product quality. It is probably wise for a potential adopter of agile methodologies to first explicitly define the business model (including expected resource constraints and product quality), and then pick an appropriate process model (using a combination of scientific quantitative and qualitative methods) based on that information and the project characteristics (33).

BIBLIOGRAPHY

1. B. Boehm, Get ready for agile methods, with care, *IEEE Computer*, **35**(1): 64–69, 2002.
2. W. S. Humphrey, *A Discipline for Software Engineering*. Reading, MA: Addison Wesley, 1995.
3. W. S. Humphrey, *PSP$_{sm}$: A Self-Improvement Process for Software Engineers*. Upper Saddle River, NJ: Addison-Wesley, 2005.
4. P . Kroll and P. Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Boston, MA: Addison-Wesley, 2003.
5. P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd ed. Boston, MA: Addison-Wesley, 2004.
6. B. Boehm, A spiral model for software development and enhancement, *Computer*, **21**(5): 61–72, 1988.
7. B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
8. V. R. Basili and A. J. Turner, Iterative enhancement: A practical technique for software development, *IEEE Transactions on Software Engineering*, **1**(4): 266–270, 1975.
9. R. Fairley, *Software Engineering Concepts*. New York: McGraw-Hill, 1985.
10. C. Larman, *Agile and Iterative Development: A Manager's Guide*. Boston, MA: Addison-Wesley, 2004.
11. C . Larman and V. Basili, A history of iterative and incremental development, *IEEE Computer*, **36**(6): 47–56, 2003.
12. L . Williams and A. Cockburn, Special issue on agile methods, *IEEE Computer*, **36**(3): 2003.
13. J. Highsmith, *Agile Software Development Ecosystems*. Boston, MA: Addison-Wesley, 2002.
14. A. Cockburn, *Agile Software Development*. Reading, MA: Addison Wesley Longman, 2001.
15. P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, New directions in agile methods: A comparative analysis, *International Conference on Software Engineering (ICSE 203)*, Portland, OR, 2003, pp. 244–254.
16. B. Curtis, Three Problems Overcome with Behavioral Models of the Software Development Process (Panel), *International Conf. on Software Engineering*, Pittsburgh, PA, 1989, pp. 398–399.
17. T . Potok and M. Vouk, The effects of the business model on the object-oriented software development productivity, *IBM Syst. J.,* **36**(1): 140–161, 1997.
18. F. P. Brooks, *The Mythical Man-Month, Anniversary Edition*. Reading, MA: Addison-Wesley, 1995.
19. M. C. Paulk, B. Curtis, and M. B. Chrisis, Capability maturity model for software version 1.1, *Software Engineering Institute CMU/SEI-93-TR*, February 24, 1993.
20. Lehigh University, Agile competition is spreading to the world,1991. Available: *http://www.ie.lehigh.edu/*
21. R. Dove, *Response Ability: The Language, Structure and Culture of the Agile Enterprise*. New York: Wiley.
22. M. Aoyama, Agile software process and its experience, *International Conference on Software Engineering*, Kyoto, Japan, 1998, pp. 3–12.
23. M . Fowler and J. Highsmith, The Agile Manifesto, in *Software Development*, August 2001, pp. 28–32.
24. K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, The Agile Manifesto, 2001, http://www.agileAlliance.org.
25. J. Highsmith, *Adaptive Software Development*. New York: Dorset House, 1999.
26. A. Cockburn, *Crystal "Clear": A human-powered software development methodology for small teams*. Boston, MA: Addison-Wesley, 2005.
27. J. Stapleton, *DSDM: The Method in Practice*, 2nd ed: Addison-Wesley Longman, 2003.
28. K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley, 2000.
29. S. R. Palmer and J. M. Felsing, *A Practical Guide to Feature-Driven Development*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.
30. P. Coad, E. LeFebvre, and J. DeLuca, *Java Modeling in Color with UML.*Englewood Cliffs, NJ: Prentice Hall, 1999.
31. K . Schwaber and M. Beedle, *Agile Software Development with SCRUM*. Upper Saddle River, NJ: Prentice-Hall, 2002.
32. K. Schwaber, *Agile Project Management with SCRUM*. Redmond, WA: Microsoft Press, 2004.
33. B . Boehm and R. Turner, Using risk to balance agile and plan-driven methods, *IEEE Computer*, **36**(6): 57–66, 2003.
34. B. Boehm, *Software Risk Management*. Washington, DC: IEEE Computer Society Press, 1989.
35. K. Beck, *Extreme Programming Explained: Embrace Change*, 2nd ed., Reading, MA: Addison-Wesley, 2005.
36. R. Jeffries, A. Anderson, and C. Hendrickson, *Extreme Programming Installed*. Upper Saddle River, NJ: Addison-Wesley, 2001.
37. L. Williams, The XP programmer: The few minutes programmer, *IEEE Software*, **20**(3): 16–20, 2003.
38. M . Vouk and A. T. Rivers, Construction of reliable software in resource-constrained environments, in W. R. Blischke and D. N. P. Murthy, (eds.) *Case Studies in Reliability and Maintenance*, Hoboken, NJ: Wiley-Interscience, John Wiley and Sons, 2003, pp. 205–231.
39. D . Bellin and S. S. Simone, *The CRC Card Book*. Reading, MA: Addison-Wesley, 1997.
40. K. El Emam, Finding success in small software projects, *Agile Project Management*, **4**(11): 2003.
41. T. DeMarco, *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency.*New York: Broadway, 2002.
42. L . Williams and R. Kessler, *Pair Programming Illuminated*. Reading, MA: Addison-Wesley, 2003.
43. K. Beck, *Test Driven Development – by Example*. Boston, MA: Addison Wesley, 2003.
44. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the design of existing code*. Reading, MA: Addison-Wesley, 1999.
45. P. Abrahamsson. Extreme programming: First results from a controlled case study, *29th EUROMICRO Conference*, Belek, Turkey, 2003.
46. J. Grenning, Launching extreme programming at a process-intensive company, *IEEE Software*, **18**(6): 27–33, 2001.
47. L. Layman, L. Williams, and L. Cunningham, Exploring extreme programming in context: An industrial case study, *Agile Development Conference*, Salt Lake City, UT, 2004, pp. 32–41.
48. L. Layman, L. Williams, and L. Cunningham, Motivations and measurements in an agile case study, *ACM SIGSOFT Foundation in Software Engineering Workshop Quantitative*

Techniques for Software Agile Processes (QTE-SWAP), Newport Beach, CA, 2004.

49. L. Williams, W. Krebs, L. Layman, A. Antón, and P. Abrahamsson, Toward a framework for evaluating extreme programming, *Empirical Assessment in Software Eng. (EASE) 2004*, Edinburgh, Scotland, 2004, pp. 11–20.
50. J. Drobka, D. Noftz, and R. Raghu, Piloting XP on Four Mission-Critical Projects, *IEEE Software*, **21**(6): 70–75, 2004.
51. L. Laymana, L. Williams, D. Damian, and H. Buresc, Essential communication practices for extreme programming in a global software development team, *Information and Software Technology (TBD)*, **48**(9): 781–794, 2006.
52. W . Wood and W. Kleb, Exploring XP for scientific research, *IEEE Software*, **20**(3): 42–54, 2003.
53. A. Martin, R. Biddle, and J. Noble, The XP customer role in practice: Three studies, *Agile Development Conference*, Salt Lake City, UT, 2004.
54. M. Fowler, *UML Distilled*, 3rd ed. Reading, MA: Addison-Wesley, 2004.
55. J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Reading, MA: Addison-Wesley, 1999.
56. S. W. Ambler, *Agile Modeling*. New York: Wiley, 2002.
57. M . Poppendieck and T. Poppendieck, *Lean Software Development*. Boston, MA: Addison-Wesley, 2003.

LAURIE WILLIAMS
MLADEN VOUK
North Carolina State University
Raleigh, North Carolina

ANALYTICAL CUSTOMER RELATIONSHIP MANAGEMENT

INTRODUCTION

As bandwidth continues to grow, and newer information appliances become available, marketing departments everywhere see this as an opportunity to get in closer touch with potential customers. In addition, with organizations constantly developing more cost-effective means of customer contact, the amount of customer solicitation has been on a steady rise. Today, with the internet as the ultimate low-latency, high-bandwidth, customer contact channel with practically zero cost, customer solicitation has reached unprecedented levels.

Armed with such tools, every organization has ramped up its marketing effort, and we are witnessing a barrage of solicitations targeted at the ever-shrinking attention span of the same set of customers. Once we consider the fact that potentially good customers, i.e., “those likely to buy a product,” are much more likely to get a solicitation than those who are not so good, the situation for the good customers is even more dire. This issue is really testing the patience of many customers, and thus, we have witnessed a spate of customers signing up to be on “no solicitation” lists, to avoid being bombarded with unwanted solicitations.

From the viewpoint of the organizations, the situation is no better. Even though the cost of unit customer communication has dropped dramatically, the impact of unit communication has dropped even faster. For example, after a lot of initial enthusiasm, it is now widely accepted that the impact of web page banner advertisements in affecting customer opinion is practically negligible. On the other hand, the impact of targeted e-mails, especially with financial offers, is quite high. In essence, each organization is spinning its wheels in trying to target the same set of good customers, while paying insufficient attention to understanding the needs of the “not so good customers” of today, and converting them into good customers of tomorrow. A clear example of this mutual cannibalism of customers is the cellular phone industry, where each service provider is constantly trying to outdo the others. “Customer churn” is a well-accepted problem in this industry.

A well-accepted wisdom in the industry is that it costs five to seven times as much to acquire a new customer than to retain an existing one. The reason is that the organization already has the loyalty of existing customers, and all that is required for retention is to meet the customer’s expectations. For customer acquisition, however, the customer must be weaned away from another organization, which is a much harder task. As a result, it is crucial that the selection of customers to target is done with care, and that the right message be sent to each one. Given these needs, it becomes important for an organization to understand its customers well. Thus, one can consider customer relationship management to consist of two parts as follows:

$$\text{CRM} = \text{customer understanding} + \text{relationship management}$$

This equation is not new, because in the classic “neighborhood store” model of doing business, the store had a highly localized audience, and the store owner knew practically everyone in the neighborhood—making it easy for him to meet the needs of his customers. It is the big corporations, serving a mass customer base, that have difficulty in understanding the needs of individual customers. The realization of this gap of knowledge has been one of the driving factors for the rapid adoption of CRM software by many corporations. However, the initial deployment of CRM software has been for the second part of the CRM equation, namely “relationship management.” As described, relationship management efforts without an understanding of the customer can be marginally effective at best, and sometimes even counterproductive.

The approach that resolves this dilemma is the use of data analytics in CRM, with the goal of obtaining a better understanding of the needs of individual customers. Improved customer understanding drives better customer relationship management efforts, which leads to better and more frequent customer responses and in turn leads to more data collection about the customer—from which a more refined customer understanding can be gained. This positive feedback cycle—or “virtuous loop” as it is often called—is shown in Fig. 1.

Although this picture is very desirable, unfortunately several technical and organizational challenges must be overcome to achieve it. First, much customer data are collected for operational purposes and are not organized for ease of analysis. With the advance of data analysis techniques, it is becoming feasible to exploit these data for business management, such as to find existing trends and discover new opportunities. Second, it is critical that this knowledge cover all channels and customer touch points, so that the information base is complete and delivers a holistic and integrated view of each customer. This knowledge includes customer transactions, interactions, customer denials, service history, characteristics and profiles, interactive survey data, click-stream/browsing behavior, references, demographics, psychographics, and all available and useful data surrounding that customer. This information may also include data from outside the business as well, for example, from third-party data providers such as Experian or Axciom. Third, organizational thinking must be changed from the current focus on products to include both customers and products, as illustrated in Fig. 2. Successful adoption of CRM requires a change in focus by marketing from “who I can sell this products to?” to “what does this customer need?” It transforms marketing from tactical considerations, i.e., “how do I get this campaign out of the door” to strategic focus, i.e., “what campaigns will maximize customer value?”

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

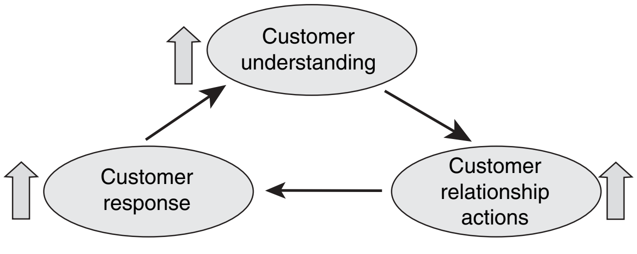


Figure 1. "Virtuous circle" of CRM.

The goal of this article is to introduce the data analytics opportunities that exist in customer relationship management, especially in the area of customer understanding. As the data collected about customers is becoming more complete, the time is ripe for the application of sophisticated data mining techniques towards better customer understanding. The rest of this paper is organized as follows: in Section 2 we introduce the concept of analytical customer relationship management. Section 3 briefly describes the underlying technologies and tools that are needed, namely data warehousing and data mining. Section 4 describes a number of organizational issues that are critical to successful deployment of CRM in an organization, and Section 5 concludes the paper.

ANALYTICAL CUSTOMER RELATIONSHIP MANAGEMENT

Significant resources have been spent on CRM, leading to the success of CRM software vendors such as Seibel, Oracle, and Epiphany. However, in the initial stages sufficient attention was not paid to analyzing customer data to target the CRM efforts. Simple heuristics and "gut-feel" approaches led to profitable customers being bombarded with offers (often turning them off), while there being little attempt to develop today's the "less valuable" customers into tomorrow's valuable ones. This lack of attention to customer needs is the cause of decreasing customer satisfaction across a wide variety of industries.[1]

Fortunately, however, the tremendous advancement in data management and analysis technologies is providing the opportunity to develop fine-grained customer understanding on a mass scale and to use it to better manage the relationship with each customer. It is this approach to developing customer understanding through data analysis, for the purpose of more effective relationship management, that we call *analytical customer relationship management (ACRM)*. ACRM can make the customer interaction functions of a company much more effective than they are currently.

Customer Segmentation

Customer segmentation is the division of the entire customer population into smaller groups, called *customer segments*. The key idea is that each segment is fairly homogeneous from a certain perspective, although not necessarily from other perspectives. Thus, the customer base is first segmented by the value they represent to an organization, and then by the needs they may have for specified products and services.

The purpose of segmentation is to identify groups of customers with similar needs and behavior patterns, so that they can be offered more tightly focused products, services, and communications. Segments should be identifiable, quantifiable, addressable, and of sufficient size to be worth addressing. For example, a vision products company may segment the customer population into those whose eyesight is perfect and those whose eyesight is not perfect. As far as the company is concerned, everyone whose eyesight is not perfect falls in the same segment, i.e., of potential customers, and hence, they are all the same. This segment is certainly not homogeneous from the perspective of a clothing manufacturer, who will perhaps segment on attributes like gender and age.

A company's customer data are organized into *customer profiles*. A customer's profile consists of three categories of data, namely (*1*) identity, (*2*) characteristics, and (*3*) behavior. These categories correspond to the following questions: *Who is the person? What attributes does he/she have? How does he/she behave?* Two types of segmentation can be performed based on the profile, namely

- Group customers based on common characteristics, and identify their common patterns of behavior.
- Group customers based on common patterns of behavior, and identify their common characteristics.

As shown in Fig. 3, each customer segment represents a different amount of profit per customer; the treatment of each segment can be different. The figure shows examples of the type of questions the company can ask about segments. Also included are some overall strategic questions about which segments to focus on, and how much.

Customer Communication

A key element of customer relationship management is communicating with the customer. This communication consists of two components, namely (*1*) deciding what message to send to each customer segment, and (*2*) selecting the channel through which the message must be sent. Message selection for each customer segment depends on the strategy being followed for that segment, as shown in Fig. 4. The selection of the communication channel depends on several characteristics of each channel, including cost, focus, attention, and impact.

Typical communication channels include television, radio, print media, direct mail, and e-mail. Television is a broadcast channel, which is very good at sending a common message to a very large population. Although it is very effective in building brand recognition, it is difficult to target a specific segment, as well as to measure response at the individual customer level. Radio, like television, is a broadcast medium, and hence, it is difficult to use for targeted communication to individual customers. Some television and radio stations, e.g., public radio and public television, develop a fairly accurate sample of their listener/

[1] Of course, customer expectation keeps rising over time, and the source of dissatisfaction today is very different than that of a few years ago. However, all organizations must constantly fight this battleindustries, as illustrated in Fig. .[2]

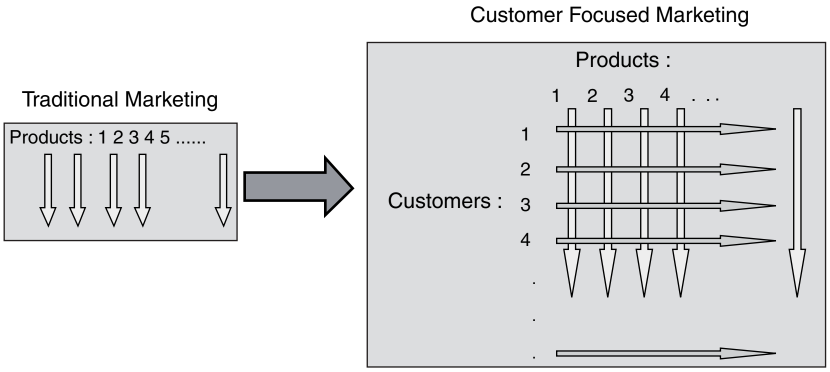


Figure 2. Change of focus from product only to customer + product.

viewer base through periodic fundraisers. Print media like newspapers and magazines can be used for much more focused communication, because the subscriber's profile is known. However, the readership of print media is usually much larger than the subscription base—a ratio of 1:3 in the United States—and hence, for a large part of the readership base, no profile is available. Direct mail is a communication channel that enables communicating with individual customers through personalized messages. In addition, it provides the ability of measuring response rates of customers at the individual level, because it enables the contacted customer to immediately respond to the message, if so desired. Finally, given its negligible cost, e-mail is becoming the medium of choice for customer contact for many organizations.

Figure 4, courtesy of Stevens and Hegarty (2), illustrates the problem of formulating the customer communication strategy. Each communication channel has its own characteristics in terms of cost, response rate, and attention. The goal of communication strategy optimization is to determine the (set of) communication channel(s) for each customer that minimizes cost or maximizes sale, profit, and so on. Although communication channel optimization has been a well-studied problem in the quantitative marketing literature, characteristics of new channels such as e-mail and the Web are not well understood. Thus, there is a need to revisit these problems.

Sending the message to each customer through the chosen communication channel is not enough. It is crucial to measure the impact of the communication. This measurement is done by using an approach called *response analysis*. As shown in Fig. 5, response analysis metrics, e.g., number of respondents, acquired customers, number of active customers, and number of profitable customers, can be calculated. These are analyzed to (*1*) determine how effective the overall customer communication campaign has been, (*2*) validate the goodness of customer segmentation, and (*3*) calibrate and refine the models of the various communication channels used. Although response analysis for traditional communication channels is fairly well understood, for new channels like e-mail and the Web, hardly anything is known. Understanding how customers relate to these new medium, which aspects they like and which they do not, and what are the right set of metrics to measure the usage of the medium, are all open questions and have attracted much research (3–5).

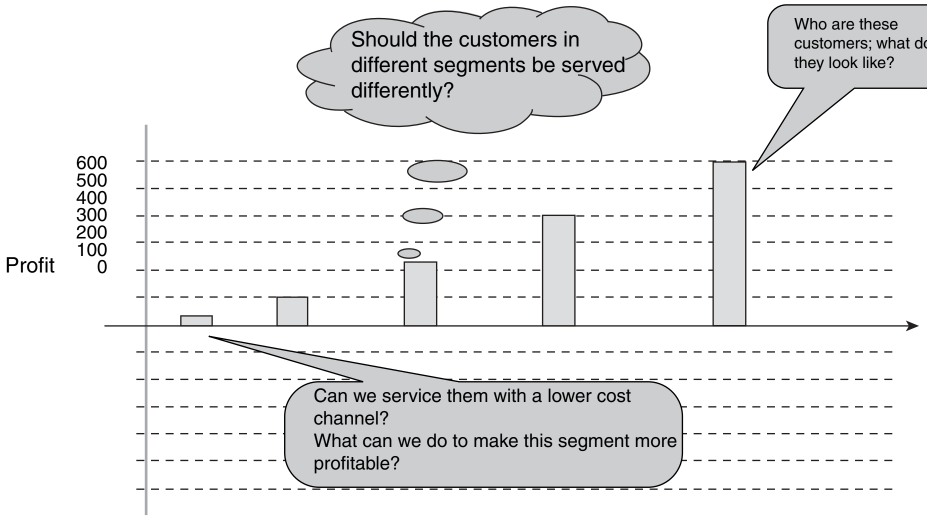


Figure 3. Segmentation of customers by profitability.

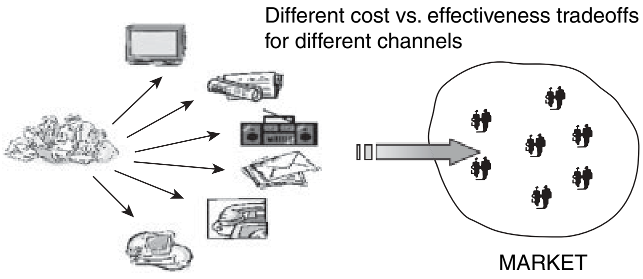


Figure 4. Formulating the optimal customer communication strategy.

Customer Retention

Customer retention is the effort carried out by a company to ensure that its customers do not switch over to the competition's products and services. A commonly accepted wisdom, which is acquired through substantial experience, is that it is five to seven times more expensive to acquire a new customer than to retain an existing one. Thus, it is of paramount importance to retain customers, especially highly profitable ones. A good loyal customer base that persists for a long time is one of the best advertisements for a business, creating an image of high quality. This image helps in attracting other customers who value long-term relationships and high-quality products and services.

Figure 6 shows how a company thinks of its various customer segments, from a current and future profitability perspective. Clearly, the quadrants on the right bottom and the right top should be targeted for retention. In addition, the right top customer quadrant must be targeted for strengthening the relationship, as there is significant unrealized potential.

A successful customer retention strategy for a company is to identify opportunities to meet the needs of the customer in a timely manner. A specific example is of a bank that used the event "ATM request for cash is rejected due to lack of funds" to offer unsecured personal loans to credit-worthy customers the next day. This offer was found to have a very high success rate, with the additional advantage of building customer loyalty. Classically, this analysis has been done at an aggregate level, namely for customer segments. Given current-day analytic tools, it should be possible to achieve it at the level of individual customers.

Customer Loyalty

From a company's perspective, a loyal customer is one who prefers the company's products and services to those of its competition. Loyalty can range from having a mild preference all the way to being a strong advocate for the company. It is well accepted in consumer marketing that an average customer who feels closer to a company (high loyalty) is significantly more profitable than one who feels less close (low loyalty). Thus, ideally a company would like all its customers to become loyal, and then to quickly advance up the loyalty chain.

Figure 7, courtesy of Heygate (1), illustrates the concept of tracking a customer to identify events in his/her life. Many of these events offer opportunities for strengthening the relationship the company has with this customer. For

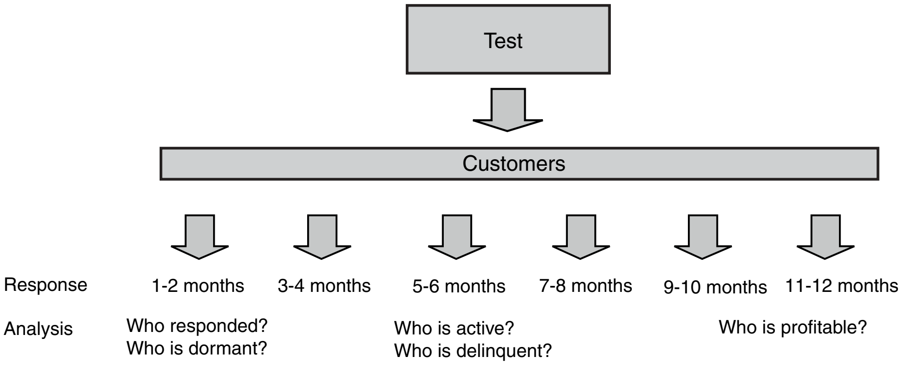


Figure 5. Analyzing the response to customer communications.

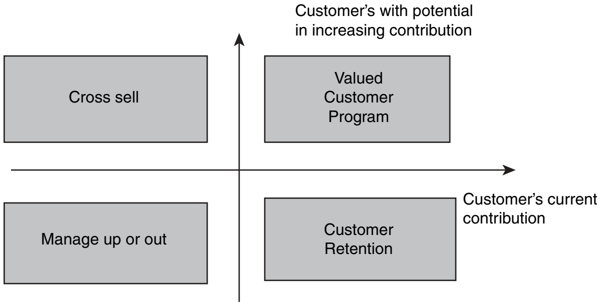


Figure 6. Treatment of various customer segments.

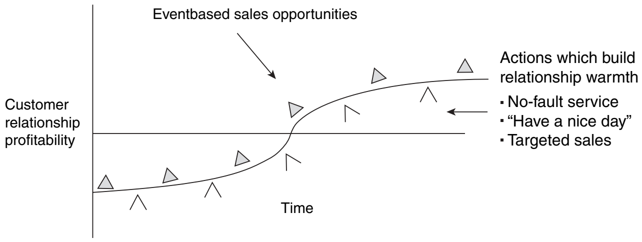


Figure 7. Lifetime impact of customer loyalty.

example, sending a greeting card on a customer's birthday is a valuable relationship-building action—with low cost and high effectiveness.

In marketing language, this strategy is called "event marketing," where the idea is to use the occurrence of events as marketing opportunities. Sometimes even negative events can be used to drive sales. For example, a bank adopted the policy of offering a personal loan to every customer whose check bounced or there were insufficient funds for ATM withdrawal. This program was very successful and enhanced the reputation of the bank as being really caring about its customers.

The data mining community has developed many techniques for event and episode identification from sequential data. There is a great opportunity for applying those techniques here, because recognizing a potential marketing event is the biggest problem here.

DATA ANALYTICS SUPPORT FOR ANALYTICAL CRM

In this section we describe the back-end support needed for analytical CRM. Specifically, we first outline a generic architecture and then focus on the two key components, namely data warehousing and data mining.

Data Analytics Architecture

Figure 8 shows an example architecture needed to support the data analytics needs of analytical CRM. The key components are the data warehouse and the data analysis tools and processes.

Data Warehouse

Building a data warehouse is a key stepping stone in getting started with analytical CRM. Data sources for the warehouse are often the operational systems, which provide the lowest level of data. Data sources are designed for operational use, not for decision support, and the data reflect this fact. Multiple data sources are often from different systems, running on a wide range of hardware, and much of this software is built in-house or highly customized. Data from multiple sources are mismatched. It is important to clean warehouse data because critical CRM decisions will be based on it. The three classes of data extraction tools commonly used are as follows: data migration that allows simple data transformation, data scrubbing that uses domain-specific knowledge to scrub data, and data auditing that discovers rules and relationships by scanning data and detects outliers.

Loading the warehouse includes some other processing tasks, such as checking integrity constraints, sorting, summarizing, and build indexes. Refreshing a warehouse requires propagating updates on source data to the data stored in the warehouse. The time and frequency to refresh a warehouse is determined by usage, types of data source, and so on. The ways to refresh the warehouse include data shipping, which uses triggers to update the snapshot log table and to propagate the updated data to the warehouse, and transaction shipping, which ships the updates in the transaction log. For technical details on transforming, refreshing, and maintaining data warehouses, see Refs. 6–13.

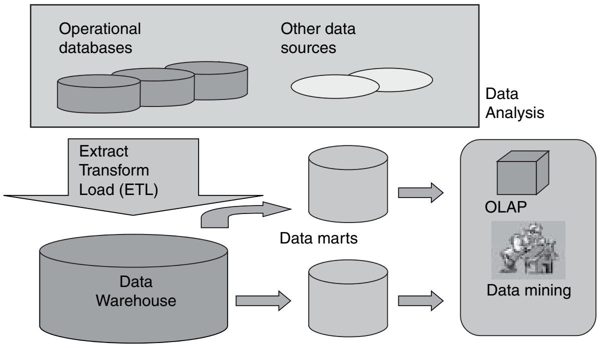


Figure 8. Data analytics architecture.

The key entities required for CRM include *Customer, Product, and Channel*. Usually information about each of these entities is scattered across multiple operational databases. In the warehouse these databases are consolidated into complete entities. For example, the Customer entity in the warehouse provides a full picture of who a customer is from the entire organization's perspective, including all possible interactions, as well as their histories. For smaller organizations the analysis may be done directly on the warehouse, whereas for larger organizations, separate data marts may be created for various CRM functions like customer segmentation, customer communication, and customer retention. A typical approach for designing entities in a data warehouse is called *dimensional modeling*, which organizes data into *fact tables* and *dimension tables*. Fact tables contain measurements or metrics of business processes and foreign keys of dimension tables. Dimension tables store the context of measurement, including the demographics of customers, time and place. A typical example is to store the content of each transaction such as transaction amount and products purchased in a fact table, while recording the characteristics of the associated customer, the channel, and the products in dimension tables. Interested readers are referred to Refs. 14 and 15, for modeling in data warehouse.

Data Mining

The next generation of analytic CRM requires companies to span the analytical spectrum and focus more effort on looking forward. The "what has happened" world of report writers and the "why has it happened" OLAP worlds are not sufficient. Time-to-market pressures, combined with data explosion, are forcing many organizations to struggle to stay competitive in the "less time, more data" scenario. Coupled with the need to be more proactive, organizations are focusing their analytical efforts to determine what will happen, what they can do to make it happen, and ultimately to automate the entire process. Data mining is now viewed today as an analytical necessity. The primary focus of data mining is to discover knowledge, previously unknown; predict future events; and automate the analysis of very large datasets.

The data mining process consists of several steps. First the data collected must be processed to make it mine-able. It requires several steps to clean the data and to handle mismatches in format, structure, semantics, and normalization and integration. A very good book on the subject is Ref. 16. Once the data have been cleaned, various data mining algorithms can be applied to extract models from it. Several data mining techniques have been developed, and the one to be applied depends on the specific purpose at hand. Reference 17 provides an excellent introduction to various data mining algorithms, whereas Ref. 18 shows how they can be applied in the context of marketing.

The most common data mining task is *classification*, which refers to assigning a new object one of the predefined classes by examining its features. The classification task is to learn a prediction model from a training set consisting of examples with features and preassigned classes. Such a model can be then used to predict the class of a new object. For example, one may classify customers as *good, okay*, and *bad* customers by using classification techniques based on their transaction records and provide incentives only to a handful of good customers for product/service promotion. Classification techniques can be extended to handle nondiscrete outcomes, e.g., estimating the lifetime value of a customer. Bayesian classifier (19,20), decision trees (21–24), decision rules (25,26), support vector machine (27,28), neural networks (29–31), and genetic algorithms (32,33) are among the most popular classification techniques.

Another common data mining task is *clustering*, which partitions a group of objects into a set of more homogeneous subgroups. Clustering is also called *unsupervised learning*, especially in the artificial intelligence community, as opposed to classification (also known as *supervised learning*) that requires domain experts assigning classes to examples in a training set. In clustering, objects are grouped based on pairwise similarities. It is up to the designer to determine the appropriate similarity metrics in a specific domain. Commonly used similarity metrics include reciprocal Euclidean distance, correlation coefficient, and cosine function. For example, customers can be segmented by their socioeconomical status, and different promotion packages can be tailor-made for different clusters of customers. Well-known clustering techniques include k-means, k-nearest neighbors (34,35), partitioning or hierarchical clustering (36–39), and expectation-maximization (EM) algorithm (40,41).

A common data mining task that was more recently proposed is *association mining*, which determines groups of closely related objects. A well-known example is the *market basket example* that identifies the set of products that often go together in customers' shopping carts; e.g., people who buy milk tend to also buy bread. Such association rules can be used to plan item placement on shelves and to design attractive packages. Association rule mining techniques can be extended to identify trends in time-variant data. For example, time series patterns derived from the transactions of former customers may help us better understand the ex-customer's behavior and subsequently retain (good) customers. Various techniques have been proposed to identify association rules (42–52) and sequential and time series patterns (53–65).

Once a model has been developed, it can be used for two kinds of purposes. The first purpose is to gain an understanding of the current behavior of the customers. A model used for this purpose is called a *descriptive model*. Both clustering and classification intend to find a descriptive model from existing data. The second purpose is to use the model to make predictions about the future behavior of the customers. A model used for this purpose is called a *predictive model*. The descriptive model, which is extracted from past behavior, is used as a starting point from which a predictive model can be built. Such an approach has been found to be quite successful, as it is based on the assumption that past behavior is a good predictor of future behavior, with appropriate adjustments. This assumption holds quite well in practice.

Finally, a data mining task that has been widely adopted by many online stores is *personalization*, which intends to provide information about products/services that fits a customer's personal need. A personalization system, often better known as a *recommendation system*, maintains an interest profile for each customer and recommends to a customer only those products/services that are highly related to his/her interest profile. Although traditional approaches require users to explicitly specify their interest profiles, modern recommendation systems adopt data mining techniques to build interest profiles from previous transactions. Two types of approaches have been proposed for building personal interest profiles: the *content-based approach* and the *collaborative approach*. The content-based approach derives the *content* features of a customer's past interaction data and builds a prediction model for each customer that, when given the content features of an item, indicates the probability that the customer likes the item. Various classification techniques have been applied in this context (66-70). The collaborative approach considers the *social* features of users' interests and recommends items to a customer by taking into account other customers' preferences. There are two broad categories of approaches for estimating the preference of an unseen item to a customer: *memory-based* and *model-based* approaches (71). The memory-based approach uses a rating matrix, with rows being customers and columns being items, to represent customers' ratings on items. It computes a weighted sum on rows or columns of the rating matrix for predicting the preference of a customer to an item (72–77). Possible weighting schemes include correlation, cosine and regression. The model-based approach builds a coherent prediction model for *all* customers based on customers' ratings. The prediction model takes as input a customer and an item and outputs the probability that the customer likes the item. Various techniques in classification (71,78–80), clustering (81–83), and association mining (84–88) have been proposed for building such a prediction model.

ORGANIZATIONAL ISSUES IN ANALYTICAL CRM ADOPTION

Although the promise of analytical CRM, both for cost reduction and revenue increase, is significant, this cannot be achieved unless there is successful adoption of it within an organization. Adaptation of an analytical CRM system within an organization is considered a business transformation project. In addition to deployment of new technology, it requires significant changes to existing processes that define how an enterprise should interface and treat all its customers and partners. In this section we describe some key organizational issues in CRM adoption.

Customer First Orientation

Companies that offer several products and services have traditionally organized their customer facing teams, e.g., sales, marketing, customer service, and so on, vertically along product lines, called "Lines of Business (LOBs)." The goal of any such product marketing team is to build the next product for its LOB; the goal of the sales team is to identify the customers who would be likely to buy this product, and customer service is trained to support the specifc product sets. Organizations adopting such a model, also known as functional-based organizations, causes customer needs and companies overall goals to be treated as secondary when compared with the LOB's priorities and needs. In such an organizational model, initiatives such as cross-selling become a big challenge because either the sales team does not have knowledge of other products that the company has to offer or does not have any financial incentive to offer products from other LOBs that customers may want to buy. The vertical organizational model also influences the enterprise architecture by having each LOB have its own customer database and associated business applications resulting in a complex enterprise architecture.

In addition to the architecture, a functional-based organizational model dictates what data about the customers should be collected. The collected data in such an organization are usually transactional in nature and less focused on customer's behavior and needs and make some analytical computations very difficult and less reliable, if not impossible. Additionally, it is possible for each LOB to have its own data about a given customer. The LOB-specific data about a customer can be inconsistent, resulting in a data inconsistency problem.

To overcome the above challenges and improve the chance of success in a CRM analytic project, the customer-focusing teams of an organization must be reoriented to make them focus on customers in addition to product lines while eliminating barriers caused by the vertical organizational model. These teams can be organized around well-defined customer segments, e.g., infants, children, teenagers, young professionals, and so on, and each given the charter of defining product design, marketing, sales, and service strategies that are geared to satisfying the needs of their customer segment. As part of this, some activities might be targeted to individual customers.

Attention to Data Aspects of Analytical CRM

The most sophisticated analytical tool can be rendered ineffective if the appropriate data are not available. To truly excel at CRM, an organization needs detailed information about the needs, values, and wants of its customers. Leading organizations gather data from many customer touch points and external sources, and bring these data together in a common, centralized repository, in a form that is available and ready to be analyzed when needed. This process helps ensure that the business has a consistent and accurate picture of every customer and can align its resources according to the highest priorities. Given this observation, it is critical that sufficient attention be paid to the data aspects of the CRM project, in addition to the software.

As discussed, most functional-based organizations lack that data required to make the analytical CRM project successful. The challenges are two-fold. The first challenge is when data (e.g., customer data) coming from each LOB need to be integrated into an enterprise-wide database to be used by ACRM applications. Differences in database schema and data content between LOBs may create data quality issues that can make the analytical CRM application less reliable. The second challenge is to adjust ongoing business processes to collect the necessary data and use the analytic CRM to focus on customer and companies priorities rather than an each LOB's priorities. In many cases, such adjustment requires changes in the way sales, marketing, and executive teams are compensated as well as how each LOB contribution to overall company is recognized.

Organizational "Buy In"

Although data mining and data warehousing are very powerful technologies, with a proven track record, there are also enough examples of failures when technology is deployed without sufficient organizational "buy in." As described, the parts of the organization that will benefit the most from analytical CRM are the business units, i.e., marketing and sales, and not the IT department. Thus, it is crucial to have "buy in" from the business units to ensure that the results will be used appropriately.

As described, deployment of analytical CRM is a company-wide business transformation project. It requires deployment of new technologies and transition from LOB-centric processes to company-centric processes. Example of such processes are how commissions are computed for sales, marketing, and executive teams as well as the way each LOB's contribution to the company is recognized. Transitioning from LOB-centric processes to company- centric processes requires some specific actions.

First, a CRM project needs to be owned and sponsored by a corporate executive from the business side who manages the business and technical development teams, and a cross-functional team who manages the scope of the project. The team is responsible to identify a road map for migration from the existing environment to the new one. As part of the road map, business and technology changes and impact to the different groups (i.e., business case) needs to be identified and communicated early in the project lifecycle to all affected units. It is important to make sure all impacted units have representation in the cross-functional team. The ideal (cross-functional) team should have enthusiastic members, who are committed, and are also viewed as leaders in their respective parts of the organization. This will make the dissemination of the successes much easier.

Second, all affected LOBs must support the road map and make commitments to adapt the new processes introduced by analytical CRM systems. Third, the new processes need to have an appropriate set of measurable metrics, to ensure that all steps for project success are being taken. Fourth, the project needs to be regularly reviewed by the executive team to ensure that any potential issue is addressed in a timely manner. Finally, incentives for performing well on the project should be included as part of the reward structure to ensure motivation.

Incremental Introduction of CRM

Introducing CRM into an organization must be managed carefully. Given its high initial cost, and significant change on the organization's processes, it is quite possible that insufficient care in its introduction leads to high expense, seemingly small early benefits, which can lead to low morale and excessive finger-pointing and possibility of failure.

As shown in Fig. 9, courtesy of Forsyth (89), it is better to have an incremental "pay-as-you-go" approach rather than a "field-of-dreams" approach. The benefits accrued from the first stage become evident and act as a catalyst for accelerating the subsequent stages. This makes the choice of the first project and its team very critical. Ideally, the project must be in a potentially high-impact area, where the current process is very ineffective. The process transformation and end-user training are two major success factors of a CRM project. The "pay-as-you-go" approach provides a way to transform the process and to train end users gradually. The gradual adaptation of a new system is an important factor in the success of any large project like CRM that impacts many aspects of a business.

Architectural Challenges

As discussed, in many companies, the enterprise architecture is a reflection of how LOBs operate. In a LOB-centric business model, each LOB has its own set of applications and databases and associate processes. To migrate to a single corporate-wide CRM system, a migration plan should be put in place to sunset LOB-centric systems and processes and move all existing data and interfaces to work in the new environment. This migration creates a set of special challenges, including scalability, support for company and LOB-specific functions, data integration, and performance. The architecture should also comply with any applicable corporate-wide standards.

Deployment Platform

In the recent decade, many companies have been dealing with the "buy" vs. "build" question for complex systems like CRM. Many companies decided to stop internal deployment of CRM systems and to license CRM applications from companies like Siebel, SAP, and PeopleSoft. However, deployment of these licensed applications has taken more time and cost more thanexpected. because their integration into the existing environment and process impacts have been greater than expected. Thus, it is important for the team to put a realistic plan together when starting a new CRM project.

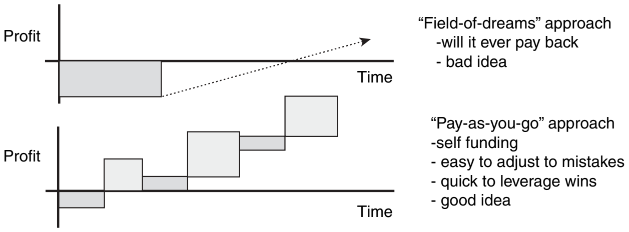


Figure 9. Incremental approach to CRM adoption.

Advancements in the application hosting and high cost of deployment and operation of CRM systems have created a market for hosting the CRM applications. In this business model, a company instead of having its CRM application will subscribe to a CRM application service. The application service provider is responsible for operation and maintenance of the environment, and the company pays a subscription fee. This new approach was pioneered by salesforce.com and has been followed by other CRM companies.

CONCLUSION

The Internet has emerged as a low-cost, low-latency, and high-bandwidth customer communication channel. In addition, its interactive nature provides an organization with the ability to enter into a close, personalized dialog with its individual customers. The simultaneous maturation of data management technologies like data warehousing, and analysis technologies like data mining, has created the ideal environment for making customer relationship management a much more systematic effort than it has been. Although there has been a significant growth of software vendors providing CRM software, and of using them, the focus so far has largely been on the "relationship management" part of CRM rather than on the "customer understanding" part. Thus, CRM functions such as e-mail-based campaigns management and online ads are being adopted quickly. However, ensuring that the right message is being delivered to the right person and that multiple messages being delivered at different times and through different channels are consistent is still in a nascent stage. As a result, companies are overcommunicating with their best customers, while insufficient attention is being paid to develop new ones into the best customers of the future.

In this article we have described how ACRM can fill the gap. Specifically, we described how data analytics can be used to make various CRM functions like customer segmentation, communication targeting, retention, and loyalty much more effective.

BIBLIOGRAPHY

1. R. Heygate, How to build valuable customer relationships, 2001. Available at: http://www.crm-forum.com/library/sophron/sophron-022/brandframe.html.
2. P. Stevens and J. Hegarty, CRM and Brand Management - do they fit together?, 1999. Available at: http://www.crm-forum.com/library/sophron/sophron-002/brandframe.html.
3. J. P. Benway and D. M. Lane, Banner blindness: web searchers often miss obvious links, *Internetworking*, **1**(3), 1998, availableat http://www.internettg.org/newsletter/dec98/banner_blindness.html.
4. T. L. Cheyne and F. E. Ritter, Targeting audiences on the Internet, *Comm. ACM*, **44**(4): 94–98, 2001.
5. R. D. Gopel, G. Walter, and A. K. Tripathi, Amediation: a new horizons in effective email advertise, *Comm. ACM*, **44**(12): 91–96, 2001.
6. Y. Cui and J. Widom, Lineage tracing for general data warehouse transformations, *Proc. of International Conference on Very Large Databases (VLDB)*, 2001, pp. 471–480.
7. Y. Cui and J. Widom, Practical lineage tracing in data warehouses, *16th International Conference on Data Engineering* (ICDE), 2000, pp. 367–378.
8. P. Vassiliadis, M. Bouzeghoub, and C. Quix, Towards quality-oriented data warehouse usage and evolution, *Proc. of the 11th Conference on Advanced Information Systems Engineering (CAiSE)*, 1999, pp. 164–179.
9. C. Hurtado, A. Mendelzon, and A. Vaisman, Updating OLAP dimensions, *Proc. of ACM DOLAP*, 1999, pp. 60–66.
10. H. Garcia-Molina, W. Labio, and J. Yang, Expiring data in a warehouse, *Proc. of International Conference on Very Large Databases (VLDB)*, 1998, pp. 500–511.
11. H. V. Jagadish, P. P. S. Narayan, S. Seshadri, S. Sudarshan, and R. Kanneganti, Incremental organization for data recording and warehousing, *Proc. of International Conference on Very Large Databases (VLDB)*, 1997, pp. 16–25.
12. P. Scheuermann, J. Shim, and R. Vingralek, Watchman: a data warehouse intelligent cache manager, *Proc. of International Conference on Very Large Databases (VLDB)*, 1996, pp. 51–62.
13. Y. Zhuge, H. Garcia-Molina, and J. L. Wiener, The strobe algorithms for multi-source warehouse consistency, *International Conf. on Parallel and Distributed Information Systems (PDIS)*, 1996, pp. 146–157.
14. W. Inmon, *Building the Data Warehouse*, 3rd ed. New York: John Wiley & Sons, Inc., 2002.
15. R. Kimball, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, 2nd ed. New York: John Wiley & Sons, Inc., 2002.
16. D. Pyle, *Data Preparation for Data Mining*, San Francisco, CA: Morgan Kaufmann Publishers, 1999.
17. D. J. Hand, H. Mannila, and P. Smythe, *Principles of Data Mining*, Cambridge, MA: MIT Press, 2000.
18. O. C. Rud, *Data Mining Cookbook: Modeling Data for Marketing, Risk and Customer Relationship Management*, New York: John Wiley and Sons, 2000.
19. P. Domingos and M. Pazzani, On the optimality of the simple Bayesian classifier under zero-one loss, *Mach. Learning*, **29** (2–3): 103–130, 1997.
20. R. Kohavi, Scaling up the accuracy of naïve-Bayes classifiers: A decision-tree hybrid, *Proc. the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 202–207.
21. J. R. Quinlan, Induction to decision trees, *Mach. Learning*, **1**(1): 81–106, 1986.
22. J. R. Quinlan, *C4.5: Programs for Machine Learning*, San Mateo, CA: Morgan Kaufman, 1993.
23. S. K. Murthy, Automatic construction of decision trees from data: A multi-disciplinary survey, *Data Mining Knowledge Disc.*, **2**(4): 345–389, 1998.
24. V. Ganti, J. Gehrke, and R. Ramakrishnam, Mining very large databases, *IEEE Computer*, **32**: 38–45, 1999.
25. P. Clark and T. Niblett, The CN2 induction algorithm, *Machine Learning*, **3**(4): 261–283, 1989.
26. P. Clark and R. Boswell,. Rule induction with CN2: Some recent improvements, *Proc. Fifth European Working Session on Learning*, 1991, pp. 151–163.
27. B. Scholkopf, C. J. C. Burges, and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond,* Cambridge, MA: MIT Press, 2001.
28. N. Cristianini, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge, UK: Cambridge University Press, 2000.
29. L. V. Fausett, *Fundamentals of Neural Networks*, Upper Saddle River, N.J.: Prentice Hall, 1994.
30. K. Smith and J. Gupta, *Neural Networks in Business: Techniques and Applications*, Hershey, PA: Idea Group Publishing, 2002.
31. G. P. Zhang, *Neural Networks in Business Forecasting*, Hershey, PA: Information Science Publishing, 2003.
32. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.
33. M. Mitchell, *An Introduction to Genetic Algorithms*, Cambridge, MA: MIT Press, 1998.
34. B. V. Dasartyed, *Nearest Neighbor: Pattern Classification Techniques.* IEEE Computer Society, 1991.
35. T. Hastie and R. J. Tibshirani, Discriminant adaptive nearest neighbor classification, *IEEE Trans. Pattern Anal. Mach. Intell.*, **18**(6): 607–612, 1996.
36. M. J. Berry and G. Linoff, *Data Mining Techniques: for Marketing, Sales, and Customer Support*, New York: John Wiley & Sons, 1997.
37. R. T. Ng and J. Han, Efficient and effective clustering methods for spatial data mining, *Proc. of Int'l Conf. on VLDB*, 1994.
38. L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, New York: John Wiley & Sons, 1990.
39. A. K. Jain and R. C. Dubes, *Algorithms for clustering data*, Upper Saddle River, N.J.: Prentice-Hall, 1988.
40. A. P. Dempster, N. M. Laird, and D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, *J. Royal Statistical Soc.*, **39**: 1–38, 1977.
41. G. J. McLachlan and T. Krishnam, *The EM Algorithm and Extensions*, New York: Wiley and Sons, 1998.
42. R. Agrawal, T. Imielinski, and A. Swami, Mining associations between sets of items in massive databases, *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Washington D.C., 1993, pp. 207–216.
43. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, Fast discovery of association rules, *Advances in Knowledge Discovery and Data Mining*, Chapter 12. Cambridge, MA: AAAI/MIT Press, 1995, pp. 307–328.

44. R. Agrawal and R. Srikant, Fast algorithms for mining association rules, *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, 1994, pp. 487–499.
45. R. Srikant, Q. Vu, and R. Agrawal, Mining association rules with item constraints, *Proc. of the 3rd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Newport Beach, CA, 1997.
46. R. Srikant and R. Agrawal, Mining generalized association rules, *Proc. of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, 1995.
47. J. S. Park, M. Chen, and P. S. Yu, Using a hash-based method with transaction trimming for mining association rules, *IEEE Trans. Knowledge Data Engin.*, **9**(5): 813–825, 1997.
48. M. Zaki, S. Parthasarathy, M. Ogihara, and Wei Li, New algorithms for fast discovery of association rules, *3rd International Conference on Knowledge Discovery and Data Mining (KDD'97)*, Newport Beach, CA, 1997, pp. 283–286.
49. R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang, Exploratory mining and pruning optimizations of constrained associations rules, *Proc. of1998 ACM-SIGMOD Conf. on Management of Data*, Seattle, WA, 1998. Available at http://db.cs.sfu.ca/sections/publication/kdd/kdd.html.
50. S. Brin, R. Motwani, and C. Silverstein, Beyond narket baskets: Generalizing association rules to correlations, *Proc. 1997 ACM SIGMOD*, Montreal, Canada, 1997.
51. C. Bettini, X. S. Wang, S. Jajodia, and J. Lin, Discovering frequent event patterns with multiple granularities in time sequences, *IEEE Transactions on Knowledge and Data Engineering*, **10**(2): 222–237, 1998.
52. J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation, *Proc. of Int. Conf. on Management of Data*, Dallas, TX, 2000.
53. H. Mannila, H. Toivonen, and A. Inkeri Verkamo, Discovery of frequent episodes in event sequences, Technical Report C-1997-15, Department of Computer Science, University of Helsinki, Finland, 1997.
54. R. Agrawal and R. Srikant, Mining sequential patterns, *Proc. of the Int'l Conference on Data Engineering (ICDE)*, Taipei, Taiwan, 1995, pp. 3–14.
55. R. Srikant and R. Agrawal, Mining sequential patterns: Generalizations and improvements, *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, Avignon, France, 1996, pp. 3–17.
56. G. Berger and A. Tuzhilin, Discovering unexpected patterns in temporal data using temporal logic, in O. Etzion, S. Jajodia, and S. Sripada (eds.), *Temporal Databases: Res. Prac.*, Berlin: Springer-Verlag, 1998, pp. 281–309.
57. B. Padmanabhan and A. Tuzhilin, A belief-driven method for discovering unexpected patterns, *Proc. of the 4rd International Conference on Knowledge Discovery and Data Mining (KDD-98)*, 1998, pp. 94–110.
58. M. Zaki, Efficient enumeration of frequent sequences, *7th International Conference on Information and Knowledge Management*, Washington DC, 1998, pp. 68–75.
59. K. Wang, Discovering patterns from large and dynamic sequential data, *J. Intell. Inf. Syst.*, **9**(1): 33–56, 1997.
60. V. Guralnik, D. Wrjesekera, and J. Srivastava, Pattern directed mining for frequent episodes, *Proc. of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York, 1998, pp. 51–57.
61. A. C. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter*, Cambridge, UK: Cambridge University Press, 1989.
62. G. Dong and J. Li, Efficient mining of emerging patterns: Discovering trends and differences, *Proc. SIGKDD*, 1999, pp. 43–52.
63. B. Liu, W. Hsu, and Y. Ma, Mining association rules with multiple minimum supports, *Proc. of the ACM SIGKDD Int'l. Conf. on Knowledge Discovery and Data Mining*, 1999, pp. 337–341.
64. J. Han, G. Dong, and Y. Yin, Efficient mining of partial periodic patterns in time series database, *Proc. ICDE*, 1999, pp. 106–115.
65. B. Ozden, S. Ramaswamy, and A. Silberschatz, Cyclic association rules, *Proc. ICDE*, 1998, pp. 412–421.
66. R. Mooney and L. Roy, Content-based book recommending using learning for text categorization, *Proc. of the ACM Conf. on Digital Libraries*, 2000, pp. 195–204.
67. M. Pazzani, J. Muramatsu, and D. Billsus, Syskill & Webert: Identifying interesting Web sites, *Proc. of the Nat. Conf. Artif. Intell.*, **13**(5–6): 54–61, 1996.
68. J. Rucker and M. J. Polanco, Siteseer: personalized navigation for the Web, *Comm. ACM*, **35**(12): 73–75, 1992.
69. B. Krulwich and C. Burkey, The infoFinder agent: Learning user interests through heuristic phrase extraction, *IEEE Expert*, **12**(5): 22–27, 1997.
70. K. Lang, Newsweeder: learning to filter Netnews, *Proc. of the 12th Intl. Conf. on Machine Learning*, 1995, pp. 331–339.
71. J. Breese, D. Heckerman, and C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 43–52.
72. U. Shardanand and P. Maes, Social information filtering: Algorithms for automating word of mouth, *Proc. of the Conference on Human Factors in Computing Systems*, 1995, pp. 210–217.
73. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, GroupLens: An open architecture for collaborative filtering of netnews, *Proc. of the ACM Conference on Computer Supported Cooperative Work*, 1994, pp. 175–186.
74. J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, GroupLens: applying collaborative filtering to Usenet news, *Comm. ACM*, **40**(3): 77–87, 1997.
75. J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, An algorithmic framework for performing collaborative filtering, *Proc. of the 1999 Conference on Research and Development in Information Retrieval*, 1999, pp. 230–237.
76. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, Analysis of recommendation algorithms for e-Commerce, *Proc. of the 2'nd Conference on Electronic Commerce*, 2000, pp. 158–167.
77. B. Sarwar, G. Karypis, J. Konstan, and J. Reidl, Item-based collaborative filtering recommendation algorithms, *Proc. of the Tenth International World Wide Web Conference on World Wide Web*, 2001, pp. 285–295.
78. A. Ansari, S. Essegaier, and R. Kohli, Internet recommendation systems, *J. Market. Res.*, **37**(3): 67–85, 2000.
79. D. Billsus and M. Pazzani, Learning collaborative information filters, in J. Shavlik (ed.), *Machine learning: Proc. of the Fifteenth International Conference*, San Francisco, CA: Morgan Kaufmann, 1998, pp. 46–54.
80. D. Pennock, E. Horvitz, S. Lawrence, and C. Giles, Collaborative filtering by personality diagnosis: a hybrid memory- and model-based approach, *Proc. of the Conf. on Uncertainty in Artificial Intelligence*, 2000, pp. 473–480.

81. T. W. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal, From user access patterns to dynamic hypertext linking, *Comp. Networks*, **28**(7–11): 1007–1014, 1996.

82. B. Mobasher, R. Cooley, and J. Srivastava, Creating adaptive Web sites through usage-based clustering of URLs, *Proc. of the IEEE Knowledge and Data Engineering Exchange Workshop*, 1999, pp. 19–25.

83. J. Srivastava, R. Cooley, M. Deshpande, and P. Tang, Web usage mining: discovery and applications of usage patterns from Web data, *SIGKDD Explorations*, **1**(2): 12–23, 2000.

84. W. Lin, S. A. Alvarez, and C. Ruiz, Collaborative recommendation via adaptive association rule mining, *Proc. of the WebKDD Workshop*, Boston, MA, 2000.

85. B. Mobasher, H. Dai, T. Luo, M. Nakagawa, and J. Witshire, Discovery of aggregate usage profiles for Web personalization, *Proc. of the WebKDD Workshop*, 2000.

86. J. Herlocker and J. Konstan, Content-independent task-focused recommendation, *IEEE Internet Comput.*, **5**(6): 40–47, 2001.

87. J. Pitkow and P. Pirolli, Mining longest repeating subsequences to predict World Wide Web surfing, *Proc. of the USENIX Symposium on Internet Technologies and Systems*, 1999, pp. 139–150.

88. M. Deshpande and G. Karypis, Selective markov models for predicting Web-page accesses, *Proc. of the First International SIAM Conference on Data Mining*, 2001. Available at: http://www.siam.org/meetings/sdm01/pdf/sdm0l_04.pdf.

89. R. Forsyth, Avoiding Post-Implementation Blues Managing the Skills, 2000. Available at: http://www.crm-forum.com/library/pre/pre-025/brandframe.html.

JAIDEEP SRIVASTAVA
University of Minnesota,
Minneapolis, Minnesota
JAMSHID A. VAYGHAN
IBM Corporation
Rochester, Minnesota
EE-PENG LIM
Nanyang Technological,
University
Singapore
SAN-YIH HWANG
National Sun Yat-sen
University
Kaohsiung, Taiwan
JAU-HWANG WANG
Central Police University,
Taoyuan, Taiwan

ASPECT-ORIENTED SOFTWARE DEVELOPMENT: AN INTRODUCTION

INTRODUCTION

Software development is a challenging task because of the complexity, heterogeneity, and distributed nature of modern software systems and the interactions and dependencies that need to be managed therein. An established approach to solving complex problems is by breaking them into smaller, more manageable, and easily understandable parts, for example, modules. After the subproblems are solved, all the solutions are combined to provide a solution to the initial complex problem. In software engineering, the same division and combination principles are applied: the division into subproblems is termed as *modularization*, and the combination of solutions is termed *composition*.

Modularity of software has long been the primary tool used in software development and a subject for computer science research. Fundamental software-structuring mechanisms such as procedures, functions, objects, and components are essential modularization mechanisms used in contemporary software-development paradigms. Each of these modules is intended to represent a clearly identified part of the software, and altogether they interact to provide the complete solution expected from the software application. This practice improves the individual module's understandability and facilitates its change, when needed. This practice is what has been referred to as the principle of *separation of concerns* by Dijkstra in his seminal book *A Discipline of Programming* (1).

A *concern* in software development is an interest, that pertains to the system's development, its operation, or any other matters that are critical or otherwise important to a stakeholder (2). The principle of separation of concerns thus states that each concern that is relevant to the software application is best treated separately. For instance, in *object-orientation* (one of today's most widely used development paradigms) separation of concerns is achieved by decomposing an application into individual real-life or conceptual objects and exploiting the object-oriented software-development principles (i.e., encapsulation, polymorphism, inheritance, and delegation) for application development. In an ideal situation, each object pertains to a single concern. Nevertheless, despite the wide support for separation of concerns in modern software-development paradigms (such as object-orientation), concerns still exist that are difficult or even impossible to modularize by means of the established mechanisms. For instance, such concerns as synchronization policies in multithreaded systems, logging, persistence, error handling, real-time constraints, and faulttolerance are often intermixed with (or, in other words, spread across) several other modules in the system. Such concerns are said to crosscut the other modules and, therefore, are called *crosscutting concerns*.

A practical example of the extent of crosscutting in a system is provided by Kiczales et al. in Ref. (3) and is reproduced in Fig. 1. This figure shows the modules (object-oriented classes) of the Apache Tomcat Web server represented as vertical bars, whereas the implementation of the *logging* concern is indicated by the horizontal lines (which represent lines of logging code). The logging concern clearly crosscuts many distinct modules.

Crosscutting concerns, such as logging in Fig. 1, complicate software development because they do not fit the primary decomposition of the development paradigm (the object boundaries in Fig. 1). As a result, the responsibilities of the individual modules are not clearly demarcated. Not only is an object responsible for its core state and behavior (pertaining to its business functionality, for instance) but also for that of the crosscutting concern (such as logging). This phenomenon is referred to as *tangling*. At the same time, we observe the *scattering* effect whereby a given crosscutting concern, for example, logging, is not modularized in a single element and hence hampers our ability to reason about its effect and behavior with respect to the other modules; such reasoning requires inspection of all the modules where logging is realized. In practice, this straightforwardly translates into an inability to reuse the implementation of a crosscutting concern, without resorting to code duplication. This practice goes against the established best practice of separation of concerns that we highlighted above because the crosscutting effect results in software that is difficult to understand and maintain. Addressing this problem of crosscutting is the main focus of *aspect-oriented software-development* (AOSD) techniques.

AOSD techniques provide systematic means for identification, modularization, and composition of concerns that normally would be crosscutting in other development paradigms (e.g., object-oriented or procedural paradigms) (4). With AOSD, such a concern will continue to exert a broad, crosscutting influence on other concerns, but its representation and realization can be localized in a single module.

AOSD CONCEPTS

AOSD techniques provide systematic means for identification, modularization, and composition of crosscutting concerns throughout software development lifecycle. Modules in other contemporary paradigms are all rooted in functional decomposition (e.g., subroutines, functions, objects, and components). To characterize these modules, Kiczales et al. (5) coined the term *generalized procedure* and argued that such generalized procedures are to be invoked explicitly via a direct call "by name" at the *syntactic location* in the software artifact where their execution is desired.

For instance, consider the code fragment in Fig. 2, taken from the implementation of a concurrent buffer. Two major concerns exist in this implementation: buffer functionality and concurrency locks. The concurrency-locking concern is

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

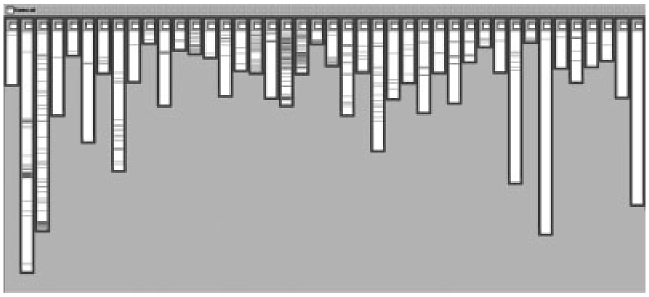
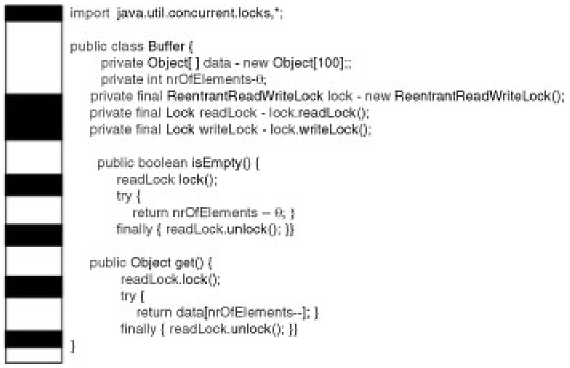


Figure 1. An example of a crosscutting concern (logging) in a software application [3].



```
import java.util.concurrent.locks.*;

public class Buffer {
        private Object[ ] data - new Object[100];;
        private int nrOfElements-0;
      private final ReentrantReadWriteLock lock - new ReentrantReadWriteLock();
      private final Lock readLock - lock.readLock();
      private final Lock writeLock - lock.writeLock();

       public boolean isEmpty() {
          readLock lock();
          try {
             return nrOfElements – 0; }
          finally { readLock.unlock(); }}

      public Object get() {
           readLock.lock();
           try {
              return data[nrOfElements--]; }
           finally { readLock.unlock(); }}
}
```

Figure 2. A fragment of a concurrent buffer in Java.

a crosscutting concern because it crosscuts the implementation of each method of the buffer, as illustrated by the black lines on the left-hand side of the figure. To invoke the locking/unlocking functionality, the *lock()* and *unlock()* methods must be called by name at the syntactic locations where their functionality is required. Furthermore, the buffer object must hold the functional buffer state as well as the locking semaphores. Aspect-orientation proposes a fundamentally new kind of modularization and composition that goes beyond generalized procedures. The elements of AOSD supporting this new modularization and its composition techniques are presented next.

Aspect

A crosscutting concern is encapsulated within a dedicated module called an *aspect;* an *aspect* is a modularized crosscutting concern. The specification of an aspect's interaction with the other modules is its primary distinction with respect to modules based on the concept of generalized procedures. For aspects in the implementation of a software artifact, this means that, in contrast to the explicit invocation of a generalized procedure, an aspect's invocation is not bound to such an explicit, syntactically located invocation statement. Instead, an explicit specification about when an aspect will be applied can be either attached to an aspect (e.g., as in Ref. 6) or provided in separate dedicated *composition specifications* (e.g., Refs. 7 and 8). For aspects in requirements, architecture, and design, this, distinction means reasoning about the crosscutting concerns and about their dependencies and interactions with other concerns without having to inspect multiple modules.

The aspect's invocation mechanism has also been referred to as *implicit invocation* by Xu et al. (9) and was uniformly characterized with respect to generalized procedures by Kiczales and Mezini in Ref. (10): *"one useful way to uniformly characterize the different mechanisms is as establishing different kinds of bindings along a path from points in a program to the implementation of an operation that must execute at those points."* Whereas procedures are one way to introduce bindings from a point in a program (i.e., where the procedure call resides) to their implementation, pointcuts and advice are a means to introduce additional points and bindings. The points to which advices can be bound are termed joinpoints, and the bindings are specified by pointcuts. These concepts are described next.

Joinpoint and Joinpoint Model

In all aspect-oriented techniques, aspects can only be invoked, or composed with other modules, at some well-defined and principled points within the software artifacts. These points are referred to as *joinpoints*. All possible kinds of joinpoints for the given AOSD approach are described in a *joinpoint model*. As stated in Ref. 2: "A joinpoint is a point of interest in some artifact in the software lifecycle through which two or more concerns may be composed. A joinpoint model defines the kinds of joinpoints available and how they are accessed and used." For example, in most aspect-oriented programming languages, the possible kinds of joinpoints are method calls, field references and assignments, exception handler executions, method executions, and so forth. In requirements, joinpoints can be specific requirements or concerns encapsulating those requirements. In design, elements of a UML diagram, for instance, can serve as joinpoints.

Pointcut

The set of joinpoints at which a given aspect should interact with some other modules is specified via a *pointcut*. Pointcuts can be defined *by extension*, for example, enumerating each joinpoint relevant for the given aspect application, or *by intension*, for example, via some kind of more abstract joinpoint selection criteria. These criteria typically are properties of the joinpoints to be selected (e.g. name patterns, structural or dynamic relations with other joinpoints, etc.). Because aspects normally have a broad influence on the system, affecting a number of other concerns, defining their interactions by enumerating joinpoints is rather inefficient. Consequently, in AOSD, pointcuts are normally defined by intension. Thus, a pointcut is a predicate that matches joinpoints.

Advice

As discussed above, an aspect affects a set of other concerns at the joinpoints. In AOSD terminology, it is said that aspects *advise* other concerns. An *advice* represents the semantics of an aspect that will interact with some other concern at a joinpoint. Depending on the AO approach used and the lifecycle stage of aspect modeling, the advice can take a variety of forms. For instance, in Fig. 2, the *lock()* and *unlock()* operations of the synchronization concern will be contained in code-level advices that impose additional behavior on the buffer object (see the section titled "Concurrent Buffer Example in AOP").

Traditionally, an *advice* in AOSD implies a behavior-related interaction between aspectual and nonaspectual artifacts. Such an interaction is also defined in respect with some temporal, conditional, or unconditional order. For instance, the behavioral modification can be applied by an advice *before, after, instead of*, or *concurrently* with the original behavior of the advised artifact.

Intertype Declarations

Although an advice normally modifies behavior at the advised joinpoints, some AOSD approaches use an additional mechanism called *inter-type declarations* (or *introductions*) for directly modifying the structure of the original artifacts. For instance, an intertype declaration may insert a new requirement into a viewpoint or a new variable declaration into a class or even change subtype relations, and so forth.

Composition/Weaving

In AOSD, traditionally the *composition* or *weaving* is the integration of the separated crosscutting elements back into the modules crosscut by them. For instance, during composition the *lock()* advice of a concurrency aspect would be inserted into the *buffer* class and would produce the same result as shown in Fig. 2. In the case of implementation-level aspects, such weaving is often part of the compilation to executable code, but load-time and run-time weaving is also possible in many aspect languages and frameworks.

Recently, some AOSD works have pointed out that, especially in the case of nonimplementation artifacts, it is not always necessary to integrate the aspectual elements in other modules; often a *composition specification* is sufficient for reasoning about aspectual and nonaspectual module interactions (8, 11, 12). The *composition specification* defines which aspectual elements (advice, intertype declarations, and so forth.) affect which joinpoints (selected by pointcuts) of which nonaspectual modules and defines what are the temporal, conditional, or unconditional circumstances of aspect invocation.

ASPECT-ORIENTED SOFTWARE-DEVELOPMENT LIFECYCLE

Aspect-oriented software-engineering techniques have the same objectives as any other software-engineering techniques: to understand, represent, and realize software systems. The unique contribution of AOSD is in providing additional abstraction and reasoning capabilities via the constructs discussed above. These constructs assist in modularizing concerns that would otherwise be scattered (i.e., included piece by piece) in other concern modules. Such concern scattering also results in the entanglement of elements of independent concerns in the same module. By using aspects such scattering and tangling issues are removed, which in turn leads to simpler and more-cohesive modules for concern representation.

AOSD does not enforce any particular sequence or lifecycle activities; individuals choosing to use this technology may continue to use their own established activities along with the additional modularization and reasoning mechanisms of AOSD. Nevertheless, a number of AOSD approaches currently strive to provide a whole AO development process covering the set of activities that leads from requirements to implementation. For instance, the AOSD with Use-cases approach (13) provides support for requirement modeling via use cases, their refinement into design, and guidelines for mapping these designs onto an AO implementation language. Similarly the Theme (14, 15) approach consists of Theme\Doc part for requirements analysis and Theme\UML part for design representation of the aspects from Theme\Doc. Detailed guidelines then are provided (16) for realizing Theme designs in the AspectJ and Hyper/J AO programming languages.

However, being a relatively young area, AOSD has not yet consolidated its development activities into an established set, as has, for instance, object-oriented development. Thus, several analysis, architecture, design, and implementation approaches exist that are not necessarily linked into an end-to-end development process. In the rest of this article, we discuss a subset of current AOSD approaches, without focusing on one specific approach that covers the end-to-end software development process. This choice is motivated by the goal of outlining the diverse and varied approaches that constitute AOSD rather than detailing a specific approach. Readers interested in any particular approach are pointed to further sources via relevant references. At the same time, we detail a specific requirements, architecture, design, and implementation approach to outline how the AOSD concepts fit with the established lifecycle artifacts. Because it is not possible to cover all facades of AOSD in this introduction, we also provide additional references to related topics, where appropriate.

ASPECT-ORIENTED REQUIREMENTS ENGINEERING

Crosscutting concerns, such as logging, distribution, and fault tolerance, first manifest themselves in requirements, and this is where one must start treating them. Aspect-oriented requirements-engineering (AORE) approaches aim to facilitate identification and analysis of such crosscutting concerns during requirements engineering to understand their potential effects and tradeoffs with respect to other stakeholder requirements. Often AORE approaches extend existing requirements-engineering techniques with additional support for modularization and composition of crosscutting concerns. Such support is missing in most contemporary requirements-engineering techniques. For instance, in the classical use-cases approach (17), nonfunctional requirements (NFR) cannot be readily modeled. Although techniques such as goal-based approaches

(18, 19) support modularization and analysis of such NFRs, they lack effective composition mechanisms to explore the complex dependencies and interactions fully, as such an analysis entails. Thus, AORE focuses on providing systematic means for modularization and composition of crosscutting concerns in requirements.

Aspects in AORE

Presently, a number of AORE approaches are researched actively (4, 8, 11, 13, 20). Some provide an additional modularization unit to represent requirements-level aspects (e.g., Refs. 4, 13, and 20), whereas others strive to use uniform modules for both aspectual and nonaspectual requirements artifacts (e.g., Refs. 8 and 11). In both cases, a requirements-level aspect is an artifact that encapsulates a concern that has a broad influence on the requirements in several other requirements-level artifacts. A summary of key concepts and techniques for AORE is provided by Baniassad et al. in Ref. 21, whereas a detailed comparative analysis between AORE and other contemporary requirements-engineering techniques is provided in the survey by Chitchyan et al. (22).

One may note that some (although not many) established non-AO requirements approaches already provide systematic modularization support for crosscutting concerns. For instance, the goal-based approaches (18, 19) support modularization of both functional and nonfunctional goals. However, the main distinction between the AO and non-AO requirements approaches is in the support for composition in the former. Contemporary (non-aspect-oriented) RE approaches hardly provide any composition support, with compositions often entangled with the requirement artifact development process. For instance, the composition of goals and softgoals is an integral part of goal interdependency graph development; goal compositions cannot be represented or reasoned about in isolation from the whole set of goals in the graph. In AORE, the compositions are normally represented in separate dedicated composition modules to reason about the dependencies and interactions exerted by crosscutting requirements rather than attached to the aspectual artifacts.

In the rest of this section, we will use the AORE with Arcade approach (4) to illustrate the AORE concepts. AORE with Arcade builds on the viewpoints-based requirements-engineering approach (23, 24) and provides a dedicated module for aspect representation. An aspectual module here contains requirements that affect requirements of several viewpoints. For instance, an aspect may contain requirements about security, concurrency, and so forth. As common in all RE approaches, this viewpoint-based approach provides a unique identifier for each concern (via its name) and each of its distinct requirements (via number-based ids unique within the scope of each concern).

Compositions in AORE with Arcade are defined in dedicated composition modules. The joinpoint model exposes unique identifiers of requirements and viewpoint/aspect names. The pointcuts are defined by extension, which enumerates the module name and id pairs for each requirement that participates in a composition. A pointcut can also use a *wildcard*, for example, a universal quantifier, such as *all*. An example of a pointcut with use of a wildcard is, for instance, "*all* requirements with id = 1". Of course, pointcuts by extension are only one way of representing aspect compositions in requirements. Name-pattern-based (25, 26) and semantics-based (8) pointcut definitions are also used in several AORE approaches. An example of a name-pattern-based pointcut in the aspect-oriented V-graph-based approach (25) is "Register.*". This pointcut references all goals of "Register" type in a goal graph, independent of their topic names. A semantics-based pointcut definition, however, refers to the content (i.e., meaning) of the requirement that participates in a composition rather than its id or name patterns. One such approach is proposed in Ref. 8; it uses the grammatical semantics of the natural language as a basis for composition.

In AORE with Arcade the requirements in the aspects are the advices that are imposed upon viewpoints. This approach does not use intertype declarations and does not execute an actual composition to produce new requirements specification with advising requirements inserted into the viewpoints. Instead, the Arcade compositions are used as an analysis tool to detect requirement interactions and conflict points and to enable trade-off analysis. This same approach is taken in several other AORE works, (e.g., Refs. 8 and 27). The AO V-graph approach (25), however, does have a dedicated intertype declaration mechanism, whereby new goals are inserted into the goal graph at the identified points. AO V-graph also provides a tool-supported mechanism for transforming the original goals and aspects into an integrated-goalgraph. This integrated graph, however, is not used for analysis but only for validating the expected composition results.

In summary, AORE uses the joinpoint, pointcut, advice, and intertype declaration notions of AOSD to provide improved separation of concerns and composition at the requirements level. The composition definitions are often used as an analysis tool for conflict-point identification and interaction understanding. However, the actual transformational compositions are also realized in some approaches. It is also essential to note that not all aspectual artifacts identified at the requirements level will subsequently be represented as code-level aspects. On the contrary, some may well transform into other software artifacts (e.g., architectural topology) or business-related decisions (e.g., procedures for security policy used by the business) before an application is implemented. In addition, new aspects, often related to the selected development technology, will develop at the other stages of software development, but these will not be visible in requirements.

Concurrent Buffer Example in AORE

The Arcade example for the buffer and its related concurrency requirements is illustrated in Fig. 3. Note, that the XML notation of the Arcade approach is used, where viewpoints, aspects, and composition elements are defined, each of which contains a set of requirement elements. A set of composition actions and operators is also used (here touched upon briefly) where necessary to explain the general meaning of the composition. Details of the Arcade approach can be found in Ref. 4.

```
<Viewpoint name= "Buffer">
      <Requirement id ="1"> Check if the buffer is empty. </Requirement>
      <Requirement id ="2"> Get an element from the buffer</Requirement>
</Viewpoint>                                                          (a)
```

```
<Concern name ="Concurrency">
      <Requirement id ="1"> System locks the buffer. </Requirement>
      <Requirement id ="2"> System unlocks the buffer. </Requirement>
</Concern>                                                            (b)
```

```
Composition: ConcurrentBuffer
<Composition>
      <Requirement aspect="Concurrency" id="1">
         <Constraint action="enforce" operator="before">
            <Requirement viewpoint="Buffer" id="all"/>
         </Constraint>
         <Outcome action="fulfilled"/>
      </Requirement>
      <Requirement aspect="Concurrency" id="2">
         <Constraint action="enforce" operator="after">
            <Requirement viewpoint="Buffer" id="all"/>
         </Constraint>
         <Outcome action="fulfilled"/>
      </Requirement>
</Composition>                                                        (c)
```

Figure 3. AORE with Arcade representation of Buffer, Concurrency, and their composition.

Figure 3(a) shows the Buffer viewpoint, which has two requirements: first, stating the need to check if the buffer is empty, and second, instructing to get an element from the Buffer. The Concurrency aspect is demonstrated in Fig. 3(b), and contains two requirements: one for locking the Buffer and another for unlocking it. Finally, the interaction between the Buffer viewpoint and the Concurrency aspect is shown in the composition defined in Fig. 3(c). From this composition, we see that the requirements with id numbers 1 and 2 of the Concurrency aspect influence all the requirements of the Buffer viewpoint. Thus, Concurrency is a crosscutting concern with respect to Buffer. The requirement with id = 1 of the Concurrency aspect will be *enforced* (enforce action) *before* (before operator) all requirements of Buffer, whereas the requirement with id = 2 will be *enforced after* (after operator) all requirements of the Buffer. Thus, the Concurrency requirements with ids 1 and 2 advise the Buffer's requirements.

As shown in Fig. 3, the pointcuts for composition are quantified as viewpoint = "Buffer" and id = "all", and advice is defined by reference to concern and id pairs (e.g., aspect= "Concurrency", and id = "1"). The composition states that the advice must be imposed before/after the joinpoints selected via the pointcuts. The composition does not carry out any immediate transformation of the composed requirements. Instead, it is an analysis tool that explicitly shows the effect of the aspects on the viewpoints and the interactions between the two.

ASPECT-ORIENTED ARCHITECTURE

The architecture of a computing system is concerned with the description of its general structure. With the emergence of AOSD, several architectural-design approaches have focused on researching how aspects manifest in the architecture and how they should be represented (22, 28–30). A discussion of several aspect-oriented architecture (AOA) approaches is provided in Ref. 22. In this section, we discuss the work on aspect-oriented architecture description languages (ADLs) because these have been the main focus of development of aspect-oriented techniques at the architecture level.

Aspects in Aspect-Oriented Architecture (AOA)

In accordance with Medvidovic and Taylor (3), the necessary elements of an ADL are components, connectors, and architectural configurations. Components and connectors must have associated interfaces that detail the operations, messages, and variables the components use to interact with each other. These interactions and their rules are modeled via connectors. Most ADLs aiming to incorporate AOSD concepts have defined several extensions to this base set of ADL constructs. While discussing AO ADLs, we will use the work of Pinto and colleagues (32–34) as an example AO ADL, primarily because we believe that this work uses only the minimal set of extensions necessary to accommodate AOSD concepts in an ADL.

The majority of AO ADLs [such as Prisma (35), Fractal (36), and DAOP-ADL (37, 38)] have opted to add a new *aspect* construct into the ADL to represent a crosscutting concern at the architectural level. The AO ADL by Pinto and colleagues (32–34), however, suggests that no need exists to use a special type of component for an aspect. Component interface in this AO ADL is described as either provided by a given component or used (i.e., required) by it. The interface contains a set of operations and state variables. The operations are given role names.

AO ADL (32–34) encapsulates the broad crosscutting influence of aspectual concerns in the architectural connectors. Thus, instead of defining new *aspect* components, this approach refines the connectors by extending their semantics with *aspectual bindings*. Whereas regular bindings in the connectors represent the composition of architectural elements, the *aspectual bindings* represent the composition between components that have a broadly scoped interaction with the other components.

The *joinpoint model* of the AO ADL by Pinto and colleagues (32–34) comprises the role names and operation names exposed by the components. When defining an aspectual binding, a *pointcut* may be defined as, for instance, “any component playing a particular role” or “any component that has an operation with a given name in one of its provided interfaces”, and so forth.

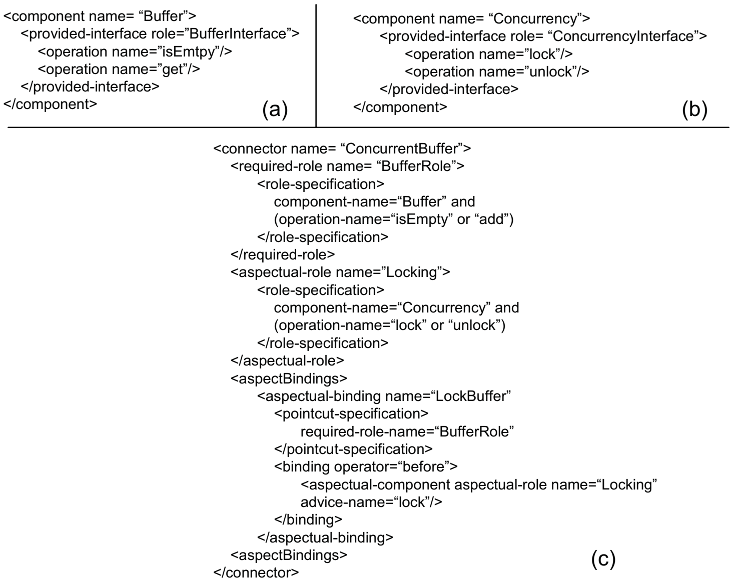
An *advice* in the AO ADL by Pinto and colleagues (32–34) is an operation, which is bound in the aspectual binding to a set of joinpoints selected by a pointcut. This advice operation can be imposed before, after, or concurrently with the advised operation. The AO ADL by Pinto and colleagues (32–34) does not use any constructs for *intertype declarations*. Instead, when it is necessary to introduce a new state variable or operation, the traditional notion of *composite component* is used. The modified component then is represented by a compound component with the original and additional interfaces. So far, the notion of intertype declaration has not been used much in AOA work (except for in Ref. 35), and the above discussed use of the traditional compound component for this purpose makes definition of such new construct unnecessary.

Thus, as shown by the above discussion, the accommodation of aspects within architecture design can be achieved without drastic changes to the traditional ADL constructs. For instance, Pinto and colleagues (32–34) do this by only extending the connector semantics. With such an approach, when no aspects are used in a system, the same component bindings as in non-AO systems can be used, which both minimizes changes to ADL constructs and enables component reuse.

It must be noted that in architecture, crosscutting concerns are not necessarily always related to individual components. For instance, work by Garcia et al. (39) illustrates that aspects are also useful in representing *architectural decisions* that affect multiple architectural views. In addition, not all identified crosscutting concerns can be modularized into components; some of them will continue to reside in multiple components (40) (e.g., beacuse of processing efficiency or other reasons). In such cases, it could still be useful to have a way of aspect documentation.

Concurrent Buffer Example in AOA

The AO ADL example for the buffer and its related concurrency elements is shown in Fig. 4 below. Here both Buffer and Concurrency concerns are represented as components, with their related operations. In this case, the Buffer provides two operations: *isEmpty* and *get* [Fig. 4(a)]. Similarly, the Concurrency component provides the *lock* and *unlock* operations [Fig. 4(b)]. The connector for these



```
<component name= “Buffer”>
   <provided-interface role=”BufferInterface”>
      <operation name=”isEmtpy”/>
      <operation name=”get”/>
   </provided-interface>
</component>
```

(a)

```
<component name= “Concurrency”>
     <provided-interface role= “ConcurrencyInterface”>
          <operation name=”lock”/>
          <operation name=”unlock”/>
     </provided-interface>
</component>
```

(b)

```
<connector name= “ConcurrentBuffer”>
   <required-role name= “BufferRole”>
        <role-specification>
           component-name=“Buffer” and
           (operation-name=“isEmpty” or “add”)
        </role-specification>
   </required-role>
   <aspectual-role name=”Locking”>
        <role-specification>
           component-name=“Concurrency” and
           (operation-name=“lock” or “unlock”)
        </role-specification>
   </aspectual-role>
   <aspectBindings>
        <aspectual-binding name=“LockBuffer”
           <pointcut-specification>
                required-role-name=“BufferRole”
           </pointcut-specification>
           <binding operator=“before”>
                <aspectual-component aspectual-role name=“Locking”
                advice-name=“lock”/>
           </binding>
        </aspectual-binding>
   <aspectBindings>
</connector>
```

(c)

Figure 4. AOA representation of Buffer and Concurrency concerns and their connector.

two concerns is shown in Fig. 4(c), in which an aspectual binding is defined to pre-advise (operator = before) all operations of the Buffer (pointcut specification: “Buffer-Role” where component name = “Buffer” with operation names “isEmtpy” or “get”) with the lock (advice name = “lock”) operation of the Concurrency component. If desired (although not shown here), a similar binding could have been defined to post-advise (if used with operator = after) all Buffer operations with the *unlock* operation of the Concurrency component.

ASPECT-ORIENTED DESIGN

In a vein similar to AORE and AOA, several approaches have been proposed for modeling and analysis of aspects during detailed design. Mostly, these approaches are based on the UML or its extensions; they provided support for modularization of crosscutting concerns and for specifying their compositions with other design elements [for a detailed analysis of aspect-oriented design techniques, see the survey by Chitchyan et al. (22)]. Most interesting approaches in this context are Theme/UML (15, 16) and the works by Klein et al. (41) and Cottenier et al. (42). We discuss Theme/UML in more detail below. The work by Klein et al. focuses on semantics of design composition and treats aspect composition as a model-composition problem. The approach proposed by Cottenier et al. exposes the state semantics of the design elements and bases the aspect compositions on these state chart specifications (exposed as part of the joinpoint model). This approach is rooted in the notion of model-driven engineering whereby weaving takes place at the design level, and code for the target platform is generated.

Aspects in Aspect-Oriented Design (AOD)

The vast majority of the work on AOD has opted to extend UML to accommodate aspect-oriented concepts. Some approaches have extended the UML meta-model (14, 15, 43), others have provided dedicated AOD profiles (44), and yet others have used a subset of UML’s built-in extension mechanisms, such as stereotypes (28, 29, 45). In the rest of this section, we will use the Theme/UML (15, 16) approach to demonstrate the AO concepts in design because this approach is one of most referenced and generic AOD approaches developed so far.

In the Theme/UML approach, the UML meta-model is extended to support modularization of designs into *themes. Themes* are UML packages that may encapsulate any feature, concern, or requirement that must be handled in the system (16). Here, the notion of a *crosscutting theme* is used for aspect representation. A crosscutting theme is a theme that provides some behavior that crosscuts other behavior in designs (16). In Theme/UML, the behavior is modeled via sequence diagrams, whereas structural crosscutting (i.e., nonbehavioral elements that could be included into a base theme via a crosscutting theme) can be modeled via class and package diagrams. Other approaches also use other UML modeling elements, such as activity diagrams (13), or focus on a subset of crosscutting, for example modeling only the structural elements as aspects via a dedicated ≪aspect≫ stereotype (43), or modeling only the various joinpoint selection queries (46, 47) and so forth.

The joinpoints in Theme/UML are represented as parameterized class and operation names that are defined and referenced in a theme’s template parameters section. These parameters define the Theme/UML pointcuts. During theme composition, these parameterized names are assigned to specific classes and operations via *binding* specifications. The binding specification also elaborates on the details of composition (e.g., merge or override the operation from the base theme with that of the crosscutting one). Thus, the binding in Theme/UML specifies the precedence of operations (e.g., which operation will be executed first), the type of advice (e.g., before/after or around), and, where necessary, conflict resolution between elements of themes to be merged. The work on joinpoint-designation diagrams (JPDD) (46, 47) provides a set of somewhat-modified (because of parameterization, addition of indirect relationships, etc.) UML-based modeling elements for capturing the intention of joinpoint selection queries, i.e., pointcuts. The JPDDs represent the conceptual models of queries: whether the joinpoints are selected on basis of object interaction (control flow-based queries realized via UML-based sequence diagrams), data interchange (data flow-based queries realized via UML-based activity diagrams), or state change (state-based queries realized via UML-based state charts). Each of these JPDD types can be used in combination with the others and the (modified) class/object diagrams.

Although Theme/UML does not explicitly define *introduction* as a construct, the theme merging requires inclusion of the nonparameterized elements of the contributing crosscutting themes into the final result. Thus, such nonparameterized crosscutting theme elements as classes, operations, attributes, and relationships (e.g., inheritance) can be perceived as introductions into the composed theme: They modify/crosscut the structure of the original base themes. With JPDDs, the introductions can be asserted via class/object diagrams. Moreover, here the powerful notion of *indirect relationships*(46, 47) is also used, whereby the existence of the relationship is stated without naming its precise location. For example, a new class in the class hierarchy can be introduced without stating its precise location. Instead, using the indirect inheritance relation, one may state that this class will be added somewhere in the given hierarchy.

Concurrent Buffer Example in AOD

The Theme/UML example for the buffer and its crosscutting concurrency concern is shown in Fig. 5 below.

Here both Concurrency and Buffer concerns are represented as themes. Concurrency [shown in Fig. 5(a)] is the crosscutting theme. Concurrency has *lockedOp()* and *unlockedOp()* template operations intended to replace respectively the original*_locked()* and *unlocked()* operations of some ConcurrenctClass by pre-advising these original operations with *lock()* operation of Concurrency and by post-advising them with its *unlock()* operation. The execution order of the operations (i.e., pre-advising, original operation, and post-advising) would normally be

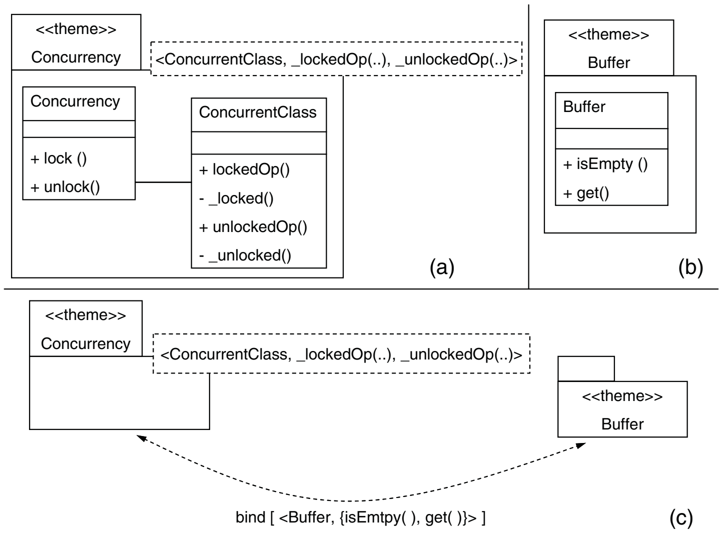


Figure 5. AOD with Theme representation of Concurrency and Buffer themes and their binding.

presented via an additional UML interaction diagram that is omitted in Fig. 5.

The base Buffer theme, which contains two public operations, *isEmpty()* and *get(),* is presented in Fig. 5(b). To specify that the Buffer theme is crosscut by the Concurrency theme, they are bound, as shown in Fig. 5(c). The binding specification [shown at the bottom of Fig. 5(c)] states, that

- the Buffer class of the Buffer theme instantiates the parameterized ConcurrentClass of the Concurrency theme, and
- both *isEmtpy()* and *get()* operations instantiate the *_lockedOp()* and *_unlockedOp()* operations. Each *isEmtpy()* and *get()* operation will be advised by both the *_lockedOp()* and *_unlockedOp()* operations, because the two former operations are grouped via curly brackets.

ASPECT-ORIENTED PROGRAMMING (AOP)

The aspect-oriented paradigm is endowed with a large and vibrant body of AOP languages (see http://www.aosd.net) and frameworks (e.g., Refs. 7, 48 and 49), that can be applied in the development stage of the software lifecycle. This situation can be explained by the fact that AOP first emerged as a programming-level technique but is perhaps more accurately motivated by the fact that almost every AOP language has its particular purpose and characteristics. A survey of the contemporary work on AOP languages and language execution models is available from Ref. 50.

Aspects in AOP

In this section, we discuss the instantiation of the aspectual elements for AOP by using as an example a particular programming language: AspectJ (6, 5). AspectJ can be considered as the flagship language of aspect orientation as it has made a transition from being a research topic to the state-of-the-practice.

AspectJ extends the Java language with aspect-oriented constructs for pointcuts, advice, and inter type declarations; it permits to encapsulate these into a class-like module: the *aspect*. Like an ordinary class, an aspect can also have methods, fields, and initializers of its own. Advices in AspectJ (and most other aspect languages) can be seen as anonymous methods, bound to a pointcut rather than to a method name and with bodies not essentially different from regular Java methods. The advice body contains regular statements and expressions that are executed when the advice is invoked at a joinpoint captured by its pointcut. Most often, specific keywords and pseudo-variables are available for use in advices only. AspectJ, for example, offers access to the currently executing joinpoint through the pseudo variable *thisJoinPoint*. Nevertheless, in some AOP languages, interesting exceptions from such an advice model occur. Most notably, *domain-specific aspect languages* (DSALs) permit the implementation of an aspect's advice in a dedicated domain-specific language. Interestingly, some earliest aspect languages are DSALs: D is a DSAL for synchronization concerns (51) and RG was designed specifically as an optimization aspect performing loop fusions. More recently, DSALs have emerged for aspects such as business rules (52) and advanced transaction management (53).

The invocation of any advice is determined by a pointcut, which is defined (traditionally, although not always) within the aspect itself by means of a set of predefined *pointcut predicates*. Such predicates permit to quantify over joinpoints (and their properties) to identify the joinpoints of interest to the aspect. In AspectJ, joinpoints are principled points in the execution of the program, such as field references or assignments, method executions, calls, and returns, to name but a few. These joinpoints can be qualified by the signatures and names of the fields, by methods and classes they apply to, and also with respect to the

```
1    public aspect Locking {
2      private final ReentrantReadWriteLock lock = new ReentrantReadWriteLock();
3      private final Lock readLock = lock.readLock();
4
5      pointcut lockedMethods(): execution(* SimpleBuffer.*(..));
6
7      before(): lockedMethods() {
8       readLock.lock();
9      }
10
11      after(): lockedMethods() {
12             readLock.unlock();
13      }
14  }
```

Figure 6. The Locking aspect in AspectJ.

dynamic scope in which the joinpoint is executed. Most aspect languages feature a similar pointcut specification language, with some notable exceptions using the language's reflective facilities (e.g., AspectS (59)) or computational logic (e.g., Carma (55, 56) and Alpha (57)) to define pointcuts.

As a result of composition, most AOP languages produce a merged executable code from the aspectual and nonaspectual modules. In AspectJ, a compiler is used to compose the aspects' implementation with the Java classes by "weaving" the aspect code into the classes that are crosscut by the aspect. The term "weaver" has historically been used for aspect-oriented compilers, because these had generally been implemented as a preprocessor to other compilers. The AspectJ compiler, for example, was using such an approach but now performs weaving on the bytecode of (compiled) classes. Currently, an entire body of research investigates efficient execution and compilation mechanisms for aspect languages (58, 59). Research is under way, for example, for efficient weaving of aspects with complex and dynamic pointcuts (60) or weaving of aspects that can be dynamically removed from the running software application. These weavings require sophisticated weaving techniques that are sound and that minimize the run-time performance overhead.

Concurrent Buffer Example in AOP

We can now change the implementation of our concurrent buffer example from Fig. 2 so that the crosscutting concern of concurrency locking is modularized as an aspect. From the code in Fig. 2, one can distill that the joinpoints of the concurrency locking aspect and the buffer functionality are the execution of the methods of the buffer (i.e., *isEmpty(), get(), . . .*). Indeed, whenever the method *isEmpty()* executes, we need to lock the *readLock* and unlock it when the execution has finished. Figures 6 and 7 show the implementations of the *Locking* aspect and the *SimpleBuffer* class using the AspectJ language. The code for both concerns is now cleanly separated. The *Buffer* class is merely dealing with the functionality of a buffer and the entire locking concern is modularized in the aspect.

For simplicity, the *lockedMethods()* pointcut of the aspect captures all method execution joinpoints of the *SimpleBuffer* class. Line 5 of Fig. 6 defines this pointcut by means of the *execution* pointcut-predicate. This predicate accepts a method signature as an argument and captures all joinpoints that are executions of methods characterized by that signature. The signature consists of the return type, class name, method name, and argument types of method. In the definition of such a signature, wildcards can be used to describe a pattern, rather than concrete signatures. In the example, we use the "*" wildcard to match any return type and method name. Similarly, the ".." wildcard is used to match any type pattern in the argument list. Therefore, the *lockedMethods()* pointcut will capture all executions of any method defined on the SimpleBuffer class.

The actual aspect behavior is expressed in a *before* advice that executes the locking and in an *after* advice that executes the unlocking. Each of these advices is defined such that they are executed when a joinpoint captured by the *lockedMethods()* pointcut occurs. The binding of the advice to the pointcut is specified in the header of each advice (lines 7 and 11). As such, when the *isEmpty()* method executes, the aspect is implicitly invoked and the before advice executes, followed by the *isEmpty()* method body. The after advice is executed when the *isEmpty()* method execution finishes.

```
public class SimpleBuffer {
 private Object[] data = new Object[100];
   private int nrOfElements=0;

   public boolean isEmpty() {
          return nrOfElements == 0;
   }

public Object get() {
    return data[nrOfElements--];
}
}
```

Figure 7. The SimpleBuffer class without synchronization code.

AOSD APPLICATIONS AND OPEN RESEARCH CHALLENGES

Several nontrivial applications of AOSD techniques have been reported in literature, both in terms of nontrivial studies in academic research labs and in real-world industrial settings. A significant number of these relate to middleware and distributed systems, both from the perspective of taming the complexity of the middleware itself (61, 62), and supporting modularization of crosscutting concerns in distributed applications (63–65). Other nontrivial application studies pertain to modularizing persistence-related functionality, (66, 67), and evolution of operating-system code, (68). Several industrial AOP frameworks, for example JBoss (48) and Spring (69), target enterprise component architectures, whereas the mysql database system (http://www.mysql.org) now uses AspectJ as the underlying logging mechanism. In terms of industrial adoption, AspectJ has made a transition from state-of-the-art to becoming state-of-the-practice, whereas frameworks such as JBoss and Spring are also seeing increasing use in Enterprise Java applications.

Seveal empirical studies of AOSD techniques have also been reported, (70–73). These studies quantify the comparative benefits and drawbacks of AOSD compared with other software-development paradigms, such as OO. Similarly, studies demonstrating the nontrivial nature of crosscutting concerns in industrial code (74) highlight the fundamental complexity of software systems that AOSD techniques aim to address.

Despite the promising applications and empirical studies, several open research challenges remain. The major challenge to industrial adoption is in the area of aspect testing. Although a few techniques have been proposed to date, (75–77), extensive research is needed to understand whether current software testing techniques can be applied in an AOSD context and, more importantly, what specific testing challenges need to be addressed for using AOSD in an industrial setting. Related to this is the issue of composition fragility. Most AOSD techniques employ syntactic references (or wildcards quantifying over such syntactic references) as a basis of composition specifications.[1] This use strongly couples aspects to other modules, which not only makes testing of aspects very challenging but also leads to composition fragility whereby changes in other modules invalidate pointcut specifications.[2] This problem has been highlighted by recent works, (8, 30), that advocate composition specifications based on high-level models and semantics. Finally, the deployment of aspects in heterogeneous environments is an area largely unexplored. Most current AOP techniques tend to focus on deploying aspects within the context of a single system or virtual machine. Aspect deployment in a heterogeneous distributed environment is a topic that requires close attention to make AOP techniques relevant to the emerging class of large-scale distributed systems such as those in ubiquitous and GRID-based systems.

BIBLIOGRAPHY

1. E. W. Dijkstra, *A Discipline of Programming*. Englewood Cliffs, NJ: Prentice Hall, 1976.
2. K. van de Berg, J.-M. Conejero, and R. Chitchyan, AOSD ontology 1.0: Public ontology of aspect orientation, in *Report of the EU Network of Excellence on AOSD*, 2005.
3. G. Kiczales, AspectJ: Aspect-oriented programming using java technology (0.7), in *JavaOne Conference*, 2000.
4. A. Rashid, A. Moreira, and J. Araujo, Modularisation and composition of aspectual requirements, 2nd International Conference on Aspect-Oriented Software Development (AOSD'02), 2003, pp. 11–20.
5. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, and J. Irwin, Aspect-oriented programming, 11th European Conference on Object-Oriented Programming (ECOOP), 1997, Springer LNCS 1241, pp. 220–242.
6. The AspectJ Project. Available: http://www.eclipse.org/aspectj/, 2007.
7. JAC - A Framework for Aspect-Oriented Programming in Java. Available: http://jac.objectweb.org/, 2007.
8. R. Chitchyan, A. Rashid, P. Rayson, and R. Waters, Semantics-based composition for aspect-oriented requirements engineering, 6th International Conference on Aspect-Oriented Software Development, 2007, ACM Press, pp. 36–48.
9. J. Xu, H. Rajan, and K. Sullivan, Understanding aspects via implicit invocation, Automated Software Engineering (ASE), 2004, IEEE Computer Society Press, pp. 332–335.
10. G. Kiczales and M. Mezini, Separation of concerns with procedures, annotations, advice and pointcuts, European Conference on Object-Oriented Programming (ECOOP), 2005, Springer LNCS 3586, pp. 195–213.
11. A. Moreira, J. Araujo, and A. Rashid, Multi-dimensional separation of concerns in requirements engineering, 13th IEEE International Conference on Requirements Engineering Conference (RE 05), 2005, IEEE Computer Society, pp. 285–296.
12. A. Rashid and A. Moreira, Domain models are NOT aspect free, *Proc. MoDELS / UML*, 2006, pp. 155–169.
13. I. Jacobson and P.-W. Ng, *Aspect-Oriented Software Development with Use Cases*. Addison Wesley Professional, 2005.
14. E. Baniassad and S. Clarke, Theme: An approach for aspect-oriented analysis and design, International Conference on Software Engineering, 2004, IEEE Computer Society, pp. 158–167.
15. S. Clarke and E. Baniassad, *Aspect-Oriented Analysis and Design: the Theme Approach*. Addison-Wesley, 2005.
16. S. Clarke and R. Walker, Generic aspect-oriented design with Theme/UML, in zilla Elrad. R. E. Filman, S. Clarke, M. Aksit, (eds.) *Aspect-Oriented Software Development*. Addison-Wesley, 2005, pp. 425–458.
17. I. Jacobson, M. Chirsterson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach* 4th ed. Addison-Wesley, 1992.
18. A. Dardenne, A. v. Lamsweerde, and S. Fickas, Goal-directed requirements acquisition, *Science of Computer Programming*, **20**. pp. 3–50, 1993.
19. A. Lamsweerde, Goal-oriented requirements engineering: A guided tour, 5th IEEE International Symposium on Requirements Engineering, 2001, IEEE Press, pp. 249–263.
20. J. Whittle and J. Araujo, Scenario modeling with aspects, *IEEE Proceedings - Software*, Vol. **151**, No. 4, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, pp. 157–172, 2004.
21. E. L. A. Baniassad, P. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan, Discovering Early Aspects, *IEEE Software*, **23** (1): pp. 61–69, 2006.
22. R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. Pinto, J. Bakker, B. Tekinerdogan, S. Clarke, and A. Jackson, Survey of (aspect-oriented) analysis and design approaches, Lancaster

[1] This problem is not a problem specific to AOSD. Most contemporary software-development paradigms employ such syntactic references for method calls, attribute accesses, and so forth.

[2] Note that the issue of composition fragility is not limited to AOP languages. The problem has also been highlighted in analysis and design techniques, for example, in the context of requirements engineering (8).

University, Lancaster, Survey Report AOSD-Europe-ULANC-9, 2005.

23. I. Sommerville and P. Sawyer, PREview viewpoints for process and requirements analysis, Lancaster University, Lancaster REAIMS/WP5.1/LU060, 29 May 1996.
24. I. Sommerville and P. Sawyer, Viewpoints: Principles, problems and a practical approach to requirements engineering, *Annals of Software Engineering*, **3**: 101–130, 1997.
25. L. F. Silva, A guided strategy the modeling aspect-oriented requirements (in Portuguese), in *Computer Science*, vol. PhD. Rio de Janeiro, Brazil: Catholic University of Rio de Janeiro (PUC-Rio), 2006.
26. Y. Yu, J. C. S. d. P. Leite, and J. Mylopoulos, From goals to aspects: Discovering aspects from requirements goal models, International Conference on Requirements Engineering, 2004, IEEE Computer Society, pp. 38–47.
27. A. Moreira, J. Araújo, and I. Brito, Crosscutting quality attributes for requirements engineering, 14th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2002, ACM, pp. 167–174.
28. W.-M. Ho, F. Pennaneach, J.-M. Jezequel, and N. Plouzeau, Aspect-oriented design with the UML, in *Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000)*, 2000.
29. W. M. Ho, J.-M. Jezequel, F. Pennaneac'h, and N. Plouzeau, A toolkit for weaving aspect oriented UML designs, *Proc. 1st Int' Conf. on Aspect-Oriented Software Development (AOSD-2002)*, G. Kiczales, (ed.), 2002, pp. 99–105.
30. A. Kellens, K. Mens, J. Brichau, and K. Gybels, Managing the evolution of aspect-oriented software with model-based pointcuts, European Conference on Object-Oriented Programming (ECOOP), 2006, Springer LNCS 4067, pp. 501–525.
31. N. Medvidovic and R. Taylor, A classification and comparison framework for software architecture description languages, *IEEE Trans. Soft. Eng.*, **26**(1): 70–93, 2000.
32. R. Chitchyan, M. Pinto, A. Rashid, and L. Fuentes, COMPASS: composition-centric mapping of aspectual requirements to architecture, *Transactions on Aspect-Oriented Software Development*, Vol. 4, 2007, pp. 3–53.
33. M. Pinto and L. Fuentes, AO-ADL: An ADL for describing aspect-oriented architectures. Early Aspect Workshop (held with AOSD'07), 2007.
34. M. Pinto, N. Gámez, and L. Fuentes, Towards the architectural definition of the health watcher system with AO-ADL, Early Aspect Workshop (held with ICSE'07), 2007.
35. J. Pérez, I. Ramos, J. Jaén, P. Letelier, and E. Navarro, PRISMA: Towards quality, aspect-oriented and dynamic software architectures, 3rd IEEE Intl Conf. on Quality software (QSIC 2003), 2003, IEEE Computer Society, pp. 59–66.
36. N. Pessemier, L. Seinturier, and L. Duchien, Components, ADL and AOP: Towards a common approach, RAMSE: Workshop on Reflection, AOP, and Meta-Data for Software Evolution (held with ECOOP'04), 2004, pp. 61–69.
37. M. Pinto, L. Fuentes, and J. Troya, A dynamic component and aspect platform, *The Computer Journal*, **48**(4): 401–420, 2005.
38. M. Pinto, D. Jiménez, and L. Fuentes, Developing dynamic and adaptable applications with CAM/DAOP: A virtual office application, 4th International Conference on Generative Programming and Component Engineering (GPSE), 2005, LNCS, 3676, pp. 438–441.
39. A. Garcia, T. Batista, A. Rashid, and C. Sant'Anna, Driving and Managing Architectural Decisions with Aspects, *ACM SIGSOFT Software Engineering Notes*, **31**(5), 2006.
40. C. Sant'Anna, E. Figueiredo, A. F. Garcia, and C. J. P. d. Lucena, On the modularity of software architectures: A concern-driven measurement framework, First European Conference on Software Architecture (ECSA'07), 2007, LNCS, 4758, pp. 207–224.
41. J. Klein, L. Helouet, and J. Jezequel, Semantics-based weaving of scenarios, International Conference on Aspect-Oriented Software Development (AOSD), 2006, ACM, pp. 27–38.
42. T. Cottenier, A. van den Berg, and T. Elrad, Joinpoint inference from behavioral specification to implementation, European Conference on Object-Oriented Programming (ECOOP), 2007, Springer LNCS 4609, pp. 476–500.
43. J. Suzuki and Y. Yamamoto, Extending UML with aspects: Aspect support in the design phase, Workshop on Aspect-Oriented Programming (held with ECOOP 1999), 1999.
44. L. Fuentes, M. Pinto, and A. Vallecillo, How MDA can help designing component- and aspect-based applications, Enterprise Distributed Object Computing Conference (EDOC), 2003.
45. J. L. Herrero, F. Sanchez, F. Lucio, and M. Toro, Introducing separation of aspects at design time, Workshop on Aspects and Dimensions of Concerns (held with ECOOP 2000), 2000.
46. S. Hanenberg, D. Stein, and R. Unland, From aspect-oriented design to aspect-oriented programs: Tool-supported translation of JPDDs into code, 6th International Conference on Aspect-Oriented Software Development, 2007, ACM, pp. 49–62.
47. D. Stein, S. Hanenberg, and R. Unland, Expressing different conceptual models of join point selections in aspect-oriented design, 5th International Conference on Aspect-Oriented Software Development, 2006, ACM, pp. 15–26.
48. JBoss Aspect Oriented Programming Webpage. Available: http://www.jboss.org/products/aop: JBoss, 2007.
49. The Spring Framework. Available: http://www.springframework.org/, 2007.
50. Survey of aspect-oriented languages and execution models, VUB, Brussels, Belgium, AOSD-Europe Project Report (D12) AOSD-Europe-VUB-01, 2005.
51. C. V. Lopes, D: A language framework for distributed programming, Ph.D. dissertation, Boston, MA: Northeastern University, 1997.
52. M. D'Hondt and V. Jonckers, Hybrid aspects for weaving object-oriented functionality and rule-based knowledge, International Conference on Aspect-Oriented Software Development (AOSD), 2004, ACM, pp. 132–140.
53. J. Fabry and T. Cleenewerck, Aspect-oriented domain specific languages for advanced transaction management, International Conference on Enterprise Information Systems (ICEIS'05), 2005, pp. 428–432.
54. R. Hirschfeld, Aspects - Aspect-Oriented Programming with Squeak, in M. Askit, M. Mezini, and R. Unland, (eds.), *Objects, Components, Architectures, Services and Applications for a Networked World*. Springer, 2003, pp. 216–232.
55. J. Brichau, A. Kellens, K. Gybels, K. Mens, R. Hirschfeld, and T. D'Hondt, Application-specific models and pointcuts using a logic meta language, in W. D. Meuter (ed.), *Advances in Smalltalk*, vol. LNCS. Springer-Verlag, 2006, pp. 1–22.
56. K. Gybels and J. Brichau, Arranging language features for more robust pattern-based crosscuts, Second International Conference on Aspect-Oriented Software Development (AOSD), 2003, ACM, pp. 60–69.
57. K. O stermann, M. Mezini, and C. Bockisch, Expressive pointcuts for increased modularity, European Conference on

Object-Oriented Programming (ECOOP), 2005, Springer-Verlag, LNCS, pp. 214–240.

58. C. Bockisch, M. Haupt, M. Mezini, and K. Ostermann, Virtual Machine Support for Dynamic Join Points, International Conference on Aspect-Oriented Software Development (AOSD'04), 2004, ACM, pp. 83–92.

59. M. Haupt and M. Mezini, Virtual machine support for aspects with advice instance tables, First French Workshop on Aspect-Oriented Programming, 2004.

60. C. Bockisch, S. Kanthak, M. Haupt, M. Arnold, and M. Mezini, Efficient Control Flow Quantification, International Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA), 2006, ACM Sigplan, pp. 125–138.

61. E. Truyen, B. N. Joergensen, and W. Joosen, Customization of object request brokers through dynamic reconfiguration, TOOLS Europe, 2000, IEEE Computer Society Press, pp. 181–194.

62. C. Zhang and H.-A. Jacobsen, Resolving feature convolution in middleware systems, ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 2004, ACM, pp. 188–205.

63. A. Colyer and A. Clement, Large-scale AOSD for middleware, International Conference on Aspect-Oriented Software Development (AOSD), 2004, ACM, pp. 56–65.

64. M. A. Kersten and G. C. Murphy, Atlas: A case study in building a web-based learning environment using aspect-oriented programming, *SIGPLAN Notices, OOPSLA*, **34**(10): 340–352, 1999.

65. S. Soares, E. Laureano, and P. Borba, Implementing distribution and persistence aspects with AspectJ, Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), 2002, ACM, pp. 174–190.

66. A. Rashid, *Aspect-Oriented Database Systems*. Springer-Verlag, 2003.

67. A. Rashid and R. Chitchyan, Persistence as an aspect, 2nd International Conference on Aspect-Oriented Software Development, 2003, ACM, pp. 120–129.

68. Y. Coady and G. Kiczales, Back to the future: a retroactive study of aspect evolution in operating system code, International Conference on Aspect-Oriented Software Development (AOSD), 2003, ACM, pp. 50–59.

69. Spring, Aspect-Oriented Programming with Spring. Available: http://www.springframework.org/docs/reference/aop.html, 2007.

70. N. Cacho, C. Sant'Anna, E. Figueiredo, A. Garcia, T. Batista, and C. Lucena, Composing design patterns: A scalability study of aspect-oriented programming, International Conference on Aspect-Oriented Software Development (AOSD), 2006, ACM, pp. 109–121.

71. A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. v. Staa, Modularizing design patterns with aspects: A quantitative study, International Conference on Aspect-Oriented Software Development (AOSD), 2005, ACM, pp. 3–14.

72. P. Greenwood, T. Bartolomei, E. Figueiredo, M. Dosea, A. Garcia, N. Cacho, C. Sant'Anna, U. Kulesza, S. Soares, P. Borba, and A. Rashid, On the impact of aspectual decompositions on design stability: An empirical study, European Conference on Object-Oriented Programming (ECOOP'07), 2007, Springer LNCS 4609, pp. 176–200.

73. C. Lopes and S. Bajracharya, An analysis of modularity in aspect oriented design, International Conference on Aspect-Oriented Software Development (AOSD), 2005, ACM, pp. 15–26.

74. M. Bruntink, A. v. Deursen, M. D'Hondt, and T. Tourwe, Simple crosscutting concerns are not so simple: Analysing variability in large-scale idioms-based implementations, International Conference on Aspect-Oriented Software Development (AOSD), 2007, ACM, pp. 199–211.

75. T. Xie and J. Zhao, A framework and tool supports for generating test inputs of aspectJ programs, International Conference on Aspect-Oriented Software Development (AOSD), 2006, ACM, pp. 190–201.

76. D. Xu and W. Xu, State-based incremental testing of aspect-oriented programs, International Conference on Aspect-Oriented Software Development (AOSD), 2006, ACM, pp. 180–189.

77. J. Zhao, Data-flow-based unit testing of aspect-oriented programs, International Computer Software and Applications Conference (COMPSAC'03), 2003, IEEE Computer Society, pp. 188–197.

Johan Brichau
Université catholique
de Louvain
Louvain-la-Neuve, Belgium
Ruzanna Chitchyan
Awais Rashid
Lancaster University
Lancaster, United Kingdom
Theo D'Hondt
Vrije Universiteit Brussel
Brussels, Belgium

ASSEMBLY LANGUAGE

INTRODUCTION

An assembly language is a symbolic representation of a corresponding machine language. Whereas a machine language program consists of bit patterns, an assembly language program consists of alphanumeric names (symbols), numbers, and other special characters. The names describe operations to be performed (mnemonics) as well as storage locations and registers from which data (operands) are to be fetched or stored. While the machine language instruction is the only instruction a computer can "understand," it is very difficult for humans to write a program using pure machine language directly. Assembly language was the first step taken some 60 years ago to facilitate programming and it led to the development of more powerful programming languages (higher level languages).

A program called the assembler is used to translate the assembly language code to its machine language equivalent. This translation process is known as assembling (*or assembly*). The *object code* produced is then loaded into the computer usually with the help of a *loader* (and a *linker*) before it is *run* (*or execute*) t here.

Today, with the availability of powerful high level languages, assembly language is not used for writing programs directly except in some special situations. Nevertheless, most compilers still translate programs written in high level language first to assembly language code and then use the assembler to generate the object code in machine language. Therefore, assembly language is still playing a key role in the operation of all computers. In situations when performance and/or resource limitations are essential and compiler-generated code may not be optimal, people often rely on assembly language programming to maximize program performance (e.g., the innermost loop of computationally very intensive program) or to preserve resources (e.g., memory space in embedded systems).

The idea of using symbolic code to write programs was developed soon after the first computers were built. Certainly, something was available on the EDSAC (done by David Wheeler) called "initial orders"(1). One punched three items into paper tape (a letter, a decimal address, and a final letter) and the Initial Orders (which were actually stored in a read only memory made from telephone uniselectors) would convert the letter to the binary machine language operation code (*op code*), convert the decimal address to binary, and then add 1 of 12 constants that had been preset by the programmer to that address depending on the final letter.

FORMAT OF ASSEMBLY STATEMENTS

An assembly language program consists of a collection of statements. An assembly statement is usually entered on one line and may consist of as many as four fields: the label, the operation, the operand, and the comment fields. The general format of a statement is:

```
Label Operation Operand Comment
```

For example:

`BEGIN ADD ALPHA,BETA,GAMMA` $\gamma = \alpha + \beta$

Some assemblers require that the fields start in particular columns. Such assemblers are said to require the statements to be in fixed format. Other assemblers allow a free format, wherein the field can be separated by one or more blanks or by special symbols.

1. Label field. A label is a user-defined symbol that is used as the name of a memory location. The name may then be used by other statements to refer to that memory location instead of the numerical machine address of the location. A label on a statement is needed only when the statement is referred to elsewhere in the program. It may be left blank.
2. Operation field. The operation field contains a symbolic name of an operation. The operation specifies an action to be performed by the computer either at assembly time or at run time, which depends on the type of operation. An operation may be one of three types.
   a. An executable instruction. This operation is to be carried out by the computer when running the program. It is represented here symbolically as a mnemonic of a machine language instruction (such as ADD, DIV, or BR). At assembly time, the mnemonic is translated by the assembler into a binary string, the *opcode*.
   b. A directive (pseudo operation). Directives are commands to the assembler to perform certain function during assembly time. For example, a directive may be called "DS" (Define Storage) which tells the assembler to reserve a block of memory space for the program.
   c. A macro call. A *macro* assigns a name to a sequence of assembly statements, called the body of the macro. When the name of the macro appears in the operation field of an assembly statement, the assembler will insert the body of the macro in that location of the program. Not every assembler provides such a facility.
3. Operand field. The operand field contains information needed by the operator in the operation field. It may consist of several subfields; each contains an operand, depending on the nature of the operation. An operand may be in the form of a symbolic name, such as ALPHA, which represents the memory location at which the operand resides. It may be a special reserved

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

name, such as R4, which specifies an internal register of the computer in which the operand resides. It may be a constant, such as 314, which defines the value of the operand directly. Or, it may even be a simple arithmetic expression made up of symbols, constants, and arithmetic operators, such as ALPHA + 4

4. Comment field. The comment field contains a description of the function of the statement. The comment field is strictly for the benefit of the programmer and is not part of the program itself. Most assemblers provide such a field but ignore it during assembly.

EXAMPLES OF ASSEMBLY LANGUAGE CODE

In his paper "Programming the EDSAC (2)," Campbell-Kelly showed an example of code using the first form of the initial order (Figure 1).

Location	Order	Notes
100	T L	Clear accumulator using location 0 as a rubbish bin
101	A 8L	Add location 8 into accumulator
102	E 105S	Transfer control to location 105 if accumulator ≥ 0
103	S 8L	Subtraction location 8 from accumulator
104	S 8L	Subtraction location 8 from accumulator
105	T 8L	Store accumulator in location 8, leave accumulator clear

Figure 1. An example of an early assembly language program to convert a number to its absolute value.

Only the symbols in the "Order" column are entered into the computer without any spacing. Basically, only the operation and the operand fields are used. Because absolute addresses are used, the program is not relocatable. The location field, like the comment field (Notes), is ignored. The S and L in the second field of each instruction specify the length of the operand. The first version of the initial order was soon replaced by an improved version. The program above can now be written in the subroutine form as below (Figure 2). Symbolic names are now used in the operand field and the program is more relocatable. The first line of the code is not an executable instruction, but rather is a directive (pseudo-operation) that instructs the initial order to save the location of θ which will be used in the subsequent code to calculate the address.

Location	Order	Notes
	G K	Control Combination
0	T D	Clear accumulator
1	A H	Add location h into accumulator
2	E 5θ	Transfer control to location 5θ if accumulator ≥ 0
3	S H	Subtraction location H from accumulator
4	S H	Subtraction location H from accumulator
5	T H	Store accumulator in location H, leave accumulator clear

Figure 2. Assembly language of program in Figure 1 written as a subroutine

The same example, using a popular computer architecture in the 1980s viz. the VAX-11, may have the following code (data input omitted):

```
NUM: .BLKW    1           ;reserve a 32 bit
                            memory space for
                            input
ABS: .BLKW    1           ;reserve a 32 bit
                            memory space for
                            result
     .ENTRY   ABSOLUTE,0 ;a directive indicat-
                            ing the entry point
     MOVW     NUM,R2      ;Is NUM ≥ 0?
     BGEQ     POSITIVE    ;yes
     MNEGW    R2,R2       ;No, negate num
POSITIVE:
     MOVW     R2,ABS      ;absolute value is
                            in R2
     $EXIT_S              ;System call to return
                            control
     .END     ABSOLUTE    ;directive
```

The convention used in this assembly language is:

- All labels end with a colon(:)
- All directives start with a period(.)
- Any character string after a semicolon(;) on a line belongs to the comment field
- In the operation field, a system macro begins with the dollar sign ($)

On an MIPS computer, an example of reduced instruction set computer (RISC) architecture, the same program may take the following form:

```
       .data                # data section
num:   .word    −35
abs:   .word    0
       .text                # code section
main:                       # starts execution
                              at main.
       la       $s1, num    # load address of
                              number1 into $s1.
       lw       $t1, 0($s1)
       bge      $t1, $0, store
       neg      $t2, $t1
store: sw      $t2, 4($s1)  #put absolute into
                              memory location
                              number2.
       li       $v0, 10     # syscall code 10 is
                              for exit.
       syscall              # make the syscall.
```

The convention used in this assembly language is similar to that of the VAX, except instead of a ";" the symbol "#" is used to mark the beginning of a comment field and a macro does not begin with a special symbol.

Table 1. Example of address modes used in the VAX-11

Addressing mode	Assembler symbol	Effect	Example
Register	Rx $x = 0{-}14$	Operand is in the register named	ADDL R1,R2
Register deferred (indirect)	(Rx) $x = 0{-}14$	The memory address of the operand is in Rx	ADDL (R1),R2
Immediate (literal)	#constant	Operand is specified directly as a constant following the symbol #	ADDL #14,R1
Autoincrement	(Rx)+ $x = 0{-}14$	Same as register deferred except that the contents of the register are incremented after it is accessed	ADDL (R1)+,R2
Autoincrement deferred	@(Rx)+ $x = 0{-}14$	Same as autoincrement except the contents of Rx are used as a memory address of the operand	ADDL @(R1)+,R2
Autodecrement	−(Rx) $x = 0{-}14$	Same as autoincrement except that the contents of the register are decremented BEFORE the value is used	ADDL −(R2),R5
Displacement	displ(Rx) $x = 0{-}14$	The value represented by displ is added to the contents of Rx to form the effective address	ADDL 200(R3),R9
Displacement deferred	@displ(Rx)	Same as displacement except that the address calculated is the location of the effective address of the operand	ADDL @20(R3),R9
Relative	displacement	Same as displacement except that instead of using a general register, the program counter (R15) is used	ADDL X,R1
Relative deferred	@displacement	Same as displacement deferred using the program counter instead of the general register	ADDL @X,R1
Index	baseaddress[Rx]	The contents of the index register Rx are added to the base address, which can be specified using any of the above-mentioned addressing modes	ADDL X[R2],R6
Absolute	@#absoluteaddr	The memory address specified is the physical memory address independent of the program location	ADDL @#1000,R5

ADDRESSING MODES

As mentioned above, the assembly language is basically a symbolic representation of the machine language. As machine languages become more complicated, additional symbols are introduced to represent the new features in the machine language. One of the first features introduced is indexing and the use of the index registers. For example, an assembly language instruction for the IBM 704 may be as follows:

```
ADD ARRAY(4)
```

The effective memory address is computed by adding the contents of index register to the memory address represented by the symbol ARRAY. In the 1970s, as more complex instruct sets were devised to match the higher level language constructs, new addressing modes were also introduced. A machine such as the VAX-11 has no less than 12 different addressing modes. Table 1 summarizes those addressing modes and how they are represented in the VAX-11 assembly language.

Although each assembler may choose to use different symbols to represent different addressing modes, what is used in the VAX-11 assembly language is typical. With the emphasis on simplifying the instruction set, modern computers usually also have fewer addressing modes than that of the VAX-11. The Itanium has, for example, only four addressing modes: immediate, register, register indirect, and autoincrement.

DIRECTIVES (PSEUDO-OPERATIONS) AND PSEUDO-INSTRUCTIONS

Directives, also known as pseudo-operations in the early days of assembly language programming, are statements in assembly language that direct the assembler to perform certain functions during the assembly process. They may reserve some memory locations needed by the program or assign a constant value to a particular memory location. Directives do not produce executable machine language code. Some may mark the beginning of a program and data segment. In the examples above, .ENTRY, .END, .BLKW, .word, .data etc. are typical examples of directives.

In an attempt to limit the size of the instruction set, modern computers tend not to include an instruction if its function can be carried out by other instructions in the set. Although the practice has the advantage of simplifying the design of the processor, it makes assembly language less intuitive. As a result, the assembly languages of these machines often include pseudo-instructions, which have no corresponding machine language instructions, to facilitate programming and place the burden of translating these instructions to real machine language instructions to the assembler (Fig. 3). For example, in the MIPS example above, the instructions la (load address), abs (absolute value), and li (load immediate) are pseudo- instructions. The assembler translates each of these instructions to one or more regular assembly language instructions and then generates the machine code. For example li is converted to

```
lui     $17, 4097 [num]    ; la     $s1, num      # load address of number1
                                                  # into $s1.
lw      $9, 0($17)          ; lw     $t1, 0($s1)
addu     $10, $0, $9       ; abs    $t2, $t1
bgez    $9, 8
sub     $10, $0, $9
sw      $10, 4($17)         ; sw     $t2, 4($s1)  # put absolute into memory
                                                  #  location number2.
ori     $2, $0, 10         ; li     $v0, 10       # syscall code 10 is for exit.
syscall                    ; syscall              # make the syscall.
```

Figure 3. Examples of pseudo-instruction conversion: The source program on the right (after the semicolon on each line) is translated to that on the left. Note that the pseudo-instruction abs generates three instructions.

ori (OR immediate) and abs is converted to a sequence of three instructions.

MACROS

Another powerful feature incorporated in many assemblers is macro (also referred to as open subroutine in early literature). Assembly language programmers frequently need to repeat similar segments of code at various sites in a program. In such cases, the segment of code can be defined as a macro; then, during the assembly time, wherever the name of the macro appears in the program, the assembler replaces the name by the statements within the macro. This replacement is termed *macro expansion*.

To add flexibility and generality to macros, most assemblers allow macros to have (dummy) parameters. When invoking a macro, the programmer specifies corresponding arguments to replace the parameters in the macro. This causes the assembler to generate different code for different invocations of the macro.

A macro is similar to a subroutine because it assigns a name to a sequence of statements. However, several important differences exist between a macro and a subroutine.

1. A macro is invoked at *assembly time*. A subprogram is invoked at *execution time*.
2. When a macro is invoked, the *assembler* substitutes the macro body for the macro call. When a subroutine is invoked, the computer *hardware* transfers control to the beginning of the subroutine; upon completion, the hardware transfers control back to the calling program.
3. As many copies of the macro body exist in the object code as calls to the macro. Only one copy of the subprogram exists in the object code, regardless of the number of calls to the subprogram.

In the examples above, $EXIT_S in the VAX-11 program and *syscall* in the MIPS program are macros that are part of the system library.

BIBLIOGRAPHY

1. D.J. Wheeler, Programme organization and initial orders for the EDSAC, *Proc. Roy. Soc. (A)* 202: 573–589, 1950.
2. M. Campbell-Kelly, Programming the EDSAC: Early Programming Activity at the University of Cambridge, *Annals History Comput.*, **2** (1): 4–48, 1980.

FURTHER READING

No lack of text books on assembly languages, exists often under the title Computer Organization and Programming. The following are a few examples:

El-Asfouri, Johnson, and King, *Computer Organization and Programming* Reading, MA: Addison Wesley, 1984.

C.W. Gear, *Computer Organization and Programming*, 3rd edition. New York: McGraw-Hill, 1980.

K.R. Irvine, *Assembly Language for Intel-Based Computers*, 5th edition. Englewood Cliffs, NJ: Pearson Prentice Hall, 2007.

D.H. Stabley, *Logical Programming with System/360*, New York: Wiley, 1970.

Dr. Willis King
University of Houston
Houston, Texas

AUTONOMOUS DECENTRALIZED SYSTEMS

Autonomous decentralized systems (ADS) are distributed computing systems, each of which is composed of subsystems with autonomy to control itself and coordinate with other subsystems. The ADS are constructed on the basis of the ADS concept; that is, a system is treated as result of integration of autonomous subsystems with the objective of resolving online property of online expansion, fault tolerance, and online maintenance (1,2). The data field (DF) architecture for the ADS makes each subsystem autonomous (3,4). Each autonomous subsystem includes its own management system, an autonomous control processor (ACP), and application software modules. Subsystems are connected mutually only through the DF. The ACP broadcasts data together with a content code, which is defined uniquely based on content of data, and it independently selects to receive data based on the content code (content-code communication). The ACP executes an application software module upon receiving all necessary data for the module (data-driven mechanism). Under the DF architecture, subsystems need not know the direct relation with others for communication and execution and need not inform others upon its addition to or deletion from the system. Then the subsystem can be constructed, modified, added, and deleted during operation of the other subsystems (online expansion). The subsystem independently checks to select correct data from multiple data with the same content code sent from the replicated subsystems to the DF. This independent checking mechanism enables each subsystem to prevent the propagation of faults from occurring in other subsystems (fault tolerance). In the DF, data are broadcasted with a test flag as well as with the content code. The ACP selects to receive data by the content code and to change subsystem operation mode from online to test when received data attaches a test flag. It starts test execution using test data, but it does not output executed result data to devices. Online and test modes coexist in the system, but the test subsystem does not interrupt online subsystems (online maintenance). The ADS concept is applied to the various fields of technologies (5,6), such as networks, including Internet, communication, multicomputers, software, control, and robotics, and they have been realized in application systems such as transportation, factory automation, office automation, and telecommunication. In these applications, the ADS improve lifecycle cost, software productivity, flexibility, and adaptability.

ADS CONCEPT

The cost-reduction constraints on computing resources, including networks, has lessoned the requirement for efficient utilization but has raised the need for making them easy to use and easy to construct. Computing systems increasingly have been required to be adaptable to applications (7). For these reasons, the ADS has the objective of meeting the following requirements of online property:

1. Online expansion. As system size increases, its step-by-step construction and expansion should be possible without stopping whole system operation.
2. Fault tolerance. Even if part of the system fails, the system should be able to continue operation without fault propagation.
3. Online maintenance. Maintenance and test procedures should be possible without suspending the system operation.

Systems requiring the online property have the following attributes:

1. The system always has faulty parts.
2. It changes constantly alternating among operation, maintenance, and expansion.
3. It hopes to accomplish its objective and function almost completely.

That is, the system is defined under the following standpoints:

1. Being faulty is “normal.”
2. The system is result of integration of subsystems.

On this standpoint, the system is called the autonomous decentralized system if the following two properties are satisfied as such a living thing, which is composed of largely autonomous and decentralized subsystems.

1. Autonomous controllability. If any subsystem fails, is repaired, and/or is newly added, the other subsystems can continue to manage themselves and to perform their own responsible functions.
2. Autonomous coordinability. If any subsystem fails, is repaired, and/or is newly added, the other subsystems can coordinate their individual objectives among themselves and it can operate in a coordinated manner.

These two properties assure online property of the system. Each autonomous subsystem requires its own intelligence to manage itself without directing to or being directed by other subsystems and to coordinate with the other subsystems. To realize an autonomous decentralized system with two properties, each subsystem is required to satisfy the following conditions:

1. Equality. Each subsystem is equal in function. No master–slave relation exists among subsystems.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

2. Locality. Each subsystem manages itself and coordinates with others based only on local information.
3. Uniformity. Each subsystem is uniform in structure and self-contained so that it manages itself and coordinates with others.

DF

The ADS is realized under a DF architecture with no central operating or coordinating system. Each subsystem has its own management system, the ACP to manage itself and to coordinate with the others. The subsystem includes application software modules and an ACP called "Atom." All subsystems are connected only through the DF (Fig. 1); all data are broadcasted into the DF as messages. The DF in the Atom is called the Atom data field (ADF). Individual data include a content code uniquely defined based on the content of the data. A subsystem selects to receive a message on the basis of content code (content-code communication) (Fig. 2). The sender need not point out the receiver's address. Physically, the DF corresponds to a network or memory. In the network, the broadcast message physically is deleted by its originating subsystem or the terminator in the network after the message is transmitted over entire system. When the DF corresponds to the memory, the first-in–first-out (FIFO) memory is used and messages in memory are deleted after all subsystems check whether to accept it or not. This content-code communication enables each subsystem to be autonomous in sending and receiving data. That is, subsystems need not know the relationship among sources and the destinations. This feature of the content-code communication ensures the locality of information necessary for each subsystem.

DATA-DRIVEN MECHANISM

The application software module in the subsystem starts execution after all necessary data are received (data-driven mechanism). This mechanism loosely couples modules. Each subsystem independently judges and controls its own execution. Required content codes for application software modules are preregistered in the ACP, which can dynamically assigns content codes based on changes in application software modules. The subsystem need not inform other subsystems if content codes assigned to the ACP are changed.

Each ACP has functions of managing the data, checking the data, and supporting the test and diagnosis (Fig. 3). The function of the application software module is characterized by the relation between the content codes of the input data and the output data. The data-driven mechanism is realized by the following two management modules of the ACP.

DF Management

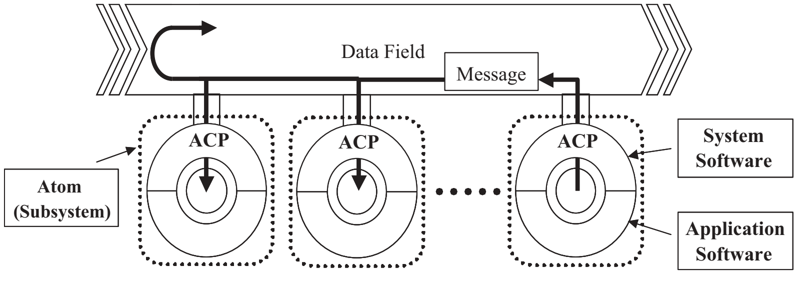
The DF management module acts as the interface between the DF and the ADF. The ADF includes the table of the relationship between application software modules in the Atom and content codes required to execute each application software module. According to registered content codes, the DF management module receives the data from the DF and stores it in the corresponding area in the ADF. The data that originates within the Atom is broadcasted into the DF by this management module.

Execution Management

The execution management module monitors the ADF. As soon as all the data necessary to the application software module is received in the ADF by the DF management module, the execution management module drives the application software module. With this execution management module, application software modules run asynchronously and freely. This autonomous execution property ensures that the application software module cannot be directed to execute by any other application software module so that it can continue its operation even in the event of fault occurrence, expansion, and maintenance of the other application software modules.

ADS TECHNIQUES

The online property of the system is resolved by following three techniques.



ACP: Autonomous Control Processor

Figure 1. DF architecture.

F	CC	SA	C	Data	CRC	F

F: Flag

CC: Content Code

SA: Sender Address

C: Control Code

CRC: Cyclic Redundancy Check

Figure 2. Message format.

Online Expansion

In the Atom-level expansion, application software modules are newly installed in or moved to others. They need only to register necessary content codes in their own ACPs and do not need to inform others. This local generation within the Atom requires no application software modules or subsystems to be revised and needs no interrupt operation. In the system-level expansion, different ADSs are integrated into one. Two types of systems integration are designed (Fig. 4). In the first type, different systems are combined into one system in which all data from systems are broadcasted to the combined DF (Fig. 4). In the second type, different systems are connected by a gateway, which selects the data to be passed through based on content codes (Fig. 4). The ACP of the gateway registers content codes necessary for the system on DF-A to pass through from DF-B, and data to be passed to DF-B from DF-A. During the registration of content codes in the gateway ACP, the subsystems need not stop operation.

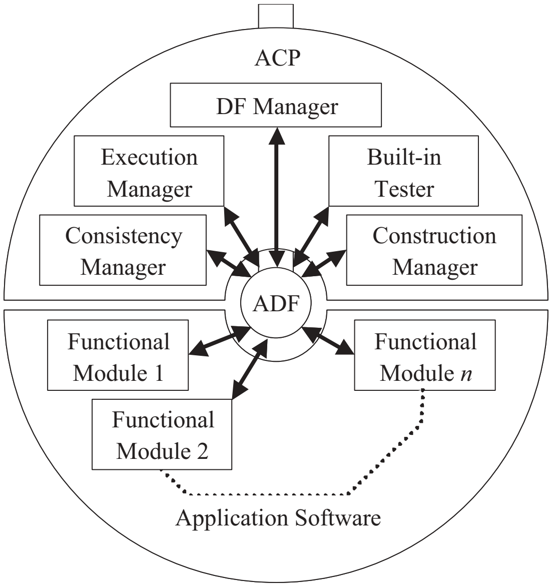


Figure 3. Modular software structure.

Fault Tolerance

DF architecture and its data-driven mechanism ensure that application software modules run freely and asynchronously. When fault tolerance of application software module is required, the module is replicated. Replicated application software modules run independently and send out processed results with the same content code to the DF. Faulty data are also sent out to the DF. The ACP in each subsystem receives all data with the same content code from replicated modules and selects the correct data from the “same” data (Fig. 5). Here, the data consistency management module in the ACP identifies the “same” data both by content code and by event number induced with the data. This event number is located in the message. The event number is set originally at the module receiving the information from an external source via input devices such as sensors and terminals. Although application software modules process data successively, the original event number is preserved in these processes. The “same” data with the same content code and event number is collected from the DF within a predetermined time interval or when it reaches a predetermined number. Correct data are selected from among the “same” data through majority voting logic flexibly adapted to correspond to the predetermined time interval or to the total number of received data. Under this logic, fault occurrence is detected and each application software module avoids being affected by fault propagation. After fault detection, faulty application software modules are recovered. In DF architecture, a subsystem with a replicated module can intercept any data broadcast from other replicated modules. Even if the subsystem includes a faulty application software module, it detects the internal faults via this interception. If an application software module is faulty, the subsystem continues operation by using correct data received from the other replicated application software module, not using its generating data. This recovery does not stop the entire system.

This data consistency mechanism ensures fault tolerance and easily can be adapted to system reconfiguration without stopping the operation.

Online Maintenance

DF architecture makes it easy for application software modules to be tested while the system is operating. An online test is supported by a BIT (built-in tester module) in

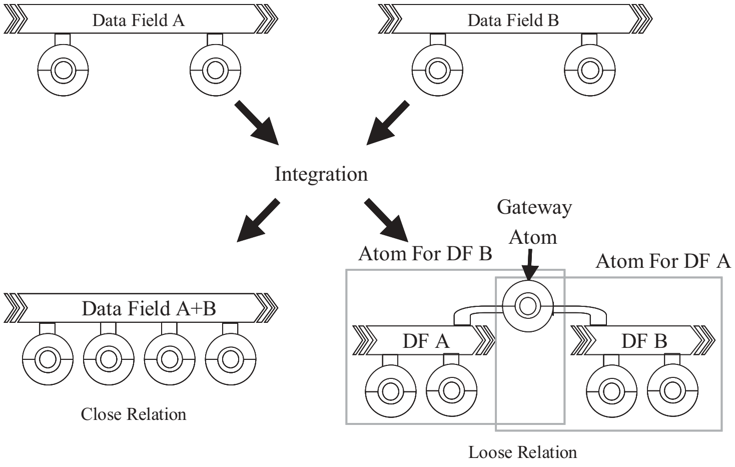


Figure 4. Online expansion.

each ACP and by an EXT (external tester module) as the application software module (Fig. 6). The BIT module in the subsystem sets its application software module in the test mode, generates test data, and checks test result. The application software module being in test mode receives data from the DF and processes it. It broadcasts test result data with a test flag to the DF. The BIT of the system in test mode prevents the signal from being sent to output devices such as controllers. Test result data are used successively to test other application software modules. The EXT monitors test data and test result data in the DF. By correlating test data with test result data, the EXT checks the fault occurrence in the application software module in the test mode and broadcasts the fault detection. The EXT also detects how a fault propagates among modules by monitoring these data. The BIT independently decides whether to change the test mode to online mode based on test results. This test mechanism makes it possible for both online and test modes to coexist in the system.

Software Productivity

In addition to evaluating online property, the ADS improves software productivity. Input and output data only via the DF encapsulated the application software module and has no direct relationship to other modules. The data-driven mechanism need not have the linkage among modules. Environment generation for the application software module only registers necessary content codes in its ACP. These software features make it possible to produce the application software module independently of other modules. The relationship between input data of module and output data of other modules generates data flow among the modules. In the design phase, incorrectness of data flow such as an infinite loop among the application software modules or incompleteness of modules for generating data is checked. Fault propagation is detected by using the data flow. Based on the analysis of data flow, modules on the critical path of the data flow are

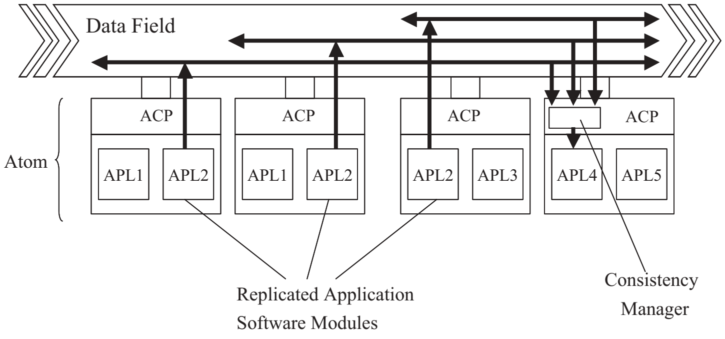


Figure 5. Fault tolerance.

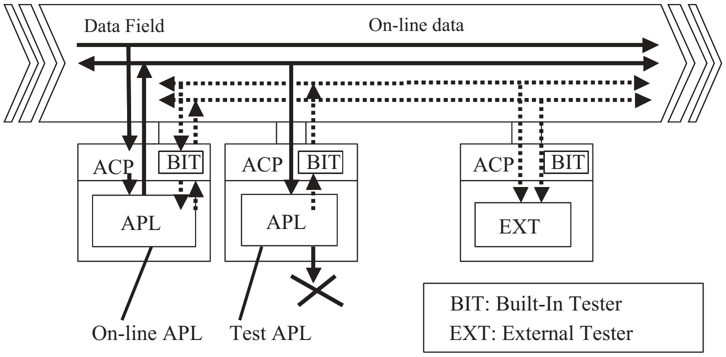


Figure 6. Online maintenance.

replicated to attain fault tolerance. This distributed software development helps to improve the productivity, especially for the software design and testing in a building-block manner.

APPLICATION

The ATOS of the Tokyo metropolitan-area railway system covers 23 train lines and 289 stations on these lines. The total line length is approximately 1100 km. This system serves around 14 million passengers per day. Train service runs about 22 hours a day and is running nonstop throughout the year. The minimum interval between trains at rush hour is 2 minutes. Recently the service types of train traffic have been increasing for the passengers. For example, there are the trains through the specially arranged routes across several different train lines, which is event-related train service not listed in the standard timetable (8).

One requirement for this system is that the construction takes place step by step without stopping the train service and without disrupting the operation of the current-installed parts of the system. This system has been developed over 10 years, and some of the current parts gradually will be replaced even before the entire system construction is completed. This system continuously is evolving and will continue to expand.

Figure 7 shows the structures of the overall system and the station subsystem. Each computer is equipped with the ACP to achieve the online property. The networks for

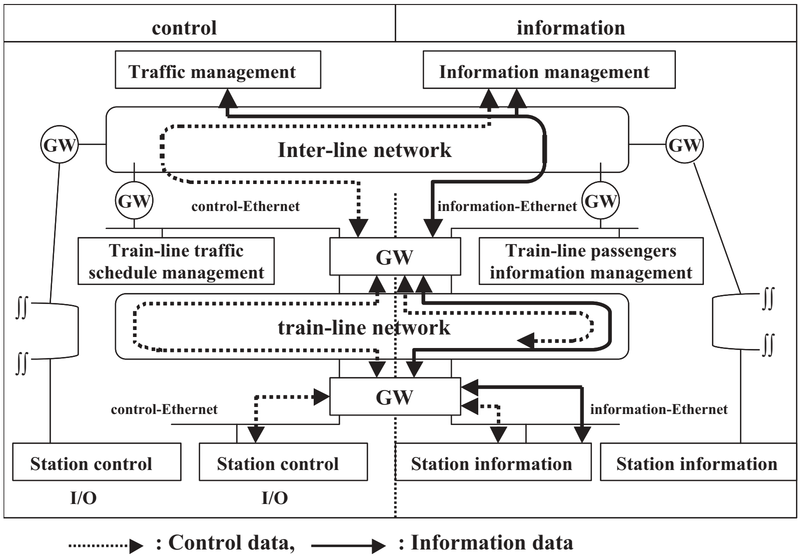


Figure 7. System structure of ATOS.

connecting among the station subsystems in the train line and among the train lines are used for both the control and the information missions. In the station subsystem, the computers for the control and the information are divided and connected by their own mission-oriented networks, the control Ethernet, and the information Ethernet, through the gateway.

The total system is composed of one interline network and 17 train-line networks. The system-wide traffic management subsystem produces the train-traffic schedules and monitors the traffic, and the system-wide information management subsystems supply the traffic information services, which are connected by an interline network. The system for one train line is composed of the station subsystems, a train line, traffic schedule management subsystem, and a train-line passenger's information service management subsystem. The train-line traffic schedules management subsystem distributes the train-line traffic schedules to the train station subsystems, monitors positions of the trains in the train-line, and makes the minor changes in the schedules. In the train-line traffic schedule management subsystem, the control Ethernet connects the train-line traffic reschedule management computer and the maintenance management computer. The train-line traffic reschedule management computer is replicated for fault tolerance. The train-line passengers' information service management subsystem connected to the interline network includes several computers connected by the information Ethernet. Each subsystem is composed of several computers connected by the Ethernet. In the system, the communication uses the ADS content-code communication protocol in the UDP mode. The bandwidth of the common network is divided into control and information missions.

FUTURE TREND OF ADS

The ADS concept and technologies have been applied in various fields of transportation, factory automation, utility management, satellite on-board control, newspaper printing factory, information services, e-commerce, community service, and so on. Most of them use the ACP as middleware run on standard operating systems, such as Windows and UNIX, and on a standard transmission protocol of TCP/IP. Some ADS technologies were approved to be the de facto standard of the ODVA (Open DeviceNet Vendor Association) in 1996, the Factory Automation System in Japan in 2000, the BAS (Building Automation System) in Japan in 2000, and the OMG (Object Management Group) in 2000.

BIBLIOGRAPHY

1. K. Mori, S. Miyamoto, and H. Ihara, Proposition of autonomous decentralized concept, *Trans. IEE of Japan*, 104C, (12) 303–340, 1984.
2. K. Mori, Autonomous decentralized systems: concepts, data field architecture and future trends, *IEEE Proc. of ISADS93*, 1993, pp. 28–34.
3. K. Mori, H. Ihara, Y. Suzuki, K. Kawano, M. Koizumi, M. Orimo, K. Nakai, and H. Nakanishi, Autonomous Decentralized Software Structure and its Application, *IEEE Proc. of FJCC86*, 1986, pp. 1056–1063.
4. H. Ihara and K. Mori, Autonomous Decentralized Computer Control System, *IEEE Computer*, **17** (8): 57–66, 1984.
5. S. Yau and G. H. Oh, An object-oriented approach to software development for autonomous decentralized systems, *IEEE Proc. of ISADS93*, 1993, pp. 37–43.
6. K. H. Kim and C. Subbaraman, Interconnection schemes for RTO.k objects in loosely coupled real-time distributed computer systems, *IEEE Proc. of COMPSAC97*, 1997, pp. 121–128.
7. K. Mori, Expandable and fault tolerant computers and communications systems: autonomous decentralized systems, *IEEE Proc. of ISCC99*, 1999, pp.228–234.
8. K. Mori, Trend of autonomous decentralized systems, *IEEE Proc. of FTDCS04*, 2004, pp.213–216.

KINJI MORI
Tokyo Institute of Technology
Tokyo, Japan

CAPABILITY MATURITY MODELS (CMM)

INTRODUCTION

Today software is a major asset of many companies. Research and development investment primarily goes into software development for a majority of applications and products. To stay competitive with software development, many companies are putting in place improvement initiatives of their key processes that are generally engineering processes first. Often the improvement programs also include a broader reengineering perspective.

Strengthened process capability is key. If you do not know where you are and where you want to go, change will never lead to more added value for your business. Effective process improvement is achieved using the well-known capability maturity models (CMM and from now on CMMI as CMM has been sun set as of the end of 2005). This model provides a framework for process improvement and is used by many software-intensive development organizations; software could be the entire system or only one component, but the advantage of the newly promoted CMMI is that multiple disciplines can be addressed: software, systems, hardware, and services. The maturity model defines five levels of maturity plus an improvement framework for process maturity and, as a consequence, quality and predictability.

This model must be combined with a strong focus on business objectives and metrics for follow-up of change implementation. Otherwise, the main risk is to focus on processes exclusively and lose track of what is essential for customers and shareholders.

Model-based process improvement involves the use of a model to guide the improvement of an organization's processes. Process improvement grew out of the quality management work of Deming, Crosby, and Juran and is aimed at increasing the capability of work processes. Essentially, process capability is the inherent ability of a process to produce planned results. As the capability of a process increases, it becomes predictable and measurable, and the most significant causes of poor quality and productivity are controlled or eliminated.

Models provide a common set of process requirements that capture best practices and practical knowledge in a format that can be used to guide priorities. By using a model, organizations can modify or create processes using practices that have been proven to increase process capability.

THE ORIGIN

In 1986, Watts Humphrey, the SEI, and the Mitre Corporation responded to a request by the U.S. Federal Government to create a way of evaluating the software capability of its contractors. The group used IBM's concepts to create a software maturity framework, a questionnaire, and two appraisal methods. Over the next few years, this work was continued and refined.

In 1991, the SEI published the CMM for Software version 1.0, a model that describes the principles and practices underlying software process maturity. The CMM is organized to help software organizations improve along an evolutionary path, growing from an ad hoc, chaotic environment toward mature, disciplined software processes. The CMM was used and evaluated for two years and then revised and released as version 1.1 in 1993.

A similar revision was planned for 1997 as version 2.0; this version was developed but never released as an independent model. However, the proposed revision was used as the source for the CMMI integration effort.

The Software CMM (SW-CMM) focused primarily on process management. Among the "key process areas" in the model, only one, "Software Product Engineering," specifically targets the core engineering tasks, which range from the analysis of software requirements to software design, coding, integration, and testing at the concluding end. All other SW-CMM key process areas were written such that they could easily be applied to development work other than software. Along with the success of the SW-CMM in improving software development, this flexibility may explain the interest in applying the CMM concepts to disciplines beyond software; much of what was believed to be good with the SW-CMM had a utility that was not restricted to just the software area.

The five maturity levels of CMM and their respective impact on performance are described in Table 1.

To understand why and how the "CMMI project" has been started at the SEI, it is probably useful to have a quick look at the various sources that were available at the end of the 1990s. The various models in use were the following:

MODELS DESCRIPTION

SW-CMM: The original CMM developed at the Software Engineering Institute (SEI). In 1986, the SEI, with assistance from MITRE Corporation, began developing a process maturity framework intended to assist organizations in improving their software processes. In 1987, the SEI released a brief description of the process maturity framework and a maturity questionnaire (CMU/SEI-87-TR-23). The fully developed model (version 1.1) was released in 1993.

SE-CMM The Systems Engineering Capability Maturity Model (SE-CMM) describes the elements of an organization's systems engineering process that are essential to good systems engineering. This model was developed by the Enterprise Process Improvement Collaboration (EPIC), which included industry, government, and academic members. It was merged in 1998 with the INCOSE SECAM to form the Electronics Industry Alliance's EIA 731.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Table 1.

CMM Level	Title	Focus	Key Process Areas
5	Optimizing	Continuous process improvement on all levels	Process change management Technology change management Defect prevention
4	Managed	Predictable product and process quality	Quality management Quantitative process management
3	Defined	Standardized and tailored engineering and management process	Organization process focus Organization process definition Training program Integrated project management Software product engineering Inter-group coordination Peer reviews
2	Repeatable	Project management and commitment process but still highly people-driven	Requirement management Software project planning Software project tracking and oversight Software quality assurance Software configuration management Software subcontract management
1	Initial	Heroes and massive effort with chaotic results	

SA-CMM: A collaborative effort among the U.S. Department of Defense, the SEI, industry, and other U.S. government agencies, the Software Acquisition Capability Maturity Model (SA-CMM) supports benchmarking and improvements of the software acquisition process.

People CMM: This model addresses the ability of software organizations to attract, develop, motivate, organize, and retain competences and good skills.

IPD-CMM: The Integrated Product Development CMM (IPD-CMM) was published only in draft form.

FAA-iCMM: The first completed attempt at integration; this model developed at the U.S. Federal Aviation Administration (FAA) integrates material from many sources: the SE-CMM, the SA-CMM, the SW-CMM, EIA 731, Malcolm Baldrige, ISO/IEC 15504, ISO/IEC 15288, and ISO/IEC 12207. Now at Version 2, it is being used as a unified means of guiding process improvement across the entire FAA. It has also been adopted by Aviation Authorities in Europe.

ISO/IEC 12207: An international standard on software life-cycle processes; it was first issued in 1995 and amended in 2002. ISO is the International Organization for Standardization.

ISO/IEC 15504: A draft international standard that defines the requirements for performing process assessment as a basis for use in process improvement and capability determination. This initiative started in 1992 and changed several times the scope of the required part versus the informative part of the target standard. CMMI is fully compliant with ISO/IEC 15504 requirements.

Fundamentally, process improvement integration has a major impact in four areas: cost, focus, process integration, and flexibility.

By applying a single model, organizations that would otherwise use multiple models can reduce the cost of notably training, appraisals, and maintenance of redundant process assets.

An integrated process improvement program can clarify the goals and business objectives of the various initiatives. By integrating process improvement activities across a wider range of disciplines, it becomes easier to rally the troops to the process improvement banner.

A final benefit provided by integration is the ability to add disciplines as the business or engineering environment changes.

CMMI OVERVIEW

The CMMI Product Suite contains an enormous amount of information and guidance to help your organization improve its processes.

1. Materials to help you evaluate the content of your processes—information that is essential to your technical, support, and managerial activities.
2. Materials to help you improve process performance—information that is used to increase the capability of your organization's activities.

This integration was intended to reduce the cost of implementing multidiscipline model-based process improvement by

- Eliminating inconsistencies.
- Reducing duplication.
- Increasing clarity and understanding.
- Providing common terminology.
- Providing consistent style.
- Establishing uniform construction rules.
- Maintaining common components.
- Assuring consistency with ISO/IEC 15504.
- Being sensitive to the implications for legacy efforts.

The project milestones between 1997 and 2002 were as follows:

1997 CMMI initiated by U.S. Department of Defense and NDIA.

1998 First team meeting held.

1999 Concept of operations released; first pilot completed.

2000 Additional pilots completed.

CMMI-SE/SW version 1.0 released for initial use.

CMMI-SE/SW/IPPD version 1.0 released for initial use.

CMMI-SE/SW/IPPD/SS version 1.0 released for piloting.

2002 CMMI-SE/SW version 1.1 released.

CMMI-SE/SW/IPPD version 1.1 released.

CMMI-SE/SW/IPPD/SS version 1.1 released.

CMMI-SW version 1.1 released.

Since August 2006, a CMMI V1.2 has been available for use. The new version includes simplifications, reduction of the number of practices to implement, and restructuring, which were recommended by users (Table 2).

The fundamental organizational feature of all CMMI models is the "process area."

Any process improvement model must include a scale relating to the importance and role of the materials contained in the model. In the CMMI models, a distinction is drawn among the terms "required," "expected," and "informative."

The sole *required* component of the CMMI models is the "goal." A goal represents a desirable end state, the achievement of which indicates that a certain degree of project and process control has been achieved. Each process area has between one and four specific goals; the entire CMMI-SE/SW/IPPD/SS model (version 1.1) includes a total of 55 specific goals.

Examples include: Requirements Management REQM SG 1: Requirements are managed, and inconsistencies with project plans and work products are identified.

Project Monitoring and PMC SG 2: Corrective actions are control managed to closure when the project's performance or results deviate significantly from the plan.

In contrast to a specific goal, a generic goal has a scope that crosses all process areas. Generic goals are characteristics of the maturity. Consider, for example, GG 2: "The process is institutionalized as a managed process." In the CMMI glossary (CMMISE/ SW/IPPD/SS, version 1.1, Appendix C), a "managed process" is a performed process that is planned and executed in accordance with policy; employs skilled people having adequate resources to produce controlled outputs; involves relevant stakeholders; is monitored, controlled, and reviewed; and is evaluated for adherence to its process description.

The only *expected* component of the CMMI models is the statement of a "practice." A practice represents the "expected" means of achieving a goal. Every practice in the CMMI models is mapped to exactly one goal. Between two and seven specific practices are mapped to each specific goal; the entire CMMI-SE/SW/IPPD/SS version 1.1 model includes a total of 189 specific practices, which are mapped to the 55 specific goals.

In contrast to a specific practice, a generic practice has a scope that crosses all process areas. For example, one generic practice that is mapped to the generic goal to institutionalize a managed process (GG 2) addresses the training of people. Consider GP 2.5: "Train the people performing or supporting the process as needed."

Table 2. Description of maturity levels and process areas in V1.1

Level	Focus	Process Area
5: Optimizing	Continuous process improvement	Causal analysis and resolution Organizational innovation and deployment
4: Quantitatively Managed	Quantitative management	Quantitative project management Organizational process performance
3: Defined	Process Standardization	Organizational process focus Organizational process definition Organizational training Integrated project management Risk management Decision analysis and resolution Requirements development Technical solution Product integration Verification Validation
2: Managed	Basic project management	Requirements management Project planning Project monitoring and control Measurement and analysis Process and product quality assurance Configuration management Supplier agreement management
1: Initial		

CMMI models contain 10 types of *informative* components. The major ones are as follows:

Purpose. Each process area begins with a brief statement of purpose for the process area.

Reference. Explicit pointing from one process area to all or part of another process area is accomplished with a reference.

TypicalWorkProducts. When a practice is performed, there will often be outputs in the form of work products.

Subpractices. For many practices in the CMMI models, subpractices provide a decomposition of their meaning and the activities that they might entail as well as an elaboration of their use.

Discipline Amplifications. One of the most distinctive aspects of CMMI as compared with prior source models is the fact that the CMMI model components are discipline-independent. To maintain the usefulness of the discipline-specific material found in its source models, CMMI provides discipline amplifications that are introduced with phrases such as "For software engineering" or "For systems engineering."

Amplifications are informative material, so they are not required in an appraisal.

The move to V1.2 includes the following changes specified in Table 3.

A NEW REPRESENTATION: THE CONTINUOUS REPRESENTATION

One source model for CMMI, the SW-CMM, was a "staged" model. Another source model, the Systems Engineering Capability Model, was a "continuous" model.

A staged model provides a predefined road map for organizational improvement based on proven grouping and ordering of processes. The term "staged" comes from the way that the model describes this road map as a series of "stages" that are called "maturity levels." Each maturity level has a set of process areas that indicate where an organization should focus to improve its organizational process.

We have already emphasized that the key process areas at level 2 of the SW-CMM focus on the software project's concerns related to establishing basic project management controls. Level 3 addresses both project and organizational issues, as the organization establishes an infrastructure that institutionalizes effective software engineering and management processes across all projects.

Continuous models provide less specific guidance on the order in which improvement should be accomplished. They are called continuous because no discrete stages are associated with organizational maturity. EIA 731 and ISO/IEC 15504 are examples of continuous models.

In continuous models, the generic practices are grouped into capability levels (CLs), each of which has a definition that is roughly equivalent to the definition of the maturity levels in a staged model.

In a continuous appraisal, each process area is rated at its own capability level. An organization will most likely have different process areas rated at different CLs. The results can be reported as a capability profile.

Continuous models describe improvement through the capability of process areas, singly or collectively. A capability level includes a generic goal and its associated generic practices that are added to the specific goals and practices within the process area. When the organization

Table 3.

Level	Process Area V1.1	Process Area V1.2
5: Optimizing	Causal analysis and resolution	Causal analysis and resolution
	Organizational innovation and deployment	Organizational innovation and deployment
4: Quantitatively Managed	Quantitative project management	Quantitative project management
	Organizational process performance	Organizational process performance
3 : Defined	Organizational process focus	Organizational process focus
	Organizational process definition	Organizational process definition + IPPD practices
	Organizational training	Organizational training
	Integrated project management	Integrated project management + IPPD practices
	Risk management	Risk management
	Decision analysis and resolution	Decision analysis and resolution
	Requirements development	requirements development
	Technical solution	Technical solution
	Product integration	Product integration
	Verification	Verification
	Validation	Validation
	Integrated product and project development	
	Integrated supplier management	
2: Managed	Requirements management	Requirements management
	Project planning	Project planning
	Project monitoring and control	Project monitoring and control
	Measurement and analysis	Measurement and analysis process and product quality
	Process and product quality	assurance
	Assurance Configuration management	Configuration management
	Supplier agreement management	Supplier agreement management + ISM
1: Initial		

meets the process area-specific goals and generic goals, it achieves the capability level for that process area.

Staged models describe the maturity of organizations through successful implementation of ordered groups of process areas. These groups, or stages, improve processes together, based on achievements in the previous stage.

We will not go into detail regarding the differences between these two representations; the discussion might become unclear at this stage, and this is a question to be debated with sponsors of process improvement initiatives: which representation fits the best with their target. Nevertheless, we can reinforce the concept of one model with two views or representations: CMMI provides a mapping to move from the continuous to the staged perspective. For maturity levels 2 and 3, the concept is straightforward and easy to understand. If an organization using the continuous representation has achieved capability level 2 in the seven process areas that make up maturity level 2 (in the staged representation), then it can be said to have achieved maturity level 2.

Similarly, if an organization using the continuous representation has achieved capability level 3 in the seven process areas that make up maturity level 2 and the 14 process areas that make up maturity level 3 (a total of 21 process areas in CMMI-SE/SW/IPPD/SS), then it can be said to have achieved maturity level 3.

USING CAPABILITY MODELS

Organizations generally have many different business objectives, such as produce quality products or services, create value for the stakeholders, be an employer of choice, enhance customer satisfaction, increase market share, or implement cost savings and best practices.

To meet any of these objectives, organizations must have a clear understanding of what it takes to produce products or services. To improve, they need to understand the variability in the processes that are followed, so that when adjusted, they will know whether the adjustment is advantageous.

As systems grow more complex, the processes used to develop them will follow suit. The complexity of processes inevitably increases to keep pace with the number of individuals who are involved in performance.

The CMMI Product Suite offers a growing number of multi- and single-discipline models, all developed with integrated process improvement in mind.

The best combination of disciplines for an organization will depend on its business, organization, environment, and process improvement objectives.

For Version 1.1 of the model, four combinations of disciplines are available from which to choose, with an increasing scope of coverage.

The CMMI-SW model covers software engineering (SW); the CMMI-SE/SW model covers both systems engineering (SE) and software engineering; the CMMI-SE/SW/IPPD model adds in integrated product and process development (IPPD); and finally, the CMMI-SE/SW/IPPD/SS model provides additional emphasis on supplier sourcing (SS) and managing suppliers.

In Version 1.2 and the constellation concept, three disciplines are covered: development (CMMI for development), acquisition (CMMI for Acquisition), and Services (CMMI for services). The two last disciplines are not yet officially released but announced for 2007 by the SEI. The three variants will share a core of 16 process areas. It is obvious that the more the CMMI is used, the more attempts are made to fit the needs in the field.

The relevant factors for the selection of CMMI model are as follows:

Core business of the organization: The organization's fundamental activities, business objectives, and organizational culture all influence the choice of the appropriate CMMI model. Ideally, the disciplines chosen are the ones that are most critical to the organization's success.

Organization: It might be helpful to align process improvement plans with any organizational changes that are concurrently under way. The IPPD extensions are particularly useful in change management.

Improvement scope and objectives: Reducing the number of disciplines in which improvement effort is deployed is not good in the long term. Organizations who started with SW-CMM and focused only on software discipline during a couple of years finally move to CMMI just because it will bring benefit to the whole if maturity models concepts are also applied in other disciplines.

Selecting the IPPD extension mentioned above with the CMMI-SE/SW model provides two additional process areas (integrated teaming and organizational environment for integration), plus one expanded process area (integrated project management). Selecting the SS extension of the CMMI-SE/SW/IPPD model provides one additional process area (integrated supplier management). There will be an interest in this model extension if the organization is part of an acquisition-oriented bigger organization or participates in a project where the acquisition of products and services from an external source is a central issue of concern.

APPRAISALS

One part of the CMMI Product Suite that deals with appraisals is the Appraisal Requirements for CMMI (ARC), version 1.1. This document comprises 42 "requirements" that provide a mixed set of requirements and design constraints on an appraisal method. For those who have been used to the CBA-IPI assessment method, there are several variances. The CBA IPI method was a mixture of document's checking and discovery of the reality of projects through interviews with a bigger focus on the discovery aspect when SCAMPI—the CMMI—based appraisal method, Standard CMMI Appraisal Method for Process Improvement, focuses more on verification.

One key principle is the process implementation indicator (PII), a proof that a practice is really implemented.

Table 4.

Characterstics	Class A	Class B	Class C
Amount of objective evidence gathered (relative)	High	Medium	Low
Ratings generated	Yes	No	No
Resource needs (relative)	High	Medium	Low
Team size (relative)	Large	Medium	Small
Appraisal team leader requirements	Lead appraiser	Lead appraiser or person trained and expeienced	Person trained and experienced

In CMMI, PIIs refer to the "footprints" that are the necessary or incidental consequence of practice implementation.

PIIs include artifacts as well as information gathered from interviews with managers and practitioners. There are three PIIs types:

Direct Artifacts: Tangible outputs resulting directly from implementation of a practice (e.g., Typical Work Products).

Indirect Artifacts: Artifacts that are a side effect or indicative of performing a practice (e.g., meeting minutes, reviews, logs, and reports).

Affirmations: Oral or written statements confirming or supporting implementation of the practice (e.g., interviews and questionnaires).

PII-based process appraisal uses PIIs as the focus for verification of practice implementation. This process is in contrast to an observation-based approach (CBA IPI) that relies on the crafting of observations that pertain to model implementation strengths or weaknesses.

Essential SCAMPI method attributes are as follows:

1. *Accuracy*: Level of confidence that the appraisal results reflect the strengths and weaknesses of the assessed organization; i.e., no significant strengths and weaknesses are left undiscovered.
2. *Repeatability*: The degree to which the ratings and findings of an appraisal are likely to be consistent with those of another independent appraisal conducted under comparable conditions; i.e., another appraisal of identical scope will produce consistent results.
3. *Cost/Resource Effectiveness*: Person-hours spent planning, preparing, and executing an appraisal. A reflection of the organizational investment in obtaining the appraisal results.
4. *Meaningfulness of Results*: Usefulness of the results (findings) to the organization in establishing improvement initiatives.
5. Arc Compliance.

CMMI appraisal premises are as follows:

- Goal achievement is a function of the extent to which the corresponding practices are present in the planned and implemented processes of the organization.
- Practice implementation at the organizational unit level is a function of the degree of practice implementation at the instantiation level (e.g., projects).
- The aggregate of objective evidence available to the appraisal team is used as the basis for determination of practice implementation.
- Appraisal teams are obligated to seek and consider objective evidence of multiple types in determining the extent of practice implementation.

Three classes of appraisals have been defined by the SEI. They can be differentiated as in Table 4.

Most organizations use the three classes in combination in order to achieve a better result.

CMMI SCAMPI Class B/Class C Appraisals, as currently defined by the SEI Appraisal Requirements for CMMI, Version 1.1 (ARC), are used primarily to gauge the progress made toward meeting applicable capability or maturity level-specific or generic goals as determined by compliance with the guidance from specific and generic practices, identify remaining gaps, and deciding on plans for future process improvement actions. One way is to structure Class B appraisals to better understand an organization, their business goals, and more importantly, how the organization has interpreted the CMMI to meet specific business and quality goals.

Interpretation of the CMMI goals differs from one organization to the next, even when they are within the same company. A Class B appraisal gives the opportunity to appraise to what extent each of the process areas has been implemented, and how. It is important to gain this understanding and, more specifically, to recognize when and if alternative practices have been implemented. Alternative practices, for some organizations, are an acceptable approach and are usually tied to business goals, types of projects (new development, enhancements/maintenance, etc.), technology in use, and/or organizational cultures. However alternative practices must continue to meet the goals of the process areas.

One primary output of an appraisal is a characterization of practice implementation at the project level (instance of implementation). A practice may be as outlined in Table 5.

This output is based on data collection and aggregation. In performing such activities, two requirements of SCAMPI family have to be looked at:

- *Corroboration*: Must have direct artifacts, combined with either indirect artifact or affirmation.

Table 5.

Fully implemented (FI)	· Direct artifacts present and appropriate · Supported by indirect artifact and/or affirmation · No weaknesses noted
Largely implemented (LI)	· Direct artifacts present and appropriate · Supported by indirect artifact and/or affirmation · One or more weaknesses noted
Partially implemented (PI)	· Direct artifacts absent or judged inadequate · Artifacts or affirmations indicate some aspects of the practice are implemented · One or more weaknesses noted
Not implemented (NI)	· Any situation not covered by above, e.g., insufficient objective evidence to be characterized by one of the above

- *Coverage*: Must have sufficient objective evidence for implementation of each practice, for each instance. Must have face-to-face (F2F) affirmations (avoid "paper only appraisals"):
  - At least one instance for each practice.
  - At least one practice for each instance.
  - Fifty percent of practices for each PA goal, for each project, have at least one F2F affirmation data point.

A SCAMPI appraisal is divided into three phases: 1) initial planning and preparation, 2) on-site appraisal, and 3) reporting of results. Each phase includes multiple steps. The first phase involves analyzing requirements, developing a plan, selecting and preparing the team, obtaining and analyzing the initial objective evidence, and preparing for the collection on site of additional objective evidence.

The second phase focuses on collecting and examining objective evidence, verifying and validating that evidence, making sure that it is adequately documented, and developing the appraisal results (findings and ratings).

The third phase involves the presentation of the final findings to the sponsor, conducting any needed executive briefings, and appropriately packaging and archiving the appraisal assets, including submission of all information needed by the CMMI Steward (SEI).

Only a SCAMPI lead appraiser, who has been trained and authorized by the CMMI Steward, may lead a SCAMPI A appraisal.

To start an improvement program that leads to a full SCAMPI appraisal, an organization could use several Class C appraisals leading up to a Class B appraisal. Process improvements are made based on the findings of the quick looks at parts of the organization. The lessons learned in this way can be used to provide broader organizational improvements.

As improvements indicate that the part of the organization is ready for the next step, a Class B appraisal can be performed and its findings subsequently are used to prepare for the full SCAMPI A appraisal.

USING APPRAISAL RESULTS FOR PROCESS IMPROVEMENT

Software process improvement is a systematic, collaborative, and long-range method to evolve the way software work is organized and performed. Improvement methods include the IDEAL method, which is an integrated approach for PI defined by the SEI. IDEAL identifies five phases: initiating, diagnosing, establishing, acting, and leveraging. Each of these phases is centered on a particular activity:

- Specify business goals and objectives that will be realized or supported (**Initiating**).
- Identify the organization's current state with respect to a related standard or reference model **(Diagnosing)**.
- Develop plans to implement the chosen approach **(Establishing)**.
- Bring together everything available to create a "best guess" solution specific to organizational needs—for example, existing tools, processes, knowledge, and skills—and put the solution in place **(Acting)**.
- Summarize lessons learned regarding processes used to implement IDEAL **(Leveraging)**.

Software capability is a sophisticated mixture of these preceding statements, but some concepts must be considered as part of the software capability.

Capability implies that an organization can learn from the past, especially from mistakes, learn from others, and translate lessons learned into process evolutions. Improvement iteration cycles will only be complete if measurements are defined to quantify the improvement.

When maturity of the organization improves, the standard process changes. Getting from Level 2 to Level 3 implies that all good practices within projects are institutionalized and that an assessment process is in place that will help to identify the best practices across projects, which will be documented into the organization standard software process (OSSP).

The Steps to Implement Process and Tools in an Organization

The OSSP of the CMM and CMMI is a new process in the organization.

If an organization decides to develop an organization-wide environment, a project to develop the organization's development environment has to be initiated. Such a project will work very closely with the software development project teams.

The process implementation project is divided into several phases where all four IDEAL steps are performed in each phase until the project is ready and the process and tools are deployed and successfully used by the entire organization. A process implementation project can be divided into phases. The four phases address:

- *Phase 1*: Sell the process implementation project to the sponsors.
- *Phase 2*: Handle the major risks.
- *Phase 3*: Complete everything—templates, guidelines, and examples of Development Cases are ready, and a training curriculum is in place.
- *Phase 4*: Deploy it to the entire organization.

The recommendations for successfully implementing a process change are to:

- Identify change agents at various levels in the organizations.
- Plan the change in small, reasonable, and measurable steps.
- Communicate the changes using ground-level language appropriate to the level of the organization.

Measurement

Measurement is one common feature in the CMM. A common feature exists for each key process area from Level 2 to Level 5 and indicates when a practice is institutionalized. Metrics are so important that in the CMMI there is a measurement process area.

What is key in the CMM is to measure processes to determine their adequacy at Level 2 and their effectiveness

Table 6. Return-on-investment report: defect phase containment

Over 80% of defects found before Delivery

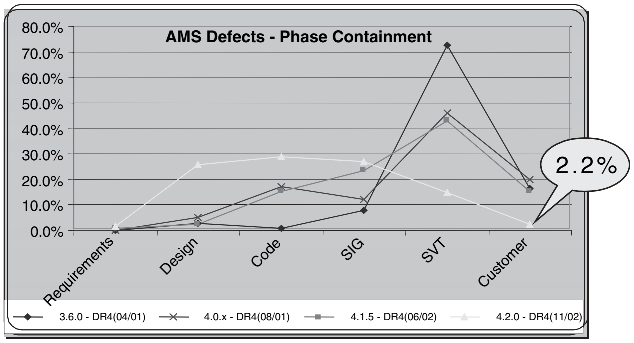
Caputo Impressed by Results
Code Reviews Play a Vital Role in Improvement

By the middle of 2002, the program was coming under more and more pressure to improve field quality and reduce the number of patches required to keep the product running.
The team decided to find ways to improve defect detection and removal during development.

What It Took To Get Defects Removed
Here is a listing of actions taken to improve detect detection and removal with focus on CMM Level 3:
Establishing phase containment programs.

- Increase code review coverage to 100%
- All requirements, design, coding, and test plans are inspected, results are recorded using tool & Compliance is reviewed via score card
- Established error tracking for document views

Defect Removal Results



AMS was able to substantially increase the number of defects found **prior** to SVT (System Verification Test)

- version 4.1.5: 41%
- version 4.2: 81%

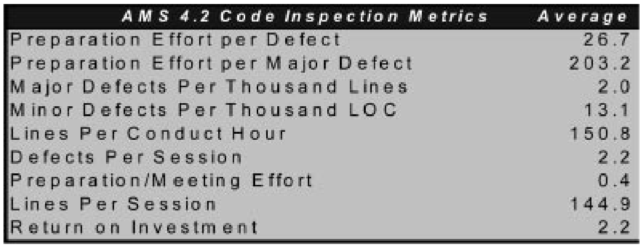
Cost Savings Apparent

Estimates show substantial ROI

Over $600K Saved in One Year
This estimate includes a reduction in the number of patches associated with version 4.2 as compared to version 4.1.5, and the labor associated with correcting bugs in the future.

Number of Patches Declines

As of December 2003, there have only been 3 patches since the release of Release 4.2 in June 2003, as compared to 18 patches for version 4.1.5, totaling over $600,000 cost savings, as a direct result in the ROI of the CMM Level 3 phase containment/ code reviews process improvement program.



AMS 4.2 Code Inspection Metrics	Average
Preparation Effort per Defect	26.7
Preparation Effort per Major Defect	203.2
Major Defects Per Thousand Lines	2.0
Minor Defects Per Thousand LOC	13.1
Lines Per Conduct Hour	150.8
Defects Per Session	2.2
Preparation/Meeting Effort	0.4
Lines Per Session	144.9
Return on Investment	2.2

The return in investment (ROI) is measured by estimating the cost to fix defects if they leak out testing to the customer, divided by the time invested in conducting the code inspections. The 2.2 ROI indicates that 2.2 bug fixing hours were saved for every 1 hour spent in code inspection. This ROI of 2.2 for version 4.2 was an improvement over the 0.9 ROI for version 4.1.5.

at Level 3. The shift from reporting to analyzing and acting on metrics is generally difficult, but a clear sign that the projects are more mature. Small and early successes managing a project quantitatively lead to accepting and understanding the benefits of measurement.

In addition to these general requirements, some key process areas have specific requirements for measures such as software project planning, software project tracking and oversight, and integrated software management. These requirements are related to project data that enable project estimates and project control.

The specific PA "measurement and analysis" of the CMMI integrates project control metrics and measurement made against business goals.

The project measurements artifact stores the project's metrics data. It is kept current as measurements are made or become available. It also contains the derived metrics calculated from the primitive data and should also store information, such as procedures and algorithms, about how the derived metrics were obtained. Reports on the status of the project—for example, progress toward goals (functionality, quality, and so on), expenditures, and other resource consumption—are produced using the project measurements.

PROCESS MANAGEMENT

Successful processes are not static. Processes must be managed. Having defined processes on any level of an organization also asks for process change management. To facilitate change, any process element should refer to a process owner who typically serves as a focal point for change proposals and change decisions. A process owner is an expert for a specific process and guides any type of evolution, improvement, or coaching. Any single instance of a process element should be placed under configuration control, which allows managing change in the context of several parallel projects.

Table 6 gives a good example of ROI calculation following process management.

CONCLUSION

The community of users of maturity models is rapidly growing all over the world. Several indicators can be looked at as follows:

- The variety of organizations adopting the maturity approach: from defense suppliers to the general industry players to banking and services.
- The various countries represented in the SEI training sessions: Europe and the Far East are now well represented.
- The wide audience of the annual SEPG conference and the sister events: European SEPG and Asian SEPG.
- The number of lead instructors and lead appraisers qualified by the SEI.

Process improvement based on maturity models is now reaching the mass market but will remain a competitive advantage for the organizations that adopt it and align process change management with their business strategy.

FURTHER READING

D. M. Ahern, A. Clouse, and R. Turner, *CMMI Distilled*, 2nd ed.Reading, MA: Addison-Wesley, 2003.

B. W. Boehm, Anchoring the software process. *IEEE Softw.*, 73–82, 1996.

L. Carter, et al., The Road to CMMI: Results of the First Technology Transition Workshop, CMU/SEI-2002-TR-007, Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, February 2002.

M. Chrissis, M. Konrad, and. S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement,* Boston: Addison-Wesley, 2003.

CMMI Product Development Team, Standard CMMI Assessment Method for Process Improvement: Method Description (SCAMPI), Version 1.0 (CMU/SEI-2000-TR-009).

CMMI Product Development Team, Assessment Requirements for CMMI (ARC), Version 1.0 (CMU/SEI-2000-TR-011), August 2000.

CMMI Product Team, CMMI Version 1.1, CMMI-SE/SW/IPPD/SS, V1.1 (CMU/SEI-2002-TR-011 and ESC-TR-2002-011), *Improving Processes for Better Products*, Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, March 2002.

R. McFeeley, *IDEAL: A User's Guide for Software Process Improvement*, Pittsburgh, PA: Software Engineering Institute, 1996.

W. S. Humphrey, *Managing the Software Process*, Reading, MA: Addison-Wesley, 1989.

M. Paulk, et al., *Capability Maturity Model for Software, Version 1.1*, Pittsburgh, PA: Software Engineering Institute, 1993.

M. Paulk, et al., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Reading, MA: Addison-Wesley, 1995.

Annie Combelles
DNV
Arcueil, France

CLASS AND OBJECT

INTRODUCTION

Object-oriented technologies have become the main stream in software system development. To improve the quality of software systems, object-oriented methods help software developers produce reusable, extensible, robust, and portable systems. The object-oriented approach encompasses the entire software lifecycle from requirement and design to implementation and testing. Methods and languages, as well as supporting tools, are applied to different stages. Evolving from modular programming and abstract data type (ADT), object orientation separates the interface from its implementation so that the implementation is hidden. The software developer programs to the interface not to the implementation. Instead of designing and developing a software system based on processes and functions, objects become the first-class element. In this way, system decomposition is not based on functionalities in terms of main program and subroutines. It is based on the entities in a system. These entities with their data and operations are encapsulated into objects.

At the foundation of object-oriented technologies, class and object are important concepts. In the rest of this article, we provide a detailed description of these two concepts.

CLASS

With the increasing complexity and scale of software systems, extendibility, reusability, and reliability become important goals in software development. New software components should be easily added in the software systems. Existing components can be reused in different systems. Software systems should also be tolerant to faults. Traditional approaches use functions as a basis for the architecture of software systems. The designer structures the software systems around functions. This top-down functional decomposition is easy to obtain the system architecture initially. However, it is difficult to change the system architecture in the future. Most software systems undergo numerous changes after their first delivery. The model of software development should not only consider the period leading to that delivery but also consider the subsequent era of change and revision. Although functional decomposition can be useful for developing individual algorithms and close to computer architecture, it is not intuitive to model and analyze the requirement and design of real-world applications that are often organized into entities and their relationships.

The architecture of functional decomposition is main program/subroutine. In these top-down approaches, the main program is typically first identified. The subroutines are then developed. In contrast, object-oriented approaches decompose software systems around data from the bottom up. For example, the height, weight, and age of a person are identified first. They are then grouped into a class. The choice of main function is one of the very last steps to be taken in the software development process. The developer delays the description and implementation of the topmost function of the system as late as possible. Instead, the types of objects in a system are analyzed first. Software design progresses through the successive improvements of the understanding of these object types. This bottom-up process allows building robust and extensible solutions to parts of the problem and combining them into more and more powerful assemblies. These components, if assembled differently, may combine with other components and form other systems.

In the remainder of this section, we examine the basic building block of object-oriented systems: class.

Static Structure

Class is the basic notion from which everything in object technology derives. A class corresponds to an abstract concept and encapsulates a set of data together with the operations that apply to the data. The set of data is often called the attributes of the class. For example, “Person” is a concept and can be represented as a class. The class “Person” may contain the attributes like name, height, weight, and birth year. These attributes describe the information of the class, which is often hidden. Information hiding is an explicit mechanism for defining visibility/scope, which allows other classes to have limited rights to access these attributes. Information hiding is achieved via signatures, interfaces, and directives.

A class is a type that describes a set of possible data structures that may be represented in the memory of a computer and manipulated by a software system at the implementation level. A type is the static description of certain dynamic data elements that are processed during the execution of a software system. The notion of type is a semantic concept that directly influences the execution of a software system by defining the form of data structures that the system will create and manipulate at run time. The set of types often include primitive types such as integer, float, and character as well as user-defined types such as record types, pointer types, and array types. A class can be considered a user-defined type.

Intuitively, a class can be seen as a mold from which the instance of the class, called an object, can be built. Similar to a cake mold that can be used to make many cakes with the same shape, the instances of a class share common characteristics (attributes and operations). Any particular execution of a system may use the classes to create objects (data structures). Each such object is derived from a class. From another point of view, a class can be considered as a “frame” representing an abstract concept, whereas the instances, or objects, of the class are concrete individuals under the frame/concept. For example, “Person” is a frame/concept, whereas “John,” “Mary,” and “Bob” are instances/objects of “Person.” These objects share the common attributes such as name, height, weight, and birthyear.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

The software programs of a system are embedded in a set of classes in terms of operations. The program text is static and exists independently of any execution. In contrast, an object derived from a class is a dynamically created data structure, existing only in the memory of a computer during the execution of a system. Object-oriented programming languages include some module facility for organizing program text together with some type system for data structure. A module, in various forms such as operations, routines, and packages, is a unit of software decomposition. Software program texts are decomposed into module structures syntactically. The decomposition is typically around a set of data that is manipulated. All modules (operations) that control the same set of data are organized into the same class.

In non-object-oriented approaches, the type and module concepts remain distinct. In contrast, the notion of class merges the two concepts into a single construct in object-oriented approaches. A class is both a type and a module. This merger facilitates the encapsulation of the data that, for example, can be accessed only by the operation modules in the class. Information hiding is one of the important principles in object-oriented methods.

Attribute

Each class contains several attributes that characterize the properties of a set of objects. All objects of the same class have the same set of attributes. The values of these attributes distinguish different objects of the same class. For example, the class "Person" may contain the attributes such as name, height, weight, and birth year, which may be defined in object-oriented programming languages as follows:

```
class Person {
        string name;
        float height;
        float weight;
        int birthyear;
}
```

All instances of this class have these attributes but with potentially different values. One particular person may have the name of "John," height of 6 feet, weight 180 pounds, and born in 1980. Another person may have the name of "Mary," height of 5.5 feet, weight 120 pounds, and born in 1981. In addition to primitive types, class attributes may include other classes. For instance, each person has parents who are instances of the "Person" class as well. In this case, two additional attributes with user-defined type (the class "Person") can be added into the previous class definition as follows:

```
class Person {
        string name;
        float height;
        float weight;
        int birthyear;
        Person* father;
        Person* mother;
}
```

In object-oriented analysis and design, a system is decomposed around attributes. In a typical approach, classes and their attributes are identified by searching for the nouns (entities) in the requirements. These entities are the candidates of classes. Their properties are the candidates of the corresponding class attributes. In non-object-oriented approaches, conversely, system decomposition is around operations by searching for the verbs (processes) in the requirements.

Operation

The operation defines a certain computation (algorithm) applicable to all instances of a class. In general, the operations of a class manipulate the attributes in the class. In object-oriented analysis and design, the operations are identified in terms of the attributes in a class so that the computation and algorithm may apply to the attributes. For instance, the operation "age" can be defined to calculate the age of a person as follows:

```
int age() {
        return thisyear - birthyear;
}
```

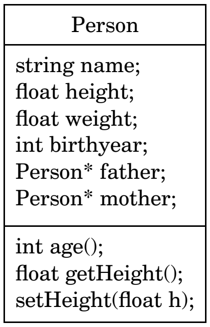
An operation consists of an interface (header) and a body. The interface of an operation may include the name, parameters, and return type. The body of an operation is a sequence of instructions describing the implementation of the computation and algorithm. The implementation of an operation is hidden from the user who only needs the interface information to invoke the operation (e.g., the user of a calculator does not need to know how the calculator performs its function). Some object-oriented programming languages separate the interface from its implementation and call the interface "operation" and the implementation "method." In this way, the same interface (operation) may have different implementations (methods) that can be dynamically selected at run time, called dynamic binding. This separation also facilitates the modification of the implementation as long as the interface stays unchanged. For the example of the Y2K problem, suppose the "birthyear" stores a two-digit format of data originally. The change from two-digit to four-digit for "birthyear" and "thisyear" only affects the implementation of the "age" operation. It does not affect the interface of the operation. Thus, the client who uses the operation need not be aware of this change.

In practice, it is common to have "getter" and "setter" operations to access the attributes of a class. For example, the operations "getHeight" and "setHeight" can be defined to access the "height" attribute as follows. These operations may be used to retrieve the height information of a person and change the height value when the person grows up:

```
float getHeight() {
        return height;
}
setHeight(float h) {
        height = h;
}
```

Class Diagram

In the previous sections, we introduced the basic concept of class and discussed attributes and operations using some examples at the implementation level. At the design level, graphical notations are often used for conveying concepts and information. For example, the Unified Modeling Language (UML) provides several diagrams, such as class, object, sequence, and collaboration diagrams, to model and represent object-oriented design. UML is a general-purpose language for specifying, constructing, visualizing, and documenting artifacts of software-intensive systems. It provides a collection of visual notations to capture different aspects of the system under development. In particular, a class diagram is one of the important diagrams provided by UML. In a class diagram, each class is represented as a rectangle typically with three compartments containing the name, attributes, and operations of the class. For example, the "Person" class can be represented as follows in the class diagram:



```
Person

string name;
float height;
float weight;
int birthyear;
Person* father;
Person* mother;

int age();
float getHeight();
setHeight(float h);
```

In class diagrams, only the important interface information of a class is represented so that the designers are able to concentrate on essential issues at the early stage of software development. They do not need to consider low-level implementation issues until necessary. Graphical notations, such as class diagrams, can also help the designers to communicate their design decisions.

Type

Similar to the record and array types, a class is a user-defined type from a programming language point of view. It combines a set of attributes as a single type. All instances of a class have the same set of attributes of the class. Unlike record type, a set of operations is also defined in the class type. Thus, all instances of a class may perform these operations on their own set of attributes. For instance, the "Person" class is a user-defined type. Its instance can be defined as follows:

```
Person p;
```

Like other variables in programming languages, the value of "p" can be assigned, changed, and accessed. Once "p" is assigned an instance/object of "Person," the operation defined in "Person" can be invoked to perform on "p," such as "p.age()" that calculates the age in terms of the attribute values of "p."

The type-checking facility is also available for the class type in object-oriented programming languages. Type mismatch can be checked by the compiler.

In general, the objective of a type system is to prevent illegal operations from being performed on inappropriate values. A strong typed language can guarantee that operations are only performed on values of the appropriate type. Strong typed languages, such as Java, can perform most their analysis and checks at the compile time.

Accessibility

Information hiding is one of the important principles in object-oriented approaches for the design of coherent and flexible architecture. The attributes of an object are encapsulated to restrict the accesses from outside the object. They usually can be manipulated only by the operations defined inside the class. That is, the class type defines the accessibility of attributes for all its objects. As well as the attributes, the operations can be hidden partially or completely. In this way, the class is more reusable and resilient to change.

Object-oriented programming languages often provide different levels of accessibility to the attributes and operations of objects. At one extreme, no attributes and operations can be accessed by the clients outside the object. At the other extreme, all attributes and operations can be accessed by the operations outside the object. Some object-oriented programming languages, such as C++ and Java, use "public" and "private" to represent unrestricted and no access, respectively. In the following declarations, for example, the "height" attribute cannot be accessed by any clients and the "getHeight" operation can be accessed by all clients:

```
private float height;
public float getHeight();
```

Most object-oriented programming languages also provide partial accessibility in between the two extremes. A certain kind of clients may be able to access the attributes and operations of an object. This set of clients can be defined in the class of the object. For example, "protected" is used, in C++ and Java, to refer to the corresponding attribute or operation that can only be accessed by the objects of the class and its subclasses. In addition, some programming languages separate different kinds of access rights, such as read, write, and execution. The clients may be further assigned by a class different access rights to the attributes of its objects.

Class Relationships

The primary tasks of object-oriented analysis and design include discovering classes and identifying relationships among classes. A class represents some useful concept and thus can be discovered by looking for nouns in the requirement specifications, whereas the behavior and operations of a class can be found by looking for verbs in the requirement specifications. Various relationships exist among different classes, which may be useful for the system designer to develop reusable and robust systems. We briefly introduce three important class relationships in this section, including inheritance, association/aggregation, and dependency.

Inheritance is an "is-a" relationship between classes. For example, a student is a person. A class "Student" can be identified as a kind of "Person" described in the previous sections. The class "Person" is called the superclass of the class "Student," whereas "Student" is a subclass of "Person." A subclass inherits the attributes and operations of its superclass. Moreover, a subclass may have its own specific attributes and operations. For example, the class "Student" may have specialized attributes such as GPA, in addition to all attributes of the class "Person." A subclass can overwrite some operations defined in the superclass, which may lead to polymorphism, where the different versions of the operation (defined in the class and subclasses) may be invoked through dynamic binding during the program execution.

Association or aggregation is a "has-a" relationship among classes. This relationship is usually implemented by using a class as the type of attributes. For example, a "Person" may have an attribute "address," and "address" can be identified as another class "Address" that has attributes "number," "street," "city," "state," "zip," etc. Generally, a class is associated with another class if its attributes refer to the other class. A class may also be associated with itself, by having itself as the type of attributes. For instance, a "Person" may have another "Person" as the type of its "father" or "mother," as described in the previous section.

Dependency is a "uses" relationship among classes. One class depends on another class if it comes into contact with the other class in some way. For example, the method age() of "Person" can be implemented by getting the system date that may be a type of the class "Date," extracting the current year from the system date, and subtracting the "birthyear" from the current year. In this way, the class "Person" depends on the class "Date" by using an object of "Date" to calculate the age. Association/aggregation is a stronger form of dependency.

OBJECT

An object is a run-time instance of some class. It is the concrete manifestation of a class. An object-oriented system creates a certain number of objects and lets these objects interact with each other at any time during its execution. The run-time structure is the organization of these objects and of their relationships. The dynamic and unpredictable nature is part of the reason for the flexibility of object-oriented approaches. Whereas class, discussed in the previous section, is concerned mostly with conceptual and structural issues, object includes behavioral aspects, in particular, the management of memory in the execution of object-oriented systems.

A simple class model may render complex instances, which reflects in part the power of object-oriented methods. Some interactions among the objects may be defined in the static structure in the classes. Many other object interactions may only be available at run time. It is often impossible to prevent the runtime object structures of systems from becoming large and complex. Such run-time complexity does not have to affect the static structure that should be kept as simple as possible. A small software text can describe a huge computation containing millions of objects from a simple class structure at execution time.

Dynamical Structure

The highly dynamic nature of the object-oriented model is described in terms of a run-time object structure. As opposed to traditional static approaches, the object-oriented environment lets systems create objects as needed at run time, based on a pattern that is often impossible to predict by a mere static examination of the static structure. Each object has to be created before it can be used in the software systems. At the initial state, only one object, the root object, is created. The systems repetitively create new objects on the object structure.

In object-oriented programming languages, the creation of a new object may involve the allocation of memory space for the attributes and the initialization of their values. Some special operation may be invoked to accomplish the initialization process when an object is created. The initial values of the attributes may be modified in the later stage at run time.

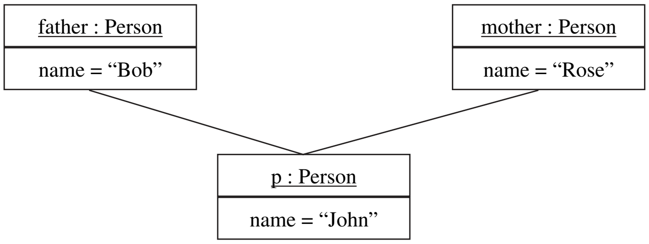
Similar to a record in traditional programming languages, an object, the instance of a class, is allocated memory space for its attributes defined in the class. The values of these attributes are stored in this memory space. For example, an object of the class "Person" may have the attribute values shown as follows:

Attributes	Values
name	"John"
height	6
weight	180
birthyear	1980

Unlike a record, an object has access not only to the attributes but also to the operations defined in its class. After an object is created, the client can use the reference to this object to refer to its attributes and invoke its operations.

Object Diagram

As well as a class diagram modeling classes and the static structure of object-oriented systems, UML provides an object diagram to model the dynamic structure of the systems. Object diagrams model the instances of things contained in class diagrams. They visually describe the objects and their interactions in the systems. An object diagram shows a set of objects and their relationships at a point in time. Graphically, it is a collection of vertices and arcs representing objects and links, respectively. For example, an object diagram of the objects of the "Person" class can be shown as follows:



In object diagrams, the “:” symbol is used to separate the object identifier from its class identifier. Both of them are underlined. An anonymous instance may be defined by omitting the object name as, for example, “: Person.” Each occurrence of an anonymous object is considered distinct from all other occurrences. It is also possible to omit the class name if it is unknown. In this case, the object name should be explicitly given. The attribute values can be shown to explicitly represent the state of the object at a given time. Objects may be connected by links to represent the relationships among them.

Object Identity

Each object created at the execution of an object-oriented system has a unique identity that is independent from the values of its attributes. Two objects of the same class have different identities. They may share the same values on their attributes, respectively. The values of the attributes may change at run time. However, object identity cannot change. Each object is assigned an identity ID (called OID) that is held during its lifetime in the system. For example, the object variables “p” and “q” of the class “Person” may be declared as follows:

Person p, q;

where “p” and “q” can refer to two distinct objects with different identities although they may share the same values of their attributes, for example, as follows:

Attributes	Values of p	Values of q
name	“John”	“John”
height	6	6
weight	180	180
birthyear	1980	1980

These two objects have exactly the same values of their attributes, but they are two distinct objects. Most object-oriented programming languages do not consider two objects with the same attributes and the attribute values as the same object. The concept of object identity compromises some operators, such as assignment and comparison, in traditional software systems. One object cannot simply assign its attribute values to the attributes of another object. Two objects cannot be compared by a comparison operator (< , >, ==) to render a definite result. For example, “ p == q” cannot be used to check whether “p” and “q” share the same values of their attributes, respectively. “ p = q” is incorrect if the user wants to assign all attribute values of “q” to the attributes of “p”, respectively. In these cases, the user has to compare or assign each field individually. At run time, the attribute values of “p” and “q” may be changed, e.g., as follows:

Attributes	Values of p	Values of q
name	“John”	“Mary”
height	6	5.5
weight	180	120
birthyear	1980	1981

In addition to attributes of basic types, objects may have attributes that represent other objects. For instance, each person having a father and a mother may have the following additional fields:

Attributes	Values
name	“John”
height	6
weight	180
birthyear	1980
father	f
mother	m

where “f” and “m” are the references to the objects of the “Person” class.

Member Access

In object-oriented programming, each attribute or operation belongs to some object. The user has to know the corresponding object before he/she can access an attribute or operation. Thus, not only the name of an attribute or operation and its parameters, but also the name of the targeting object needs to be available in order to access the member attribute or operation. The attributes and operations of an object can be accessed by other objects using the dot notation (with the “.” symbol), where the object name is before the dot and the attribute or operation name is after the dot. This kind of access is said to be qualified in that the targeting object of the call is explicitly identified. To get the height of John (object “p”), for example, we can use “p.getHeight()”.

To refer the member attributes or operations within the same object, one does not need the object name if no ambiguity. This is called the unqualified call because the targeting object is implicit. For example, the “getHeight” operation defined in the "previous Operation" section returns the value of "height" from the same object.

Current Instance

A class describes the properties and behavior of objects of a certain type. Sometimes, we may need to refer to the current instance of a class explicitly by using a reserved word ("this" in C++ and Java) in an object-oriented programming language. For instance, in the following method:

```
setHeight(float height) {
        this.height = height;
}
```

the assignment "this.height = height" avoids ambiguity because the two "height" variables refer to two different variables. The first "height" refers to an attribute defined in the current object. The second "height" refers to the argument of the operation "setHeight". Therefore, "this" is prefixed to the first "height" to explicitly identify the correct variable. Otherwise, "height = height" cannot accomplish the desired result.

Similarly, when an operation of the superclass is overwritten by a subclass, an object of the subclass has two definitions of the same operation: one defined in the superclass and the other defined in the subclass. The object may need to explicitly refer to the operation defined in the superclass by using a reserved word ("super" in C++ and Java) in object-oriented programming languages. For example, suppose a student has to be over five years old. Thus, the "age" operation in the "Person" class can be overwritten in the "Student" class:

```
int age() {
        int result = super.age();
        if (result < 6) // print out { \\ error processing }
        return result;
}
```

where "super.age()" is used to refer to the "age" operation defined in the superclass (the "Person" class in this case) to avoid confusion with the "age" operation defined in the subclass (the "Student" class in this case).

Memory Management

At run time, object-oriented systems dynamically create objects and let the objects interact with each other. Each object is allocated certain memory space at creation. The particular size of memory space for each object depends on the types of the attributes of the corresponding class. For example, the object of the "Person" class contains six attributes: one string type, two float types, one integer type, and two reference types. Thus, the size of memory space allocated to the object is the total size of all six types. It normally cannot be changed although the value of each attribute may be changed at run time.

When an object finishes its tasks and is no longer useful, it should be deleted from the system so that the memory space it occupies can be released. This task is important because these useless objects may eventually take huge memory space such that the system cannot create new objects anymore. This task is called garbage collection. Some object-oriented programming languages, such as Java, can manage automatic garbage collection. The developer does not need to worry about deleting useless objects. The run-time system of the language may search the memory and delete waste objects from time to time. Other object-oriented languages, like C++, do not provide the facility for automatic garbage collection. In this case, the developer has to explicitly delete an object when it is no longer useful.

SUMMARY

Class and object are the fundamental concepts of object-oriented technologies. They model the static and dynamic structures of object-oriented systems, respectively. A class declares several attributes and operations that can be accessed by the clients with limited right. An object is an instance of a class. Objects are created, used, and deleted at run time. Object-oriented systems manage objects and the memory space they occupy through their identities. Class and object are the basis for other object-oriented concepts, such as generalization, association, aggregation, and polymorphism.

FURTHER READING

K. Arnold and J. Gosling, *The Java Programming Language*, Reading, MA: Addison-Wesley, 1996.

G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Reading, MA: Addison-Wesley, 1999.

P. Coad, D. North, and M. Mayfield, *Object Models—Strategies, Patterns, and Applications*, Englewood Cliffs, NJ: Prentice Hall, 1995.

P. Coad and E. Nash Yourdon, *Object–Oriented Analysis*, Englewood Cliffs, NJ: Prentice Hall, 1990.

S. Cook and J. Daniels, *Designing Object Systems*, Englewood Cliffs, NJ: Prentice Hall, 1994.

M. Ellis and B. Stroustrup, *The Annotated C++ Reference Manual*, Reading, MA: Addison-Wesley, 1990.

C. Ghezzi and M. Jazayeri, *Programming Language Concepts*, 3rd Ed., New York: Wiley, 1998.

B. Meyer, *Object–Oriented Software Construction*, Englewood Cliffs, NJ: Prentice Hall, 1997.

J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, Reading, MA: Addison-Wesley, 1999.

R. Sethi, *Programming Languages: Concepts and Constructs*, 2nd Ed., Reading, MA: Addison-Wesley Longman, 1996.

L. B. Wilson and R. G. Clark, *Comparative Programming Languages*, 3rd ed., Reading, MA: Addison-Wesley, 2001.

JING DONG
University of Texas at Dallas
Dallas, Texas
JIANCHAO HAN
California State University,
Dominguez Hills
Carson, California

COMPONENT-BASED SOFTWARE ENGINEERING

INTRODUCTION

Software systems are pushed continually to address increasing demands of scalability and correctness. To meet these requirements, software development has evolved into a process of reusing existing software assets rather than constructing a new software system completely from scratch (1). By reducing time-to-market, software reuse has improved the economic and productivity factors of software production (2).

The granularity of software reuse has evolved in tandem with the capabilities of existing programming languages—from functions/procedures found in imperative programming languages, to the object/class mechanisms available in object-oriented programming languages. The current context of software reuse also scales from stand-alone software development for a single machine to capabilities supporting distributed software systems. Component-based software engineering (CBSE) (3) is becoming an accepted engineering discipline for promoting software reuse throughout the software engineering lifecycle. Beyond software reuse, CBSE also offers a promising way to manage the complexity and evolution of the development process through a unique means of encapsulation and separation of concerns at different abstraction levels (4).

This article begins with a description of the key characteristics of a software component and then provides a survey of the major component models based on those key characteristics. The latter part of this article discusses the existing engineering issues in CBSE and examines the major research projects that are addressing the challenges. Finally, this article explores recent software engineering technology toward automation in CBSE, which promises to offer yet another channel for boosting the productivity of software development.

SOFTWARE COMPONENTS UNVEILED

Software Component and Component Interface

The definition of a software component has been addressed widely in the literature (5). A software component essentially represents a precompiled independent module, possibly from a third-party vendor, and can be composed to create an even larger software component or system. A precompiled module comes in binary form, thus third-party integration requires a software component to have a mechanism to describe itself before it can be used, just like a typical user's guide for a commercial product. This special role is played by the *component interface.*

How component interfaces are defined varies, but such definitions are mainly intended to expose just enough information for third-party applications to consume without having to disclose underlying implementation details. This type of information hiding is a common practice in industry for intellectual property protection. Most common interface constituents include some form of the following characteristics:

- *public methods* of the underlying components, which represent their functional offerings to third-party applications.
- *properties,* which are generally used for specifying component configuration and deployment requirements. A component is not only a functional unit for composition, but also exercises its role in the overall software product configuration and deployment. In addition to functional properties, distributed software components sometimes specify nonfunctional properties such as throughput, availability, and end-to-end delay (for a more detailed list, see Ref 6. In a distributed environment, a software product generated by component composition is evaluated on the overall functionality and satisfaction of nonfunctional properties.
- *events,* which are used for provisioning notification among components when something of interest happens. The notified components will trigger event handlers, which are registered beforehand as event listeners.

Although the component interface specification can be Unified Modeling Language (UML; see http://www.omg.org/uml/) -based (7), or formal methods-based (8), the most common approach in application development is to use an Interface Definition Language (IDL; see http://en.wikipedia.org/wiki/Interface_Definition_Language). To compose the components, third-party software applications need to make either (1) *static invocation* via interfaces that compile the component IDL into the whole application to generate the executable code, or (2) *dynamic invocation* to discover and invoke the component interface at run time without extra compilation. Although most IDLs dictate the programming language used for third-party applications (e.g., Java IDL can only be used in Java applications), certain IDLs are language neutral for third-party software applications (i.e., a language-specific IDL compiler can be used to compile the IDL into the designated language of the third-party software applications in order to compose the corresponding component). One such example is Common Object Request Broker Architecture (CORBA; see http://www.omg.org/corba) IDL, which permits interoperability of components written in different languages.

The remaining part of this section briefly surveys the most representative commercial component techno-

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

logy models in the market based on the afore-mentioned characteristics.

Major Component Technology Models

Microsoft COM and DCOM. Component Object Model (Com; see http://www.microsoft.com/com) is the binary standard set by Microsoft for all software components on the Windows platform. Every COM component can implement any number of interfaces, each representing a different view or behavior of the object. The Distributed COM(DCOM) extends COM with distribution based on the Remote Procedure Call (RPC) mechanism. Microsoft Transaction Server (MTS) further extends DCOM with a container adding transaction and other services, which constitutes the COM+ component model.

- **Environment:** Windows platform. DCOM further needs MTS as a container.
- **IDL:** Microsoft (MS) IDL. Each interface distinguishes itself by including a universally unique identifier (UUID) in the MS IDL interface definition. As COM represents a binary standard, the IDL can be used for various MS languages such as MS C++ and Visual Basic.
- **Property:** fields defined in IDL.
- **Event:** An event in a COM component is realized by implementing a miniature COM object called the *connection* point, which implements a standard event dispatch.
- **Invocation:** supports both static invocation and dynamic invocation. To enable dynamic invocation, a COM component needs to implement a specific interface rather than relying on any infrastructural support, which is in contrast to CORBA components described later.

Sun Enterprise Javabeans (EJB). EJB (see http://java.sun.com/products/ejb) is the server-side component model for developing enterprise business applications in Java. It is tailored to Java-based applications in which a "bean" is a component.

- **Environment:** An EJB is contained in an EJB container running on a J2EE Server. The container provides added services to EJB, such as transactions and security.
- **IDL:** There is no specific IDL to expose the EJB component as the EJB component implements standard interfaces. A client invokes the stub code of those interfaces to access the EJB component at the server side.
- **Property:** Properties of EJB components are represented as deployment attributes in an extended markup language (XML; see http://www.w3.org/xml) deployment descriptor file.
- **Event:** EJB provides event services by using a message-driven bean (MDB).
- **Invocation:** supports both static invocation and dynamic invocation. The latter actually leverages Java reflection, which is more of a Java feature rather than an EJB feature.

OMG CORBA Component Model (CCM). CORBA is the initiative of the Object Management Group (OMG; see http://www.omg.org) for enabling interconnections among distributed software components across heterogeneous platforms. CCM (see http://www.omg.org/technology/documents/formal/components.htm) was introduced with CORBA 3.0. In contrast to the prior CORBA object model, CCM is designed for loose coupling between CORBA objects, facilitating component reuse, deployment, configuration, extension, and management of CORBA services.

- **Environment:** Running inside a CCM container over a CORBA ORB.
- **IDL:** CORBA Component Implementation Definition Language (CIDL), which is implementation independent and can be compiled to different languages based on the requirement of the underlying ORB.
- **Property:** attributes defined in CIDL, which are used mainly for component configuration.
- **Event:** events are defined directly in CIDL. Events are transmitted via an *event channel*.
- **Invocation:** supports both static invocation and dynamic invocation. The latter is enabled only if the CORBA ORB implements the dynamic invocation interface (DII), which is in contrast to COM components, for which dynamic invocation relies on an individual component's implementation.

Microsoft .NET Component Model. This component model is listed last because it is a fairly new model compared with the preceding three component models. In the Microsoft .NET (see http://www.microsoft.com/net) framework, an *assembly* is a component that runs on the Microsoft Common Language Runtime (CLR) (9). Each .NET language (e.g., C#, VB.NET, C++.NET) can be compiled into assembly files in the form of intermediate code, which are further just-in-time compiled into native code that can be executed in the CLR. The interoperability for an assembly is at the logical, intermediate code level rather than strictly at the physical, binary level, which makes assembly components easier to use and integrate when compared with COM components. Additionally, the .NET component model provides a significant number of benefits including a more robust, evidence-based security model, automatic memory management, and native Web Services support.

- **Environment:** Windows platform, .NET framework, CLR.
- **IDL:** MS CIL, (Microsoft Common Intermediate Language).
- **Property:** represented as *metadata* in the MS CIL, which is readable and writable within the CLR.

- **Event:** events are defined as first-class language constructs in .NET programming languages.
- **Invocation:** supports both static invocation and dynamic invocation. The latter is realized by .NET reflection.

This section described major characteristics and representative samples of software component models. The next section discusses engineering principles for CBSE.

ENGINEERING COMPONENT-BASED SOFTWARE SYSTEMS

To achieve the benefits of CBSE mentioned earlier, there are many issues that need to be addressed. We describe a list of representative issues and supporting technologies.

Componentizing Large Software Projects. Componentization for a software application needs to identify reusable assets from the initial phase of the software engineering lifecycle. Domain analysis (10) can be applied to identify the similarities among software applications, and those similarities can be used as a basis for implementing software components. The most popular domain analysis technique is Feature-Oriented Domain Analysis (FODA) (11), for which the relationship between parent feature and child feature are specified. The result of FODA is a feature set, with each feature corresponding to a reusable component. Componentization via FODA can be either manual or automatic (12). Moreover, a product line (13) can be derived during the domain analysis phase to derive a family of software components that implement common features, but satisfy different specific needs (such as with different nonfunctional properties).

In CBSE, there are concerns that crosscut the modularization boundaries of individual components (e.g., Quality of Service (QoS), distribution, and synchronization). Consequently, there is a need for componentization to capture those concerns in a modular way as well. The idea of aspect-oriented programming (AOP) (14) can be applied to CBSE. AOP essentially provides a means to capture crosscutting aspects in a modular way with new language constructs and a new type of translator called a *weaver* to compose the aspects into the base components. AOP can provide benefits to CBSE in the sense that crosscutting assets can be identified during domain analysis (15) or software modeling (16). The crosscutting assets can be used to derive aspectual components (17) that are weaved into the main components to realize component composition.

Ensuring the Reusability of Software Components. Software component reusability requires comprehensive support ranging from language design and infrastructural enabling to application architecture.

- **Language Design:** The language in which the components are written needs to be flexible and descriptive with reduced run time overhead [an example of such a language is Cecil (18,19)]. The reusability of .NET components has also been greatly enhanced with the logical interoperability of CIL. In contrast, although COM is a binary standard for components, its reusability is restricted to the physical code level, which requires much more complicated effort from the component user.
- **Infrastructural Enabling:** Infrastructure support provides separation of concerns—security, transaction, persistence, and memory management can all be managed by the underlying infrastructure with the software component implementation remaining as lean as possible to achieve a finer granularity, thus making it more reusable. Examples of such infrastructure include J2EE servers and EJB containers for EJB components, ORB and CCM containers for CORBA components, and CLR for .NET components.
- **Application Architecture:** CBSE not only motivates the reusable components in the small, but also motivates the reusable design in the large. Design patterns (20) offer a reusable solution to recurring problems in software design, which can help in developing a library of components accommodating a relatively fixed set of concepts in a specific problem domain.

Predictable Assembly for Software Components. Although component composition largely targets syntactical composability, it is also desirable to provide behavioral predictability for an assembled software system at design time before component composition. Predictability is not possible for any arbitrary design; with constraints such as memory space and battery life omnipresent, real-time embedded systems have sufficient grounds to apply predictable assembly based on computation of constraints and design space pruning (21,22).

Publishing and Discovering Software Components. Software components can only be used by registering with a central repository that provides for both static discovery and dynamic discovery, which can take two forms: proprietary or public. Proprietary publishing can only be discovered and used by third-party software with the designated (matching) platform or component technology. One example is CORBA components, which are published via an interface repository and discovered via the CORBA naming service. Recent emergence of service-oriented architecture (SOA) (23) can be seen as a component-based software system with public publishing and discovery infrastructure based on the Universal Description, Discovery, and Integration (UDDI) standard, for which Web services can be consumed across a heterogeneous distributed environment (23).

TOWARD AUTOMATION IN CBSE

CBSE boosts productivity in terms of reusability and manageability of software components (5). Recent progress in software engineering has seen yet another dimension for enhancing productivity through automation, particularly automation in both component generation and component assembly.

Automation in Component Generation

There are two directions toward automatic component generation. One is a model-driven approach (e.g., model-driven architecture; MDA; see http://www.omg.org/mda) for code generation based on high-level implementation-independent models. The other is the concept of a software factory[1](24), which leverages domain-specific best practices and schematizes those best practices for automatic component generation.

MDA. MDA is an initiative from OMG for capturing the essence of a software system in a manner that is independent of the underlying implementation platform. Model-driven software development permits a software solution to be more easily targeted to different platforms while also protecting key software assets from obsolescence. Model-driven approaches, like MDA, can assist in re-engineering legacy software systems and commercial-off-the-shelf (COTS) software into platform-independent models (PIMs). A PIM can be mapped to software components on platform-specific models (PSMs), such as CORBA, J2EE, or .NET. In this way, legacy systems and COTS components can be reintegrated into new platforms efficiently and cost-effectively (25). The vision of MDA also includes standards that enable generative construction of interoperating bridges between different technologies leveraging application and platform knowledge. One of the MDA technologies is an Interworking Architecture (see http://www.omg.org/cgi-bin/doc?formal/02-06-21), which provides a bridge that allows COM and CORBA objects to interoperate from model-driven specifications.

Software Factory. With new component models and infrastructure emerging each year, CBSE is becoming more complicated from the viewpoints of design, implementation, and deployment. Nevertheless, the goal of CBSE is not only to promote software reuse but also to boost the industrialization of software components in a manner similar to the success of hardware components. Toward that end, a software factory is defined as a "software product line that configures extensible tools, process, and content using a software factory template based on a software factory schema to automate the development and maintenance of variants of an archetypical product by adapting, assembling, and configuring framework-based components" (24). Compared with MDA, the major difference is that a software factory focuses on domain-specific modeling to present the best practice, whereas MDA is more ambitious aiming at generating full component code for any PSM.

Automation in Component Assembly

Although the preceding section on automatic component generation can be seen as an effort toward creating a component factory, the concept of automatic component assembly is focused on creating a software system factory. Among related research projects, UniFrame (6,26) is a framework for assembling heterogeneous distributed components with nonfunctional property guarantees. It uses a unified meta-component model (UMM) (27) to encode the meta-information of a component, such as functional properties, implementation technologies, and cooperative attributes. In UniFrame, a generative domain model (GDM) (10) is also used to capture the domain knowledge and to elicit assembly rules for automatic generation of glue/wrapper code to bridge the heterogeneity. Upon violation of nonfunctional property constraints for the assembled system, a discovery service is triggered to identify alternative component candidates and the automatic component assembly process is repeated until nonfunctional requirements are satisfied. Complementary to the UniFrame approach, Cao et al. (28) propose a rule-inference-based approach for choosing alternative components, which simplifies the discovery process for alternative component candidates. Also, rather than using GDM for generating glue/wrapper code generation rules, Cao et al. (28) use dynamic binary code adaptation to instrument hook code for component assembly at run time.

CONCLUSION

CBSE has been recognized as an important approach for promoting reusability and manageability of software systems. This article identified the key characteristics of CBSE and introduced several mainstream component technology models. The article also examined the primary CBSE challenges and introduced related work. With the rapid evolution of the software engineering discipline, CBSE is also evolving accordingly. As such, pilot efforts on adopting automated software engineering in component-based software development, such as those described here, have much promise. Although still in their early stages, these approaches provide new opportunities for promoting productivity of component-based software development and are becoming active research topics for CBSE communities.

BIBLIOGRAPHY

1. D. McIlroy, Mass-produced software components, *Software Engineering Concepts and Techniques, 1968 NATO Conference on Software Engineering*, 138–155, 1969.
2. P. Devanbu, S. Karstu, W. Melo, and W. Thomas, Analytical and empirical evaluation of software reuse metrics, *Proc. of 18th International Conference on Software Engineering (ICSE)*, IEEE Computer Society, 1996: 189–199.
3. G. T. Heineman and W. T. Councill, *Component Based Software Engineering: Putting the Pieces Together*, Reading, M.A.: Addison-Wesley, 2001.
4. A. W. Brown, *Large-Scale Component-Based Development*, Englewood Cliffs, N.J.: Prentice Hall, 2000.
5. C. Szyperski, *Component Software*, Reading, M.A.: Addison-Wesley, 2002.
6. R. R. Raje, M. Auguston, B. R. Bryant, A. M. Olson, and C. C. Burt, A quality of service-based framework for creating distributed heterogeneous software components, *Concurrency*

[1] The term software factory is overloaded; the same term was used by Michael Evans in his 1989 book *The software factory : a fourth generation software engineering environment*. We use the concept as defined in Ref. 24.

and Computation: Practice and Experience, **14** (12): 1009–1034, 2002.

7. J. Cheesmana and J. Daniels, *UML Components*, Reading, M.A.: Addison-Wesley, 2001.
8. G. T. Leavens and M. Sitaraman, *Foundations of Component-Based Systems*, Cambridge, 2000.
9. D. Box and C. Sells, *Essential .NET Volume 1: The Common Language Runtime*, Reading, M.A.: Addison-Wesley, 2003.
10. K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Reading, M.A.: Addison Wesley, 2000.
11. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, *Technical Report, CMU/SEI-90-TR-21*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, P.A.: 1990.
12. F. Cao, Z. Huang, B. R. Bryant, C. C. Burt, R. R. Raje, A. M. Olson, and M. Auguston , Automating feature-oriented domain analysis, *Proc. of International Conference on Software Engineering, Research and Practice (SERP)*, CSREA Press, 2003. pp. 944–949,
13. J. Whithey, Investment analysis of software assets for product lines, *Technical Report, CMU/SEI-96-TR-010,* Software Engineering Institute, Carnegie Mellon University, Pittsburgh, P.A.: 1996.
14. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, Aspect-oriented programming, *Proc. European Conference on Object-Oriented Programming* (*ECOOP*), 220–242, 1997.
15. F. Cao, B. R. Bryant, R. Raje, M. Auguston, A. Olson, and C. C. Burt, A component assembly approach based on aspect-oriented generative domain modeling, *Electronic Notes in Theoretical Computer Science*, **114**: 119–136, 2005.
16. J. Gray, T. Bapty, S. Neema, and J. Tuck, Handling crosscutting constraints in domain-specific modeling, *Commun. ACM*, **44** (10): 87–93, 2001.
17. J. C. Grundy, Multi-perspective specification, design and implementation of components using aspects, *International J. Software Eng. Knowledge Eng.,* **10** (6): 713–734, Singapore: World Scientific, 2000.
18. C. Chambers, The Cecil language: specification and rationale, *Technical Report #93-03-05,* Department of Computer Science and Engineering, University of Washington, Seattle, W.A., 1993.
19. C. Chambers, Towards reusable, extensible components, *ACM Computing Surveys*, **28**(4): 192–192, 1996.
20. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Reading, M.A.: Addison-Wesley, 1995.
21. S. A. Hissam, G. A. Moreno, J. A. Stafford, and K. C. Wallnau, Enabling predictable assembly, *J. Systems Software*, **65** (3): 185–198, 2003.
22. S.-H. Liu, F. Cao, B. R. Bryant, J. G. Gray, R. R. Raje, A. Olson, and M. Auguston, Quality of service-driven requirements analyses for component composition: a two-level grammar approach, *Proc. of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 731–734, 2005.
23. S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama , *Building Web Services with Java*, Indianapolls, IN: SAMS, 2002.
24. J. Greenfield and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, New york: Wiley, 2004.
25. D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*, New York: Wiley, 2003.
26. A. M. Olson, R. R. Raje, B. R. Bryant, C. C. Burt, and M. Auguston, UniFrame-a unified framework for developing service-oriented, component-based, distributed software systems, in Z. Stojanovic and A. Dahanayake(eds.), *Service-Oriented Software System Engineering: Challenges and Practices*, Hershey, P.A.: Idea Group, 2004.
27. R. R. Raje, UMM: Unified Meta-object Model for open distributed systems, *Proc. IEEE International Conference of Algorithms and Architecture for Parallel Processing*, IEEE Computer Society, 454–465, 2000.
28. F. Cao, B. R. Bryant, R. R. Raje, A. M. Olson, M. Auguston, W. Zhao, and C. C. Burt, A non-invasive approach to assertive and autonomous dynamic component composition in service-oriented paradigm, *J. Univer. Comp. Sci.*, **11**(10): 1645–1675, 2005.

FEI CAO[‡]
JEFF GRAY
BARRETT R. BRYANT
University of Alabama at
Birmingham
Birmingham, Alabama

[‡] Now at Microsort Corporation, Redmond, Washington.

C

COMPUTER ANIMATION

INTRODUCTION

Computer animation is widely used in games, movies, and TV programs. Topics such as human animation, fluid dynamics, rigid bodies, cloth simulation, flocking, and deformable models are covered in this field. In this article, we introduce the key techniques used for such kind of animations by observing the life of a virtual character "Joe," who is living in cyberspace.

MOTION SYNTHESIS AND ANATOMICAL MODELS

Joe is a handsome, middle-age "virtual" office worker who is hardworking, rich, and an enthusiast of unusual hobbies. He wakes up every morning at 5 o'clock and goes to jog. He has three modes of jogging: keyframe animation,[1] physically based, and data driven (Fig. 1).

Joe hates his running motion generated by keyframe animation as it looks so unnatural; the animator was very lazy and used only four keyframes for this motion. All the dynamics are ignored and his body moves like a puppet controlled by a first grader. When he uses this mode, all the other pedestrians laugh at him.

The next mode is physically based.[2] In this mode, the motion of the body is simulated by forward dynamics (1–5).[3] At every frame, the torques are applied to the joints and the movements are computed. The problem is then how to compute the torque that results in a natural running motion. Joe uses a PD controller (6,7)[4] for that purpose. He thinks this mode is a bit better as his jogging motion satisfies the laws of dynamics. For example, while his body is moving in the air, his body is pulled down to the ground by the gravity, and he can feel the impact of the ground reaction force when the foot lands onto the ground. He saw the Hodgins family (Fig. 2) (8) jogging using this mode. He said "Hi" to them, but unfortunately it looked like they just ignored him and ran away. Actually, they tried to grin back but their necks were immediately pulled back to the original posture as the feedback gain was too strong. Joe liked to use this mode a few years ago, but he realized the PD controller did not work any more after he gained too much weight. PD control is very sensitive to the change of physical parameters such as mass and moment of inertia. One day, he went out to jog and fell to the ground immediately. He had to continue the locomotion miserably like a toy until a kind pedestrian helped him to switch off the PD controller.

[1] Keyframe animation is the most well known and fundamental technique to generate 3-D character animation. It can be used to generate scenes of rigid objects like boxes or balls rolling on the floor, airplanes flying in the air, or human characters moving around in the environment. The method is very simple and intuitive; several keyframe postures must be prepared, and then the system interpolates these postures by linear interpolation or polynomial curves such as B-splines (Fig. 1). The method is intuitive and simple, but whether the motion looks natural depends on the skills of the animator.

[2] Physically based animation is a common term for all sorts of animation techniques that are based on the laws of physics. Such techniques are used in almost all sorts of animation, which those of characters, trees, cloth, hair, and liquid.

[3] Forward dynamics is a term used commonly in robotics. When simulating the movements of a multibody system such as a robot or a human, the status of the system is represented by the generalized coordinates θ, which includes parameters such as the location \ orientation of the root and the joint angles of the rotational joints. Based on Newton's laws of dynamics, the motion equation of the system can be written as

$$\tau = M\ddot{\theta} + V + G \tag{1}$$

where τ is the externally added force \ torque to the body, M is the mass matrix, V is the Coriolis force, and G is the gravity force. By specifying the torque applied to the body, the acceleration of the generalized coordinates can be computed as follows:

$$\ddot{\theta} = M^{-1}(\tau - V - G) \tag{2}$$

By using $\ddot{\theta}$, the generalized coordinates and their velocities can be computed by integration. This computation, to calculate the movements of the system by giving the torque as the input, is called forward dynamics. Forward dynamics was considered as a heavy computation process in the old days (1), Only methods of $O(N^3)$ complexity existed, where N is the degrees of freedom of the system; however, nowadays, methods of $O(N)$ are available (2). The opposite procedure is called inverse dynamics, which is to compute the torque/force generated at the joints from the motion data. Inverse dynamics analysis is used in the sports and the manufacturing area to examine how much torque is required at each joint to achieve a given movement. It is also used in spacetime constraints when the objective function includes terms of joint torques (3–5).

[4] PD control was developed originally in robotics to control robot manipulators. It has been used in computer animation to control multibody systems such as humans. The equation of PD control can be written by

$$\tau = k(\theta_d - \theta) + d(\dot{\theta}_d - \dot{\theta}) \tag{3}$$

where τ is the torques to be made at the joints, $(\theta, \dot{\theta})$ are the generalized coordinates the body and their derivatives, $(\theta_d - \dot{\theta}_d)$ are their desired values at the keyframes, and k, d are constants each called elasticity and viscosity, respectively. It is possible to say that in this mode the body is pulled toward the keyframes by the elastic and the damping forces. The larger the deviation from the desired motion is, the larger the torques exerted to the joints will be. The acceleration $\ddot{\theta}$ will be computed based on Newton's laws of dynamics, and the velocity and the generalized coordinates of the body is finally updated.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

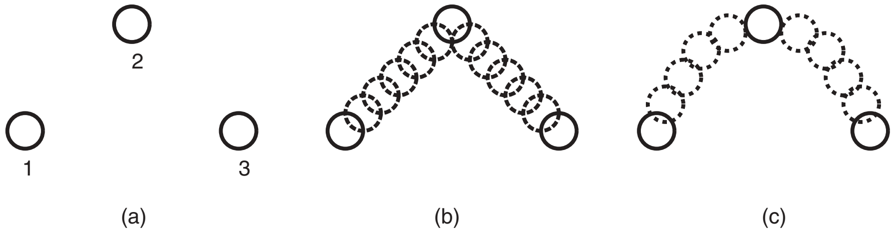


Figure 1. The trajectory of a ball flying generated from three keyframes shown in (a), by using (b) linear interpolation and (c) cubic interpolation.

Now, he simply uses the **motion capture**[5] data to control himself. The data are captured by an optical motion capture system that is mostly used today. As they are real human motion, they look realistic and natural. As several techniques to edit the motion have been developed, such as inverse kinematics[6] or spacetime constraints (9–13)[7], the same jogging motion can be used for different situations such as zigzag running or running up hills (Fig. 3).

Figure 2. The Hodgins family [Hodgins and Pollard (8)].

[5] Motion capture system is a device to digitize the 3-D trajectory of the human motion. Three different types of motion capture systems exist in the market: optical, magnetic, and mechanical. Currently, the optical system has the highest share for its high precision, robustness, and large capturing volume.

[6] Inverse Kinematics is a term originally used for a problem to compute the generalized coordinates of the system from the 3-D position of its segments/end effectors. In computer animation, it is a term for a technique to edit the posture of a virtual character by changing the location of the body segments by dragging it with a mouse. The animator can drag any position of the character in the scene, and then the generalized coordinates of the body are updated in a way that the body follows the movement of the mouse cursor naturally (Fig. 3). Inverse kinematics is a problem that makes use of the redundant degrees of freedom (DOF) of the system. The DOF of the human body is much larger than the DOF of instructions by the user, which is two/three in case the user controls the segment by a mouse. Several methods exist: cyclic coordinate descent method, pseudo inverse method, and analytical methods. The opposite word is forward kinematics, which means to calculate the 3-D position/orientation of the end effectors from the generalized coordinates.

[7] Spacetime constraints (9) is a method to calculate the trajectory of the body by optimizing an objective function while satisfying constraints specified by the user. The problem can be written in the following form:

$$\min_{\theta} J = \int f(\theta)dt - s.t. \tag{4}$$

$$C_1(\theta) = 0 \tag{5}$$

$$C_2(\theta) \leq 0 \tag{6}$$

where J is the objective function, θ are the trajectories of the generalized coordinates of the body, and C_1 and C_2 are the equality and inequality constraints, respectively. For example, it is possible to generate a natural-looking jumping motion by minimizing the following function during jumping (10,11)

$$\min_{\theta} J = \int_{t_0}^{t_f} (m(t) - M_0)^2 dt \tag{7}$$

where $m(t)$ is the linear and angular momentum of the body at the moment of jump, (t_o, t_f) are the time the jumping starts and ends, and M_o is the linear and angular momentum of the body at time t_o. Spacetime constraints have also been used for motion editing and for retargeting the captured motion to a character with different body size (12,13). This technique can be performed by solving the following optimization problem

$$\min_{\theta} J = \int_{t_0}^{t_f} (\theta(t) - \theta_0(t))^2 dt. \tag{8}$$

where $\theta_0(t)$ is the original motion data. In this case, the idea is to keep the whole motion as similar as possible to the original motion.

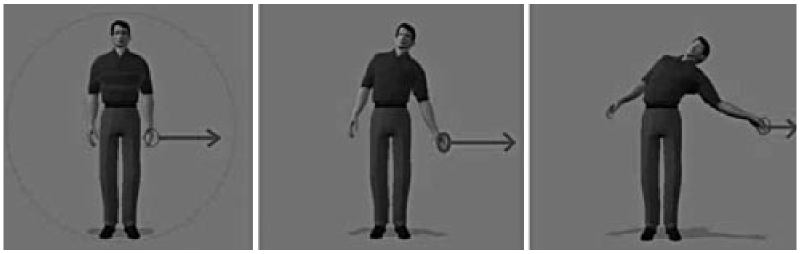


Figure 3. Changing the posture of a human figure by using inverse kinematics. The user specifies the motion of the hand, and the system calculates the joint angles of the body automatically.

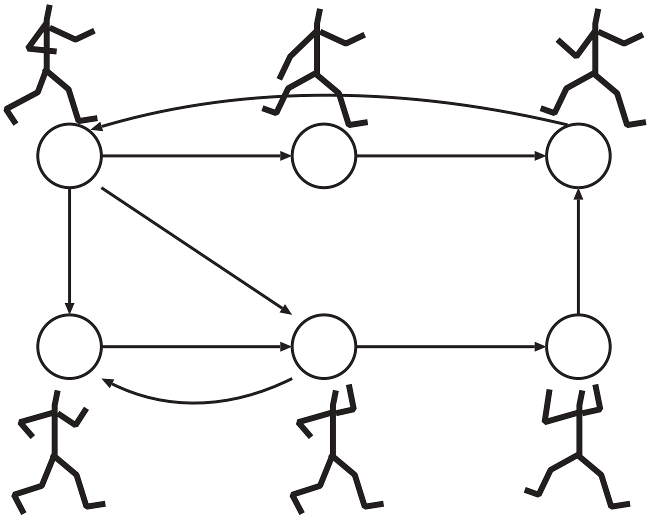


Figure 4. MotionGraph.

Recently, several real-time statistical approaches[8] have been proposed that can produce motions by interpolating various captured motions (14,15). These methods are more reliable than simple editing techniques, although they require many samples. Joe now uses such a statistical approach and feels much more comfortable because he does not face any problems as before. He can run up the stairs and change his stride smoothly to avoid stepping onto chewed gum or dog's stool. For switching from one motion to another, such as from running to walking, he uses the MotionGraph (16–18) (Fig. 4) [9]. Joe came back from jogging, took off his wet clothes, and stood in front of the mirror. He was satisfied to glare at his thick muscular body slightly sweating [Fig. 5(a)]. He bent his elbow and his bicep muscle pumped up under the skin (19) [Fig. 5(b)]. The muscle is modeled in a way that the volume is kept constant, and therefore, when it is shortened, the cross-sectional area increases (20). The body is composed of bones, muscles, the fat layer, and the skin. Collisions between the tissues are always monitored. When the muscles contract, they will deform in a way such that they do not penetrate the bones. The fat layer moves over the surface of the muscles, and finally the skin covers the whole body. His body is modeled by the musculoskeletal system generated from the Visible Human Dataset (21). Analytic models (21–28)[10] are effective

[8] Statistical approaches first search the motions in the motion database that are similar to the required motion and interpolate these motions to obtain the desired motion. For example in Refs. 14 and 15, reaching motion to arbitrary positions were generated by mixing several appropriate sample reaching motions.

[9] MotionGraph is a technique to control the avatar interactively using the captured motion data. In the MotionGraph, each posture is considered as the node of a graph, and in case the posture and the velocity of the body at the nodes are similar, they are connected by an edge (Fig. 4). As far as the avatar moves along the edges of the MotionGraph, smooth transition can be expected.

[10] Anatomic models are used to model realistic humans. Surface/volumetric data of the muscles, bones, fat, and skin from the visible human dataset (21) or from high resolution CT images are used to model the anatomic structure of the body. They have been used to model the facial expressions (19,22–24) and the surface of the body (19,20,25). The muscles deform according to the control signals. The muscles can also be used to simulate realistic kinematic movements (26–28).

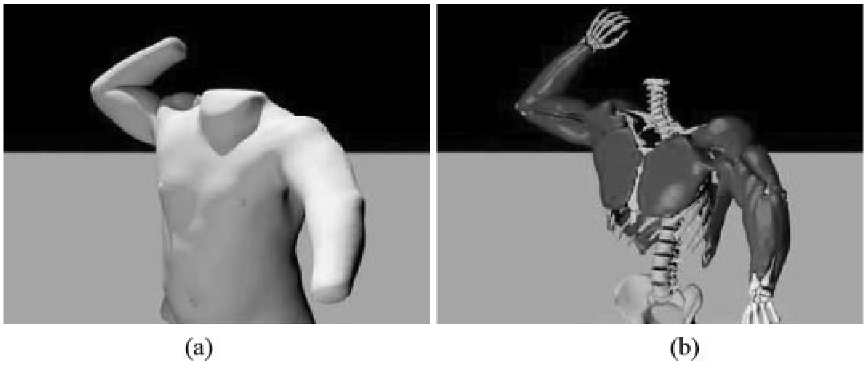


Figure 5. Joe's skin and his muscles modeled under his skin (19).

to model not only the body (19,20,25) but also the face (23,24) (Fig. 6). Using the analytic models, it is possible to generate wrinkles on the forehead and dimples at the cheeks.

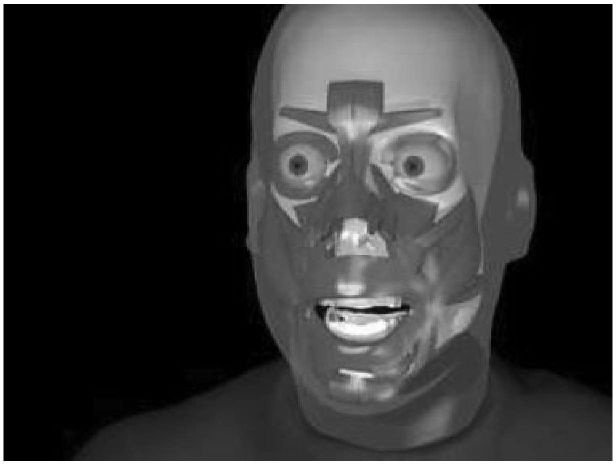


Figure 6. Joe's muscles under his face (24).

PHYSICALLY-BASED ANIMATION (RIGID BODIES, FLUIDS, HAIR, CLOTHES ETC.)

Before he took a shower, he went into the kitchen, poured a glass of water, and drank it (Fig. 7) (29). Thanks to the progress in fluid simulation(29,30),[11] he can enjoy drinking pure water instead of thick, high-viscosity drinks such as protein milk or porridge, or drinks like powder that are modeled by particles (Fig. 8).[12] He once sufferred from constipation and hemorrhoids as he was only drinking protein milk every day.

After drinking water, he went to take a shower. The splashing water coming out from the shower is modeled by the particle system. Of course, rendering a scene of taking a bath is not a difficult task anymore, but there is no point using the computer resources to calcualte the complex fluid dynamics for a bathing scene of a middle age man like Joe. After the shower, he shaved his face quickly and blowdried

[11] Fluid simulation is used to simulate flows of liquid and smoke. u, which is the velocity vector field of the flow, is updated by solving the Navier-Stokes equation:

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}p + v\nabla^2 u + F \tag{9}$$

together with the equation of mass conservation:

$$\nabla \cdot u = 0 \tag{10}$$

where ρ is the fluid density, v is the kinematic viscosity, F is the external force, and p is the pressure of the liquid. By solving Equations (9) and (10) together, the velocity field u and pressure p is updated at each time step. Usually, the space is split into Cartesian grids, and the above equations are solved for all the grids at their centers. For simulation of smoke, particles are put into the grids, and their motions are computed using the velocity vector field by Euler integration. For simulation of liquid, it is necessary to track and to visualize the boundary between the liquid and the air. The level set method (30) is used to perform liquid simulation. An implicity function ϕ that returns a negative value inside the liquid and a positive outside of it is first defined. ϕ can be updated using the vector field by solving the following equation:

$$\frac{\partial \phi}{\partial t} + u \cdot \nabla\phi = 0 \tag{11}$$

The new boundary can be calculated by tracking $\phi = 0$. Because rendering of liquid requires the visualization of the boundary and the splashes, particles can be used together with the level set method to increase the visual reality (29).

[12] The particle system is very convenient as it can model various natural phenomenon such as smoke and splashing water. The motion of every particle is modeled by simple mass point dynamics:

$$ma(t) = F \tag{12}$$

$$v(t + \Delta t) = v(t) + a(t)\Delta t \tag{13}$$

$$x(t + \Delta t) = x(t) + v(t)\Delta t \tag{14}$$

where m is the mass of the particle, x, v, a are the position, velocity, and acceleration of the particle, respectively, Δt is the sample time, and F is the external force. The problem of particle systems is that the fluid gets split easily into smaller pieces and as a result, it looks like a collection of particles (as it actually is) instead of fluids.

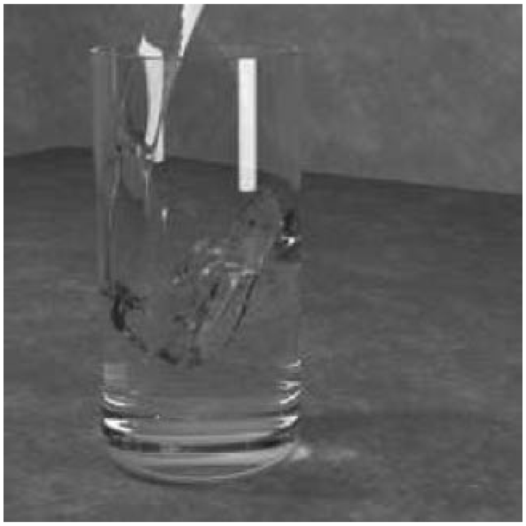


Figure 7. Pouring water in to the glass (29).

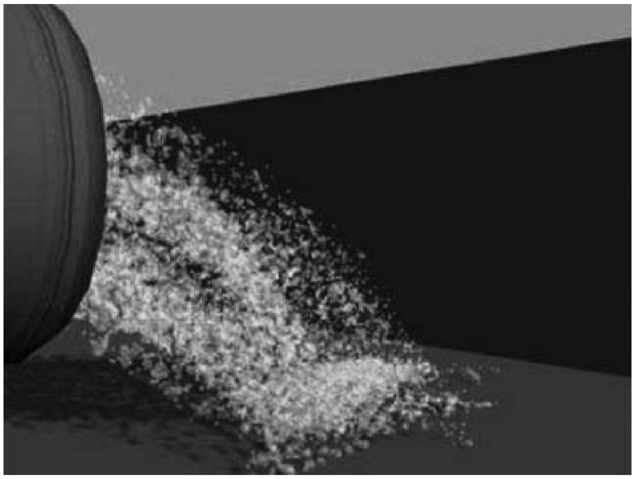


Figure 8. This is what he was drinking previously.

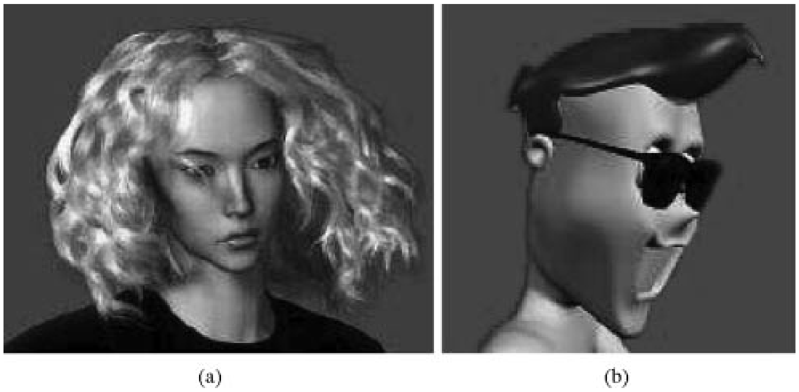


Figure 9. (a) Joe's hair blown in the wind (31). (b) His hair was like this until a few years ago.

his hair (31)[13] [Fig. 9(a)]. Watching his hair being blown dry naturally by the wind, he remembered the old days when he could only have either a skin head (so that no computation is needed for his hair) or a rigid body hair cap composed of polygons [Fig. 9(b)]. Now, he does not have to feel miserable anymore as computers are fast enough to simulate hair by physical animation.

Then, he went to the kitchen to eat breakfast. His breakfast is always cereal, and he pours the Cheelios from the box into his bowl (32) (Fig. 10). The movements of each Cheerio poured into the bowl as they collide[14] with each other was

[13] To simulate the motion of hair in a realistic manner, it is necessary to use physically based methods. The heavy computation cost is caused by the collision detection, and physical simulation of the great number of particles/rigid bodies composing the hair. Choe et al. (31) combined impulse-based simulations and the implicit integrators to compute the motion of hair efficiently.

[14] Collision detection is one of the main problems to solve when doing physical simulation. It is important not only for simulation of rigid bodies, but also for simulation of deformable objects such as cloth. It is necessary to detect when the bodies are colliding, and then add impulses/forces to the objects that are involved in the collision. Usually, rough estimation is done first by checking the collision of the bounding boxes of the objects first. The bounding box is a rectangular box that surrounds the object. Once the collisions between the bounding boxes are found, a precise collision detection based on the shapes of the objects is conducted.

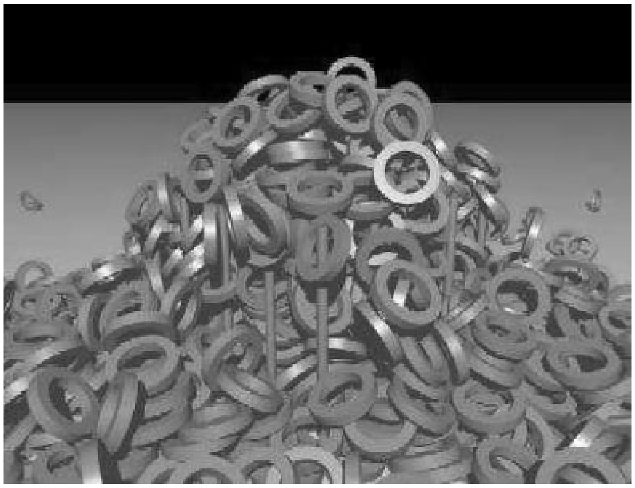


Figure 10. Joe eats Cherrios for breakfast (32).

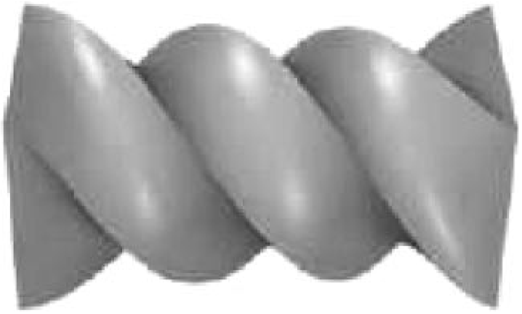


Figure 11. Using the Laplacian coordinates, a rectangular shape tube can be squeezed as shown above without any problem (35).

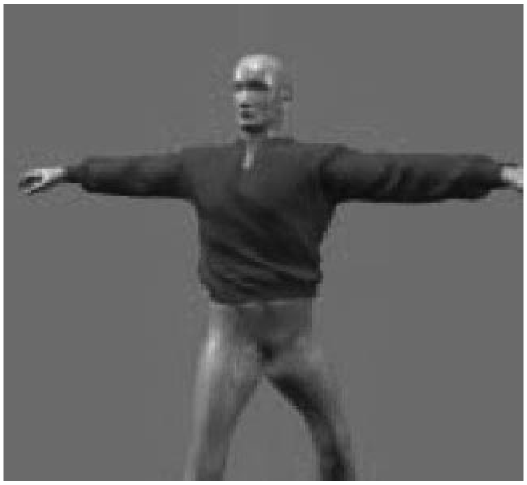


Figure 12. This is the shirt Joe loves most (38).

simulated by rigid body simulation (33,34)[15] After breakfast, Joe tried to brush his teeth. He squeezed the tube of toothpaste. But unfortunately, it was empty. Because Joe is a stingy guy, he tried to squeeze out the last bit of the toothpaste (Fig. 11) (35). The scene of the poor tube squeezed over and over (Fig. 11) was generated by using Laplacian coordinates (36,37)[16], which works robustly for drastic deformtion of 3-D three-dimensional objects.

Joe first wore his UNIQLO shirt, and spread out his hands to see whether it fit him well (38) (Fig. 12). After finding out it is OK, he wore his underwear, shirt, and suit. These clothes are all modeled by particles aligned in

[15] The main difficulty for simulating many rigid bodies in the scene is how to detect their collisions and to compute the impulses. Classic methods to simulate the collisions by Poisson's law, which is to apply elastic force proportional to the amount of penetration of objects, is unstable, and the decision of elastic parameters must be done carefully. An impulse-based simulation (33) is a popular method to simulate rigid bodies colliding. Instead of computing the forces between the objects, it computes the impulses added to the colliding bodies based on the velocity of the colliding points. Static contacts can also be simulated by a concept called micro-collisions.

Another popular method to simulate rigid bodies is developed by Jakobsen (34), which is based on particle systems. Rigid bodies are modeled by several particles that are connected by edges of fixed length. Collisions of objects are simulated simply by pushing the penetrating vertex back onto the nearest surface on the other object. In this method, the position of particles are updated by Verlet Integration, which makes the system work more stably.

[16] Laplacian coordinates enable large deformation of complex detailed meshes while keeping the shape of the details in their natural orientation. Let us define the position of the vertices in the original surface by v_i and the corresponding vertices in the deformed surface by v_i'. The Laplacian coordinate of vertex i is defined by the following equation:

$$\delta_i = v_i - \frac{1}{d_i}\sum_{j\in N_i} v_j \tag{15}$$

where N_i is the set of vertices that surrounds vertex i and d_i is the number of elements in N_i. Suppose we want to specify the location of some of the vertices in the deformed surface as

$$v_i' = u_i, i \in \{m, \ldots, n\}, m < n \tag{16}$$

and solve for the remaining vertices $\{v_i'\}, i \in \{1, \ldots, m-1\}$. In Ref. 36, this problem is solved by minimizing the difference of the Laplacian coordinates before and after the deformation:

$$\min_{\{v_i'\}, i\in\{1,\ldots,m-1\}} \sum_{i=1}^{n} \|\delta_i - \delta_i'\|^2 + \sum_{i=m}^{n} \|v_i' - u_i\|^2 \tag{17}$$

which can be solved by quadratic programming. Other additional constraints, such as keeping the volume constant (35), can be added when solving this problem.

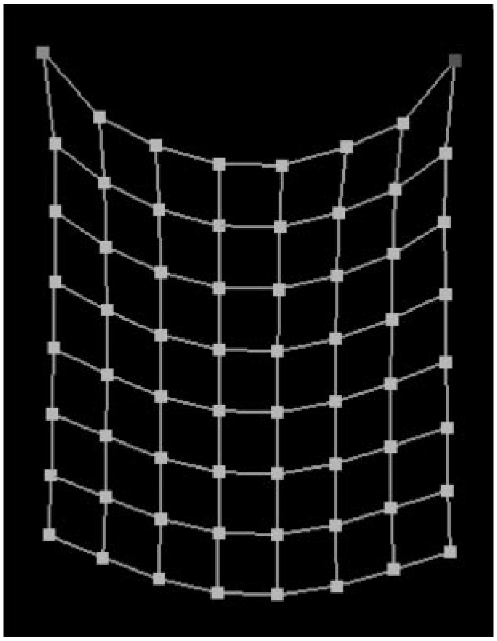


Figure 13. Cloth is modeled by particles and springs/dampers that connects them.

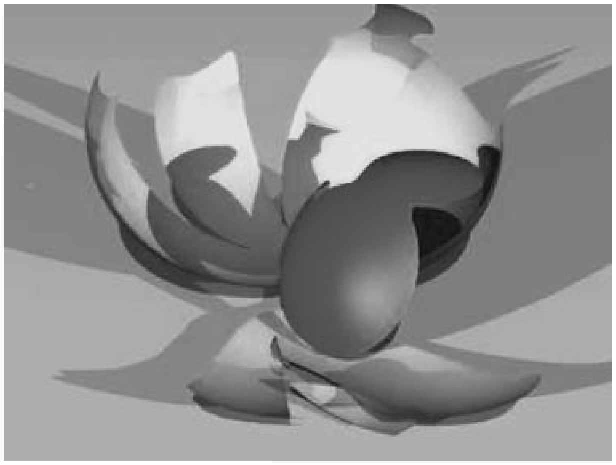


Figure 14. The ring cracked into pieces, which is definitely a bad sign (40).

grid that are connected to the adjacent particles with springs dampers (Fig. 13). The implicit integrator (38)[17] enables stable and quick simulation of clothes. Before going to work, he stood in front of the family altar, hit the ring by the stick, and prayed for his ancestors. He learned this custom when he traveled to Japan. When he hit the ring, the ring cracked (39)[18] into pieces and fell down onto the ground (Fig. 14) (40). It was a sign of bad luck.

[17] Implicit integraton enables simulation systems to take larger steps to update the position and the velocity of the particles. In the traditional explicit forward Euler method, the integration is done in the following way:

$$\begin{pmatrix}\Delta x\\ \Delta v\end{pmatrix} = h\begin{pmatrix} v_0 \\ M^{-1} f_0\end{pmatrix} \tag{18}$$

where x, v are the positions and velocities of the particles, the force f_0 is defined by $f(x_0, v_0)$, M^{-1} is the mass matrix, and h is the time step. The step size h must be selected small enough otherwise, the system could be blowed off. To avoid this error in the implicit integration method, instead of Equation (18), the following equation is solved:

$$\begin{pmatrix}\Delta x\\ \Delta v\end{pmatrix} = h\begin{pmatrix} v_0+\Delta v \\ M^{-1} f(x_0+\Delta x_0, v_0+\Delta v)\end{pmatrix} \tag{19}$$

In the forward method, we only have to calculate f_0 and integrate forward, but in the implicity backward method, we have to calculate Δx, Δv that satisfies Equation (19). This method can be performed by first expanding the Taylor series $f(x_0+\Delta x_0, v_0+\Delta v)$ at x_0,v_0 as

$$f(x_0+\Delta x_0, v_0+\Delta v) = f_0 + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial v}\Delta v \tag{20}$$

By substituting this equation into Equation (19), we get

$$\begin{pmatrix}\Delta x\\ \Delta v\end{pmatrix} = h\begin{pmatrix} v_0+\Delta v \\ M^{-1} f_0 + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial v}\Delta v\end{pmatrix} \tag{21}$$

By substituting $\Delta x = h(v_0 + \Delta v)$ into the bottom of Equation (21), we can get

$$\Delta v = hM^{-1}\left(f_0 + \frac{\partial f}{\partial x} h(v_0+\Delta v) + \frac{\partial f}{\partial v}\Delta v\right) \tag{22}$$

and solve for Δv. Then, Δx can be calculated by $h(v + v_0)$. The backward implicity method is more stable than the foward explicit method as the motion of the particles are calculated not only based on the status at $t = t_0$, but checking whether you can go back to the original position from the updated position using the derivative values at $t = t_0 + \Delta t$. As a result, the system can update the status with fixed large time steps.

[18] Simulation of objects being **cracked** and **broken** is required for many scenes in games and movies. O'Brien and Hodgins (39) simulated objects being broken by calculating where the destruction should start and how it should propagate all over the object.

Figure 15. Another bad sign in the morning.

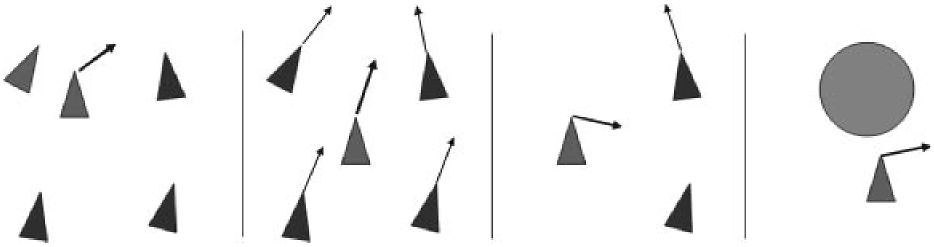


Figure 16. Forces applied to the individual flocking object: (from left to right) separation, alignment, cohesion and avoidance.

FLOCKING, CROWD, 2-D ANIMATION, REACTIVE MOTION

He went out his house and walked toward the subway station. He saw crows flocking (41)[19] in the sky while he was leaving his house (Figs. 15 and 16). At that time, the subway station was very crowded[20] and thousands of people were walking in and out (42) (Fig. 17). A long queue was at the platform waiting for the arrival of the train. Every morning, he witnesses the chaotic scene of the queue of people rushing into the train and getting pushed out by the passengers trying to get off. Joe was selecting the motion carefully to proceed most effectively without colliding with the other passengers (43) (Fig. 18). His strategy was based on reinforcement learning (43–46)[21] Joe rushed so harshly and something unexpected happened; accidentally he pushed the breast of a young lady with his arm (Fig. 19).

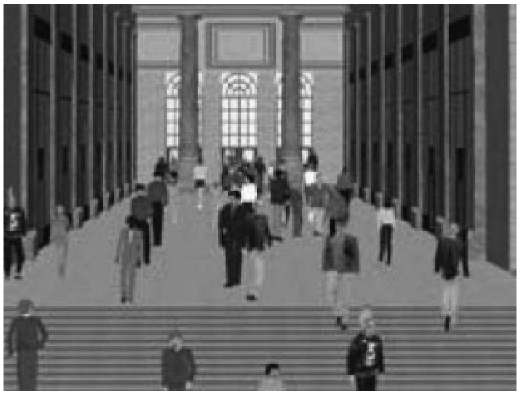


Figure 17. Here is where the battle starts in the morning .

[19] Flocking (41) is a physically based animation technique used to simulate scenes of birds flocking or fishes schooling. Each module that represents the individual is considered as a particle, and it is controlled not only by the self-driving force which is determined by the destination/direction it wants to proceed, but also by the following four external forces (Fig. 16):

1. Separation force works as potential fields that push away the individuals to avoid collision with the others.
2. Alignment force makes the individual proceed towards the same direction as the other flock-mates.
3. Cohesion force makes the individual move to the average position of local flock-mates. It spaces everyone out evenly, and applies a boundary to contain the members.
4. Avoidance keeps the flocks from running into buildings, rocks, or any other external objects in the environment.

[20] The topic to simulate thousands of characters moving around in the scene is called crowd simulation. Most crowd simulation techniques are agent based, which means that each character is self-motivated and decides his/her motion based on the objective, the destination, and the distance with the other characters. The agent-based method originates from flocking. Because humans are much more self-motivated than birds or fishes, various other techniques are combined with flocking to make the scene look realistic.

[21] Reinforcement learning is an approach to achieve real-time optimal control. It is related closely to dynamic programming in the sense that it determines the optimal motion using only the information at each state. More specifically, at each time step i, suppose the avatar selects an action and gets a reward defined by r_i. The optimal policyp π offers an action at every state that maximizes the following return value:

$$R = \sum_i \gamma^i r_i \tag{23}$$

where $0 \leq \gamma \leq 1$. The term γ^i is added because more uncertainty exists in the future. It has been used to help pedestrians avoid other obstacles/avatars walking in the streets (43,44), to control a boxer to approach and hit the target (45), and to train a computer-controlled fighter in computer games (46).

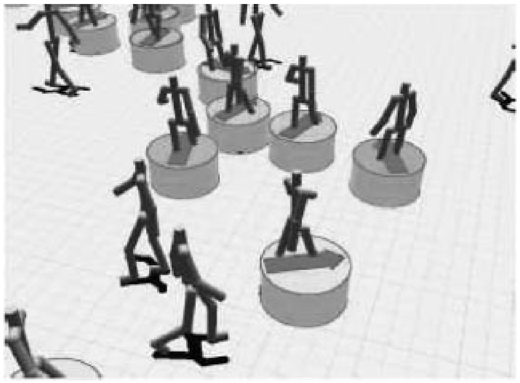


Figure 18. Joe can select the near-optimal movements that makes him proceed towards the train while avoiding the other passengers (43).

His elbow deformed[22] the soft breast of the lady. Joe was happy for awhile, but at the next moment, he was pulled out from the train by this lady, and they started to argue. Soon, the woman realized that Joe was actually a muscular man. This woman was also an enthusiast of unusual hobbies and surprisingly, started to get attracted to Joe! Joe felt so lucky that he forgot about all the bad signs in the morning. He started to talk about his hobbies, such as jogging, muscle training, and cartoon animations. Then the woman replied, “Cartoon animations!? Great, I am actually an Otaku![23] Hey, why don’t you come to my home and watch some Manga songs now?” “Absolutely!!” Joe agreed, and Joe called his boss to tell them that he was sick and he would not be coming into work that day. The woman’s name was Joie, which was the same name as the girl Joe liked in junior high. After arriving to Joie’s home, they sat on the sofa in front of the computer, and visited the YouTube website to watch the theme songs of the cartoon animations. The TV songs of the 2-D cell animations (47,48)[24] that he watched in the 1980s calms his mind, especially when he is feeling harassed like this morning (Fig. 20).

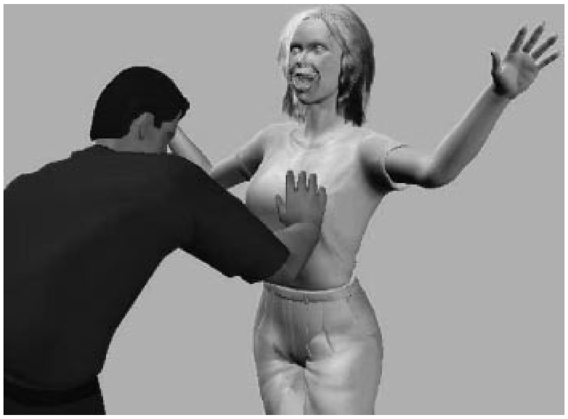


Figure 19. The breast is deformed by Laplacian coordinates with constraints of constant volume.

After watching “Maicching Machiko Sensei,” which is an animation that is not suitable for kids but acceptable for adults like Joe and Joei, they stared at each other in a good mood. Unfortunately, an even more muscular guy, who looks like the Hulk,[25] opened the door and came into the room. Joei had a lot of muscular boyfriends and unfortunately, one of her boyfriends stepped into her room at the worst time. Seeing Joe and Joei together made this boyfriend not only appear physically like the Hulk, but also act mentally like the Hulk.

The boyfriend started to destroy everything in the room and rushed to Joe. Joe was not confident to fight well, but as he was just watching some cartoon songs of the transforming robot “Daitarn 3,” he was brave enough to fight back. Joe could predict the attacks available by the boyfriend as his movements were so similar to the Hulk, and Joe was so fond of the Hulk that he had watched it many times. Joe expanded the game tree (49)[26] (Fig. 21), and he discovered that whatever motion he selects, he is going to be knocked to the ground in the end. Therefore, he selected the motion

[22] Objects such as soft tissues are called deformable objects. Techniques to handle deformable objects are required to generate animations not only for the surface of human skin but also for tissues inside the body for simulation of surgical operations and nonrigid objects like jelly or plastic objects.

[23] Otaku are people who are obsessed with cartoons, hero stories, animation figures, toys, and games. It is a Japanese term that had a negative meaning originally; it meant implicitly the people who are not good at communicating with other people, and watch cartoons and hero stories obsessively on TV. However, when the word spread internationally, thanks to the popular Japanese cartoon/hero story culture, it was taken as a positive word. It means the group of people who enjoy such hobbies. Many Otaku people are in the world now, who discuss the details and the side stories of animations and who wear the costumes of the heroes in the stories.

[24] 2-D cell animation is the traditional animation created by drawing pictures on transparent sheets called cell. Today, the costs of creating 2-D animation have been reduced significantly because of the use of computers. In the old days, all animations were drawn by the animators who required a huge amount of time and labor cost. Nowadays, computers reduce such manual labor not only through computer-based editing of 2-D cell pictures, but also by using 3-D computer animation to generate 2-D cell animation. Techniques for rendering 3-D models in 2-D cell animation style are called toon rendering (Fig. 20). In such case, nonphotorealistic rendering techniques are used to color the polygons. Techniques to render video captured movie data in 2-D cell animation style have also been developed (47). A poisson filter can be used to add cartoon taste to captured motion data (48). In TV animations such as “SD Gundam Force,” “The World of Golden Eggs,” and “Zoids” toon rendering is used for production. In games such as “Dragon Ball Z: Sagas”, “DragonQuest VIII”, and “Metal Gear AC!D2”, toon rendering is used for real-time rendering.

[25] The Hulk is a cartoon written by writer Stan Lee, penciller Jack Kirby, and inker Paul Reinman. It was first published in Marvel Comics in 1962. The hero, Dr. Robert Bruce Banner, tranforms into the Hulk when his tension reaches some threshold. The Hulk, who is enormously muscular and has green skin, is very destructive, attacks his enemies, and destroys the buildings and infrastructures of the town. It was remade recently into a Hollywood movie using new techniques of computer graphics in 2003.

[26] Game tree is used for strategy making of computer-based players of games such as tic-tac-toe, chess and go. Every node represents the ply of one of the players, and the out-going edge represents the available move at each ply. Recently, Shum et al. (49) proposed a method to apply it to simulate competitive interactions such as boxing and chasing.

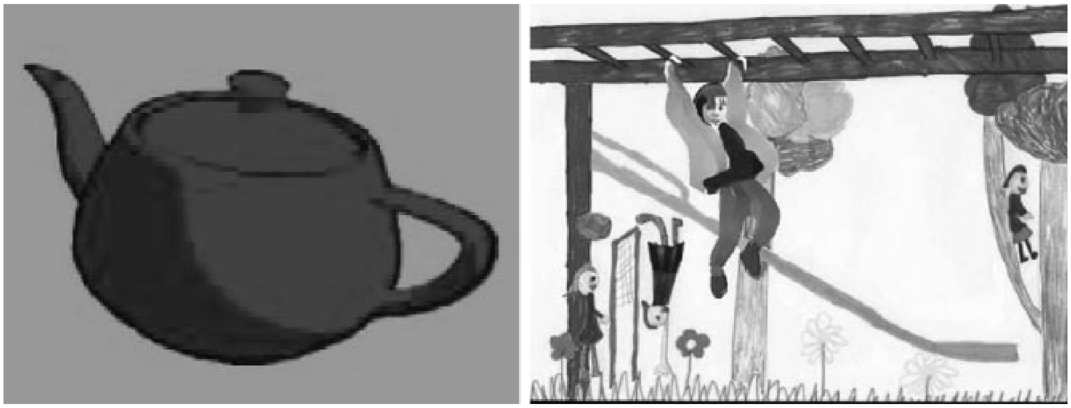


Figure 20. A toon rendered pot (left) and a frame of a tooned video (right) (40).

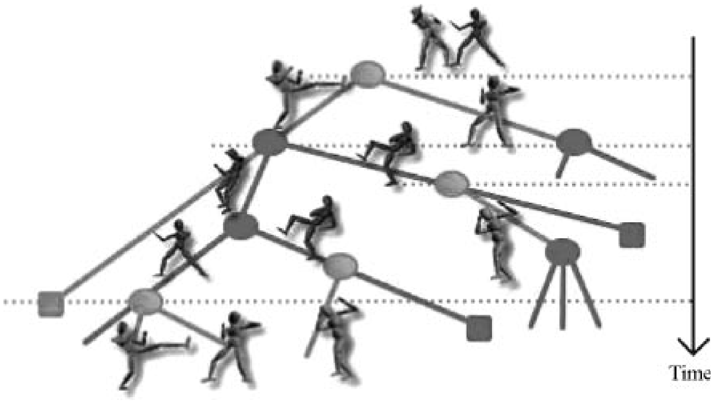


Figure 21. An expanded game tree of fighting. The distance along the vertical axis represents time. The red and blue circle nodes represent the moments each fighter launch a new action, respectively. Each edge represents the action that has been selected by the fighter.

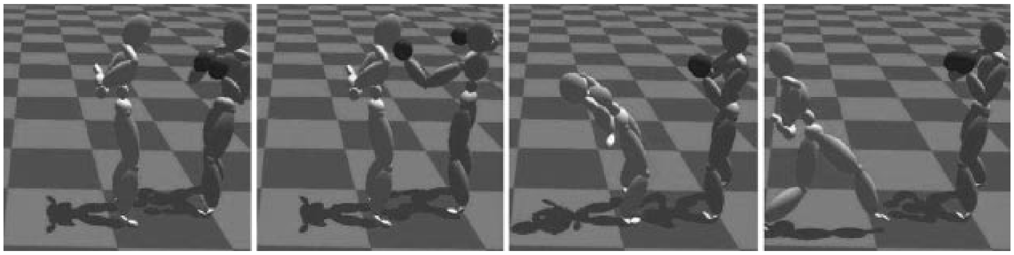


Figure 22. Actually Joe loves this (50).

that would minimize his damage, which was to receive an upper cut at the chin (50) (Fig. 22). Thanks to the momentum-based inverse kinematics, he could keep his balance by stepping backward (50,51).[27] But Joe was certainly not powerful enough to cope with the crazy boyfriend anymore. The second attack was a somersault kick; the sound of the kick passing through the air and smashing Joe's face was sound rendered (52,53)[28] by the method proposed by Dobashi et al. (53). The boyfriend's leg smashed Joe's nose and he fell to the ground like a ragdoll[29]. The motion to fall down onto the ground was simulated by Zordan's

[27] A great demand exists for scenes where people are pushed away or knocked to the ground in movies and games because recreating such dangerous scenes is difficult. Ragdoll physics are often used when the character falls to the ground. If the avatar is just supposed to take a few steps backward and start a new motion, then techniques that take advantage of motion capture data are also available (50,51).

[28] Sound rendering techinques are used to increase the sense of immersion by generating sound synchronized with the animation. In the original sound rendering paper (52), the sound was prerecorded; however, in Ref. 53, the sound was simulated fluid dynamics.

[29] Ragdoll physics is the simulation of a multibody system without actively torque or force applying at the joints. The body will just fall down like a doll in that case, but it is enough to simulate a dead body.

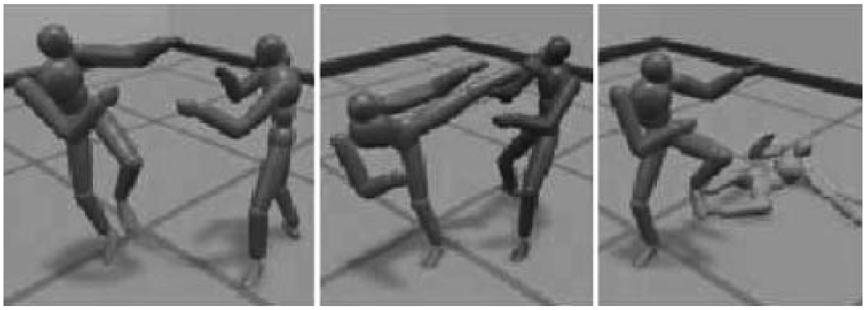


Figure 23. Joe being kicked down onto the ground (49).

technique (51) to combine PD control and motion capture data (Fig. 23) . Poor Joe, his nose was deformed drastically by Laplacian coordinates, and a huge amount of blood poured from his nose. This fight was the event the bad signs were referring to.

BIBLIOGRAPHY

1. M. W. Walker and D. E. Orin, Efficient dynamic computer simulation of robot manipulators, *ASME J. Dyn. Syst., Meas. Control* **104**: 205–211, 1982.
2. R. Featherstone, *Robot Dynamics Algorithm*, Boston, MA: Kluwer Academic Publishers, 1987.
3. P. M. Isaacs and M. F. Cohen, Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics, *Comput. Graph.* **21**(4): 215–224, 1987.
4. M. F. Cohen, Interactive spacetime control for animation, *Comput. Graph.* **26**: 293–302, 1992.
5. Z. Liu, S. J. Gortler, and M. F. Cohen, Hierarchical spacetime control, *Comput. Graph.* **28**(2): 35–42, 1994.
6. J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien, Animation of human athletics, *Comput. Graph. (SIGGRAPH)*, 1995, pp. 71–78.
7. W. L. Wooten and J. K. Hodgins, Animation of human diving, *Comput. Graph. Forum 15*, **1**: 3–13, 1994.
8. J. K. Hodgins and N. S. Pollard, *Adapting simulated behaviors for new characters Comput. Graph. (SIGGRAPH)*, 1997. 153–162.
9. A. Witkin and M. Kass, Spacetime constraints, *Proc. Comput. Graph.*, **22**: 159–168, 1988.
10. C. K. Liu and Z. Popović, Synthesis of complex dynamic character motion from simple animations, *ACM Trans. Graph.* **21**(3): 408–416, 2002.
11. Y. Abe, C. K. Liu, and Z. Popović, Momentum-based parameterization of dynamic character motion, *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004, pp. 173–182.
12. M. Gleicher, Retargetting motion to new characters, *Comput. Graph. Proc. Annual Conference Series*, 1998, pp. 33–42.
13. J. Lee and S. Y. Shin, A hierarchical approach to interactive motion editing for humanlike figures, *Proc. of SIGGRAPH'*, 199, pp. 39–48.
14. L. Kovar and M. Gleicher, Automated extraction and parameterization of motions in large data sets, *ACM Trans. Graph.***23**(3): 559–568, 2004.
15. T. Mukai and S. Kuriyama, Geostatistical motion interpolation, *ACM Trans. Graph.***24**(3): 1062–1070, 2005.
16. L. Kovar, M. Gleicher, and F. Pighin, Motion graphs, *ACM Trans. Graph.* **21**(3): 473–482, 2002.
17. O. Arikan and D. Forsyth, Motion generation from examples, *ACM Trans. Graph.* **21**(3): 483–490, 2002.
18. J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, Interactive control of avatars animated with human motion data.*ACM Trans. Graph.* **21**(3): 491–500, 2002.
19. J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw, Finite volume methods for the simulation of skeletal muscle, *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 2003, pp. 68–74.
20. F. Scheepers, R. E. Parent, W. E. Carlson, and S. F. May, Anatomy-based modeling of the human musculature, *Comput. Graph. (SIGGRAPH)*, 1997, pp. 163–172.
21. V. H. Project, Available: http://www.nlm.nih.gov/research/visible/visible_human.html.
22. S.-H. Lee and D. Terzopoulos, Heads up! Biomechanical modeling and neuromuscular control of the neck, *ACM Trans. Graph.***25**(3): 1188—1198.
23. K. Kahler, J. Haber, H. Yamauchi, and H.-P Seidel, Head shop: Generating animated head models with anatomical structure, *Proc. ACM SIGGRAPH Symposium on Computer Animation (SCA)*2002, pp. 55–64.
24. E. Sifakis, A. Selle, A. Robinson-Mosher, and R. Fedkiw, Simulating speech with a physics-based facial muscle model, *Proc. ACM SIGGRAPH Symposium on Computer Animation (SCA)*, 2006, pp. 260–270.
25. J. Wilhelms and A. V. Gelder, Anatomically based modeling, *Proc. Comput. Graph. (SIGGRAPH)*, 1997, pp. 172–180.
26. T. Komura, Y. Shinagawa, and T. L. Kunii, Creating and retargetting motion by the musculoskeletal human body model, *T. Vis. Comput.***5**: 254–270, 2000.
27. Y. Lee, D. Terzopoulos, and K. Waters, Realistic modeling for facial animation. *Proc. ACM SIGGRAPH' 95 Conference*, 1995, pp. 55–62.
28. W. Rachel, G. Eran, and F. Ronald, Impulse-based control of joints and muscles, *IEEE Trans. Visu. Comput. Graph.* In press.
29. N. Foster and R. Fedkiw, Practical animation of liquids, *Proc. SIGGRAPH*, 2001, pp. 15–22.
30. S. Osher and R. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, Berlin: Springer, 2002.
31. B. Choe, M. G. Choi, and H.-S. Ko, Simulating complex hair with robust collision handling, *ACM SIGGRAPH/*

Eurographics Symposium on Computer Animation, 2005, pp. 153–160.

32. E. Guendelman, R. Bridson, and R. Fedkiw, Nonconvex rigid bodies with stacking, *ACM Trans. Graph. (TOG)* **22**(3): 871–878, 2003.
33. B. Mirtich and J. Canny, Impulse-based simulation of rigid bodies, *Proc. Symposium on Interactive 3D Graphics*, 1995.
34. T. Jakobsen, Advanced character physics, *Proc. In Game Developers Conference*, 2001, pp. 383–401.
35. K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum, 2005. Large mesh deformation using the volumetric graph laplacian, *ACM Trans. Graph.***24**(3): 496–503, 2005.
36. O. Sorkine, Y. Lipman, D. Cohen-Or, M. Alexa, C. Rössl, and H.-P. Seidel, Laplacian surface editing, *Proc. Eurographics/ ACM SIGGRAPH Symposium on Geometry Processing*, Eurographics Association, 2004, pp. 179–188.
37. Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rossl, and H.-P. Seidel, Differential coordinates for interactive mesh editing, *smi 00*, 2004, pp. 181–190.
38. D. Baraff and A. Witkin, Large steps in cloth simulation, *Comput. Graph.*, (SIGGRAPH '98), 1998, pp. 43–54.
39. J. F. O'Brien and J. K. Hodgins, Graphical modeling and animation of brittle fracture, *Proc. SIGGRAPH*, 1999, pp. 137–146.
40. N. Molino, Z. Bao, and R. Fedkiw, A virtual node algorithm for changing mesh topology during simulation, *ACM Trans. Graph. (TOG)***23**(3): 385–392, 2004.
41. C. Reynolds, Flocks, herds, and schools: A distributed behavioral model, *Proc. SIGGRAPH 87* 21, 1987, pp. 25–34.
42. W. Shao and D. Terzopolos, Autonomous pedestrians, *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005, pp. 19–28.
43. A. Treuille, Y. Lee, and Z. Popović, Near-optimal character animation with continuous control, *ACM Trans. Graph.***26**(3): 2007.
44. L. Ikemoto, O. Arikan, and D. Forsyth, Learning to move autonomously in a hostile world, Technical Report No. UCB/ CSD-5-1395, University of California, Berkeley, 2005.
45. J. Lee and K. H. Lee, Precomputing avatar behavior from human motion data. *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004, pp. 79–87.
46. T. Graepel, R. Herbrich, and J. Gold, Learning to fight, *Proc. Computer Games: Artificial Intelligence Design and Education (CGAIDE)*, 2004, pp. 193–200.
47. J. Wang, Y. Xu, H.-Y. Shum, and M. Cohen, Video tooning, *ACM Trans. Graph. (Proc. SIGGAPH)* **23** (3): 574–583, 2004.
48. J. Wang, S. Drucker, M. Agrawala, and M. Cohen, The cartoon animation filter. *ACM Trans. Graph. (TOG)* **25**(3): 1169–1173, 2006.
49. H. P. Shum, T. Komura, and S. Yamazaki, Simulating competitive interactions using singly captured motions, *Proc. ACM Virtual Reality Software Technology*, 2007.
50. T. Komura, E. S. Ho, and R. W. Lau, Animating reactive motion using momentum-based inverse kinematics, *J. Comput. Animat. Virtual Worlds (special issue of CASA)***16**(3): 213–223, 2005.
51. V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast, Dynamic response for motion capture animation, *ACM Trans. Graph.* **24**(3): 697–701, 2005.
52. T. Takala and J. Hahn, Sound rendering, *Proc. SIGGRAPH*, 2002, pp. 211–220.
53. Y. Dobashi, T. Yamamoto, T. Nishita, Real-time rendering of aerodynamic sound using sound textures based on computational fluid dynamics. *ACM Transactions on Graphics 23*, **3**, 732–740.

Taku Komura
University of Edinburgh
Edinburgh, Scotland

CONCURRENT PROGRAMMING

INTRODUCTION

Concurrent programming refers to the development of programs that address the parallel execution of several tasks. A *process* or *task* represents the execution of a sequential program or a sequential component in a concurrent program. Each task deals with one sequential thread of execution; thus, no concurrency is allowed within a task. However, overall system concurrency is obtained by having multiple tasks executing in parallel. The tasks often execute asynchronously (i.e., at different speeds) and are relatively independent of each other for significant periods of time. From time to time, the tasks need to communicate and synchronize their operations with each other. Concurrent programming has been applied extensively in the development of operating systems, database systems, real-time systems, interactive systems, and distributed systems.

HEAVYWEIGHT AND LIGHTWEIGHT PROCESSES

The term "process" is used in operating systems as a unit of resource allocation for the processor (CPU) and memory. The traditional operating system process has a single thread of control and thus has no internal concurrency. Some modern operating systems allow a process, referred to as a *heavyweight process*, to have multiple *threads* of control, thereby allowing internal concurrency within a process. The heavyweight process has its own memory. Each thread of control, also referred to as a *lightweight process,* shares the same memory with the heavyweight process. Thus, the multiple threads of a heavyweight process can access shared data in the process's memory, although this access must be synchronized.

The terms "heavyweight" and "lightweight" refer to the context switching overhead. When the operating system switches from one heavyweight process to another, the context switching overhead is relatively high, requiring CPU and memory allocation. With the lightweight process, context switching overhead is low, involving only CPU allocation.

The terminology varies considerably in different operating systems, although the most common is to refer to the heavyweight process as a process (or task) and the lightweight process as a thread. For example, the Java virtual machine usually executes as an operating system process supporting multiple threads of control (1). However, some operating systems do not recognize that a heavyweight process actually has internal threads and only schedule the heavyweight process to the CPU. The process then has to do its own internal thread scheduling.

A *process*, which is also known as a *task*, refers to a dynamic entity that executes on a processor and has its own thread of control, whether it is a single-threaded heavyweight process or a thread within a heavyweight process (2). The task corresponds to a thread within a heavyweight process (i.e., one that executes within a process) or to a single-threaded heavyweight process. Many issues concerning task interaction apply regardless of whether the threads are in the same heavyweight process or in different heavyweight processes.

COOPERATION BETWEEN CONCURRENT TASKS

In the design of concurrent systems, several problems need to be considered that do not arise when designing sequential systems. In most concurrent applications, it is necessary for concurrent tasks to cooperate with each other to perform the services required by the application. The following three problems commonly arise when tasks cooperate with each other:

1. **The mutual exclusion problem**. This problem occurs when tasks need to have exclusive access to a resource, such as shared data or a physical device. A variation on this problem, where the mutual exclusion constraint can be relaxed in certain situations, is the multiple readers and writers problem.
2. **Task synchronization problem**. Two tasks need to synchronize their operations with each other.
3. **The producer/consumer problem**. This problem occurs when tasks need to communicate with each other to pass data from one task to another. Communication between tasks is often referred to as interprocess communication (IPC).

These problems and their solutions are described next.

MUTUAL EXCLUSION PROBLEM

Mutual exclusion arises when it is necessary for a shared resource to be accessed by only one task at a time. With concurrent systems, more than one task might simultaneously wish to access the same resource. Consider the following situations:

- If two or more tasks are allowed to write to a printer simultaneously, output from the tasks will be randomly interleaved and a garbled report will be produced.
- If two or more tasks are allowed to write to a data repository simultaneously, inconsistent and/or incorrect data will be written to the data repository.

To solve this problem, it is necessary to provide a synchronization mechanism to ensure that access to a critical resource by concurrent tasks is mutually exclusive. A task must first acquire the resource, that is, get permission to access the resource, use the resource, and then release the

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

resource. When task A releases the resource, another task B may now acquire the resource. If the resource is in use by A when task B wishes to acquire it, B must wait until A releases the resource.

The classic solution to the mutual exclusion problem was first proposed by Dijkstra (3) using binary semaphores. A binary semaphore is a Boolean variable that is accessed only by means of two atomic (i.e., indivisible) operations, *acquire* (*semaphore)* and *release* (*semaphore*). Dijkstra originally called these the P (for acquire) and V (for release) operations.

The indivisible acquire (semaphore) operation is executed by a task when it wishes to acquire a resource. The semaphore is initially set to 1, meaning that the resource is free. As a result of executing the acquire operation, the semaphore is decremented by 1 to 0 and the task is allocated the resource. If the semaphore is already set to 0 when the acquire operation is executed by task A, this means that another task, say B, already has the resource. In this case, task A is suspended until task B releases the resource by executing a release (semaphore) operation. As a result, task A is allocated the resource. It should be noted that the task executing the acquire operation is suspended only if the resource has already been acquired by another task. The code executed by a task while it has access to the mutually exclusive resource is referred to as the *critical section* or *critical region*.

EXAMPLE OF MUTUAL EXCLUSION

An example of mutual exclusion is a shared sensor data repository, which contains the current values of several sensors. Some tasks read from the data repository to process or display the sensor values, and other tasks poll the external environment and update the data repository with the latest values of the sensors. To ensure mutual exclusion in the sensor data repository example, a `sensor Data Repository Semaphore` is used. Each task must execute an *acquire* operation before it starts accessing the data repository and execute a *release* operation after it has finished accessing the data repository. The pseudocode for acquiring the sensor Data Repository Semaphore to enter the critical section and releasing the semaphore is as follows:

```
acquire (sensorDataRepositorySemaphore)
Access sensor data repository[This is the critical
section.]
release (sensorDataRepositorySemaphore)
```

The solution assumes that, during initialization, the initial values of the sensors are stored before any reading takes place.

In some concurrent applications, it might be too restrictive to only allow mutually exclusive access to a shared resource. Thus, in the sensor data repository example just described, it is essential for a writer task to have mutually exclusive access to the data repository. However, it is permissible to have more than one reader task concurrently reading from the data repository, provided there is no writer task writing to the data repository at the same time. This is referred to as the multiple readers and writers problem (2,4,5) and may be solved by using semaphores.

TASK SYNCHRONIZATION PROBLEM

Event synchronization is used when two tasks need to synchronize their operations without communicating data between the tasks. The source task executes a *signal (event)* operation, which signals that an event has taken place. Event synchronization is asynchronous.

The destination task executes a *wait (event)* operation, which suspends the task until the source task has signaled the event. If the event has already been signaled, the destination task is not suspended.

EXAMPLE OF TASK SYNCHRONIZATION

Consider an example of event synchronization from concurrent robot systems. Each robot system is designed as a concurrent task and controls a moving robot arm. A pick-and-place robot brings a part to the work location so that a drilling robot can drill four holes in the part. On completion of the drilling operation, the pick-and-place robot moves the part away.

Several synchronization problems need to be solved. First, there is a collision zone where the pick-and-place and drilling robot arms could potentially collide. Second, the pick-and-place robot must deposit the part before the drilling robot can start drilling the holes. Third, the drilling robot must finish drilling before the pick-and-place robot can remove the part. The solution is to use event synchronization, as described next.

The pick-and-place robot moves the part to the work location, moves out of the collision zone, and then signals the event `part Ready`. This awakens the drilling robot, which moves to the work location and drills the holes. After completing the drilling operation, it moves out of the collision zone and then signals a second event, `part Completed`, which the pick-and-place robot is waiting to receive. After being awakened, the pick-and-place robot removes the part. Each robot task executes a loop, because the robots repetitively perform their operations. The solution is described below and depicted using the Unified Modeling Language notation (6,7) in Fig. 1. Each robot task is depicted by a box with a thick outline. The event signals are shown as arrows.

```
pick & Place Robot:
while workAvailable do
    Pick up part
    Move part to work location
```

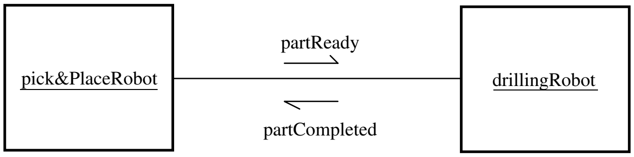


Figure 1. Example of task synchronization with two event signals.

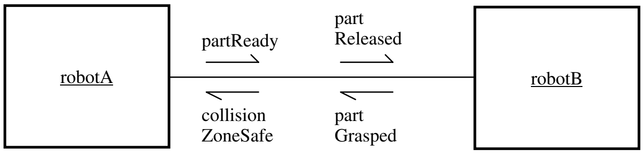


Figure 2. Example of task synchronization with four event signals.

```
    Release part
    Move to safe position
    signal (partReady)
    wait (partCompleted)
    Pick up part
    Remove from work location
    Place part
end while;
```

```
drilling Robot:
while workAvailable do
    wait (partReady)
    Move to work location
    Drill four holes
    Move to safe position
    signal (partCompleted)
end while;
```

Next, consider the case in which a giver robot hands over a part to a receiver robot. Once again, there is the potential problem of the two robot arms colliding with each other. However, this time we cannot prevent both robots from being in the collision zone at the same time because, during the hand-over, there is a time when both robots are holding the same part.

The solution we adopt is to allow only one robot to move within the collision zone at any given time. First, one robot moves into the collision zone. It then signals to the other robot that it has reached the exchange position. The second robot now moves into the collision zone. An event signal `collision Zone Safe` is used for this purpose. The giver robot signals a second event, `part Ready`, to notify the receiver robot that it is ready for the hand-over. Two more event signals are used during the hand-over, `part Grasped` and `part Released`. The part hand-over has to be as precise as a baton hand-over in a relay race. The solution is illustrated in Fig. 2 and described as follows:

```
Giver robot (robot A):
while workAvailable do
    Pick up Part
    Move to edge of collision zone
    wait (collisionZoneSafe)
    Move to exchange position
    signal (partReady)
    wait (partGrasped)
    Open Gripper to release part
    signal (partReleased)
    wait (collisionZoneSafe)
    Leave collision zone
end while;
```

```
Receiver robot (robot B):
while workAvailable do
    Move to exchange position
    signal (collisionZoneSafe)
    wait (partReady)
    Close Gripper to grasp part
    signal (partGrasped)
    wait (partReleased)
    Leave collision zone
    signal (collisionZoneSafe)
    Place part
end while;
```

Task synchronization may also be achieved by means of message communication as described next.

PRODUCER/CONSUMER PROBLEM

A common problem in concurrent systems is that of producer and consumer tasks. The producer task produces information, which is then consumed by the consumer task. For this to happen, data need to be passed from the producer to the consumer. In a sequential program, a calling operation (procedure) also passes data to a called operation. However, control passes from the calling operation to the called operation at the same time as the data.

In a concurrent system, each task has its own thread of control and the tasks execute asynchronously. It is therefore necessary for the tasks to synchronize their operations when they wish to exchange data. Thus, the producer must produce the data before the consumer can consume it. If the consumer is ready to receive the data but the producer has not yet produced it, then the consumer must wait for the producer. If the producer has produced the data before the consumer is ready to receive it, then either the producer has to be held up or the data need to be buffered for the consumer, thereby allowing the producer to continue.

A common solution to this problem is to use message communication between the producer and consumer tasks. Message communication between tasks serves two purposes:

1. Transfer of data from a producer (source) task to a consumer task (destination).
2. Synchronization between producer and consumer. If no message is available, the consumer has to wait for the message to arrive from the producer. In some cases, the producer waits for a reply from the consumer.

Message communication between tasks may be loosely coupled or tightly coupled. The tasks may reside on the same node or be distributed over several nodes in a distributed application.

With loosely coupled message communication, the producer sends a message to the consumer and continues without waiting for a response. Loosely coupled message

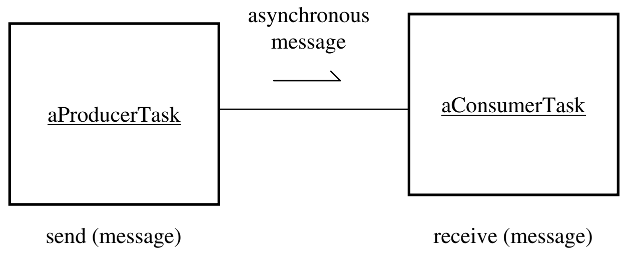


Figure 3. Loosely coupled (asynchronous) message communication.

communication is also referred to as asynchronous message communication.

With tightly coupled message communication, the producer sends a message to the consumer and then immediately waits for a response. Tightly coupled message communication is also referred to as synchronous message communication and in Ada as a rendezvous.

LOOSELY COUPLED MESSAGE COMMUNICATION

With *loosely coupled message communication*, also referred to as *asynchronous message communication*, the producer sends a message to the consumer and either does not need a response or has other functions to perform before receiving a response. Thus, the producer sends a message and continues without waiting for a response. The consumer receives the message. As the producer and consumer tasks proceed at different speeds, a first-in-first-out (FIFO) message queue can build up between producer and consumer. If there is no message available when the consumer requests one, the consumer is suspended.

An example of *loosely coupled message communication* is given in Fig. 3 using the UML notation. The producer task sends messages to the consumer task. A FIFO message queue can exist between the producer and the consumer. The message is labeled `asynchronous message`. Parameters of the message are depicted in parentheses, that is, `asynchronous message (parameter1, parameter2)`.

TIGHTLY COUPLED MESSAGE COMMUNICATION WITH REPLY

In the case of *tightly coupled message communication with reply*, also referred to as *synchronous message communication with reply,* the producer sends a message to the consumer and then waits for a reply. When the message arrives, the consumer accepts the message, processes it, generates a reply, and then sends the reply. The producer and consumer then both continue. The consumer is suspended if no message is available. For a given producer/consumer pair, no message queue develops between the producer and the consumer. In some situations, it is also possible to have *tightly coupled message communication without reply* (7).

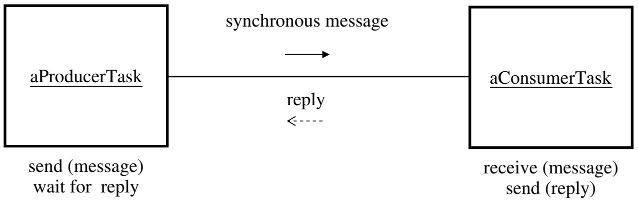


Figure 4. Tightly coupled (synchronous) message communication with reply.

An example of *tightly coupled message communication with reply* is given in Fig. 4 using the UML notation. The producer sends a message to the consumer. After receiving the message, the consumer sends a reply to the producer. The message is labeled `synchronous message`. Parameters of the message are depicted in parentheses, that is, `synchronous message (parameter1, parameter2)`. The reply is depicted in UML by a separate dashed message with the arrowhead pointing in the reverse direction of the original message.

EXAMPLE OF PRODUCER/CONSUMER MESSAGE COMMUNICATION

As an example of *tightly coupled message communication with reply*, consider the case where a vision system has to inform a robot system of the type of part coming down a conveyor, for example, whether the car body frame is a sedan or station wagon. The robot has a different welding program for each car body type. In addition, the vision system has to send the robot information about the location and orientation of a part on a conveyor. Usually this information is sent as an offset (i.e., relative position) from a point known to both systems. The vision system sends the robot a tightly coupled message, the `car ID Message`, which contains the `car Model ID` and `car Body Offset`, and then waits for a reply from the robot. The robot indicates that it has completed the welding operation by sending the `done Reply`.

In addition, the following event synchronization is needed. Initially, a sensor signals the external event `car Arrived` to notify the vision system. Finally, the vision system signals the actuator `move Car`, which results in the taking away of the car by the conveyor. The solution is illustrated in Fig. 5 and described next.

```
Vision System:
while workAvailable do
   wait (carArrived)
   Take image of car body
   Identify the model of car
   Determine location and orientation of car body
```

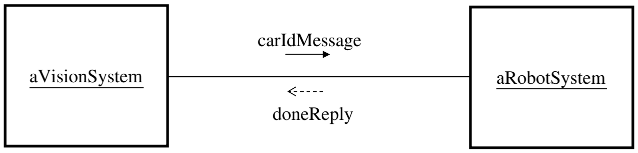


Figure 5. Example of message communication.

```
    send carIdMessage (carModelId, carBodyOffset) to Robot System
    wait for reply
    signal (moveCar)
end while;
Robot System:
while workAvailable do
    wait for message from Vision System
    receive carIdMessage (carModelId, carBodyOffset)
    Select welding program for carModelId
    Execute welding program using carBodyOffset for car position
    send (doneReply) to Vision System
end while;
```

INFORMATION HIDING APPLIED TO ACCESS SYNCHRONIZATION

The solution to the mutual exclusion described is error prone. It is possible for a coding error to be made in one of the tasks accessing the shared data, which would then lead to serious synchronization errors at execution time. Consider, for example, the mutual exclusion problem described. If the acquire and release operations were reversed by mistake, the pseudocode would be

```
release (sensorDataRepositorySemaphore)
Access sensor data repository[should be critical section]
acquire (sensorDataRepositorySemaphore)
```

As a result of this error, the task enters the critical section without first acquiring the semaphore. Hence, it is possible to have two tasks executing in the critical section, thereby violating the mutual exclusion principle. Instead, the following coding error might be made:

```
acquire (sensorDataRepositorySemaphore)
Access sensor data repository[should be critical section]
acquire (sensorDataRepositorySemaphore)
```

In this case, a task enters its critical section for the first time but then cannot leave because it is trying to acquire a semaphore it already possesses. Furthermore, it prevents any other task from entering its critical section, thus provoking a *deadlock*, where no task can proceed.

In these examples, synchronization is a global problem that every task has to be concerned about, which makes these solutions error prone. By using information hiding (8,9), the global synchronization problem can be reduced to a local synchronization problem, making the solution less error prone. With this approach, only the information hiding object need be concerned about synchronization. An information hiding object that hides details of synchronizing concurrent access to data is also referred to as a *monitor* (10), as described next.

MONITORS

A *monitor* combines the concepts of information hiding and synchronization. A monitor is a data object that encapsulates data and has operations that are executed mutually exclusively. The critical section of each task is replaced by a call to a monitor operation. An implicit semaphore is associated with each monitor, referred to as the *monitor lock*. Thus, only one task is active in a monitor at any one time. A call to a monitor operation results in the calling task acquiring the associated semaphore. However, if the lock is already taken, the task blocks until the monitor lock is acquired. An exit from the monitor operation results in a release of the semaphore; i.e., the monitor lock is released so that it can be acquired by a different task. The mutually exclusive operations of a monitor are also referred to as *guarded* operations or *synchronized* methods in Java.

Example of Monitor

An example of a monitor is given next. Consider the sensor data repository described above. The monitor solution is to encapsulate the data repository in an `Analog Sensor Repository` data abstraction object, which supports read and update operations. These operations are called by any task wishing to access the data repository. The details of how to synchronize access to the data repository are hidden from the calling tasks.

The monitor provides for mutually exclusive access to an analog sensor repository. There are two mutually exclusive operations to read from and to update the contents of the analog repository, as shown in Fig. 6. The two operations are as follows:

```
readAnalogSensor (in sensorID, out sensorValue, out upperLimit, out lowerLimit,
    out alarmCondition)
```

This operation is called by reader tasks that wish to read from the sensor data repository. Given the sensor ID, this operation returns the current sensor value, upper limit, lower limit, and alarm condition to users who might wish to manipulate or display the data. The range between the lower limit and the upper limit is the normal range within which the sensor value can vary without causing an alarm. If the value of the sensor is

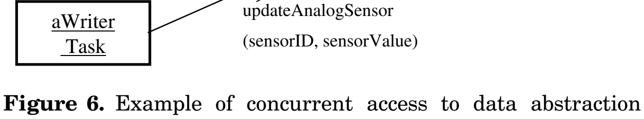


Figure 6. Example of concurrent access to data abstraction object.

below the lower limit or above the upper limit, the alarm condition is equal to low or high, respectively.

`updateAnalogSensor (in sensorID, in sensorValue)`

This operation is called by writer tasks that wish to write to the sensor data repository. It is used to update the value of the sensor in the data repository with the latest reading obtained by monitoring the external environment. It checks whether the value of the sensor is below the lower limit or above the upper limit and, if so, sets the value of the alarm to low or high, respectively. If the sensor value is within the normal range, the alarm is set to normal.

The pseudocode for the mutually exclusive operations is as follows:

```
monitor AnalogSensorRepository

readAnalogSensor (in sensorID, out sensorValue,
out upperLimit, out lowerLimit, out alarmCondition)
   sensorValue := sensorDataRepository (sensorID, value);
   upperLimit := sensorDataRepository (sensorID, upLim);
   lowerLimit := sensorDataRepository (sensorID, loLim);
   alarmCondition := sensorDataRepository
     (sensorID, alarm);
end readAnalogSensor;

updateAnalogSensor (in sensorID, in sensorValue)
sensorDataRepository (sensorID, value) := sensorValue;
if sensorValue >= sensorDataRepository (sensorID, upLim)
      then sensorDataRepository (sensorID,
        alarm) := high;
else if sensorValue <= sensorDataRepository
  (sensorID, loLim)
   then sensorDataRepository (sensorID, alarm)
     := low;
   else sensorDataRepository (sensorID, alarm)
     := normal;
end if;
end updateAnalogSensor;
end AnalogSensorRepository;
```

CONDITION SYNCHRONIZATION

In addition to providing synchronized operations, monitors support *condition synchronization*. This allows a task executing the monitor's mutually exclusive operation to block, by executing a *wait* operation until a particular condition is true, for example, waiting for a buffer to become full or empty. When a task in a monitor blocks, it releases the monitor lock, allowing a different task to acquire the monitor lock. A task that blocks in a monitor is awakened by some other task executing a *signal* operation (referred to as *notify* in Java). For example, if a reader task needs to read an item from a buffer and the buffer is empty, it executes a wait operation. The reader remains blocked until a writer task places an item in the buffer and executes a signal operation.

If semaphore support is unavailable, mutually exclusive access to a resource may be provided by means of a monitor with condition synchronization, as described next. The Boolean variable busy is encapsulated by the monitor to represent the state of the resource. A task that wishes to acquire the resource calls the `acquire` operation. The task is suspended on the `wait` operation if the resource is busy. On exiting from the wait, the task will set `busy` equal to true, thereby taking possession of the resource. When the task finishes with the resource, it calls the `release` operation, which sets `busy` to false and calls the `signal` operation to awaken a waiting task.

Below is the monitor design for mutually exclusive access to a resource. Additional examples of monitors and condition synchronization are given in Ref. (7).

```
monitor Semaphore
  -- Declare Boolean variable called busy,
     initialized to false.
private busy : Boolean = false;
  -- acquire is called to take possession of the
     resource
  -- the calling task is suspended if the resource
     is busy
public acquire ()
    while busy = true do wait;
    busy := true;
    end acquire;

-- release is called to relinquish possession of
   the resource
-- if a task is waiting for the resource, it will be
   awakened
public release ()
    busy := false;
    signal;
end release;
end Semaphore;
```

RUN-TIME SUPPORT FOR CONCURRENT PROGRAMMING

Run-time support for concurrent programming may be provided by

1. A kernel of an operating system. This has the functionality to provide services for concurrent programming. In some modern operating systems, a microkernel provides minimal functionality to support concurrent processing with most services provided by system-level tasks.
2. The run-time support system for a concurrent language.
3. A threads package, which provides services for managing threads (lightweight processes) within heavyweight processes.

With sequential programming languages, such as C, C++, Pascal, and Fortran, there is no support for concurrent tasks. To develop a concurrent multitasked application using a sequential programming language, it is therefore necessary to use a kernel or threads package.

With concurrent programming languages, such as Ada and Java, the programming language provides constructs for concurrent tasks, including task creation and deletion, as well as task communication and synchronization. In this case, the language's run-time system handles task scheduling and provides the services and underlying mechanisms to support inter-task communication and synchronization.

FURTHER READING

The body of knowledge on concurrent programming has grown substantially since Dijkstra's seminal work (3). Among the significant early contributions were Brinch Hansen (11), who developed an operating system based on concurrent tasks that incorporated semaphores and message communication, and Hoare (10), who developed the monitor concept that applies information hiding to task synchronization. Several concurrent programming algorithms were developed, such as the multiple readers and writers algorithm (12), the sleeping barber algorithm (3), the dining philosophers algorithm (13), and the banker's algorithm for deadlock prevention (3). Many of the original papers on concurrent programming are out of print. Because concurrent processing is such a fundamental concept, it has been described in textbooks for over three decades. The best modern sources of information on concurrent programming are books on operating systems, such as Silberschatz et. al. (5) and Tanenbaum (4), or books on concurrent programming languages, such as Java (14) or Ada (15). Two recommended references for further reading on concurrent programming are Bacon (2), which describes concurrent systems, both centralized and distributed, and Magee and Kramer (1), which describes concurrent programming with Java. The application of concurrent programming to the design of concurrent, distributed, and real-time applications is described in Gomaa (8).

ACKNOWLEDGMENT

Part of the material in this article is extracted from: H. Gomaa, *Designing Concurrent Distributed & Real-Time Applications with UML*, © 2000 Hassan Gomaa. Reprinted by permission of Pearson Education, Inc. Publishing as Pearson Addison Wesley.

BIBLIOGRAPHY

1. J. Magee and J. Kramer, *Concurrency: State Models & Java Programs*, New York: J. Wiley, 1999.
2. J. Bacon, *Concurrent Systems*, 2nd ed., Reading, MA: Addison Wesley, 1998.
3. E. W. Dijkstra, Cooperating sequential processes, in F. Genuys (ed.), *Programming Languages*, New York: Academic Press, 1968, pp. 43–112.
4. A. S. Tanenbaum, *Modern Operating Systems*, 2nd ed., Engle-Wood Cliggs, NJ: Prentice Hall, 2001.
5. A. Silberschatz, P. Galvin, and G. Gagne, *Operating System Concepts*, 7th ed., Reading, MA: Addison Wesley, 2004.
6. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Reading, MA: Addison Wesley, 1999.
7. H. Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Reading, MA: Addison Wesley, 2000.
8. D. Parnas, On the criteria for decomposing a system into modules, *Comm. ACM*, 1972.
9. D. Parnas, Designing software for ease of extension and contraction, *IEEE Trans. on Softw. Eng.*, 1979.
10. C. A. R. Hoare, Monitors: An operating system structuring concept, *Comm. ACM*, **17**, (10), 549–557, 1974. C. A. R. Hoare, *Communicating Sequential Processes*, Englewood Gliffs, NJ: Prentice Hall, 1985. D. Hoffman and D. Weiss, (eds), *Software Fundamentals, Collected Papers by David L. Parnas*, Reading, NA: Addison Wesley, 2001.
11. P. Brinch Hansen, *Operating System Principles*, Ebglewood Gliffs, NJ, Prentice Hall, 1973.
12. P. J. Courtois, F. Heymans, and D. L. Parnas, Concurrent control with readers and writers, *Acta Informatica*, **1**, 375–375, 1972.
13. E. W. Dijkstra, *Hierarchical ordering of sequential processes*, in C. A. R Hoare and R. H. Perrot (eds.), *Operating Systems Techniques*, New York: Academic Press, 1972.
14. D. Lea, *Concurrent Programming in Java: Design Principles and Patterns*, 2nd ed., Reading, MA: Addison Wesley, 1999.
15. J. Barnes, *Programming in Ada 95*, Reading MA: Addison Wesley, 1995.

Hassan Gomaa
George Mason University
Fairfax, Virginia

DISTRIBUTED AND COLLABORATIVE DEVELOPMENT

Rapid advancements in computer, communications, and network technologies over the years have revolutionized how computers are used. One of the emerging application areas that has gained increasing usage and visibility is computer-supported cooperative work (CSCW). The main goal of CSCW systems is to facilitate effective collaboration among users who may be distributed across geographical distance and time, using computer-based means, which may range from e-mail to instant messaging to video conferencing to chance meetings in online virtual environments.

Meanwhile, due in large part to the widespread emergence of virtual organizations and the growing trend of outsourcing, software development is fast becoming a group activity that is performed by geographically and temporally distributed team members. It is no longer unusual to have a large-scale software development project that has members located in different time zones around the world; in many cases, the members have not even-met each other. Thus, it is no coincidence that supporting distributed software development teams is area of increasing focus in CSCW system design (1–4).

The key to successful software development in distributed environments is awareness of task status and activities of team members to enable coordination and conflict detection/avoidance, to whose end online file sharing and version control systems and general-purpose communications systems (e.g., e-mail and instant messaging applications) have often been used to provide awareness information in distributed software development. However, in general, such general-purpose systems are not considered ideal as source of awareness information, largely because these systems require collaborators to perform a significant amount of extra work (e.g., diligently documenting one's actions and activities in e-mail messages or online forums and carefully following e-mail discussion threads and file check-in and check-out history) to keep track of project progress and activities. In addition, unmediated generation and unfiltered reception of awareness-related messages can lead to information overload. Therefore, in CSCW, a desired goal is to automatically provide selective awareness information to distributed team members.

Nonetheless, general-purpose file sharing and communications systems have successfully been used to support many large-scale, open-source development efforts (5–7). Success of these tools in open-source development efforts forms a sharp contrast with ongoing efforts in the CSCW community to build specialized tools and technologies for providing awareness in distributed environments, which is especially notable considering that specialized awareness tools and technologies, with the possible exception of "buddy lists" in instant messaging applications, have not yet been widely adopted by software developers in their everyday work activities.

OVERVIEW OF COMPUTER-SUPPORTED COLLABORATIVE WORK SYSTEMS

The main goal of CSCW systems is to facilitate effective collaboration among users who may be distributed across geographical distance or time using computer-based means. With factors of geographical distance and time, collaborative work can be distinguished into four types of interactions, as shown in Fig. 1 (adopted from Ref. 8). For an in-depth overview of CSCW and issues, see Ref. 8. Face-to-face interaction occurs when all the collaborators are available in the same place at the same time. Common examples of this type of interaction include face-to-face group meetings and presentations. In synchronous distributed interaction, collaborators work together at the same time but are not located in the same place. Video conferencing and distance learning are good examples of such an interaction.

Asynchronous interaction occurs when collaborators are collocated but do not work with each other directly at the same time. This kind of interaction can often be found in workplaces with different work hours or shifts (e.g., hospitals) where communication is still required to pass along information and knowledge to coordinate tasks and activities. Asynchronous and distributed interaction occurs when collaborators are not collocated and unavailable for working together at the same time. Open-source development, in which collaborating software developers are often distributed around the world and have never met each other, is an example of this type of interaction.

In practice, software development typically involves multiple types of interactions. The main mode of interaction may also change as team requirements and needs change over time. For example, in the beginning of a given project, project members may frequently hold face-to-face meetings to raise and discuss issues in depth, to generate project plans and requirements, and to get familiar with each other. Once the project is well under way, the frequency of face-to-face interaction often decreases whereas that of distributed interaction increases, with project members communicating with each other via e-mail, instant messaging, and other electronic means. Whether software development occurs in a collocated or distributed environment, online file repository and version control system [e.g., CVS (9)] is almost always employed to store, control access to, and detect conflicts in source code files, project documents, and bug reports.

CONCEPTS, TERMINOLOGY, AND ISSUES

Fundamental to enabling collaborative work over geographical and temporal distance are the concepts of awareness and shared artifacts. *Awareness* refers to the ability of

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

	Same Time	Different Time
Same Place	face-to-face interaction	asynchronous interaction
Different Place	synchronous distributed interaction	asynchronous distributed interaction

Figure 1. Taxonomy of computer-supported collaborative work.

distributed collaborators to keep track of each other's activities and coordinate their work; see the next section for a detailed description and discussion of awareness. *Shared artifacts* (or shared data) collectively refer to software entities on which collaborators perform their work. For example, in group editing (10,11), where multiple authors can work on the same document at the same time, the documents being edited would constitute the shared artifacts. In an online chat or instant messaging (IM) session, the exchanged messages would collectively form the shared artifacts. For collaborative software development, the shared artifacts may include requirements and design documents, source code files, bug reports, and code libraries/packages. As such, shared artifacts always reflect the current state of collaborative work.

An important issue in providing shared artifacts in distributed collaborative work is the system architecture (i.e., the exact manner in which shared artifacts are shared and accessed by collaborators). In a centralized architecture, shared artifacts are stored and maintained at a single location, and collaborators access and make updates on them (12,13). In a replicated architecture, collaborators work on their own copies of shared artifacts (14) and the state is synchronized among the copies synchronously or asynchronously. In general, it is easier to maintain the same state of shared artifacts under concurrent updates for all the collaborators in a central architecture than doing so in a replicated architecture. However, there are two drivers for use of replicated architectures. First is supporting disconnected use of shared artifacts. If a user is disconnected from the network or has poor connectivity, working on a local copy of a shared artifact is the only option, with merging of updates into other copies occurring later. Second, elapsed time between making an update and observing its effect on shared artifacts can be slow and unpredictable in a centralized architecture, especially over a wide area network, which can make it difficult to get interactive response times. With a replicated architecture, updates can be made locally and then immediately distributed to other participants. Conflicting updates can, of course, occur, and a number of solutions have been developed by the CSCW community, ranging from use of locks to operation transforms that result in the same state at all sites, even if participants issue operations in parallel on the same object (10,11).

Distributed software development typically has a hybrid architecture, where the shared artifacts (e.g., source code files and other documents) are stored and maintained at a central (and often remote) location, but separate local copies are created per user at check-out time, and any conflicts between multiple versions of the same artifact are detected and (manually) resolved at check-in time by means of a version control system [e.g., CVS (9)]. This architecture is feasible, in part, because once separate copies of, say, source code files, are created, programmers tend to work on them in isolation, and thus, the issues of providing a high level of interactivity among collaborators and having to always synchronize the state of shared artifact replicas do not occur. Later in this article, we describe an emerging programming practice, called Pair Programming (15), in which a pair of programmers works on the same code at the same time, and how they coordinate their activities.

In addition to awareness and shared artifacts, another concept is that of a *shared workspace*. A shared workspace provides a sense of place where collaboration takes place. It is generally associated with some part of the screen real estate of the user's computer where the user "goes" to work on shared artifacts, discovers work status, and interacts with his/her collaborators. It is often supported by the graphical user interfaces (GUIs) (i.e., "windows") of application tools used for working on shared artifacts. CSCW systems exist that are specifically designed to provide a shared workspace [e.g., XTV (16) and TeamRooms (17)]. Regardless of how it is provided, the main function of a shared workspace is to provide awareness of work status and activities of individual collaborators on shared artifacts, which, in turn, are essential for coordinating and controlling collaboration and achieving group goals.

One key issue in providing a shared workspace is the degree to which users' activities in the workspace are "public" or known to their collaborators. To illustrate, consider XTV (16), which is a predecessor to Microsoft NetMeeting in terms of its ability to support application sharing. Specifically, XTV allows any X Windows applications to be shared between multiple, distributed hosts by capturing X Windows screen update events at the server host in real time, where the shared application is running, and distributing the captured events to a client host, which draws the windows of the shared application by replaying the received events. Therefore, in XTV, all the user actions with the shared application and resulting updates to the application windows are shown to everyone. That is, the shared workspace provided by the windows of the shared application is totally public.

On the other hand, other systems allow collaborators to perform both private and public activities in the shared workspace. For example, DistView (18) allows the windows of a shared application to be selectively shared (i.e., users have control over which application windows are shared and when). Specifically, DistView allows users to export windows at any time during application execution, which then become available for others to import. When a window is thus shared, it always shows a synchronized view of application data being displayed even when collaborators update the data and maintains the same physical attributes (e.g., window size). In DistView, a shared window also shows a telepointer that indicates the mouse movements of the current owner of the window, where the ownership of the window can be passed among collaborators. Therefore, in the same shared application, DistView provides both a public workspace through shared windows and a private workspace through unshared windows of the application.

Different approaches are appropriate for different applications. For example, XTV and other application sharing systems (e.g., Microsoft NetMeeting) are well suited for distributed meeting or presentation applications, where a well-defined, formal role of a coordinator or presenter exists whose shared artifacts (e.g., viewgraphs) and actions on them, such as transitioning to a new viewgraph, should be visible to everyone in their entirety. However, they may not be appropriate for other applications, including large-scale collaborative science (19) and distributed software development, where collaborators largely work in private, and collaboration often occurs on an as-needed basis. For such applications, the ability to provide private and public shared workspace and to allow users to make transitions between the two types of shared workspaces on demand is critical.

AWARENESS

Software development is inherently a collaborative activity, which typically involves a team of programmers, testers, managers, and customers working together over a period of time. Team members collectively produce a number of artifacts, including requirements documents, design documents, progress reports, and software modules/components. Team members may be collocated and can work together at the same time (e.g., as in a single-site corporate environment) or may be distributed over geographical distance and time (e.g., as in an outsourcing situation). In either case, critical to the success of a software development team is the ability to coordinate activities of team members, which is largely facilitated by the means of what CSCW researchers refer to as *awareness*(20,21).

In general, a group of people working together requires some “sense” of who is working on what and when, and “feel” for where the group work stands with respect to what the team is ultimately trying to achieve. Such awareness information plays a critical role in the effectiveness and success of team work. As explained in Ref. 22, it “provides a context for your own activity. This context is used to ensure that individual contributions are relevant to the group’s activity as a whole, and to evaluate individual actions with respect to group goals and progress. The information, then, allows groups to manage the process of collaborative working.”

Therefore, many software engineering practices and methodologies include rigorous provisions for providing adequate and timely awareness of work status and progress. For example, Extreme Programming is a recent software development methodology that emphasizes informal and frequent requirements gathering, communication, testing, and (customer) feedback (23). Part of the Extreme Programming practice includes daily, stand-up meetings, in which each of development team members report on: “the prior day’s accomplishments; any obstacles, difficulties, or stumbling blocks faced; and what he or she plans to accomplish during the current day on the basis of the selected stories and tasks” (23). It also includes use of a new programming paradigm, called Pair Programming, in which, to quote Ref. 15, “two programmers working side-by-side, collaborating on the same design, algorithm, code, or test. One programmer, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects and also thinks strategically about the direction of the work. On demand, the two programmers can brainstorm any challenging problem. Because the two programmers periodically switch roles, they work together as equals to develop software.” In Pair Programming, the membership of a team may change as needed over the project lifecycle, which has a side effect of spreading in-depth knowledge about the design and status of individual software modules and components under development throughout the entire project team.

Although studies exist that apply the Extreme Programming practices in a distributed environment [e.g., (24,25)], its primary methods of providing awareness are built on the assumption that team members are collocated (to the extent that they can get together at a common place without difficulty) and are mostly available for collaboration at the same time. The issue of providing awareness becomes significantly difficult when team members are distributed across geographical distance and time. In distributed environments, collaborators are forced to interact with each other via electronic/computer-based media (e.g., audio/video conferencing and messaging systems), which significantly limits the availability of sensory, physiological, and environment cues (e.g., eye contacts, hand gestures, and audience attention on individuals) which are readily and seamlessly communicated in face-to-face meetings, shared offices, or collocated cubicle environments and are found to play a critical role in coordination and control of collaborative activities (6). Furthermore, information and knowledge sharing may be significantly reduced in distributed environments as opportunities do not exist for informal interactions among team members (e.g., water-cooler conversations, chance meetings in hallways, and “dropping by” colleagues’ offices for small talks). Without any provision, collaborators would have to diligently document coding, decision making, design, and other activities; collect support materials; and communicate them (say via e-mail) with their colleagues, who, in turn, would have to carefully review and understand the received materials to keep up-to-date with the current status of their work. Most of these tasks are usually “add-ons” to already heavy workloads, which can lead to significant loss of productivity. Using “off-the-shelf” audio/video conference bridges and project management tools is helpful but cannot replace rich awareness information that is readily available in collocated environments.

Therefore, much research in CSCW has focused on providing awareness for distributed teams of collaborators as seamlessly and effortlessly as possible. For example, DistEdit-based group editors (10,11) allow multiple, geographically distributed users to edit the same document at the same time and include a locking mechanism for allowing concurrent updates, in which the "locked regions" of the document are color-coded per individual author. These serve as an awareness mechanism, by which concurrent authors can easily tell who is working on which part of the document and thus avoid conflicting edits. TeamRooms (17) provides a desktop window that functions as a room-based virtual environment, where distributed users can place tools (e.g., for leaving notes for each other, text-based chatting, and brainstorming) and resources (e.g., URLs to online documents or websites) for group use. In addition to a "buddy list" of users currently in the room, TeamRooms provides a "radar view," a miniaturized version of the shared window that shows both the part of the room each user is currently viewing and the tools each user may be using. The radar view conveys some awareness of what each user is doing in the shared space. Application (or screen) sharing systems (e.g., XTV (16) and Microsoft NetMeeting) turn the desktop screen of a collaborator, in part or in their entirety, into a shared workspace on demand and allow distribute collaborators to closely keep track of each other's activities in highly interactive and synchronous sessions.

Distributed software development is a special form of collaborative work that requires a high level of awareness among team members at all times to seamlessly integrate, maintain, and keep track of progress of various software modules and related artifacts (e.g., specification of functional and operational requirements, design documents, change requests, and bug fixes). Awareness is also required to avoid conflicts and duplicate work among distributed members of development and administrative teams. Examples of awareness information specific to distributed software development include the following: who is working on what software modules, causal and temporal dependency relationships between different software modules, change history of a software module (including who has made changes and why), history of bug reports and fixes, and constituent components of a software release and their version information. In distributed software development, awareness should not only be provided in a timely and unobtrusive manner so as not to disrupt ongoing work but also persist over a long period of time (possibly beyond the lifetime of a software project) to facilitate knowledge building and information sharing among geographically and temporally distributed team members. In the remainder of this article, we describe and discuss various approaches to providing awareness for collaborative software development by distributed teams.

APPROACHES TO COLLABORATIVE SOFTWARE DEVELOPMENT IN DISTRIBUTED ENVIRONMENTS

In this section, we describe a few exemplary approaches to supporting collaborative software development for distributed teams. We note that although each of the described approaches is different in terms of specific mechanisms, the fundamental objective is the same, which is to provide adequate awareness to collaborating teams of software developers and managers to help them achieve a high level of activity coordination and efficiency in a distributed environment.

Virtual Environments as Shared Workspace

Providing shared workspace for distributed software developers and managers has been one of the main topics for CSCW researchers. One of the earlier examples in the area is ConversationBuilder (26), in which each software development activity (e.g., bug tracking) is modeled as a conversation. A conversation is essentially a software development process for a specific task and may specify a set of artifacts relevant to the task (e.g., code files and documentation), an (ordered) list of actions to be performed on the artifacts, user roles under which actions are to be performed, and per-artifact capability objects for specifying what actions can be performed on the artifact and by whom. Associated with a conversation is the concept of a conversation space, a virtual space where participants in a conversation can keep track of the current state of a conversation. To support real-life work patterns of software developers and managers, ConversationBuilder allows users to participate in multiple conversations at the same time. To accommodate new tasks, ConversationBuilder also allows users to create and enact new conversations. ConversationBuilder stores shared artifacts in a conversation in a hypertext database and provides a suite of client applications for providing configuration and versioning information of shared artifacts and graphically visualizing "relations between nodes in the hypertext, and relations among a user's conversations" (26).

Although conversation space objects in the ConversationBuilder function as the shared workspace among distributed conversation participants, they do not provide the sense of real space in the physical world; they are not intended to reflect real-life entities in a virtual world. In contrast, much research has been conducted in providing an online shared workspace built on real-life concepts or entities in the physical world. One of the main goals of such works is to minimize the "learning curve" of distributed users in adapting to and making effective use of virtual workspaces by building them with metaphors for familiar concepts, organizations, and objects that are commonly encountered in everyday life. And many such reality-based virtual workspaces have been built on the facilities provided by a pioneering virtual environment system, called Multi-User Dungeons (MUDs) (27).

Initially developed for online, multi-user gaming, a typical MUD environment includes a number of rooms that are connected with each other (via doors and hallways), objects that are contained in rooms, tools for working on or changing some state of objects, and players who represent human users. Players navigate a MUD environment by visiting connected rooms and (try to) achieve their goals by finding and performing operations on appropriate objects with provided tools. A key element in any MUD environment is that online players can meet each other, either by chance or by appointment in some room(s), and interact with each other (i.e., to collaborate or compete with each other). MUDs are generally implemented in a client-server system. That is, distributed users connect to MUD servers by using MUD client applications, which also provide a user interface (UI) for displaying a map of rooms, navigating rooms, and issuing commands. The earlier (and still predominant) means of providing UI is text-based, in which users specify their behaviors (e.g., "look east") or issue commands (e.g., "pick up") at the command line.

At the core of a MUD system is a MUD engine, a software entity that functions much like an operating system (OS) in that it allows a variety of custom systems to be built. These systems still retain the basic MUD concepts, such as rooms, objects, commands, and players, but apply them to different application domains other than online gaming and entertainment. Over the years, the number and variety of MUD-based systems have rapidly grown to the extent that a new phrase, Multi-User Dimensions, has been invented to refer to those systems in new application areas, which include online communities, distance learning, and collaborative network administration.

Given the built-in concepts of rooms, objects, and players, it is no surprise that MUDs have also been used to provide shared virtual workspaces and access to shared artifacts in collaborative, distributed software development (28,29). In a typical use scenario, distributed developers would enter a MUD environment by connecting to a MUD server specifically set up for their work and start navigating rooms. In a given room, a developer may find objects that represent project artifacts (e.g., source code files) and issue commands on the objects (e.g., "build"). A command issued in the virtual environment usually causes some preconfigured tools (e.g., a complier and linker in an IDE) to start executing out in the "physical world" (e.g., on the developer's local host). Online developers may encounter others who are connected to the MUD server at the same time and may decide to "chat" with each other via available communication tools in the environment. They may also create special-purpose rooms for, say, conducting scheduled meetings, and senior members of a project may provide "guided tours" of the project to newcomers by navigating rooms while conducting chat sessions with them at the same time (29). The presentation of a virtual environment may range from a text-based UI to a 3-D rendering of rooms, hallways, objects, and developers (via avatars).

When developing MUD-based virtual environments for distributed software development, one major issue to consider is how to map concepts, abstractions, and entities of software engineering to MUD's room-based metaphors. Depending on what a room represents, the semantics of a developer visiting a room can dramatically change. For example, if a room represents a specific activity to be performed as part of a software process, the objects contained in the room could be tools and artifacts needed for performing the activity (28). Developers can be assigned to specific activities (with appropriate access rights) in the process and work on them by visiting corresponding rooms. Constraints can be set for exiting a room to ensure that, for example, the corresponding activity has been completed to the extent that the downstream activities in the process can be performed. In another approach, rooms can represent software artifacts themselves (29), in which, for example, a room represents a software module, and objects contained in the room represent individual class files that make up the module. In this approach, visiting a room would mean that the visitor intends to work on the corresponding artifact. A comprehensive discussion of different mapping approaches to model software processes in MUD-based virtual environments can be found in Ref. 28.

Different mapping approaches have their own advantages and disadvantages. For example, when rooms represent software process activities, it is convenient to model the entire software process by way of connecting rooms and establishing appropriate exit constraints. However, modeling a software developer who works on multiple tasks at the same time is more difficult (28). When rooms represent software or project artifacts, dependency relationships among them can easily be made explicit by connecting appropriate rooms together. However, it would be not only difficult to support the multithreading work practices of individual project members but also inconvenient to represent the overall context in which modeled artifacts are used. Determining which approach to use in a given project is not trivial and depends on many factors, including the project size and required level of realism.

Awareness and Visualization Widgets

Providing awareness in virtual environments generally requires that some extra work be performed to first set up an appropriate environment (e.g., create rooms, objects, and commands in a MUD-based system). Furthermore, virtual environments are typically separated from the tools and systems that software developers and managers use to perform their work, which often means that users have to stop their work, context-switch, and go to a virtual environment to receive and generate awareness information. All of these factors can incur extra administrative and usage overheads and reduce the effectiveness of virtual environments. Ideally, users should be able to keep track of project status and others' activities without much effort and without intervening with their own work.

To this end, researchers have been working to bring awareness into users' "regular" workspaces. For example, Jazz (2,3) is an Eclipse-based integrated development environment (IDE) that includes a suite of embedded tools and mechanisms for providing awareness and enabling communication for a team of distributed developers. The basic idea is that "from the individual developer's perspective, the IDE is where coding takes place and is the home of many different development tools. If coding is a team effort, then why not add collaborative capabilities to the IDE toolset alongside the editor, compiler, and debugger?" (2). Specifically, Jazz provides a "buddy list" of developer team members as part of the IDE workspace. From this list, called the Jazz Band, team members can see the online status of each other, initiate multimedia communications sessions, and infer who is currently engaged in what activities and with whom. It also allows team members to initiate chat sessions from selected sections of code, save the exchanged messages as an annotation to the selected code, and review them at a later time. In addition, the folder and file list in the Jazz IDE provides version control information (e.g., who has checked out what items and when, update status on local copies of checked-out items, and commit status). These features allow developers to seamlessly keep track of the current status of their work as a whole and spontaneously initiate discussion and share knowledge without having to leave the IDE workspace, thus helping reduce "costs" associated with acquiring, generating, and using awareness to coordinate and collaborate in distributed software development.

CSCW tools, such as Jazz, have, as yet, not been widely used as a collaboration mechanism for distributed software developers. This use may only be a question of time, because it takes time for a new technology to be widely used, but another reason may be that, although while software development is a collaborative activity in terms of planning, designing, coordination, testing, and integration, writing code has mostly been considered as an isolated, individual activity. This perception of code-writing has recently been challenged by the advent of Pair Programming as part of the Extreme Programming (15). Pair Programming has been shown to enable (collocated) programmers to be more productive (in terms of lines of code produced per day per programmer) and to produce a higher quality of code (23). This has, in turn, provided a strong motivation for adapting Pair Programming in distributed team environments. For example, Baheti et al. (25) have created a Distributed Pair Programming (DPP) environment using Microsoft NetMeeting and have evaluated the performance of distributed student teams on a large class project. Their study has found that "software development involving Distributed Pair Programming is comparable to that developed using collocated pair programming or virtual teams without distributed pair programming." Ho et al. (24) has created a Eclipse plug-in module, called Sangam, to allow DPP in an Eclipse-based Java development environment. Sangam uses replicated state model for shared artifacts, allowing distributed partners to participate in Pair Programming without having to first synchronize their screen resolutions or refresh rates and without requiring a high network bandwidth, as is often the case with using screen sharing across a wide area network. However, it remains to be seen how well DPP would apply to large-scale projects, where team members are distributed not only geographically but also temporally across different time zones.

Another, and perhaps surprising, source of awareness for distributed software development teams is configuration management and version control systems [e.g., (CVS) (7,9)]. Dix (30) observes that when an artifact is shared in collaborative work, it is "not only the subject of communication, it can also become a medium of communication. As one participant acts upon the artifact, the other observes the effects of the action. We call this observation by the other participant feedthrough." Configuration management and version control systems work as a feedthrough mechanism by enabling users to keep track of who has worked on what modules, avoid concurrent and conflicting updates, and know whom to work with to integrate separately developed versions of the same module by examining the check-in and check-out state (or history) of individual modules. By maintaining interdependencies among software modules and their connections to other artifacts (e.g., design and requirements documents), these systems also allow users to help establish and maintain the context of their work. In addition, the logs of problem reports and fixes, along with developer descriptions and comments on the nature of a given bug and the rationale for its fix, function as a group memory mechanism, which helps newcomers to the project get up to speed, even when original members no longer work on the project or work in the same place (7).

However, configuration management and version control systems do not provide a comprehensive overview of the complete product under development and require a considerable amount of time and effort on the part of users to produce meaningful comments and descriptions of their development activities (7). Furthermore, the UI for discovering and gaining access to appropriate information in these systems is usually primitive and difficult to use.

To address these issues, research has been performed to provide the graphical means of visualizing and accessing the version history, current check-out (or owner) state, and known issues and fix status of all software modules and project artifacts. Also often represented in such visualizations are the release history of a given product and version information and dependency relationships among software modules and project artifacts that constitute a given release. The visualization techniques widely vary from a color-coded line representation of source code lines for representing age and authorship of corresponding code to a hypertext-based, interactive 3D view of source code files and interdependencies. See Ref. 4 for a comprehensive survey of exemplary works in this area. Most of these systems perform a syntactic analysis of source code files and collect change history data from a version control system to generate information to be visualized.

Open-Source Software Development

Open-source software development involves a largely dispersed group of software developers who do not necessarily know each other but have voluntarily come together to work on software problems and issues to achieve common goals. Over the years, a large number of large-scale open-source software projects have been undertaken that have not only been successful in terms of producing high-quality, high-performance software systems and tools but also have had a significant impact on the entire Information Technology (IT) and software industry. A few notable examples include: Linux operating system, Apache Web server, Mozilla Web browser, and Xerces XML parser from the Apache XML Project.

Open-source software development represents "an extreme case of geographically distributed development, where developers work in arbitrary locations, rarely or never meet face to face" (5). In addition, the process of open-source software development is not well-defined and lacks "many of the traditional mechanisms used to coordinate software development, such as plans, system-level design, schedules, and defined processes," which are "generally considered to be even more important for geographically distributed development than for collocated development" (5). There is no explicit or formal division of work (i.e., who is responsible for what) and "developers can contribute to any part of the code" (1). Furthermore, "no formal quality control programs exist and no authoritative leaders monitor the development" (6).

Given this seemingly chaotic environment, what is perhaps more surprising is that open-source developers do not employ sophisticated awareness and coordination mechanisms, other than e-mail (developer mailing lists) and version control systems [e.g., CVS (9)]. In general, e-mail (or any text-based chat tools) would not be considered as an effective or user-friendly means of providing awareness in distributed environments as e-mail is not directly used in producing/manipulating artifacts of software development. Thus, to use e-mail as the means of providing awareness implies that it is the responsibility of users (i.e., software developers and project managers) to manually compose and distribute necessary awareness information, which, in turn, would (significantly) increase the overall workload of individual users. Although a version control system is often part of a software development process, its main use is to allow users to detect and resolve conflicting code changes to the same source files according to some pre-defined policies. As discussed earlier in the section, without further provision, it is not easy to use a typical version control system as a useful awareness tool in distributed software development environment.

Despite the lack of formal processes and integrated, automated means of generating and distributing awareness, open-source developers are able to keep track of current project status by fostering an online culture of "keeping it public" (1). They create and subscribe to developer, bug-tracking, and other project-related mailing lists and take it upon themselves to carefully answer questions and closely follow discussion threads and status reports, even if they may not be directly relevant to their current tasks or interests. In addition, many subscribe to (CVS) commit logs so that they would be notified whenever code updates are made. Mailing list messages and commit log entries are also archived, and publicly accessible "how-to" and other project-related documents are provided so that newcomers to the project can be brought up-to-date without having to ask too many "newbie" questions. Open communications and public discussions are strongly encouraged to the extent that "if it doesn't happen on list, it doesn't happen" (1), and those members who do not follow the established protocol and commit code changes without first acquiring consensus from other members via public discussion may be publicly discredited. All of these factors combine to create the effect of "overhearing conversations" in open, collocated work environments and allow open-source developers to keep informed of who the "gurus" are in what subject areas, who are working on what, and who are responsible for ensuring integrity and functionality of what parts of the system/application under development.

In addition, an organizational hierarchy often emerges in which a relatively small group of contributors form a core development group that defines, designs, and implements the main functionality and architecture of the system under development, while the others become users, testers, and implementers of add-on features (1). There is no formal process for determining who should be assigned to which group. Rather, "leaders" emerge based on the level and quality of participation and (perceived) expertise in subject areas. In addition to implementing the core functionalities and maintaining the integrity of modules they are responsible for, core developers may also define and then refine (via public discussions) application programming interfaces (APIs) for add-on feature development by other contributors. On the other hand, the user group of an open-source project, which is typically much bigger in size than the core developer group (for example, see Ref. 6), provides "enough eyeballs" to catch and report bugs, which prompt core developers to create and distribute patches.

CONCLUSION

Critical to successful software development in distributed environments is awareness of current work status and activities of distributed team members (i.e., who is doing what, when, and why), which, in turn, enables seamless coordination and conflict avoidance and detection. Providing the right awareness at the right time and to the right people has been an active area of research in the field of CSCW. In this article, we have introduced and discussed key CSCW concepts (i.e., shared artifacts and shared workspace) and design issues related to providing awareness in distributed software development. In addition, we have described and discussed common collaboration practices in open-source development efforts. In sharp contrast with ongoing awareness research in CSCW, open-source developers successfully employ general-purpose communication and coordination tools (e.g., e-mail mailing lists, version control systems, and bug tracking systems) to provide awareness to a large number of widely distributed teams of volunteer software developers.

At first glance, the findings from open-source development communities seem to suggest that specialized awareness facilities are not really required in practice. However, a deeper analysis shows that open-source developers can effectively use general-purpose tools for generating and gathering awareness mainly because they work very hard at it. As described earlier, open-source developers carefully and rigorously document their actions and activities and share with others by posting them on online, public forums (e.g., e-mail mailing lists). They also diligently follow others' postings and dutifully answer questions. All of these activities take much time and effort on the part of individual contributors.

Thus, it would appear that open-source development can greatly benefit from use of specialized awareness utilities (1). However, employing such a tool would require a homogenous run-time environment for the tool, fresh download and installation of the tool by everyone, and, more importantly, commitment by everyone that they would regularly use the tool as they do with e-mail or CVS. All of these aspects may be very difficult to achieve among distributed developers who do not know each other and are not under control of any formal authority. As such, use of e-mail and CVS as sources of awareness information may have resulted, not from the superior utilities of these tools as an awareness mechanism, but from necessity and convenience of not having to deploy, learn, and use new software. This observation is in line with the current trend of online discussions and bug reporting and tracking operations migrating to the World Wide Web (WWW), which allows developers to access the same awareness information as before by using the single most widely deployed and used software application today, the Web browser.

The above observations do not account for the fact that specialized awareness facilities have not been widely adapted in software development, which we believe is largely because we do not yet have a good understanding of both complexities and subtleties of software development work and interaction patterns and requirements and work habits of software developers and managers. Most of the existing awareness facilities for distributed software development have been adapted from those developed for general workspace environments and thus may not be able to meet awareness requirements specific to distributed software development activities. Much research is still required to better understand distributed software development as a unique form of collaborative work.

BIBLIOGRAPHY

1. C. Gutwin, R. Penner, and K. Schneider, Group awareness in distributed software development, *Proc. 2004 ACM Conference on Computer Supported Cooperative Work*, Chicago, IL, 2004, pp. 72–81.
2. L.-T. Cheng, C. R. B. de Souza, S. Hupfer, J. Patterson, and S. Ross, Building collaboration into IDEs, ACM Queue, **1**(9): 40–50, 2004.
3. S. Hupfer, L.-T. Cheng, S. Ross, and J. Patterson, Introducing collaboration into an application development environment, *Proc. ACM 2004 Conference on Computer Supported Cooperative Work*, Chicago, IL, 2004, pp. 21–24.
4. M.-A. D. Storey, D. Čubranić, and D. M. German, On the use of visualization to support awareness of human activities in software development: A survey and a framework, *Proc. 2005 ACM symposium on Software visualization*, St. Louis, Missouri, May 14–15, pp. 193–202.
5. A. Mockus, R. T. Fielding, and J. D. Herbsleb, Two case studies of open source software development: Apache and Mozilla, ACM Trans. Software Eng. Methodol. (TOSEM), **11**(3): 309–346, 2002.
6. Y. Yamauchi, M. Yokozawa, T. Shinohara, and T. Ishida, Collaboration with lean media: How open-source software succeeds, *Proc. 2000 ACM Conference on Computer Supported Cooperative Work*, Philadelphia, PA, 2000, pp. 329–338.
7. R. E. Grinter, Using a configuration management tool to coordinate software development, *Proc. Conference on Organizational Computing Systems*, Milpitas, CA, 1995, pp. 168–177.
8. C. A. Ellis, S. Gibbs, and G. Rein, Groupware: Some issues and experiences, *Comm. ACM*, **34**(1): 38–58, 1991.
9. P. Cederqvist, Version Management with CVS, Technical Report. Available: http://www.cvshome.org/files/documents/19/532/cederqvist-1.11.18.pdf, 2004.
10. M. Knister and A. Prakash, Issues in the design of a toolkit for supporting multiple group editors, *Computing Systems—J. Usenix Assoc.*, **6**(2): 135–166, 1993.
11. M. Knister and A. Prakash, DistEdit: A distributed toolkit for supporting multiple group editors, *Proc. Third Conf. on Computer-Supported Cooperative Work*, Los Angeles, CA, 1990, pp. 343–355.
12. P. Dewan and R. Choudhary, Flexible user interface coupling in collaborative systems, *Proc. ACM CHI'91 Conference on Human Factors in Computing Systems*, 1991, pp. 41–48.
13. J. Patterson, R. Hill, S. Rohall, and W. Meeks, Rendezvous: An architecture for synchronous multi-user applications, *Proc. ACM 1990 Conference on Computer Supported Cooperative Work*, Los Angeles, CA, 1990, pp. 317–328.
14. R. Hall, A. Mathur, F. Jahanian, A. Prakash, and C. Rasmussen, Corona: A communication service for scalable, reliable group collaboration systems, *Proc. ACM 1996 Conference on Computer Supported Cooperative Work*, Boston, MA, 1996, pp. 140–149.
15. Pair Programming, Available: http://www.pairprogramming.com.
16. H. Adbel-Wahab and M. Feit, XTV: A framework for sharing X window clients in remote synchronous collaboration, *Proc. IEEE Tricomm '91: Communications for Distributed Applications and Systems*, Chapel Hill, North Carolina, April 18–19, 1991.
17. M. Roseman and S. Greenberg, TeamRooms: Network places for collaboration, *Proc. of the ACM 1996 Conference on Computer Supported Cooperative Work*, Boston, MA, 1996, pp. 325–333.
18. A. Prakash and H. Shim, DistView: Support for building efficient collaborative applications using replicated objects, *Proc. ACM 1994 Conference on Computer Supported Cooperative Work*, Chapel Hill, NC, 1994, pp. 153–164.
19. S. Subramanian, G. R. Malan, H. S. Shim, J. H. Lee, P. Knoop, T. E. Weymouth, F. Jahanian, and A. Prakash, Software architecture for the UARC Web-Based collaboratory, *IEEE Internet Comput.*, **3**(2): 46–54, 1999.
20. J. Hill and C. Gutwin, Awareness support in a groupware widget toolkit, *Proc. 2003 International ACM SIGGROUP Conference on Supporting Group Work*, Sanibel Island, FL, 2003.
21. C. Gutwin and S. Greenberg, A descriptive framework of workspace awareness for real-time groupware, *Computer Supported Cooperative Work*, **11**(3): 411–446, 2002.
22. P. Dourish and V. Bellotti, Awareness and coordination in shared workspaces, *Proc. 1992 ACM Conference on Computer-Supported Cooperative Work*, Toronto, Ontario, Canada, 1992, pp. 107–114.
23. L. Williams, The XP programmer: The few minutes programmer, *IEEE Software*, May/June 2003.
24. C.-W. Ho, S. Raha, E. Gehringer, and L. Williams, Sangam – A distributed pair programming plug-in for eclipse, *OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop*, Vancouver, British Columbia, Canada, 2004, pp. 73–77.
25. P. Baheti, E. Gehringer, and D. Stotts, Exploring the efficacy of distributed pair programming. XP Universe 2002, Chicago, IL, August 4–7, 2002.
26. S. M. Kaplan, W. J. Tolone, A. M. Carroll, D. P. Bogia, and C. Bignoli, Supporting collaborative software development with ConversationBuilder, *Proc. 5th ACM SIGSOFT Symposium on Software Development Environments*, Tyson's Corner, VA, 1992, pp. 11–20.

27. P. Curtis and D. Nichols, MUDs grow up: Social virtual reality in the real world, *Proc. Third International Conference on Cyberspace*, 1993.

28. J. C. Doppke, D. Heimbigner, and A. L. Wolf, Software process modeling and execution within virtual environments, *ACM Trans. Software Eng. Methodol. (TOSEM)*, **7**(1): 1–40, 1998.

29. S. E. Dossick and G. E. Kaiser, CHIME: A metadata-based distributed software development environment, *ACM SIGSOFT Software Eng. Notes*, **24**(6): 464–475, 1999.

30. A. Dix, *Computer Supported Cooperative Work – A Framework, Design Issues in CSCW*, D. Rosenburg and C. Hutchinson, (eds.), New York: Springer-Verlag, 1994, pp. 23–37.

HYONG-SOP SHIM
Telcordia Technologies
Piscataway, New Jersey
ATUL PRAKASH
University of Michigan
Ann Arbor, Michigan
JANG HO LEE
Hongik University
Seoul, Korea

EMBEDDED OPERATING SYSTEMS

INTRODUCTION

Many of the systems and devices used in our modern society must provide a response that is both correct and timely. More and more computer systems are built as integral parts of many of these systems to monitor and control their functions and operations. These embedded systems often operate in environments where safety is a major concern. Examples range from simple systems, such as climate-control systems, toasters, and rice cookers, to highly complex systems such as airplanes and space shuttles. Other examples include hospital patient-monitoring devices and braking controllers in automobiles.

We use operating systems (1) as interfaces between computer applications and computer hardware. Most noticeably, operating systems are used to access and control operations in desktop and notebook (laptop) personal computers (PCs). You are probably familiar with one or more of the following operating systems: Linux, Microsoft Windows (XP, NT, 2000, 98, 95), Apple Mac OS X, and UNIX.

In order to conveniently use a PC, we must first install and run an operating system. Operating systems are not only used to operate PCs, but also other types of microprocessor-driven devices, such as personal digital assistants (PDAs), which use smaller versions of PC operating systems such as Palm OS, Windows Pocket PC, and Embedded Linux (Embedix). These PDAs do not have a secondary memory and their main memory can vary in size from 8 MB to 64 MB. Processor speed may vary from several MHz to 400 MHz (Intel Xscale 400 MHz processor).

You will find operating systems even in devices whose main functions are not computation, such as DVD (digital video disk) players and VCRs (video cassette recorders). Microprocessors together with scaled-down versions of larger operating systems are embedded in these systems to control their operations. Time-critical or real-time systems use real-time operating systems, such as Wind River's VxWorks, which are more deterministic.

We can define an operating system (OS) as a program that provides a convenient environment for embedded applications consisting of multiple tasks. System calls are used for process/task management, memory management, input/output (I/O) drivers, and time delay. Error handling and recovery are also provided. An OS allows efficient sharing of resources among tasks in a single-user system or among users (each with one or more tasks) in a multiple-user system.

The goal of conventional, non-real-time operating systems is to provide a convenient interface between the computer applications and the computer hardware while attempting to maximize average throughput, to minimize average waiting time for tasks, and to ensure the fair and correct sharing of resources. However, meeting task deadlines is not an essential objective in non-real-time operating systems because its scheduler usually does not consider the deadlines of individual tasks when making scheduling decisions.

For real-time applications in which task deadlines must be satisfied, a real-time operating system (RTOS) with an appropriate scheduler for scheduling tasks with timing constraints must be used. Since the late 1980s, several experimental as well as commercial RTOSs have been developed, most of which are extensions and modifications of existing OSs such as UNIX. Most current RTOSs conform to the IEEE POSIX standard and its real-time extensions (2–4). Commercial RTOSs include LynxOS, RTMX O/S, QNX, VxWorks, and pSOSystem.

LynxOS is LynuxWorks' hard RTOS based on the Linux operating system. It is scalable, Linux-compatible, and highly deterministic. LynuxWorks also offers BlueCat Linux, an open-source Linux for fast embedded system development and deployment. RTMX O/S has support for X11 and Motif on M68K, MIPS, SPARC, and PowerPC processors. VxWorks and pSOSystem are Wind River's RTOSs with a flexible, scalable, and reliable architecture, and it is available for most CPU platforms. Here, we use VxWorks (5) to illustrate several RTOS features. This article is organized as follows: the next section describes process synchronization, followed by an introduction of real-time scheduling, a discussion on memory management, a focus on input/output issues, and finally, a conclusion.

INTERPROCESS COMMUNICATIONS

A process (or task) is the basic unit of work in a computer system. Here, we use the terms "process" and "task" interchangeably. A third concept is the "thread," which is usually defined as a lightweight process but with less overhead for its maintenance. Unless the system is very simple, there is usually more than one process in a real-time system. In a uniprocessor system, processes interleave their executions, giving the appearance of concurrent processing.

Processes can communicate with one another in a number of ways: (1) accessing (reading and/or writing) shared memory containing data structures; (2) via pipes or message queues; (3) using sockets (for interprocessor communication in a network) or socket-implemented remote procedure calls (RPC); and (4) signals.

As more than one process may attempt to access the same shared data structure at the same time, approach (1) requires mutual exclusion, which can be achieved by one of the following methods: (a) disabling of interrupts, (b) disallowing preemption, or (c) using semaphores.

In solution (a), interrupts are disabled before an access to the shared resource, thus the running process can access this resource while being the only process running in the CPU without the possibility of being interrupted by another ready process. After accessing this shared resource, interrupts are enabled again. This approach is the most

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

inefficient because time-critical processes or interrupt service routines (ISRs) responding to external events may not run even if they do not access the same shared resource while interrupts are disabled. Even in non-real-time systems, this solution is not appropriate for user applications.

In solution (b), the running process accessing the shared resource cannot be preempted by any other process (even with a higher priority than the running process) except ISRs. This solution can still lead to unacceptable real-time response and suffers the same problem as solution (a); that is, processes with higher priorities may not run even if they do not access the same shared resource while preemption is locked. The best mechanism for mutual exclusion and other synchronization problems in real-time and non-real-time systems is the semaphore described next.

Semaphores

A semaphore is like an arbiter for controlling access to a shared data structure, much like a traffic light used to control the flow of traffic (vehicles) passing through a shared intersection. Obviously, two vehicles cannot be simultaneously at the same spot; an attempt to do so would result in a collision. A semaphore can also be used to synchronize tasks and to guard multiple instances of a resource.

Conceptually, a semaphore is denned as follows:

Operations: wait and signal (P and V);

State of semaphore: (count, queue);

count ≥ 0 implies ‘count’ tokens (or privileges) are available;

count < 0 implies absolute value of ‘count’ $=$ number of processes waiting in this semaphore’s queue; and queue $=$ queue of processes waiting on this semaphore.

The wait and signal operations are defined as follows:

```
Wait(semaphore):
disable interrupts;
count = count - 1;
if count < 0 then
begin
    add calling process to semaphore queue;
    change this process’ state from running to
    waiting
    /* note that the calling process is now wait-
    ing in the queue */
end
enable interrupts;
return
Signal(semaphore):
disable interrupts;
count = count + 1;
if count <= 0 then
begin
    remove process from semaphore queue;
    change this process’ state from waiting to
    ready
    /* note that the removed process is not the
    calling process */
end
enable interrupts;
return
```

Note that we need to make the “wait” and “signal” operations atomic because they access and modify shared data (count and queue of waiting processes). This mutual exclusive access to shared data is ensured by disabling the interrupts before each operation and enabling the interrupts after each operation. Note that using a Test and Set Lock hardware instruction or having the real-time OS enforce the mutual exclusion are other ways to support the wait and signal operations, instead of disabling and enabling the interrupts.

In many OSs, especially embedded/real-time OSs, there are three types of semaphores optimized to handle different types of problems: (1) binary, (2) mutual exclusion, and (3) counting. All three types of semaphores are, in fact, defined the same way as given above; the initial value of the variable “count” determines the semaphore type.

UNIX, a general OS, provides the semaphore operations such as: semget() creates an array of one or more semaphores, semop() provides operations such as wait and signal on semaphores, and semctl() destroys a semaphore and deallocates associated memory. VxWorks, an RTOS, has the following semaphore operations: semBCreate() allocates and initializes a binary semaphore, semMCreate() allocates and initializes a mutual exclusion semaphore, semCCreate() allocates and initializes a counting semaphore, semTake() perfoms the wait operation on a semaphore, semGive() perfoms the signal operation on a semaphore, semFlush() unblocks all tasks waiting for a semaphore, and semDelete() destroys a semaphore and deallocates associated memory.

Now we solve several common problems with semaphores. Each example also illustrates the specific type of semaphore appropriate for the problem being solved.

Example 1: Sorting - A Synchronization/Ordering Problem. The problem is to sort an array of numbers by creating two tasks to sort each half of the array, and then merging the sorted halves into one sorted array. Before the merge can start, we have to ensure that the two sorting tasks must finish first. We use a binary semaphore “done” to solve this problem.

```
cobegin
    sort(l, n div 2)
    sort(n div 2 + 1, n)
coend
merged, n div 2, n)
Main process:
 :
done = create_semaphore(0);
create_process(first process);
create_process(second Process);
wait(done);
wait(done);
```

```
 :
Sort process:
 :
sorting steps;
signal(done);
terminate
```

Example 2: Mutual Exclusion. This standard mutual exclusion problem is to ensure that access to a shared resource or variable by more than one task results in a consistent value for this resource. Suppose that the access is to add one to the value of the shared variable. We solve this problem by first creating a mutual exclusion semaphore "mutex."

```
mutex = create_semaphore (1)
```

Then, we insert a wait operation before this mutually exclusive access (to the shared variable x) and a signal operation afterward.

```
Process a:
wait(mutex)
x = x + 1
signal(mutex)
Process b:
wait(mutex)
x = x + 1
signal(mutex)
```

Example 3: Buffer Pool - A Counting Problem. There are 10 temporary memory buffers, each can be allocated to one process only. The pseudocode for creating the counting semaphore "available_buffers," allocating a buffer, and releasing a buffer are shown below. Once the counter has been reduced to zero, the next process will be suspended waiting for a buffer to become available.

```
available_buffers = create_semaphore(10);
wait(available_buffers);
``allocate buffer'';
``return buffer to pool'';
signal(available_buffers);
```

The VxWorks RTOS provides the semCCreate() function to create a counting semaphore. For this example, we rewrite the above pseudocode using a VxWorks Wind counting semaphore, resulting in the following code:

```
available_buffers = semCCreate (10);
semTake(available_buffers, WAIT_FOREVER);
``allocate buffer'';
``return buffer to pool'';
semGive(available_buffers);
```

There are several simple programming rules that should be observed when using semaphores in concurrent programming. First, it is important that each task waits or signals the correct semaphore, so double-checking this process is critical in coding and debugging. Second, the value of variable "count" should conserve, that is, for each wait operation, there should be a corresponding signal operation that will be executed within a bounded period of time (or number of steps). We need to ensure that the execution of the tasks does not lead to deadlocks. Third, the initial value of variable count must be chosen carefully depending on the type of problem we are going to solve.

Real-Time Extensions

RTOSs often offer additional features to semaphores optimized for running real-time applications. We describe several such features in this section.

One feature employs priority inheritance algorithms to solve the priority inversion problem, which can occur when mutual exclusion is enforced. Priority inversion is a situation in which a higher-priority task is forced to wait for an indefinite period of time (which is not acceptable in a real-time system) in order for a lower-priority task to complete its execution.

The following example illustrates this situation. Suppose we have a preemptive system with a number of tasks including tasks A, B, and C. Task A has higher priority than task B, and task B has higher priority than task C. Task A and C may access the same resource controlled by a semaphore. At some point during their executions, task C (the lowest-priority task) has gained access to the resource. Now task A (the highest-priority task) waits on the mutual exclusion semaphore guarding this resource and is blocked (and must wait in this semaphore's queue), even though task A's priority is higher than task C's. This scenario is acceptable in a real-time system if task A does not need to wait longer than the time period (the critical section) for task C to use the resource. However, because this system is a preemptive system, task C may be preempted by task B (which has a higher priority) when it becomes ready and does not want to access the resource held by task C. Other tasks having higher priorities than task C's may continually preempt task C indefinitely, resulting in a indefinite waiting period for task A, which remains in the semaphore's queue.

A common solution to this problem is the priority inheritance algorithm or protocol. It ensures that a task holding a mutually exclusive resource executes at the priority of the highest-priority task waiting for this resource until it and all its previous instances, if any, signal the semaphore guarding this resource (that is, it has released all mutual exclusion semaphores for this resource). Then this task returns to its normal priority to continue execution. This protocol prevents this low-priority task accessing a mutually exclusive resource from being preempted for an indefinite period of time by tasks with lower priorities than that of the waiting task.

In the VxWorks RTOS, the SEM_INVERSION_SAFE option is provided and can be enabled as follows:

```
semMUTEX = semMCreate(SEM_Q_PRIORITY | SEM_
INVERSION_SAFE);
```

Another feature is a user-specified queuing discipline for the semaphore's queue, which, in general, follows a first-in-first-out (FIFO) order.

In the VxWorks RTOS, there are two possible queuing disciplines: priority (SEM_Q_PRIORITY) and FIFO

(SEM_Q_FIFO). We choose a queuing discipline when we create a semaphore as follows:

```
semA = semBCreate(SEM_Q_PRIORITY | SEM_EMPTY);
semB = semBCreate(SEM_Q_FIFO I SEM_EMPTY);
```

This queuing discipline selection is not available for POSIX-compatible semaphores.

Another feature is wait-timeout, which specifies how long a task will wait in a semaphore's queue. A timeout value of 0 indicates that the task does not wait at all. A bounded positive timeout value X indicates that the task will wait X time units before the wait operation fails. An infinite time value means that the task will wait indefinitely if needed; this value is the default value in the general definition of the wait operation.

In the VxWorks RTOS, these three timeout values are represented, respectively, by NO_WAIT (0), a positive value, and WAIT_FOREVER (-1):

```
semTake (newSem, NO_WAIT);
semTake (newSem, 100);
semTake (newSem, WAIT_FOREVER);
```

This timeout option is not available for POSIX-compatible semaphores. Other real-time extensions to semaphores include task-deletion safety and ownership of mutual exclusion semaphores.

PROCESS/TASK SCHEDULING

Scheduling a set of computer processes or tasks is to determine when to execute which task, thus determining the execution order of these tasks, and, in the case of a multiprocessor or distributed system (6), to also determine an assignment of these tasks to specific processors. This task assignment is analogous to assigning tasks to a specific person in a team of people. Scheduling is a central activity of a computer system, usually performed by the OS. Scheduling is also necessary is many non-computer systems such as assembly lines.

In non-real-time systems, the typical goal of scheduling is to maximize average throughput (number of tasks completed per unit time) and/or to minimize average waiting time of the tasks. In the case of real-time scheduling, the goal is to meet the deadline of every task by ensuring that each task can complete execution by its specified deadline. This deadline is derived from environmental constraints imposed by the application.

Schedulability analysis is to determine whether a specific set of tasks or a set of tasks satisfying certain constraints can be successfully scheduled (completing execution of every task by its specified deadline) using a specific scheduler.

Schedulability Test. A *schedulability test* is used to validate that a given application can satisfy its specified deadlines when scheduled according to a specific scheduling algorithm.

This schedulability test is often done before the tasks' runtime, that is, before the computer system and its tasks start their execution. If the test can be performed efficiently, then it can be done at run-time as an online test.

Schedulable Utilization. A *schedulable utilization* is the maximum utilization allowed for a set of tasks that will guarantee a feasible scheduling of this task set.

A hard real-time system requires that every task completes its execution by its specified deadline, and that failure to do so, even for a single task, may lead to catastrophic consequences. A soft real-time system allows some tasks or task instances to miss their deadlines, but a task that misses a deadline may be less useful or valuable to the system.

There are basically two types of schedulers: static and run-time (online or dynamic).

Optimal Scheduler. An *optimal scheduler* is one that may fail to meet a deadline of a task only if no other scheduler can.

Note that "optimal" in real-time scheduling does not necessarily mean "fastest average response time" or "shortest average waiting time." A task T_i is characterized by the following parameters:

S: start, release, ready, or arrival time
c: (maximum) computation time
d: relative deadline (deadline relative to the task's start time)
D: absolute deadline (wall clock time deadline)

Non-Real-Time Schedulers

First-in-First-Out. Processes in the ready queue are scheduled in the order they arrive. This scheduler is simple and fair, but the average waiting time may be long.

Shortest-Process-First. The shortest process in terms of computation time (CPU burst) is scheduled first. There are two variations: preemptive and non-preemptive. Starvation is a possibility in this scheduling strategy.

Round-Robin. Each process in the ready queue is scheduled FCFS for a time slice called the quantum. This scheduler is fair and reduces the average waiting time, but we have to ensure that the context-switch time is much less than the quantum.

Priority. Processes in the ready queue are scheduled according to their priorities (which may be fixed or dynamic).

Real-Time Scheduling and Schedulability Analysis

Scheduling of real-time tasks depends on the type(s) of tasks in the applications. Although non-realtime tasks are usually single-instance, there are two other common types of real-time tasks. A single-instance task executes only once. A periodic task has many iterations, and there is a fixed period between two consecutive executions of the same task. For example, a periodic task may perform signal processing of a radar scan once every 2 seconds, so the period of this task is 2 seconds. A sporadic task has zero or more instances, and there is a minimum separation between two consecutive releases of the same task. For example, a sporadic task may perform emergency maneuver of an airplane when the emergency button is pressed, but there is a minimum separation of 20 seconds between two emergency requests. An aperiodic task is a sporadic task with either a soft deadline or no deadline. Therefore, if the task has more than one instance (sometimes called a job), we also have the parameters p period (for periodic tasks) and minimum separation (for sporadic tasks)

The following are additional constraints that may complicate scheduling of tasks with deadlines:

(1) Resources shared by tasks.

(2) Precedence relations among tasks and subtasks.

(3) Frequency of tasks requesting service periodically.

(4) Whether task preemption is allowed.

If tasks are preemptable, we assume that a task can be interrupted only at discrete (integer) time instants unless we indicate otherwise. VxWorks uses preemptive priority scheduling of tasks, allowing the preemption of a running task if a higher-priority task arrives. The priority of a task can be based on its specified deadline or other attributes. For tasks having the same priority, VxWorks uses the round-robin scheduling algorithm to allow the CPU to be shared fairly. Preemption locks are offered to prevent task preemption, but they do not lock out interrupt handling.

Determining Computation Time

The application and the environment in which the application is embedded are main factors determining the start time, deadline, and period of a task. The computation (or execution) times of a task is dependent on its source code, object code, execution architecture, memory management policies, and actual number of I/Os.

For real-time scheduling purposes, we use the worst-case execution (or computation) time (WCET) as c. This time is not simply an upper bound on the execution of the task code without interruption. This computation time has to include the time the CPU is executing nontask code caused by this task as well as the time an I/O request spends in the disk queue.

Determining the computation time of a process is crucial to successfully scheduling it in a realtime system. An overly pessimistic estimate of the computation time would result in wasted CPU cycles, whereas an under-approximation would result in missed deadlines.

Uniprocessor Scheduling

We introduce scheduling in real-time systems by studying the problem of scheduling tasks on a uniprocessor system. Here, we describe schedulers for preemptable and independent tasks with no precedence or resource-sharing constraints. More details on real-time scheduling with resource and synchronization constraints can be found in Ref. 7.

To simplify our discussion of the basic schedulers, we assume that the tasks to be scheduled are preemptable and independent. A preemptable task can be interrupted at any time during its execution and resumed later. We also assume that there is no context-switching time. In practice, we can include an upper bound on the context-switching time (8) in the computation time of the task. An independent task can be scheduled for execution as soon as it becomes ready or released. It does need to wait for other tasks to finish first or to wait for shared resources. We also assume here that the execution of the scheduler does not require the processor, that is, the scheduler runs on another specialized processor. If there is no specialized scheduling processor, then the execution time of the scheduler must also be included in the total execution time of the task set. Later, after understanding the basic scheduling strategies, we will extend these techniques to handle tasks with more realistic constraints.

Fixed-Priority Schedulers: Rate-Monotonic and Deadline-Monotonic Algorithms. A popular real-time scheduling algorithm is the rate-monotonic (RMS or RM) scheduler, which is a fixed(static)-priority scheduler using the task's (fixed) period as the task's priority. RMS executes at any time instant the instance of the ready task with the shortest period first. If two or more tasks have the same period, then RMS randomly selects one for execution next.

Example. Consider three periodic tasks with the following arrival times (S), computation times (c), and periods (p, which are equal to their respective relative deadlines, d):

$$J_1 : S_1 = 0, \quad c_1 = 2, \quad p_1 = d_1 = 5.$$

$$J_2 : S_2 = 1, \quad c_2 = 1, \quad p_2 = d_2 = 4.$$

$$J_3 : S_3 = 2, \quad c_3 = 2, \quad p_3 = d_3 = 20.$$

The RM scheduler produces a feasible schedule as follows. At time 0, J_1 is the only ready task, so it is scheduled to run. At time 1, J_2 arrives. As $p_2 < p_1$, J_2 has a higher priority, so J_1 is preempted and J_2 starts to execute. At time 2, J_2 finishes execution and J_3 arrives. As $p_3 > p_1$, J_1 now has a higher priority, so it resumes execution. At time 3, J_1 finishes execution. At this time, J_3 is the only ready task, so it starts to run. At time 4, J_1 is still the only task, so it continues to run and finishes execution at time 5. At this time, the second iterations of J_1 and J_2 are ready. As $p_2 < p_1$, J_2 has a higher priority, so J_2 starts to execute. At time 6, the second iterations of J_2 finishes execution. At this time, the second iterations of J_1 is the only ready task, so it starts execution, finishing at time 8. The timing diagram of the RM schedule for this task set is shown in Fig. 1.

The RM scheduling algorithm is not optimal in general because there exist schedulable task sets that are not RM-schedulable. For a set of tasks with arbitrary periods, there is a simple schedulability test with a sufficient, but not necessary, condition for scheduling with the RM scheduler (9).

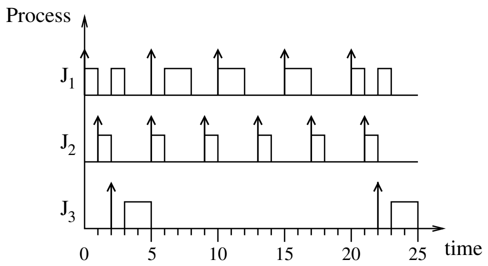


Figure 1. RM schedule.

Schedulability Test 1. Given a set of n independent, preemptable, and periodic tasks on a uniprocessor, let U be the total utilization of this task set. A sufficient condition for feasible scheduling of this task set is $U \leq n(2^{1/n} - 1)$.

However, using this simple schedulability test may underutilize a computer system because a task set whose utilization exceeds the above bound may still be RM-schedulable. There is a sufficient and necessary condition for scheduling using the RM algorithm. Its derivation is omitted here but can be found in Ref. 7.

Schedulability Test 2. Let

$$w_i(t) = \sum_{k=1}^{i} c_k \left\lceil \frac{t}{p_k} \right\rceil, \quad 0 < t \leq p_i$$

where c_k and p_k are, respectively, the computation time and the period of task J_k. The following inequality

$$w_i(t) \leq t$$

holds for any time instant t chosen as follows:

$$t = kp_j,\ j = 1, \ldots, i,\ k = 1, \ldots, \left\lfloor \frac{p_i}{p_j} \right\rfloor$$

if and only if task J_i is RM-schedulable. If $d_i not = p_i$, we replace p_i by $min(d_i, p_i)$ in the above expression.

Another fixed-priority scheduler is the *deadline-monotonic* (DM) scheduling algorithm, which assigns higher priorities to tasks with shorter *relative* deadlines. It is intuitive to see that if every task's period is the same as its deadline, then the RM and DM scheduling algorithms are equivalent. In general, these two algorithms are equivalent if every task's deadline is the product of a constant k and this task's period, that is, $d_i = kp_i$.

Dynamic-Priority Schedulers. An optimal run-time scheduler is the *earliest-deadline-first* (also known as EDF or ED) algorithm, which executes at every instant the ready task with the earliest (closest or nearest) *absolute* deadline first. The absolute deadline of a task is its relative deadline plus its arrival time. If more than one task have the same deadline, EDF randomly selects one for execution next. EDF is a dynamic-priority scheduler because task priorities may change at run-time depending on the nearness of their absolute deadlines. We now describe an example.

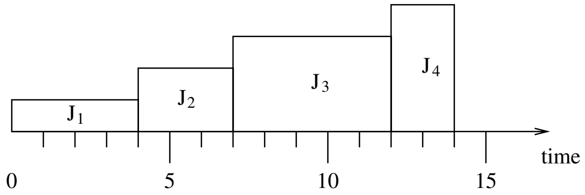


Figure 2. FIFO schedule.

Example. There are four single-instance tasks with the following arrival times, computation times, and absolute deadlines:

$$J_1 : S_1 = 0, \quad c_1 = 4, \quad D_1 = 15$$
$$J_2 : S_2 = 0, \quad c_2 = 3, \quad D_2 = 12$$
$$J_3 : S_3 = 2, \quad c_3 = 5, \quad D_3 = 9$$
$$J_4 : S_4 = 5, \quad c_4 = 2, \quad D_4 = 8$$

A first-in-first-out (FIFO or FCFS) scheduler (often used in non-real-time OSs) gives an infeasible schedule shown in Fig. 2. Tasks are executed in the order they arrive and deadlines are not considered. As a result, task J_3 misses its deadline after time 9, and task J_4 misses its deadline after time 8, before it is even scheduled to run.

However, the EDF scheduler produces a feasible schedule, shown in Fig. 3. At time 0, tasks J_1 and J_2 arrive. As $D_1 > D_2$ (J_2's absolute deadline is earlier than J_1's absolute deadline), J_2 has higher priority and begins to run. At time 2, task J_3 arrives. As $D_3 < D_2$, J_2 is preempted and J_3 begins execution. At time 5, task J_4 arrives. As $D_4 < D_3$, J_3 is preempted and J_4 begins execution.

At time 7, J_4 completes its execution one time unit before its deadline of 8. At this time, $D_3 < D_2 < D_1$, so J_3 has the highest priority and resumes execution. At time 9, J_3 completes its execution, meeting its deadline of 9. At this time, J_2 has the highest priority and resumes execution. At time 10, J_2 completes its execution 2 time units before its deadline of 12. At this time, J_1 is the only remaining task and begins its execution, finishing at time 14, meeting its deadline of 15.

Using the notion of optimality that we have defined in the introduction, the EDF algorithm is optimal for scheduling a set of independent and preemptable tasks on a uniprocessor system.

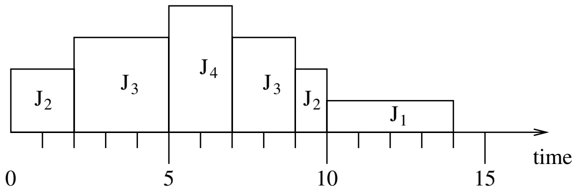


Figure 3. EDF schedule.

Theorem. Given a set S of independent (no resource contention or precedence constraints) and preemptable tasks with arbitrary start times and deadlines on a uniprocessor, the EDF algorithm yields a feasible schedule for S if and only if S has feasible schedules.

Therefore, the EDF algorithm fails to meet a deadline of a task set satisfying the above constraints only if no other scheduler can produce a feasible schedule for this task set. The proof of EDF's optimality is based on the fact that any non-EDF schedule can be transformed into an EDF schedule.

Another optimal run-time scheduler is the *least-laxity-first* (LL or LLF) algorithm (also known as the *minimum-laxity-first* (MLF) algorithm or *least-slack-time-first* (LST) algorithm). Let $c(i)$ denote the remaining computation time of a task at time i. At the arrival time of a task, $c(i)$ is the computation time of this task. Let $d(i)$ denote the deadline of a task relative to the current time i. Then the laxity (or slack) of a task at time i is $d(i) - c(i)$. Thus, the laxity of a task is the maximum time the task can delay execution without missing its deadline in the future. The LL scheduler executes at every instant the ready task with the smallest laxity. If more than one task has the same laxity, LL randomly selects one for execution next.

For a uniprocessor, both earliest-deadline-first (ED) and least-laxity-first (LL) schedulers are optimal for preemptable tasks with no precedence, resource, or mutual exclusion constraints. There is a simple necessary and sufficient condition for scheduling a set of independent, preemptable periodic tasks (9).

Schedulability Test 3. Let C_i denote the computation time of task J_i. For a set of n periodic tasks such that the relative deadline d_i of each task is equal to or greater than its respective period $p_i (d_i \geq p_i)$, a necessary and sufficient condition for feasible scheduling of this task set on a uniprocessor is that the utilization of the tasks is less than or equal to 1:

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1$$

For a task set containing some tasks whose relative deadlines d_i are less than their respective periods, there is no easy schedulability test with a necessary and sufficient condition. However, there is a simple sufficient condition for EDF-scheduling of a set of tasks whose deadlines are equal or shorter than their respective periods.

We next consider the scheduling of sporadic tasks together with periodic tasks.

Sporadic Tasks. Sporadic tasks may be released at any time instant, but there is a *minimum separation* between releases of consecutive instances of the same sporadic task. To schedule preemptable sporadic tasks, we may attempt to develop a new strategy or reuse a strategy we have presented. In the spirit of software reusability, we describe a technique to transform the sporadic tasks into equivalent periodic tasks, which makes it possible to apply the scheduling strategies for periodic tasks introduced earlier.

A simple approach to schedule sporadic tasks is to treat them as periodic tasks with the minimum separation times as their periods. Then we schedule the periodic equivalents of these sporadic tasks using the scheduling algorithm described earlier. Unlike periodic tasks, sporadic tasks are released irregularly or may not be released at all. Therefore, although the scheduler (say the RM algorithm) allocates a time slice to the periodic equivalent of a sporadic task, this sporadic task may not be actually released. The processor remains idle during this time slice if this sporadic task does not request service. When this sporadic task does request service, it immediately runs if its release time is within its corresponding scheduled time slice. Otherwise, it waits for the next scheduled time-slice for running its periodic equivalent.

MEMORY MANAGEMENT

Data and programs are stored in the memory components of a computer system. Most RTOSs do not use virtual memory to ensure that processing time is more deterministic and overhead is substantially reduced. Therefore, the memory address space is not part of a task's context. We review several memory models below, from simple to complex. The simple memory models date back to the early days of computer design. Because of their low management overhead and access time predictability, they are often used in small embedded systems.

Bare Machine

This earliest memory model is simple and flexible. It has no operating systems and provides no services. It is used in small microprocessor and, thus, in many small embedded systems.

Resident Monitor

A resident monitor uses a static fence (an address) to divide (or separate) the memory space into two sections, one used exclusively by the OS (called the resident monitor in the early days of computing) and another assigned to the user's programs and data. The RM resides from memory location 0 up to 1, the address indicated by fence. The user's space is from the address indicated by the fence to maximum address. Note that the actual memory space allocated to the user may be smaller than fence, the maximum address. The first address assigned begins with the address indicated by the fence.

In this memory model, the logical address of a user's program or data space is different from the actual or physical address. To determine the physical address given a logical address, we need to add the logical address to the fence address. Thus, physical address = fence + logical address, or in assembly code, fence(logical address).

```
For user′ s program:
    if physical address < fence
then addressing error
    may cause an interrupt
```

Relocation

Relocation, or dynamic fence, allows more memory allocation flexibility by using a transient area separating the resident monitor and the user's space. This transient area can be used by either the monitor or the user. In this model, the first address of the monitor starts from 0 (as in the above model), but the first address of the user starts from the maximum address. Hence the user's space grows backward.

As above, to determine the physical address given a logical address, we need to add the logical address to the fence address.

Swapping

With the development of lower-cost and larger-size memory components such as disks, OS designers introduce swapping, which allows user's programs and data to be stored in the larger memory component. These programs and data can be swapped into or out of the main memory as needed. For the first time, the entire user's space needs not reside in the main memory during the lifetime of the user's job. To ensure good performance, that is, the processor is working on the user's application programs, we require that the time slice allocated to a user to be much larger than the swap time. In embedded RT systems, swapping can only be used in situations where a task will not be used for some significant period of time.

Paging

Paging is a modern approach (used today) that performs memory management using noncontiguous memory allocations.

Virtual Memory Management

The main idea is that the entire address space for a process needs not reside in the main memory for the process to execute. The early solution is overlaying, which is manual memory management by the user. Overlaying is done by the user's program. For virtual memory management to be successful, there must be program locality, which means that, during any period of time, a program usually references only a small subset of its data and instructions. Another motivation for virtual memory management is the presence of a memory hierarchy, that is, there are at least two memory levels such that the main memory has a high cost and a fast access time and a secondary memory has a low cost and a slow access time. This extra layer of memory mapping/processing and frequent disk I/O requests make the virtual memory model inappropriate for many real-time applications, where response time of the tasks must be bounded. In fact, tasks with hard deadlines are locked in memory so that there are no page faults.

INPUT/OUTPUT

Embedded and real-time systems applications interact with the computer hardware and the external environment much more closely and in a variety of formats, whereas non-real-time applications' I/O are via a standard keyboard/mouse and screen display/printer. For example, in an automobile, inputs to the embedded hardware/software are through the steering wheel, pedals, gear shifter, and an increasing array of electronic switches and buttons. Outputs are sent to the display dials and screens, and result in the activation of antiskid braking mechanisms, steering-ratio changes, and muting of the radio while the phone rings (to name a few of the many output effects).

To ensure portability of the code, most RTOSs provide I/O functions that are source-compatible with I/O in non-real-time OSs such as UNIX and Windows. However, because of the dynamic nature and domain-specificity of real-time applications, RTOSs also offer additional features tailored for embedded systems. For example, VxWorks allows the dynamic installation and removal of device drivers. VxWorks also allows the preemption of device drivers because they execute in the context of the task invoking them, whereas UNIX device drivers cannot be preempted because they execute in system mode. File descriptors or IDs (fds) are unique and specific to each process in UNIX and Windows, but they are globals (except for the standard input (0), output (1), and error (2)) accessible by any task in VxWorks.

As a result of the variety of input and output devices in an embedded real-time system, RTOSs provide far more flexibility for the device driver to handle I/O and to use customized I/O protocol. In non-real-time OSs, user I/O requests are processed first and heavily in the device-independent component of the I/O system before passing them to the device drivers (for the display and keyboard). However, RTOSs allow real-time I/O requests to bypass this standard I/O processing and delegate the control to the device drivers, which makes it possible to use specialized I/O protocols and to ensure satisfaction of requests' deadlines or throughput. In VxWorks, the I/O system in this case would act like a switch routing the I/O requests directly to the specified I/O device drivers.

CONCLUSION

This article has given a brief introduction to real-time/embedded systems, task synchronization, real-time scheduling, memory management, and I/O. The requirement to satisfy hard deadlines in embedded systems means that attention must be given to every task with a hard deadline, which makes it more challenging to develop embedded applications, which necessitate a realtime/embedded OS to ensure that real-time tasks complete by their specified deadlines.

BIBLIOGRAPHY

1. A. Silberschatz et al., *Operating Systems Concepts*, 7th ed., New York: Wiley, 2005.
2. B. O. Gallmeister and C. Lanier, Early experience with POSIX 1003.4 and POSIX 1003.4A, *Proc. IEEE Real-Time Systems Symposium*, 1991, pp. 190–198.
3. B. Gallmeister, *POSIX.4: Programming for the Real World*, 1st ed., January 1995 1-56592-074-0.
4. Available: http://standards.ieee.org/regauth/posix/.
5. Wind River, VxWorks 5.5 Programmer's Guide, 2002.
6. T. Lee and A. M. K. Cheng, Multiprocessor scheduling of hard-real-time periodic tasks with task migration constraints, *Proc.*

IEEE-CS workshop on real-time computing systems and applications, Seoul, Korea, 1994.

7. A. M. K. Cheng, *Real-Time Systems, Scheduling, Analysis, and Verification*, New York: Wiley, 2002.
8. F. Jiang and A. M. K. Cheng, A context switch reduction technique for real-time task synchronization, *Proc. IEEE-CS Intl. Parallel and Distributed Processing Symp.*, San Francisco, CA, 2001.
9. C. L. Liu and J. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, *J. ACM*, **20**(1): 1973, pp. 46–61.

Albert Mo Kim Cheng
University of Houston
Houston, Texas

EMBEDDED SOFTWARE

INTRODUCTION

Electronic devices are commonplace in our lives today. Many products we buy and use contain one or more miniature integrated circuits powered by electricity. Often these integrated circuits contain one or more central processing units (CPUs), with the CPU being the core computational hardware component of a programmable computer.

We usually describe a CPU found in these everyday products as an "embedded processor" and call the computer program that this CPU executes "embedded software." A good starting definition of embedded software is:

Embedded software is software that is ultimately integrated with other electrical and mechanical components and sold to the end-user as a complete product.

This definition is not precise, and there is much room for interpretation. However, by using the term "embedded" usually we are trying to denote something unique or different to distinguish the CPUs and the software found inside our everyday products from the CPUs and software found on our desktop, in the accounting back office, or in the server room. This article explores some issues faced by the developers of embedded software, emphasizing how these issues differ from or are more challenging than the issue faced by developers of desktop or back office software.

EMBEDDED SOFTWARE EXAMPLES

Table 1 lists some common products containing embedded software. The table provides a rough estimate of the software complexity incorporated in these products, expressed as total source-lines-of-code (SLOCs). Even by today's standards of software development, the software complexity of these products is enormous. In today's products, the dominant aspects of a product's functionality are expressed through software.

Economics drives the complexity explosion of embedded software. The microprocessors, microcontrollers, and digital signal processors in today's products permit the baroque expression of product features and functions. This expression is limited physically only by the cost of the memory to store and to execute the code and by the imagination of the product designer. The per-bit cost of memory—in the form of disk drives, flash memories, random-access memories, and read-only memories—drops roughly by a factor of two every two years (1). Today even a $100 product can hold upward of 50M SLOCs. Product creators say: "I can afford to put 50 Mbytes of memory and 200 Mbytes of ROM in my handheld product. I want to fill it up with quality software features that sell!"

Table 2 lists some characteristics often associated with embedded software. No single product will have all of these, but most embedded software will have at least some of these characteristics. Each characteristic can present special challenges to the software developer.

The following sections discuss several of the most difficult issues faced by embedded software developers:

- Software cost and development productivity
- Rapid time-to-market and hardware/software codesign
- Reliability and testing
- Heterogeneous multiprocessor software development
- Real-time systems
- Energy usage and energy management
- Human computer interfaces and human factors
- Security against attack and theft

These issues are not exclusive to embedded software nor do they cover all aspects of computer science that can be applied to the development process. The issues are chosen to illustrate many critical elements of embedded software that are different or more challenging for embedded software than for desktop or for back office applications.

SOFTWARE COST AND DEVELOPMENT PRODUCTIVITY

Software development cost and schedule are critical issues with virtually all software-intensive products. The explosion in the complexity of embedded software makes this especially true for products containing embedded software. Software development is a nonrecurring cost as it is a one-time expense. The cost of manufacturing the product is a recurring cost as it is incurred each time an individual product is made. Many products containing embedded software sell at very low prices, and thus, their recurring costs must be very small. However, the nonrecurring costs of software development must be amortized across the total sales to recover its cost.

A product containing a million lines of code could cost $20–40M to develop from scratch using even the best software engineering practices (2)[1]. The amortized cost of the software across even a million units would be $20–40, with a likely unsupportable percentage of the selling price in a competitive, low-cost market. The nonrecurring cost of software has become a critical cost issue even in very expensive products such as luxury automobiles or commercial airplanes, depending on the total quantity of software involved, the very strict quality requirements placed on the software development process, and the lesser sales volumes compared with less expensive products.

In a competitive environment, software reuse is the most effective tool we have to lower the cost of software development. Reuse effectively amortizes costs across a higher sales volume, lowering the per-unit cost. Producers of

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Table 1. Some Products Containing Embedded Software

Product	SLOC (M)	Comments
Next-generation jumbo jet airliner	1,000	critically reliable, active real-time control, high potential for product liability
2006 luxury sedan	30–50	highly reliable, up to 75 distributed CPUs, cost sensitive, active real-time control
Residential gateway	10–20	very low cost, quick to market
CT medical imager	4–6	highly reliable, potential for product liability
High-end cellular telephone handset	3–10	energy efficient, very low cost, reliable, 3–6 different CPUs, quick to market
Programmable digital hearing aid	.005–.02	10–30M multiply/accumulates per second at 0.001 watt power, extremely low cost, programmable post-manufacture

Table 2. Some Characteristics of Embedded Software

- Low cost
- Small "footprint"
- Short time to market
- High reliability
- "Close" to the hardware
- Codesigned with a system on a chip
- Software/firmware in ROM/PROM
- Low power and power management
- Very high performance on a specialized task
- Heterogeneous, multiple processors
- Software on a special-purpose processor
- Observing and controlling real-world signals
- Real time
- "Nontraditional" or safety-critical user interface
- Security against attack and theft

products containing embedded software use several methods of software reuse:

- Software product line concepts
- Commercial embedded operating systems
- Software/hardware platforms
- Software standards
- Open source software
- Value chains of third-party developers

The software product line (3)[2]—often called a software product family in Europe—attempts to achieve efficiencies similar to those attained with an assembly line in modern manufacturing methods. A product line approach recognizes that most products in a market segment are very similar, varying only in certain well-defined and predictable ways. For example, many different automobile models from a manufacture share several common functions, but they differ in parameters or in options in a specific configuration. The same is true for cellular telephone handsets and television sets. Software reuse becomes easier when the commonality and differences in the software design are exploited.

The software product line approach focuses first on software architecture:

> The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationship among them (4).

Developers find it easier to create reusable components when the architecture takes into account similarities as well as points of variation across the different products in the product family. Within the constraints of the architecture, developers can create reusable software components and other software assets for the initial products, and then they can refactor continuously those components to maintain the product line as new products are created. Companies adopting software product lines have reported case studies showing factors-of-two or better improvements in software development productivity over their previous "serendipitous" reuse strategies. Also, defects were decreased significantly and time-to-market was shortened considerably (5).

The software product line approach also recognizes that domain expertise[3] is an important aspect of embedded software design. Embedded software often interacts with real-world signals or mechanical systems. Domain knowledge and special mathematical skills—digital signal processing, digital communications, real-time control, image processing, or computer graphics, for example—facilitate effective software implementation. A focused development team with the correct mixture of software engineering skills and domain expertise can make the difference between a successful and an unsuccessful product.

Commercial embedded operating systems are a tremendous source of software reuse. In many embedded applications, most lines-of-code are provided by the operating system. Windows CE^{TM}, Symbian OS^{TM}, and Embedded Linux^{TM} are examples of commonly used operating systems. Usually, the embedded operating system provides components and skeletal architectures for the run-time environment, the user interface framework, the peripheral drivers, the media encoders/decoders, and the communication protocols needed by the product. The generality and extended functionality of the operating system allows the operating systems to be used across many of embedded products.

The generality of a commercial embedded operating system also can be a curse. The embedded operating

[1] See entry on SOFTWARE ENGINEERING.

[2] See entry on SOFTWARE ENGINEERING.

[3] See entry on DOMAIN EXPERTISE.

systems must be tailored and configured to eliminate features that are not used, requiring a significant effort. Even then, the resulting executable code size may be too large for low "footprint" applications or too complex for adequate testing. These factors are critical in highly distributed systems like those found in an automobile.

A software/hardware platform is a development environment and framework of software and hardware components designed to provide common features and functions that can be reused across an application domain. Often a platform is an outcome of a software product line, but it also can evolve from legacy products. Usually the platform provides a programmer interface layer above the operating system on which many similar applications can be built. Platforms can be proprietary, commercial, or a mixture of the two.

A cellular handset manufacturer or a television set manufacturer, for example, will develop a proprietary platforms that is then customized specifically for each of the different products in the product line. Commercial embedded application platforms are becoming more common in the industry. Qualcomm's Brew$^{\mathrm{TM}}$ and Nokia's Series 60$^{\mathrm{TM}}$ on Symbian OS are two examples of commercial platforms for development of mobile wireless applications. Platforms provide independent developers with a post-manufacture development opportunity, and they offer similar productivity advantages to those of a software product line.

Software standards are an effective concept used to increase embedded software reuse. Usually, standards specify interfaces between software modules or between hardware and software. However, standards can cover software architecture, run-time environments, security, testing, and software methodology. Standards do not specify the implementation, allowing competition among vendors for creative solutions. Standards can be industry-specific or application-specific, developed through cooperation directly between otherwise competing companies. Once the standard is worked out, it may be held and maintained by a vendor-neutral standards body (6) or by a consortium of companies. The Institute of Electrical and Electronics Engineers, International Organization for Standardization, International Telecommunications Union, and the World Wide Web Consortium are a few examples of standards bodies with significant impact on embedded software reuse. Sometimes standards are established informally as "de facto" standards when everyone merely follows the industry leader's interface practices.

Open source software (7) is another form of software reuse used in embedded systems. Under the open source license, software is made available in source form, allowing the product developer to benefit from features and bug fixes added by other developers. Sometimes the developers may be creating similar products, and sometimes not. Embedded Linux is a very successful example of software reuse via an open source license. Open source software is not necessarily free for commercial use nor is it public domain software. Usually, licensing fees and legal restrictions apply for use of the intellectual property contained in the software.

Third-party developers contribute to software reuse. Software development for complex products rarely is performed completely by the product developer alone. For example, semiconductor vendors will license significant software content, software tools, and example code to the purchaser of their programmable components as a way of winning business. For the most competitive programmable semiconductor products, semiconductor vendors may license production quality software components that can be dropped directly into an embedded software product. Similarly other companies—usually called "third-party developers"—spring up to provide specialized domain expertise, software integration skills, and licensed software for specialized processors. Third-party developers often provide complete hardware/software subassemblies containing significant embedded software. A diesel engine for an automobile or a jet engine for an aircraft would be examples of these subassemblies. Because third-party developers sell their software to multiple customers for multiple products, effectively they are promoting software reuse.

Embedded operating systems, third-party software, and open source software are all examples of a "value chain (1)" (sometimes called "value web") that fosters software reuse and allows embedded software products with millions (or even billions) of lines of code to be created so that the very high nonrecurring cost of its development is amortized effectively across a very large number of products.

RAPID TIME-TO-MARKET AND HARDWARE/SOFTWARE CODESIGN

The old cliché "time is money" is certainly true when it comes to product introduction. Time-to-market is a critical aspect of embedded software development for many products. Sales of a new consumer product—a digital still camera or a music player, for example—peak just before Christmas. The difference of a few weeks in the critical development schedule can make the difference between financial success and failure in the marketplace.

Embedded software development can be challenging especially in this environment. Software development costs go up when development schedules are shortened artificially. The developer may need a software process that consciously trades short development time for programmer efficiency to maintain a tight schedule[4].

Hardware/software codesign methodology often is employed to gain rapid time-to-market for products containing embedded software that is "close to the hardware" and when one or more integrated circuits are yet-to-be-developed. The software developer cannot wait for the hardware to start development of the software. Hardware/software codesign methods[5] must be used so that the software and hardware developments can proceed in parallel. Hardware/software codesign is a methodology for simultaneous development of new hardware, new software,

[4] See entry on SOFTWARE DEVELOPMENT METHODOLOGIES AND PROCESSES.

[5] See entry on HARDWARE/SOFTWARE CODESIGN.

and new development tools. The complex interactions among the application domain, the various hardware and software components, and the development tools must be simulated or modeled at varying levels of abstraction early in and throughout the design process. Embedded software allows the inevitable changes in requirements or minor hardware fixes to be implemented quickly and late in the development cycle. Consequently, software is frequently a preferred design choice for quick time-to-market products even when a more hardware-centric approach would have lower recurring costs.

RELIABILITY AND TESTING

Many products containing embedded software have high reliability requirements. We expect our telephones to be more reliable than our desktop computers. We expect our automobiles to be more reliable than our telephones, and we expect our airplanes to be more reliable than our automobiles. Reliability is a very key component of product liability costs (8), warranty costs, software maintenance costs, and ultimately product success.

We can achieve adequate reliability through application of good software engineering practices: software architecture, design for reliability, a quality software development process[6] and extensive test coverage[7]. However, no system is 100% reliable. Several aspects of embedded systems make achieving the desired level of reliability very difficult.

Adequate test coverage is difficult to achieve for software that senses and controls real-world signals and devices. We would like to test such software against all combinations and permutations of its environment, but this is difficult because of the real-world temporal variation in inputs and external state. If the software is very complex, it is even worse because the huge combinitorics of internal state compounds the problem. The product test cycle for telecommunications products can be 9–12 months over thousands of sample products in various configurations and environments. For a commercial aircraft, the software testing process can take years.

For higher reliability systems, reliability techniques such as redundancy and voting, error-checking and recovery, formal reliability models, formal software validation tools, temporal logic models of system behavior, requirement-driven margins of safety, and executable assertions must be used to augment rigorous testing (9).

Real-time embedded software executing on complex integrated circuits is more difficult to test and debug than software with relaxed time constraints. Real-time systems often follow the uncertainty principle: “When you test them, their reliability and performance change.” To achieve adequate testing, inputs must be provided to internal components, and internal state and outputs must be collected as nonintrusively as possible. Historically, this task was assigned to a logic analyzer or to a real-time test harness. However, today’s complex integrated circuits are pin-limited and bandwidth-limited relative to their internal computation rates. It is difficult to achieve high data transfer rates on and off the chip nonintrusively. Modern programmable integrated circuits may employ special test and debug ports—IEEE 1149.1 (Joint Test Action Group) standard, for example—and add special internal nonintrusive trace circuitry, similar to a built-in logic analyzer, to capture internal data. Combined with software design-for-test concepts, this internal circuitry increases real-time test coverage.

No product is without latent defects. Latent defects are a product liability—a future cost for financial compensation for injured parties. Manufacturers warrant their product against latent defects—a future cost for recalling, repairing, or replacing defective products. Product litigation and product recalls are expensive. These future costs depend on the number and the severity of defects in the current product and on the speed and the efficacy in which defects are fixed before they cause a problem. Embedded software defects have become a major cost factor. To predict and manage these costs, the developer can create a latent defect model to help drive pricing and maintenance decisions. Usually such models are based on metrics captured over the lifecycle of the software development process. For embedded software, frequently this means adding extra software and hardware to products to capture operational/test data in the field. Extra software and hardware also may be added to enable or to lower the cost of field upgrades. Latent defect models are statistical in nature and usually are based on historical metrics associated with the software developer’s software process as well as on the specific development and test metrics captured during the specific product’s development cycle[8].

When developers use a latent defect models for pricing and product improvement decisions, they need similar models and data from their third-parties and other sources of reusable software. The lack of models and data can be a barrier to using third-party or open-source software.

HETEROGENEOUS MULTIPROCESSOR DEVELOPMENT

Products with complex embedded software content are often heterogeneous multiprocessor systems. These systems can bring big advantages. Different CPUs or computational accelerators can be specialized and optimized for the specific task demanded of them. Real-time activities can be isolated physically from nonreal-time functions to simplify the analysis and design. Whole devices can be powered down when not used to save power. Mission-critical operations can be isolated physically from less reliable code so as to eliminate the unpredictable side effects of unreliable code. Multiple processors can lower or eliminate data transmission costs, which can be more expensive and time consuming than the computation itself.

However, multiple heterogeneous processor systems come with a development penalty. Programming different CPUs usually requires different programmer training and new design skills. Specialized processors or computational

[6] See entry on RELIABILITY TEST.

[7] See entry on SOFTWARE ENGINEERING PRACTICES.

[8] See entry on DEFECT MODELS IN SOFTWARE PROCESS.

accelerators may have development tool limitations that make them harder to program.

Tool stability and versioning is very important for efficient software development but especially so for embedded software for heterogeneous processors. For example, a subassembly manufacturer in the automotive industry will have developed and tested millions of lines of code that are then reused in hundreds of different vehicles manufactured by many manufacturers. A new version of a compiler may provide improved code performance or may fix compiler defects. But changing to the new compiler would require recompilation, revalidation, and testing of all the existing code used in each of the various products and product environments. This task is daunting and time consuming. Using the old compiler version for lifecycle maintenance on older products is preferred. However, keeping track of all versions of tools for all variations of software is hard. Embedded software developers usually keep all their software tools in the same configuration management system that contains the code they are developing to avoid unnecessary or unanticipated costs and delays caused by new tool versions.

REAL-TIME SYSTEMS

Many products contain real-time (10)[9] embedded software. Real-time software, like any other software, accepts inputs, updates the internal state, and produces outputs. However, the time relationship of the outputs relative to the inputs and the implicit or explicit representation of time in the software are what make software real time. Often real-time software is part of a feedback loop controlling real-world signals and mechanical devices—an aircraft "fly-by-wire" flight control system, for example. But real-time software also is important in products such as portable music or digital video players dealing with audio and video perception. Human perception is sensitive to temporal aspects of sound and vision.

Real time is more about predictable or deterministic computational performance than about fast or high throughput. A unit of computation can have a time deadline relative to some event. When failing to complete the computation before the deadline causes a failure, we call the deadline a "hard" deadline and the system is called a hard real-time system. If the system can miss the deadline occasionally and still meet requirements, we call the deadline a "soft" deadline and the system is called a soft real-time system. A flight control system usually is a hard real-time system, whereas an audio decoder is more likely a soft real-time system. In reality, real time is a continuum between hard and soft based on the allowable statistics and the severity of missed deadlines and the developer must make a corresponding tradeoff between determinism and speed. A flight controller almost always will use deterministic software techniques over faster but less predictable ones, whereas an audio decoder may meet requirements by using high throughput but occasionally using approximate computations or even sometimes allowing noticeable artifacts in the sound.

Designing complex embedded real-time systems is a tough task. It usually helps to consider time explicitly in the design and to develop a model of computation as part of the software architecture. A model of computation (11)—or "framework"—is a set of rules or design patterns that determine the interaction of all the time-critical components of the software. The choice of computational model depends on the domain and on the specifics of the real-time requirements. A good model of computation can lead to lower development cost and higher reliability.

A real-time operating system (RTOS) can provide reusable components and a framework for the chosen model of computation. Some embedded operating systems, such as Windows CE or Symbian OS, provide significant real-time features. Additionally, commercial RTOS vendors (12)—Wind River, Green Hills Software, or LynuxWorks, for example—provide robust frameworks for highly reliable, hard real-time embedded systems.

ENERGY USAGE AND ENERGY MANAGEMENT

Many products containing embedded software are battery powered. Customers prefer infrequent battery charging or replacement, which in turn means efficient use of energy (13). Usually, embedded software is involved in the energy management of energy-efficient products. Energy usage and energy management are key elements to the software design.

System designers use many different techniques for energy management. Some examples are as follows:

- Special CPUs or other processors
- Clock and power control of circuits
- Parallel computation
- Voltage scaling

Special processors—programmable digital signal processors or programmable digital filter banks, for example—can improve greatly the energy efficiency over a conventional CPU. Often these devices require custom programming and can exacerbate the issues with the heterogeneous multiprocessor nature of the software. However, the benefits of more energy efficiency make the challenges worthwhile (14).

The programmable, in-ear digital hearing aid is an excellent example of the marriage of embedded software and a programmable special-purpose signal processor. Although current-day digital hearing aids may contain some embedded software for control, they do most of the signal processing with hard-wired digital filters implemented directly in logic. They do not employ the superior software signal processing techniques demonstrated in the research laboratories because the power consumption would use up the battery in hours or even in minutes. An embedded digital signal processor, augmented with programmable digital filters or other specialized programmable processors, can provide superior sound quality and can adapt better to the hearing impairment. Because it is programmable, the same basic design can be adapted to a

[9] See entry on REAL-TIME SOFTWARE.

wider range of hearing disabilities and may even be reprogrammed in the audiologist's office. Ultimately this energy-efficient embedded software product will benefit over 600M hearing-impaired people worldwide.

Digital integrated circuits use one or more clock signals to synchronize digital logic operations. Every time the clock toggles, it consumes energy as it charges or discharges the electrical capacitance of the on-chip interconnect wires it drives. Usually the clock toggles at twice the frequency of the rest of the logic and consequently is one of the largest consumers of energy in an integrated circuit. When the clock toggles at its full rate, it is consuming energy even when the digital logic circuits it is synchronizing are not performing any useful work. Thus energy efficient integrated circuits control the clock rates for the various internal circuits and subsystems in an on-demand manner. This clock management function usually is performed by the embedded software.

Today's fastest and most dense integrated circuits contain exceedingly small transistors with minimal geometries under 65 nm. The very low voltages and very high clock rates enabled through these small transistors have a detrimental side effect on energy usage. When powered, these small transistors leak current in a manner analogous to a leaky faucet. The current lost to a single transistor is small, but the current lost in a large circuit of 500M transistors can be huge. When circuits are not performing useful work, power must be switched off to conserve the energy that would otherwise be lost. This power switching may also be part of the embedded software function, adding yet another layer of complexity. But more importantly, most of the circuits that are powered down contain registers or memory to hold internal state information. These data must be made available again to the software and other logic functions when they are reactivated. Critical state information that could be lost must be preserved in special memories or with special memory power-down configurations, or they must be recreated again and reinitialized when the circuit is powered up. This function also can be assigned to the embedded software. Power management is now a complex feature of energy-efficient integrated circuits requiring embedded software for correct operation.

Parallel computation can be used to lower energy consumption. The rate at which energy is consumed in a CMOS digital integrated circuit is directly proportional to the clock rate, whereas the time it takes the software to perform its task is inversely proportional to the clock rate. Total energy consumed for a fixed unit of software functionality remains constant over a wide range of clock rates. However, if you can lower the integrated circuit voltage, the rate of energy consumption drops as the square of the voltage, whereas the maximum achievable clock rate drops only roughly proportionally to the voltage. Operating the integrated circuit at its lowest operating voltage saves energy, albeit at a reduced clock rate. Two CPUs operating in parallel at a slow clock rate are roughly twice as energy efficient as a single CPU operating at twice the clock rate, assuming that the parallel computation still can achieve the same computational efficiency. Parallel computation is not always easy or achievable, but it can conserve energy when used effectively in an embedded system. Programming parallel processes is a difficult aspect of energy-efficient, embedded software design.

Voltage scaling is a similar concept. Voltage scaling recognizes that in many embedded systems the computational load is not uniform over time and may not even be predictable. Voltage scaling allows the software to select its own clock rate and select the required operating voltage, computational speed, and resultant energy consumption rate. When properly scheduled, the software can complete the current computational load "just in time," and, thus, achieve the best energy efficiency. For soft and hard real-time systems, voltage scaling can save energy, but it adds yet an additional layer of complexity to the software design. In dynamic applications, effective use of voltage scaling requires extra software components to predict future computational loads in advance.

HUMAN–COMPUTER INTERFACES AND HUMAN FACTORS

Frequently, the software embedded in a product interacts directly with a user. Thus, the product is an extension of the user in performing his task. The design of the software behind this interface is critical in the success or failure of the product.

Products with physically limited input and output capabilities can be difficult to use, and superior usability is a major factor of product success. For example, sales of a digital video recorder can improve when a more user-friendly interface is implemented to capture unattended broadcasts. Some products are meant to be used in eyes-free or hands-free environments or to be accessible by persons with a visual or physical impairment. Cellular telephones and automotive navigation systems, for example, may employ voice recognition and response to augment the traditional user interface. In complex and exacting tasks, such as piloting and aircraft, the user can be overwhelmed with information. A good interface will prioritize automatically and present only the most critical information, while avoiding information overload by suppressing the less important information. In any safety-critical system, such as in aircraft or in automobile electronics, human errors are a real safety concern. The user interface software must avoid confusing protocols, repetitive monotony, and user mental fatigue that can lead to human errors or lapses in judgment. The user interface must check and confirm potentially disastrous commands while maintaining the responsiveness the user needs to perform in an emergency under stress.

Attention to human factors is a key element of the design and testing process for embedded software that interacts directly with humans. User-centered design (15), sometimes called human-centered design, is one design process that attempts to inject the user's wants, needs, and variability into the software development and maintenance lifecycle. User-centered design recognizes that the user's behavior must be viewed in the context of the full range of the product's use scenarios. Users and potential users are involved continuously throughout the design cycle via user focus groups, assessments of usage scenarios task analyses and interface mock-ups or sketches, and testing with working prototypes and preproduction software. User-centered design ensures that the user is well represented in the design process, but it does not diminish the other aspects of good software design and software reuse processes.

User-centered design is not a panacea for interacting with the user. User interfaces for safety-critical embedded software are particularly demanding. Catastrophic errors usually are rare and occur as a result of the simultaneous occurrence of two or more even rarer events. The statistical margins of variations of human characteristics, user tasks, environmental conditions, and software defects are all difficult to predict, to observe, and to characterize through direct interaction with users. User-centered design must be combined with a strong safety methodology, such as the International Electrotechnical Commission's "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems" (IEC 61508) or the joint Radio Technical Commission for Aeronautics and European Organization for Civil Aviation Equipment DO-178B Level A software development standard.

SECURITY AGAINST ATTACK AND THEFT

We are all familiar with the issues of computer viruses and electronic theft in today's desktop and World Wide Web environments (16). Although not yet commonplace, embedded software products also are susceptible to these security issues. Because so much of our society depends on embedded software, terrorism is another threat. The threats are evolving. In the past, embedded software was constrained to ROM, was not connected electronically to a network, and was never upgraded. Tampering was difficult and ineffectual, so the threat was minimal. This situation is no longer true. Cellular handset developers, automotive companies, aircraft producers, media player developers, as well as most other industry segments are now taking digital security seriously. Cellular handset manufacturers are working with semiconductor vendors to put security features in hardware to thwart cellular handset cloning and viruses.

Content providers are concerned with theft of their products. In the near future, digital rights management (17) will be included in the embedded software of virtually all audio and video players and recorders. Digital rights management is a set of security features that allow a rightful owner or licenser of digital content to use it, but it keeps anyone from copying and distributing the content. Digital rights management usually involves some sort of encryption of the digital media combined with a mechanism to bind the use of the encrypted media to a specific hardware device or player. It also can include information unobtrusively embedded in the media—often called a "watermark"—to identify uniquely the source and distribution path of the media in such a way that an illegal copy and the illegal copier can be identified and prosecuted.

Security and protection against theft are becoming every bit as important in products with embedded software are they are in desktop software products. Security and digital rights management are primarily implemented with software and are becoming yet another critical software development issue with embedded software.

SUMMARY

Embedded software is commonplace. It is a defining constituent of the many products we use daily. More and more, products depend on electronics and software to implement the many new functions we demand. As a result, the complexity of embedded software is exploding.

Development of complex embedded software is a nonrecurring cost that must be amortized across sales of all products that use the software. Because of the high cost of developing software containing millions of lines of code, software reuse, in all its forms, is the only practical way to minimize this cost to the consumer.

The embedded software developer faces many special challenges. Among these challenges are quick time-to-market with hardware/software codesign, high-quality designs with high reliability, special design-for-test features enabling high test coverage, scalable modular designs incorporating many different CPUs and instruction sets, software architectures and computational models that address real-time applications, designs that support energy-efficient use of the underlying electronics, user-centric interfaces, and protection from the risks of computer hacking, terrorism, theft, and litigation.

BIBLIOGRAPHY

1. D. G. Messerschmitt, C. Szyperski, *Software Ecosystem: Understanding an Indispensable Technology and Industry*, Cambridge MA: The MIT Press, 2003.
2. S. McConnell, *Software Estimation: Demystifying the Black Art*, Redmond, WA: Microsoft Press, 2006.
3. J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach*, London: Addison-Wesley, 2000.
4. L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston, MA: Addison-Wesley, 2003.
5. Software Engineering Institute (SEI), Software Product Lines (2006), Pittsburgh, PA: Carnegie Mellon University. Available: http://www.sei.cmu.edu/productlines/.
6. Wikipedia Foundation, Inc. Standards Organization (2006). Available: http://en.wikipedia.org/wiki/Standards_organization.
7. Open Source Initiative OSI. Available: http://www.opensource.org.
8. J. R. Hunziker and T. O. Jones, *Product Liability and Innovation: Managing Risk in an Uncertain Environment*, Washington, D.C.: National Academy, 1994.
9. D. Peled, *Software Reliability Methods*, New York: Springer, 2001.
10. H. Gomaa, *Software Design Methods for Concurrent and Real-Time Systems*, Reading, MA: Addison-Wesley, 1993.
11. E. A. Lee, What's ahead for embedded software?, *IEEE Comp. Mag.*, **33**: 18–26, 2000.
12. C. Adams, COTS operating systems: Boarding the Boeing 787, *Avionics Magazine*, April 1, 2005.

13. CMP Media, LLP., DSP Design Line, Low-power signal processing. Available: http://www.dspdesignline.com/showArticle.jhtml?articleID = 187002922. July 2008.

14. T. Glökler and H. Meyr, *Design of Energy-Efficient Application-Specific Instruction Set Processors*, Boston MA: Kluwer Academic, 2004.

15. D. Norman, Human-centered product development, in D. Norman (ed.), *The Invisible Computer*, Cambridge, MA: The MIT Press, 1998.

16. B. Schneier, *Secrets and Lies: Digital Security in a Networked World*, New York: John Wiley, 2000.

17. B. Rosenblatt, B. Trippe, and S. Mooney, *Digital Rights Management: Business and Technology*, New York: M&T Books, 2002.

JOHN LINN
Texas Instruments
Dallas, Texas

FAULT-TOLERANT SOFTWARE

INTRODUCTION

Fault tolerance is the survival attribute of a system or component to continue operating as required despite the manifestation of hardware or software faults (1). Fault-tolerant software is concerned with all the techniques necessary to enable a software system to tolerate software design faults remaining in the system after its development (2). When a fault occurs, fault-tolerant software provides mechanisms to prevent the system failure from occurring (3).

Fault-tolerant software delivers continuous service complying with the relevant specification in the presence of faults typically by employing either single-version software techniques or multiple-version software techniques. We will address four key perspectives for fault-tolerant software: *historical background*, *techniques*, *modeling schemes*, and *applications*.

HISTORICAL BACKGROUND

Most of the fault-tolerant software techniques were introduced and proposed in 1970s. For example, as one of single-version fault-tolerant software techniques, the exception handling approach began to appear in the 1970s, and a wide range of investigations in this approach led to more mature definitions, terminology, and exception mechanisms later on (4). Another technique, checkpointing and recovery, was also commonly employed to enhance software reliability with efficient strategies (5).

In the early 1970s, a research project was conducted at the University of Newcastle (6). The idea of the recovery block (RB) evolved from this project and became one of the methods currently used for safety-critical software. RB is one of three main approaches in so-called *design diversity*, which is also known as multi-version fault-tolerant software techniques. N-version programming was introduced in 1977 (7), which involved redundancy of three basic elements in the approach: process, product, and environment (8). N self-checking programming approach was introduced most recently, yet it was based on the concept of self-checking programming that had long been introduced (9).

Since then, many other approaches and techniques have been proposed for fault-tolerant software, and various models and experiments have been employed to investigate various features of these approaches. We will address them in the following part of this article.

Definitions

As fault-tolerant software is capable of providing the expected service despite the presence of software faults (7,10), we first introduce the concepts related to this technique (11).

Failures. A failure occurs when the user perceives that a software program is unable to deliver the expected service (9). The expected service is described by a system specification or a set of user requirements.

Errors. An error is part of the system state, which is liable to lead to a failure. It is an intermediate stage in between faults and failures. An error may propagate (i.e., produce other errors).

Faults. A fault, sometimes called a *bug*, is the identified or hypothesized cause of a software failure. Software faults can be classified as design faults and operational faults according to the phases of creation. Although the same classification can be used in hardware faults, we only interpret them in the sense of software here.

Design Faults. A design fault is a fault occurring in software design and development process. Design faults can be recovered with fault removal approaches by revising the design documentation and the source code.

Operational Faults. An operational fault is a fault occurring in software operation due to timing, race conditions, workload-related stress, and other environmental conditions. Such a fault can be removed by recovery (i.e., rollback to a previously saved state and executed again).

Fault-tolerant software thus attempts to prevent failures by tolerating software errors caused by software faults, particularly design faults. The progression "fault-error-failure" shows their causal relationship in a software lifecycle, as illustrated in Fig. 1. Consequently, there are two major groups of approaches to deal with design faults: (1) fault avoidance (prevention) and fault removal during the software development process, and (2) fault tolerance and fault/failure forecasting after the development process. These terms can be defined as follows:

Fault Avoidance (Prevention). To avoid or prevent the introduction of faults by engaging various design methodologies, techniques, and technologies, including structured programming, object-oriented programming, software reuse, design patterns, and formal methods.

Fault Removal. To detect and eliminate software faults by techniques such as reviews, inspection, testing, verification, and validation.

Fault Tolerance. To provide a service complying with the specification in spite of faults, typically by means of single-version software techniques or multi-version software techniques. Note that, although fault tolerance is a design technique, it handles manifested software faults during software operations. Although software fault-tolerance techniques are proposed to tolerant software errors, they can help to tolerate hardware faults as well.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

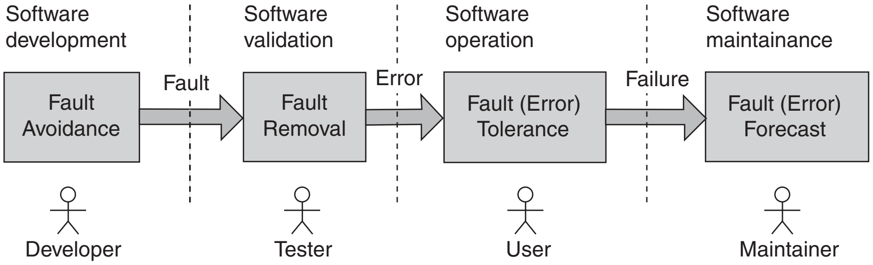


Figure 1. The transition of fault, error, and failure in a software lifecycle.

Fault/failure Forecasting. To estimate the existence of faults and the occurrences and consequences of failures by dependability-enhancing techniques consisting of reliability estimation and reliability prediction.

Rationale

The principle of fault-tolerant software is to deal with residual design faults. For software systems, the major cause of residual design faults can be complexity, difficulty, and incompleteness involved in software design, implementation, and testing phases. The aim of fault-tolerant software, thus, is to prevent software faults from resulting in incorrect operations, including severe situations such as hanging or, at worst, crashing the system. To achieve this purpose, appropriate structuring techniques should be applied for proper error detection and recovery. Nevertheless, fault-tolerance strategies should be simple, coherent, and general in their application to all software systems. Moreover, they should be capable of coping with multiple errors, including the ones detected during the error recovery process itself, which is usually deemed fault-prone due to its complexity and lack of thorough testing.

To satisfy these principles, strategies like checkpointing, exception handling, and data diversity are designed for single-version software, whereas RB, N-version programming (NVP), and N self-checking programming (NSCP) have been proposed for multi-version software. The details of these techniques and their strategies are discussed in the next section.

Practice

From a user's point of view, fault tolerance represents two dimensions: availability and data consistency of the application (12). Generally, there are four layers of fault tolerance. The top layer is composed of general fault-tolerance techniques that are applicable to all applications, including checkpointing, exception handling, RB, NVP, NSCP, and other approaches. Some of the top-level techniques will be addressed in the following section. The second layer consists of application-specific software fault-tolerance techniques and approaches such as reusable component, fault-tolerant library, message logging and recovery, and so on. The next layer involves the techniques deployed on the level of operating and database systems, for example, signal, watchdog, mirroring, fault-tolerant database (FT-DBMS), transaction, and group communications. Finally, the underlying hardware also provides fault-tolerant computing and network communication services for all the upper layers. These are traditional hardware fault-tolerant techniques including duplex, triple modular redundancy (TMR), symmetric multiprocessing (SMP), shared memory, and so on. Summary of these different layers for fault-tolerance techniques and approaches are shown in Fig. 2.

Technologies and architectures have been proposed to provide fault tolerance for some mission-critical applications. These applications include airplane control systems (e.g., Boeing 777 airplane and AIRBUS A320/A330/A340/A380 aircraft) (13–15), aerospace applications (16), nuclear reactors, telecommunications systems and products (12), network systems (17), and other critical software systems.

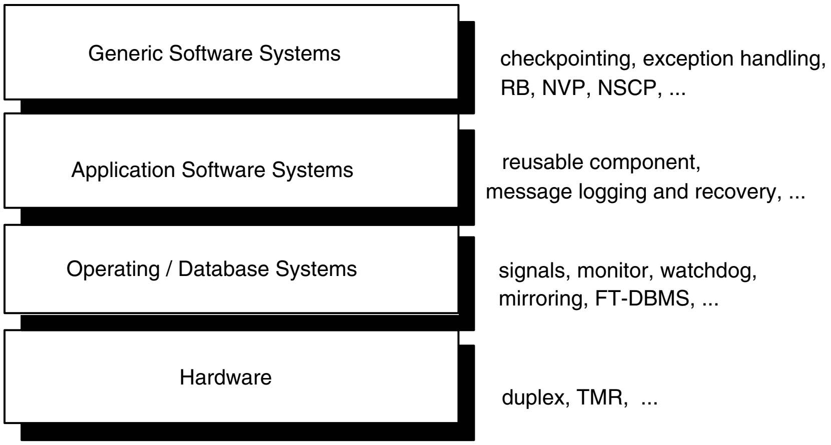

Generic Software Systems
checkpointing, exception handling, RB, NVP, NSCP, ...
Application Software Systems
reusable component, message logging and recovery, ...
Operating / Database Systems
signals, monitor, watchdog, mirroring, FT-DBMS, ...
Hardware
duplex, TMR, ...

Figure 2. Layers of fault tolerance.

FAULT-TOLERANT SOFTWARE TECHNIQUES

We examine two different groups of techniques for fault-tolerant software: single-version and multi-version software techniques (2). *Single-version techniques* involve improving the fault detection and recovery features of a single piece of software on top of fault avoidance and removal techniques. The basic fault-tolerant features include *program modularity*, *system closure*, *atomicity of actions*, *error detection*, *exception handling*, *checkpoint and restart*, *process pairs*, and *data diversity* (2,18).

In more advanced architectures, *design diversity* is employed where *multiple software versions* are developed independently by different program teams using different design methods, yet they provide the equivalent service according to the same requirement specifications. The main techniques of this multiple-version software approach are RB, NVP, NSCP, and other variants based on these three fundamental techniques.

All the fault-tolerant software techniques can be engaged in any artifact of a software system: procedure, process, software program, or the whole system including the operating system. The techniques can also be selectively applied to those components especially prone to faults because of the design complexity.

Single-Version Software Techniques

Single-version fault tolerance is based on temporal and spacial redundancies applied to a single version of software to detect and recover from faults. Single-version fault-tolerant software techniques include a number of approaches. We focus our discussions on two main methods: checkpointing and exception handling.

Checkpointing and Recovery. For single-version software, the technique most often mentioned is the checkpoint and recovery mechanism (19). *Checkpointing* is used in (typically backward) error recovery, by saving the state of a system periodically. When an error is detected, the previous state is recalled and the whole system is restored to that particular state. A *recovery point* is established when the system state is saved and discarded if the process result is acceptable. The basic idea of checkpointing is shown in Fig. 3. It has the advantages of being independent of the damage caused by a fault.

The information saved for each state includes the values of variables in the process, its environment, control information, register values, and so on. Checkpoints are snapshots of the state at various points during the execution.

There are two kinds of checkpointing and recovery schemes: single process systems with a single node and multiple communicating processes on multiple nodes (3). For *single process recovery*, a variety of different strategies is deployed to set the checkpoints. Some strategies use randomly selected points, some maintain a specified time interval between checkpoints, and others set a checkpoint after a certain number of successful transactions have been completed.

For *multiprocess recovery*, there are two approaches: asynchronous and synchronous checkpointing. The difference between the two is that the checkpointing by the various nodes in the system is coordinated in synchronous checkpointing but not coordinated in asynchronous checkpointing. Different protocols for state saving and restoration have been proposed for the two approaches (3).

Exception Handling. Ideal fault-tolerant software systems should recognize interactions of a component with its environment and provide a means of system structuring, making it easy to identify the part of the system needed to cope with each kind of error. They should produce normal and abnormal (i.e., exception) responses within a component and among components' interfaces (20). The structure of exception handling is shown in Fig. 4.

Exception handling, proposed in the 1970s (21), is often considered as a limited approach to fault-tolerant software (22). As departure from specification is likely to occur, exception handling aims at handling abnormal responses by interrupting normal operations during program execution. In fault-tolerant software, exceptions are signaled by the error detection mechanisms as a request for initiation of an appropriate recovery procedure. The design of exception handlers requires consideration of possible events that can trigger the exceptions, prediction of the effects of those events on the system, and selection of appropriate mitigating actions.

A component generally needs to cope with three kinds of exceptional situations: interface exceptions, local exceptions, and failure exceptions. *Interface exceptions* are

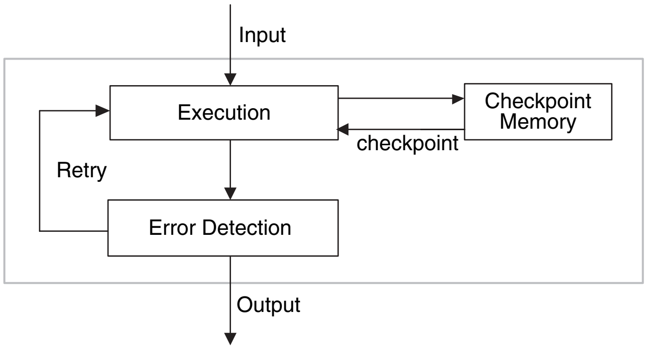


Figure 3. Logic of checkpoint and recovery.

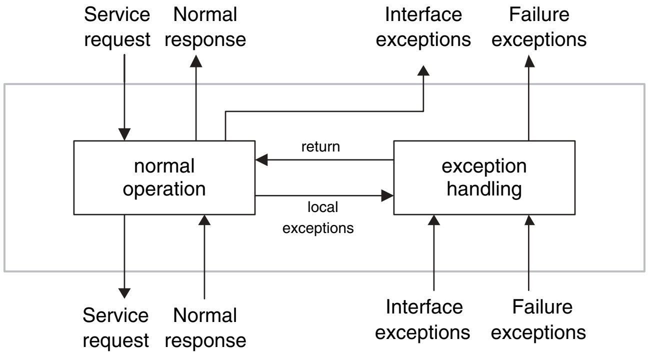


Figure 4. Logic of exception handling.

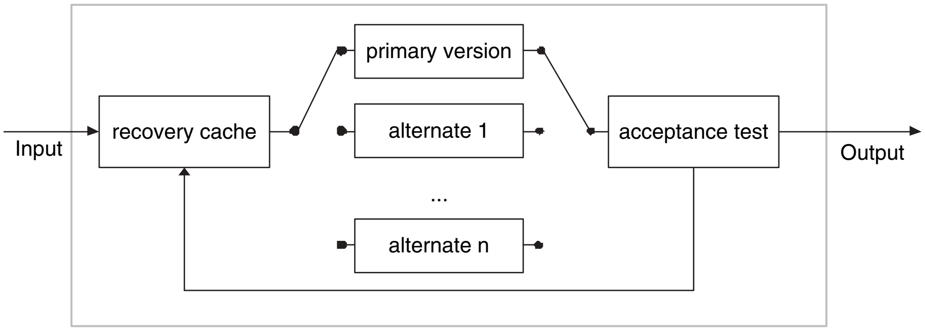


Figure 5. The recovery block (RB) model.

signaled when a component detects an invalid service request. This type of exception is triggered by the self-protection mechanisms of the component and is treated by the component that made the invalid request. *Local exceptions* occur when a component's error detection mechanisms find an error in its own internal operations. The component returns to normal operations after exception handling. *Failure exceptions* are identified by a component after it has detected an error that its fault-processing mechanisms were unable to handle successfully. In effect, failure exceptions notify the component making the service request that it has been unable to provide the requested service.

Multi-Version Software Techniques

The multi-version fault-tolerant software technique is the so-called *design diversity* approach, which involves developing two or more versions of a piece of software according to the same requirement specifications. The rationale for the use of multiple versions is the expectation that components built differently (i.e., different designers, different algorithms, different design tools, and so on) should fail differently (7). Therefore, in the case that one version fails in a particular situation, there is a good chance that at least one of the alternate versions is able to provide an appropriate output.

These multiple versions are executed either in sequence or in parallel, and can be used as alternatives (with separate means of error detection), in pairs (to implement detection by replication checks) or in larger groups (to enable masking through voting). Three fundamental techniques are known as RB, NVP, and NSCP.

Recovery Block. The RB technique involves multiple software versions implemented differently such that an alternative version is engaged after an error is detected in the primary version (6,10). The question of whether there is an error in the software result is determined by an acceptance test (AT). Thus, the RB uses an AT and backward recovery to achieve fault tolerance. As the primary version will be executed successfully most of the time, the most efficient version is often chosen as the primary alternate and the less efficient versions are placed as secondary alternates. Consequently, the resulting rank of the versions reflects, in a way, their diminishing performance.

The usual syntax of the RB is as follows. First of all, the primary alternate is executed; if the output of the primary alternate fails the AT, a backward error recovery is invoked to restore the previous state of the system, then the second alternate will be activated to produce the output; similarly, every time an alternate fails the AT, the previous system state will be restored and a new alternate will be activated. Therefore, the system will report failure only when all the alternates fail the AT, which may happen with a much lower probability than in the single-version situation. The RB model is shown in Fig. 5, while the operation of RB is shown in Fig. 6.

The execution of the multiple versions is usually sequential. If all the alternate versions fail in the AT, the module must raise an exception to inform the rest of the system about its failure.

N-Version Programming. The concept of NVP was first introduced in 1977 (7). It is a multi-version technique in which all the versions are typically executed in parallel and the consensus output is based on the comparison of the outputs of all the versions (2). In the event that the program

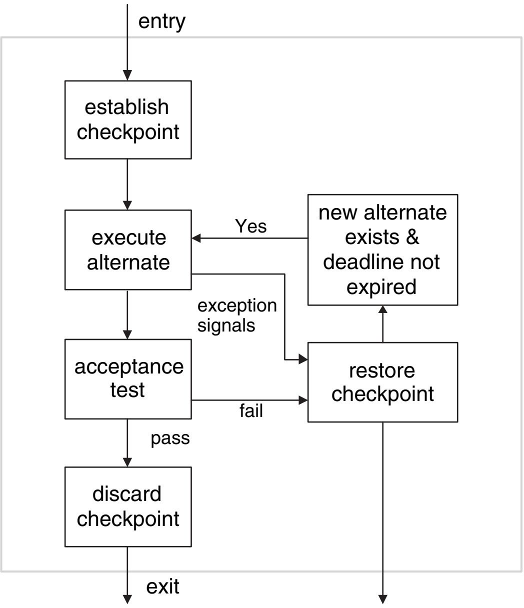


Figure 6. Operation of recovery block.

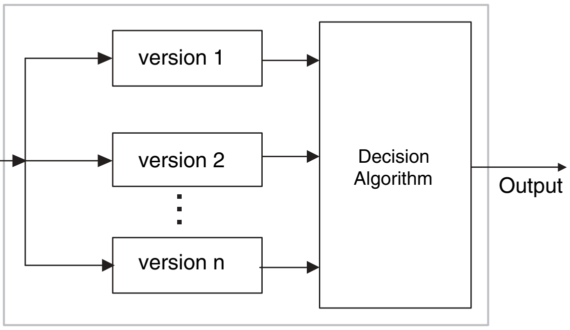


Figure 7. The N-version programming (NVP) model.

versions are executed sequentially due to lack of resources, it may require the use of checkpoints to reload the state before a subsequent version is executed. NVP model is shown in Fig. 7.

The NVP technique uses a decision algorithm (DA) and forward recovery to achieve fault tolerance. The use of a generic decision algorithm (usually a voter) is the fundamental difference of NVP from the RB approach, which requires an application-dependent AT. The complexity of the DA is generally lower than that of the AT. In NVP, because all the versions are built to satisfy the same specification, it requires considerable development effort but the complexity (i.e., development difficulty) is not necessarily much greater than that of building a single version. Much research has been devoted to the development of methodologies that increase the likelihood of achieving effective diversity in the final product (8,23–25).

N-Self Checking Programming. NSCP was developed in 1987 by Laprie et al. (9,26). It involves the use of multiple software versions combined with structural variations of the RB and NVP approaches. Both ATs and DAs can be employed in NSCP to validate the outputs of multiple versions.

The NSCP method employing ATs is shown in Fig. 8. Same as RB and NVP, the versions and the ATs are developed independently but each designed to fulfill the requirements. The main difference of NSCP from the RB approach is in its use of different ATs for different versions. The execution of the versions and tests can be done sequentially or in parallel, but the output is taken from the highest-ranking version that passes its AT. Sequential execution requires a set of checkpoints, and parallel execution requires input and state consistency algorithms.

NSCP engaging DAs for error detection is shown in Fig. 9. Similar to NVP, this model has the advantage of using an application-independent DA to select a correct output. This variation of self-checking programming has the theoretical vulnerability of encountering situations where multiple pairs pass their comparisons but the outputs differ between pairs. That case must be considered and an appropriate decision policy should be selected during the design phase.

Comparison Among RB, NVP, and NSCP. Each design diversity technique, RB, NVP, and NSCP, has its own advantages and disadvantages compared with the others. We compare the features of the three and list them in Table 1.

The differences between AT and DA are: (1) AT is more complex and difficult in implementation, but it can still produce correct output when multiple distinct solutions exist in multiple versions, and (2) DA is more simple, efficient, and liable to produce correct output because it is just a voting mechanism; but it is less able to deal with multiple solutions.

Other Techniques. Besides the three fundamental design diversity approaches listed above, there are some other techniques available, essentially variants of RB, NVP, and NSCP. They include consensus RB, distributed RB, hierarchical NVP, t/(n-1)-variant programming, and others. Here, we introduce some of these techniques briefly.

Distributed Recovery Block. The distributed recovery block (DRB) technique, developed by Kim in 1984 (27), is adopted in distributed or parallel computer systems to realize fault tolerance in both hardware and software. DRB combines RBs and a forward recovery scheme to achieve fault tolerance in real-time applications. The DRB uses a pair of self-checking processing nodes (PSP) together with both the software-implemented internal audit function and the watchdog timer to facilitate real-time hardware fault tolerance. The basic DRB technique consists of a primary node and a shadow node, each cooperating with a RB, and the RBs execute on both nodes concurrently.

Consensus Recovery Block. The consensus RB approach combines NVP and the RB technique to improve software reliability (28). The rationale of consensus RBs is that RB and NVP each may suffer from its specific faults. For example, the RB ATs may be fault-prone, and the DA in

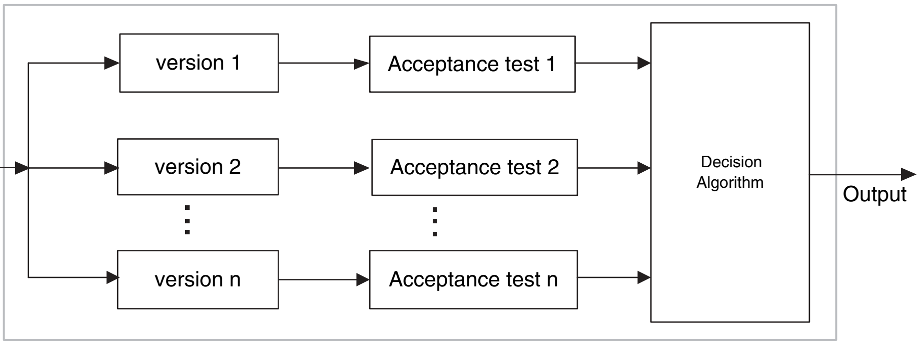


Figure 8. N self-checking programming using acceptance test.

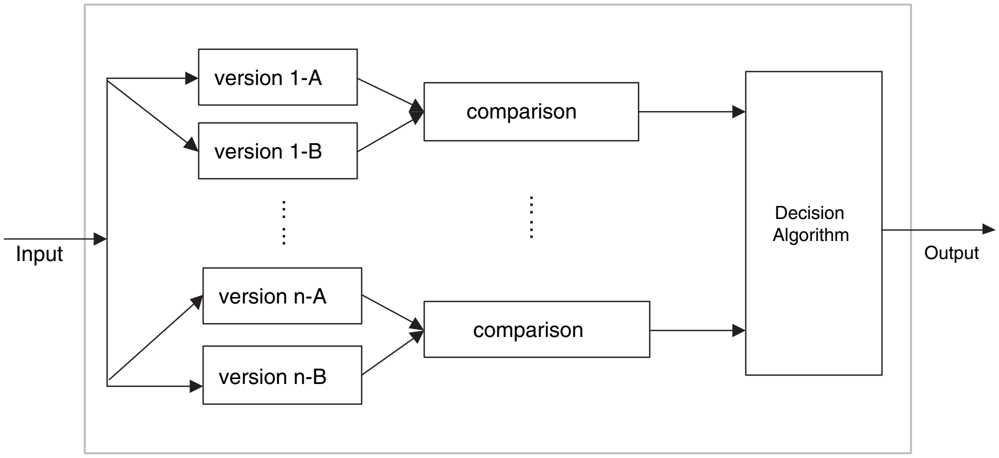


Figure 9. N self-checking programming using decision algorithm.

NVP may not be appropriate in all situations, especially when multiple correct outputs are possible. The consensus RB approach employs a DA as the first-layer decision. If a failure is detected in the first layer, a second layer using ATs is invoked. Obviously, having more levels of checking than either RB or NVP, consensus RB is expected to have an improved reliability.

t/(n-1)-Variant Programming. t/(n-1)-variant programming (VP) was proposed by Xu and Randell in 1997 (29). The main feature of this approach lies in the mechanism engaged in selecting the output among the multiple versions. The design of the selection logic is based on the theory of system-level fault diagnosis. The selection mechanism of t/(n-1)-VP has a complexity of O(n)—less than some other techniques—and it can tolerate correlated faults in multiple versions.

MODELING SCHEMES ON DESIGN DIVERSITY

There have been numerous investigations, analyses, and evaluations of the performance of fault-tolerant software techniques in general and of the reliability of some specific techniques (3). Here we list only the main modeling and analysis schemes that assess the general effectiveness of design diversity.

To evaluate and analyze both the reliability and the safety of various design diversity techniques, different modeling schemes have been proposed to capture design diversity features, describe the characteristics of fault correlation between diverse versions, and predict the reliability of the resulting systems. The following modeling schemes are discussed in chronological order.

Table 1. Comparison of Design Diversity Techniques

Features	Recovery block	N-version programming	N self-checking programming
Minimum no. of versions	2	3	4
Output mechanism	Acceptance Test	Decision Algorithm	Decision Algorithm and Acceptance Test
Execution time	primary version	slowest version	slowest pair
Recovery scheme	backward recovery	forward recovery	forward and backward recovery

Eckhardt and Lee's Model

Eckhardt and Lee (EL Model) (30) proposed the first probability model that attempts to capture the nature of failure dependency in NVP. The EL model is based on the notion of "variation of difficulty" over the user demand space. Different parts of the demand space present different degrees of difficulty, making the program versions built independently more likely to fail with the same "difficult" parts of the target problem. Therefore, failure independency between program versions may not be the necessary result of "independent" development when failure probability is averaged over all demands. For most situations, in fact, positive correlation between version failures may be exhibited for a randomly chosen pair of program versions.

Littlewood and Miller's Model

Littlewood and Miller (31) (LM model) showed that the variation of difficulty could be turned from a disadvantage into a benefit with forced design diversity (32). "Forced" diversity may insist that different teams apply different development methods, different testing schemes, and different tools and languages. With forced diversity, a problem that is more difficult for one team may be easier for another team (and vice versa). The possibility of negative correlation between two versions means that the reliability of a 1-out-of-2 system could be greater than it would be under the assumption of independence. Both EL and LM models are "conceptual" models because they do not support predictions for specific systems and they depend greatly on the notion of *difficulty* defined over the possible demand space.

Dugan and Lyu's Dependability Model

The dependability model proposed by Dugan and Lyu in Ref. 33 provides a reliability and safety model for fault-tolerant hardware and software systems using a combination of fault tree analysis and the Markov modeling process. The reliability/safety model is constructed by three parts: A Markov model details the system structure and two fault trees represent the causes of unacceptable results in the initial configuration and in the reconfigured state. Based on this three-level model, the probability of unrelated and related faults can be estimated according to experimental data.

In a reliability analysis study (33), the experimental data showed that DRB and NVP performed better than NSCP. In the safety analysis, NSCP performed better than DRB and NVP. In general, their comparison depends on the classification of the experimental data.

Tomek and Trivedi's Stochastic Reward Nets Model

Stochastic reward nets (SRNs) are a variant of stochastic Petri nets. SRNs are employed in Ref. 34 to model three types of fault-tolerant software systems: RB, NVP, and NSCP. Each SRN model is incorporated with the complex dependencies associated with the system, such as correlation failures and separate failures, detected faults and undetected faults. A Markov reward model underlies the SRN model. Each SRN is automatically converted into a Markov reward model to obtain the relevant measures. The model has been parameterized by experimental data in order to describe the possibility of correlation faults.

Popov and Strigini's Reliability Bounds Model

Popov and Strigini attempted to bridge the gap between the *conceptual* models and the *structural* models by studying how the conceptual model of failure generation can be applied to a specific set of versions (32). This model estimates the probability of failure on demand given the knowledge of subdomains in a 1-out-of-2 diverse system. Various alternative estimates are investigated for the probability of coincident failures on the whole demand space as well as in subdomains. Upper bounds and likely lower bounds for reliability are obtained by using data from individual diverse versions. The results show the effectiveness of the model in different situations having either positive or negative correlations between version failures.

Experiments and Evaluations

Experiments and evaluations are necessary to determine the effectiveness and performance of different fault-tolerant software techniques and the corresponding modeling schemes. Various projects have been conducted to investigate and evaluate the effectiveness of design diversity, including UCLA Six-Language project (2,35), NASA 4-University project (23,32,36), Knight and Leveson's experiment (24), Lyu–He study (33,37), and so on.

These projects and experiments can be classified into three main categories: (1) evaluations on the effectiveness and cost issues of the final product of diverse systems (7,24,38–42); (2) experiments evaluating the design process of diverse systems (8); and (3) adoption of design diversity into different aspects of software engineering practice (37,43).

To investigate the effectiveness of design diversity, an early experiment (7), consisting of running sets of student programs as 3-version fault-tolerant programs, demonstrated that the NVP scheme worked well with some sets of programs tested, but not others. The negative results were natural because inexperienced programmers cannot be expected to produce highly reliable programs. Another student-based experiment (24) involved 27 program versions developed differently. Test cases were conducted on these program versions in single-and multiple-version configurations. The results showed that NVP could improve reliability; yet correlated faults existed in various versions, adversely affecting design diversity. In another study, Kelly et al. (38) conducted a specification diversity project, using two different specifications with the same requirements. Anderson et al. (39) studied a medium-scale naval command and control computer system developed by professional programmers through the use of the RB. The results showed that 74% of the potential failures could be successfully masked. Another experiment evaluating the effectiveness of design diversity is the Project on Diverse Software (PODS) (40), which consisted of three diverse teams implementing a simple nuclear reactor protection system application. There were two diverse specifications and two programming languages adopted in this project. With good quality control and experienced programmers, high-quality programs and fault-tolerant software systems were achieved.

For the evaluation of the cost of design diversity, Hatton (41) collected evidence to indicate that diverse fault-tolerant software techniques are more reliable than producing one good version, and more cost effective in the long run. Kanoun (42) analyzed work hours spent on variant design in a real-world study. The results showed that costs were not doubled by developing a second variant.

In a follow-up to the work of Avizienis and Chen (7), a six-language NVP project was conducted using a proposed *N-version Software Design Paradigm*(44). The NVP paradigm was composed of two categories of activities: standard software development procedures and concurrent implementation of fault-tolerance techniques. The results verified the effectiveness of the design paradigm in improving the reliability of the final fault-tolerant software system.

To model the fault correlation and measure the reliability of fault-tolerant software systems, experiments have been employed to validate different modeling schemes. The NASA 4-University project (36) involved 20 two-person programming teams. The final 20 programs went through a three-phase testing process, namely, a set of 75 test cases for AT, 1100 designed and random test cases for certification test, and over 900,000 test cases for operational test. The same testing data have been widely employed (23,31,32) to validate the effectiveness of different modeling schemes.

The Lyu–He study (37) was derived from an experimental implementation involving 15 student teams guided by the evolving NVP design paradigm in Ref. 8. Moreover, a comparison was made between the NASA 4-University project, the Knight–Leveson experiment, the Six-Language project, and the Lyu–He experiment in order to further investigate and discuss the effectiveness of design diversity in improving software reliability. The results were further used in Ref. 33 to evaluate the prediction accuracy of Dugan and Lyu's Model. Lyu et al. (43) reported a multi-version project on The Redundant Strapped-Down Inertial Measurement Unit (RSDIMU), the same specification employed in the NASA 4-University project. The experiment developed 34 program versions, from which 21 versions were selected to create mutants. Following a systematic rule for the mutant creation process, 426 mutants, each containing a real program fault identified during the testing phase, were generated for testing and evaluation. The testing results were subsequently engaged to investigate the probability of related and unrelated faults using the PS and DL models.

Current results indicate that, for design diversity techniques, NSCP is the best candidate to produce a safe result, whereas DRB and NVP tend to achieve better reliability than NSCP, although the difference is not significant.

APPLICATIONS

There are many application-level methodologies for fault-tolerant software techniques. As we have indicated, the applications include airplane control systems (e.g., Boeing 777 airplane (14) and AIRBUS A320/A330/A340/A380 aircraft (15,45)), aerospace applications (16), nuclear reactors, telecommunications products (12), network systems (17), and other critical software systems such as wireless network, grid-computing, and so on. Most of the applications adopt single-version software techniques for fault tolerance (i.e., reusable component, checkpointing and recovery, and so on). The design diversity approach has only been applied in some mission-critical applications, for example, airplane control systems, aerospace, and nuclear reactor applications. There are also emerging experimental investigations into the adoption of design diversity in practical software systems, such as SQL database servers (46).

We may summarize the fault-tolerant software applications into four categories: (1) reusable component library (e.g., Ref. 12); (2) checkpointing and recovery schemes (e.g., Refs. 19 and 47); (3) entity replication and redundancy (e.g., Refs. 48 and 49); (4) early applications and projects on design diversity (e.g., Refs. 14,45,46). An overview of some of these applications is given below.

Huang and Kintala (12) developed three cost-effective reusable software components (i.e., watchd, libft, and REPL) to achieve fault tolerance in the application level based on availability and data consistency. These components have been applied to a number of telecommunication products.

According to Ref. 19, the new mobile wireless environment poses many challenges for fault-tolerant software due to the dynamics of node mobility and the limited bandwidth. Particular recovery schemes are adopted for the mobile environment. The recovery schemes combine a state saving strategy and a handoff strategy, including two approaches (*No Logging* and *Logging*) for state saving, and three approaches (*Pessimistic, Lazy, and Trickle*) for handoff. Chen and Lyu (47) have proposed a message logging and recovery protocol on top of the CORBA architecture, which employs the storage available at the access bridge to log messages and checkpoints of a mobile host in order to tolerate mobile host disconnection, mobile host crash, and access bridge crash.

Entity replication and modular redundancy are also widely used in application software and middleware. Townend and Xu (48) proposed a fault-tolerant approach based on job replication for Grid computing. This approach combines a replication-based fault-tolerance approach with both dynamic prioritization and dynamic scheduling. Kalbarczyk et al. (49) proposed an adaptive fault-tolerant infrastructure, named Chameleon, which allows different levels of availability requirements in a networked environment, and enables multiple fault-tolerance strategies including dual and TMR application execution modes.

The approach of design diversity, on the other hand, has mostly been applied in safety critical applications. The most famous applications of design diversity are the Boeing 777 airplane (14) and AIRBUS A320/A330/A340/A380 aircraft (15,45). The Boeing 777 primary flight control computer is a triple-triple configuration of three identical channels, each composed of three redundant computation lanes. Software diversity was achieved by using different programming languages targeting different lane processors. In the AIRBUS A320 series flight control computer (45), software systems are designed by independent design teams to reduce common design errors. Forced diversity rules are adopted in software development to ensure software reliability. In an experimental exploration of adopting design diversity in practical software systems, Popov and Strigini (46) implemented diverse off-the-shelf versions of relational database servers including Oracle, Microsoft SQL, and Interbase databases in various ways. The servers are distributed over multiple computers on a local network, on similar or diverse operating systems. The early results support the conjecture that reliability increases with the investment of design diversity.

SUMMARY

Fault-tolerant software enables a system to tolerate software faults remaining in the system after its development. When a fault occurs, fault-tolerant software techniques provide mechanisms within the software system to prevent system failure from occurring.

Fault-tolerant software techniques include single-version software techniques and multiple-version software techniques. There are two main techniques for single-version software fault tolerance: checkpointing and exception handling. Three fundamental techniques are available for multi-version fault-tolerant software: RB, NVP, and NSCP. These approaches are also called design diversity.

Various modeling schemes have been proposed to evaluate the effectiveness of fault-tolerant software. Furthermore, different applications and middleware components have been developed to satisfy performance and reliability demands in various domains employing fault-tolerant software. Fault-tolerant software is generally accepted as a key technique in achieving highly reliable software.

ACKNOWLEDGMENT

This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4205/04E).

BIBLIOGRAPHY

1. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*, Piscatawag, NJ: IEEE Standards, 1990.
2. M. R. Lyu (ed.), *Software Fault Tolerance*. New York: Wiley, 1995.
3. L. L. Pullum, *Software Fault Tolerance Techniques and Implementation*. Boston: Artech House, 2001.
4. F. Cristian, Exception handling and tolerance of software faults, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 81–107.
5. V. F. Nicola, Checkpointing and the modeling of program execution time, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 167–188.
6. B. Randell and J. Xu, The evolution of the recovery block concept, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 1–21.
7. A. Avizienis and L. Chen, On the implementation of N-version programming for software fault tolerance during execution, *Proc. of the Computer Software and Application Conference (COMPSAC77)*, Chicago, Illinois: 1977, pp. 149–155.
8. A. Avizienis, Dependable computing depends on structured fault tolerance, *Proc. of the 1995 6th International Symposium on Software Reliability Engineering*, Toulouse, France, 1995, pp. 158–168.
9. J. C. Laprie, J. Arlat, C. Beounes, and K. Kanoun, Architectural issues in software fault tolerance, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 47–80.
10. B. Randell, System structure for software fault tolerance, *IEEE Trans. Software Eng.*, **1**(2): 220–232, 1975.
11. J. C. Laprie and K. Kanoun, Software reliability and system reliability, in M. R. Lyu (ed.), *Handbook of Software Reliaiblity Engineering*, New York: McGraw-Hills, 1996, pp. 27–69.
12. Y. Huang and C. Kintala, Software fault tolerance in the application layer, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 231–248.
13. R. J. Bleeg, Commercial jet transport fly-by-wire architecture considerations, *AIAA / IEEE 8th Digital Avionics Systems Conference*, October 1988, pp. 399–406.
14. A. D. Hills and N. A. Mirza, Fault tolerant avionics, *AIAA / IEEE 8th Digital Avionics Systems Conference*, October 1988, pp. 407–414.
15. R. Maier, G. Bauer, G. Stoger, and S. Poledna, Time-triggered architecture: a consistent computing platform, *IEEE Micro*, **22**(4): 36–45, 2002.
16. P. G. Neuman, *Computer Related Risks*. Boston: Addison-Wesley, 1995.
17. K. H. Kim, The distributed recovery block scheme, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 189–210.
18. W. Torres-Pomales, Software fault tolerance: a tutorial, NASA Langley Research Center, Hampton, Virginia, Tech. Rep. TM-2000-210616, Oct. 2000.
19. D. K. Pradhan, *Fault Tolerant Computer System Design*. Englewood Cliffs, NJ: Prentice Hall, 1996.
20. P. A. Lee and T. Anderson, *Fault Tolerance: Principles and Practice*. New York: Springer-Verlag, 1990.
21. J. B. Goodenough, Exception handling: issues and a proposed notation, *Commun. ACM*, **18**(12): 683–693, 1975.
22. F. Cristian, Exception handling and software fault tolerance, *Proc. of the 10th International Symposium on Fault-Tolerant Computing (FTCS-10)*, 1980, pp. 97–103.
23. D. E. Eckhardt, A. K. Caglavan, J. C. Knight, L. D. Lee, D. F. McAllister, M. A. Vouk, and J. P. J. Kelly, An experimental evaluation of software redundancy as a strategy for improving reliability, *IEEE Trans. Software Eng.*, **17**(7): 692–702, 1991.
24. J. C. Knight and N. G. Leveson, An experimental evaluation of the assumption of independence in multiversion programming, *IEEE Trans. Software Eng.*, **12**(1): 96–109, 1986.
25. P. G. Bishop, Software fault tolerance by design diversity, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 211–230.
26. J. C. Laprie, J. Arlat, C. Beounes, K. Kanoun, and C. Hourtolle, Hardware and software fault tolerance: definition and analysis of architectural solutions, *Proc. of the 17th International Symposium on Fault-Tolerant Computing (FTCS-17)*, Pittsburgh, PA: 1987, pp. 116–121.
27. K. H. Kim, Distributed execution of recovery blocks: an approach to uniform treatment of hardware and software faults, *Proc. of the 4th International Conference on Distributed Computing Systems*, 1984, pp. 526–532.
28. R. K. Scott, J. W. Gault, and D. F. McAllister, Fault tolerant software reliability modeling, *IEEE Trans. Software Eng.*, **13**(5): 582–592, 1987.
29. J. Xu and B. Randell, Software fault tolerance: t/(n-1)-variant programming, *IEEE Trans. Reliability*, **46**(1): 60–68, 1997.
30. D. E. Eckhardt and L. D. Lee, A theoretical basis for the analysis of multiversion software subject to coincident errors, *IEEE Trans. Software Eng.*, **11**(12): 1511–1517, 1985.
31. B. Littlewood and D. Miller, Conceptual modeling of coincident failures in multiversion software, *IEEE Trans. Software Eng.*, **15**(12): 1596–1614, 1989.
32. P. T. Popov, L. Strigini, J. May, and S. Kuball, Estimating bounds on the reliability of diverse systems, *IEEE Trans. Software Eng.*, **29**(4): 345–359, 2003.
33. J. B. Dugan and M. R. Lyu, Dependability modeling for fault-tolerant software and systems, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 109–138.
34. L. A. Tomek and K. S. Trivedi, Analyses using stochastic reward nets, in M. R. Lyu (ed.), *Software Fault Tolerance*, New York: Wiley, 1995, pp. 139–165.
35. J. Kelly, D. Eckhardt, M. Vouk, D. McAllister, and A. Caglayan, A large scale generation experiment in multi-version software: description and early results, *Proc. of the 18th International Symposium on Fault-Tolerant Computing*, 1988, pp. 9–14.
36. M. A. Vouk, A. Caglayan, D. E. Eckhardt, J. Kelly, J. Knight, D. McAllister, and L. Walker, Analysis of faults detected in a large-scale multi-version software development experiment, *Proc. of the Digital Avionics Systems Conference*, 1990, pp. 378–385.
37. M. R. Lyu and Y. T. He, Improving the N-version programming process through the evolution of a design paradigm, *IEEE Trans. Reliability*, **42**(2): 179–189, 1993.
38. J. P. Kelly and A. Avizienis, A specification-oriented multi-version software experiment, *Proc. of the 13th Annual International Symposium on Fault-Tolerant Computing (FTCS-13)*, Milano, 1983, pp. 120–126.

39. T. Anderson, P. A. Barrett, D. N. Halliwell, and M. R. Moulding, Software fault tolerance: an evaluation, *IEEE Trans. Software Eng.*, **12**(1): 1502–1510, 1985.

40. P. G. Bishop, D. G. Esp, M. Barnes, P. Humphreys, G. Dahll, and J. Lahti, PODS - a project on diverse software, *IEEE Trans. Software Reliability*, **12**(9): 929–940, 1986.

41. L. Hatton, N-version design versus one good version, *IEEE Software*, pp. 71–76, Nov/Dec 1997.

42. K. Kanoun, Real-world design diversity: a case study on cost, *IEEE Software*, pp. 29–33, July/August 2001.

43. M. R. Lyu, Z. Huang, K. S. Sze, and X. Cai, An empirical study on testing and fault tolerance for software reliability engineering, *Proc. of the 14th IEEE International Symposium on Software Reliability Engineering (ISSRE'2003)*, Denver, Colorado, 2003, pp. 119–130.

44. M. R. Lyu, A design paradigm for multi-version software, Ph.D. dissertation, UCLA, Los Angeles, May 1988.

45. P. Traverse, Dependability of digital computers on board airplanes, *Proc. of the 2nd IFIP Working Conference on Dependable Computing for Critical Applications*, Tucson, Arizona, 1991, pp. 133–152.

46. P. Popov and L. Strigini, Diversity with off-the-shelf components: a study with SQL database servers, *Proc. of the International Conference on Dependable Systems and Networks (DSN 2003)*, 2003, pp. B84–B85.

47. X. Chen and M. R. Lyu, Message logging and recovery in wireless corba using access bridge, *Proc. of the 6th International Symposium on Autonomous Decentralized Systems (ISADS2003)*, Pisa, Italy, 2003, pp. 107–114.

48. P. Townend and J. Xu, Fault tolerance within a grid environment, *Proc. of the UK e-Science All Hands Meeting 2003*, Nottingham, UK, 2003, pp. 272–275.

49. Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi, and K. Whisnant, Chameleon: a software infrastructure for adaptive fault tolerance, *IEEE Trans. Parallel Distrib. Sys.*, **10**(6): 560–579, 1999.

MICHAEL R. LYU
XIA CAI
The Chinese University of
Hong Kong
Shatin, Hong Kong

F

FORMAL PROGRAM VERIFICATION

The objective of formal verification is to produce a mathematical proof that a given implementation (or code) is correct; i.e., it behaves as specified. The specifications of behavior must be formal to achieve formal verification (see the article Formal Specification). Formal verification offers the highest level of software quality assurance, and it is critical for ensuring correctness of systems where life, mission, or security might be at stake.

Testing is currently the primary technique used for quality assurance. Most commercial software endures extensive testing until no more serious errors are revealed by testing and the customers choose to accept the reliability of the resulting code. The quality of assurance when based on testing depends on the quality of the test cases. The difficulty lies in the process of choosing "good" test cases. A test case is one element of the domain of possible inputs for the software. In most cases, it is impractical and impossible to apply testing for all elements of the input domain because the domain is vast and, often, infinite. Therefore, the chosen test cases must include a reasonable coverage of all possible inputs. However, even with a wide variety of well-chosen test cases, testing can only reveal errors; it cannot guarantee a lack of errors: On the other hand, verification can provide a guarantee of correct, error-free code; i.e., the code will produce specified outputs for *all* valid inputs, which is the topic of this article.

Formal verification is only concerned with one aspect of software quality assurances–code correctness with respect to specifications. Validation is a complementary aspect of quality assurance that establishes whether the mapping from the customer's requirements to the program specification is appropriate. Validation is a challenging problem because of the difficulty in interpreting the needs of the client and in developing suitable specifications (see the article Verification and Validation). Assuming the behavior of the software has been properly and adequately specified, this article will explain how to verify that code meets that specification.

MOTIVATION FOR VERIFICATION

In 1969, James C. King proposed a program verifier to prove the correctness of a program (1). At about the same time, Tony Hoare presented formal proof rules for program verification in his landmark paper on the topic (2). In the absence of mechanical verification, use of informal "proof arguments" can result in seemingly correct software with hidden errors. To understand how this is possible and why formal proofs are important, it is useful to discuss a recent example from the literature. In 2006, Joshua Bloch reported an error in using a binary search algorithm that develops when searching large arrays and observed that this problem is typical of such algorithms (3). His example is reproduced on the next page.

The pseudocode for this algorithm, which has been in use for decades, fails in line 6 if the sum of the low value and the high value is greater than die maximum positive integer value. In cases with a large number of elements, the value of "low + high" may overflow, which causes the algorithem not to perform as expected. Amazingly, this simple error has remained hidden in a common piece of code for many years. If this fairly simple and widely used code has an error, it is possible that nearly all current-day software, including safety-critical software, has similar errors, unless it has been verified formally.

```
1: public static int binarySearch(int[] a,
     int key) {
2:   int low = 0;
3:   int high = a.length -1;
4:
5:   while (low <= high) {
6:     int mid = (low + high) / 2;
7:     int midVal = a[ mid] ;
8:
9:     if (midVal < key)
10:         low = mid + 1;
11:     else if(midVal > key)
12:         high = mid - 1;
13:     else
14:         return mid; // key found
15:  }
16: return -(low + 1); // key not found.
17: }
```

Bloch also noted in Ref. 3 that the binary search code had been "proved" to be correct, although in actuality only a typical, informal argument had been given and not a formal proof. If integer bounds are specified and the code undergoes verification through a mechanical verification system (such as the one detailed later), the error would have been caught in a straightforward manner. A key goal is to replace informal proofs with automated ones. The example shows that a verification system must consider all aspects of correctness—including checking that variables stay within their specified bounds. Of course, this means that the verification system must include language support for writing mathematical specifications for the code.

Verification of modern object-oriented software systems involves several challenges:

- It must be scalable and enable independent proofs of correctness for each component implementation using only the specifications of reused components, not their code.
- It must enable full verification to guarantee that the implementation fulfills the completely specified behavior.
- It must be mechanical, requiring programmers to supply assertions, where necessary, but not be concerned with constructing the actual formal proofs.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Verification must be usable not only in relatively simple software, but also in large software systems. To provide scalable verification, the verification process must provide a method to reason about and to verify individual components. For example, suppose that there is a specification S and that I is one of its many implementations. Suppose also that I relies on other components with specifications S1 and S2. To verify the correctness of I with respect to S, the verification system must only require the specifications of reused components (S1 and S2) but not the corresponding implementations. A consequence of this requirement is that a specification should capture all the information needed to reason about and use a component, without divulging implementation details. If the verification system is component-based, allowing for specification and verification of each component in the system, then it might be scaled up for verification of larger systems, which also allows for reasoning about any level of a system even before all implementations are written, resulting in relative correctness proofs, meaning that the entire system is correct, if each subsystem is correct.

We distinguish full verification from “lightweight verification” that is based on lightweight specifications (see the article Formal Specification) with the intent of checking only certain characteristics of the code. Lightweight verification can be used to check for common, specification-independent programming errors such as dereferencing null pointers (4) or unexpected cycles in pointer-based data structure (5). Lightweight verification does not require specification or internal assertions to prove the absence of these simple errors. But other errors may remain. So, to prove the correctness of software (that the realization implements a complete specification of the behavior of the component), full specification and verification are necessary.

For verification to be practical and reliable, it must be mechanical. As observed in the example of the binary search algorithm, nonmechanical verification (because of the many details it relies on) is prone to human error. Given an implementation annotated with suitable assertions, corresponding specifications, and appropriate theorems from mathematics, an automated verification system will postulate a correctness assertion, mechanically. The implementation will be deemed correct if and only if the correctness assertion can be proved, also mechanically.

EXAMPLE CODE AND SPECIFICATIONS FOR VERIFICATION

To illustrate the principles of formal verification involving specification of components, we consider a simple example along with its full verification. To ensure that the results are general and applicable to modem object-based languages, we consider a data abstraction example that encapsulates objects. However, the same principles discussed here can be applied to detect (and correct) the errors in the binary search code. The code given below is intended to reverse a Stack object or variable, and it is typical of the kind of code written in modern imperative languages.

```
Procedure Flip(updates S: Stack);
        Var S_Reversed: Stack;
        Var Next_Entry: Entry;

        While (Depth(S) /= 0) do
                Pop(Next_Entry, S);
                Push(Next_Entry, S_Reversed);
        end;
        S :=: S_Reversed;
end Flip;
```

To complete formal verification of this code, we must have a precise snecification of Stack behavior and the Flip operation in a formal specification language. Figure 1, contains the specification of a Stack component in RESOLVE specification notation; see the Formal Specification article, where the specification of a queue component (similar to this) is described in detail.

The specification of a concept (such as Stack_Template) presents a mathematical model for the type provided by the concept and explains formally the behavior of operations to manipulate variables of that type. In this example, the Stack type is modeled by a mathematical string of entries. The exemplar clause introduces a Stack, S, which is used to describe a generic stack variable in this specification. The concept provides initialization details, constraints for the variables of the type, and specifications for each operation. As implied by the name, the initialization clause describes the initial state of a variable of the type. In this example, initially a Stack is empty. Since the mathematical model for a Stack is a string, the initial value of a Stack is the empty string. The constraint clause formally expresses that every Stack object is always constrained to be within bounds; i.e., the length of the string must be less than Max_Depth, which must be provided when Stack_Template is instantiated for a particular use.

```
Concept Stack_Template(type Entry; evaluates Max_Depth: Integer);
  uses Std_Integer_Fac, String_Theory;
  requires Max_Depth > 0;

  Type Family Stack is modeled by Str(Entry);
    exemplar S;
    constraint |S| <= Max_Depth;
    initialization ensures S = empty_string;

  Operation Push(alters E: Entry; updates S: Stack);
    requires |S| < Max_Depth;
    ensures  S = <#E> o #S;

  Operation Pop(replaces R: Entry; updates S: Stack);
    requires |S| > 0;
    ensures #S = <R> o S;

  Operation Depth(restores S: Stack): Integer;
    ensures Depth = (|S|);

  Operation Rem_Capacity(restores S: Stack): Integer;
    ensures Rem_Capacity = (Max_Depth - |S|);

  Operation Clear(clears S: Stack);

end Stack_Template;
```

Figure 1. A specification of a stack concept.

```
Operation Flip(updates S: Stack);
        ensures S = Rev(#S);
```

Figure 2. A specification of an operation to flip a stack.

The specification for an operation can be viewed as a contract between the client and the implementer. Before a call of any operation, the precondition (or requires clause) must be true. In this example, the Push operation requires that there is room in the Stack for another element. Similarly, to guarantee correct functionality, the Pop operation requires that there is at least one element in the Stack. The implementation of an operation must guarantee that the postcondition (or ensures clause) is satisfied at the end of the procedure if the precondition holds. The ensures clause for Push provides the guarantee that S is updated so that it becomes the original value of E (a parameter of Push) concatenated with the original value of S. RESOLVE denotes the incoming values with a # symbol to differentiate between the incoming and the outgoing values of a parameter in the specification. Pop removes the top entry from the parameter Stack S and replaces the parameter R with the top entry.

Given the specification of the Stack component and the mathematical modeling of Stacks as strings of entries, the Flip operation can be specified formally as in Fig. 2 using the mathematical string reversal operator (Rev). This specification can be written without knowledge (or even existence) of any implementation or Stack_Template.

To facilitate mechanical verification of the Flip code, programmers must annotate loops with suitable assertions as shown in Fig. 3.

To verify code involving a loop, the programmer must include a loop invariant (using the **maintaining** clause), a progress metric expression that decreases with each iteration of the loop (using the **decreasing** clause), and a list of all variables that the loop may change (using the **changing clause**). To prove termination, the decreasing metric must be an ordinal (i.e., it must have a least element). The metric cannot be an integer expression, for example, because it can be decreased f‘orever. Providing the list of changing variables in a loop makes it unnecessary to assert in the invariant that variables not affected by loops remain unchanged. The loop annotations are necessary, in general, to prove the correctness and termination of the loop. If a weak or wrong annotation were supplied, the ability to prove correctness of the operation would be compromised. The literature makes a distinction between partial and total correctness proofs. If code is only partially correct, there is a guarantee of correctness only if the code terminates. Total correctness additionally requires a proof that the code will terminate. In this article, we consider proofs of total correctness.

```
Procedure Flip(updates S: Stack);
        Var S_Reversed: Stack;
        Var Next_Entry: Entry;

        While (Depth(S) /= 0)
                changing S, S_Reversed, Next_Entry;
                maintaining #S = Rev(S_Reversed) o S;
                decreasing |S|;
        do
                Pop(Next_Entry, S);
                Push(Next_Entry, S_Reversed);
        end;
        S :=: S_Reversed;
end Flip;
```

Figure 3. An implementation of an operation to flip a stack.

The programmer-supplied invariant must be true at the beginning and at the end of each iteration, including the first and last iterations. When forming an invariant, the goal of the loop (and the entire operation) must be considered. For example, if Flip had a "**maintaining** |S| + |S_Reversed| = |#S|" clause, it would be a true invariant, but it would not fully describe the behavior of the loop and would not give the verifier the ability to prove the code to be correct with respect to the given specification. Alternatively if the assertion, "**maintaining** #S = S_Revetsed o S," were provided as the invariant, at the time the while loop is processed, the verifier will flag it because the assertion cannot be established to be an invariant. Similarly, if the decreasing clause is incorrect, no proof of the total correctness of the operation can be provided, because the verification systetm cannot guarantee the termination of the loop. Invariants and other annotations should be valid and should be goal-directed, i.e., sufficient to establish code correctness with respect to given specifications.

FORMAL VERIFICATION BASICS

Formal verification must be based on a *sound* and *complete* proof system. Soundness guarantees that if the code is incorrect, the verifier will not say that the code is correct. Completeness, on the other hand, assures that if the code is correct, the verifier will never say that the code is incorrect. Completeness can be only relative because of the inherent incompleteness in any nontrivial mathematical theory, on such as number theory on which proofs of programs are based. The more practical problems for completeness because of inadequate assertions, inexpressive languages for writing necessary assertions, or inadequate proof rules.

A proof system consists of proof rules for each statement or construct in a language. Given the goal and code of an implementation, the verifier applies proof rules (which replace code with mathematical assertions) and then simplifies the assertions with the objective of reducing the final assertion to "true."

For example, consider the following piece of assertive code (a combination of code, facts, and goals), also called a Hoare triple. In the example, S and T are two Stack variables. The swap statement (also the last statement in the Flip code in the previous section) exchanges the values of the participating variables, without introducing aliasing. All code is written and verified within a context, and the *Context* here includes mathematical String_Theory, the Stack_Template specification, as well as declarations of Stack variables. It is not listed explicitly in this article.

```
Context \
        Assume S = empty_string;
                T :=: S;
        Confirm T = empty_string;
```

To simplify the assertive code, a proof rule for the swap statement needs to be applied. In the rule shown below, it is necessary and sufficient to prove what is above the line to prove what follows below the line. This is the typical format of a formal proof rule. In the rule, *C* stands for Context. The notation, RP[x ⇝ y, y ⇝ x], means that concurrently, every x is replaced with y and every y is replaced with x. Intuitively, this rule means that to confirm what follows after the swap statement the same assertion needs to be confirmed before the swap statement but with x and y in exchanged in the assertion.

Proof Rule for the Swap Statement:

C \ code; **Confirm** RP[x ⇝ y, y ⇝ x];

C \ code; x :=: y; **Confirm** RP;

After the application of the swap rule, the following assertive code remains:

```
Assume S = empty_string;
Confirm S = empty_string;
```

The next statements to be processed by the verifier are Assume and Confirm clauses. The rule for removing the Assume clause has the effect of making the resulting assertion an implication. The rule for handling the Confirm clause is simply syntactic: Eliminate the keyword **Confirm**.

Assume Rule:

C \ code; **Confirm** IP **implies** RP;

C \ code; **Assume** IP; **Confirm** RP;

Confirm Rule:

C \ RP;

C \ **Confirm** RP;

In our example, after the application of the Assume Rule, we have the following assertion:

```
Confirm S = empty_string implies
    S = empty_string.
```

Subsequently following the application of the Confirm Rule produces the final assertion:

```
S = empty_string implies
    S = empty_string.
```

Since this implication is provable mechanically using mathematical logic, the assertion simplifies to:

true.

Thus, we can see that our final assertion is true; therefore, assuming the soundness of the proof rules we have employed, we can conclude that the original assertive code is correct. However, if we started out with an incorrect assertive code, as shown below, the verifier would produce a false assertion, assuming completeness of our rules.

Initial (incorrect) assertive code	Generated (unprovable) assertion

Assume S = empty_string; T: = :S;
Confirm S = empty_string;
	S = empty_string
	implies T = empty_string;

EXAMPLE VERIFICATION OF THE STACK FLIP CODE

To illustrate aspects of verifying more typical code, in this section, we consider verification of the Stack Flip code in Fig. 3 with respect to its specification in Fig. 2.

Given a specification and an implementation, the first step in verification is to generate the corresponding assertive code, in which assertions from specifications and programming statements are combined. The rule for generating the assertive code is not shown, but it is straightforward for this example. The requires clause of the operation becomes an assumption at the beginning. Because flip has no requires clause, the assumption is true trivially. Also, it is necessary that constraints on parameters to the operation become an assumption at the start of the assertive code. The ensures clause of the operation needs to be confirmed after the code.

```
Remember;
Assume |S| <= Max_Depth; // Constraints on parameter Stack S
Assume true; // Assumes the requires clause of Flip
        Var S_Reversed: Stack;
        Var Next_Entry: Entry;

        While (Depth(S) /= 0)
            changing S, S_Reversed, Next_Entry;
            maintaining #S = Rev(S_Reversed) o S;
            decreasing |S|;
        do
            Pop(Next_Entry, S);
            Push(Next_Entry, S_Reversed);
        end;
        S :=: S_Reversed;

Confirm S = Rev(#S); // Confirm the ensures clause of Flip
```

Also, the verifier generates the **Remember** statement at the beginning that allows the mechanical assertion generator to maintain the difference between values of the incoming and the outgoing variables until all the statements have been processed and the beginning of the code has been reached. At that time, the Remember Rule allows all “” signs to be removed.

Once assertive code is formed, we apply proof rules for statements in a goal-oriented manner starting with the last statement. If the proof rules are sound, then each application leads to a shorter assertive code (with one less statement) that if proved guarantees that the original assertive code is correct. In this example, the last statement is a swap statement. So the rule for the swap statement (discussed in the last subsection) is applied first, and this leads to the assertive code shown below.

```
Assertive code after processing swap statement:

Remember;
Assume |S| <= Max_Depth;
Assume true;
        Var S_Reversed:Stack;
        Var Next_Entry:Entry;
        While (Depth(S) /= 0)
        ...
        end;
Confirm S_Reversed = Rev(#S);
```

The next statement is a while loop statement. Before we discuss a rule for that statement and apply it, it is instructive to study a rule for the simpler if–then–else statement that is given below.

```
    If-then-else Rule:

C\ code; Confirm Invk_Cond(BE);
  Assume Math(BE); code; Confirm RP;
  C\ code; Assume not(Math(BE));
  code_2; Confirm RP;
---------------------------------------------------
C\ code; If BE then code; else code_2;
   end if; Confirm RP;
```

The if–then–else rule creates two assertions. One assertion assumes that the condition is true and alters the confirm statement based on the code in the “then” section. The other assumes that the condition is false and alters the confirm statement based on the code in the “else” section. These assertions are based on the mathematical form of the conditional statements, denoted by **Math** (BE) in the rule. For example, the condition Depth(S)/ = 0 is transformed to the mathematical statement | S | / = 0 based on the specification of Depth.

The proof rule must also check that any requirements for the use of the condition in the if statement are satisfied. For example, if the if statement has the condition, “Depth(S) + X / = 0,” then the verification system must be able to check the common problem of computational integer overflow also observed in the binary search code example. This is the purpose of the clause **“Confirm** Invk_Cond(BE)” in the rule, where Invk_Cond simply is a conjunction of constraints and preconditions emanating from die evaluation of the condition BE.

We have a while loop to be verified as the next statement in our example assertive code, and it has the following general format with suitable annotations:

```
C \ code; While BE changing VLst; maintaining
   Inv; decreasing P_Exp; do body; end;
   Confirm RP;
```

The rule has three parts (2), two of which are concerned with proving the invariance of the invariant through induction:

Base Case: It must be confirmed that the invariant is true before the while loop.

```
code; Confirm Inv;
```

Inductive Case: If the invariant is assumed true at the start of an iteration of the loop, it can be proved true at the end of the iteration. (For total correctness, it must also be shown that with each iteration of the while loop, the decreasing expression does decrease.)

```
Assume Inv and Math(BE); body; Confirm Inv;
```

The invariant must be strong enough to prove the assertion following the loop statement:

```
(Inv and not Math(BE)) implies RP;
```

A formal version of the while loop proof rule using the if–then–else rule is shown below, where the “then” part corresponds to the inductive proof of the invariant and the “else” part corresponds to the proof of the assertion after the loop statement. After applying this rule, and the rule for an if–then–else statement, the assertions simplify to what is discussed above.

```
          While Loop Rule:

C \ code; Confirm Inv; Change VLst; Assume Inv
    and ?P_Val = P_Exp; If BE then body; Confirm
    Inv and P_Exp < ?P_Val; else Confirm
    RP; end;
---------------------------------------------------
C \ code; While BE changing VLst; maintaining
    maintaining Inv; decreasing P_Exp; do
    body; end; Confirm RP;
```

As a result of the formation of the rule as one unit (instead of the three previously discussed) and to simplify invariants, the verifier-introduced **Change** statement is necessary. The Change statement differentiates between variables that are altered in the loop and variables that the loop leaves unaltered. Without it, the verifier would assume that in the inductive case, when the “**Assume** inv **and** BE” clause is processed, that these are unaltered variables that are modifiable by the application of rules on the code before the while loop. Thus, the statement has the effect of introducing new names for each variable listed in the changing clause by aging them with a “?” and by replacing each variable X witn ?X in subsequent assertions. In the case when a variable has been aged already and ?X is found in subsequent assertions, the verifier will introduce ??X and so on, as necessary. So, all verification-introduced variables will be preceded by one or more question marks, and they are all quantified universally.

The while loop rule yields two pieces of assertive code, one for each path in the if–then–else construct. In part one (when the loop condition is true), we confirm the invariant before the loop and also show that if true at the beginning of an iteration, it will be true at the end of the iteration. In our example, the invariant is the assertion, #S = (Rev(S_Reversed) o S), where #S refers to the value of S at the beginning of the operation and S refers to the current value of S at the start or end of the loop. Therefore, this invariant states that the concatenation of the reversal of the Stack, S_Reversed, and of the current Stack S, will equal the value of S at the start of the operation. Part one also forms an assertion that checks that the decreasing expression does decrease with each iteration by setting the value of the verification variable ?**P_Val** to the ordinal expression, in this example |S|, and then checking that the value of that variable has decreased after the loop body. Type checking will guarantee that the decreasing expression is an ordinal, so no verification obligation is raised by the decreasing clause.

The while rule yields a second piece of assertive code, "part two," associated with exiting the loop. Based on the invariant and the negation of the conditional statement, it checks that the original Confirm statement after the loop can be proven. However, it does not include the first confirm statement (the one confirming the invariant, Inv) because that is already an obligation in part one.

The two pieces of assertive code that result from applying the loop rule (and subsequently the if–then–else statement rule) on the example code are shown below.

Assertive code after processing while statement—Part One:

```
Remember;
Assume |S| <= Max_Depth;
Assume true;
        Var S_Reversed:Stack;
        Var Next_Entry:Entry;
Confirm #S = (Rev(S_Reversed) o  S);
Change S, S_Reversed, Next_Entry;
Assume (#S = (Rev(S_Reversed) o  S) and  ?P_Val = |S|);
Confirm true;           // Pre-Condition of Depth(S)
Assume |S| /= 0;
        Pop(Next_Entry, S);
        Push(Next_Entry, S_Reversed);
Confirm (#S = (Rev(S_Reversed) o  S) and  (|S| <  ?P_Val));
```

Assertive code after processing while statement—Part Two:

```
Remember;
Assume  |S| <= Max_Depth;
Assume true;
        Var S_Reversed:Stack;
        Var Next_Entry:Entry;
Change S, S_Reversed, N ext_Entry;
Assume #S = (Rev(S_Reversed) o  S);
Assume |S| = 0;
Confirm S_Reversed = Rev(#S);
```

First we discuss the proof of the second assertive code because it is easier to discharge. Application of the rules for the Change statement (that renames changing variables in assertions after the change statement with ? marks), the Remember statement (that strips the "#" signs off the values of the incoming variables), and handling of Assume and Confirm clauses lead to the implication shown below. (The verifier also applies the variable declaration rules although they have no impact because neither S_Reversed nor Next_Entry are in the assertion to be confirmed.)

```
Confirm (S = (Rev(?S_Reversed) o  ?S)
                implies  (|?S| = 0
                        implies  ?S_Reversed = Rev(S)));
```

The final clause "?S_Reversed = Rev(S)" must be shown. A mechanical proof system can complete the proof relying on mathematical units for definitions as shown below.

```
Proof:

Goal ?S_Reversed = Rev(S)
≡?S_Reversed = Rev(Rev(?S_Reversed) o  ?S)
  by replacement using implication fact
  S = (Rev(?S_Reversed) o  ?S)
≡?S_Reversed = Rev(Rev(?S_Reversed))
  by simplification using implication fact |?S| = 0
≡?S_Reversed = ?S_Reversed
  by double reverse theorem in String_Theory
≡ true
```

The first part of the while loop rule requires processing the code inside the loop. For this assertive code, the next statement to be processed is a call to Push. Before we present a proof rule to process a call to Push, first it is useful to understand the simpler function invocation rule.

Function Call Rule:

```
C \ code; Confirm Pre_F[x⇝u] and RP[v⇝ f(x)[x⇝
u]] ;
--------------------------------------------------
C \code; v := F(u); Confirm RP;
```

Where the context C includes the specification of operation F:

```
Operation  F(restores  x:T1): T2;
        requires Pre_F(x);
        ensures F = f(x);
```

The rule supposes that the specification of the function is in the context. The function restores and leaves its parameter x unchanged. The return value or the result is some function of the parameter x. Pre_F is the precondition (or the requires clause) for the function, and it must be satisfied before the function is called. Given mis function specification, suppose that the next statement to be processed is a function assignment v := F(u). The proof rule states that to prove the assertion RP after the assignment, it is sufficient to prove that the precondition for the call holds and that RP holds if v is replaced with the result of the function; of course, in the preconditions and postconditions, proper actual parameters must be substituted for formal parameters.

This function assignment rule is the one that needs to be applied to handle the assignment statements in the binary search code, given at the beginning of this article. The "expression assignment rule" for the assignment statement v := E in Ref. 2 is just a special case of function assignment rule, and it assumes expression assignment does not involve any precondition checking, such as for overflows.

Simple Integer Expression Assignment Rule:

```
C \ code; Confirm RP[v ⇝ E] ;
________________________________________
C \code; v := E; Confirm RP;
```

To handle the call to Push in the same code, an operation call rule is needed. In the simple case, the output is expressed as a function of the inputs in the ensures clause and the rule is similar to the one for invoking functions:

Simple Operation Call Rule:

```
C \ code; Confirm Pre_P[ x ⇝ u, y ⇝ v]
   and RP[v ⇝ f(#x,#y)[#x ⇝ u, #y ⇝ v]] ;
________________________________________
C\ code; P(u,v); Confirm RP;
```

Where the context includes the specification of operation P:

```
Operation  P(alters  x: T1, updates  y: T2);
          requires  Pre_P;
          ensures y = f(#x, #y);
```

The ensures clause of Push "S = <#E> o #S" in the Stack_Template specification in Fig. 1 is such that the output value (S) is expressed directly in terms of the inputs (#S and #E). So the simple operation call rule can be applied for the call Push(Next_Entry, S_Reversed) in the example assertive code. After replacing #E and #S in the expression <#E> o #S with, respectively, Next_Entry and S_Reversed, we have Next_Entry o S_Reversed, so we need to replace any S_Reversed in the Confirm assertion with (Next_Entry o S_Reversed). In addition, we need to confirm the requires clause of Push (after proper substitutions), which leads us to the assertive code below where the modified clauses are shown *italicized*.

Assertive code after processing the call Push(Next_Entry, S_Reversed):

```
Remember;
         ...
         Pop(Next_Entry, S);
Confirm
         ((|S_Reversed| <  Max_Depth) and
         (#S = (Rev(<Next_Entry> o  S_Reversed) o  S) and  (|S| <  ?P_Val)));
```

Next, the verification system must process the procedure call Pop(Next_Entry, S). This process is the case of a more general call to an operation where the ensures clause is not expressed as a function of the input values. In fact, the specification might be relational, and there might be many outputs for the same inputs. So we have to prove the Confirm assertion after the call for whatever outputs result from the call, as long as the outputs satisfy the ensures clause. To maintain the difference between variables before and after a procedure call—in which they are possibly altered—the name ?X is used for the value of the variable X after the call. The new variables are implicitly, quantified, universally, which leads us to the general operation call rule shown below.

Operation Invocation Rule:

```
C \ code; Confirm Pre_ P[y ⇝ b]   and
          (Post_P[x ⇝ a,  y ⇝ b]    implies
          RP[a ⇝ a, b ⇝ b]);
________________________________________
C \ code; P( a, b); Confirm RP;
```

Where the context C includes the specification of Operation P:

```
Operation P(replaces x, updates y);
          requires Pre_P;
          ensures Post_P;
```

In Ref. 6, Kulczycki et al. explain how to generalize the operation call rule even more to address calls with repeated arguments and how to pass parameters without introducing aliasing. For this article the rule above is adequate. Application of the rule to the call to Pop in our example leads to the following assertive code after the substitution of actual parameters for formal ones and to use of new names to distinguish values after the call.

Assertive code after processing the call Pop(Next_Entry, S):

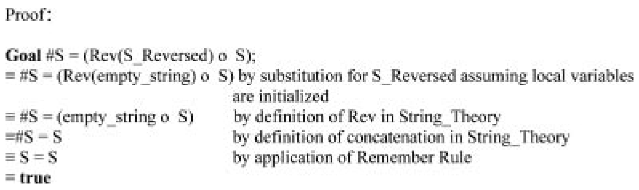
```
Remember;
Assume  |S| <= Max_Depth;
Assume true;
         Var S_Reversed:Stack;
         Var Next_Entry:Entry;
Confirm #S = (Rev(S_Reversed) o  S);
Change S, S_Reversed,  Next_Entry;
Assume (#S = (Rev(S_Reversed) o  S) and  ?P_Val = |S|);
Confirm true;
Assume |S| /= 0;
Confirm
         ((|S| >  0)  and  (S = (<?Next_Entry> o  ?S)  implies
                  ((|S_Reversed| <  Max_Depth) and
                  (#S = (Rev((<?Next_Entry> o  S_Reversed)) o  ?S) and
                  (|?S| <  ?P_Val)))));
```

Our assertive code at this point has no executable statements and two assertions to be confirmed. For ease of explanation, we separate the assertive code and its proof into two parts: The first part corresponds to the base case and confirms the requirement that the invariant holds at the beginning of the while loop. The second part corresponds to the inductive part of the proof of the invariance of the loop invariant.

Assertive code for establishing the invariant just before the loop:

```
Remember;
Assume |S| <= Max_Depth;
Assume true;
         Var S_Reversed:Stack;
         Var Next_Entry:Entry;
Confirm #S = (Rev(S_Reversed) o  S);
```

The proof of the Confirm assertion to show that the invariant Lodz before the loop is given below.



```
Proof:

Goal #S = (Rev(S_Reversed) o  S);
= #S = (Rev(empty_string) o  S) by substitution for S_Reversed assuming local variables
                                 are initialized
= #S = (empty_string o  S)       by definition of Rev in String_Theory
=#S = S                          by definition of concatenation in String_Theory
≡ S = S                          by application of Remember Rule
= true
```

After applying rules for Assume and Confirm clauses to prove the second confirm assertion, we have the following resulting assertive code where the underlined clauses need to be proved.

```
Assertive code for proving the invariant:

Remember;
Assume  |S| <= Max_Depth;
Assume true;
        Var S_Reversed:Stack;
        Var Next_Entry:Entry;
Change S, S_Reversed,  Next_Entry;
Confirm ((#S = (Rev(S_Reversed) o  S) and  ?P_Val = |S|) implies
             (|S| /= 0 implies
                    ((|S| >  0) and  (S = (<?Next_Entry> o  ?S) implies
                          ((|S_Reversed| <  Max_Depth) and
                          (#S = (Rev(<?Next_Entry> o  S_Reversed) o  ?S) and
                          (|?S| <  ?P_Val)))))));
```

After the application of the appropriate rules, the assertion can be proved mechanically by an automated theorem prover, such as Isabelle (7). Because of space considerations, we do not present additional steps of simplification, except to note mat only names change in the confirm assertion to be proved. The Change statement will have the effect of changing names S to ??S (because ?S is already used in the assertion to be proved) and S_Reversed to ?S_Reversed. No Next_Entry exists in the assertion, so no changes are to be made for that variable. By design, after these substitutions, the resulting assertion is devoid of names S_Reversed or Next_Entry, so the variable initialization clause has no effect. With the stripping off of # signs on handling of the remember statement, the resulting assertion to be proved is similar to what needs to be confirmed above, except that the names have changed. So we consider the same assertion as in the assertive code above without name changes, and we simply outline how the obligations can be discharged by a mechanical prover using suitable theorems.

```
Proofs:

Goal (|S| > 0);
≡ true                    by previous implication fact that |S| /= 0 and definition of "| |" in
                          String_Theory

Goal (|?S| <  ?P_Val);
≡ (|?S| <  |S|)            by replacement using implication fact ?P_Val = |S|
≡ (|S - 1| <  |S|)         by replacement using implication fact S = <?Next_Entry> o ?S;
≡ true

Goal #S = (Rev((<?Next_Entry> o  S_Reversed)) o  ?S);
≡ Rev(S_Reversed) o S = Rev(<?Next_Entry> o S_Reversed) o ?S
                          by replacement using implication fact #S = (Rev(S_Reversed) o S
≡ Rev(S_Reversed) o  S = Rev(S_Reversed) o  (<?Next_Entry> o  ?S)
                          by definition of Rev from String_Theory and reassociation of
                           concatenation
≡ Rev(S_Reversed) o  S = Rev(S_Reversed) o   S
                          by replacement using implication fact S = (<?Next_Entry> o  ?S)
≡ true
```

```
Goal |S Reversed| <  Max_Depth;
≡ |Rev(S_Reversed)| < Max_Depth
                          by definition of Rev in String_Theory
≡ |#S| - |S| < Max_Depth
                          by definition of concatenation and length in String_Theory from
                          assumption #S = (Rev(S_Reversed) o S
≡ |#S| <= Max_Depth
                           by proof that |S| > 0
≡ true
                           by constraint that |#S| <= Max_Depth

≡ true
```

Thus, the proposed implementation of Flip has been verified fully with respect to the Flip and Stack specifications assuming that the proof rules presented and discussed here, as a collection, are sound and that all logical steps described here have been checked by a sound, mechanical proof-checking system. The process described here is scalable in that Flip does not need to be reverified for each implementation of Stack_Template: It is verified once for all against the Stack_Template specification. Furthermore, this process is scalable in that client programs using Flip would be verified with respect to Flip's specification: The verification of this Flip implementation need not be (and should not be) revisited with each new call to Flip.

With the proof rules used above, it is also possible to attempt verification on the binary search implementation discussed in the introduction of this article. To verify the binary search algorithm, the code must first be specified properly. Annotations for the while loop must also be included. Then, assuming suitable specification of integer and array operations, the verifier would follow the process discussed in this article, but produce an improvable assertion indicating that either an error exists or that mechanical proof was impossible.

SOUNDNESS AND COMPLETENESS

The development of proof rules, such as the ones described in this article, that can be applied mechanically to assertive programs to establish their correctness is a significant step toward formal reasoning about computer programs. However, two important questions about such a proof system need to be addressed. One of these questions is both obvious and absolutely vital; the other is more subtle, but nevertheless important. The vital issue—whether the rules are sound—can be expressed informally by the question, "Might the rules permit one to prove that a program is correct when, in fact, it is not." The second, more subtle, issue—whether the rules satisfy the completeness property—can be expressed informally by the question, "Are the rules adequate to prove the correctness of every valid program?"

To answer these questions about soundness and completeness, a formal system for verification must define semantics for assertive programs, in addition to the proof rules for establishing correctness. The semantics define the intuitive notion of program *validity*. To define the meaning of "valid" formally, denotational semantics are defined based on the states of a program. Typically, states are described as mappings from variable names to values, but in general, these mappings will need to be relations. To formalize the validity of assertive programs involving specifications, it is necessary to enhance the state space with, three special status denoted by, say, *MW*, *VC*, and $\perp$ Here, *VC* stands for "Vacuously Correct," and it is the result when one of the assumptions of the code fails. For example, the Flip code assumes Push works properly, so it goes to state *VC* and becomes vacuously correct, if Push fails to satisfy its guarantees. *MW* stands for "Manifestly Wrong," and it is the result when an assertive program starts in a normal state but fails to meet one of its obligations. This obligation may be its ensures clause or the requirement of one of the called operations. For example, the Flip code needs to call Pop properly and not pass it an empty stack. If it fails that obligation or fails to reverse a stack, then it goes to *MW*. The symbol, $\perp$ is used to designate "spinning," i.e., the state of a program that has gone into an infinite loop. Once a program has entered any one of these three special states, it remains in that state.

An assertive program is valid if, for all starting states that are normal, the program does not enter *MW* or $\perp$. This explanation corresponds to "total correctness," because it includes the requirement that the program terminate. For partial correctness, validity simply requires that a program not enter the state *MW*.

Using these semantics, soundness is proven inductively by establishing for each proof rule that if its hypotheses are valid semantically, then its conclusion line must also be valid semantically. To examine the completeness question, we need to assume validity of the conclusion and then show that the hypotheses are valid. Of course, here we are talking about relative completeness as defined in the literature.

RELATED WORK

Various aspects of formal verification have received much attention in the literature. An excellent summary of many of the current areas of research in verification may be found in Leavens et al. (8). Principles of reasoning used in this article in the context of RESOLVE notation may be found in Refs. 9–11. Also see the "Formal Specification" article for related specification issues.

Some verification efforts are integrated, whereas some others address specific aspects of verification. An integrated method of verification is based on refinement. This process consists of refinement between levels of abstraction that are based on abstraction relations. Starting from higher levels of abstraction (written as a specification) through refinement a correct lower level result (such as an implementable solution) is developed. Verification then becomes the process of checking the correctness of refinement steps. The Vienna Development Model (VDM) is based on this process (12,13). Each step of refinement creates proof obligations that show the refinement process does not alter the meaning of the original specification.

PVS is both a specification language and a theorem prover (14,15). The included specification language is based on higher order logic and provides a type system. The specification language is accompanied closely by an interactive proof system that together provides the ability to complete verification of large systems.

"Why" is a software verification tool (16). It is directed toward construction of functional programs with assertions, though imperative constructs such as iteration are available. The focus is on typically built-in types, such as arrays rather than modularization or generic data abstractions. Tools associated with the "Why" system can be used to generate verification conditions, similar to the ones given in this article.

Model checking is often used as an alternative to full verification of behavior. Typically, the goal is to check whether (in the context of verification) the model (or implementation) has certain properties (the specification) (17). Property verification is an area of model checking that verifies that certain specific characteristics (or properties) are evident in the implementation. An excellent summary of model checking efforts as well as a specific system for model checking Java programs using JPF (Java Path Finder) can be found in Ref. 18. Symbolic execution principles have been employed in SLAM, which is a model checking system for C programs (19). Verification of safety specifications (20) is an area of ongoing research in property verification.

Research into verification of existing languages must deal with situations, such as aliasing, that greatly complicate modular reasoning. Using Isabelle, a theorem prover (7), Verisoft provides an integrated system for full verification of CO programs, a subset of the C language (20). By design, CO precludes several inherent verification difficulties that exist with the C language, such as aliasing. Correct pointer manipulation, on the other hand, is one goal of the ESC-Java effort (4).

Much research also exists on modular verification of object-oriented programs. Leino et al. and Muller et al. have dealt with verification of pointer behavior for object-oriented programs (4,21). JML, short for Java Modeling Language, is a specification language for Java. In JML, subclasses must have stronger specifications (stronger postconditions, weaker preconditions) than those of their superclass (22,23). Although the initial focus of JML has been on specification and run-time assertion checking, more recent efforts include verification. A precursor to JML is Larch that provides a two-tiered style of specification that requires specifications written in two languages: the Larch Interface Language and the Larch Shared Language (24). Some programs specified using Larch have beeen checked using LP, the Larch Prover. LSL specifications are algebraic. For more details on algebraic specifications, including an example, please see the article "Formal Specification."

ACKNOWLEDGMENTS

This work is funded in part from grants CCR-011381, DMS-0701187, and DUE-0633506 from the U.S. National Science Foundation and by a grant from NASA through the SC Space Grant Consortium. We thank Bill Ogden for his role in developing the proof rules given in this article. Our sincere thanks are due to Bruce Weide and the anonymous referees for their careful reading of this manuscript and for their suggestions for improvement. The assertions in this article have been generated mechanically using the RESOLVE verifier available at www.cs.clemson.edu/∼resolve.

BIBLIOGRAPHY

1. J. King, Symbolic execution and program testing, *Commun. ACM*, **19** (7): 385–394, 1976.
2. C. A. Hoare, An axiomatic basis for computer programming, *Commun. ACM*, **12**(10): 576–580, 1969
3. J. Bloch, http://googleresearch.blogspot.com/2006/06/extra-extra-readall-about-it-nearly.html, 2006.
4. K. R .M. Leino, G. Nelson, J. B. Saxe, *ESC/Java User's Manual*, Technical Note 2000–002, Compaq Systems Research Center, 2000.
5. B. Hackett, R. Rugina, Region-based shape analysis with tracked locations, *Proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'05)*, Long Beach, CA, 2005.
6. G. Kulczycki, M. Sitaraman, W. F. Ogden, B. W. Weide, *Clean Semantics for Calls with Repeated Arguments*, Technical Report RSRG-05-01, Department of Computer Science, Clemson University, Clemson, SC. 2005.
7. The Isabelle Theorem Proving Environment, Developed by Larry Paulson at Cambridge University and Tobias Nipkow at TU Mumich. Available: http://www.cl.cam.ac.uk/Research/HVG/Isabell.
8. G. T. Leavens, J. Abrial, D. Batory, M. Butler, A. Coglio, K. Fisler, E. Hehner, C. Jones, D. Miller, S. Peyton-Jgnes, M. Sitaraman, D. R. Smith, A. Stump, Roadmap for enhanced lnguages and methods to aid verification, *Proceedings of the 5th international Conference on Generative Programming and Component Engineer, 2006.* GPCE '06. New York: ACM Press, 2006, PP. 221–236.
9. J. Krone. The role of verification in software reusability, Ph.D. Thesis, Columbus, OH: The Ohio State University, 1988.
10. W. Heym, Computer program verification: improvements for human reasoning, Ph.D. Thesis, Columbus, OH: The Ohio State University, 1995.
11. M, Sitaraman, S. Atkinson, G. Kulczycki, B. Weide, T. J. Long, P. Bucci, W. Heym, S. Pike, J. E. Hollingsworth, Reasoning about software-component behavior, *Proceedings of the 6th International Conf. on Software Reuse*. Berlin: Springer-Verlag 2000, pp. 266–283.
12. C. B. Jones, *Systematic Software Development using VDM*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall International, 1990.
13. A.A. Koptelov, A.K. Petrenko, VDM vs. programming language extensions or their integration, *Proceedings of the First International Overture Workshop*, Newcastle, 2005.
14. S. Owre, J. M. Rushby, N. Shankar, PVS: A prototype verification system, *Proceedings of the 11th International Conference on Automated Deduction*, 1992, pp. 748–752.
15. S. Owre, J. Rusby, N. Shankar, F. von Henke, Formal verification for fault-tolerant architectutes: prolegomena to the design of PVS, *IEEE Trans. Software Engineer*. **21** (2): 107–125, l995.
16. J. Fillitre, C. Marché, The Why/Krakatoa/Caduceus platform for deductive program verification, in W. Damm and H. Hermanns (eds.), *19th International Conference on Computer Aided Verification*, Lecture Notes in Computer science, Berlin, Germany, 2007.
17. W. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, Cambridge, MA: The MIT Press, 2000.
18. W. Visser, K. Havelund, G. Brat, S. Park, Model checking programs, *IEEE International Conference on Automated Software Engineering*, 2000.
19. T. Ball, R. Majumdar, T. Millstein, S. K. Rajamani, Automatic predicate abstraction of C programs, *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation* (Snowbird, Utah, United States). PLDI '01. New York: ACM Press, 2001, pp. 203–213.
20. D. Leinenbach, W. Paul, E. Petrova, Towards the formal verification of a C0 compiler: code generation and implementation correctness, *Software Engineering and Formal Methods, 2005. SEFM 2005. Third IEEE International Conference on*, 2005.
21. P. Muller, A. Poetzsch-Heffter, Modular specification and verification techniques for object-oriented software components, in G.T. Leavens, M. Sitaraman, (eds). *Foundation of component-Based Systems.*Cambridge: Cambridge University Press, UK 2000.
22. G.T. Leavens, A. L. Baker, C. Ruby, Preliminary design of JML: a behavioral interface specification language for Java, *ACM SIGSOFT Software Engineering Notes*, **31** (3): 1–38, 2006
23. L. Burdy, Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G.T. Leavens, K.R.M. Leino, E. Poll, An overview of JML tools and applications, *Internat. J. Software Tools Technol. Transfer*, **7** (3): 212–232, 2005.
24. J.V. Guttag, J.J. Horning, *Larch: Languages and Tools for Formal Specification*. New York: Springer-Verlag, 1993.

HEATHER K. HARTON
MURALI SITARAMAN
Clemson University
Clemson, South California

JOAN KRONE
Denison University
Granville, Ohio

FORMAL SPECIFICATION

A formal specification of a software component is a mathematical description of the component's behavior. The specification of a component tells its users, or clients, what it does and lets its implementers know what behavior needs to be provided. A specification does not indicate or dictate the details of a how a component is or should be implemented. When the specification is formal, it serves as an unambiguous contract between component users and implementers, which can reduce significantly the costs of software integration and testing that result from miscommunication between users and developers of components in large-scale software development. A formal specification is essential to achieve formal verification (see the article "Formal Program Verification"), whose goal is to prove that an implementation is correct with respect to its specification. Most current techniques for formal specification focus on specifying the *functional* behavior of a software component, but specification of *performance* behavior is becoming just as important.

INTRODUCTION TO FORMAL SPECIFICATION

A specification describes some essential aspect of the software's behavior, such as functionality or performance, without divulging or prescribing implementation details. For a simple example, consider the following Sort operation on Queues:

> **Operation** Sort(Q: Queue); **ensures** In_Order(Q) **and** Is_Permutation(#Q, Q);

Just like programming languages, specification languages have well-defined syntax and semantics. The specification above is given in RESOLVE, which is an integrated programming and specification language (1,2). A behavioral description of sorting in other specification languages may differ syntactically, but the essence of the specified behavior would be the same. For example, some specification languages use different keywords, (e.g., *post* instead *of ensures* clause) to express behavior. Some use alternative notations (e.g., Q and Q′ in the ensures clause instead of #Q and Q) to distinguish the values of parameters before and after operations. A variety of other specification notations, including Larch (3), VDM (4), and Z (5), are summarized in a later section of this article. We have chosen RESOLVE for this article because it is designed to support both full specification and verification of behavior, serving to explain not only specification principles in this article, but also verification principles in the related article "Formal Program Verification". RESOLVE also includes notations for specifying performance behavior, which is an emerging area of importance.

In the specification of the Sort operation, the value of the parameter Q is updated. In general, specifications of such operations may include a *requires* clause that specifies the precondition of the operation, i.e., what must be true in the program state just before the operation begins for the operation call to be legal. In the current case, the Sort operation has no requires clause, which means that the caller can invoke it on any Queue. The *ensures* clause specifies the postcondition of the operation. It indicates what a correct implementation will guarantee just after the operation as long as the caller satisfies the preconditions at the time of the call. In this case, a correct implementation of the Sort operation makes two guarantees: All elements of the resulting Q will be "in order," *and* the current value of Q (i.e., its resulting value at the end of the operation) is a permutation of the incoming or previous value of Q, denoted by #Q. The functions In_Order and Is_Permutation are mathematical functions defined on (the abstract values of) Queues. Although the above specification describes the behavior of the Sort operation, it is not biased toward or based on any one implementation technique. A programmer can implement the operation using any comparison-based sorting algorithm, such as insertion sort, quicksort, or merge sort.

Benefits of Specification

The specification of an operation can be viewed as a contract between the client and the imple-menter of the operation.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

The contract indicates the rights and responsibilities of both parties. The client has a responsibility to satisfy the precondition before invoking the operation, whereas the implementer has a responsibility to satisfy the postcondition if the precondition is met. The same notion of a contract also applies to components and their specifications.

A formal specification is necessary to allow a software developer to prove mathematically that program code is correct with respect to the specification. Although the goal of full program verification is the most fundamental one, formal specification can benefit software developers in other ways as well. In his article "Seven Myths of Formal Methods" (6), Anthony Hall noted that merely developing a formal specification for their project "helped us to clarify the requirements, discover latent errors and ambiguities, and make decisions about functionality at the right stages." Formal specifications can be checked for consistency or completeness even before the code is written, allowing developers to catch certain mistakes early in the development cycle. Developers can use the specifications as unambiguous requirements for their program code, and once the code is written, static checking tools (7) and runtime assertion checkers (8) may be employed to detect a variety of errors.

Informal, Lightweight, and Full Specification

Specifications can be either formal or informal, and the use of one does not preclude the other. Informal specifications use natural language, pictures, or real-world metaphors to describe a component. An informal specification for the sort operation may simply comprise one-line comments for the preconditions and postconditions, as in the following code:

Operation Sort(Q: Queue); **ensures** (* the elements of Q are in sorted order *);

The signature of the operation alone is not a specification because—apart from the appropriately chosen name—it does not describe the behavior of the operation. A good example of a well-organized library of informal specifications is the online documentation for the Java API. These documents indicate how each class or interface behaves. Java interfaces, in contrast, indicate only the signatures of the methods that implementing classes must provide. Some specification languages, such as the Java Modeling Language (JML) (9), allow informal descriptions to be written as comments in syntactic slots where formal assertions are typically required. This process allows programmers to keep their specifications organized when they may not have the resources to write full formal descriptions.

Although informal specifications can give programmers a quick understanding of how a component can be used, they have at least two drawbacks. First, the descriptions they provide are often imprecise, ambiguous, or incomplete. This issue can prevent a client from fully understanding or making proper use of a component. Second, informal descriptions cannot be understood by programming tools such as static checkers and runtime assertion checkers. For example, static checkers such as ESC/Java (7) can use JML specifications to help statically detect potential programming errors and errors in specifications. Tools that perform runtime assertion checking (8,10) determine whether a particular execution of the program is consistent with the machine-readable specifications. Also, tools exist for generating verification conditions (see the article "Formal Program Verification") and for theorem proving [such as PVS (11) or Isabelle/HOL (12)]. New tools continue to be developed that use formal specifications to assist in program testing, debugging, and verification.

A lightweight specification is a formal specification that captures only some aspects of the behavior of an operation or a component. Lightweight specifications vary in their difficulty and utility. At one end are specifications that are extremely simple, but easy to check, such as a specification of an Enqueue operation that merely asserts that the length of the resulting queue is one element longer (10). An alternative lightweight specification might specify that a given sequence of operations does not create cycles or null-pointer accesses (3). In general, lightweight specifications are useful in runtime assertion checking and extended static checking, but their utility is limited in full verification. In contrast to lightweight specification, full specification requires attention to a full range of specification features, including mathematical models for all types, computational bounds, invariants for all loops, and abstraction relations for all implementations. These and other specification features are described below. Full specifications are necessary for full verification of program behavior.

Specification-Based Reasoning

A fundamental principle of component-based software development is that a component should be usable solely on the basis of its specification. Figure 1 shows a design-time diagram of a software system in which circles represent component specifications and rectangles represent implementations (or realizations) of the specifications. A realization will typically use and therefore depend on other components, but reasoning about the correctness of the realization should be modular; i.e., it should only involve the specifications (not the implementations) of reused components. In Fig. 1, for example, to reason that the Queue_Based realization of Messenger is correct, the

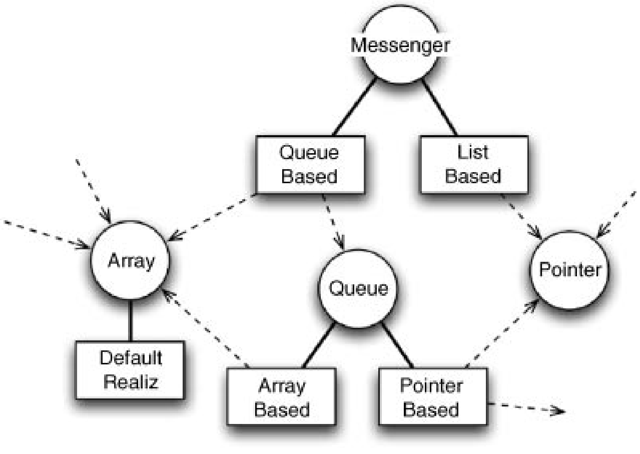


Figure 1. A design-time diagram to illustrate specification-based reasoning.

details of how the Queue specification is implemented should not be necessary. One result of this specification-based style of reasoning is that a developer may substitute one implementation of a component for another without affecting the functional behavior of the system. For example, a straightforward array-based implementation of the Queue component can be substituted with a circular array implementation, and only the performance of the system—not the behavior—will change. Modular or specification-based reasoning is necessary if reasoning about software systems is to be scalable.

SPECIFICATION OF A TYPICAL COMPONENT

This section illustrates the specification of a typical software component. A specification may be model-based or algebraic. We use a model-based style for the central example in this article, deferring a discussion of an algebraic approach to a later section that summarizes various other specification efforts. In a model-based approach, a mathematical model is used to explain each programming type whether it is built in or user defined. Each variable has an abstract value based on the mathematical model of its type, and preconditions and postconditions are written for each operation using that model.

Specification for a Queue Component

Figure 2 shows a model-based specification for a Queue component. The Queue example is sufficiently complex to illustrate the basic ideas of formal specification.

```
Concept Queue_Template (type Entry; evaluates Max_Length: Integer);
    uses String_Theory;
    requires Max_Length > 0;

Type Family Queue is modeled by Str(Entry);
    exemplar Q;
    constraints | Q | ≤ Max_Length;
    initialization ensures | Q | = 0;

Operation Enqueue(alters E: Entry; updates Q: Queue);
    requires | Q | < Max_Length;
    ensures Q = #Q o ⟨#E⟩;

Operation Dequeue(replaces E: Entry; updates Q: Queue);
    requires | Q | > 0;
    ensures #Q = ⟨E⟩ o Q;

Operation Length(restores Q: Queue): Integer;
    ensures Length = | Q |;

Operation Rem_Capacity(restores Q: Queue): Integer;
    ensures Rem_Capacity = ( Max_Length – | Q | );

Operation Swap_Front(updates E: Entry; updates Q: Queue);
    requires | Q | > 0;
    ensures ∃α: String(Entry) ∍ #Q = ⟨E⟩ o α and Q = ⟨#E⟩ o α;

Operation Clear(clears Q: Queue);
    end Queue_Template;
```

Figure 2. A specification for a bounded Queue.

The Queue_Template specification in Fig. 2 is generic—it is parameterized by an Entry type and an integer, Max_Length, which dictates the upper bound for a Queue. It must be properly instantiated with appropriate arguments before it can be used. The specification requires that the expression the user passes as an argument for Max_Length (during instantiation) must be a positive integer. The *uses* clause lists the dependencies. Here, the specification uses String_Theory—a purely mathematical compilation unit that contains properties and definitions related to mathematical strings, including those used in this specification. Automated tools depend on mathematical units when type checking mathematical expressions in specifications, as discussed in a later section in this article, and for generating verification conditions for formal verification.

In RESOLVE, the state space always consists of the abstract values of the currently defined variables. The abstract value space for a type is defined in the specification that provides it. For example, the *type family* declaration in the Queue_Template concept introduces the programming type Queue and associates mathematical strings of entries as the space for the values of Queue variables. Therefore, users can reason about a programming variable of type Queue as a mathematical string of entries. In this concept, the term *type family* is used instead of just *type* because the concept (and, therefore, the type) is generic until it is instantiated, so the declaration of Queue here encompasses an entire family of types. The notion that a programming variable can be viewed abstractly as a pure mathematical value is central to model-based specification and simplifies specification-based reasoning. All variables, even those of basic types, have an abstract (mathematical) interpretation. For example, an array variable may be viewed as a mathematical function from natural numbers to its contents, and an integer variable may be viewed as a mathematical integer, with suitable constraints to capture computational bounds.

The *exemplar* declaration in Queue_Template introduces a variable Q of type Queue to describe properties that hold for any arbitrary Queue variable. For example, the *constraints* clause immediately following the exemplar declaration indicates that the length of any Queue must always be less than Max_Length. Like the requires and ensures clauses, the constraints clause is a mathematical expression. Therefore, the type of Q in the constraints clause is a mathematical string of entries. The String_Theory math unit imported by the uses clause defines the bar outfix operator $|\alpha|$ as the length of string α. The *initialization ensures* clause indicates that each newly declared Queue has a length of zero. The only string that has a length of zero is the empty string, so this is the same as saying that all newly declared Queue objects can be viewed abstractly as empty strings.

A good component specification should provide a suitable set of operations. Together, the operations should be complete functionally, yet minimal. Guidelines for this core set of operations that we call primary operations are given in Ref. 13. To manipulate Queue variables, the current concept describes five operations: Enqueue, Dequeue, Length, Rem_Capacity, Swap_Front, and Clear.

A variety of specification parameter modes appear in the operation signatures. These modes are unique to the RESOLVE specification language, and they have been conceived especially to make specifications easier to understand. The Enqueue operation, for example, specifies its Queue parameter in the *updates* mode, allowing the ensures clause to indicate how it will be modified. In contrast, it lists the Entry parameter in the *alters* mode and indicates only that the Entry may be modified, but it does not indicate how. From this specification, a client knows only that the resulting Entry contains a valid but unspecified value of its type. Therefore, an implementer of Enqueue is not forced to copy the Entry. Copying a variable of an arbitrary type may be expensive, so this specification also allows the implementer to swap Entries, which can be done in constant time (14).

When a parameter is specified in the *replaces* mode, as in the Dequeue operation, its value will be replaced as specified in the ensures clause, regardless of what value it had when the operation was called. Again, this design makes it unnecessary to copy and return the item at the front of the queue, allowing more efficient swapping to be used.

The *restores* parameter mode used in the specification of Length indicates that the value of the parameter after the operation is the same as the value of the parameter before the operation, although the code for the operation may change it temporarily. A restored parameter Q adds an implicit conjunct to the ensures clause that $Q = \#Q$. If a parameter is specified to be in the *preserves* mode, it may not be modified during the operation. In other words, the preserves mode specifies that the concrete state as well as the abstract state remains unmodified, whereas the restores mode specifies only that abstract state remains unmodified. Function operations (operations with return values) should not be side-effecting, so typically all their parameters must be restored or preserved.

The *clears* parameter mode indicates that, after the operation, the parameter will have an initial value. For this reason, the Clear operation does not need an ensures clause: Its only purpose is to give the queue an initial value, which is specified by the clears parameter mode.

The specifications of the operations are given using the requires and ensures clauses. The requires clause of the Enqueue operation states that the length of the incoming Queue Q must be strictly less than Max_Length. The ensures clause states that the new Queue Q has a value equal to the string containing only the old value of E concatenated with the old value of Q. A variable inside of angle brackets, such as $\langle \#E \rangle$, denotes the unary string containing the value of that variable. A small circle represents string concatenation, so $\alpha \circ \beta$ denotes the concatenation of strings α and β. The angle brackets and the concatenation operator are defined in String_Theory.

As an example, suppose $P = \langle \Psi, \Delta, \Phi \rangle$ is a Queue of Tree objects whose maximum length is ten, and suppose $X = \Xi$ is a Tree. Before a client invokes the operation Enqueue(X, P), he is responsible for ensuring that the length of the Queue parameter is strictly less than ten. Since the length of P in our example is three, he can invoke the operation knowing that after the call, $P = \#P \circ \langle \#X \rangle$, or $P = \langle \Psi, \Delta, \Phi \rangle \circ \langle \Xi \rangle = \langle \Psi, \Delta, \Phi, \Xi \rangle$. Since the Entry X is specified in alters mode, the client knows only that X has a valid value of its type: It may be Δ, it may be Ξ, or it may be some other Tree value.

The RESOLVE language has an implicit frame property (15) that states that an operation invocation can only affect parameters to the operation—represented here by P and X. Therefore, the client knows that no other variables in the program state will be modified. This simple rule is possible in RESOLVE, but not necessarily in other languages, such as Java, because in RESOLVE, common sources of aliasing are avoided (for example, by using swapping rather than reference assignment).

Reasoning about the Dequeue operation is similar to reasoning about the Enqueue operation. The Length operation is a function. Like most function operations, this operation has no requires clause. The ensures clause states that $\text{Length} = |Q|$, indicating that the return value of the function is just the length of Q. The Swap_Front operation allows the front Entry of a Queue to be examined (and returned with a second call), without displacing it.

The Queue_Template specification can be implemented in variety of ways. However, users of Queues can ignore those details because all they need to know is described in the specification unambiguously. This developmental independence is crucial for large-scale software construction.

Mathematical Types and Type Checking in Specifications

Specifications that import mathematical types to explain program types give rise to two kinds of typing for the same variable, depending on whether the variable is used in a programming or a mathematical context. Specification languages include mathematical types for this purpose. Extensible specification languages allow new types to be defined and composed from other types. Typical mathematical types include booleans, natural numbers, integers, real numbers, sets, strings, functions, and relations. This small set of types can be composed and reused to specify a variety of computing concepts. For example, mathematical strings can be used in specifying a variety of programming concepts, such as stacks, queues, priority queues, and lists.

Mathematical types, definitions, and appropriate theorems involving those definitions may be described in mathematical theory units that themselves must be robust enough to allow specifications to be built on top of them. For example, the definitions of the string-forming outfix operator "⟨ ⟩" and string concatenation operator "o," both of which are used in the specification of Queue_Template, are given in Fig. 3 from the String_Theory mathematical unit.

…
Definition $\langle x\text{: } \Gamma \rangle\text{: Str}(\Gamma) = \text{ext}(\Lambda, x)$;

Inductive Definition $(s\text{: Str}(\Gamma)) \text{ o } (t\text{: Str}(\Gamma))\text{: Str}(\Gamma)$ **is**
(i) $s \text{ o } \Lambda = \Lambda$;
(ii) $\forall x\text{: } \Gamma,\ s \text{ o } (\text{ext}(t, x)) = \text{ext}(s \text{ o } t, x)$;
…
Inductive Definition $|s\text{: Str}(\Gamma)|$: **N is**
…

Figure 3. Example mathematical definitions in String_Theory.

The mathematical unit String_Theory defines strings over some set Γ, which is a local (mathematical) type used to represent an arbitrary set. Strings are syntactically identified to be the mathematical type Str using two definitions: Λ, the empty string, and ext, a function that extends a string with an object of type Γ. A comprehensive string theory that defines these and other mathematical string notations has been specified in RESOLVE, but its inclusion here is beyond the scope of this article.

When programming objects appear in assertions in specifications, their mathematical types are used rather than their programming types. For the purposes of type checking of these mathematical assertions, we need only know the signatures and types of the definitions involved. For example, the ensures clause of Enqueue "Q = #Q o ⟨#E⟩" is checked for type consistency starting with the values of #Q and #E. The type of #Q evaluates to Str(Entry), and #E has type Entry. The string-forming operator ⟨ ⟩ applied to #E returns an expression of type Str(Entry). The concatenation operator o applied to #Q and ⟨#E⟩ also yields an expression of type Str(Entry). This is compared with the left-hand side of the equality and the type of Q, which also has type Str(Entry). The types match, and the statement is found to be consistent. For another example, if Stacks and Queues are both modeled by mathematical strings of entries, then an ensures clause such as "S = #Q o ⟨x⟩" (where x is of type Entry) would type-check correctly even if S were a Stack and Q were a Queue.

SPECIFICATION OF ASSERTIONS WITHIN IMPLEMENTATIONS

The use of mathematical assertions is not confined to component specifications. Assertions such as abstraction relations, representation invariants, and loop invariants are forms of internal implementation-dependent specifications that need to be supplied along with code. They serve two purposes. First, they help human programmers. They formally document the design intent of the implementers, and they facilitate team development (within an implementation) and ease later maintenance and modification. Second, the assertions are necessary for automated program verification systems that cannot, in general, deduce these assertions that capture design intent.

To illustrate the role and use of implementation-specific, internal specifications, a portion of an array-based implementation for Queue_Template is given in Fig. 4. The Queue data type is represented by a record with one array field (Contents) and two integer fields (Front and Length). The Contents array holds the elements of the Queue, Front is the index of the array that holds the first element in the Queue, and Length is length of the Queue.

```
Realization Circular_Array_Realiz for Queue_Template;

    Type Queue = Record
            Contents: Array 0..Max_Length – 1 of Entry;
            Front, Length: Integer;
        end;

    conventions 0 ≤ Q.Front < Max_Length and
                0 ≤ Q.Length < Max_Length;
    correspondence
```

$$\textbf{Conc}.Q = \prod_{k=Q.Front}^{Q.Front+Q.Length-1} \langle Q.Contents(k \textbf{ mod } Max\_Length)\rangle;$$

```
  Procedure Enqueue(alters E: Entry; updates Q: Queue);
      Q.Contents((Q.Front + Q.Length) mod Max_Length) :=: E;
      Q.Length := Q.Length + 1;
  end Enqueue;

   (* implementation of other Queue operations *)

end Circular_Array_Realiz;
```

Figure 4. A portion of an array-based implementation of Queues.

The conventions clause—the representation invariant—indicates properties that must hold before and after the code for each exported (i.e., not private) Queue operation. The conventions here indicate that both the Front and the Length fields must always be between zero and the value of the Max_Length variable from the Queue_ Template. The correspondence clause—the abstraction relation—plays a fundamental role in specification-based reasoning. It defines the value of the conceptual Queue (**Conc**.Q) as a function of the fields in the Queue's representation. In this abstraction relation, the Π notation indicates string concatenation over a range of values. The relation states that the conceptual Queue is the mathematical string resulting from the concatenation from k = Q. Front to Q.Front + Q. Length − 1 of the unary strings whose elements are given by expression Q.Contents(k **mod** Max_Length). For example, if Max_Length = 5, Contents = [Ψ, Δ, Φ, Θ, Ξ], Length = 3, and Front = 3, then the conceptual Q would be ⟨Contents(3)⟩ o ⟨Contents(4)⟩ o ⟨Contents(0)⟩ = ⟨Θ, Ξ, Ψ⟩. Note that, in this implementation, some elements in the array have no effect on the conceptual value of the Queue. For example, an array value of [⟨Ψ, Γ, Ω, Θ, Ξ⟩] in the above example would yield the same conceptual Queue value. The Π notation is defined such that when the index at the top is smaller than the one at the bottom, it becomes the empty string. This is the reason in the initial state when Front and Length are set to 0 that the conceptual Queue corresponds to the empty string as specified in the initialization ensures clause.

To understand how the representation invariant and abstraction relation are used in reasoning, consider the implementation of the Enqueue operation given in Fig. 6. Let the representation value of the Queue parameter Q be as described above: Q.Contents = [Ψ, Δ, Φ, Θ, Ξ] and Q.Length = Q.Front = 3. Suppose that the element E that

```
Enhancement Sort_Capability (definition (x: Entry) ≾ (y: Entry): B)
        for Queue_Template;
    uses Basic_Ordering_Relations_Theory;
    requires Is_Total( ≾ ) and Is_Transitive( ≾ );

    Operation Sort(updates Q: Queue);
        ensures Is_Conformal_with(≾, Q) and Is_Permutation(#Q, Q);

end Sort_Capability;
```

Figure 5. An enhancement for sorting a Queue.

```
Realization Insertion_Sort_Realiz (
        operation Are_Ordered(restores x, y: Entry): Boolean;
            ensures Are_Ordered = ( x ≾ y );
        ) for Sort_Capability of Queue_Template;

    Procedure Sort(updates Q: Queue) is
        Var Temp: Queue;
        Var E: Entry;
        While Length(Q) /= 0
            maintaining Is_Conformal_with(≾, Temp) and
            Is_Permutation(#Q, Q o Temp);
            decreasing | Q |;
        do
            Dequeue(E, Q);
            Insert_In_Order(E, Temp);
        end;
        Q :=: Temp;
    end Sort;


    Operation Insert_In_Order(alters E: Entry; updates Q: Queue);
        requires | Q | < Max_Length and Is_Conformal_with(≾, Q);
        ensures Is_Conformal_with(≾, Q) and Is_Permutation(#Q o ⟨E⟩, Q);
    Procedure
        (* procedure body goes here. *)
    end Insert_In_Order;

    end Insertion_Sort_Realiz;
```

Figure 6. An implementation for the Queue sort operation.

we want to enqueue has a value of Ω. The conceptual value of the Queue, $\langle\Theta, \Xi, \Psi\rangle$, indicates how the Queue is viewed by someone reading the concept or specification. Therefore, when we check the precondition and postcondition as it is given in the concept, we have to use the abstraction relation to translate the representation value of Q into its conceptual value. This instance of the representation is consistent with all the preconditions of the Enqueue operation. The representation invariant is satisfied, since Length and Front are both between 0 and 4. The precondition of the operation is satisfied, since the length of the conceptual Queue, $\langle\Theta, \Xi, \Psi\rangle$, is strictly less than Max_Length. The implementation of Enqueue first swaps Contents(3 + 3) **mod** 5 = Contents(1) with E, so that Contents(1) becomes Ω and E becomes Δ. Then it increases Length by one so that Length becomes 4. Thus, after the procedure, Q.Contents = $[\Psi, \Omega, \Phi, \Theta, \Xi]$, Q.Length = 4, Q.Front = 3, and E = Δ. This result is consistent with the representation invariant, since Q.Length = 4 is still strictly less than Max_Length = 5. The conceptual value of Q is now $\langle\Theta, \Xi, \Psi, \Omega\rangle$, and the ensures clause, Q = #Q o $\langle$#E$\rangle$, is satisfied since $\langle\Theta, \Xi, \Psi, \Omega\rangle = \langle\Theta, \Xi, \Psi\rangle$o$\langle\Omega\rangle$.

When at least one realization has been implemented for a concept, a developer can create a usable factory or facility by instantiating the concept and indicating the realization that will be used to implement it. Variables can then be declared using any type defined in this way. The code below shows how this is done in RESOLVE:

Facility Int_Queue_Fac is Queue_Template(Integer, 500) **realized** by Circular_Array_Realiz; $\cdots$**Var** Q Int_Queue_Fac. Queue;

A Queue Sorting Enhancement

Figure 5 gives an example of an enhancement for sorting a Queue. In RESOLVE, an enhancement is a way to add additional functionality to a concept without altering the concept specification or its implementations. The enhancement Sort_Capability specifies a *secondary* operation. The use of secondary operations facilitates data abstraction and information hiding and allows developers to keep the number of primary operations in a component to a minimum. The Sort operation can be implemented using a combination of Queue primary operations without directly depending on the internal details of any particular Queue implementation.

The Sort_Capability enhancement is generic. It is parameterized by a mathematical definition of the relation $\precsim$, which takes two parameters of type Entry and returns a Boolean value. The requires clause states that $\precsim$ must be total and transitive (i.e., a total preordering), ensuring that the entries can be sorted. The specification of the sort operation itself is the same as that given in the beginning of this article, except that we have used the idea of "conformal" a higher order predicate: A string Q is conformal with the ordering $\precsim$, if it is arranged according to that order. Both the predicates used in the specification, namely Is_Conformal_with and Is_Permutation, are defined in the mathematical unit String_Theory (imported by Queue_Template).

Figure 6 gives one possible implementation of the Sort operation—an insertion sort. The insertion sort implementation takes a programming operation, Are_Ordered, as a parameter. Any operation can be passed into the implementation as long as it has the same signature as Are_Ordered and has an ensures clause that is consistent with the ensures clause of Are_Ordered. The Are_Ordered operation simply provides a means to check programmatically whether two Entry variables are ordered according to the mathematical definition of $\precsim$.

The developer of an implementation involving a loop must give an invariant for the loop, which is introduced here via the **maintaining** clause. A loop invariant is an assertion that (i) must be an invariant, i.e., true at the beginning and end of each iteration of the loop, and (ii) must be strong enough to help establish the postcondition of the operation.

The loop invariant given in the procedure body for the sort operation is "Is_Conformal_with($\precsim$, Temp) and Is_Permutation(#Q, Q o Temp)." Proving that this invariant is true at the beginning and end of each iteration is done by a verification tool using induction (see the article "Formal Program Verification"). Here, we explain informally why the given assertion is an invariant for this particular instance. Consider a Queue of Trees Q whose value at the beginning of the procedure is $\langle\Theta_7, \Xi_4, \Psi_6\rangle$, where Tree T_i represents a Tree with i nodes, and the Trees are ordered based on the number of nodes they have. We can refer to the incoming value of Q at any state in the procedure as the old value of Q, or #Q. At the beginning of the first loop iteration, Temp has an initial value of its type, so that $\mathrm{Q} = \langle\Theta_7, \Xi_4, \Psi_6\rangle$ and Temp = $\langle\ \rangle$. The loop invariant is true since Temp is in agreement with the order and $Q\ \mathrm{o\,Temp} = \langle\Theta_7, \Xi_4, \Psi_6\rangle\ \mathrm{o}\ \langle\rangle = \langle\Theta_7, \Xi_4, \Psi_6\rangle = \mathrm{Q}$. The body of the loop dequeues the first Tree, Θ_7, from Queue Q and inserts it, in the correct order, into Temp, so at the end of the first loop iteration, $\mathrm{Q} = \langle\Xi_4, \Psi_6\rangle$ and $\mathrm{Temp} = \langle\Theta_7\rangle$. The loop invariant is true since Temp is in order and $Q\ \mathrm{o\,Temp} = \langle\Xi_4,\ \Psi_6,\ \Theta_7\rangle\mathrm{O}\langle\ \rangle = \langle\Xi_4, \Psi_6, \Theta_7\rangle$ is a permutation of #Q. The program state at the beginning of the second iteration is the same as the program state at the end of the first iteration, so the loop invariant remains true. During the second iteration, Ξ_4 is dequeued from Q and inserted in order into Temp so that $\mathrm{Q} = \langle\Psi_6\rangle$ and $\mathrm{Temp} = \langle\Xi_4, \Theta_7\rangle$. The loop invariant holds again since $Q\ \mathrm{o\,Temp} = \langle\Psi_6\rangle\mathrm{o}\langle\Xi_4, \Theta_7\rangle = \langle\Psi_6, \Xi_4, \Theta_7\rangle$ is a permutation of #Q. At the end of the final iteration, $\mathrm{Temp} = \langle\Xi_4, \Psi_6, \Theta_7\rangle$, and $\mathrm{Q} = \langle\rangle$, so the invariant still holds.

A verification tool will also use the invariant to prove the postcondition: that the new Queue value is conformal with the given order and a permutation of the old Queue value. The general case is easy to explain, so we do not restrict ourselves here to $\#\mathrm{Q} = \langle\Theta_7, \Xi_4, \Psi_6\rangle$. At the end of the loop, we know that the loop condition, Length(Q) /= 0, is false and that the loop invariant is true. Therefore, we know that Q is empty, Temp is in order, and Temp is a permutation of Q. When we swap the values of Temp and Q, Q is in order and Q is a permutation of #Q, which is what we needed to show.

Another use of specification in this procedure is the decreasing clause. The decreasing clause introduces a progress metric, which is used to prove that the loop terminates. The progress metric is a natural number that must decrease with each iteration of the loop. Since natural numbers cannot be negative, a proof that the metric decreases implies that the loop terminates (see thie article "Formal Verification"). In the example where $\#\mathrm{Q} = \langle\Theta_7, \Xi_4, \Psi_6\rangle, |\mathrm{Q}| = 2$ at the end of the first iteration, 1 at the end of the second, and 0 at the end of the third. Progress metrics are also used to show termination for recursive procedures.

The following code is an example of a facility declaration that includes the sort enhancement:

Facility Int_Queue_Fac is Queue_Template(Integer, 500) **realized** by Circular_Array_Realiz **enhanced** by Sort_Capability($\leq$) **realized** by Insertion_Sort_Realiz(Int_Less_Eq); .

PERFORMANCE SPECIFICATION

Although specification of functionality has received much attention, specification of performance characteristics, such as time and space requirements, are also necessary for reliable software engineering. When multiple implementations of the same specification occur, developers can use the performance specifications to choose one over the other depending on the needs of their application. This flexibility is essential, since different implementations provide tradeoffs and no single implementation of a concept is likely to be appropriate universally. Just as formal specifications of functionality are necessary for mechanized verification, formal specifications of performance are necessary for verification of performance correctness, which is a key requirement for embedded and other critical systems. In this article, we only show specifications of duration (time requirements) for components using the Queue example. For more details, including analysis of space requirements, please see Ref. 16.

In RESOLVE, performance specifications are given through the profile construct. Figure 11 shows a part of a performance profile called QSC for a class of "space conscious" Queue implementations that keep the internal array free of unutilized garbage (16). The profile in the figure does not make any assumptions about the generic terms Entry and Max_Length. Consequently, its expressions are written using these terms. Although a profile is implementation dependent, it should be free of nonessential implementation details. This capability is provided using a *defines* clause. This clause allows a profile to use constants ($\mathrm{QSC_I}$, $\mathrm{QSC_{I1}}$, $\mathrm{QSC_E}$, $\mathrm{QSC_D}$, $\mathrm{QSC_{Sfe}}$, etc.), whose values will come from the implementation. $\mathbf{R}^{\geq 0.0}$ indicates that their values must be positive real numbers. For each operation, a profile supplies the time necessary to execute the operation using a *duration* clause.

In Fig. 7, the duration expression for initialization is the summation of two terms. The first term, $\mathrm{QSC_I}$, is an implementation-based overall constant overhead. The second term is calculated in two steps. First, the sum of $\mathrm{QSC_{I1}}$ (an implementation-based constant overhead for each Entry) and Entry.**I_Dur** (duration to create an initial valued Entry) is calculated. Then the sum is multiplied by Max_Length to obtain the total time to initialize every Entry in the array structure that is used to represent the Queue internally.

To understand the duration expression for Enqueue, consider the following implementation of the Enqueue

```
Profile QSC short_for Space_Conscious for Queue_Template;
    defines
      $QSC_{I}$, $QSC_{I1}$, $QSC_{E}$, $QSC_{D}$, $QSC_{Sfe}$, $QSC_{L}$, $QSC_{RC}$, $QSC_{C}$: $\mathbf{R}^{\geq 0.0}$;
    Type Family Queue;
        initialization
            duration $QSC_{I}$ + ($QSC_{I1}$ + Entry.I_Dur) * Max_Length;

    Operation Enqueue(alters E: Entry; updates Q: Queue);
        ensures Entry.Is_Init(E);
        duration $QSC_{E}$;

    Operation Dequeue(replaces R: Entry; updates Q: Queue);
        duration $QSC_{D}$ + Entry.I_Dur + Entry.F_Dur(#R) ;

  ...
```

Figure 7. Part of a duration profile for bounded Queue implementations.

operation, which assumes that the Queue is implemented as space-conscious circular array:

```
Procedure Enqueue(alters E Entry; updates Q Queue);
Q.Contents((Q.Front + Q.Length) mod. Max_ Length) :=: E;
Q.Length := Q.Length + 1; end Enqueue;
```

In this implementation, the Enqueue procedure performs the following actions: It accesses a record a total of five times; it swaps an array element once; and it performs one integer assignment, two additions, and a mod operation (Fig. 12). Therefore, for this implementation of the Enqueue operation, QSC_E, used in the profile in Fig. 12, is given the following definition:

$$\text{Definition } QSC_E\text{: } R^{\geq 0.0} = \text{Dur}_{\text{Call}}(2) + 5\cdot\text{Record.Dur} + \text{Array.Dur}_{:=:} + \text{ Int.Dur}_{:=:} + 2.\text{ Int.Dur}_{+} + \text{ Int.Dur}_{\text{mod}};$$

In this expression, $\text{Dur}_{\text{Call}}(2)$ denotes the time to call an operation with two arguments. The duration expression of Dequeue is slightly more complex because it involves initialization of a new Entry variable and a variable finalization.

SUMMARY OF VARIOUS FORMAL SPECIFICATION EFFORTS

The RESOLVE specification language has been used in developing an extensive component library, teaching graduate and undergraduate courses (17,18), and developing commercial software (19). Several other specification languages have found wide use. Formalism is a shared objective of all these languages. This section contains a summary of various efforts.

The Z notation specification language, which was developed at Oxford University Computing Laboratory, is based on set theory and first-order predicate logic (5). A Z statement value can be either true or false and cannot be undefined. Like RESOLVE, Z is typed language: Every variable has a type, reducing errors in specification. For smaller problems, the mathematical notation can be understood easily, but specifications become unattractive as the problem size increases. This obstacle is overcome by introducing schema notation. A schema replaces several statements with a single statement, and it can be composed of several other schemas. This gives Z a modular structure. Just as Z provides logical operators on predicates, it also provides matching operators for schemas. Z specification statements are human readable and, in general, nonexecutable. Z provides both formatting and type-checking tools. Many systems have been built using Z specification, including hardware systems, transaction processing systems, communication systems, graphics systems, HCI systems, and safety-critical systems (20).

VDM-SL (4,21,22) is a model-oriented specification language that originated in the IBM Laboratory in Vienna. It uses propositional calculus and predicate logic. VDM-SL functions do not have side effects and are defined by their signature and preconditions and post-conditions. The Vienna Development Method (VDM) is a program development method based on VDM-SL and tool support. Its object-oriented version is called VDM++. The VDM development cycle starts with an abstract specification and ends with an implementation. The cycle is based on two steps: data reification and operation decomposition. *Data reification* (a VDM term commonly known as data refinement) involves the transition from abstract to concrete data types and the justification of this transition. A reification step is taken if behavior of the reifying and original definitions is guaranteed to be the same. A concrete definition of a function is said to reify or satisfy its abstract definition if for all arguments of the required type satisfying the precondition, the transformation process yields results that are of the required type and satisfy the postcondition.

RAISE (Rigorous Approach to Industrial Software Engineering) is a formal method technique based on VDM (23). It has been used to specify and develop software systems for industrial use. RSL, the specification language of RAISE, supports concurrent and sequential programming features (24).

Larch (3,25) is one of the earlier specification languages and is designed as a family of languages with two tiers of specification: The top tier is a *behavioral interface specification language* (BISL), and the bottom tier is the *Larch Shared Language* (LSL), which is an algebraic style specification language. The LSL is language-independent and is used to describe the mathematical vocabulary used in the preconditions and postcondition specifications. LSL specifications are algebraic rather than model-based. Instead of using mathematical types to model programming types, they introduce a set of axioms that together define the behavior of the component. Figure 8 gives a portion of a Queue specification similar to the one in Ref. 3. In the specification, E is the type for elements in the queue and C is the queue type. Functions are declared using the keyword *introduces*, and their behaviors are defined through the axioms in the *asserts* clause. For additional examples of LSL specifications, see Ref. 3.

Using the shared language, BISL is designed for a given programming language to specify both the interface and the behavior of program modules in that language. The modules are implemented in a particular programming language. Since a BISL is based on a specific programming language, the specification is

```
Queue (E, C): trait
    introduces
        empty: → C
        enqueue: E, C → C
        front: C → E
        dequeue: C → C
        length: C → Int
        isEmpty: C → Bool
        …
    asserts
        C generated by empty, enqueue
        ∀ q: C, e: E
            …
            front(enqueue(e, q)) == if q = empty then e else front(q);
            dequeue(enqueue(e, q)) ==
            if q = empty then empty else enqueue(e, dequeue(q));
            length(empty) == 0;
            length(enqueue(e, q)) == length(q) + 1;
            isEmpty(q) == q = empty;
        …
```

Figure 8. A portion of an LSL specification for a queue.

easy to understand and use. Currently, the available BISLs are Larch/CLU for CLU, Larch/Ada for Ada, LCL for ANSI C, LM3 for Modula-3, Larch/Smalltalk for Smalltalk-80, Larch/C++ for C++, and Larch/ML for Standard ML. Different features of BISL, such as abstraction, side effects, exception handling, name visibility, concurrency, and iterators, depend on how these features are handled by the specific programming language. The LSL checker and LP (Larch Prover) can be used to check Larch statements. First, the LSL checker is used to check the consistency of LSL specification statements and to help generate proof obligation statements. LP uses proof by induction or contradiction to show the correctness of newly created statements. LP is an interactive proof assistant that supports all of the Larch languages.

JML is a BISL tailored for Java (9,26). In JML, specification statements are written just before the header of the method using the Design-by-Contract (DBC) approach. JML specifications are written as special comments to the source file. Hence, it is easier for the programmer to understand than special-purpose mathematical notations. JML can be used with DBC, runtime assertion checking, static checking, specification browsing, and formal verification using theorem prover tools. In JML, inheritance relationships must adhere to the notion of behavioral subtyping: The specifications of the methods in a class must conform to the specifications of the methods they override in the parent class, which ensures that an object of a given type can always be substituted for an object of the parent type without violating the contract described by the specification (27). The Spec# language is similar in spirit to JML but is designed to be used with C# (28).

Other well-known specification languages include Euclid, Eiffel, ANNA, and SPARK. The Euclid programming language, based on Pascal, was developed for system programming and program verification (29–31). Eiffel is designed to support lightweight specifications (10). It was one of the first languages to facilitate run-time assertion checking. ANNA, a language extension of Ada, was designed to develop annotations so that formal methods of specification and documentation can be applied to Ada programs (32). SPARK is also based on Ada and is designed to be used for safety-critical applications (33).

ACKNOWLEDGMENTS

This work is funded in part from grants CCR-0113181, DMS-0701187, and DUE-0633506 from the U.S. National Science Foundation and by a grant from NASA through the SC Space Grant Consortium. We thank the referees, Bill Ogden, and Bruce Weide for their comments on various aspects of this article.

BIBLIOGRAPHY

1. M. Sitaraman and B. W. Weide, eds., *Special Feature: Component-Based Software Using RESOLVE, ACM SIGSOFT Software Engineering Notes 19, No. 4*, 1994, pp. 21–67.
2. M. Sitaraman, S. Atkinson, G. Kulczyski, B. W. Weide, T. J. Long, P. Bucci, W. Heym, S. Pike, J. Hollingsworth, Reasoning about software-component behavior, *Proceedings of the Sixth International Conference on Software Reuse*, Springer Verlag, Vienna, Austria, 2000, pp. 266–283.
3. J. V. Guttag, J. J. Horning, S. J. Garland, K. D. Jones, A. Modet, J. M. Wing, *Larch: Languages and Tools for Formal Specification*, Berlin: Springer-Verlag, 1993.
4. C. B. Jones, *Systematic Software Development using VDM*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall International, 1990.
5. J. M. Spivey, *The Z Notation: A Reference Manual*, Englewood Cliffs, NJ: Prentice-Hall, 1992. Available: http://spivey.oriel.ox.ac.uk/mike/zrm/index.html.
6. A. Hall, Seven myths of formal methods, *IEEE Software*, Vol. **7**(5): 11–19, 1990.
7. K. R. M. Leino, G. Nelson, J. B. Saxe, *ESC/Java User's Manual.* Technical Note 2000–002, Compaq Systems Research Center, 2000.
8. G. T. Leavens, Y. Cheon, C. Clifton, C. Ruby, D. R. Cok, How the design of JML accommodates both runtime assertion

checking and formal verification, *Science of Computer Programming*, Vol. 55. New York: Elsevier, 2005, pp. 185–205.

9. G. T. Leavens, A. L. Baker, C. Ruby, Preliminary design of JML: a behavioral interface specification language for java, *ACM SIGSOFT Software Engineering Notes*, **31** (3): 1–38, 2006.
10. B. Meyer, *Reusable Software: The Base Object-Oriented Component Libraries*, Englewood Cliffs, NJ: Prentice Hall, 1994.
11. S. Owre, N. Shankar, J. Rushby, PVS: A prototype verification system, *Proceedings CADE 11*, Saratoga Springs, NY, 1992.
12. T. Nipkow, L. C. Paulson, M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, LNCS, Vol. 2283. New York: Springer 2002.
13. B. W. Weide, W. F. Ogden, S. H. Zweben, Reusable software components, *Advances in Computers*, Vol. 33, M. Yovits (ed). New York: Academic Press, 1991, pp. 1–65.
14. D. E. Harms, B. W. Weide, Copying and swapping: influences on the design of reusable software components, *IEEE Trans. Software Engineering*, Vol. **17**(5): 424–435, 1991.
15. A. Borgida, J. Mylopoulos, R. Reiter, "...And nothing else changes": the frame problem in procedure specifications, *Proceedings of the 15th International Conference on Software Engineering*, Baltimore, MD, 1993, pp. 303–314.
16. J. Krone, W. F. Ogden, M. Sitaraman, Performance analysis based upon complete profiles, In *Proceedings SAVCBS 2006*, Portland, OR, 2006.
17. M. Sitaraman, T. J. Long, T. J. , B. W. Weide, E. J. Harner, L. Wang, A formal approach to component-based software engineering: education and evaluation, *Proceedings of the Twenty Third International Conference on Software Engineering*, IEEE, 2001, pp. 601–609.
18. B. W. Weide, T. J. Long, *Software Component Engineering Course Sequence Home Page.* Available: http://www.cse.ohio-state.edu/sce/now/.
19. J. Hollingsworth, L. Blankenship, B. Weide, Experience report: using RESOLVE/C++ for commercial software, *Eighth International Symposium on the Foundations of Software Engineering*, ACM SIGSOFT, 2000, pp. 11–19.
20. J. Bowen, *Formal Specification and Documentation Using Z: A Case Study Approach*, International Thomson Computer Press, 1996, Revised 2003.
21. A. A. Koptelov, A. K. Petrenko, VDM vs. programming language extensions or their integration, *Proceedings of the First International Overture Workshop*, Newcastle, 2005.
22. *VDM Specification Language*, 2007. Available: http://en.wikipedia.org/wiki/VDM_specification_language.
23. M. Nielsen, C. George, The RAISE language, method, and tools, *Proceedings of the 2nd VDM-Europe Symposium on VDM—The Way Ahead*, Dublin Ireland, 1988, pp. 376 –405.
24. B. Dandanell, Rigorous development using RAISE, *ACM SIGSOFT Software Engineering Notes, Proceedings of the Conference on Software for Critical Systems SIGSOFT '91*, **16**(5), 29–43, 1991.
25. J. M. Wing, Writing Larch interface language specifications, *ACM Transactions on Programming Languages and Systems*, **9**(1): 1–24, 1987.
26. L. Burdy, Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G. T. Leavens, K. R. M. Leino, E. Poll. An overview of JML tools and applications, *International Journal on Software Tools for Technology Transfer*, **7**(3): 212–232, 2005.
27. B. H. Liskov, J. M. Wing, A behavioral notion of subtyping, *ACM Transactions on Programming Languages and Systems*, **16**(6): 1811–1841, 1994.
28. M. Barnett, K. R. M. Leino, W. Schulte, The Spec# programming system: an overview, *CASSIS 2004*, LNCS Vol. 3362, Springer, 2004.
29. R. C. Holt, D. B. Wortman, J. R. Cordy, D. R. Crowe, The Euclid language: a progress report, *ACM-CSC-ER Proceedings of the 1978 Annual Conference*, December, 1978, pp. 111–115.
30. G. J. Popek, J. J. Horning, B. W. Lampson, J. G. Mitchell, R. L. London, Notes on the design of Euclid, *Proceedings of an ACM Conference on Language Design for Reliable Software*, March, 1977, pp. 11–18.
31. D. B. Wortman, J. R. Cordy, Early experiences with Euclid, *Proceedings of ICSE-5 IEEE Conference on Software Engineering*, San Diego, CA, 1981, pp. 27–32.
32. D. Luckham, *Programming with Specifications: An Introduction to ANNA, a Language for Specifying Ada Programs*, LNCS 260, Berlin: Springer-Verlag, 1990.
33. B. Carre?, J. Garnsworthy, SPARK—an annotated Ada subset for safety-critical programming, *Proceedings of the Conference on TRI-ADA '90 TRI-Ada '90*, 1990, pp. 392–402.

Gregory Kulczycki
Virginia Polytechnic Institute
Blacksburg, Virginia
Murali Sitaraman
Kimberly Roche
Clemson University
Clemson, South Carolina
Nighat Yasmin
The University of Mississippi
University, Mississippi

LAMBDA-CALCULUS

INTRODUCTION

The λ-calculus is an abstract language and system of rules for *higher order programming,* in the sense that in this calculus one can represent programs that modify other programs, as well as programs that operate on numbers.

It was invented in 1928 by an American logician, Alonzo Church, as part of a logical system in which he hoped to describe the foundations of mathematics. This larger system turned out to be inconsistent and was abandoned, but the λ-calculus at its core survived, and Church's group found that, using it, they could give a precise definition of what computability meant. From this definition, they discovered the first rigorous proof that certain important problems could never be solved by computer. (It was published in 1936, see Ref. 1.)

But until the 1970s, λ-calculus had very little use in actual computing, as most practical programming languages were only first order.

Since then, however, many higher-order programming languages have been developed. They incorporate either a form of λ-calculus or something equivalent to it, and earlier studies of λ-calculus have helped to show what these languages can do. Roughly speaking, techniques can be tried out and developed on λ-calculus, and then applied to the more complex practical languages.

To give the reader the flavor of λ-calculus as quickly as possible, we shall describe here its simplest, "pure," form, but with the warning that most applications use more complicated variants.

Additional information on λ-calculus is available in many websites and books on computing, as well as in the introductory account by Hindley and Seldin [2] and the comprehensive book by Barendregt [3].

SYNTAX OF λ-CALCULUS

An arithmetical expression such as "x^2+3" defines a function of x; Church denoted this function by

$$\lambda x\cdot x^2+3$$

Associated with this notation is a rule: for all numbers n,

$$(\lambda x\cdot x^2+3)(n)=n^2+3$$

Church's notation is useful in dealing with expressions that contain more than one variable: For example, the expression "$x+2y$" can be viewed as defining either a function of x, with y held constant, or a function of y, with x held constant. In the λ-notation, these two functions are easily distinguished; they are called, respectively,

$$\lambda x\cdot x+2y,\quad \lambda y\cdot x+2y$$

We have

$$\begin{aligned}(\lambda x\cdot x+2y)(n)&=n+2y\\(\lambda y\cdot x+2y)(n)&=x+2n\end{aligned}$$

Church's λ-notation led to a formal language, whose expressions are called λ-*terms* and are intended to denote operators or programs or mathematical functions.

Definition 1 (λ-terms). (Ref. 2, Def. 1.1.) Assume given an infinite sequence of variables $x, y, z, x_1, y_1, z_1, x_2, y_2, z_2, \ldots$ (to denote arbitrary programs or operators). Then λ-*terms* are constructed as follows:

(a) each variable is a λ- term;
(b) from any λ- terms M and N, construct a new λ-term $(M\,N)$ (to denote the application of operator M to input N);
(c) from any variable x and λ-term M, construct a new λ-term $(\lambda x\cdot M)$ (to denote the function of x that M defines).

Notation 2. A term (MN) is called an *application* and $(\lambda x\cdot M)$ an *abstraction*. (In mathematics the application of M to N is usually called "$M(N)$"; the reason it is called "(MN)" in λ-calculus is merely a historical accident.) To denote arbitrary λ-terms, we shall use capital letters. We shall write

$$M\equiv N$$

to mean that M is the same term as N. Parentheses and repeated λs will often be omitted in such a way that, for example,

$$\begin{gathered}M\,N\,PQ\equiv(((M\,N)P)Q),\\ \lambda xyz\cdot MN\equiv(\lambda x\cdot(\lambda y\cdot(\lambda z\cdot(MN))))\end{gathered}$$

In the rest of this section, let x, y, z, u, v be any distinct variables.

Examples of λ-terms:

(a) $(\lambda x\cdot(xy))$,
(b) $((\lambda y\cdot y)(\lambda x\cdot(xy)))$,
(c) $(x(\lambda x\cdot(\lambda x\cdot x)))$
(d) $(\lambda x\cdot(yz))$

In (c) there are two occurrences of λx in one term; this is allowed by the definition of "λ-term," although discouraged in practice. In (d), there is a term of form $(\lambda x\cdot M)$ such that x does not occur in M; this is allowed, and such terms denote constant-functions.

To show how λ-terms are used as programs, some more apparatus is needed. We shall just give a brief sketch.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Definition 3 (Free and bound variables). (Ref. 2, Def. 1.11.) Any occurrence of a variable x in a term $\lambda x \cdot M$ is said to be *bound* by the λx. The x in λx is said to be *binding and bound*. Any nonbound occurrence in a term is said to be *free*. The set of all variables that occur free in a term P is called

$$FV(P)$$

A *combinator* or *closed term* is a term in which no variable occurs free.

Warning. When free or bound variables are mentioned, it is really *occurrences* of variables that are meant. A variable can have bound occurrences and free occurrences in the same term. For example, consider the term

$$P \equiv (\lambda \upsilon \cdot x)(\lambda y \cdot yx(\lambda x \cdot y \upsilon x))$$

In this term, the leftmost υ is bound and binding, the other υ is free, the leftmost two x's are free, the other two x's are bound, and all three y's are bound. Also

$$FV(P) = \{\upsilon, x\}$$

Definition 4 (Substitution). (Ref. 2, Def. 1.12.) For any terms M, N and any variable x, define $[N/x]\,M$ to be the result of substituting N for each free occurrence of x in M, and changing any λy's in M to prevent variables free in N from becoming bound in $[N/x]\,M$. In detail:

(a) $[N/x]x \equiv N$;
(b) $[N/x]y \equiv y$ (assuming $y \not\equiv x$);
(c) $[N/x](PQ) \equiv [N/x]P[N/x]Q$;
(d) $[N/x](\lambda x \cdot P) \equiv \lambda x \cdot P$;
(e) $[N/x](\lambda y \cdot P) \equiv \lambda y \cdot P$ if $x \notin FV(P)$;
(f) $[N/x](\lambda y \cdot P) \equiv \lambda y \cdot [N/x]P$ if $x \in FV(P)$ and $y \notin FV(N)$;
(g) $[N/x](\lambda y \cdot P) \equiv \lambda z \cdot [N/x][z/y]P$ if $x \in FV(P)$ and $y \in FV(N)$;
(Here z is a variable chosen to be $\notin FV(NP)$.)

Example. Let $M \equiv \lambda y \cdot yx$.

$$\begin{array}{llll} \text{If } N \equiv x\upsilon: & [(x\upsilon)/x](\lambda y \cdot yx) & \equiv \lambda y \cdot [(x\upsilon)/x](yx) & \text{by (f)} \\ & & \equiv \lambda y \cdot y(x\upsilon) & \text{by (a)} - \text{(c)} \\ \text{If } N \equiv xy: & [(xy)/x](\lambda y \cdot yx) & \equiv \lambda z \cdot [(xy)/x](zx) & \text{by (g)} \\ & & \equiv \lambda z \cdot z(xy) & \text{by (a)} - \text{(c)} \end{array}$$

Remark. If (g) were omitted from the definition of substitution, then we would have the undesirable fact that, although $\lambda y \cdot x$ and $\lambda \upsilon \cdot x$ both denote the same operator (the constant-operator whose output is always x), they would come to denote different operators when υ was substituted for x:

$$[\upsilon/x](\lambda y \cdot x) \quad \text{would be } \lambda y \cdot \upsilon, \quad [\upsilon/x](\lambda \upsilon \cdot x) \quad \text{would be } \lambda \upsilon \cdot \upsilon$$

Informally speaking, terms that differ only by changing bound variables have the same meaning. For example, $\lambda x \cdot x$ and $\lambda y \cdot y$ both denote the identity-operator. The process of changing bound variables is defined formally as follows.

Definition 5 (Changing bound variables, α-conversion). (Ref. 2, Def. 1.17.) If $y \notin FV(M)$, we say

$$(\alpha) \quad \lambda x \cdot M \equiv_{\alpha} \lambda y \cdot [y/x]M$$

If P changes to Q by a finite (perhaps empty) series of replacements of form (α), we say

$$P \equiv_{\alpha} Q$$

The relation $\equiv_{\alpha}$ can be proved symmetric (i.e. if $P \equiv_{\alpha} Q$ then $Q \equiv_{\alpha} P$) cf. Ref. 2, Lemma 1.19.

The λ-calculus analog of computation is defined as follows.

Definition 6 (β-contraction, β-reduction). (cf. Ref. 2, Def. 1.24.) A term of form

$$(\lambda x \cdot M)N$$

is called a β-*redex*. (It represents an operator applied to an input.) If it occurs in a term P, and we replace one occurrence of it by

$$[N/x]M$$

then we say that we have *contracted* that occurrence of it. If this contraction changes P to a term P', then we say

$$P \rightarrow_{\beta} P'$$

A finite (perhaps empty) or infinite series of contractions and changes of bound variables is called a β-*reduction*. If it is finite and changes P to Q, then we say that P β-*reduces to* Q or

$$P \twoheadrightarrow_{\beta} Q$$

Example. Here are some reductions; the redex contracted at each step is underlined. In (c) the reduction is infinite, even though the term never changes.

$$\begin{array}{llll} \text{(a)} & \underline{(\lambda x \cdot ((\lambda y \cdot xy)u))(\lambda \upsilon \cdot \upsilon)} & \rightarrow_{\beta} & \underline{(\lambda y \cdot (\lambda \upsilon \cdot \upsilon)y)u} \\ & & \rightarrow_{\beta} & \underline{(\lambda \upsilon \cdot \upsilon)u} \\ & & \rightarrow_{\beta} & u. \\ \text{(b)} & (\lambda x \cdot \underline{((\lambda y \cdot xy)u)})(\lambda \upsilon \cdot \upsilon) & \rightarrow_{\beta} & \underline{(\lambda x \cdot xu)(\lambda \upsilon \cdot \upsilon)} \\ & & \rightarrow_{\beta} & (\lambda \upsilon \cdot \upsilon)u \\ & & \rightarrow_{\beta} & u. \\ \text{(c)} & \underline{(\lambda x \cdot xx)(\lambda x \cdot xx)} & \rightarrow_{\beta} & \underline{(\lambda x \cdot xx)(\lambda x \cdot xx)} \\ & & \rightarrow_{\beta} & \cdots. \end{array}$$

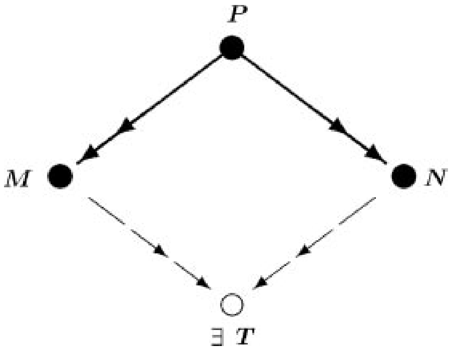


Figure 1. Church-Rosser theorem.

Definition 7 (β-normal form). A term N that contains no β-redexes is called a β-*normal form* or β-*nf*. If $M \twoheadrightarrow_\beta N$, then N is called a β-*normal form of M*.

Not every term has a β-normal form; in fact, Example (c) above shows that

$$(\lambda x \cdot xx)(\lambda x \cdot xx)$$

has none. Examples (a) and (b) above raise the question of whether a term can have more than one β-normal form. In fact it cannot, except for changes of bound variables. This result is a consequence of the following general theorem.

Theorem 8 (Church-Rosser theorem for β-reduction). (cf. Ref. 2, Thm. 1.32.) If $P \twoheadrightarrow_\beta M$ and $P \twoheadrightarrow_\beta N$ (see Fig. 1), then a λ-term T exists such that

$$M \twoheadrightarrow_\beta T, \quad N \twoheadrightarrow_\beta T$$

The Church-Rosser theorem implies that no term P can have more than one β-normal form, as follows. Suppose P could be reduced to two normal forms N_1 and N_2. Then by the theorem, N_1 and N_2 could both be reduced to a term T. But N_1 and N_2 contain no redexes, because they are normal forms. Hence, their reductions to T must have no contractions, only perhaps changes of bound variables. Thus N_1, N_2 and T must be identical except for bound variables.

In Definition 7, N is often called *the* β-normal form of M.

Note 9 (Some combinators). Here are some of the most commonly used combinators, with their standard names and reduction properties. {In this table, "X", "Y", "F", and "G" denote arbitrary λ-terms.}

Note 10 (Functions of many variables). A λ-term F represents an operator that accepts an input X and produces an output that we call *(FX)*. But in mathematics, some functions such as addition need two inputs before they can produce an output. No such functions are given in λ-calculus. However, mathematical many-place functions can be represented indirectly by one-input operators if we choose these operators to be higher order (i.e., operators whose outputs are other operators). For example, let *f* be a 2-place function that accepts numbers m, n as inputs and produces an output-number $f(m, n)$. Define a corresponding one-input operator f^* as follows. First, to each input-value m there corresponds a one-input operator f_m such that, in standard mathematical notation,

$$f_m(n) = f(m, n) \quad \text{for all}\, n$$

Then define f^* by setting

$$f^*(m) = f_m \quad \text{for all}\, m$$

This gives, for all m, n,

$$(f^*(m))(n) = f_m(n) = f(m, n)$$

In the computing community, the act of representing *f* by f^* is often called *currying*, after Haskell Curry, who was one of the contributors to λ-calculus. (Although Curry never claimed it to be his own idea.)

Note 11 (Computable functions). (cf. Ref. 2, Ch. 4.) If the combinators $\overline{0}, \overline{1}, \overline{2}, \ldots, \overline{n}, \ldots$ in Note 9 are taken to represent the numbers 0, 1, 2, …, n, …, then certain other combinators represent mathematical functions or operators. For example, let

$$\overline{Add} \equiv \lambda uvxy \cdot ux(vxy) \quad \overline{Mult} \equiv \lambda uvx \cdot u(vx)$$

Then, it can be shown that, for all natural numbers m and n,

$$\overline{Add}\,\overline{m}\,\overline{n} \twoheadrightarrow_\beta \overline{m+n} \quad \overline{Mult}\,\overline{m}\,\overline{n} \twoheadrightarrow_\beta \overline{m \times n}$$

$\mathbf{I} \equiv \lambda x \cdot x$	(identity combinator):	$\mathbf{I}X \twoheadrightarrow_\beta X$
$\mathbf{B} \equiv \lambda xyz \cdot x(yz)$	(composition),	$\mathbf{B}FGX \twoheadrightarrow_\beta F(GX)$
$\mathbf{C} \equiv \lambda xyz \cdot xzy$	(commutator),	$\mathbf{C}FXY \twoheadrightarrow_\beta FYX$
$\mathbf{K} \equiv \lambda xy \cdot x$	(constant-forming),	$\mathbf{K}XY \twoheadrightarrow_\beta X$
$\mathbf{S} \equiv \lambda xyz \cdot xz(yz)$	(substitution & composition),	$\mathbf{S}FGX \twoheadrightarrow_\beta FX(GX)$
$\mathbf{W} \equiv \lambda xy \cdot xyy$	(doubling),	$\mathbf{W}FX \twoheadrightarrow_\beta FXX$
$\mathbf{Y} \equiv (\lambda ux \cdot x(uux))(\lambda ux \cdot x(uux))$	(fixed-point combinator, cf. Ref.2 §3B),	$\mathbf{Y}F \twoheadrightarrow_\beta F(\mathbf{Y}F)$
$\overline{0} \equiv \lambda xy \cdot y$	(to represent zero),	$\overline{0}FX \twoheadrightarrow_\beta X$
$\overline{n} \equiv \lambda xy \cdot x^n y$	(to represent number n),	$\overline{n}FX \twoheadrightarrow_\beta F^n X$

where

$$F^n X \equiv F(F(\ldots(FX)\ldots)) \text{ with } n \text{ “}F\text{”s}$$

In general, a k-argument function f of natural numbers is said to be *represented* or λ-*defined* by a λ-term M when

$$M\overline{n_1}\ldots\overline{n_k} \twoheadrightarrow_\beta f(n_1,\ldots,n_k)$$

for all natural numbers $n_1, \cdots, n_k$. It can be proved that every function that is computable (by any of the standard idealized computers in the literature, such as a Turing machine) is λ-definable.

In this sense, all computable functions can be programmed in λ-calculus.

TYPES IN λ-CALCULUS

Types are expressions that are intended to denote sets: When a program or function or operator f changes members of a set denoted by σ to members of a set denoted by τ, we may say that f has the *type*$(\sigma \to \tau)$. Types were introduced into λ-calculus by Church in Ref. 4; he assigned to every variable a unique type and constructed composite typed terms by two rules, which may be written in modern notation as follows:

$$\left\{\begin{array}{l}\text{(i) from typed terms} M^{(\sigma\to\tau)} \text{ and} N^{\sigma}\text{, construct } (M^{(\sigma\to\tau)}N^{\sigma})^{\tau}\\ \text{(ii) from a variable} x^{\sigma} \text{ and term} M^{\tau}\text{, construct } (\lambda x^{\sigma}\cdot M^{\tau})^{\sigma\to\tau}\end{array}\right\} \qquad (1)$$

Systems of λ-calculus in which types are part of a term's construction in this way are called *Church-style* systems. In such a system, for example, the expression $(\lambda x \cdot x)$ is not a term; but, for every type τ, there is a variable v^τ from which we can build a term

$$(\lambda v^\tau \cdot v^\tau)^{\tau\to\tau}$$

Roughly speaking, for every definable set a Church-style system has an identity-operator on that set, but it does not have a "universal" identity operator.

In contrast, in a *Curry-style* system, the λ-terms are constructed without types, and types are assigned to terms by formal rules (as in Definition 13 below). For example, in such a system, $(\lambda x \cdot x)$ is a term and it receives an infinite number of types:

$$a\to a,\ b\to b,\ (a\to b)\to(a\to b) \quad \text{etc.} \qquad (2)$$

Also in the Curry style, types may contain parameter-symbols called *type-variables;* if a is a type-variable, then all the types in Equation (2) can be obtained from the single type $a\to a$ by substitution; $a\to a$ is called a *principal type* of $(\lambda x \cdot x)$.

Many different type-systems are based on many varieties of λ-calculus, and some of these are intermediate between these two styles.

To show one type-system in more detail, we shall here describe a simple Curry-style system, which is designed for use with functions of natural numbers {0, 1, 2, 3, ...}. It is a variant of system $\mathrm{TA}_\lambda^\to$ in Ref. 2, Ch. 12 and of system λ → -*Curry* in Ref. 5, §3.

Definition 12 (Simple types). (cf. Ref. 2, Def. 11.1.) Let **N** be a symbol to denote the set of all natural numbers; we shall call **N** a *type-constant*. Let $a, b, c, \ldots$ be a sequence of other symbols; we shall call them *type-variables*. Then, *types* are expressions constructed from type-constants and type-variables by this rule:

$$\text{from any types } \sigma, \tau, \text{ construct } (\sigma\to\tau) \qquad (3)$$

Examples. The following are types:

$$a,\quad \mathbf{N},\quad (a\to b),\quad (a\to\mathbf{N}),\quad ((\mathbf{N}\to a)\to(b\to\mathbf{N}))$$

Notation. Greek letters ρ, σ, τ, will denote arbitrary types. Parentheses may be omitted from types in such a way that, for example,

$$\rho\to\sigma\to\tau \equiv (\rho\to(\sigma\to\tau))$$

Definition 13 (System $\mathbf{TA}_\lambda^\to$). (cf. Ref. 2, Def. 12.6.) A *type-assignment formula* or *TA-formula* is any expression M: τ, where M is a λ-term (as in Definition 1), and τ is a type.

$\mathrm{TA}_\lambda^\to$ has the following three rules. Roughly speaking, each rule means that from the expressions above the line, one may deduce the expression below. Deductions are built as trees, with one formula at the bottom and assumptions at the tops of branches. (See Ref. 2, §§12.1–12.6 for details; $\mathrm{TA}_\lambda^\to$ is very like Gentzen's "Natural Deduction" systems in logic, cf. Ref. 6.)

$$(\to e),\ \text{the} \to\text{-elimination rule}: \quad \frac{M:\sigma\to\tau \quad N:\sigma}{(MN):\tau,}$$

$$(\to i),\ \text{the} \to\text{-introduction rule}: \quad \frac{\begin{array}{c}[x:\sigma]\\ \vdots\\ M:\tau\end{array}}{(\lambda x\cdot M):(\sigma\to\tau),}$$

$$(\equiv_\alpha),\ \text{rule of bound variables}: \quad \frac{M:\tau \quad M\equiv_\alpha N}{N:\tau.}$$

Explanation. Rule (→ i) means that if we already have made a deduction of M: τ from x: σ and perhaps a set Γ of other assumptions not containing x, then we can deduce, from Γ alone, the statement $(\lambda x \cdot M) : (\sigma\to\tau)$. Also, after we use rule (→ i), we enclose the assumption $x:\sigma$ in brackets wherever it occurs in the deduction-tree above M: τ, to show that it is now no longer regarded as an assumption. It is now called a *cancelled* or *discharged* assumption (Cf. Ref. 2, §12.1.). So a deduction grows in two ways, by adding new conclusions at the bottom, and by adding brackets to assumptions. A deduction with all assumptions discharged is called a *proof*.

Example. Let $\mathbf{S}\equiv \lambda xyz\cdot xz(yz)$ and let ρ, σ, τ be any types. Fig. 2 shows a proof of

$$\mathbf{S}:(\rho\to(\sigma\to\tau))\to((\rho\to\sigma)\to(\rho\to\tau))$$

Note that each assumption is undischarged at the start of the deduction and then becomes discharged later.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : \rho\to(\sigma\to\tau)]\ [z:\rho]}{xz:\sigma\to\tau}\,(\to\mathrm{e}) \quad \dfrac{[y:\rho\to\sigma]\ [z:\rho]}{yz:\sigma}\,(\to\mathrm{e})}{xz(yz):\tau}\,(\to\mathrm{e})}{\lambda z\cdot xz(yz):\rho\to\tau}\,(\to\mathrm{i})\ \text{(discharge both occurrences of “}z:\rho\text{”)}}{\lambda yz\cdot xz(yz):(\rho\to\sigma)\to(\rho\to\tau)}\,(\to\mathrm{i})\ (\text{dis. “}y:\rho\to\sigma\text{”})}{\lambda xyz\cdot xz(yz):(\rho\to(\sigma\to\tau))\to((\rho\to\sigma)\to(\rho\to\tau))}\,(\to\mathrm{i})(\text{dis. “}x:\rho\to(\sigma\to\tau)\text{”})$$

Figure 2.

Definition 14. If $x_1,\ldots,x_n$ are distinct variables and there is a deduction of M: τ with all assumptions discharged except those in the set $\{x_1\colon \rho_1, \ldots, x_n\colon \rho_n\}$, then we say the formula M: τ has been *deduced from* $\{x_1\colon \rho_1, \ldots, x_n\colon \rho_n\}$, or

$$x_1 : \rho_1, \ldots, x_n : \rho_n \vdash M : \tau$$

In the special case that $n = 0$ (i.e. that there exists a proof of M: τ), we say

$$\vdash M : \tau$$

For additional reading on λ-calculus with types we recommend Ref. 5 for a survey and comparison of several systems, Ref. 2 Chs. 10–13 for introductions to some of these, or Ref. 7 for a comprehensive and computer-oriented textbook.

λ-CALCULUS IN COMPUTER SCIENCE

The λ-calculus can be considered the smallest commonly used universal programming language. Quoting Peter Landin [8]: “Whatever the next 700 programming languages turn out to be, they will surely be variants and extensions of λ-calculus.”

In this section, we will survey more interesting developments of computer science in which the λ-calculus has played a crucial role. Our main source for the historical information will be Cardone and Hindley [9].

Computability Theory

As we mentioned in Note 11, one can represent numbers and computable functions on numbers by λ-terms. We gave λ-terms to represent two standard mathematical functions, but we did not show how to λ-define the predecessor function, which is much more tricky. Indeed, when Kleene, who was a student of Church, showed to his teacher a λ-definition he had constructed for the predecessor function, Church conjectured that all intuitively computable functions must be λ-definable (Ref.10, P.186). This intuition of Church led to what came to be known as “Church’s Thesis”:

> λ-*definability exactly captures the informal notion of effective computability*.

In fact, Kleene analyzed the notion of λ-definability and showed that every recursive function can be coded (by means of “normal forms”) into the λ-calculus [11, 12]. Church related the notion of effective computability to that of recursive function and hence, to λ-definability. He proved at the same time that the question of the equivalence of two λ-terms (not in normal-form) is undecidable [1]. Immediately after the work of Church but independently of it, Turing introduced his machine approach to computation and proved the undecidability of the halting problem [13]. Then, learning of Church’s work, he proved the equivalence between his notion of computability and that of λ-definable function [14].

Functional Programming

At the end of the 1950s, John McCarthy at Stanford University proposed the first functional language, *LISP* (LISt Processing) [15]. He wrote: “To use functions as arguments, one needs a notation for functions, and it seemed natural to use the λ-notation of Church.” *LISP* allows the reduction of a β-redex (see Definition 6)

$$(\lambda x \cdot M)N$$

only if N is a *value* (i.e., either a λ-abstraction or a constant). Usually such a β-redex is called a β_υ-redex [16]. Moreover, *LISP* does not allow reduction under a λ-abstraction (as we did in Example (b) after Definition 6); and substitution in *LISP* captures free variables, that is, clauses (f)-(g) of Definition 4 are replaced by:

$$(\mathrm{f}') \quad [N/x](\lambda y \cdot P) \equiv \lambda y \cdot [N/x]P$$

where any occurrences of y that may be free in N (l.h.s.) are captured by the λ-abstraction in the r.h.s. This difference is usually expressed by saying that the λ-calculus uses *static binding*, whereas *LISP* uses *dynamic binding*. New *LISP* dialects (*Common LISP* [17], *SCHEME* [18], etc.) use static binding. *LISP* today is a family of computer programming languages used in artificial intelligence, Web development, finance, computer science education, and a variety of other applications.

Although *LISP* is an untyped language, *ML* (Meta-Language) is a functional programming language based on type assignment for λ-calculus (cf. Definition 14): The programmer can write untyped programs, but the compiler will either infer types or return an error message [19, 20]. Several languages are in the *ML* family today: *SML* (Standard ML) [21], and *CAML* (Categorical Abstract Machine Language) [22]. *ML’s* applications include language design and manipulation (compilers, analyzers, theorem provers), bioinformatics, financial systems, a genealogical database, and a peer-to-peer client/server program.

Evaluation Strategies

Examples (a) and (b) after Definition 6 showed that the same λ-term can be reduced in two different ways by choosing different β-redexes to contract. The Church-Rosser Theorem for β-reduction (Theorem 8) shows that this choice does not influence the final result, if any. But it can influence the number of contractions and hence affect efficiency, because for example:

$$\begin{aligned}\underline{(\lambda x\cdot xx)((\lambda y\cdot y)(\lambda z\cdot z))} &\rightarrow_\beta \underline{(\lambda y\cdot y)(\lambda z\cdot z)}((\lambda y\cdot y)(\lambda z\cdot z))\\ &\rightarrow_\beta \underline{(\lambda z\cdot z)((\lambda y\cdot y)(\lambda z\cdot z))}\\ &\rightarrow_\beta \underline{(\lambda y\cdot y)(\lambda z\cdot z)}\\ &\rightarrow_\beta (\lambda z\cdot z)\end{aligned}$$

while

$$\begin{aligned}(\lambda x\cdot xx)\underline{((\lambda y\cdot y)(\lambda z\cdot z))} &\rightarrow_\beta \underline{(\lambda x\cdot xx)(\lambda z\cdot z)}\\ &\rightarrow_\beta \underline{(\lambda z\cdot z)(\lambda z\cdot z)}\\ &\rightarrow_\beta (\lambda z\cdot z)\end{aligned}$$

Furthermore, the choice of β-redexes can lead either to a terminating or to a diverging computation, as in:

$$\begin{aligned}\underline{(\lambda xy\cdot y)((\lambda z\cdot zz)(\lambda z\cdot zz))} &\rightarrow_\beta (\lambda y\cdot y)\\ (\lambda xy\cdot y)\underline{((\lambda z\cdot zz)(\lambda z\cdot zz))} &\rightarrow_\beta (\lambda xy\cdot y)\underline{((\lambda z\cdot zz)(\lambda z\cdot zz))}\\ &\rightarrow_\beta (\lambda xy\cdot y)\underline{((\lambda z\cdot zz)(\lambda z\cdot zz))}\\ &\rightarrow_\beta \cdots\end{aligned}$$

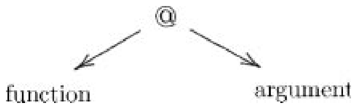
In the first reductions of both the above examples, we chose always to contract the left-most outer-most β-redex, whereas in the second reductions we chose always the left-most outer-most β_υ-redex.

An *evaluation strategy* is a systematic way of indicating which β-redex or β_υ-redex to contract in an arbitrary λ-term. The two strategies shown above are called, respectively, *call-by-name* and *call-by-value* [16]. They correspond respectively to call-by-name and call-by-value parameter passing in procedures of programming languages. For example, considering the procedure $square(x) = x \times x$, the evaluation of $square(2+1)$ when x is a call-by-name parameter gives $(2+1)\times(2+1)$, whereas when x is a call-by-value parameter, it gives $square(3)$.

An interesting strategy is that of *lazy evaluation*, which increases efficiency by waiting until the last possible moment to evaluate a term and by never reducing under an abstraction. This strategy allows one also to deal with infinite data structures. For example, one could construct a function that creates the infinite list of Fibonacci numbers. The calculation of the nth Fibonacci number would be merely the extraction of that element from the infinite list. The entire infinite list would never be calculated, only the values that influence a particular calculation. *MIRANDA* [23] and *HASKELL* [24] are examples of lazy functional programming languages.

The study of evaluation strategies for λ-calculus strongly influenced the implementation of functional programming languages.

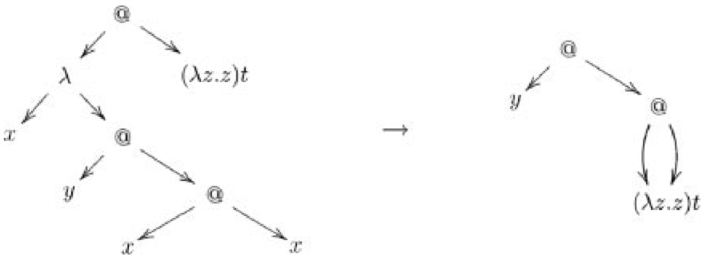
Various other techniques have been developed to get more efficient implementations; we just mention here *graph reduction* [25]. In graph reduction, a λ-term is represented as a directed graph without cycles:



and duplicated computation of shared subterms is avoided by duplicating arrows in the graph. For example, the contraction

$$(\lambda x\cdot y(xx))((\lambda z\cdot z)t) \rightarrow y(((\lambda z\cdot z)t)((\lambda z\cdot z)t))$$

is written as



The efficiency of graph reduction is enhanced when a functional program is translated to a fixed set of functions without free variables (*combinators*) [26] or when a functional program is translated to a set of functions without free variables and the members of the set are selected to be optimal for that program (*super combinators*) [27].

Thanks to these techniques and to the computational power of modern computers, functional languages have achieved a degree of practicality that was previously the prerogative of imperative languages.

Semantics

ALGOL 60 (ALGOrithmic Language) [28] was an imperative language designed in 1960 by the Working Group 2.1 of IFIP (International Federation for Information Processing). Peter Landin translated the core of *ALGOL 60* into λ-calculus extended with assignment [29]. This translation was the first formal semantics of a real programming language.

Two key problems of programming language semantics are:

- the semantics of looping constructs (**while, until**, . . . statements), and of recursive procedures, requires the solution of recursion equations. For example Euclid's algorithm is a recursive procedure: $gcd(n,m) = \textbf{if}\ m = 0\ \textbf{then}\ n\ \textbf{else}\ gcd(m, n\ \textbf{modulo}\ m)$
- (higher order) procedures can receive other procedures as arguments, so the meaning of a procedure must be both a function and an argument.

These problems arise also in giving the semantics of λ-calculus, because:

- recursion equations can be solved using the fixed-point combinator (see Note 9 and [Ref. 2, Corollary 3.3.1]);
- each λ-term can appear either in function or in argument position, so its meaning must be both a function and an argument.

Dana Scott gave the first actual model of λ-calculus in 1969. He solved the domain equation

$$\mathcal{D} = [\mathcal{D} \rightarrow \mathcal{D}]$$

in the category of complete lattices and continuous functions [30], and his model became the basis of the denotational semantics of programming languages [31].

Introductions to models of λ-calculus in general and Scott's model in particular can be found in Ref. 3, Chs. 5 and 18–20, and Ref. 2, Chs. 14–16 .

More recently, the λ-calculus provided models of biological phenomena [32]. Moreover the λ-calculus of objects (i.e., a λ-calculus enriched with suitable operators) was used to give the semantics of, and to study the types of, object-oriented languages [33]. Last, we mention that a linear and reversible λ-calculus with constants can represent atomic quantum logic gates [34].

Proofs and Programs

It is easy to check that by erasing λ-terms from the rules of the system $\mathrm{TA}_{\lambda}^{\rightarrow}$ of Definition 13 and omitting type-constants, we get the intuitionistic logic of implication. This observation is at the basis of the *Curry-Howard isomorphism* [35, 36], which involves the following correspondences:

types	⇔	logical formulas
closed terms	⇔	proofs
β-reduction	⇔	cut elimination

Certified Programming is centered on the Curry-Howard isomorphism. The key idea in fact is that the development of a program to satisfy a specification is the same as finding a proof of a logical formula. In this way, the program obtained comes with its correctness proof. The typed λ-calculus is used as a programming language, a specification language and a programming logic. For example, a constructive proof of the sentence "for all pairs of integers n_1, n_2 there is an integer m such that $m = n_1 + n_2$" is a program that computes the sum of two integers. (Usually, the programs obtained in this way are fairly large and, for efficiency, methods of deleting noncomputational parts are currently being studied.)

In the late 1960s, N. G. de Bruijn started the *AUTOMATH* (AUTOmated MATHematics) project [37], which was based on a λ-calculus with *dependent types* (types which can contain terms). He designed a language for expressing complex mathematical theories in such a way that a computer can verify a proof's correctness. The full textbook "Grundlagen der Analysis" of E. Landau has been translated into this language and verified by computer (Part D of Ref. 38).

In *Martin-Löf's Constructive Type Theory* [39–41] the following identifications can be made:

- a is an element of the set τ
- a is a proof of the proposition τ
- a is an object with the type τ
- a is a program with the specification τ
- a is a solution to the problem τ.

Martin-Löf developed his type theory (based on a λ-calculus with dependent types) between 1970 and 1980 as a foundational language for mathematics. He designed a functional programming language that includes its own logic.

The PRL (Proof/Program Refined Logic) Project [42] focuses on implementing computational mathematics and on providing logic-based tools that help to automate programming. A proof of a logical formula is compiled into an executable and certified code (essentially a λ-term). *Nuprl* (pronounced "new pearl") [43] is a family of proof development systems for the incremental verification of software systems' properties. Martin-Löf's type theory strongly influenced the development of Nuprl.

Coc (Calculus of Constructions) [44] combines dependent types with universal quantification on type variables. It is a higher-order typed λ-calculus in which types are first-class values: it allows one to define functions from, say, integers to types and types to types, as well as functions from integers to integers. Coc is the basis of *Coq* [45], which is a proof assistant that

- handles mathematical assertions,
- mechanically checks proofs of these assertions,
- helps to find formal constructive proofs,
- extracts a certified program from the constructive proof of its formal specification.

Coq is written in the *OCAML* (Objective Caml) [46] system, which is the main implementation of the *CAML* language.

BIBLIOGRAPHY

1. A. Church, An unsolvable problem of elementary number theory, *Am. J. Mathemat.*, **58**: 345–363, 1936.
2. J. R. Hindley and J. P. Seldin, *Lambda-calculus and Combinators, an Introduction*. Cambridge, U.K.: Cambridge University Press, 2008.
3. H. P. Barendregt, *The Lambda Calculus, its Syntax and Semantics*, 2nd ed. Amsterdam, The Netherlands: North-Holland Co., 1984.
4. A. Church, A formulation of the simple theory of types, *J. Symbol. Logic*, **5**: 56–68, 1940.
5. H. P. Barendregt, Lambda calculi with types, in S. Abramsky, D. Gabbay, and T. Maibaum (ed.), *Handbook of Logic in Computer Science, Volume 2, Background: Computational Structures*. Oxford, U.K.: Clarendon Press, 1992, pp. 117–309.
6. D. Prawitz, *Natural Deduction*. Stockholm, Sweden: Almqvist and Wiksell, 1965.
7. B. C. Pierce, *Types and Programming Languages*. Cambridge, MA: M.I.T. Press, 2002.

8. P. J. Landin, The next 700 programming languages, *Communicat. ACM*, **9** (3): 157–166, 1966.

9. F. Cardone and J. R. Hindley, Lambda-calculus and combinators in the 20th century, in D. Gabbay and J. Woods (eds.), *Handbook of the History of Logic, Volume 5*: Logic from Russell to Church. Amsterdam, Netherlands, Elsevier 2008, pp. 533–627.

10. H. P. Barendregt, The impact of the lambda calculus, *Bull. Symbol. Logic*, **3** (2): 181–215, 1997.

11. S. C. Kleene, Lambda-definability and recursiveness, *Duke Mathemat. J.*, **2**: 340–353, 1936.

12. S. C. Kleene, A theory of positive integers in formal logic, *Am. J. Mathemat.*, **57**: 153–173, 219–244, 1935.

13. A. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Mathematical Society*, **42** (2): 230–265, 1936. Correction: *Proc. London Mathematical Society*. **43**: 544–546, 1937.

14. A. Turing, Computability and λ-definability, *J. Symbol. Logic*, **2**: 153–163, 1937.

15. J. McCarthy, Recursive functions of symbolic expressions and their computation by machine, *Communicat. ACM*, **3**: 184–195, 1960.

16. G. D. Plotkin, Call-by-name, call-by-value and the λ-calculus, *Theoret. Comp. Sci.*, **1** (2): 125–159, 1975.

17. P. Seibel, *Practical Common Lisp*. Berkeley, CA: A press, 2005. Available: http://www.gigamonkeys.com/book/.

18. H. Abelson and G. J. Sussman, *Structure and Interpretation of Computer Programs*, 2nd ed. Cambridge, MA: M.I.T. Press, U.S.A., 1996.

19. J. R. Hindley, The principal type-scheme of an object in combinatory logic, *Trans. Am. Mathemat. Soc.*, **146**: 29–60, 1969.

20. R. Milner, A theory of type polymorphism in programming, *J. Comp. Sys. Scie.*, **17**: 348–375, 1978.

21. L. C. Paulson, *ML for the Working Programmer*, 2nd ed. Cambridge, U.K.: Cambridge University Press, 1996.

22. G. Cousineau and M. Mauny. *The Functional Approach to Programming*. Cambridge, U.K.: Cambridge University Press, 1998.

23. D. A. Turner, Miranda—a non-strict functional language with polymorphic types, in J. P. Jouannaud (ed.), *Functional Programming Languages and Computer Architectures,* volume 201 of *Lecture Notes in Computer Science*. Berlin: Springer Verlag, 1985, pp. 1–16.

24. S. L. Peyton Jones, *Haskell 98 Language and Libraries: the Revised Report*. Cambridge, U.K.: Cambridge University Press, 2003.

25. C. Wadsworth, Semantics and pragmatics of the Lambda-calculus, PhD thesis, Oxford, U.K.: University of Oxford, Programming Research Group, 1971.

26. D. Turner, A new implementation technique for applicative languages, *Software—Practice Exper.*, **9**: 31–49, 1979.

27. R. J. M. Hughes, The design and implementation of programming languages, PhD thesis, Oxford, U.K.: University of Oxford, 1984.

28. J. Backus, F. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Samuelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. *Revised Report on the Algorithmic Language Algol 60*. IFIP, 1963. Available: http://www.masswerk.at/algol60/report.htm.

29. P. J. Landin, A correspondence between ALGOL 60 and Church's lambda notation, *Communicat. ACM*, **8**: 89–101, 158–165, 1965.

30. D. S. Scott, Continuous lattices, in F. W. Lawvere (ed.), *Toposes, Algebraic Geometry and Logic,* volume 274 of *Lecture Notes in Mathematics*, Berlin: Springer-Verlag, 1972, pp. 97–136.

31. J. E. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics*. Cambridge, MA: M.I.T. Press, 1977.

32. W. Fontana, W. Günter, and L. W. Bass, Beyond digital naturalism, *Artif. Life*, **1/2**: 211–227, 1994.

33. M. Abadi and L. Cardelli, *A Theory of Objects*. Berlin: Springer-Verlag, 1996.

34. A. van Tonder, A lambda calculus for quantum computation, *S. I. A. M. J. Comput.*, **3**: 1109–1135, 2004.

35. H. Curry, Some properties of equality and implication in combinatory logic, *Ann. Mathemat., Series 2*, **35**: 849–860, 1934.

36. W. A. Howard, The formulae-as-types notion of construction, in J. P. Seldin and J. R. Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. New York: Academic Press, 1980, pp. 479–490.

37. N. G. de Bruijn, The mathematical language AUTOMATH, its usage and some of its extensions, in M. Laudet, D. Lacombe, and M. Schuetzenberger (eds.), *Symposium on Automatic Demonstration,* volume 125 of *Lecture Notes in Mathematics* Berlin: Springer Verlag, 1970, pp. 29–61. Also in Ref. 38.

38. R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer, eds. *Selected Papers on Automath,* volume 133 of *Studies in Logic and the Foundations of Mathematics*. Amsterdam, The Netherlands: North-Holland Co., 1994.

39. P. Martin-Löf, An intuitionistic theory of types: predicative part, in H. E. Rose and J. C. Shepherdson, (eds.), *Logic Colloquium '73,* volume 80 of *Studies in Logic and the Foundations of Mathematics*. Amsterdam, Netherlands: North-Holland Co., 1975, pp. 73–118.

40. P. Martin-Löf, Constructive mathematics and computer programming, in *Logic, Methodology and Philosophy of Science, VI*. Amsterdam, Netherlands: North-Holland Co., 1982, pp. 153–175.

41. P. Martin-Löf, Intuitionistic Type Theory. Studies in Proof Theory. Napoli Italy: Bibliopolis, 1984.

42. R. L. Constable, Constructive mathematics and automatic program writers, *Proc. IFIP Congress*, Amsterdam, Netherlands: North-Holland Co., 1971, pp. 229–233. Available: http://www.cs.cornell.edu/Info/Projects/.

43. R. L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

44. T. Coquand and G. Huet, The calculus of constructions, *Informat. Computat.*, **76**: 95–120, 1988.

45. G. Huet, G. Kahn, and C. Paulin-Mohring, *The Coq Proof Assistant, A Tutorial*. Paris, France: INRIA, 2006. Available: http://coq.inria.fr/V8.1/tutorial.html.

46. X. Leroy, *The Objective Caml system release 3.10*. Paris, France: INRIA, 2007. Available: http://caml.inria.fr/pub/docs/manual-ocaml/index.html.

Mariangiola Dezani-Ciancaglini
University of Turin
Turin, Italy

J. Roger Hindley
Swansea University
Swansea, Wales, United Kingdom

M

MIDDLEWARE FOR DISTRIBUTED SYSTEMS

MIDDLEWARE IS PART OF A BROAD SET OF INFORMATION TECHNOLOGY TRENDS

Middleware represents the confluence of two key areas of information technology (IT): distributed systems and advanced software engineering. Techniques for developing distributed systems focus on integrating many computing devices to act as a coordinated computational resource. Likewise, software engineering techniques for developing component-based systems focus on reducing software complexity by capturing successful patterns of interactions and creating reusable frameworks for integrating these components. Middleware is the area of specialization dealing with providing environments for developing systems that can be distributed effectively over a myriad of topologies, computing devices, and communication networks. It aims to provide developers of networked applications with the necessary platforms and tools to (1) formalize and coordinate how parts of applications are composed and how they interoperate and (2) monitor, enable, and validate the (re)configuration of resources to ensure appropriate application end-to-end quality of service (QoS), even in the face of failures or attacks.

During the past few decades, we have benefited from the commoditization of hardware (such as CPUs and storage devices), operating systems (such as UNIX and Windows), and networking elements (such as IP routers). More recently, the maturation of software engineering focused programming languages (such as Java and C++), operating environments (such as POSIX and Java Virtual Machines), and enabling fundamental middleware based on previous middleware R&D (such as CORBA, Enterprise Java Beans, and SOAP/Web services) are helping to commoditize many common-off-the-shelf (COTS) software components and architectural layers. The quality of COTS software has generally lagged behind hardware, and more facets of middleware are being conceived as the complexity of application requirements increases, which has yielded variations in maturity and capability across the layers needed to build working systems. Nonetheless, improvements in software frameworks (1), patterns (2,3), component models (4), and development processes (5) have encapsulated the knowledge that enables COTS software to be developed, combined, and used in an increasing number of real-world applications, such as e-commerce websites, avionics mission computing, command and control systems, financial services, and integrated distributed sensing, to name but a few.

Some notable successes in middleware for distributed systems include:

- Distributed Object Computing (DOC) middleware (6–10) (such as CORBA, Java RMI, SOAP), which provides a support base for objects that can be dispersed throughout a network, with clients invoking operations on remote target objects to achieve application goals. Much of the network-oriented code is tool generated using a form of interface definition language and compiler.
- Component middleware (11) (such as Enterprise Java Beans, the CORBA Component Model, and .NET), which is a successor to DOC approaches, focused on composing relatively autonomous, mixed functionality software elements that can be distributed or collocated throughout a wide range of networks and interconnects, while extending the focus and tool support toward lifecycle activities such as assembling, configuring, and deploying distributed applications.
- World Wide Web middleware standards (such as web servers, HTTP protocols, and web services frameworks), which enable easily connecting web browsers with web pages that can be designed as portals to powerful information systems.
- Grid computing (12) (such as Globus), which enables scientists and high-performance computing researchers to collaborate on grand challenge problems, such as global climate change modeling.

Within these middleware frameworks, a wide variety of services are made available off-the-shelf to simplify application development. Aggregations of simple, middleware-mediated interactions form the basis of large-scale distributed systems.

MIDDLEWARE ADDRESSES KEY CHALLENGES OF DEVELOPING DISTRIBUTED SYSTEMS

Middleware is an important class of technology that is helping to decrease the cycle time, level of effort, and complexity associated with developing high-quality, flexible, and interoperable distributed systems. Increasingly, these types of systems are developed using reusable software (middleware) component services, rather than being implemented entirely from scratch for each use. When implemented properly, middleware can help to:

- Shield developers of distributed systems from low-level, tedious, and error-prone platform details, such as socket-level network programming.
- Amortize software lifecycle costs by leveraging previous development expertise and capturing implementations of key patterns in reusable frameworks, rather than rebuilding them manually for each use.
- Provide a consistent set of higher-level network-oriented abstractions that are much closer to application and system requirements to simplify the development of distributed systems.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

- Provide a wide array of developer-oriented services, such as logging and security that have proven necessary to operate effectively in a networked environment.

Middleware was invented in an attempt to help simplify the software development of distributed computing systems, and bring those capabilities within the reach of many more developers than the few experts at the time who could master the complexities of these environments (7). Complex system integration requirements were not being met from either the *application perspective*, in which it was too hard and not reusable, or the *network or host operating system perspectives*, which were necessarily concerned with providing the communication and endsystem resource management layers, respectively.

Over the past decade, middleware has emerged as a set of software service layers that help to solve the problems specifically associated with heterogeneity and interoperability. It has also contributed considerably to better environments for building distributed systems and managing their decentralized resources securely and dependably. Consequently, one of the major trends driving industry involves moving toward a multi-layered architecture (applications, middleware, network, and operating system infrastructure) that is oriented around application composition from reusable components and away from the more traditional architecture, where applications were developed directly atop the network and operating system abstractions. This middleware-centric, multi-layered architecture descends directly from the adoption of a network-centric viewpoint brought about by the emergence of the Internet and the componentization and commoditization of hardware and software.

Successes with early, primitive middleware, such as message passing and remote procedure calls, led to more ambitious efforts and expansion of the scope of these middleware-oriented activities, so we now see a number of distinct layers taking shape within the middleware itself, as discussed in the following section.

MIDDLEWARE HAS A LAYERED STRUCTURE, JUST LIKE NETWORKING PROTOCOLS

Just as networking protocol stacks are decomposed into multiple layers, such as the physical, data-link, network, and transport, so too are middleware abstractions being decomposed into multiple layers, such as those shown in Fig. 1.

Below, we describe each of these middleware layers and outline some of the technologies in each layer that have matured and found widespread use in COTS platforms and products in recent years.

Host Infrastructure Middleware

Host infrastructure middleware leverages common patterns (3) and best practices to encapsulate and enhance native OS communication and concurrency mechanisms to create reusable network programming components, such as reactors, acceptor-connectors, monitor objects, active objects, and component configurators (13,14). These components abstract away the peculiarities of individual operating systems, and help eliminate many tedious, error-prone, and nonportable aspects of developing and maintaining networked applications via low-level OS programming APIs, such as Sockets or POSIX pthreads. Widely used examples of host infrastructure middleware include:

- The Sun Java Virtual Machine (JVM) (15), which provides a platform-independent way of executing code by abstracting the differences between operating systems and CPU architectures. A JVM is responsible for interpreting Java bytecode and for translating the bytecode into an action or operating system call. It is the JVM's responsibility to encapsulate platform details within the portable bytecode interface, so that applications are shielded from disparate operating systems and CPU architectures on which Java software runs.

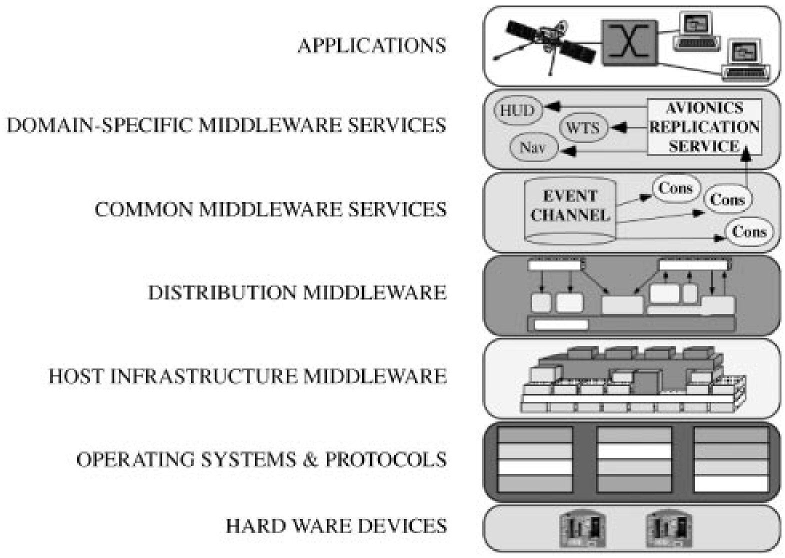

APPLICATIONS
DOMAIN-SPECIFIC MIDDLEWARE SERVICES
HUD
WTS
Nav
AVIONICS REPLICATION SERVICE
COMMON MIDDLEWARE SERVICES
EVENT CHANNEL
Cons
Cons
Cons
DISTRIBUTION MIDDLEWARE
HOST INFRASTRUCTURE MIDDLEWARE
OPERATING SYSTEMS & PROTOCOLS
HARD WARE DEVICES

Figure 1. Layers of middleware and surrounding context.

- .NET (16) is Microsoft's platform for XML Web services, which are designed to connect information, devices, and people in a common, yet customizable way. The common language runtime (CLR) is the host infrastructure middleware foundation for Microsoft's .NET services. The CLR is similar to Sun's JVM (i.e., it provides an execution environment that manages running code and simplifies software development via automatic memory management mechanisms, cross-language integration, interoperability with existing code and systems, simplified deployment, and a security system).
- The Adaptive Communication Environment (ACE) (13,14) is a highly portable toolkit written in C++ that encapsulates native OS network programming capabilities, such as connection establishment, event demultiplexing, interprocess communication, (de)marshaling, concurrency, and synchronization. The primary difference between ACE, JVMs, and the .NET CLR is that ACE is always a compiled interface rather than an interpreted bytecode interface, which removes another level of indirection and helps to optimize runtime performance.

Distribution Middleware

Distribution middleware defines higher-level distributed programming models whose reusable APIs and components automate and extend the native OS network programming capabilities encapsulated by host infrastructure middleware. Distribution middleware enables clients to program distributed systems much like stand-alone applications (i.e., by invoking operations on target objects without hard-coding dependencies on their location, programming language, OS platform, communication protocols and interconnects, and hardware). At the heart of distribution middleware are request brokers, such as:

- The OMG's Common Object Request Broker Architecture (CORBA) (6) and the CORBA Component Model (CCM) (17), which are open standards for distribution middleware that allows objects and components, respectively, to interoperate across networks regardless of the language in which they were written or the platform on which they are deployed. The OMG Real-time CORBA (RT-CORBA) specification (18) extends CORBA with features that allow real-time applications to reserve and manage CPU, memory, and networking resources.
- Sun's Java Remote Method Invocation (RMI) (10), which is distribution middleware that enables developers to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other JVMs, possibly on different hosts. RMI supports more sophisticated object interactions by using object serialization to marshal and unmarshal parameters as well as whole objects. This flexibility is made possible by Java's virtual machine architecture and is greatly simplified by using a single language.
- Microsoft's Distributed Component Object Model (DCOM) (19), which is distribution middleware that enables software components to communicate over a network via remote component instantiation and method invocations. Unlike CORBA and Java RMI, which run on many OSs, DCOM is implemented primarily on Windows.
- SOAP (20), which is an emerging distribution middleware technology based on a lightweight and simple XML-based protocol that allows applications to exchange structured and typed information on the Web. SOAP is designed to enable automated Web services based on a shared and open Web infrastructure. SOAP applications can be written in a wide range of programming languages, used in combination with a variety of Internet protocols and formats (such as HTTP, SMTP, and MIME), and can support a wide range of applications from messaging systems to RPC.

Common Middleware Services

Common middleware services augment distribution middleware by defining higher-level domain-independent services that allow application developers to concentrate on programming business logic, without the need to write the "plumbing" code required to develop distributed systems by using lower-level middleware directly. For example, application developers no longer need to write code that handles naming, transactional behavior, security, database connection, because common middleware service providers bundle these tasks into reusable components. Whereas distribution middleware focuses largely on connecting the parts in support of an object-oriented distributed programming model, common middleware services focus on allocating, scheduling, coordinating, and managing various resources end-to-end throughout a distributed system using a component programming and scripting model. Developers can reuse these component services to manage global resources and perform common distribution tasks that would otherwise be implemented in an ad hoc manner within each application. The form and content of these services will continue to evolve as the requirements on the applications being constructed expand. Examples of common middleware services include:

- The OMG's CORBA Common Object Services (CORBAservices) (21), which provide domain-independent interfaces and capabilities that can be used by many distributed systems. The OMG CORBAservices specifications define a wide variety of these services, including event notification, logging, multimedia streaming, persistence, security, global time, real-time scheduling, fault tolerance, concurrency control, and transactions.
- Sun's Enterprise Java Beans (EJB) technology (22), which allows developers to create n-tier distributed systems by linking a number of pre-built software services—called "beans"—without having to write much code from scratch. As EJB is built on top of Java technology, EJB service components can only be implemented using the Java language. The CCM

(17) defines a superset of EJB capabilities that can be implemented using all the programming languages supported by CORBA.

- Microsoft's .NET Web services (16), which complements the lower-level middleware .NET capabilities and allows developers to package application logic into components that are accessed using standard higher-level Internet protocols above the transport layer, such as HTTP. The .NET Web services combine aspects of component-based development and Web technologies. Like components, .NET Web services provide black-box functionality that can be described and reused without concern for how a service is implemented. Unlike traditional component technologies, however, .NET Web services are not accessed using the object model-specific protocols defined by DCOM, Java RMI, or CORBA. Instead, XML Web services are accessed using Web protocols and data formats, such as HTTP and XML, respectively.

Domain-Specific Middleware Services

Domain-specific middleware services are tailored to the requirements of particular domains, such as telecom, e-commerce, health care, process automation, or aerospace. Unlike the other three middleware layers, which provide broadly reusable "horizontal" mechanisms and services, domain-specific middleware services are targeted at vertical markets. From a COTS perspective, domain-specific services are the least mature of the middleware layers today. This immaturity is due partly to the historical lack of distribution middleware and common middleware service *standards*, which are needed to provide a stable base upon which to create domain-specific services. As they embody knowledge of a domain, however, domain-specific middleware services have the most potential to increase system quality and decrease the cycle time and effort required to develop particular types of networked applications. Examples of domain-specific middleware services include the following:

- The OMG has convened a number of Domain Task Forces that concentrate on standardizing domain-specific middleware services. These task forces vary from the *Electronic Commerce Domain Task Force*, whose charter is to define and promote the specification of OMG distributed object technologies for the development and use of electronic commerce and electronic market systems, to the *Life Science Research Domain Task Force,* who do similar work in the area of life science, maturing the OMG specifications to improve the quality and utility of software and information systems used in life sciences research. There are also OMG Domain Task Forces for the health-care, telecom, command and control, and process automation domains.
- The Siemens Medical Solutions Group has developed *syngo*. (See http://www.syngo.com), which is both an integrated collection of domain-specific middleware services as well as an open and dynamically extensible application server platform for medical imaging tasks and applications, including ultrasound, mammography, radiography, magnetic resonance, patient monitoring systems, and life support systems. The *syngo* middleware services allow health-care facilities to integrate diagnostic imaging and other radiological, cardiological, and hospital services via a black-box application template framework based on advanced patterns for communication, concurrency, and configuration for business and presentation logic supporting a common look and feel throughout the medical domain.

OVERARCHING BENEFITS OF MIDDLEWARE

The various layers of middleware described in the previous section provide essential capabilities for developing and deploying distributed systems. This section summarizes the benefits of middleware over traditional non-middleware approaches.

Growing Focus on Integration Rather than on Programming

This visible shift in focus is perhaps the major accomplishment of currently deployed middleware. Middleware originated because the problems relating to integration and construction by composing parts were not being met by *applications*, which at best were customized for a single use; *networks*, which were necessarily concerned with providing the communication layer; or *host operating systems*, which were focused primarily on a single, self-contained unit of resources. In contrast, middleware has a fundamental integration focus, which stems from incorporating the perspectives of both OSs and programming model concepts into organizing and controlling the composition of separately developed components across host boundaries. Every middleware technology has within it some type of request broker functionality that initiates and manages intercomponent interactions.

Distribution middleware, such as CORBA, Java RMI, or SOAP, makes it easy and straightforward to connect separate pieces of software together, largely independent of their location, connectivity mechanism, and the technology used to develop them. These capabilities allow middleware to amortize software lifecycle efforts by leveraging previous development expertise and reifying implementations of key patterns into more encompassing reusable frameworks and components. As middleware continues to mature and incorporate additional needed services, next-generation applications will increasingly be assembled by modeling, integrating, and scripting domain-specific and common service components, rather than by being programmed from scratch or requiring significant customization or augmentation to off-the-shelf component implementations.

Focus on End-to-End Support and Integration, Not Just Individual Components

There is now widespread recognition that effective development of large-scale distributed systems requires the use of COTS infrastructure and service components. Moreover, the usability of the resulting products depends heavily on the weaving of the properties of the whole as derived from its parts. In its most useful forms, middleware provides the end-to-end perspective extending across elements applicable to the network substrate, the platform OSs and system services, the programming system in which they are developed, the applications themselves, and the middleware that integrates all these elements together.

The Increased Viability of Open Systems Architectures and Open-Source Availability

By their very nature, distributed systems developed by composing separate components are more open than systems conceived and developed as monolithic entities. The focus on interfaces for integrating and controlling the component parts leads naturally to *standard* interfaces, which, in turn, yields the potential for multiple choices for component implementations and open engineering concepts. Standards organizations, such as the OMG, The Open Group, Grid Forum, and the W3C, have fostered the cooperative efforts needed to bring together groups of users and vendors to define domain-specific functionality that overlays open integrating architectures, forming a basis for industry-wide use of some software components. Once a common, open structure exists, it becomes feasible for a wide variety of participants to contribute to the off-the-shelf availability of additional parts needed to construct complete systems. As few companies today can afford significant investments in internally funded R&D, it is increasingly important for the IT industry to leverage externally funded R&D sources, such as government investment. In this context, standards-based middleware serves as a common platform to help concentrate the results of R&D efforts and ensure smooth transition conduits from research groups into production systems.

For example, research conducted under the DARPA Quorum, PCES, and ARMS programs focused heavily on CORBA open systems middleware. These programs yielded many results that transitioned into standardized service definitions and implementations for CORBA's real-time (9,18), fault-tolerant (23,24), and components (17) specifications and productization efforts. In this case, focused government R&D efforts leveraged their results by exporting them into, and combining them with, other on going public and private activities that also used a standards-based open middleware substrate. Before the viability of common middleware platforms, these same results would have been buried within a custom or proprietary system, serving only as the existence proof, not as the basis for incorporating into a larger whole.

Advanced Common Infrastructure Sustaining Continuous Innovation

Middleware supporting component integration and reuse is a key technology to help amortize software lifecycle costs by leveraging previous development expertise (e.g., component middleware helps to abstract commonly reused low-level OS concurrency and networking details away into higher-level, more easily used artifacts). Likewise, middleware also focus efforts to improve software quality and performance by combining aspects of a larger solution together (e.g., component middleware combines fault tolerance for domain-specific elements with real-time QoS properties).

When developers need not worry as much about low-level details, they are freed to focus on more strategic, larger scope, application-centric specializations concerns. Ultimately, this higher-level focus will result in software-intensive distributed system components that apply reusable middleware to get smaller, faster, cheaper, and better at a predictable pace, just as computing and networking hardware do today, which, in turn, will enable the next generation of better and cheaper approaches to what are now carefully crafted custom solutions, which are often inflexible and proprietary. The result will be a new technological paradigm where developers can leverage *frequently used common components*, which come with steady innovation cycles resulting from a multi-user basis, in conjunction with *custom domain-specific components*, which allow appropriate mixing of multi-user low cost and custom development for competitive advantage.

KEY CHALLENGES AND OPPORTUNITIES FOR NEXT-GENERATION MIDDLEWARE

This section presents some of the challenges and opportunities for next-generation middleware. One such challenge is in supporting new trends toward distributed "systems of systems," which include many interdependent levels, such as network/bus interconnects, embedded local and geographically distant remote endsystems, and multiple layers of common and domain-specific middleware. The desirable properties of these systems of systems, both individually and as a whole, include predictability, controllability, and adaptability of operating characteristics with respect to such features as time, quantity of information, accuracy, confidence, and synchronization. All these issues become highly volatile in systems of systems, because of the dynamic interplay of the many interconnected parts. These parts are often constructed in a similar way from smaller parts.

Many COTS middleware platforms have traditionally expected static connectivity, reliable communication channels, and relatively high bandwidth. Significant challenges remain, however, to design, optimize, and apply middleware for more flexible network environments, such as self-organizing peer-to-peer (P2P) networks, mobile settings, and highly resource-constrained sensor networks. For example, hiding network topologies and other deployment details from networked applications becomes harder (and often undesirable) in wireless sensor networks because applications and middleware often need to adapt according to changes in location, connectivity, bandwidth, and battery power. Concerted R&D efforts are therefore essential to devise new middleware solutions and capabilities that can fulfill the requirements of these emerging network technologies and next-generation applications.

There are significant limitations today with regard to building the types of large-scale complex distributed systems outlined above that have increasingly more stringent requirements and more volatile environments. We are also discovering that more things need to be integrated over conditions that more closely resemble a dynamically changing Internet than they do a stable backplane. One problem is that the playing field is changing constantly, in terms of both resources and expectations. We no longer have the luxury of being able to design systems to perform highly specific functions and then expect them to have life cycles of 20 years with minimal change. In fact, we more routinely expect systems to behave differently under different conditions and complain when they just as routinely do not. These changes have raised a number of issues, such as end-to-end-oriented adaptive QoS, and construction of systems by composing off-the-shelf parts, many of which have promising solutions involving significant new middleware-based capabilities and services.

To address the many competing design forces and runtime QoS demands, a comprehensive methodology and environment is required to dependably compose large, complex, interoperable distributed systems from reusable components. Moreover, the components themselves must be sensitive to the environments in which they are packaged. Ultimately, what is desired is to take components that are built independently by different organizations at different times and assemble them to create a complete system. In the longer run, this complete system becomes a component embedded in still larger systems of systems. Given the complexity of this undertaking, various tools and techniques are needed to configure and reconfigure these systems so they can adapt to a wider variety of situations.

An essential part of what is needed to build the type of systems outlined above is the integration and extension of ideas that have been found traditionally in network management, data management, distributed operating systems, and object-oriented programming languages. But the goal for next-generation middleware is not simply to build a better network or better security in isolation, but rather to pull these capabilities together and deliver them to applications in ways that enable them to realize this model of adaptive behavior with tradeoffs between the various QoS attributes. The payoff will be reusable middleware that significantly simplifies the building of applications for systems of systems environments. The remainder of this section describes points of emphasis that are embedded within that challenge to achieve the desired payoff:

Reducing the Cost and Increasing the Interoperability of Using Heterogeneous Environments

Today, it is still the case that it costs quite a bit more in complexity and effort to operate in a truly heterogeneous environment, although nowhere near what it used to cost. Although it is now relatively easy to pull together distributed systems in heterogeneous environments, there remain substantial recurring downstream costs, particularly for complex and long-lived distributed systems of systems. Although homogeneous environments are simpler to develop and operate, they often do not reflect the long-run market reality, and they tend to leave open more avenues for catastrophic failure. We must, therefore, remove the remaining impediments associated with integrating and interoperating among systems composed from heterogeneous components. Much progress has been made in this area, although at the host infrastructure middleware level more needs to be done to shield developers and end users from the accidental complexities of heterogeneous platforms and environments. In addition, interoperability concerns have largely focused on data interoperability and invocation interoperability. Little work has focused on mechanisms for controlling the overall behavior of integrated systems, which is needed to provide "control interoperability." There are requirements for interoperable distributed control capabilities, perhaps initially as increased flexibility in externally controlling individual resources, after which approaches can be developed to aggregate these into acceptable global behavior.

Dynamic and Adaptive QoS Management

It is important to avoid "all or nothing" point solutions. Systems today often work well as long as they receive all the resources for which they were designed in a timely fashion, but fail completely under the slightest anomaly. There is little flexibility in their behavior (i.e., most of the adaptation is pushed to end users or administrators). Instead of hard failure or indefinite waiting, what is required is either *reconfiguration* to reacquire the needed resources automatically or *graceful degradation* if they are not available. Reconfiguration and operating under less than optimal conditions both have two points of focus: individual and aggregate behavior. To manage the increasingly stringent QoS demands of next-generation applications operating under changing conditions, middleware is becoming more adaptive and reflective. *Adaptive middleware* (25) is software whose functional and QoS-related properties can be modified either (1) *statically* (e.g., to reduce footprint, leverage capabilities that exist in specific platforms, enable functional subsetting, and minimize hardware/software infrastructure dependencies or (2) *dynamically* (e.g., to optimize system responses to changing environments or requirements, such as changing component interconnections, power levels, CPU/network bandwidth, latency/jitter, and dependability needs.

In mission-critical distributed systems, adaptive middleware must make such modifications dependably (i.e., while meeting stringent end-to-end QoS requirements). Reflective middleware (26) techniques make the internal organization of systems, as well as the mechanisms used in their construction, both visible and manipulable for middleware and application programs to inspect and modify at run time. Thus, reflective middleware supports more advanced adaptive behavior and more dynamic strategies keyed to current circumstances (i.e., necessary adaptations can be performed autonomously based on conditions within the system, in the system's environment, or in system QoS policies defined by end users.

Advanced System Engineering Tools

Advanced middleware by itself will not deliver the capabilities envisioned for next-generation distributed systems. We must also advance the state of the system engineering tools that come with these advanced environments used to build and evaluate large-scale mission-critical distributed systems. This area of research specifically addresses the immediate need for system engineering tools to augment advanced middleware solutions. A sample of such tools might include:

- *Design time tools*, to assist system developers in understanding their designs, in an effort to avoid costly changes after systems are already in place (which is partially obviated by the late binding for some QoS decisions referenced earlier).
- *Interactive tuning tools*, to overcome the challenges associated with the need for individual pieces of the system to work together in a seamless manner.
- *Composability tools*, to analyze resulting QoS from combining two or more individual components.
- *Modeling tools for developing system models* as adjunct means (both online and offline) to monitor and understand resource management, in order to reduce the costs associated with trial and error.
- *Debugging tools*, to address inevitable problems that develop at run time.

Reliability, Trust, Validation, and Assurance

The dynamically changing behaviors we envision for next-generation middleware-mediated systems of systems are quite different from what we currently build, use, and have gained some degrees of confidence in. Considerable effort must, therefore, be focused on validating the correct functioning of the adaptive behavior and on understanding the properties of large-scale systems that try to change their behavior according to their own assessment of current conditions before they can be deployed. But even before that, long-standing issues of adequate reliability and trust factored into our methodologies and designs using off-the-shelf components have not reached full maturity and common usage, and must therefore continue to improve. The current strategies organized around anticipation of long lifecycles with minimal change and exhaustive test case analysis are clearly inadequate for next-generation dynamic distributed systems of systems with stringent QoS requirements.

TAKING STOCK OF TECHNICAL PROGRESS ON MIDDLEWARE FOR DISTRIBUTED SYSTEMS

The increased maturation of, and reliance on, middleware for distributed systems stems from two fundamental trends that influence the way we conceive and construct new computing and information systems. The first is that IT of all forms is becoming highly commoditized (i.e., hardware and software artifacts are getting faster, cheaper, and better at a relatively predictable rate). The second is the growing acceptance of a network-centric paradigm, where distributed systems with a range of QoS needs are constructed by integrating separate components connected by various forms of reusable communication services. The nature of the interconnection ranges from the very small and tightly coupled, such as embedded avionics mission computing systems, to the very large and loosely coupled, such as global telecommunications systems.

The interplay of these two trends has yielded new software architectural concepts and services embodied by middleware. The success of middleware has added new layers of infrastructure software to the familiar OS, programming language, networking, and database offerings of the previous generation. These layers are interposed between applications and commonly available hardware and software infrastructure to make it feasible, easier, and more cost effective to develop and evolve systems via reusable software. The past decade has yielded significant progress in middleware, which has stemmed, in large part, from the following:

Years of Iteration, Refinement, and Successful Use. The use of middleware is not new (27,28). Middleware concepts emerged alongside experimentation with the early Internet (and even its predecessor the ARPAnet), and middleware systems have been continuously operational since the mid-1980s. Over that period of time, the ideas, designs, and (most importantly) the software that incarnates those ideas have had a chance to be tried and refined (for those that worked), and discarded or redirected (for those that did not). This iterative technology development process takes a good deal of time to get right and be accepted by user communities and a good deal of patience to stay the course. When this process is successful, it often results in *standards* that codify the boundaries, and *patterns and frameworks* that reify the knowledge of how to apply these technologies, as described in the following subsections.

The Maturation of Open Standards and Open Source. Over the past decade, middleware standards have been established and have matured considerably, particularly with respect to mission-critical distributed systems that possess stringent QoS requirements. For instance, the OMG has adopted the following specifications in recent years: (1) *Minimum CORBA* (29), which removes nonessential features from the full OMG CORBA specification to reduce footprint so that CORBA can be used in memory-constrained embedded systems; (2) *Real-time CORBA* (18), which includes features that enable applications to reserve and manage network, CPU, and memory resources more predictably end-to-end; (3) *CORBA Messaging* (30), which exports additional QoS policies, such as timeouts, request priorities, and queuing disciplines, to applications; and (4) *Fault-tolerant CORBA* (23), which uses entity redundancy of objects to support replication, fault detection, and failure recovery. Robust implementations of these CORBA capabilities and services are now available from multiple suppliers, many of whom have adopted open-source business models. Moreover, the scope of open systems is extending to an even wider range of applications with the advent of emerging standards, such as the Real-Time Specification for Java (31), and the Distributed Real-Time Specification for Java (32).

The Dissemination of Patterns and Frameworks. Also during the past decade, a substantial amount of R&D effort has focused on developing *patterns* and *frameworks* as a means to promote the transition and reuse of successful middleware technology. Patterns capture successful solutions to commonly occurring software problems that occur in a particular context (2,3). Patterns can simplify the design, construction, and performance tuning of middleware and applications by codifying the accumulated expertise of developers who have confronted similar problems before. Patterns also raise the level of discourse in describing software design and programming activities. Frameworks are concrete realizations of groups of related patterns (1). Well-designed frameworks reify patterns in terms of functionality provided by the middleware itself, as well as functionality provided by an application. A framework also integrates various approaches to problems where there are no a priori, context-independent, optimal solutions. Middleware frameworks (14) can include strategized selection and optimization patterns so that multiple independently developed capabilities can be integrated and configured automatically to meet the functional and QoS requirements of particular applications.

In the brief space of this article, we can only summarize and lend perspective to the many activities, past and present, that contribute to making middleware technology an area of exciting current development, along with considerable opportunity and unsolved challenging R&D problems. We have provided references to other sources to obtain additional information about ongoing activities in this area. We have also provided a more detailed discussion and organization for a collection of activities that we believe represent the most promising future directions for middleware. The ultimate goals of these activities are to:

1. Reliably and repeatably construct and compose distributed systems that can meet and adapt to more diverse, changing requirements/environments, and
2. Enable the affordable construction and composition of the large numbers of these systems that society will demand, each precisely tailored to specific domains.

To accomplish these goals, we must overcome not only the technical challenges, but also the educational and transitional challenges, and eventually master and simplify the immense complexity associated with these environments, as we integrate an ever-growing number of hardware and software components together via advanced middleware.

BIBLIOGRAPHY

1. R. Johnson, Frameworks = Patterns + Components, *CACM*, **40**(10), 1997.
2. E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, MA: Addison–Wesley, 1995.
3. D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, New York: Wiley, 2000.
4. C. Szyperski, *Component Software– Beyond Object-Oriented Programming,* Reading, M.A.: Addison-Wesley, 1998.
5. I. Jacobson, G. Booch, J. Rumbaugh, *Unified Software Development Process*, Reading, M.A.: Addison-Wesley, 1999.
6. Object Management Group, The Common Object Request Broker: Architecture and Specification Revision 3.0.2, OMG Technical Document, 2002.
7. R. Schantz, R. Thomas, and G. Bono, The architecture of the cronus distributed operating system, *Proceedings of the 6th IEEE International Conference on Distributed Computing Systems*, Cambridge, M.A., 1986.
8. R. Gurwitz, M. Dean and R. Schantz, Programming support in the cronus distributed operating system, *Proceedings of the 6th IEEE International Conference on Distributed Computing Systems*, Cambridge, MA, 1986.
9. D. Schmidt, D. Levine, and S. Mungee, The Design and Performance of the TAO Real-Time Object Request Broker, *Computer Communications Special Issue on Building Quality of Service into Distributed Systems,* **21** (4), 1998.
10. A. Wollrath, R. Riggs, J. Waldo, A distributed object model for the java system, *USENIX Computing Systems*, **9** (4), 1996.
11. G. Heineman and B. Councill, *Component-Based Software Engineering: Putting the Pieces Together*, Reading, MA: Addison-Wesley, 2001.
12. I. Foster, and K. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann, 1999.
13. D. Schmidt, S. Huston, *C++ Network Programming Volume 1: Mastering Complexity with ACE and Patterns*, Reading, M.A.: Addison-Wesley, 2002.
14. D. Schmidt, S. Huston, *C++ Network Programming Volume 2: Systematic Reuse with ACE and Frameworks*, Reading, M.A.: Addison-Wesley, 2003.
15. T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, Reading, MA: Addison-Wesley, 1997.
16. T. Thai and H. Lam, *.NET Framework Essentials*, Cambridge, M.A.: O'Reilly, 2001.
17. Object Management Group, CORBA Components, OMG Document formal/2002-06-65.
18. Object Management Group, Real-Time CORBA, OMG Document formal/02-08-02, 2002.
19. D. Box, *Essential COM*, Reading, MA: Addison-Wesley, 1997.
20. J. Snell, K. MacLeod, *Programming Web Applications with SOAP*, Cambridge, M.A.: O'Reilly, 2001.
21. Object Management Group, CORBAServices: Common Object Service Specification, OMG Document formal/98-12-31, edition, 1998.
22. A. Thomas, Enterprise Java Beans Technology, 1998. Available: http://java.sun.com/products/ejb/white_paper.html.
23. Object Management Group, *Fault Tolerance CORBA Using Entity Redundancy RFP*, OMG Document orbos/98-04-01 edition, 1998.
24. M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. Sanders, B. Bakken, M. Berman, D. Karr, R. Schantz, AQuA: An adaptive architecture that provides dependable distributed objects, *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 245–253.
25. J. Loyall, J. Gossett, C. Gill, R. Schantz, J. Zinky, P. Pal, R. Shapiro, C. Rodrigues, M. Atighetchi, D. Karr, Comparing and contrasting adaptive middleware support in wide-area and embedded distributed object applications. *Proceedings of the 21st IEEE International Conference on Distributed Computing System*, Phoenix, AZ, 2001.

26. G.S. Blair, F. Costa, G. Coulson, H. Duran, et al., The design of a resource-aware reflective middleware architecture, *Proceedings of the 2nd International Conference on Meta-Level Architectures and Reflection*, St.-Malo, France, Springer-Verlag, LNCS, Vol. **1616**, 1999.

27. R. Schantz, *BBN and the Defense Advanced Research Projects Agency*, Prepared as a Case Study for America's Basic Research: Prosperity Through Discovery, A Policy Statement by the Research and Policy Committee of the Committee for Economic Development (CED), June 1998, Available: http://www.dist-systems.bbn.com/papers/1998/CaseStudy.

28. P. Bernstein, Middleware, A model for distributed system service, *CACM*, **39**: 2, 1996.

29. Object Management Group, Minimum CORBA, OMG Document formal/00-10-59, 2000.

30. Object Management Group, CORBA Messaging Specification, OMG Document orbos/98-05-05, 1998.

31. G. Bollella and J. Gosling, The real-time specification for java,*Computer*, June 2000.

32. D. Jensen, Distributed Real-Time Specification for Java, 2000 Available: java.sun.com/aboutJava/communityprocess/jsr/jsr_050_drt.html.

FURTHER READING

BBN, Quality Objects Toolkit for Adaptive Distributed Applications. Available: http://quo.bbn.com.

Sun Microsystems, Jini Connection Technology. Available: http://www.sun.com/jini/index.html.

B. Sabata, S. Chatterjee, M. Davis, J. Sydir, T. Lawrence, Taxonomy for QoS Specifications, *Proceedings of the Workshop on Object-oriented Real-time Dependable Systems (WORDS 97)*, February 1997.

RICHARD E. SCHANTZ
BBN Technologies
Cambridge, Massachusetts
DOUGLAS C. SCHMIDT
Vanderbilt University
Nashville, Tennessee

OPTIMIZING COMPILERS

INTRODUCTION AND MOTIVATION

Optimization is achieved by analyzing a given input program and applying a variety of code transformations to it. Which optimizing steps and code transformations may be applied depends on the semantics of the source programming language and the results of the analyses. Optimization is not performed on the source code but on a compiler-internal program representation; the form of the latter can significantly influence both the optimizations that can be applied as well as their effect. Some optimizations are essential on many systems, especially RISC architectures, without which programs would be very inefficient. There is no guarantee that an optimized program will execute faster, or that more extensive optimization will lead to an improvement in performance; however, improvement will typically occur, especially for large, complex programs with extensive datasets, in short, for just those programs in which manual analysis is difficult to carry out. Moreover, any manual analysis and modification of intermediate code carries with it the danger of accidentally changing the program's semantics. In contrast, the code transformations applied by an optimizing compiler are guaranteed to leave the semantics unchanged. Moreover, significant time savings can often be obtained by an optimizing compiler at very little cost, because the optimization phase of a compiler is typically executed only once and the program may be executed very many times. (Strictly speaking, the term "optimization" is, of course, a misnomer because it is almost guaranteed that the resulting object code is not optimal; however, for historical reasons, we will use the term "optimization" in its intended meaning of *program improvement*, with this caveat.)

There are many design choices facing the developers of an optimizing compiler. The ability of the system to improve a large variety of input programs may depend on the accuracy of the analysis performed. Yet the required analyses can be highly expensive and difficult to implement. Modern compilers typically perform optimizations in multiple phases, each with a distinct purpose. Typically, certain sequences of analyses and transformations are combined into an *optimization strategy* that is accessible via a compiler switch. Thus, the user may choose between several predefined collections of optimizations when the compiler is invoked.

Not all languages are created equal with regard to optimization. For example, the potential for aliasing of variables in a code can have a major impact on the outcome of optimization. As each program change requires us to reason about the state a program will be in at a given point during execution, if two variables may share the same memory location at some point, the analyses must consider how the desired transformation will affect each of them, which may severely limit the potential for optimizing a program, especially when translating programs written in languages that permit uncontrolled use of pointers (if these features are extensively exploited in the code).

BASIC OPTIMIZATIONS

A variety of well-known optimizations are useful for improving code written in many different programming languages and for execution on most modern architectures. As such, they are widely implemented. They include optimizations to eliminate statements that will never be executed (useless code); to replace certain operations by faster, equivalent ones (e.g., strength reduction); and to eliminate redundant computations, possibly by moving statements in the code to a new location that permits the results to be used in multiple locations subsequently. Examples of this last optimization include hoisting code from loops, so that it is executed just once, rather than during each loop iteration, and partial redundancy elimination, variants of which attempt to move statements so that an expression is computed once only in a given execution path. Another popular optimization called constant propagation attempts to determine all variable references that have a constant value no matter what execution path is taken, and to replace those references with that value, which, in turn, may enable the application of further optimizations.

These optimizations are generally known as *scalar optimizations*, because they are applied to scalar variables without regard to the internal structuring of a program's complex data objects, and thus consider the individual elements of arrays to be distinct objects. They may be applied to small program regions in the form of so-called peephole optimizations, but also to entire procedures or even beyond. In order to perform them on a given program, it is necessary to analyze and represent the structure of each procedure being translated in such a way that all of its possible execution paths are identified. The implementation must then efficiently identify all points in the code where a given optimization is applicable and perform the specified translation.

Data Flow Analysis

Collectively, the analysis required to perform this work is known as *data flow analysis* (DFA), which studies the flow of values of data objects throughout a program. The analysis that determines the structure of a program is known as control flow analysis (CFA). Intraprocedural CFA constructs a flowgraph (or program graph), a directed graph with a single entry node, whose nodes represent the procedure's basic blocks and whose edges represent transfers of control between basic blocks. Basic blocks are maximal length sequences of statements that can only be entered via the first and only be exited via the last statement; they partition a procedure. The single-exit property can be enforced on the flowgraph if it is needed. Loops, including

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

implicitly programmed loops, can be identified in the flowgraph and a variety of node orderings defined and computed that enable data flow problems to be efficiently applied to it. Although scalar optimizations can be performed easily within basic blocks, their application to an entire procedure gives them considerably greater power.

For example, consider the following fragment of pseudo code.

```
x := A[j] ;
z := 1;
if x < 4
   then z := x;
fi
c := z + x
```

Figure 1 shows a flowgraph representing the flow of control in these statements. Each node in the graph corresponds to one of its basic blocks. The edges represent the flow of control between them.

Many data flow optimizations are closely related to the so-called use-definition (UD) and definition-use (DU) chains. (In fact, below we introduce one way of representing a procedure internally that makes DU chains explicit in the code.) A UD chain links a use of a variable to the set of all definitions of that variable that may reach it (i.e., all possible sources of the value that will be used); a DU chain links a definition of a variable to all of its possible uses. For example, the value of x defined in the first statement of the above code is used three times subsequently (on lines 3, 4, and 6 of the text), so the DU chains will link the definition to each of these uses. UD analysis will link each of the uses to this definition. The interactions of a basic block with the remainder of a program may be modeled by identifying its *outward-exposed* variable definitions, those definitions that may have uses in other basic blocks, and its outward-exposed uses, those variable uses that are defined in other basic blocks and may be used by this block. For example, basic block B3 in Fig. 1(a) has an outward-exposed definition of variable c and outward-exposed uses of variables x and y. This information has many applications.

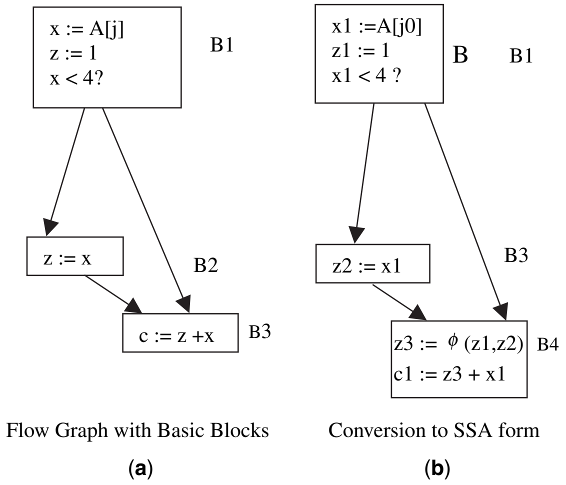


Figure 1. Representing control flow in a procedure.

Strategies for register allocation, for instance, may benefit from knowing which variables are live at the exit of a basic block. A variable is live if it may be used subsequently (i.e., if there is a path from the corresponding node in the flowgraph to a node with an outward-exposed use of that variable) and the path is definition-free for that variable. A variable that is not live should not be kept in a register. UD chains provide the required information. For example, x and z are live at the end of basic block B1 in Fig. 1(a), because they are subsequently used. The UD chains will link these uses of x and z to the definitions in B1.

Data Flow Problems

The task of determining all the points in the program where a specific optimization is applicable, or where a specific property holds, is known as a data flow problem. For example, the live variables problem may be solved by traversing a single-exit flowgraph, starting with its unique exit node, and propagating information on outward-exposed uses to nodes that precede them on paths from the start node. Any variable definition for which there is a subsequent outward-exposed use is live. The available expressions problem computes the expressions that are available on entry to a basic block. It requires a traversal of the flowgraph that begins with the start node and then visits subsequent nodes after all nodes that may precede them have been visited. The analysis gathers the expressions that are computed in a basic block and are outward-exposed (i.e., the variables used in the computation have not been redefined) and combines them with the expressions reaching this basic block that are preserved (i.e., the variables used in that computation have not been redefined). If there is an outward-exposed use of the same expression within the associated code, the resulting set of expressions will be available in the successor nodes of the callgraph. If there is an outward-exposed use of the same expression within the associated code, the computation is redundant and may be eliminated. For example, if we assume that Fig. 1(a) is part of a larger flowgraph, then the expression z+x will be available to a successor node of B3 if and only if neither of these variables have been redefined in the interim (*and* all paths to the node pass through B3). Any computation of z+x in such nodes is redundant and may be eliminated. There are a variety of approaches to detect and handle redundant computations in practice.

Data flow problems may be classified as *top-down* or *bottom-up* problems, depending on the order of information propagation. Live variables analysis is a bottom-up problem, and the available expressions computation is an example of a top-down problem. They may also be classified as *existence* problems or *all* problems. In the former case, the task is to find a path satisfying a given property; in the latter, the property must hold for all paths. For a variable to be live, we only need identify one path with an outward-exposed use that reaches the node in question; so it is an existence problem. In contrast, the available expressions problem is an all problem, because the expression is only available (and hence redundant) at a given point in the code if it is available no matter what execution path was taken to reach that point.

Techniques for Solving Data Flow Problems

The practical solution of data flow problems was enhanced by a strategy that exploits the fact that they can be modeled by monotone data flow systems (MDSs). A semilattice is used to represent data flow information and monotone functions model the way data flow information is modified when control passes through a basic block in the procedure. Virtually all data flow problems may be elegantly formulated in terms of MDSs; the associated solution mechanism is a simple iterative fixpoint algorithm whose implementation is efficient and very reasonable in its memory requirements for many data flow problems. In order to apply the iterative algorithm to a given data flow problem, this problem must be classified as either top down or bottom up; it must be formulated in terms of information to be computed at each node of the flowgraph, which is typically a set of variables, variable references, or statements. The effect a basic block has on the information computed for the nodes preceding it (or succeeding it, in the case of bottom-up problems) must be specified. Finally, it must be specified how information sets are merged when two paths meet in the flowgraph. Then the algorithm will begin at either the start (top-down) or exit (bottom-up) node of the flowgraph, compute the specified information, and propagate that information to the nodes succeeding or preceding it, respectively. This information will be updated and the resulting set propagated in the same direction. The presence of loops and, therefore, cycles in the flowgraph necessitates that this process be repeated on the current data until no more changes in information are observed. For example, the live variables problem begins by computing the set of variables in a basic block that have outward-exposed uses, which are passed to preceding nodes in the flowgraph, where all those that remain outward-exposed are augmented with those that have outward-exposed uses in that basic block and, in turn, are propagated. More powerful strategies using so-called interval analysis exist; they require considerably more implementation effort, but execute faster and make it easier to incrementally update data flow information. Both the iterative algorithm and the interval analysis expect a flowgraph to represent a procedure.

Typical Data Flow Optimizations

Among the many other optimizations that have been defined and are applied to basic blocks and procedures is copy propagation, which replaces a variable with one that is equivalent to it. Constant folding evaluates expressions at compile time when their operands are known to be constant, and is especially applied when they are integers. Common subexpression elimination and value numbering are two subtly different techniques that support the identification and removal of computations that are unnecessary because the values have already been determined. Partial redundancy elimination is a more powerful approach to handling this problem and is increasingly preferred over these alternatives. Bounds checking elimination is important for programming languages where it must be tested whether array references are within the defined range of index values. Loop-invariant code motion finds computations that produce the same result each time a loop is executed and moves them out of the loop. Some optimizations are performed multiple times, which is especially true with respect to dead code elimination, because during the course of applying transformations, the compiler may introduce code that is not needed and that can be subsequently removed. The compiler must also be able to simplify algebraic and logical expressions, both to reduce the work of computing them, but also to facilitate the implementation of those optimizations that require them to be compared or evaluated.

New Approaches to Data Flow Analysis

A relatively new approach to performing optimizations on a procedure requires the prior conversion of the program to a static single assignment (SSA) form. This representation makes the DU chains explicit by requiring that variables be defined once only. It is achieved by renaming variables and by introducing so-called join (ϕ) functions at any point in the code where two or more definitions may reach a single use. It is possible to optimize this representation to minimize the number of join functions required. In Fig. 1(b), we show the SSA form of the flowgraph given in standard form in Fig. 1(a). It uses a ϕ function in basic block B3 to indicate that the value of z may come from either B1 or B2.

The SSA representation makes several optimizations, including strength reduction, partial redundancy elimination, and constant propagation, easier to specify and more efficient to implement in comparison with the techniques described above; its use is growing. However, it is relatively hard to perform alias analysis on this representation. A compiler is likely to convert code to SSA form only temporarily during the overall optimization process.

INTERPROCEDURAL OPTIMIZATIONS

The strategies described above are typically applied to the individual procedures of a program. However, it is also possible to optimize code across procedure boundaries. The growing use of structured programming techniques has led to the increased modularization of programs, which may consist of a large number of relatively small procedures; thus, it has become important to consider how to improve code in a way that takes procedure and function invocations into account. Interprocedural analysis (IPA) is the name given to techniques that gather information about the calling relationships between different program units, and optimizations based on them are called interprocedural optimizations.

Interprocedural Analysis

Just as the flowgraph is the basis for optimizations within a procedure, so the callgraph is the foundation for interprocedural optimizations. This data structure is designed to represent the relationships between the procedures and functions of a program. The nodes of the callgraph represent the individual program units; there is a directed edge between nodes if and only if the procedure corresponding to the source invokes the procedure corresponding to the sink at least once. A traditional callgraph has only one edge between a pair of nodes per direction, no matter how often the sink procedure is called, and thus provides less information about the paths that are taken through a program than does a (procedure's) flowgraph; alternative multigraph representations of the calling sequences represent each invocation separately and thus provide additional information. In order to perform interprocedural optimizations, the actual arguments at each callsite must also be saved. It is possible to augment this information with details that help identify those sequences of calls, or call chains, that may occur at run time.

The only tricky part of constructing a call graph occurs when procedures are passed as arguments to other procedures. An example of this is given in Fig. 2 and is based on one given by Muchnik. Here, procedure m invokes g twice, but it is not reflected directly in the call graph where there is one edge from m to *g*. In turn, procedure g makes two calls, one to procedure j and one to procedure p. But p is a variable: Our analysis of the code shows that it will be associated with h, so that g actually calls j and h. Edges are inserted accordingly. Procedure h calls i, and thus we add an edge from caller to callee. Procedure j calls a, which again is not an actual procedure, so we must determine which values it may assume at run time. Our code inspection shows that it may only be associated with i, so we must insert an edge from j to i in the call graph. We now determine that procedure i invokes g and add a corresponding edge to the call graph in order to complete it. Note that nodes in the call graph correspond to actual procedures and procedure parameters do not occur in it.

```
procedure m ( )
begin
    call g( )
    call g( )
end

procedure g( )
begin
 procedure   p
    p:= h
    call p( )
    call j( i )
end

procedure h ( )
begin
    call( i )
end

procedure i( )
begin
    call g( )
end

procedure j(a)
 Procedure    a
begin
    call a( )
end
```

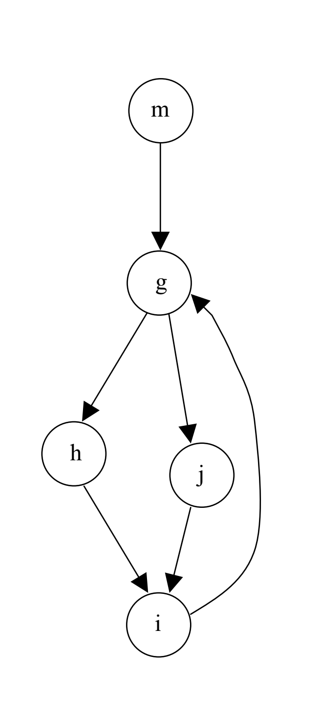


Figure 2. Program with procedure variables and resulting call graph.

Strategies for Interprocedural Optimization

A compiler generally translates an input code one procedure at a time (known as separate compilation). Strategies for applying optimizations interprocedurally must take this fact into account, and thus they are often separated into analysis phases that identify call sites and build the callgraph and a transformation phase that occurs after some analyses have been applied to each procedure individually, possibly spread among several phases of compilation to support other kinds of optimization, such as that described in the Array and Loop Optimizations Section, or possibly only shortly before the final code generation.

Purpose of Interprocedural Optimization

One reason that interprocedural analysis might produce superior results is that, without it, worst-case assumptions must be made with respect to the impact of procedure calls during (intraprocedural) DFA: It must be assumed that the call modifies every variable that is visible to both it and the calling procedure, including every global variable. Thus IPA can be used to improve the results of standard DFA. It may also be used explicitly to improve code that spans multiple procedures. Before we give examples of more sophisticated uses of IPA, we describe some of the simpler optimizations. At the end of this section, we consider the limitations of IPA.

Reducing Runtime Cost of Procedure Invocations

There are some interprocedural optimizations that may be implemented and carried out with relatively little effort. Each time a procedure is called at run time, information essential to the execution of the caller must be saved, the execution environment for the callee must be set up and control transferred. Upon its termination, the caller's environment must be restored, which incurs nontrivial overheads. Several optimizations are able to reduce these overheads, either directly or by reducing the number of procedure calls. One popular optimization in the latter category is known as procedure inlining: It replaces a procedure call in the code by the suitably adapted body of the procedure. If the call is made within a loop, this may sometimes lead to considerable savings. Unfortunately, however, it has proved to be very hard to come up with a good strategy for performing this optimization in general. If applied too frequently, the size of the object code may increase substantially, introducing other overheads. Strategies to control its application may take into account the frequency with which a call is expected to be executed, the length of the procedure's code, its location relative to the program hotspots, or some combination of these.

Advanced Interprocedural Optimization

A sophisticated compiler may be able to improve performance of a program by moving code between procedures or by applying basic optimizations on code that spans multiple procedures. For instance, code hoisting might extract code from a procedure into its calling procedures, or vice versa. As an example of the latter, register allocation might be performed between procedures in the late stages of compilation. If constant propagation proves that one or more arguments to a procedure assume constant values at a given callsite, it might be possible to exploit this information to create a particularly fast, specialized version of the procedure for that callsite.

Limitations of Interprocedural Analysis

One of the main purposes of interprocedural analysis is to determine which data are accessed, generally subdivided into the tasks of determining what values are defined and which are used, as the result of a given procedure invocation. Note that this determination also depends on the data accessed by those procedures that are invoked by the one under consideration. One of the difficulties with interprocedural analysis and optimizations based on it is that it is, in general, impossible to represent precisely the information gathered by IPA. For instance, a procedure may include several different possible execution paths, each of which accesses a different region of an array. When the compiler summarizes the impact of this procedure, it will combine this information and assume that all of the array elements that may be accessed on some path will indeed be accessed, as it has no way of knowing which of these paths will be executed. Moreover, it may be impossible to represent precisely the set of array elements that may be accessed, and the compiler must always be conservative. A variety of approaches to representing these regions has been proposed, from simple array sections (representation via a lower bound, upper bound, and stride in each array dimension) via linearization of accesses, lists of accesses, and representation as a convex region. The more complex the representation, the more time it will take to compute the region corresponding to a procedure call. Yet the usefulness of some interprocedural optimizations rests heavily on obtaining maximum precision in this analysis.

ARRAY AND LOOP OPTIMIZATIONS

Data Dependence Analysis

The optimizations introduced above are applied to individual scalar variables and are unable to explicitly consider structured data objects such as arrays. In particular, they are unable to deal with subscripted variables or to analyze the data access patterns in loops, where a statement may be executed many times, each time reading and writing a different set of variables. As a result, important optimizations may be missed. Consider the following pseudo-Fortran code fragment:

```
DO I:=1:N
    S:=S+A[I]*B[I]
    B[I]:= 2*C[I] +A[I]
OD I
```

If one were to look only at the variables without differentiating individual vector elements, the code would appear as follows:

```
DO I:=1:N
    S:=S+A*B
    B:= 2*C+A
OD I
```

On the other hand, the ability to distinguish between different elements of the vectors A, B, and C permits us to recognize that the code can be executed in a very different order (something that would not be valid for the undifferentiated version):

```
DO I:=1:N
    S:=S+A[I]*B[I]
OD I
DO I:=1:N
    B[I]:= 2*C[I] +A[I]
OD I
```

The ability of the compiler to analyze accesses to structured data objects, especially arrays, in the presence of nonconstant subscript expressions is crucial for a number of advanced optimization techniques, the foundation of which is (data) dependence analysis. Dependence analysis is a collection of techniques that allow the automatic determination at compile time whether two references to an array will both refer to one or more elements of an array (i.e., whether the regions of the array accessed by them will overlap). If they do not overlap, the compiler is free to reorganize the code in these statements as desired to optimize it. If they do, and one of them defines the variable, then it is essential that the relative order of those accesses be maintained. Indeed, the results of this analysis will enable the compiler to determine whether certain code transformations are semantically valid (produce the same results) in a specific context.

Numerous dependence tests have been developed and published; they are either exact or approximate. Exact tests determine precisely whether there is a dependence. Approximate tests will test for a condition whose validity implies that there is a dependence; if the condition is not satisfied, one assumes that a dependence is present. (known as a “nonfatal” assumption: It may be that, in fact, no dependence is present even though the condition is not satisfied; however, because the presence of a dependence merely impedes the application of a code transformation, not being able to transform the code will leave the semantics unchanged. We may simply miss out on some possible optimization, which an exact test would have allowed us to carry out.) Exact tests tend to be computationally intensive, if not infeasible, which is why approximate tests are commonly used in compilers dedicated to this type of optimization (typically vectorizing and parallelizing compilers).

Code Transformations

Once a complete dependence analysis for a given code has been carried out, code transformations can be applied. Paramount is, of course, that the semantics of the code not be affected by these transformations. As loops tend to account for a significant portion of the computation time of many programs, most code transformations focus on loops and arrays. Very common are loop distribution (replacing one big loop by several smaller ones—see for example, the code fragment above) and loop interchange (where the inner and the outer loop of a loop nest are interchanged—see the examples in I/O Management Section). Other code transformations are the wavefront method, replication and alignment, loop fusion and fission, and strip mining. These techniques were designed with specific objectives in mind, typically automatic vectorization or parallelization.

Vectorization

Vectorization is the attempt to produce "vector code," which is really pipelined code: What is conceptually a vector operation, for example, A[1:N]: = B[1:N]+C[1:N], is implemented as a pipeline of N operations A[i]: = B[i]+C[i]. A vectorizing compiler takes ordinary scalar code and produces equivalent vector code, which is typically done automatically, with no or very little user intervention. Semantically valid code transformations are applied to the given program in order to obtain code that can be vectorized. Automatic vectorization has been spectacularly successful, so much so that today very little manual vectorization is done. A good rule of thumb is that through vectorization, a program may run perhaps five or more times faster than the corresponding scalar version, which should require no more than 5% of the development cost of the original scalar program.

Parallelization

The resounding success of automatic vectorization raised expectations that automatic parallelization is similarly feasible, which proved to be quite unrealistic. Although the fundamental ideas are similar, the granularity of parallel code must be coarser ("more" must be executed) than that of vector code, because the start-up costs of a pipeline are much lower than those of a process or even a lightweight thread. As a result, automatic parallelization has not delivered on its promise, in spite of almost two decades of intensive work.

Language Support

One way in which people have attempted to avoid the difficulties encountered by automatic parallelization is to employ language support, some of which has always occurred in conjunction with vectorization where vectorization directives indicated to a compiler knowledge of the programmer that the compiler did not have or was unable to acquire. For example, an approximate test might be unable to exclude the possibility of a dependence, thereby impeding a possible optimization, while the programmer, alerted to this difficulty, may know that no dependence is present and can indicate this dependence to the compiler via a vectorization directive. Parallelizing compilers must rely on this type of language support to a much greater extent, given the much greater difficulties in parallelizing code automatically.

I/O MANAGEMENT

Motivation

In conventional programming, source code is compiled by a compiler; then the resulting object code is turned over to a run-time support system operating under the operating system (OS). This OS knows very little to nothing about the given program; in contrast, a compiler, especially a high-performance compiler, knows a good deal about the program. This knowledge is especially useful if the program is a "regular" program, such as one involved in many aspects of scientific computation. The information we are interested in is routinely collected by the compiler (in other words, the information collected is neither special nor unusual) and consists mainly of dependence information, which in turn determines which code transformations are semantically valid for a given program fragment.

It is instructive to keep a few key numbers in mind: Access to a single number residing in main memory today takes a few nanoseconds; if the same number resides on magnetic disk, access to it may require tens of milliseconds. Thus, if we fail to keep a number in main memory, which is needed sometime later, it may take 10 million times longer to get to it! Note that the corresponding factor for cache misses is less than 10 (caches are typically less than ten times faster than main memory); the same holds for bank conflicts (it takes four to six cycles to access an item, so pipelining accesses, which banks facilitate, will speed things up by at most that value).

It is important to understand that effective I/O management requires knowledge that is routinely available to a compiler, but is not accessible to the OS. Specifically, it is at the compiler level that decisions are made how to map multidimensional arrays into the one-dimensional memory space. It is at the compiler level where it can be determined whether certain code transformations preserve the semantics of a given source code. Both of these aspects are of crucial importance to the efficiency of I/O, as the following examples indicate.

Example 1. Bank conflicts: Assume that memory is organized in 64 banks and that one memory access takes four clock cycles. Consider the following code:

```
DO I:=1:65536:S
  A[I]:=I*I
OD I
```

where A is an array of size 1:65,536 and S may assume different integer values. It is important to understand the way in which array elements are mapped to memory banks: While the first array element, A[1] resides in some arbitrary bank, say bank b, the next element, A[2], resides in the next bank, namely bank b+1, A[3] in bank b+2, and so on, until the bank number exceeds 64, in which case one starts with bank 1. Now, if S is 64, it follows that every array element accessed by this code resides in bank b; therefore, each access will take four clock cycles. However, if S = 1, it is obviously possible to pipeline the accesses, resulting in an overall time requirement of essentially one cycle per access (altogether, we need 65,539 cycles to retrieve 65,536 elements). It should be clear that it is the value of S that creates problems in this case. However, consider the following code (under the exact same assumptions):

```
DO I:=1:1024
  DO J:=1:1024
    A[I,J]:=I*I
  OD J
OD I
```

Let us assume the mapping of the two-dimensional array A occurs in column-major order (as is the case for all Fortran-based languages). In this case, one can verify that each array access requires almost exactly **four** cycles: The code accesses the array A in rows, and given the stated assumptions, all the elements of any row reside in the same memory bank. Intriguingly, if the array were mapped in row-major order, or alternatively, if the code were replaced through loop interchange by the (completely equivalent) code

```
DO J:=1:1024
  DO I:=1:1024
    A[I,J]:=I*I
  OD I
OD J
```

the access cost per array element would be **one** clock cycle.

Example 2. Virtual memory management: Consider the following code fragment (similar to the last of Example 1):

```
DO J:=1:65536
  DO I:=1:65536
    A[I,J]:=I*I
  OD I
OD J
```

Assume Virtual Memory Management (VMM) is used, with an active memory set size of 1024 pages and a pure LRU replacement strategy (the page **L**east **R**ecently **U**sed is replaced whenever the active memory set size is exceeded). Furthermore, assume that a page holds exactly 2048 elements of the array A; consequently, 2M (or 2,097,152) pages are needed to hold A. As the active memory set is relatively small, it should be clear that paging will occur. Exactly how much depends, however, on the mapping of the 2-D array A into the memory space: If the mapping is in row-major order, every page will be retrieved exactly once, resulting in 2M page transfers. On the other hand, if the mapping is in column-major order, the first 1024 elements of the first row will each correspond to a page; initializing A[1,1025] will invoke the replacement function, because only 1024 pages can be accommodated, the page corresponding to A[1,1] will be displaced, which proceeds until the end of the first row is reached; then the second row of A is initialized. At this point, one observes that the page corresponding to A[1,1] is also that of A[2,1]; unfortunately, this page had been replaced long ago and must now be installed again. The upshot of this process is that, for each array element, a page has to be installed; as there are more than 4 billion (4,294,967,296) elements, the difference, in numbers of pages transferred, between the row-major and the column-major mappings amounts to a factor of 2048.

Several observations are in order: First, the vast majority of programmers are not aware of the mapping function that is employed (in spite of the rule of thumb that Fortran uses column-major and all other languages row-major). Second, and more important for our discussion, a compiler could easily determine that a loop interchange is semantically valid; therefore, if the language dictates column-major mapping (where we would need over 4 billion page transfers), the interchange of the I and the J loops would result in an equivalent code, but one that requires only about 2 million page transfers! Note that this interchange can only be done at the compiler level; once things are turned over to the OS, the context within which page transfers occur has been lost and with it the ability of restructuring accesses in semantically valid ways.

Automatic Minimization of Memory Bank Conflicts

Let us first look at memory bank conflicts. Given a program, together with information about memory mapping, number of cycles required to access main memory, and number of memory banks, a compiler can carry out an analysis (at compile time) of the number and type of bank conflicts that the program causes. This analysis is based on the assumption that the dimensions of the arrays are known at compile time (true for many languages, including Fortran and C). There are two ways in which a compiler can attempt to reduce bank conflicts, by changing the shape of arrays, and by inserting a filler of an appropriate length.

Changing the Shape of an Array. Consider the second code fragment of Example 1 above, assuming column-major mapping. Although, of course, it is possible to do a loop interchange in this case, matrix multiplication, for example, will require access by rows and by columns; therefore, this approach would not work in general. However, one can redefine the shape of the matrix; instead of defining it as (1:1024, 1:1024), one could define it as (1:1025, 1:1024), assuming the mapping is column-major. (If the mapping is row-major, the array should be defined as (1:1024, 1:1025).) In this way, traversing a row will not result in bank conflicts, as now A[1,1] is in bank 1, A[1,2] is in bank 2 (instead of in bank 1, as before the reshaping), and so on. Note that it is only the definition of the array that must be changed; all of the code manipulating the array remains completely unaffected. Furthermore, the amount of "wasted" memory is relatively small, one single column (or row). This process can be done at compile time, driven by the bank conflict analysis (in other words, if the analysis indicates a significant amount of conflicts, the reshaping operation is carried out and the resulting code is subjected to a bank conflict analysis; if the reshaped version has fewer conflicts than the original one, the new version is used, otherwise the original is restored).

Inserting a filler. Consider the code fragment

```
S:=0
DO I:=1:1024
     S:=S+A[I]*B[I]
 OD I
```

where we assume as before that we have 64 banks and each access takes four cycles. Moreover, we assume that the two arrays are of size 1024 and are declared in a way that causes them to be allocated contiguously in main memory. One can easily see that A[1] and B[1] will end up in the same bank, and so will A[2] and B[2], A[3] and B[3], and so on, which causes a significant amount of conflicts, which the compiler can determine at compile time. Inserting a filler of length 4 between the two arrays causes these conflicts to evaporate, as now A[1] will be in bank 1, but B[1] in bank 5; similarly, A[2] will be in bank 2 and B[2] in bank 6, A[3] in bank 3 and B[3] in bank 7, and so on. Again, the memory bank conflict analysis of the compiler can drive this process. More sophisticated analyses can also be carried out, taking into account the fact that the length of the filler can be varied as required. In this example, any length between 4 and 60 would work, allowing the compiler to consider and combine constraints imposed by other code fragments.

Automatic Minimization of Block Transfers: I/O Profiling

The code transformations referred to in the Code Transformations Section can be carried out with the objective of minimizing I/O, instead of vectorization or parallelization, which can be visualized by considering the Example 2 above for column-major mapping. It should be obvious that significant savings in I/O can be achieved provided a loop interchange is semantically valid. For this purpose, it is necessary to first establish the I/O profile of a given program, which can be done automatically at compile time and provides a measure of the amount of I/O occurring during execution. Based on the dependence analysis, the compiler can now carry out semantically valid code transformations, with the goal of reducing I/O. The resulting modified code is again I/O profiled; if the new version has a better I/O profile than the original one, it is retained, otherwise different code transformations are applied. In this way, it can either be established that the original program was already I/O efficient, or else a more I/O efficient program is obtained.

Compiler-Driven I/O Management

It should be clear from the previous paragraphs that all techniques described can be carried out automatically on the basis of information that is available to the compiler (at compile time). In fact, it is not very difficult to comprehend that the compiler has significantly more knowledge about the program (access patterns, use of complex structures and arrays, etc.) than the operating system. Therefore, it should also be obvious that I/O management is best accomplished by the compiler. In other words, it should be the compiler that determines which blocks are to be transferred between disk and main memory or between main memory and cache, not the OS. This gives rise to compiler-driven I/ O management, which will likely play an increasingly important role because external memory devices, primarily magnetic disk drives, have not increased in access speed over the past decade or so (and there is no indication that this trend will change), while CPUs have become significantly faster. This fact implies that more and more formerly CPU-bound programs will become I/O-bound, emphasizing the increasing need for intelligent I/O management.

LOW-LEVEL OPTIMIZATIONS

Among the most important optimizations applied in practice are those that attempt to examine and improve a version of the program that is a representation of the actual machine code that will be generated. At this low level, it is possible to analyze the use of data and to assign registers carefully, and to examine the sequences of instructions generated and consider how they will be mapped to the hardware resources. The use of such information to improve the selection and ordering of instructions, as well as details of the assignment of data to registers, is highly machine-specific. For instance, if it is known that it takes a certain number of cycles to load an integer into a register and the generated code uses an integer in the instruction immediately following a load, then the compiler might attempt to reorder the instructions so that other work is performed while the value is being loaded. Similarly, knowledge of the time it takes to handle a branch in the code might permit the insertion of other instructions in order that cycles are not wasted. Other low-level optimizations will already have attempted to optimize the number of branches required by the code. They may also attempt to perform branch prediction, which attempts to determine the most likely path that will be taken at run time, or may attempt to identify sequences of code that may be translated in a particularly efficient way on the target machine. Such *machine idioms* may be very short sequences of code, often the combination of two instructions into a single instruction in the target machine's instruction set.

Register Allocation

Possibly the best-known low-level optimization is register allocation. The purpose of register allocation is to make the best possible use of registers, a limited set of highest-speed memory locations that allow for the most efficient machine instructions to be applied. Note that some architectures require that all data used as operands be in registers.

As with most optimization problems, there is no practical algorithmic solution to the problem of assigning variables optimally to registers, and thus heuristics are used to provide approximate solutions instead. The standard approaches to dealing with this issue require an analysis of the live range of variables (i.e., the determination of the code region where a variable is referenced), which is used to construct a so-called *interference graph* to represent variables and their live ranges. Nodes correspond to variables and there is an edge between a pair of nodes if and only if their live ranges overlap. If there is no edge between a pair of nodes, it will be possible to assign them to the same register, because they are needed at different times during execution. On the other hand, if there is an edge, then we need different registers to hold the current values of the variables involved. By associating a distinct color with each hardware register, we may equate the problem of assigning distinct registers to these variables with the problem of coloring the interference graph in such a way that no connected nodes have the same color. In practice, several approaches have been proposed to find colorings for the interference graph, typically by simplifying this graph (by removing nodes and the edges from them) until one is found for which the given hardware-dependent number of registers can indeed provide such a mapping. As there are seldom sufficient registers to hold all live variables simultaneously, the mapping is typically achieved by spilling registers, the term given to the storing of values temporarily in memory until they can be restored to a register. Spillage will occur for all variables whose nodes were eliminated from the graph during the construction of a solution. The increase in instruction-level parallelism, or potential to exploit multiple functional units simultaneously, has increased the need for more variables to be in registers at any given time, and has made it harder to provide good solutions to this optimization problem.

Instruction Scheduling

Instruction scheduling refers to the ordering of machine instructions for execution, which is complicated by the need to keep a number of different functional units busy during execution. As this order obviously also affects the live range of the variables referenced, the scheduling of instructions is not independent of register allocation, and this interdependence is one of the difficulties of low-level optimization.

Typically, a machine will permit the independent, concurrent execution of specific kinds of instructions and it is the job of this phase in compilation to determine independent sets of computations that can be fed to the corresponding functional units simultaneously. An obvious limiting factor in this scheduling is the occurrence of a branch in the code; a variety of approaches have been proposed to help the machine carry out useful work despite those branches, including speculative techniques that guess at the branch that will be taken and begin to perform the work in that execution path. If the guess turns out to be incorrect, work is needed to undo the instructions performed.

Many current machines permit software pipelining, which is analogous to hardware pipelining, and performance improvements are obtained by overlapping the execution of distinct operations, which is typically most useful for loops, in which a set of instructions is repeated many times and in which data reuse may be high and the reuse distance known. The instruction scheduling required to facilitate software pipelining may be supported by loop unrolling, in which the body of a loop is increased by replicating the code (with suitable adaptation) for a given number of loop iterations known as the unroll factor. That is, there will be fewer loop iterations, but each of them will have more code and presumably higher levels of reuse of certain data objects. If the unroll factor chosen is too large, then the data required might not fit into cache, which is likely to offset any performance gains provided by this approach, so the unroll factor must be carefully chosen.

Emerging ideas on instruction scheduling consider how this process can be supported by having additional “helper” threads to determine the branches that will be taken, or to prefetch data to avoid some of the inevitable delays when instructions cannot be reordered in such a way that stalls (or waits) are offset.

PRACTICAL CONSIDERATIONS

Optimizing compilers are ordinary compilers with a well-developed optimization phase. There are often various options to select, depending on what optimizations are to be attempted. Clearly, no optimization should be performed if the program is not yet debugged or in its final form. It is important to view optimization as an investment: Carrying out the required analyses comes at a cost, namely the time the compiler requires to complete them. Unless there is a reasonable expectation that this investment will bring an acceptable return, it makes no sense to do optimization. For example, a program that is used once and whose execution time is short is not a good candidate for optimization. On the other hand, a program that is executed frequently and runs for long periods should probably be extensively optimized.

Another point that must be stressed is that optimization typically will reduce the running time of a program by a constant factor. This factor may be quite attractive, say 60%, but it will not grow asymptotically. To illustrate, consider two methods of sorting n numbers, one taking $4n\log_2(n)$ instructions, the other $n^2/4$ instructions. In this case, the advantage of the first method increases with n. Optimization does not work in this manner—if it saves 60% of a given program, this percentage will remain essentially unchanged and independent (for the most part) of the size of the input.

Finally, we note that a “good” optimizing compiler is by no means one that attempts to apply all possible analyses and code transformations. Indeed, such a compiler would be extremely costly to use because all this work will require a great deal of time and effort. As a result, a large number of programs would never be able to recover the investment in optimization through a reduction of the aggregate running time (taken over the life of the program). Instead, a good optimizing compiler tends to be one that applies a limited number of strategies very efficiently. In this way, a much larger number of programs can benefit properly from optimization.

OUTLOOK

As the complexity of hardware continues to grow, the need for re-evaluation and further development of optimization strategies is great. Current challenges include the need to enable code to exploit increasing levels of architectural parallelism at a low level, and to take the impact of a variety of new hardware and operating system mechanisms for multithreading into account. The widespread proliferation of a variety of handheld devices and telecommunications applications has led to the broader use of Java as a programming language and the necessity of choosing combinations of interpretation and dynamic compilation to minimize download time, execution time, memory usage, and power consumption. Multisite and grid computing has led to interest in the portability of code and thus to optimization at run time (or a separation of basic translations from the major optimizations). These ideas have given impetus to the exploitation of dynamic optimization for traditional languages, as well as the somewhat simpler gathering of profile data by a compiler and its use in a feedback loop that aims to improve the application of optimizations, possibly in a manner that is specific to a single execution. There are costs associated with the replacement of code fragments during execution, as well as the more obvious cost of any dynamic instrumentation needed to determine the suitability of such replacement, which is an area of active research and development. With the continued innovation in computer architectures and the growth in size of applications, computer jobs, and system configurations, we expect to see innovations in the area of compiler optimization for some time to come.

FURTHER READING

A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques and Tools*, Reading, MA: Addison-Wesley.

E. L. Leiss, *Parallel and Vector Computing: A Practical Introduction*, New York: McGraw-Hill.

S. S. Muchnik, *Advanced Compiler Design and Implementation*, San Francisco, CA: Morgan Kaufmann Publishers.

H. Zima and B. M. Chapman, *Supercompilers for Parallel and Vector Computers*, Reading, MA: Addison-Wesley.

Barbara M. Chapman
Ernst L. Leiss
University of Houston
Houston, Texas

P

PARAMETER PASSING

Modern programming languages use various techniques to pass parameters to a function (or method). We discuss the techniques used in some of the more popular programming languages in current use, namely C, C++, Ada, Java, and C#, and mention some techniques found in other languages.

INTRODUCTION

Suppose our programming language defines a function:

```
int idx = 1;
int arr[ ] ={ 37, 12, 17 } ;
// Array of length 3, indices 0...2
void foo ( int param )
{
  idx++;
  param++;
  print ( param ) ; // What is printed here?
}
```

In this example, as in all others, the parameters that are listed with the function declaration are the *formal parameters*. The parameters supplied in the function call are the *actual arguments* (sometimes also known as the *actual parameters*). The function can be invoked as shown below:

```
foo ( arr[ idx ] ) ; // What happens to arr?
```

The natural question, of course, is to ask what is printed on the last line of function `foo`, and what are the array contents that are printed after invoking `foo`?

Of course, experienced programmers will answer easily the question in the language of their choice, but the answer depends on the underlying decisions made in the design of the particular programming language that dictates how the parameters are actually being passed. For instance, here are some possible scenarios (all of which add one to `idx`):

1. `param` is simply a copy of `arr[ 1]`. In this case, `param` is initially 12 and is incremented, and the value 13 is printed. However, `arr[ 1]` (and thus the remainder of `arr`) are unchanged.
2. `param` is a synonym for `arr[ 1]`. In this case, `arr[ 1]` is incremented to 13.
3. `param` is a synonym for `arr[ idx]`, and since `idx` is incremented to 2, `param++` increments `arr[ 2]`.

Because all of the above results could plausibly be the desired outcome, each programming language provides mechanisms to achieve at least one of these results. Needless to say, the example above, with just one integer parameter and a few lines of code, is fairly simple. Thus, a full programming language will have many complications and subtleties that are beyond the scope of this article. The remainder of this article describes the most common parameter passing mechanisms in five fairly similar languages that are in common use today: C, C++, Ada, Java, and C#. We also briefly mention parameter passing in other languages.

PARAMETER PASSING IN C

Call-By-Value

Before discussing call-by-value in C, it is helpful to explain how values are treated in C (and similar languages such as C++). C makes use of the notion of two types of values: an L-value and an R-value. In C, an L-value is an expression that refers to a region of storage that can be examined and stored into. An R-value refers to a data value that is stored at some address in memory. A literal constant can serve as an R-value. Variables in C, such as `x`, `*p`, or `a[ i]` have two values: the L-value (address) and the R-value (contents of that address).

In C, all parameters to functions are passed using the mechanism of call-by-value. In call-by-value, the actual argument is copied into the formal parameter (this is Scenario 1 in the example in the introduction). So in call-by-value, the formal parameter is a new local variable that is initialized with the R-value of the actual argument. Call-by-value has some limitations.

The first limitation is that call-by-value is expensive if used on parameters that are large. Because C is not an object-oriented language, most parameters are primitive types or pointer variables. Thus, they are inexpensive to copy. In original C, `struct` types (i.e., aggregates) could not be passed as a parameter, and instead they are passed indirectly by passing a pointer to the `struct`. Modern C does allow passing of `struct` types; however, because of the cost of copying, `struct` types are almost always passed by using a pointer. Arrays could theoretically be expensive to pass via call-by-value, but in C this is not an issue because arrays in C are represented by a pointer variable that points at a suitably large block of memory. What is being passed is not the array but is the address of the large block of memory that stores the array items. (Unfortunately this means that an additional parameter that represents the array size must also be passed, or some other mechanism must be used.) Similarly, strings are actually arrays of characters, and pointer variables are used.

The second limitation is that call-by-value does not allow the function to change the actual argument, which makes routines such as `swap`, which would swap its two parameters, impossible to write directly.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.
Copyright © 2008 John Wiley & Sons, Inc.

Call-By-Passing Pointers

In C, as described in the previous section, arrays and strings are passed by using a pointer to the block of memory in which they are stored, as are `struct`s. Call-by-value guarantees that the value of the pointer variable is used, so after the call returns, the pointer variable will be viewing the same block of memory, but the contents of the block of memory can change. This mechanism is also useful for writing routines, such as `swap`, that change two variables. Sample code is shown below:

```
void swap( int *px, int *py )
{
  int tmp = *px;
  *px = *py;
  *py = tmp;
}
```

In this routine the two parameters represent the addresses (i.e., the L-values) of the two integers that are being swapped. To invoke `swap`, we simply pass in two addresses (L-values) as shown in the following fragment.

```
int x = 5;
int y = 7;
swap( &x, &y );
```

Call-By-Macro Expansion

Another alternative in C is the use of preprocessor macros. These macros technically are not functions, and have significant differences from functions. Because they look enough like functions and are often used like functions, we describe its parameter passing mechanism.

In the *call-by-macro expansion*, the actual arguments are *substituted textually* in the macro body in all places where the corresponding formal parameters appear. Then the macro body replaces the macro call. Thus, in C, given the macro:

```
#define CUBE( a ) ( (a) * (a) * (a) )
```

the statement

```
y = CUBE( x + 3 ) * 5;
```

is interpreted as

```
y = ( (x + 3) * (x + 3) * (x + 3) ) * 5;
```

Scenario 3, as described in the Introduction, can be implemented straightforwardly in C if macros and the comma operator are used:

```
#define foo(param) (idx++,param++,print(param))
```

Macros have some distinct advantages over functions, but also they have some tremendous liabilities. On the positive side, macro parameters are typeless (one can pass an `int`, `double`, etc. as a parameter). Macros avoid the overhead of a copy, which can save time (but not as much time as was the case in the 1970s, at the height of C's popularity). The time savings was the original motivation for the use of macros. Macros can be used to evade call-by-value restrictions. Macros can also be used for very tricky code in which several parameters can be combined to form a resulting string that can then be the name of a variable. The major problem with macros is that they are not semantically equivalent to function calls. Most obviously, without the excessive parenthesis shown in the macros the programmer runs the risk of the macro expansion generating code that is wrong because of precedence rules. For instance, with no parenthesis, the macro is interpreted as

```
y = x + 3 * x + 3 * x + 3 * 5
```

Even with parenthesis, macro arguments are evaluated as many times as needed in the expansion, so for instance `CUBE(sin(theta))` calls the expensive `sin` function three times, thus losing any speed benefit that might have accrued. And if the macro argument contains a side effect, as in `CUBE(++x)`, the macro call is unpredictable but definitely not the same as a corresponding function call.

An additional concern with macros has to do with problems that can arise when variables in the macro expansion collide with variables in the function that contains a call to the macro. In this article's last section, we see *call-by-name*, which was prominently used in Algol 60. Call-by-name uses variable capture to avoid these issues.

PARAMETER PASSING IN C++

C++ is designed for the most part as a superset of C, so all parameter passing idioms described for C are valid in C++. However, because passing pointers and using macros are notoriously error-prone, C++ provides alternatives that are much safer than the corresponding C constructs.

Inline Optimization

In C++, functions can be declared inline. When the compiler inlines a function call, the effect is that the function call is replaced by the body of the function, with the formal parameters replaced carefully by the actual arguments. However, if a formal parameter is used more than once, it is evaluated only once, and its value is saved and reused for the subsequent occurrences. This process makes an inline function semantically equivalent to a normal function call, differentiating it from a macro expansion.

When a C++ function is declared inline, the declaration is considered nonbinding advice to the compiler. If the body of the function is not suitably compact, or the actual arguments are not relatively simple (or are aliased), the compiler is likely to avoid the optimization.

Call-By-Reference

In call by reference, the formal parameter is a synonym (alias) for the evaluated argument (they have the same L-values). A formal parameter that is declared with an `&` is considered to be passed using call-by-reference semantics.

Thus, Scenario 2 in the Introduction can be implemented with the following code:

```
void foo( int & param )
{
   idx++;
   param++;
   print( param ); // What is printed here?
}
```

When the call to foo is made, as before:

```
foo( arr[ idx ] ); // What happens to arr?
```

param becomes another name for arr[1] (because idx is 1 at the time of the call). Call-by-reference makes it easy to write a swap routine without using pointer variables:

```
void swap( int & x, int & y )
{
   int tmp = x;
   x = y;
   y = tmp;
}
```

In this routine, the two parameters represent the two integers that are being swapped. To invoke swap, we simply pass in two integers as shown in the following fragment:

```
int xx = 5;
int yy = 7;
swap( xx, yy );
```

Call-by-reference requires that the actual argument be a modifiable L-value (i.e., it can be assigned to; thus objects that we declared with const are not acceptable arguments) and of the same type, or in the case of inheritance, the actual argument can be a public subclass of the formal parameter type. Call-by-reference is typically implemented by the compiler by passing invisibly the address of the actual argument, and then by dereferencing the pointer variable that is received by the function.

Call-By-Reference to a Constant

In C, when large aggregates need to be passed to a function, a pointer variable is used to pass the address of the aggregate, rather than an entire copy of the aggregate. But working with pointer variables is clumsy. Consider the following code:

```
int binarySearch( vector<int> arr, int x )
{ /* implementation not shown */ }
```

In this code, even though binarySearch is a fast operation (using $O(\log N)$ time), the code will require $O(N)$ time because call-by-value will mandate a copy of the N-element vector that is passed as the first parameter. An alternative, supported in other older languages such as Pascal, is pass by reference:

```
int binarySearch( vector<int> & arr, int x )
{ /* implementation not shown */ }
```

This code will be much faster because it avoids the copy of the vector. However, it is NOT semantically equivalent. If the implementation resizes arr, or makes a change to arr, the original version will not reflect a change in the actual argument, whereas the new version will. In addition, the original version works with constant (i.e., immutable) vectors; the new version does not. Both problems are solved using call-by-reference to a constant:

```
int binarySearch( const vector<int> & arr, int x )
{ /* implementation not shown */ }
```

In this code, the actual argument is still being passed by reference. However, in the scope of binarySearch, arr is treated as a constant. Thus attempts to resize or change arr will result in a compiler error. Consequently there is a guarantee that when binarySearch returns, the actual argument will not have been changed (it is possible to trick the compiler by using type casts to remove the const, but doing so requires at least some effort).

Call-by-reference to a constant also is sometimes known as *call-by-constant reference.*

C++ Parameter Passing Summary

For the most part, parameter passing in C++ either uses call-by-value (the default), call-by-reference, or call-by-reference to a constant. Macros are now rarely used, and modern optimizing compilers perform inline optimization whether or not it is requested by the programmer. The decision on which parameter passing mechanism to use is critical, and the following table summarizes the general principles on which mechanism is appropriate:

Call-by-reference	Actual argument **may need to be changed.** In this case, it does not matter what is the type of the actual argument.
Call-by-value	Actual argument **should not be changed**, and **copy cost is minimal**. The actual argument is usually a primitive or pointer type or a class type that is unusually easy to copy.
Call by reference to a constant	Actual argument **should not be changed**, and **copy cost is expensive**. The actual argument is a class type such as vector.

PARAMETER PASSING IN ADA

Ada's approach to parameter passing is somewhat different from C++'s approach. In C++, when we have an actual argument that we want to be sure will not be changed by the function, the programmer will declare that either call-by-value or call-by-reference to a constant is used, because either mechanism provides (for the most part) the guarantees that we want. However, the programmer must make a decision on whether a copy of the actual argument should be made (call-by-value) or hidden pointers should be used (call-by-reference to a constant). Ada takes the position that this choice is best left to the compiler. The programmer should specify simply that the formal parameter should be "read only."

Thus, in Ada, parameters are passed in one of three modes, representing roughly "read only," "read and write," and "write only." These parameters are `in`, `in out`, and `out`, respectively. The default mode is `in`.

A second difficulty is that in the call `bar(x,y)`, there is no way to tell, without looking at the signature of `bar`, whether it is possible that `x` and `y` can be changed in the call. Ada avoids this problem by differentiating between "functions" and "procedures." A function in Ada must have a return value, and the caller cannot ignore the return value. A procedure in Ada cannot have a return value. Ada requires that all parameters to functions are `in` parameters. Thus, it is guaranteed that the actual arguments to a function will not be changed during the execution of the function call, and it is easy to distinguish between a function and a procedure. However, as in C++, in a procedure call, there is no way to tell how parameters are being passed without looking at the procedure's signature.

In Mode

The formal parameter is a constant and permits only reading of the value of the associated actual parameter. The formal parameter cannot be assigned to (so it cannot appear on the left-hand side of an assignment statement). If the parameter is a primitive, then a copy is made. Otherwise, a reference is used. Thus, the Ada compiler makes these decisions that can cause significant performance problems for C++ programmers.

In Out Mode

The formal parameter is not a constant and permits both reading and writing the value of the associated actual parameter. The formal parameter can appear on both sides of an assignment statement.

If the parameter is large, then the compiler will probably elect to pass by reference, as in C++. Arrays are passed this way, and some types are required by the language specification to be passed by reference.

Call-By-Value Return

However, if the parameter is a primitive, then the language specification requires that a copy is made when the procedure commences (recall that all function parameters are passed using `in` mode). When the procedure returns, the current state of the formal parameter is copied back into the actual argument. This is known as call-by-value return, or sometimes as copy-in, copy-out.

In most circumstances, call-by-value return and call-by-reference achieve the same semantic result as allowing the actual argument to be changed by the procedure. However, there are some subtle cases, many of which involve parameter aliasing, when call-by-value return and call-by-reference give different results. Consider the following contrived example:

```
u : integer := 5;
procedure silly( x: in out integer ) is
begin
 u := 0;
 x := x + 1;
end silly;
```

Suppose we invoke `silly(u)`. Using call-by-reference as the parameter passing mode would set `u` to 1. However, the result is different if call-by-value return is used, because since `x` will be a copy of `u`, the value of `x` prior to the procedure return will be 6 and then that value is copied back to `u`.

The Ada Language Specification allows the compiler to choose either method to pass record types using *in out* mode, and thus in the presence of aliasing, the compiler's choice affects the behavior of the program. Ada programs that rely on knowing which parameter passing mechanism has been chosen by the compiler are considered nonportable. The program above is portable because, as was mentioned, primitive types must use call-by-value return to pass `in out` parameters.

Out Mode

In early versions of Ada, the compiler permitted only to write the value of the associated actual parameter, and the formal parameter could appear only on the left-hand side of an assignment statement. This rule was relaxed, so now the formal parameter is a variable and may be assigned values; however, its initial value should be considered undefined (because if the initial value is important, then `in out` is the correct parameter passing mode). Technical issues involve the idea that the actual argument is undefined, but might be partially constructed, and its partial initialization should not be lost. As a result, parameter passing is similar to `in out` parameters.

Named Parameters

The print procedure shown below is typical of procedures that have lots of parameters: Many of the parameters have the same type, and it is difficult to remember the order of the parameters. The normal way of invoking the procedure is to use positional parameters: The caller must list the parameters in the same order that the function does. If the caller of the procedure supplies the last parameters in the wrong order, the program will still compile, unless parameters of different types are interchanged. Obviously this procedure is prone to error.

```
procedure print( file_name : String;
                 indent : integer;
                 line_len : integer;
                 lines_per_page : integer );
```

Ada allows the use of named parameters. For instance the following calls are all acceptable:

```
print( file_name => "foo.txt", indent => 5,
       line_len => 72, lines_per_page => 62 );
```

```
print( file_name => "foo.txt", indent => 5,
       lines_per_page => 62, line_len => 72 );
print( lines_per_page => 62, indent => 5,
       line_len => 72, file_name => "foo.txt" );
```

Each actual argument is associated specifically with a formal parameter; the order does not matter. It is also possible to mix the positional arguments and the named arguments, but the positional arguments must all come first, and in their correct positions. So it is customary to make sure the most important formal parameters are specified first in the signature. Here are examples of a valid mixed parameter call:

```
print( "foo.txt", line_len => 72, indent => 5,
   lines_per_page => 62 );
print( "foo.txt", indent => 5,
   lines_per_page => 62, line_len => 72 );
print( "foo.txt", 5,
   lines_per_page => 62, line_len => 72 );
```

PARAMETER PASSING IN JAVA

In Java, the only parameter passing mechanism available is call-by-value. For value types (which are primitive types, only), the value is copied. For reference types (which are everything else, including strings, arrays, and collections), this means that the value of the reference variable, rather than the entire object being referenced, is copied. Thus, the copy is never expensive.

As in C, a mechanism exists to evade the call-by-value restriction and to simulate call-by-reference. In Java, because the state of the object that is being accessed by the formal parameter's reference variable can always be changed, information can be passed back to the caller by embedding it in an object and passing a reference to the object. Sometimes the object is simply an array of length 1. Other times it is a more complex entity. Some routines, such as `swap`, are impossible to write cleanly in Java.

PARAMETER PASSING IN C#

The C# model is similar to the Java model, with a few additions. As in Java, a distinction exists between value types and reference types, and although value types can include types that are not primitives (known as `struct` types), the general expectation still exists that parameter passing using call-by-value should never be expensive. Thus, almost all parameter passing in C# is similar to Java and the default is call-by-value.

However, C# also allows call-by-reference. Both the formal parameter and actual arguments must be preceded with the keyword `ref` (if exactly one of the parameter/argument pair contains `ref`, it is an error). Here is sample C# code for `swap`:

```
void swap( int ref x, int ref y )
{
  int tmp = x;
  x = y;
  y = tmp;
}
```

In this routine, the two parameters represent the two integers that are being swapped. To invoke `swap`, we simply pass in two integers as shown in the following fragment.

```
int xx = 5;
int yy = 7;
swap( ref xx, ref yy );
```

Requiring the `ref` prior to the actual argument solves C++'s problem that in C++, the caller cannot distinguish a parameter passed using call-by-reference from call-by-value or call-by-constant reference without seeing the corresponding function signature.

C# also provides `out` parameters that behave somewhat like in Ada. As with `ref` parameters, the keyword `out` must be used prior to the formal parameter and the actual argument. The compiler will assume that the `out` formal parameter is uninitialized on entry to the function and will verify that it is definitely assigned before the function return.

PASSING A FUNCTION AS A PARAMETER

All of the languages that we have examined provide the ability to pass a function (or procedure) as a parameter to a function (or procedure). In all cases, the syntax is definitely nontrivial, but one of two competing philosophies is as follows.

1. Pass a pointer to the function as a parameter (C, C++, Ada).
2. Embed the function inside a class type, and pass a reference (or copy) of the class type as a parameter (C++, Ada, Java, C#). This idea is often known as a *function object.*

Passing a Pointer to a Function in C, C++, and Ada

Passing the pointer is generally considered an inferior solution; among the languages we have examined, this solution is most appropriate in C. The following function applies function `func` to every element in array `input` and produces the answer in the corresponding slots of array `output`:

```
void evaluate( const double input[ ],
               double output[ ],
               int n, double (*func) (double x))
{
  int i = 0;
  for( i = 0; i < n; i++ )
    output[ i ] = (*func) ( input[ i ] );
}
```

The onerous syntax for a pointer to function in C can be simplified in modern C with:

```
void evaluate( const double input[ ],
               double output[ ],
               int n, double func( double x ))
{
  int i = 0;
  for( i = 0; i < n; i++ )
    output[ i ] = func( input[ i ] );
}
```

In either case, the following code fragment computes some square roots and logarithms:

```
double arr[ ] ={ 8.5, 7.9, 4.2, 7.3 } ;
double roots[ 4 ] ;
double logs[ 4 ] ;
evaluate( arr, roots, 4, sqrt );
evaluate( arr, logs, 4, log10 );
```

This code also works, unchanged in C++, but as mentioned it is considered by modern C++ programmers to be an inferior solution to the one shown later that makes use of function objects. The same basic logic can be used in Ada95, as shown in the following code:

```
with Text_IO; use Text_IO;
with Numerics.Elementary_Functions;
    use Numerics.Elementary_Functions;
procedure Function_Pointers is
    type Array_Type is array( Integer range <> )
       of Float;
    type Math_Func is access function (X : Float) :
       Float;
procedure Evaluate( Input : Array_Type;
                    Output : out Array_Type;
                    func : Math_Func ) is
begin
  for( I in Input’ range ) loop
    Output[ I ] := Func.all( Input[ I ] );
  end loop;
end Evaluate;

begin
  Arr : Array_Type(1..4 ) :={ 8.5, 7.9, 4.2, 7.3} ;
  Root : Array_Type( 1..4 );
  Logs : Array_Type( 1..4 );
  Evaluate( Arr, Roots, Sqrt’ access );
  Evaluate( Arr, Logs, Log’ access );
end Function_Pointers;
```

Function Objects in Ada, Java, and C#

In these languages, functions are passed to parameters by embedding each function in a class type, and then by creating an instance of the class type. Then, a reference to the object (containing the function) can be passed as a parameter. In these languages, inheritance in the form of an interface is used to specify the signature of the function being passed. Here is a Java example:

```
interface MathFunctionObject
{
  double func( double x );
}
class FunctionPointers
{
   public static void evaluate
                        ( double[ ] input,
                          double[ ] output,
                          MathFunctionObject f )
   {
     for( int i = 0; i < input.length; i++ )
       output [ i ] = f.func( input[ i ] );
   }
   public static void main( String[ ] args )
   {
     double  [ ] arr ={ 8.5, 7.9, 4.2, 7.3 } ;
     double  [ ] roots = new double[ 4 ] ;
     double  [ ] logs = new double[ 4 ] ;

     evaluate( arr, roots, new SqrtObject( ) );
     evaluate( arr, logs, new Log10Object( ) );
   }
   private static class SqrtObject
                  implements MathFunctionObject
   {
     public double func( double x )
       { return Math.sqrt( x ); }
   }
   private static class Log10Object
                  implements MathFunctionObject
   {
     public double func( double x )
       { return Math.log10( x ); }
   }
}
```

Function Objects in C++

In C++, inheritance is replaced by template expansion and overloading of `operator()`. The syntactic tricks are that `evaluate` is expanded once for each function object type and `func.operator()` is replaced by simply `func`.

```
template <typename MathFunctionObject>
void evaluate( const vector<double> & input,
               vector<double> & output,
               MathFunctionObject func )
{
  for( int i = 0; i < input.size( ); i++ )
    output[ i ] = func( input[ i ] );
}
class SqrtObject
{
  public:
    double operator() ( double x ) const
      { return sqrt( x ); }
} ;
class Log10Object
{
  public:
```

```
  double operator() ( double x ) const
    {  return log10( x ); }
};

int main( )
{
   vector<double> arr( 4 );
   arr[ 0 ] = 8.5; arr[ 1 ] = 7.9;
   arr[ 2 ] = 4.2; arr[ 3 ] = 7.3;
   vector<double> roots( 4 );
   vector<double> logs( 4 );

   evaluate( arr, roots, SqrtObject( ) );
   evaluate( arr, logs, Log10Object( ) );
}
```

Passing Functions in Functional Languages

In other languages, particularly functional languages such as Scheme, ML, or Haskell, functions are treated as just another kind of value and do not require the baroque syntax of the languages we have illustrated in this section.

ADDITIONAL FEATURES

Two interesting features that are somewhat common are the use of *default parameters* and *variable numbers of arguments*.

Default Parameters

Default parameters are found in both C++ and Ada. In these languages, formal parameters can be provided with default values that will be used if the actual argument is omitted.

Here is a C++ example:

```
double myLog( double n, int base = 10 )
{
    return log10( n ) / log10( base );
}
```

In this example, the call `myLog(n,2)` is valid, and so is `myLog(n)`. In the later case, base will be presumed to be 10. Significant rules exist regarding when and where default parameters can be used, and the parameters do not mix well with other features, such as inheritance. The Ada code is comparable with its C++ equivalent.

Because many languages support function overloading (allowing the same function name to be used as long as parameter signatures differ), default parameters are not essential and can be viewed as strictly a syntactic convenience.

Variable Arguments

Variable argument lists, in which an unknown number of actual arguments can be passed, are found in C, C++, C#, and later versions of Java (starting with Java 5). In all instances, zero or more "known" actual arguments are passed, followed by the "unknown" group. Strictly speaking, variable arguments are a convenience, because one can always achieve the same effect by using an array to encapsulate the unknown group of actual arguments. Not surprisingly, then, Java 5 and C# take a similar approach in which the unknown actual arguments are accessed by an array. Here is example code for implementing a variable argument `max` function in Java:

```
public static int max( int first, int ... rest )
{
   int maxValue = first;
   for( int i = 0; i < rest.length; i++ )
     if( rest[ i ] > maxValue )
        maxValue = rest[ i ];
   return maxValue;
}
```

The same idiom is used in C#, via `params` arrays:

```
int max( int first, params int[ ] rest )
{
   int maxValue = first;
   for( int i = 0; i < rest.Length; i++ )
     if( rest[ i ] > maxValue )
        maxValue = rest[ i ];
   return maxValue;
}
```

In both languages, these functions support calls such as `max(3,5,2,1,4)`, `max(3,5)`, and `max(3)`. Also supported is `max(3,new int[ ]{ 5,2} )`, which illustrates how the compilers are really handling the situation (and how similar C# and Java really are).

C and C++ use a significantly uglier strategy of invoking macros that manipulate the runtime stack. The calls to the macros are platform independent, although the implementation of the macros obviously is not.

PARAMETER PASSING IN OTHER LANGUAGES

Call-By-Name

Call-by-name is a parameter passing mechanism that is most associated with the influential 1960s programming language, Algol-60. In call-by-name, the actual arguments are substituted in the macro body in all places where the corresponding formal parameters appear. Although this sounds exactly like call-by-macro expansion, which is used in C (and also C++), the important difference is that the substitution is not textual. Rather, it is *capture avoiding*, meaning that care is taken to ensure that actual arguments and local function variables that have identical names are treated differently. For instance, if the actual argument is `arr[ idx]` and the function also contains a local variable named `idx`, when `arr[ idx]` is substituted for all occurrences of the formal parameter, `idx` represents the variable in the caller's context, rather than the local variable named `idx` in the function. This is done using a structure known as a *thunk*.

Call-by-name has two desirable properties. First, if an actual argument is not actually needed in the function, it is not evaluated. Here is a simple example:

```
int foo( bool cond, int x, int y )
{
    if( cond )
        return x;
    else
        return y;
}
```

Consider either of the following calls: `foo(true,u,1/u)` or `foo(false,loop(u),bar(u))`. In the first call, if u is 0, the C parameter passing mechanism, which is call-by-value, will cause a divide-by-zero error. But using call-by-name, because the formal parameter y is never needed, no divide-by-zero will occur. In the second case, if function `loop` is nonterminating, if call-by-value is used, then `foo` never finishes (actually it never starts). With call-by-name, `loop` is never called. This process makes it easier to prove program properties mathematically.

The second desirable property is that it allows functions to be passed as parameters via a mechanism known as *Jensen's device*. The classic Algol example is given by the following Algol code:

```
real procedure SIGMA(x, i, n);
   value n;
   real x; integer i, n;
begin
   real s;
   s := 0;
   for i := 1 step 1 until n do
      s := s + x;
   SIGMA := s;
end
```

To find the sum of the first 15 cubes, we can call `SIGMA(i*i*i,i,15)`. In this call, formal parameter x is replaced with `i*i*i`.

Unfortunately, call by name has some significant problems. First, it can be challenging to write even seemingly simple routines like `swap`, because of the potential of calls such as `swap(v,arr[ v])`. With call-by-name, once v is changed in the `swap` routine, it will be impossible to change the correct element in `arr`. Second, the implementation of thunks is somewhat cumbersome. And third, actual arguments are reevaluated every time the corresponding formal parameter is used, which can be very inefficient. Consequently, although Algol 60 was itself an extremely influential language, and introduced call-by-value parameter passing which is still used today, call-by-name parameter passing has not stood the test of time, and is mostly of historical interest.

Call-by-Need

Call-by-need is like call-by-name, except that when an actual argument is evaluated, its value is saved, in a process called *memoization*. If the formal parameter reappears, rather than reevaluating the actual argument, the saved value is used. In imperative languages, such as all of the languages described earlier in this article, this strategy does not work, because the value of the actual argument could change because of side effects. However, in purely functional languages, with no effects, call-by-need produces the same results as call-by-name, with each actual argument evaluated at most once (and sometimes not at all). In addition, routines such as swapping are not expressible anyway, and thus call-by-need can be practical, and is in fact implemented in some functional languages, most notably *Haskell*.

SUMMARY

Although parameter passing seems like a simple topic, in reality, many options and subtleties can emerge. One appeal of functional languages is the relatively simple syntax involved in parameter passing. C and Java limit parameter passing to call-by-value and have standard workarounds to allow call-by-reference to be simulated, and to pass functions. Ada's parameter passing is nice because it distinguishes between the mode (in, out, or in out) rather than the underlying implementation used to achieve the effect. C++ has the most complex parameter passing mechanisms, including the unfortunate requirement for the programmer to choose between call-by-value and call-by-reference to a constant. C# parameter passing blends features from Java, Ada, and C++, combining the best features.

FURTHER READING

B. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed., Englewood Cliffs, NJ: Prentice-Hall, 1988.

B. Stroustrup, *The C++ Programming Language*, 3rd ed., Reading, MA: Addison-Wesley, 1997.

Annotated Ada Reference Manual, ISO/IEC 8652:1995(E) with Technical Corrigendum 1, 2000.

J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java Programming Language Specification*, 3rd ed., Boston, MA: Addison-Wesley, 2005.

A. Hejlsberg, S. Wiltamuth, and P. Golde, *The C# Programming Language*, 2nd ed., Boston, MA: Addison-Wesley, 2006.

P. Naur, Revised Report on the Algorithmic Language ALGOL 60, *Commun. ACM*, **3**: 299–314, 1960.

S. P. Jones, *Haskell 98 Language and Libraries: The Revised Report*, Cambridge: Cambridge University Press, 2003.

R. W. Sebesta, *Concepts of Programming Languages*, 8th ed., Boston, MA: Addison-Wesley, 2008.

Mark Allen Weiss
Florida International
University
Miami, Florida

PROGRAM TRANSFORMATION: WHAT, HOW, AND WHY

WHAT: THE MANIPULATION OF COMPLEX VALUES

A typical computer program consists of a sequence of instructions that manipulate values belonging to a variety of *simple* data types. In this context, a data type is considered to be simple if its values have a simple syntactic structure. Integers, reals, Booleans, strings, and characters are all examples of simple data types. In contrast, when viewed as a value, the sequence of characters that make up a program written in a high-level language such as Java or C++ can be seen as having a highly complex syntactic structure.

Informally speaking, a good litmus test for determining whether a particular value is simple is to consider the complexity of user-defined methods capable of reading in such a value from a file, storing this value internally within a program, and writing this value to a file. Thinking along these lines reveals that typical computer languages provide input/output (I/O) support for simple types (e.g., getc, read, input1, inputN, put, print, and write) as well as primitive support for basic operations on these types (e.g., equality comparison, relational comparisons, addition, and subtraction).

A similar level of support is generally not provided for values having syntactic structures that cannot be directly modeled in terms of simple values. Thus, as the structure of the data becomes more complex, a greater burden is placed on the programmer to develop methods capable of performing desired operations (e.g., I/O, equality comparison, internal representation, and general manipulations). In the limit, the techniques employed for structure recognition include the development of domain-specific parsers, reuse of general-purpose context-free parsers such as LL, LALR, LR parsers (1), and even state-of-the-art parsers such as Scannerless Generalized LR (SGLR) parsers (2,3). The values constructed by these tools are typically output using sophisticated algorithms such as abstract pretty printers (4,5).

Parsers such as LL, LALR, LR, and SGLR parsers all ultimately make use of powerful parsing algorithms for recognizing the structure of a sequence of symbols. From a theoretical perspective, these parsing algorithms are capable of recognizing the class of languages known as context-free languages. This class of languages is interesting because it represents the most complex class that can be efficiently recognized by a computer using general-purpose algorithms. The syntactic structure of modern programming languages typically fall in the class of context-free languages or slight variations thereof (6).

Figure 1 gives an example of an extended-BNF grammar fragment describing the syntactic structure of a simple imperative language we will call *Imp*. The directives *%LEFT_ASSOC ID* and *%PREC ID* are used to declare and assign precedence and associativity to operations and productions in the grammar (for more on precedence and associativity, see Ref. 1). These assignments allow portions of the grammar that would otherwise be ambiguous to be uniquely parsed. Informally summarized, the language described by the grammar fragment defines an *Imp* program as consisting of a single block containing a statement list. In turn, a statement list consists of zero or more labeled statements. A label may be optionally associated with a statement. A statement can either be a block, one of three different kinds of "if" statements, a "while" loop, an assignment, a "goto" statment, or a statement called "skip" whose execution does nothing (i.e., "skip" is a no-op). Programs written in this language can be parsed using an LALR parser that has been extended with associativity and precedence.

As a result of their context-free roots, the structure of character sequences corresponding to typical computer programs can be modeled in terms of a tree structure (also known as a term structure). Tree structures come in two basic flavors: parse trees, which literally reflect the structure described by the context-free grammar used to define the programming language, or abstract syntax trees, which capture the essence of the structure described by the context-free grammar (for more on extended-BNF grammars and abstract syntax, see Ref. 7). More compact internal representations such as directed acyclic graphs (DAGs) are also possible; but a discussion of these representations lies beyond the scope of this article.

HOW: EQUATIONAL REASONING – THE ESSENCE OF PROGRAM TRANSFORMATION

Program transformation concerns itself with the manipulation of programs. Conceptually speaking, a (program) transformation system accepts a *source program* as its input data and produces a transformed program known as a *target program* as its output data. Thus, a transformation system treats programs in much the same way that traditional programs treat simple data. In general, systems that share this view of programs-as-data are called *meta-programming systems*. A compiler is a classic example of a meta-programming system.

In spirit, the goal in program transformation is to manipulate programs using techniques similar to the techniques used by mathematicians when they manipulate expressions. For example, in mathematics, the expression $x \wedge true$ can be simplified to x. Similarly in Java, the sequence of assignments $x = 5; x = x$ can be simplified to

This work was supported in part by the United States Department of Energy under Contract DE-AC04-94AL85000. Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy. Victor Winter was also partially supported by NSF grant number CCR-0209187.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

```
%LEFT_ASSOC L1. %LEFT_ASSOC L2. %LEFT_ASSOC L3. %LEFT_ASSOC L4. %LEFT_ASSOC L5.

pgm           ::=  { stmt_list }
stmt_list     ::=  labeled_stmt ; stmt_list | ϵ
labeled_stmt  ::=  label stmt
label         ::=  id : | ϵ
stmt          ::=  block | cond | loop | assign | jump | continue
block         ::=  { stmt_list }
cond          ::=  if E labeled_stmt
cond          ::=  if E then block                      %PREC L1
cond          ::=  if E then block else block           %PREC L2
loop          ::=  while E do block
assign        ::=  id = E
jump          ::=  goto label
continue      ::=  skip
E             ::=  E op E | ( E ) | !( E ) | base
op            ::=  < | > | <= | >= | = | !=             %PREC L3
op            ::=  + | −                                %PREC L4
op            ::=  ∗ | div | mod                        %PREC L5
base          ::=  id | integer
id            ::=  identifier
```

Figure 1. A grammar fragment of a simple imperative language called Imp.

the single assignment $x = 5$. In Boolean algebra, the expression $e1 \vee e2$ is equivalent to $e2 \vee e1$ for any arbitrary Boolean expressions $e1$ and $e2$. However, in Java, Boolean expressions are conditionally evaluated[1] and, as a result, $e1 \| e2$ is not equivalent to $e2 \| e1$ (consider the evaluation of the Boolean expression $true \| 4/0 < 5$). On the other hand, in Java, a conditional statement of the form *if (BE) stmt*$_1$; *else stmt*$_2$; is equivalent to *if (!(BE)) stmt*$_2$; *else stmt*$_1$; for any Java Boolean expression *BE* and Java statements *stmt*$_1$ and *stmt*$_2$. Having seen a few examples of manipulation, let us take a more detailed look at how mathematical expressions can be manipulated in general through a process known as *equational reasoning*.

Equational Reasoning: A Technique for Mathematical Manipulation

In mathematics, there are axioms (i.e., laws) and theorems stating how expressions of a certain type (e.g., Boolean expressions) can be manipulated. Axioms and theorems are oftentimes given in the form of equations relating two syntactically distinct expressions. Figure 2 gives a standard set of axioms defining a Boolean algebra.

The axioms for Boolean algebra provide us with the basis for manipulating Boolean expressions. In mathematics, when manipulating a mathematical expression, a common goal is the *simplification* of that expression. In math classes, problems are often given in which the goal is to simplify an expression until it can be simplified no further. This activity is referred to as *solving* the expression, and the simplified form of the expression is called the *answer*. In the context of equational reasoning, such an *answer* is called a *normal form*. For example, the *normal form* of $7 * 7 + 1$ is 50. In this article, we will use the terms *rewriting* and *simplification* interchangeably.

In addition to expression simplification, in mathematics, one is also interested in knowing whether one expression is equal to another expression. This activity is known as *theorem proving*. Theorems have the general form: $e1 = e2$ *if cond*, where *cond* defines the conditions under which $e1 = e2$ holds. In the degenerative case, where $e1 = e2$ always holds, one may drop the conditional portion and simply write the theorem as $e1 = e2$.

Suppose that one is interested in knowing whether $or(b, b) = b$ is a theorem, where $or(b, b)$ is the prefix form of the Boolean expression $b \vee b$. How does one go about proving such a theorem? One approach for proving a theorem of the form $e1 = e2$ is to separately try to *rewrite* $e1$ and $e2$ into their normal forms and then compare the results. A variation of this idea is to pick whichever term $e1$ or $e2$ is more complex and rewrite it in the hopes that it can be simplified to the other term. Having said that, we will view the proof of $or(b, b) = b$ in terms of a simplification problem. In particular, we are interested in rewriting the expression $or(b, b)$ to b, which conveniently already happens to be in its normal form, thereby proving the theorem $or(b, b) = b$. The proof of $or(b, b) = b$ is shown in Fig. 3. An important thing to note about the sequence of "simplifications" that are applied to $or(b, b)$ is that they are anything but simple. It turns out that, in the context of first order logic, there is no universal definition for the notion of *simplification* that can be used to prove all theorems. Indeed, it is well known that theorem proving in the realm of first-order logic is, in fact, undecidable. The implications of this observation is that the complete automation of Boolean simplification is not realistic.

Operationally, the simplifications shown in Fig. 3 are accomplished through a process known as *equational reasoning*, which is based on equational logic (8). Informally

[1] This form of evaluation is also referred to as short-circuiting.

Commutativity

$$or(x, y) = or(y, x)$$
$$and(x, y) = and(y, x)$$

Identity

$$or(x, false) = x$$
$$and(x, true) = x$$

Distributivity

$$or(x, and(y, z)) = and(or(x, y), or(x, z))$$
$$and(x, or(y, z)) = or(and(x, y), and(x, z))$$

Complement

$$or(x, not(x)) = true$$
$$and(x, not(x)) = false$$

Figure 2. The standard axioms for a Boolean algebra.

stated, equational reasoning is the notion that "equals may be substituted for equals"(8). The axioms of Boolean algebra shown in Fig. 2 provide us with an initial set of equal quantities in the form of equations, and it is instances of these axioms that are used in the proof shown in Fig. 3.

Equational reasoning is a cornerstone of mathematics and is an indispensable tool at the mathematician's disposal when it comes to reasoning about expressions. In theory, the concepts and mechanisms underlying equational reasoning should also be adaptable to reason about and manipulate programs. Just as in mathematics, in computer science there are axioms and theorems stating how program structures belonging to a given language relate to one another. Realizing this fact, our original definition of *program transformation* can be refined as follows:

> *Program transformation involves the discovery and development of suitable axioms and theorems and their application to programs in accordance with the laws of equational logic to achieve a particular goal.*

The Mechanism of Equational Reasoning

In order to consider manipulating programs in the way mathematicians manipulate expressions, it is helpful to first analyze and abstract the techniques and concepts underlying equational reasoning. In addition, we are interested in knowing the extent to which various techniques and processes can be automated. Ideally, we are aiming for a fully automated system that, when presented with a program and a goal (e.g., simplification), will produce an output program satisfying that goal.

Variables and Matching. In equational reasoning, the *variable* plays an important role. For example, the axioms in Fig. 2 make use of the variables x, y, and z. Variables allow equations to be written that capture general relationships between expression structures. *Matching* (8) is an activity involving variables that is very important in equational reasoning. Let *e* denote an expression we are interested in manipulating, and let $e1 = e2$ denote the equation we are considering using to manipulate *e*. Matching allows us to determine whether *e* is an instance of e_1 or e_2. For example, in the proof in Fig. 3 it is possible to rewrite *or*(*b, b*) to *and*(*or*(*b, b*), *true*) using the equation *and*(*x, true*) $=$ *x* and realizing that *or*(*b, b*) is an instance of *x* (i.e., the variable *x* can denote a quantity like *or*(*b, b*)). Similarly, it is possible to rewrite the expression *or*(*b, and*(*b, not*(*b*))) to *or*(*b, false*) by using the equation *and*(*x, not*(*x*)) $=$ *false* and realizing that the subexpression *and*(*b, not*(b)) is an instance of *and*(*x, not*(*x*)).

Let *e* denote an expression that may contain one or more variables and let *t* denote an expression containing no variables. We will write $e \ll t$ to denote the attempt to *match* *e* with *t*. We will refer to $e \ll t$ as a *match equation*. A match equation is a Boolean-valued test that either succeeds, or fails. If a match equation succeeds, then it means that *t* is an instance of *e*, which more specifically means that there exist values that when substituted for the variables in *e* will produce the expression *t*. For example, if we substitute *b* for *x* in the expression *and*(*x, not*(*x*)), we get *and*(*b, not*(*b*)), thus *and*(*x, not*(*x*)) $\ll$ *and*(*b, not*(*b*)) succeeds under the substitution $x \mapsto b$. Substitutions are abstractly denoted by the symbol σ. The act of replacing the variables in an expression *e* as defined by is known as *applying* the substitution σ to *e* and is written $\sigma(e)$.

Matching-related concepts have been heavily researched. Under suitable conditions, it is appropriate to use more powerful algorithms to construct an expression that is an instance of two other expressions. These algorithms include *unification* (9), *AC-matching* (10), *AC-unification* (11), and even *higher-order unification and matching* (12).

Equation Orientation, Confluence, and Termination. Given an expression *t*, a crucial aspect of equational reasoning is how one makes the decision regarding which equation should be used to simplify *t* or one of its subexpressions. In the realm of rewriting, the complexity of the decision-making process has been simplified by orienting equations. For example, instead of writing $e_1 = e_2$, one would write $e_1 \rightarrow e_2$. An oriented equation of the form $e_1 \rightarrow e_2$ is called a *rewrite rule*. The orientation $e_1 \rightarrow e_2$ constrains the equational reasoning process to the replacement of instances of e_1 by instances of e_2 and not the other way around[2].

Orienting equations into rewrite rules greatly simplifies the task of deciding which rewrite rule should be applied to a given term. However, equation orientation does not eliminate the decision altogether. In general, expressions still exist to which two or more competing rules apply (see the next subsection for more details on rule application). Under such conditions, we say that the rules *interfere* with one another. The simplest example of a pair of interfering rules are two rewrite rules having identical left-hand sides (e.g., $e_1 \rightarrow e_2$ and $e_1 \rightarrow e_3$). Ideally, we would like to have a set of rules that do not interfere with each other, or at least know that if rules do interfere with one another the interference somehow does not matter. A consequence of the notion of "interference not mattering" is that the normal form for an

[2] A discussion of the techniques used to decide how equations should be oriented lies beyond the scope of this article.

$or(b, b)$	Given
$and(or(b, b), true)$	*Identity*: $x \rightarrow and(x, true)$ where x is $or(b, b)$
$and(or(b, b), or(b, not(b)))$	*Complement*: $true \rightarrow or(x, not(x))$ where x is b
$or(b, and(b, not(b)))$	*Distributivity*: $and(or(x, y), or(x, z)) \rightarrow or(x, and(y, z))$ where x is b, y is b and z is $not(b)$
$or(b, false)$	*Compliment*: $and(x, not(x)) \rightarrow false$ where x is b
b	*Identity*: $or(x, false) \rightarrow x$ where x is b

Figure 3. An example of axiom-based manipulations of Boolean expressions.

expression, when it exists, must be unique. In general, rule sets having the property of "interference not mattering" are said to be *confluent* or equivalently *Church–Rosser* (8,13). Formally, the *Church–Rosser* property is defined as $e_1 \overset{*}{\leftrightarrow} e_2 \Rightarrow e_1 \downarrow e_2$. Informally, this property means that expressions that are equal can always be *joined* through the application of rewrite rules (i.e., oriented equations) in the (Church–Rosser) rule set. In other words, given a rule set R, we say that two expressions can be *joined* if they both can be rewritten to the same expression using only the rewrite rules found in R.

An important result concerning the confluence/Church–Rosser property is that it is possible to mechanically check whether a rule set possesses this property. It is also possible in certain cases to convert a rule set that is not confluent into an equivalent rule set that is confluent (8).

Confluence is a highly desirable property for a rule sets to possess because it implies that the decision of which order rules should be applied during the course of an equational reasoning session is immaterial. Thus, the algorithm driving the equational reasoning process is trivial, one simply applies rules where ever and whenever possible secure in the knowledge that the rewriting process will always arrive at the same *normal form*, when it exists.

When does a normal form not exist? Given a confluent rule set, the only circumstances under which a normal form does not exist is if the rule set is nonterminating. For example, consider the rule set consisting of the single rule $x \rightarrow f(x)$. This rule set is trivially confluent but is nonterminating and therefore produces no normal forms. Using this rule set to "simplify" the expression b will yield the nonterminating sequence of rewrites $b \rightarrow f(b) \rightarrow f(f(b))) \rightarrow \ldots$. A rule set is said to be *terminating* if every simplification sequence eventually produces a normal form. The combination of confluence and termination let us conclude that all expressions have a normal form and that their normal forms are unique.

In general, the problem of showing that a rule set is terminating is undecidable. However, in practice one can often show that a particular rule set is terminating. As a result of the highly desirable properties of rule sets that are confluent and terminating, the termination problem is a heavily researched area in the field of rewriting (8).

Rule Extensions and Application. The basic notion of a rewrite rule can be extended in two important ways. The first extension allows a label to be associated with a basic rewrite rule. The result is called a *labeled rewrite rule*. Labeled rewrite rules typically have the form *label*: $lhs \rightarrow rhs$, where lhs and rhs are expressions. A transformation system supporting labeled rewrite rules allows the option of labeling rewrite rules and treats a reference to a label as a shorthand for a reference to the rule.

In the second extension, a labeled rewrite rule can be extended with a condition. The result is called a *labeled conditional rewrite rule*. Conditions can take on a number of forms, but all ultimately can be understood as a Boolean condition that enables or prohibits a rewrite rule from being applied. Consider the rule $x/x \rightarrow 1$ *if* $x \neq 0$. In this article, a labeled conditional rewrite rule has the form *label*: $lhs \rightarrow rhs$ *if condition*. We will also only consider a restricted form of condition consisting of Boolean expressions involving *match equations* as defined in the variables and matching subsection.

Let r denote an arbitrary rewrite rule and let e denote an expression. If r is used as the basis for performing a manipulation of e, we say that r is applied to e, which is what we mean when we say *rule application*. More specifically, when using a conditional rewrite rule of the form $lhs \rightarrow rhs$ *if cond* to simplify an expression t, one first evaluates the Boolean expression $lhs \ll t \wedge cond$. If this Boolean expression evaluates to *true* and produces the substitution σ, then t is *rewritten* to rhs', where $rhs' = \sigma(rhs)$ is the instance of rhs obtained by applying the substitution σ to the expression rhs.

Program Fragments as "Expressions". Thus far, we have given an overview of the mechanisms underpinning rewriting. However, we have not said much about notations for describing expressions. When manipulating Boolean expressions, the choice of notation is fairly straightforward. One can, for example, write a Boolean expression in infix form $e_1 \vee e_2$ or in prefix form $or(e_1, e_2)$. How do these ideas translate to program structures? One possibility is to express code fragments in prefix form. However, there are some disadvantages to such an approach. One disadvantage is that there is some notational complexity associated with prefix forms because it is not how we write programs in general. This conceptual gap holds in the realm of Boolean algebra as well. For example, most readers will probably find $x \vee y \wedge z$ to be more readable than $or(x, and(y, z))$. This problem is amplified as the complexity of the structure expressed increases (and code fragments can have a complex structure). To address the comprehensibility problem, we will express code fragments in an infix form that we call a *parse expression* (14,15). A parse expression is essentially a shorthand for a parse tree and assumes that the syntax of the programming language has been defined by an extended-BNF. In general, a parse expression has the form $B[\![\alpha']\!]$, where B is a nonterminal in the gram-

$$
\begin{array}{lll}
\mathit{assign\_simplify1} & : & \mathit{assign}\llbracket \mathit{id}_1 = (E_1)\rrbracket \longrightarrow \mathit{assign}\llbracket \mathit{id}_1 = E_1\rrbracket \\
 & & \\
\mathit{assign\_simplify2} & : & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{id}_1 = !(E_1)\ ;\ \mathit{stmt\_list}_1\rrbracket \ \longrightarrow \\
 & & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{id}_2 = E_1;\ \mathit{id}_1 =!(\mathit{id}_2);\ \mathit{stmt\_list}_1\rrbracket \\
 & \text{if} & \neg(E_1 \ll E\llbracket \mathit{base}_2\rrbracket) \wedge \mathit{id}_2 \ll \mathit{new} \\
 & & \\
\mathit{assign\_simplify3} & : & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{id}_1 = E_2\ \mathit{op}_1\ E_3;\ \mathit{stmt\_list}_1\rrbracket \ \longrightarrow \\
 & & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{id}_2 = E_2; \mathit{id}_3 = E_3; \mathit{id}_1 = \mathit{id}_2\ \mathit{op}_1\ \mathit{id}_3;\ \mathit{stmt\_list}_1\rrbracket \\
 & \text{if} & \neg(E_2 \ll E\llbracket \mathit{base}_2\rrbracket\ \wedge E_3 \ll E\llbracket \mathit{base}_3\rrbracket) \wedge \mathit{id}_2 \ll \mathit{new} \wedge \mathit{id}_3 \ll \mathit{new} \\
 & & \\
\mathit{jump1} & : & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{if}\ \ E_1\ \mathit{then}\ \mathit{block}_1;\ \mathit{stmt\_list}_1\rrbracket \ \longrightarrow \\
 & & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{if}\ \ !(E_1)\ \mathit{goto}\ \mathit{id}_1;\ \mathit{block}_1;\ \mathit{id}_1 : \mathit{skip};\ \mathit{stmt\_list}_1\rrbracket \\
 & \text{if} & \mathit{id}_1 \ll \mathit{new} \\
 & & \\
\mathit{jump2} & : & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{if}\ \ E_1\ \mathit{then}\ \mathit{block}_1\ \mathit{else}\ \mathit{block}_2;\ \mathit{stmt\_list}_1\rrbracket \ \longrightarrow \\
 & & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{if}\ \ !(E_1)\ \mathit{goto}\ \mathit{id}_1; \\
 & & \qquad\qquad \mathit{block}_1;\ \mathit{goto}\ \mathit{id}_2; \\
 & & \qquad\qquad \mathit{id}_1 : \mathit{block}_2; \\
 & & \qquad\qquad \mathit{id}_2 : \mathit{skip};\ \mathit{stmt\_list}_1\rrbracket \\
 & \text{if} & \mathit{id}_1 \ll \mathit{new} \wedge \mathit{id}_2 \ll \mathit{new} \\
 & & \\
\mathit{jump3} & : & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{while}\ E_1\ \mathit{do}\ \mathit{block}_1;\ \mathit{stmt\_list}_1\rrbracket \ \longrightarrow \\
 & & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{skip}; \\
 & & \qquad\qquad \mathit{id}_1 : \mathit{if}\ !(E_1)\ \mathit{goto}\ \mathit{id}_2; \\
 & & \qquad\qquad \mathit{block}_1;\ \mathit{goto}\ \mathit{id}_1; \\
 & & \qquad\qquad \mathit{id}_2 : \mathit{skip};\ \mathit{stmt\_list}_1\rrbracket \\
 & \text{if} & \mathit{id}_1 \ll \mathit{new} \wedge \mathit{id}_2 \ll \mathit{new} \\
 & & \\
\mathit{simplify\_if} & : & \mathit{stmt\_list}\llbracket \mathit{label}_1\ \mathit{if}\ \ E_1\ \mathit{labeled\_stmt}_1;\ \mathit{stmt\_list}_1\rrbracket \ \longrightarrow \\
 & & \mathit{stmt\_list}\llbracket \mathit{id}_1 = E_1; \mathit{label}_1\ \mathit{if}\, \mathit{id}_1\ \mathit{labeled\_stmt}_1;\ \mathit{stmt\_list}_1\rrbracket \\
 & \text{if} & \neg(E_1 \ll E\llbracket \mathit{base}_1\rrbracket) \wedge \mathit{id}_1 \ll \mathit{new}
\end{array}
$$

Figure 4. Rewrite rules capable of transforming Imp source programs into equivalent target programs.

mar and the derivation $B \overset{+}{\Rightarrow} \alpha$ is possible. The difference between α as it occurs in the derivation and α' as it occurs in the parse expression is that in α' all nonterminal symbols have been subscripted, making them *variables*. In particular, when we say *variable* we mean a symbol that can participate in matching as described.

Let us consider the grammar fragment for Imp shown in Fig. 1. The parse expression $\mathit{assign}\llbracket \mathit{id}_1 = E_1\rrbracket$ denotes a parse tree whose root is the nonterminal *assign* and whose leaves are id_1, $=$, and E_1. As id_1 and E_1 are variables, this parse expression denotes the most general form of an assignment statement. The expression $\mathit{assign}\llbracket \mathit{id}_1 = E_1\ \mathit{op}_1\ E_2\rrbracket$ denotes a less general form of an assignment in which an identifier id_1 is bound to an expression $E_1\, \mathit{op}_1\, E_2$, that is, an expression containing a least one binary operator.

Matching works for parse expressions just as would be expected. For example, the match equation $\mathit{assign}\llbracket \mathit{id}_1 = E_1\rrbracket \ll \mathit{assign}\llbracket x = 5 + 4\rrbracket$ succeeds with the substitution $\mathit{id}_1 \mapsto \mathit{id}\llbracket x\rrbracket$ and $E_1 \mapsto E\llbracket 5 + 4\rrbracket$. Similarly, the match equation $\mathit{assign}\llbracket \mathit{id}_1 = E_1\ \mathit{op}_1\ E_2\rrbracket \ll \mathit{assign}\llbracket x = 5 + 4\rrbracket$ also succeeds with the substitution $\mathit{id}_1 \mapsto \mathit{id}\llbracket x\rrbracket, E_1 \mapsto E\llbracket 5\rrbracket$, and $E_2 \mapsto E\llbracket 4\rrbracket$. We are now ready to look at a more concrete example of program transformation.

Example: A Pseudo-Compiler for Imp

A compiler takes a source program as input and produces an assembly program as output. As such, a compiler is a meta-programming system. In this section, we look at an example of how an Imp program can be partially compiled via rewriting. The goal in our example is to take an Imp *source program* and transform it into an Imp *target program*. We claim, without proof, that the rewrite rules presented for accomplishing this goal are both confluent and terminating. The normal form of an Imp *source program* is an Imp *target program*, and it can be obtained by the exhaustive application of the labeled conditional rewrite rules shown in Fig. 4.

In order to be considered a *target program*, an Imp program should satisfy the following properties:

- All expressions in the target program should be *simple expressions*. An expression is a *simple expression* if it satisfies one of the following properties: (1) the expression consists solely of a base value (i.e., either an integer or an identifier), (2) the expression consists of a binary operation involving two base values (e.g., $15 + 27$), or (3) the expression consists of a unary operation on a base value (e.g., $!(x)$). All other expressions are not simple.
- A target program may contain no "*while*" *loops*.

[3] The ability to generate a *new* identifier name is supported by most program transformation systems.

$\boxed{x = 3 + 4}$ $\Longrightarrow$

```
LOAD 3, R1
LOAD 4, R2
ADD R1, R2, R3
STORE R3, x
```

Figure 5. An example of how an assignment statement in an Imp target program can be transformed into a sequence of assembly instructions.

- A target program may contain no *"if-then"* or *"if-then-else"* statements, which makes the *"if" statement* the only remaining conditional construct.
- The Boolean expression associated with the *"if" statement* must be an identifier (e.g., it may not be an expression of the form e1 *op* e2).

As a result of their simple structure, Imp target programs are similar to assembly programs. In fact, Imp target programs are just one step away from assembly programs and can be transformed into assembly programs on a statement by statement basis. Figure 5 gives an example of how an assignment statement can be directly transformed into a sequence of assembly instructions.

We hope the reader is convinced by this concrete example that the bulk of the general transformation from Imp target programs to assembly code is straightforward. Thus, we return our attention to the problem of transforming Imp source programs into Imp target programs.

Figure 6 shows an Imp source program and the target program that is obtained after applying the labeled conditional rewrite rules given in Fig. 4. In Fig. 4, the rewrite rules *assign_simplify1*, *assign_simplify2*, and *assign_simplify3* collectively account for the three cases that need to be considered when simplifying an expression in the context of an assignment statement. The rule *assign_simplify1* is a conditional rule that removes (unnecessary) outermost parenthesis from an expression. The rule *assign_simplify2* transforms the assignment of an identifier to a negated expression into a sequence of two assignment statements, provided the negated expression is not a base value. For example, the assignment $x = !(3 < 4)$ will be transformed to $x\_1 = 3 < 4; x = !(x\_1)$, where x_1 is a *new* identifier. Notice that to carry out this kind of manipulation, one must have the ability to generate a *new* (heretofore unused) identifier. In the rewrite rules shown, this functionality is realized by the function *new*, which we do not discuss further in this article[3]. And lastly, note that without the conditional check $\neg\,(E_1 \ll E[\![base_1]\!])$, the rule *assign_simplify2* would be nonterminating.

The rule *assign_simplify3* transforms an assignment statement containing a nonsimple expression (e.g., an expression containing two or more binary operators) into a sequence of three assignment statements. For example, the assignment $x = 4 + 5 * 6 * 7$ would be rewritten into the assignment sequence $x\_1 = 4$; $x\_2 = 5 * 6 * 7; x = x\_1 + x\_2$. Notice that the assignment $x\_2 = 5 * 6 * 7$ still contains a complex expression and will again be simplified by the *assign_simplify3* rule. In the rule *assign_simplify3*, the parse expression $stmt\_list[\![id_1 = E_2\, op_1 E_3; stmt\_list_1]\!]$ denotes a statement list whose first statement is the assignment of the form $id_1 = E_2\, op_1\, E_3$. Analysis of the problem shows that matching this structure is a necessary but not sufficient condition to ensure that an expression is not simple. In order for an expression to be not simple, it must also not be the case that both E_2 and E_3 are *base* structures. Formally, this property is captured in the conditional portion of *assign_simplify3* by the Boolean expression $\neg\,(E_2 \ll E[\![base_2]\!] \wedge E_3 \ll E[\![base_3]\!])$. The remaining portion of the condition $id_2 \ll new \wedge id_3 \ll new$ is

```
{
  x = 27;
  while x! = 1 do {
    if x mod 2 == 0 then { x = x div 2;}
    else { x = 3 * x + 1; };
}
```

$\Longrightarrow$

```
{  x = 27;
   skip;
   L_1 : x_8 = x != 1;
   y_3 = !( x_8 );
   if y_3 goto L_2;
   {  x_11 = x mod 2;
      x_12 = 0;
      x_10 = x_11 == x_12;
      y_9 = !( x_10 );
      if y_9 goto L_4;
      { x = x div 2; };
      goto L_5;
      L_4 : {
         x_6 = 3 * x;
         x_7 = 1;
         x = x_6 + x_7;
      };
      L_5 : skip;
   };
   goto L_1;
   L_2 : skip;
}
```

Figure 6. An Imp source program and an equivalent Imp target program.

responsible for binding the variables id_2 and id_3 to *new* identifier names (e.g., $id_2 \mapsto id[\![x\_1]\!]$).

The remaining rules in Fig. 4 make use of notational constructs similar to those we have just discussed. The rules *jump1*, *jump2*, and *jump3* are respectively responsible for rewriting "*if-then*" statements, "*if-then-else*" statements, and "*while*" loops into equivalent sequences consisting of "*if-statements*", *labels*, "*goto*" statements, and "*skip*" statements. Here, the "*skip*" statement is used to provide a point, beyond a given block, to which a "goto" can jump. In many cases, additional optimizing transformations can be applied to remove unneeded "skip" statements. However, the "*skip*" statement cannot be removed entirely (consider the case where the last portion of a program is a block that one wants to jump over).

And lastly, the *simplify_if* rule makes sure that the Boolean condition associated with an "*if*" *statement* consists of a base value.

Program Transformation Frameworks

The Equation orientation, confluence, and termination subsection mentioned that confluence and termination are highly desirable properties for rule sets because the problem of deciding which rule to apply then becomes immaterial. Unfortunately, when transforming programs it is often the case that rewrite rules are created that are neither confluent nor terminating and cannot be made so. Under these conditions, if transformation is to succeed, explicit control must be exercised over when, where, and how often rules are applied within a term structure. A specification of such control is referred to as a *strategy*, and systems that provide users with constructs for specifying control are known as *strategic programming systems*.

The control mechanisms in a strategic programming system fall into two broad categories: *combinators* and *traversals*. The computational unit in a rewrite system is the *rewrite rule*. Similarly, the computational unit in a strategic programming system is the *strategy*. A strategy can be inductively defined as follows:

- A rewrite rule is a strategy.
- A well-formed expression consisting of strategies, combinators, and traversals is a strategy.

Of central importance to a framework exercising explicit control over the application of rules is the ability to observe the outcome of the application of a rule to a term. Specifically, to exercise control, a system needs to be able to answer the question "Did the application of rule *r* to term *t* *succeed* or *fail*?" In summary then, a strategic programming system can be thought of as a rewriting system that has been extended with mechanisms for explicitly controlling the application of rules where the notion of *failure* plays a central role.

Strategic Combinators. A *combinator* is an operator (generally unary or binary) that can be used to compose one or more *strategies* into a new *strategy*. Let s_1 and s_2 denote two strategies. Typical combinators include

- *sequential composition* denoted $s_1; s_2$. The application of $s_1; s_2$ to a term t will first apply s_1 to t and then apply s_2 to the result.
- *left-biased choice* denoted $s_1 <+ s_2$. When applied to a term t, the strategy $s_1 <+ s_2$ will first try to apply s_1 to t; if that succeeds and produces the result t', then t' is the result of applying $s_1 <+ s_2$ to t. Otherwise, s_2 is applied to t. If this application succeeds and produces t'' as it's result, then t'' is the result of applying $s_1 <+ s_2$. However, if the application of s_2 to t fails, then the application of $s_1 <+ s_2$ is said to fail.
- *right-biased choice* denoted $s_1 +> s_2$. The strategy $s_1 +> s_2$ is equivalent to $s_2 <+ s_1$.
- *nondeterministic choice* denoted $s_1 + s_2$. If both s_1 and s_2 can be applied to a term t, then s_1 or s_2 is nondeterministically chosen and applied to t. If only one strategy can be applied, that strategy is selected, and if both strategies do not apply, then the application of $s_1 + s_2$ to the term t fails.

Traversals. The combinators described in the previous subsection provide the ability to discriminate and sequence the application of strategies to a term. When a strategy contains a combinator, the application of that strategy to a term is defined with respect to the structure of the strategy, irrespective of the structure of the term. In contrast, a *traversal* focuses on the structure of the term, but does not consider the structure of the strategy. Broadly speaking, a traversal specifies the order in which a term and its subterms are visited. Thus, a traversal can be understood as a mechanism for sequencing term structures. Typically, when a term is visited, some action is performed like the application of a strategy to the term.

Some traversals capture sequencing notions that are broadly applicable across a wide range of applications. Such traversals are called *generic traversals*. A typical and very useful generic traversal is one that performs a top-down left-to-right traversal of a tree structure and uniformly applies a given strategy to all subtrees encountered. Another generic traversal is one that performs a bottom-up left-to-right traversal of a tree structure. And a third generic traversal is one in which the traversal is stopped after the first successful application of a given strategy. Other generic traversals have been identified in the literature (14,16,17).

In some cases, the notion of generic traversal has direct analogies with traditional models of computation. For example, a top-down (outside-in) approach to evaluation corresponds to a lazy evaluation style where functions are applied to arguments without (first) evaluating the arguments.

In contrast, a bottom-up (inside-out) approach corresponds to a strict evaluation where the arguments to functions are evaluated before functions are applied.

Strategic Frameworks. In addition to the combinators and traversals, strategic programming frameworks may contain a variety of additional features. These features can include (1) the ability to create rewrite rules and strategies dynamically (14,15,18), (2) the ability to define strategy application via *congruences*(16,19), and (3) constructs that allow user-defined generic traversals to be created (14,18).

ELAN (16), TL (14), the ρ-calculus (14), the S'_{γ}-calculus, and Stratego (20) are examples of strategic programming frameworks. Of these frameworks, ELAN and Stratego have implementations, and a dialect of TL is implemented in a system called HATS (21).

WHY: APPLICATIONS

Abstractly, a program transformation system can be viewed as a system that transforms a *source program* into a *target program*. In Ref. 22, an excellent overview is given of a wide variety of software-related activities that can be approached from a transformation-oriented perspective. Activities are broadly classified as either belonging to the category of *rephrasing* or *translation*. In this section, we present a taxonomy similar to the one given in Ref. 22, but with a greater emphasis placed on semantics. In particular, our taxonomy is motivated by the relationship between the semantic models necessary to understand the source and target programs. Within this classification system, we identify seven major bi-directional goals of program transformation:

- *Clarity*. This goal focuses on separation and encapsulation of functional and behavioral concerns.
- *Efficiency*. This goal focuses on changing the resource usage of an executing program. Resources of primary concern are time and space.
- *Computability*. This goal focuses on the translation between noncomputable and computable program representations. Technically speaking, the goal of a compiler is to take a source program that cannot be directly executed on a computer and translate it into a target program that can be executed on a computer. In most cases, this goal involves moving between semantic models at two different levels of abstraction.
- *Simplicity*. This goal focuses on transforming a source program to a target program where the semantic model for the source program is either a subset or superset of the semantic model of the target program.
- *Functionality*. This goal focuses on changing the functional behavior of the source program. The semantic model for the source and target program are the same.
- *Translation*. This goal focuses on transforming a source program into an equivalent target program having a different syntax and generally a different semantic model. Here, both semantic models are roughly at the same level of abstraction.
- *Computation*. This goal focuses on using transformations to perform computations, that is, one is interested in some form of evaluation of a program or expression.

Transformations that Shift between Semantic Models

Compilation is a classic example of a fully automatic transformation whose source and target programs are understood with respect to different semantic models. The goal is *computability*. Source programs define computations that are typically understood in terms of semantic models containing high-level concepts such as variables, data structures, and recursion whereas target programs define computations that are understood in terms of semantic models consisting of registers, memory locations, bytes, and bits.

Synthesis and *refinement* are two examples of activities in which source programs having specification-like characteristics are transformed into executable implementations. The goal is *computability*. Transformations in this realm are typically not fully automatic (otherwise they would be called compilers) and require some form of attention on a per problem basis. Specification languages can, and oftentimes do, make use of constructs that are not computable. Thus, the semantic shift between source and target programs can be dramatic.

Migration is an activity in which a program written in one language is transformed into an equivalent program written in another language where both the source and target languages are roughly at the same level of abstraction (e.g., C++ and Java). The goal is *translation*. Such transformation can involve subtle shifts in semantic models. For example, the expression $(x{+}{+}){+}x$ has a precise semantics in Java and is, technically speaking, undefined C++.

Aspect-oriented programming is a paradigm in which cross cutting aspects of software are defined separately (23,24). These aspects are then *woven* into a base program that can then be compiled and executed in a traditional fashion. The *weaving* of aspects into a program is typically approached from a transformation-oriented perspective. The goal in *weaving* is *translation*.

Transformations that Remain within a Single Semantic Model

In *partial evaluation* (25), knowledge that a general-purpose source program will be used in a context where one or more of its inputs are fixed is used as the basis for transformation. The goal is *computability*. In particular, the target program produced is one in which all computations that can be performed statically have been carried out, which oftentimes results in a dramatic improvement in the efficiency of the resulting program.

Desugaring is an activity in which the goal of transformation is *simplification*. In desugaring, the target program that is produced belongs to a language that is a strict subset of the language of the source program. The pseudo-compiler example given earlier is an example of a desugaring transformation.

Renovation is an activity focusing on altering the behavior of a software system that is currently in use. The goal is *functionality*. The need for *renovation* is driven by changing requirements that are placed on the software.

Program *optimization* is a highly researched area in computer science. The goal in optimization is *efficiency*. Optimizations can occur at a variety of abstraction levels. A classic example can be found in Ref. 26 where an exponential algorithm for calculating Fibonacci numbers is transformed into a linear time algorithm. Well-known optimizations include constant propagation, constant folding, strength reduction, and common subexpressions elimination (1).

In the following sections, we take a more in-depth look at two transformational activities that are, in some sense, at the opposite ends of the conceptual spectrum.

Refactoring. When developing software, it is common to reach a point where some unanticipated structural or functional dependencies make the resulting software architecture difficult to understand or resistant to future modification. When such a point is reached, programming effort needs to be expended to "clean up" the software. *Refactoring* is the term used to describe general techniques and methods that can be used to "clean up" software. More formally stated, the goal of refactoring is to restructure software to make it clearer (e.g., improving its design) while preserving its functionality. In contrast, the goal in *obfuscation* is to make software harder to understand.

Examples of refactoring range from simple to complex and include:

- *Identifier renaming.* The goal of identifier renaming is to give a identifier a new name that more accurately describes its purpose.
- *Method extraction.* The goal of method abstraction is to abstract a sequence of statements into a method.
- *Object-oriented generalization.* The goal of generalization is to identify a collection of classes that share common features (e.g., methods and fields) and to migrate these common features to a super class.
- *Object-oriented specialization.* The goal of object-oriented specialization is to identify a class containing a general abstraction whose realization consists of a number of distinct special cases. When such a class is discovered, a number of subclasses should be generated and each special case should be migrated into its own subclass.

Ideally, refactoring is accomplished by carrying out a sequence of small transformations each of which are so simple that they are "obviously" correctness-preserving. In addition to simplicity, these transformations also should build on one another in a cumulative fashion. Under these circumstances, a sequence of simple transformations can have an overall effect that results in a dramatic refactoring of the program. In many cases, refactoring is subjective activity. As a result, the ideal refactoring system is one that has an interactive dimension to it allowing users to actively participate in the refactoring process. Furthermore, such a system should support an undo operation that allows refactorings to be retracted, thereby allowing a variety of refactoring possibilities to be explored.

William Opdyke's PhD thesis (27) is generally cited as the first major work that extensively looks at software refactoring as an area of research in its own right. However, in spite of this origination, the importance and implications of refactoring were not fully appreciated until popularized by Martin Fowler et al. in a book titled *Refactoring–Improving the Design of Existing Code* (28). Since then, software refactoring has become widespread. A number of tools are available to help software developers perform refactorings. Among these tools are Transmorgrify, Eclipse, RECODER, and RefactorIT. Refactoring has also been identified as an essential component of extreme programming (29).

The Evaluation of λ-expressions. Functional programming languages have their origins in a formalism known as the λ-calculus (30). The syntax of the λ-calculus is extremely simple. The elements of the language of λ-calculus are called λ-expressions or expressions for short. A λ-expression can be a constant, a variable, the application of one λ-expression to another λ-expression, or a λ-abstraction of the form $(\lambda\, id.E)$, where *id* is an identifier and E is a λ-expression.

The λ-calculus is a powerful notation for describing general-purpose computation. In fact, it has been shown that any computable function can be described in terms of a λ-calculus expression. In this framework, computation consists of the evaluation of λ-expressions. The goal in an evaluation is to simplify a λ-expression until it can be simplified no further. If such a point is reached, we say the expression is in its *normal form*.

The manipulation of λ-expressions is governed by the three axioms shown below. The first two axioms make use of an operation that substitutes a value for all free occurrences of a variable within a λ-expression. Let E denote a λ-expression, let x a variable, and let v denote a value (i.e., a λ-expression). The expression $E[x \mapsto v]$ denotes the instance of E that is obtained by replacing all free occurrences of x in E by v. The first and third axioms make use of the ability to determine whether a variable *occurs free* within a λ-expression. The formal definitions of $E[x \mapsto v]$ and *occurs free* are straightforward but lie beyond the scope of this article. For more information, see Ref. 30.

- **Axiom 1.** *Alpha-conversion (variable renaming).* $\lambda x.E \underset{\alpha}{\leftrightarrow} \lambda y.\ E[x \mapsto y]$*provided* y *does not occur free in* E.
- **Axiom 2.** *Beta-conversion (function application).* $(\lambda x.E_1)E_2 \underset{\beta}{\leftrightarrow} E[x \mapsto E_2]$.
- **Axiom 3.** *Eta-conversion (redundant layers of λ-abstraction).* $(\lambda x.F\,x) \underset{\eta}{\leftrightarrow} F$ *provided* x *does not occur free in* F *and* F *is a λ-abstraction.*

The equivalences in these axioms can be oriented from left to right to form corresponding *reductions* or rewrite rules. A λ-expression is simplified by applying reductions until the *normal form* of the expression is reached. When reducing λ-expressions, the workhorses of reduction are the rewrite rules derived from the second and third axioms, and a λ-expression to which these rules can be applied is called a *redex*.

An important corollary to a famous theorem known as the *Church–Rosser Theorem* states that normal forms for λ-expressions are unique (up to variable renaming). Given the knowledge of the uniqueness of normal forms, an interesting question to ask is: "Given a λ-expression E, can the normal form of E be reached by applying reductions in any order to any subexpression of E?" A second *Church–Rosser* theorem states that one is guaranteed to reach the normal form of a λ-expression (if it exists) by always reducing the left-most, outer-most redex, and only applying α-conversion when needed to avoid the *name capture problem* (see Ref. 30 for more on the name capture problem).

SUMMARY AND CONCLUSION

In this article, *program transformation* is defined as a mechanism for manipulating programs (and other software artifacts) having its roots firmly grounded in *equational reasoning*. On an intuitive level, equational reasoning can be thought of as the notion that “equals can be replaced by equals” (3). Formalization of this notion makes use of concepts such as (1) *matching/unification*, (2) *confluence*, and (3) *termination*. The practical adaptation of the ideas underlying equational reasoning to the realm of meta-programming (i.e., program transformation) requires the use of parsing technology to automatically recognize the complex term structures that are typically possessed by computer programs. These term structures can be defined using context-free grammars and can be stored internally by the transformation system as (1) *parse trees*, which directly reflect the structure defined by the grammar; (2) *abstract syntax trees*, which capture the essence of the structure described by the context-free grammar; or even (3) DAGs.

Applications lending themselves to a transformational perspective can be found in numerous areas including: compilation, refactoring, synthesis, refinement, and even computation.

Interest in program transformation is driven by the idea that, through their repeated application, a set of simple rewrite rules can affect a major change in a software artifact. From the perspective of dependability, the explicit nature of transformation exposes the software development process to various forms of analysis that would otherwise not be possible.

FURTHER READING

T. Mens and T. Tourw, A Survey of Software Refactoring.*IEEE Trans. on Softw. Eng.*, **30** (2): 126–129, 2004.

V. L. Winter, S. Roach, and G. Wickstrom, Transformation-Oriented Programming: A Development Methodology for High Assurance Software, in M. Celkwoitz, (ed.), *Advances in Computers*, vol. 58, Academic Press, Amsterdam.

BIBLIOGRAPHY

1. A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers Principles, Techniques, and Tools*, Reading, M.A.: Addison Wesley, 1988.
2. M. Tomita, *Efficient Parsing for Natural Languages – A Fast Algorithm for Practical Systems*. Dordreoht, The Netherlands: Kluwer Academic Publishers, 1986.
3. M. van den Brand, A. Sellink, and C. Verhoef, *Current Parsing Techniques in Software Renovation Considered Harmful*, Ischia, Italy,1998.
4. R. D. Cameron, An abstract pretty printer. *IEEE Softw.*, **5** (6): 61–67, 1988.
5. L. F. Rubin, Syntax-directed pretty printing a first step towards a syntax-directed editor, *IEEE Trans. Softw. Eng.*, SE-**9** (2): 119–127, 1983.
6. J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction fo Automata Theory, Languages, and Computation*, 2nd ed., Reading, M.A., Addison Wesley, 2001.
7. R. Stansifer, *The Study of Programming Languages*, Englewood Cliffs, N.J: Prentice Hall, 1995.
8. F. Baader and T. Nipkow, *Term Rewriting and All That*, Cambridge, U.K.: Cambridge University Press, 1998.
9. A. Martelli and U. Montanari, An efficient unification algorithm, *ACM Trans. Prog. Lang. Syst.*, **4** (2): 258–282, 1982.
10. S. Eker, Associative-commutative matching via bipartite graph matching. *Comp. J.*, **38** (5): 381–399, 1995.
11. D. Kapur and P. Narendran, *Double-exponential Complexity of Computing a Complete Set of AC-Unifiers*. Logic in Computer Science (LICS), Santa Cruz, CA, June 1992.
12. G. Dowek, Higher-order unification and matching, *in Handbook of Automated Reasoning*, Vol. 2, 2001, pp. 1009–1062.
13. A. V. Aho, R. Sethi, and J. D. Ullman, *Code Optimization and Finite Church-Rosser Systems*. Design and Optimization of Compilers, R. Rustin, (ed.), Englewood CIiffs, NJ: Prentice Hall, 1972, pp. 89–106.
14. V. L. Winter and M. Subramaniam, *The Transient Combinator, Higher-Order Strategies, and the Distributed Data Problem.* Sci. Comp. Prog., **52**: 165–212, 2004.
15. V. L. Winter, *Strategy Construction in the Higher-Order Framework of TL*. The 5*th* International Work-shop on Rule-Based Programming (RULE 2004), *Electr. Notes Theor. Comput. Sci.*, **124**: 141–170, 2005.
16. H. Cirstea and C. Kirchner, *Intoduction to the rewriting calculus*. INRIA Research Report RR-3818, December 1999.
17. R. Lämmel, Typed generic traversal with term rewriting strategies. J. Logic, Algebra. Prog., **54**: 1–64, 2003.
18. E. Visser, Scoped dynamic rewrite rules, in M. van den Brand and R. Verma, (eds.), Rule Based Programming (RULE’01), volume 59/4 of Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers, New York, September 2001.
19. P. Borovansky, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen, An overview of ELAN, in C. Kirchner, and H. Kirchner, (eds.), *International Workshop on Rewriting Logic and its Applications, vol. 15 of Electronic Notes in Theoretical Computer Science*, France: Elsevier Science, New York, 1998.
20. E. Visser, Z. Benaissa, and A. Tolmach, Building Program Optimizers with Rewriting Strategies, *Proc. Third ACM SIGPLAN International Conference on Functional Programming (ICFP’98)*, 1998.
21. HATS. Available http://faculty.ist.unomaha.edu/winter/hats-uno/HATSWEB/index.html.
22. E. Visser, A survey of rewriting strategies in program transformation systems, in B. Gramlich and S. Lucas, (eds.), *Workshop on Reduction Strategies in Rewriting and Programming (WRS’01), vol. 57 of Electronic Notes in Theoretical Computer Science*, Utrecht, The Netherlands, 2001. Elsevier Science Publishers, New York.
23. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin, *Aspect-Oriented Programming*, New York: Springer-Verlag, 1997.
24. C. V. Lopes and G. Kiczales, *D: A Language Framework for Distributed Programming*. Technical report SPL9710047 Xerox Palo Alto Research Center, February 1997.

25. N. Jones, C. Gomard, and P. Sestoft, *Partial Evaluation and Automatic Program Generation*. Englewood Cliffs, N.J: Prentice Hall, 1993.
26. R. M. Burstall and J. Darlington, A transformation system for developing recursive programs, JACM, **24** (1): 44–67, 1977.
27. W. F. Opdyke, *Refactoring Object-Oriented Frameworks*. PhD Thesis, University of Illinois at Urbana-Champaign, Champaign, IL.
28. M. Fowler, K. Beck, J. Bryant, W. Opdyke, and D. Roberts, *Refactoring – Improving the Design of Existing Code*, Reading, M.A.: Addison-Wesley, 1999.
29. K. Beck, *eXtreme programming eXplained: Embrace Change*. Reading, P.A.: Addison-Wesley, 1999.
30. H. P. Barendregt. The lambda calculus: Its syntax and semantics, in *Studies in Logic and the Foundations of Mathematics, vol. 103.* Revised ed. Amsterdam: North-Holland, 1984.

VICTOR L. WINTER
University of Nebraska at Omaha
Omaha, Nebraska

RAPID PROTOTYPING

INTRODUCTION: WHY RAPID PROTOTYPING IS NEEDED

Explicit process models for software development have evolved in response to various problems encountered in the development of large, complex software systems. These problems include cost/schedule overruns and the production of systems that do not operate as specified or do not meet customer needs. Process models have converged on rapid prototyping methods to reduce the risks of software misdevelopment.

Before process models were made explicit, software development suffered from chaotic implementation without comprehensive, prior, requirements analysis or design. Formulating requirements that accurately represent the needs of customers is a limiting factor in the success of software, particularly for large systems that serve diversified user communities. Different people have partially overlapping and sometimes contradictory viewpoints on different aspects of the requirements that are associated with their particular job functions. Requirements analysts must create precise, formal models of unfamiliar problems, based on imprecise communication with system stakeholders, each of whom only has a partial understanding of the system requirements. The situation is worse for applications where computers are being introduced for the first time. The new system fundamentally may redefine the job functions of the customers so that the introduction of the system can cause changes in the job functions it is intended to support. The full impact of a proposed software requirement, therefore, can be very difficult to predict and assess.

The accuracy of the transition from fluctuating informal views of the problem to a fixed formal model is fundamentally uncertain. Ideally, we would like to have dynamic formal models that can be adapted easily to changing situations. Reasonably accurate approximate models can be created by using an iterative guess/check/modify cycle that relies on prototype demonstrations and customer feedback to converge to a consensus about the requirements. The purpose of system and software prototyping is to help customers understand and criticize the proposed systems and to explore the possibilities that computer solutions can bring to their problems in a timely and cost effective manner. Measurements of prototypes can reduce uncertainty about the properties of a proposed design before it is implemented and support assessments of suitability, feasibility, performance, relative merits of alternative designs, and impact on stakeholder organizations.

The main incentive for using prototypes is economic: Scale models and prototype versions of most systems are much less expensive to build than the final versions. Prototypes, therefore, should be used to evaluate proposed systems if acceptance by the customer or the feasibility of development is in doubt. As complexity of the proposed system increases, so does the probability of requirements errors creeping into the specification and the cost of implementing the system. Software prototyping is an appropriate tool for increasing the probability of project success and for potentially reducing cost in such situations.

WHAT IS RAPID PROTOTYPING?

A prototype is an executable model of a proposed system that accurately reflects a chosen subset of its properties, such as display formats, functionality, or response times. Prototypes are useful for formulating and validating requirements, resolving technical design issues, and speeding up development of proposed systems. Rapid prototyping refers to the capability to create a prototype with significantly less effort than it takes to produce an implementation for operational use.

RELATION TO THE FINAL SYSTEM

Prototypes can be developed either to be thrown away after producing sought insight or to evolve into the product version of the software. A tradeoff exists between these approaches, where the choice depends on the context of the effort.

A software prototype may not satisfy all constraints on the final version of the system. For example, the prototype may

- provide only a subset of all the required functions,
- be expressed in a more powerful or more flexible language than the final version,
- run on a machine with more resources than the proposed target architecture,
- be less efficient in both time and space than the final version,
- be limited in capacity (databases may be implemented in main memory),
- not include the full facilities for error checking and fault tolerance, and
- not have the same degree of concurrency as the final version.

Such simplifications often are introduced to make the prototype easier and faster to build. To be effective, partial prototypes must have a clearly defined purpose that determines what aspects of the system must be reproduced faithfully and which ones safely can be neglected.

The Throw-Away Approach

The main advantage of the throw-away approach is that it enables the use of special-purpose languages and tools even if they introduce limitations that would not be acceptable in an operational environment or even if they are not capable of expressing and addressing the entire problem. The throw-away approach is most appropriate in the project

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

acquisition phase where the prototype is used to demonstrate the feasibility of a new concept and to convince a potential sponsor to fund a proposed development project.

The main disadvantage of a throw-away prototype is spending implementation effort on code that will not contribute directly to the final product. Also, the temptation exists to skip or abbreviate documentation for throw-away code. This temptation is harmful because the lessons learned from the prototyping effort may be lost if they are not recorded. Lack of documentation and degradation of the initial design simplicity may block the evolution of the prototype before it reaches a form that captures customer needs, with respect to the scope of the prototyping effort.

The Evolutionary-Build Approach

The evolutionary-build approach produces a series of prototypes in which the final version becomes the software product. This approach depends on special tools and techniques because usually it is not possible to put a prototype into production use without significant changes to its implementation to optimize the code and to complete all details. The conceptual models and designs contained in a prototype usually can be used in the final version. Precise specifications for the components of a prototype and clear documentation of its design, therefore, are critical for effective software prototyping, as are tools for transforming and completing designs and implementations.

RELATION TO THE SOFTWARE EVOLUTION PROCESS

Rapid prototyping should be an integral part of the software development and evolution process. Prototyping can reduce the amount of maintenance effort spent on correcting requirements errors after systems have been delivered. However, prototyping requires explicit planning for iterations because the feedback process generally needs several cycles to converge. To keep the process predictable and visible to project managers, schedule and budget for multiple cycles should be arranged at the outset.

RELATION TO SOFTWARE AUTOMATION

To be effective, prototypes must be constructed and modified rapidly, accurately, and inexpensively. They do not have to be efficient, complete, portable, or robust, and they do not have to use the same hardware, system software, or implementation language as the delivered system. The automated construction of programs is needed to support the evolutionary approach to rapid prototyping, and such tools can be very useful in this context even if the resulting programs are not very efficient.

RELATION TO OTHER DEVELOPMENT APPROACHES

Rapid prototyping is related to other development approaches that emphasize customer involvement and iterative development, such as agile programming, rapid application development, and the spiral model. The common theme is using customer feedback to ensure the system will solve the intended problems.

Agile programming is a specialization of incremental development that emphasizes the frequent delivery of working code and intensive communication with customers over detailed documentation and design. Some specific approaches in this category include extreme programming, Scrum, and Crystal. These approaches use deliverable code instead of prototypes to elicit customer feedback and requirements adjustments. They depend on the assumptions that the code can be modified cheaply and reliably and that this process can be done with minimal informal documentation. They work best on a small scale and with availability of intensive involvement of knowledgeable customers. Current agile approaches have relatively little automation support because of the informality of the designs and documentation.

Rapid Application Development was introduced in the 1980s as an approach to combine rapid prototyping and computer aided software engineering (CASE). Most CASE tools claim to support the approach, which is targeted at large-scale systems and seeks to capture requirements in forms that can enable tools to generate at least some parts of the code.

The Spiral Model was proposed in 1988 by Boehm. It seeks to develop large systems via a combination of risk assessment, prototyping, and incremental development. The approach seeks to mitigate several kinds of risks; the risks related to misunderstood requirements and to customer acceptance of the new system are addressed via prototyping.

RELATION TO REQUIREMENTS ENGINEERING

Typical challenges in requirements engineering involve dealing with ambiguity, incompleteness, inconsistency, and unstated requirements. Prototyping is one method for addressing these issues.

Communication between people with different areas of expertise is problematic because the people may be associating different meanings with the same terminology without realizing it. Specialized terms typically are well defined and clear to experts in each field but ambiguous in the wider context. Readers may not be aware of the specialized senses of the words and may assume their common meanings instead or assume different specialized meanings for the same word drawn from their own area of expertise. Prototyping provides concrete examples of proposed system behavior that can help expose this type of problem. By helping system stakeholders visualize how the proposed system will affect their jobs and responsibilities, prototype demonstrations also can elicit previously unstated requirements and expose incomplete descriptions that are likely to be interpreted differently by stakeholders and developers.

Inconsistencies arise naturally in requirements for systems with many different stakeholders, particularly if they have different responsibilities and associated conflicts of interest. The resulting conflicts and inconsistencies can be very difficult to identify if the documentation is lengthy. Because a prototype must be consistent with the requirements it seeks to demonstrate, constructing a prototype can provide early detection of inconsistencies that lie within the scope of the demonstration.

USER'S VIEW OF AN INTEGRATED PROTOTYPING ENVIRONMENT

A prototyping environment is a set of tools for supporting prototyping. The main functions that should be provided by an integrated prototyping environment are a convenient interface for formulating and viewing the specifications and design of a prototype, execution and analysis capabilities, support for evolution and reuse, and optimization capabilities.

The designer interface should provide decision support for the designer and a high-level model of the decisions that the designer must make so that major choices can be made explicitly and implied details can be supplied by a design management system. Such facilities are essential for reducing the amount of detail that a prototype designer must consider explicitly. Static analysis tools should help the designer assess prototype properties such as type consistency, feasibility of timing constraints, consistency between the levels of a hierarchical description, preconditions on input parameters and generic parameters, constraints on relative rates of producer and consumer processes, an absence of deadlocks in distributed and parallel systems, an absence of unhandled exceptions, and so forth.

Execution support should include methods for executing incomplete specifications and facilities for controlling, monitoring, measuring, and summarizing the results of execution, as well as debugging. Reusable components and default assumptions are needed to realize specified behavior if details of algorithms and data structures have not been given by the designer.

The environment should include a design database that supports the evolution of the prototype by managing the dependencies between the requirements and the prototype design, supporting change impact analysis, recording the history of the prototype development, coordinating concurrent updates to the design, and providing facilities for combining design alternatives in different combinations. Meaning-changing transformations also are important for supporting evolution.

Optimization facilities are needed to support the transition from the prototype version to the software product. Such facilities are needed to improve the performance of quick and simple first implementations of requirements within the scope of a prototype that are not covered by existing reusable components with mature implementations. This optimization process can be partially automatic and can be partially guided by the designer via annotations that provide implementation advice. Such annotations, in some cases, can enable the details of the product code to be generated automatically from the same source as the prototype, while allowing designers to tune the performance of the implementation by selectively overriding the default implementation methods used during the prototype execution with more sophisticated data structures and algorithms. Such facilities are essential for a mature, integrated prototyping environment because they enable product quality performance while preserving the flexibility inherent in a prototype description tailored to support system maintenance and rederivation.

SUPPORTING TECHNOLOGY FOR EFFECTIVE PROTOTYPING

Computer-aided prototyping depends on emerging technologies and is migrating gradually into practical use as these technologies mature. The relevant technologies involve the following, as explained below:

1. Prototyping languages
2. Execution support
3. Software reuse
4. Computer-aided design

Prototyping Languages

Rapid prototyping languages are used to create software prototypes, which are executable descriptions of simplified models of proposed software systems. They also support processes that document, analyze, and adjust the models. A prototyping language is used by both people and the software tools in a prototyping environment. To support the human users, a prototyping language should make it easy to write, understand, and modify the models. To support the tools, the language should be easy to analyze and transform to reflect requirement changes. An example of a prototyping language is prototype system description language (PSDL). PSDL provides a simple representation of system decompositions by using data flow diagrams that are augmented with nonprocedural control constraints and timing constraints (maximum response times, maximum execution times, minimum inter stimulus periods, periods of periodic operators, and deadlines). The language represents both periodic and data-driven tasks and both discrete (transaction-oriented) and continuous (sampled) datastreams.

A prototyping language should provide a simple computational model and primitives that match the problem domain as closely as possible. This goal can be met either via domain-specific prototyping languages or by providing domain-specific components and toolkits.

In addition to supporting an execution capability, languages used in prototyping must simplify the description of the software and capture specifications and requirements. Specifications and links to requirements are needed to record which aspects of the prototype are system requirements so that they can be distinguished from accidental consequences of execution support mechanisms. This distinction affects the presentation and analysis of prototype demonstrations and the transformation of stable prototypes into software products.

Execution Support Technology

Execution support for a prototyping language requires extending conventional translation techniques with transformations and application-specific techniques for automatically generating programs to allow the execution of incompletely specified facilities. This extension can be performed with the help of default assumptions for unspecified decisions and scheduling processes that meet real-time constraints.

Scheduling requires models of the target hardware configuration. Because the components of a prototype may not be fully optimized and may run on different hardware than the product version, demonstrations of prototypes with real-time constraints often require simulations that provide linearly scaled real-time performance that faithfully represents the behavior of the intended system, possibly at a reduced speed. In cases where control of the physical systems must be part of the demonstration, either suitably time-scaled models of the physical systems must be constructed or the software must be hosted on hardware that is sufficiently fast to run the simulations in actual real time to keep up with the dynamics of the real physical system.

Real-Time Scheduling

Prototyping of embedded software presents special challenges because such software often is associated with real-time constraints that must be met under all operating conditions. Concurrent control loops also are common in embedded systems. Explicit control over the scheduling of parallel tasks usually is necessary to guarantee that such hard real-time constraints can be met because the scheduling capabilities provided by most operating systems are somewhat removed from the level of support needed for implementing hard real-time systems. The execution support system for a prototyping language that addresses real-time systems, therefore, should provide higher-level facilities for scheduling real-time operations. No generally effective and universally accepted approach to real-time scheduling exists. Thus, the execution support system for a prototyping language should provide the designer with several choices of scheduling methods and should generate the code structures necessary to realize those methods in practice.

Program Transformations

Transformations that add detail are needed to execute incompletely specified components. Such transformations should supply reasonable default values for attributes necessary for execution if the designer does not specify them explicitly. This supply of default values is essential for rapidly testing and demonstrating partially completed prototypes.

The quality of the choices is less important than the ability to replace default assumptions. Quickly and easily with increasingly accurate alternatives, which can be drawn from predefined domain-specific toolkits and policies. For example, each data type can have a built-in default output representation as a string in a text box, with a selectable list of optional alternative representations. For numerical values, alternative representations such as gauges, plots, or moving graphics can enable visualization in different ways. Fine tuning can be done via controllable parameters.

Default values can be overridden explicitly to produce more accurate models of the system or to improve its performance. Default implementations can be created by simple or increasingly sophisticated techniques, such as interactively asking the user to supply values, using random selections from a fixed set of responses, using internet searches or responses from online games, using logic programming to simulate black-box specifications, or using transformation techniques to generate effective implementations from black-box descriptions.

Automated Program Construction

The prototyping systems with the highest levels of automation support have been designed for specific problem domains. Such prototyping systems have been developed for problem domains that include business information processing, user interfaces, computer languages, and real-time systems.

Generators for business information systems provide graphical interfaces to databases to define database schemas, queries, and reports by graphically defining table layouts. Many commercially available tools exist in this category.

Interface generation systems generate graphical user interfaces based on a set of predefined components, such as windows, menus, scroll bars, and buttons. These components are placed and adapted interactively.

Generators for language processors are based mostly on attribute grammars. These systems can generate various tools for computer languages based on a context-free grammar for the language, augmented with equations that define computed attributes for the nodes of the parse tree. This technology can be used to prototype tools for computer languages, including translators, interpreters, pretty printers, type checkers, dataflow analyzers, and so forth. Applications span programming languages, specification languages, and data definition languages for databases; they span hardware description languages and command languages for applications programs. Attribute grammar processors have been coupled to generators for syntax-directed editors and program transformation systems.

The most powerful systems are domain-specific and include built-in knowledge about effective solution methods for typical problems in the domain. This knowledge is typically materialized in the form of reusable components, special-purpose code generators, or inference engines with domain-specific rules for combining and adapting components. Many tools come with GUI generators and support rapid component composition via drag-and-drop interfaces that let users create annotated graphical pictures of the intended connection structure and then generate the corresponding connection and control code. For example, the CAPS system for prototyping real-time control systems uses a scheduling algorithm to find a schedule that meets the hard real-time constraints associated with the components and then generates the control module that connects the components and executes them according to the schedule. Other systems recognize interface mismatches and are capable of generating adapter code to correct some types of mismatches. For example, the AMPHION system for spacecraft mission planning automatically inserts transformations between different coordinate systems where they are needed to bridge the gaps between components that use different kinds of coordinates.

Software Reuse and Open Architectures

Software reuse is essential for rapid prototyping because it can enable the designer to avoid many details that have been considered previously. The environment should assist the designer in retrieving reusable components and in tailoring and combining available components to fulfill queries that do not exactly match any of the reusable components explicitly stored in the software base. Reuse can be applied at the levels of code (algorithms and data structures), design (system decompositions), and requirements models (domain-specific concepts). A difficulty with this approach is the cost of obtaining suitable components, re-engineering them to be reusable in wider contexts, and ensuring that they can be interconnected without conflicts.

Open architectures are useful for enhancing the effectiveness of reuse for software prototyping and system evolution. An architecture is the common structure of a family of systems that span a particular problem domain. This structure consists of subsystem slots, their organization and interconnections, and the constraints, protocols, and standards associated with the slots and connections. A subsystem slot is a place in the architecture that can be filled with a plug-in component that conforms to the associated constraints, protocols, and standards. An architecture is open if the details of the architecture are known publicly and have been specified accurately enough to enable any component or connector that meets the given constraints to be used together with any combination of other components that satisfy the architecture to form a properly working system. Open architectures define families of reusable plug-compatible components and create associated market incentives for many vendors to create such components. They support flexible systems in prototyping that can be reconfigured by swapping components for other plug-compatible components with different characteristics.

Computer-Aided Design

Computer-aided design relevant to prototyping includes configuration management, integration of subsystems, high-level debugging, explanations, and optimization. Many of these design processes are amenable to a model-based generation approach, in which many details are derived automatically from simplified high-level models.

System Integration

The individual subsystems that comprise a large system commonly are developed by different teams of developers. Integration tools can aid the process of combining such subsystems by supporting validation of the decomposition before to dividing up the work and can be used for comparison purposes when assessing whether delivered subsystems conform to their requirements. Both testing and proof technologies are relevant to this validation process.

High-Level Debugging

A mature prototyping environment should support debugging and error handling at the level of abstraction provided by the prototyping language. Errors and failures during prototype execution should be mapped from the underlying programming-language level to the level of the prototype language to keep low-level programming details from intruding when the designer tests and demonstrates a prototype.

Explanations

Justifications for decisions made by the tools should be available as a feedback mechanism for the designer in cases where the automated design completion procedures fail. Such a failure explanation facility is needed to support systematic computer-aided design in situations where complete automation is not possible.

CONCLUSION

Ideally, prototyping should be integrated with the process that produces the final implementation. To produce deliverable software, prototyping tools should provide optimization capabilities to produce programs whose efficiency is comparable to the designs of competent programmers. This goal generally is feasible only in the context of specific application domains that have mature solution techniques. The beginnings of the required technologies are visible: Correctness-preserving transformations and performance-estimation techniques can be used to guide derivation strategies that systematically produce efficient implementations.

In the longer term, prototyping systems will have reasoning capabilities and extensive knowledge bases that may include generic models of the problem domain, common goals of customers, common system structures, and generators producing specifications and code for classes of software components. Facilities for supporting formal verification of prototype decompositions are desirable to ensure that the proposed decompositions are viable, especially if the subcomponents are to be built by different developers.

FURTHER READING

D. Dampier, Luqi, and V. Berzins, Automated merging of software prototyes, *J. Systems Integration*, **4**(1): 33–49, 1994.

F. Kordon (ed.), Special issue on Rapid System Prototyping, Vols. 8(3–5): of *Distributed Systems Online*, IEEE, 2007.

F. Kordon, Luqi , and L. Wills, (eds.) Special issue on Rapid System Prototyping, Vol. 70(3): of *Journal of Systems and Software*, Elsevier, 2003.

X. Liang, L. Zhang, and Luqi , Automatic Prototype Generating via Optimized Object Model, *ACM ADA Letters*, **23**(2): 22–31, 2003.

Luqi ,Computer-Aided Prototyping for a Command-and-Control System Using CAPS, *IEEE Software*, **9**(1): 56–67, 1992.

Luqi, Real-Time Constraints in a Rapid Prototyping Language, *J. Computer Languages*, **18**(2): 77–103, 1993.

Luqi (ed.), Special issue on Computer Aided Prototyping, Vol. 6(1–2) of *J. Systems Integration*, Kluwer, 1996.

Luqi , C. Chang, and H. Zhu, Specifications in Software Prototyping, *J. Systems and Software*, **42**(2): 189–197, 1998.

Luqi , Z. Guan, V. Berzins, L. Zhang, D. Floodeen, V. Coskun, J. Puett, and M. Brown, Requirements Document Based Prototyping of CARA Software, *Int. J. Software Tools for Technology Transfer*, **5**(4): 370–390, 2004.

LUQI
Naval Postgraduate School
Monterey, California

REQUIREMENTS SPECIFICATION AND ANALYSIS

The first step in engineering a software system is to understand what the system should do—this is referred to as the specifications of the requirements for the software system or, alternatively, as requirements specifications or software requirements specifications (SRS). Once the requirements have been specified, very often the next step is to analyze them to update, modify, and prioritize the requirements so that the software development team develops a better understanding of the system to be developed, the constraints on the system, and initial estimates on cost and schedule for developing the system—this step is referred to as requirements analysis. Thus, requirements specification and analysis is a two-step process that is also iterative—analysis may lead to revised specifications that could entail additional analysis. Requirements specification and analysis is a phase in software engineering; from a software engineering perspective, the requirements specification and analysis phase is followed by the software design phase.

The requirements specification and analysis phase of a software project is the most important phase of software development and should not be omitted under any condition. More than half of software projects have failed because of errors in the requirements specification and analysis phase, and the cost for correcting errors committed during this phase increases exponentially as software system development progresses through the remaining phases of design, implementation, testing, and maintenance. It has been estimated (1), as shown in Fig. 1, that if an error detected and fixed during the requirements phase will incur a cost ratio of 1, then the same error if detected and fixed during the design phase will incur a cost ratio of between 3 and 6, during the implementation phase will incur a cost ratio of 10, during the development testing will incur a cost ratio of between 15 and 40, during the acceptance testing will incur a cost ratio of between 30 and 70, and if the error is detected and fixed when the system is under operation (or during the maintenance phase), the organization will incur a cost ratio of between 40 and 1000. Therefore, proper requirements specification and analysis is important during software development.

A basic rule of SRS development is to capture "what" the software system should do and never rush to the "how" the requirement is to be achieved—violation of this simple rule will lead to an early commitment to design without suitably exploring alternative designs. Even though this rule seems simple enough, the history of requirements specifications is replete with examples of breaches of this rule that may be a reason for the large number of software project failures—the temptation to rush to design or even implementation without first understanding the requirements is sometimes too much for software engineers. Thus, very often solutions are produced for the wrong problem or wrong solutions are produced for the actual problem!

Requirements for a software system can be classified into two types: functional and nonfunctional. Functional requirements specify what the system should do, whereas nonfunctional requirements specify the global characteristics of the software system. Thus, the requirement "*The input to the software system should be by means of a keyboard entry, mouse click, or stylus movement*" is a functional requirement, whereas "*The software system should have fast responses to user inputs*" is a nonfunctional requirement (NFR) since the "fast responses" requirement is usually not achievable by means of just a few components in the software system but is a globally observed characteristic of the software system. Functional requirements include requirements related to inputs, outputs, processes (or functions), and stored data for the new system. Functional requirements also capture the interfaces between the system and its users as well as the interfaces between systems. Nonfunctional requirements (also referred to as quality requirements or system attributes) capture requirements related to characteristics such as performance, usability, security, reliability, availability, maintainability, and portability; effort, budget, and schedule estimations; documentation and training needs; quality management; and constraints under which the final system may be expected to operate, for example, operating system, processing speed, network bandwidth, memory size, or implementation language.

An SRS typically includes

1. Verbal descriptions of functional and nonfunctional requirements.
2. Analysis artifacts such as requirements prioritization, dependencies, versioning, cost and staff size estimations, formulation of acceptance tests, and analysis of NFRs.
3. All associated models developed during analysis.

Not only does the SRS serve as the starting point for subsequent phases of software development, SRS also often serves as a legal contract between the software developing organization and the customer who will actually be purchasing and/or using the software.

The process of developing software requirements and analyzing them is illustrated in Fig. 2, and the focus during this phase is on the customers and users of the proposed software system. The process starts invariably with an understanding of the current system and its problems—the current system may be manual or computerized. This task is also called problem analysis. The techniques to help with this task include the PIECES (performance, information, economics, control, efficiency, and service) framework, Ishikawa diagrams (also called fish-bone diagrams), study of forms and documentation used with the current system, interviews with users of current system, and observations of use of the current system. The outputs of this task include the problem statement and system improvement objectives

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

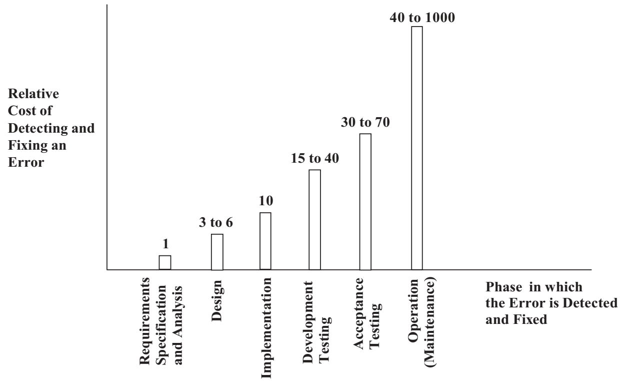


Figure 1. Relative costs of detecting and fixing an error in different software development phases (not drawn to scale).

that are used during the second task to guide elicitation of requirements from the users and customers. Usually, a subtle distinction is made between a user (also called an end user) and a customer. Customer is the organization or person(s) who will be paying for the software or sponsoring the software project, whereas users are the actual users of the software system who will be interacting with the software for the benefit of their organization; of course, customers may be users as well. The most important technique to elicit requirements is interviewing, but recently JRP (joint requirements planning) and discovery prototyping have emerged as supplementary techniques. The outputs of this task include notes, minutes, voice recordings, and updated prototypes. The third task uses the outputs of requirements elicitation to develop the initial SRS that documents the requirements using any combination of tabular listing, use-cases, and user stories. The first, second, and third tasks collectively form the requirements specifications step referred to earlier. The fourth task analyzes the initial SRS using reviews and customer/user clarifications to identify priorities, dependencies, versions, cost and manpower estimates, acceptance tests, and NFR analyses. The output of this task is the updated SRS. The fifth and final task models the requirements using any combination of UML (Unified Modeling Language) diagrams, ERD (Entity Relationship Diagrams) (2), DFD (Data Flow Diagrams), and formal methods. The output of the fifth task, also called logical design, is the updated SRS with models. The fourth and fifth tasks together form the requirements analysis step mentioned earlier. The SRS is now ready for the next phase of software development, namely, design, which is also called the physical design since implementation or physical aspects of the software system will have to be considered. As indicated in Fig. 2, the process is not linear but iterative; it is possible to go from one task to any of the earlier tasks whenever an error or omission is detected. Each technique is discussed in more detail in the subsequent sections. Typically, requirements specification and analysis phase lasts one third of the total project time and the durations of the individual tasks within this period are distributed almost evenly.

TASK 1: UNDERSTAND PROBLEMS WITH THE CURRENT SYSTEM

Very frequently the main driving force behind the development of new software systems is the set of problems faced by the end users in using the current system and processes, both of which may include a combination of manual and computerized aspects. Therefore, a good understanding of the problems with the current system will give the developers a set of measurable objectives that the new system

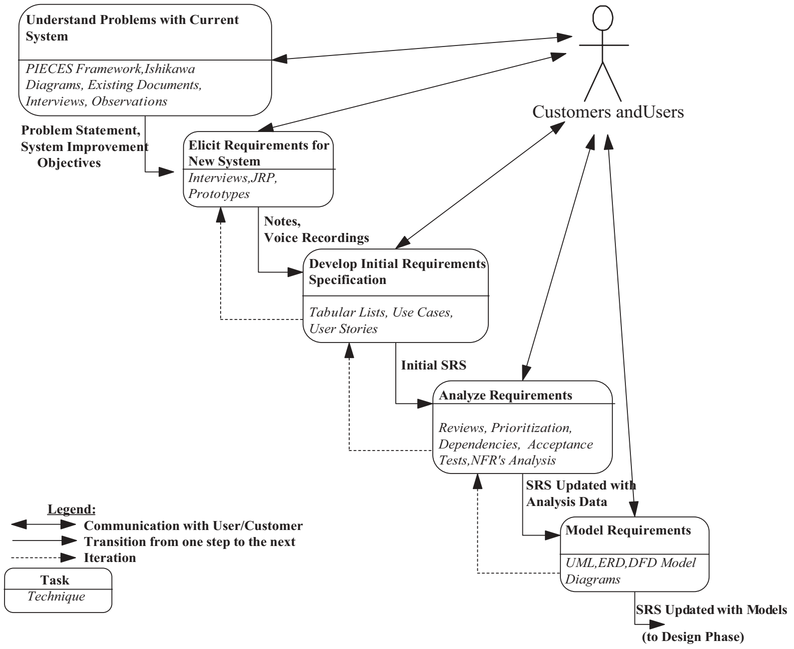

Understand Problems with Current System
PIECES Framework,Ishikawa Diagrams, Existing Documents, Interviews, Observations
Customers andUsers
Problem Statement, System Improvement Objectives
Elicit Requirements for New System
Interviews,JRP, Prototypes
Notes, Voice Recordings
Develop Initial Requirements Specification
Tabular Lists, Use Cases, User Stories
Initial SRS
Analyze Requirements
Reviews, Prioritization, Dependencies, Acceptance Tests,NFR's Analysis
SRS Updated with Analysis Data
Model Requirements
UML,ERD,DFD Model Diagrams
SRS Updated with Models
(to Design Phase)
Legend:
Communication with User/Customer
Transition from one step to the next
Iteration
Task
Technique

Figure 2. Process of software requirements specification and analysis.

should satisfy. Examples of measurable objectives are "*Number of orders processed should increase by 10%*" or "*Response time decreases by 20%.*" These objectives can be achieved by fixing certain problems with the current system, and therefore, those problems provide not only opportunities for improving the current system but also serve as starting points for developing the requirements for the new and improved system. However, how does one identify the problems? The PIECES framework (3) is an excellent tool to document problems in an existing system—very frequently problems occur in the categories of performance, information, economics, control (and/or security), efficiency, and service. Other techniques include interviewing users of the current system, examining problem reports on the current system, and observing actual usage of the current system. When a large number of end users are involved, questionnaires may also be used. Once the problems are identified, they are documented in a problem statement matrix that lists the problems identified, the urgency of each problem, the monetary benefits in fixing each problem, and how each problem might be fixed such as, for example, by developing a new system or simply patching the existing one. Once the problems have been identified, it becomes necessary to determine their causes, and for this purpose, Ishikawa diagrams (also called fish-bone diagrams) (1) may be used. For each problem, an Ishikawa diagram is developed wherein the problem is mentioned along the main bone and the side bones represent categories under which causes for the problem may be grouped; the actual causes are listed off the side bones. Ishikawa diagrams help distinguish between symptoms and problems, and once all the problems are identified, a systematic effort may be undertaken to determine how the problems may be eliminated—called opportunities for improvement—and in this process, the preliminary requirements for the new system can be identified. As a result of this process, we develop the system improvement objectives matrix that lists, for each problem in the problem statement matrix, the probable causes and the measurable objectives for verifying the absence of the problem in the new system.

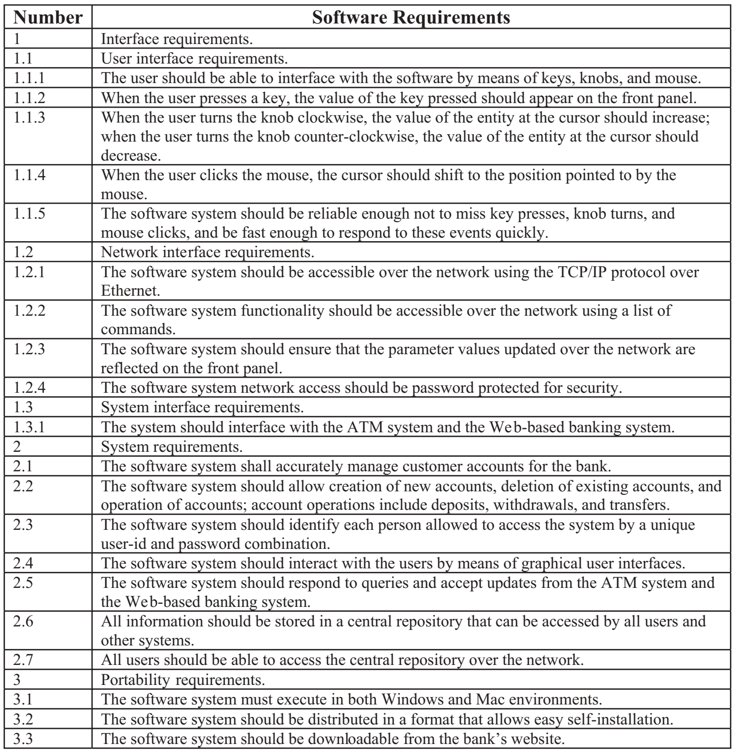
TASK 2: ELICIT REQUIREMENTS FOR NEW SYSTEM

As one might expect, software requirements are usually provided by the end users of the proposed software system. This process of gathering requirements from end users is called requirements elicitation. Even though elicitation seems like a simple activity, the trouble is that very frequently users are not sure of what they want (4)! Eliciting requirements should preferably be handled by software engineers well versed in communication and an understanding of human nature, because they will be required, quite often, to lead the process of elicitation by careful questioning of the end users. Interviews are the most common technique for eliciting requirements from end users. During interviewing, the questions necessary to elicit functional and nonfunctional requirements should be put to the end users. For example, questions such as "*What should the system do?*", "*What inputs will the system receive?*", "*What outputs should the system generate?*", and "*What are the data formats?*" are examples of questions that will help elicit functional requirements, whereas questions such as "*What is the maximum tolerable throughput of the system?*", "*What types of access control are required?*", "*What is the expected mean time between failures?*", "*What operating systems should the system work in?*", and "*What programming language will be used to implement the system?*" are examples of questions that will help elicit nonfunctional requirements. The user responses to these questions may be captured as notes or minutes on paper, or voice may be recorded digitally or on tapes. Important points to keep in mind are the types of questions to ask such as open-ended (for example, "*What is the maximum number of expected users for the system?*") and closed-ended (for example, "*Will the maximum number of expected users for the system be less than ten or between ten and hundred or more than hundred?*"), whether the interviews should be structured or unstructured, understanding body language since only about 7% information (1) is communicated verbally, and understanding proxemics or the relationship between people and space around them. When a large number of end users exists it may be cost effective to use questionnaires. An important factor in developing questionnaires is whether free-format questions or fixed-format questions need to be asked.

Another source of requirements are the physical forms and documents that are used for the existing system, including user manuals, filled-in forms used for data capture, standard operating procedures, notes, memos, e-mails, and forms capturing user feedback. Often, the new software system replaces physical forms and documents, and the requirements engineer will have to understand the form's contents and its usage, along with the processes listed in the documents to formulate requirements for the new software system.

JRP is another technique for eliciting requirements—JRP is part of the JAD (Joint Applications Development) philosophy that requires the relevant stakeholders (users and their managers) to participate in a workshop for collaboratively gathering the requirements. The JRP sessions are facilitated usually by an external moderator who ensures that the sessions focus on their objectives and not let egos dominate the proceedings. JRP requires attendance from the technical staff as well so that any requirements that may not be feasible may be immediately pointed out to the concerned users. JRP ensures that collective responsibility for system development occurs and that the users are actively involved in the development process, which encourages them to take ownership of the project.

The discovery prototype is yet another technique for eliciting requirements in which a small-scale working model (sometimes referred to as a "quick-and-dirty" implementation) of the proposed system is developed and given to the users to work with. Based on the philosophy that the users will understand the requirements once they actually see a working sample, the users are frequently able to better articulate their requirements based on their experiences with the prototypes. Discovery prototypes are useful in eliciting requirements that are not clearly understood (sometimes by the users themselves). It must be kept in mind that the prototypes cannot be considered as the first



Number	Software Requirements
1	Interface requirements.
1.1	User interface requirements.
1.1.1	The user should be able to interface with the software by means of keys, knobs, and mouse.
1.1.2	When the user presses a key, the value of the key pressed should appear on the front panel.
1.1.3	When the user turns the knob clockwise, the value of the entity at the cursor should increase; when the user turns the knob counter-clockwise, the value of the entity at the cursor should decrease.
1.1.4	When the user clicks the mouse, the cursor should shift to the position pointed to by the mouse.
1.1.5	The software system should be reliable enough not to miss key presses, knob turns, and mouse clicks, and be fast enough to respond to these events quickly.
1.2	Network interface requirements.
1.2.1	The software system should be accessible over the network using the TCP/IP protocol over Ethernet.
1.2.2	The software system functionality should be accessible over the network using a list of commands.
1.2.3	The software system should ensure that the parameter values updated over the network are reflected on the front panel.
1.2.4	The software system network access should be password protected for security.
1.3	System interface requirements.
1.3.1	The system should interface with the ATM system and the Web-based banking system.
2	System requirements.
2.1	The software system shall accurately manage customer accounts for the bank.
2.2	The software system should allow creation of new accounts, deletion of existing accounts, and operation of accounts; account operations include deposits, withdrawals, and transfers.
2.3	The software system should identify each person allowed to access the system by a unique user-id and password combination.
2.4	The software system should interact with the users by means of graphical user interfaces.
2.5	The software system should respond to queries and accept updates from the ATM system and the Web-based banking system.
2.6	All information should be stored in a central repository that can be accessed by all users and other systems.
2.7	All users should be able to access the central repository over the network.
3	Portability requirements.
3.1	The software system must execute in both Windows and Mac environments.
3.2	The software system should be distributed in a format that allows easy self-installation.
3.3	The software system should be downloadable from the bank's website.

Figure 3. Spreadsheet (or tabular) listing of software requirements for a bank account management system.

version of the system and that their intent was requirements discovery only.

TASK 3: DEVELOP INITIAL SOFTWARE REQUIREMENTS SPECIFICATION

Tabular SRS

Typically, software requirements are listed in categories with each requirement given a unique number for easy reference. Figure 3 shows an example list of requirements for a hypothetical software system that manages accounts for a bank. The software requirements are categorized at the highest level into Interface Requirements, System Requirements, and Portability Requirements; under each category, the requirements are subcategorized—for example, under the Interface Requirements category, the subcategories include User Interface Requirements, Network Interface Requirements, and System Interface Requirements. The simplest way to capture the nesting of levels of software requirements is to use the nested decimal notation as shown in Fig. 3. Typically, a requirement that a software system should have (also called a mandatory or essential requirement) is indicated by using words such as "*shall*," "*should*," or "*must*," with the importance decreasing in this order—that is, a requirement with a "*shall*" is more important than a requirement with a "*should*," and this is more important than a requirement with a "*must.*" In practice, an organization may adopt its own conventions and follow it consistently—it should be noted that modality of terms used in SRS could be different from their usage in ordinary conversation. More importantly, requirements should be unambiguous, correct, consistent, verifiable, complete, and unique (5); an ambiguous requirement is not specific, for example, "*When the user presses the key and/or turns the knob the value displayed should be updated*" —here it is not clear what value should be displayed when both the key and the knob are turned at the same time; a consistent requirement does not conflict with another requirement; a correct requirement is within the domain of the software system, that is, it is a requirement of the software system and not an extraneous factor; a verifiable requirement is one that can be subsequently verified as having been included in the final completed software system; a complete set of requirements capture responses to all possible classes of inputs; and the uniqueness of requirements ensures the that they do not repeat either in the same or another form.

Figure 3 shows both functional and nonfunctional requirements. Requirement 1.1.5 and requirement 2.1 are nonfunctional requirements, whereas the others are functional requirements.

Use-Cases

Another technique for documenting requirements, which is increasingly becoming popular, is use-cases (6); this includes both the use-case diagram and the use-case narratives. The use-case diagram captures all the interactions between the software system and the external entities, whereas the use-case narrative captures the sequential processing of an interaction. Together, they define the scope of the system, give a detailed insight

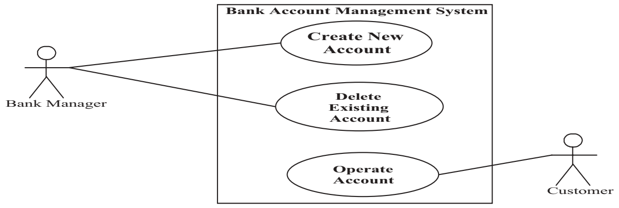


Figure 4. Modeling requirements with UML use-case diagrams.

into the software system to be developed, and allow more accurate effort and schedule estimates. Consider, for example, the use-case diagram shown in Fig. 4, where the stick figure represents a user in a specific role, also called an actor, interacting with a specific functionality or use-case of the software system represented by the oval shape. In Fig. 4, a *Bank Manager* actor exists, and its use-case is *Create New Account* that describes the sequence of activities occurring during interaction between the software system Bank Account Management System and the user *Bank Manager*. Also, it may be noted from Fig. 4 that the *Bank Manager* interacts with the use-case *Delete Existing Account* as well while the *Customer* interfaces only with the *Operate Account* use-case. More importantly, from the use-case diagram of Fig. 4, one may infer that the *Bank Manager* does not interact with the use-case *Operate Account* just as the *Customer* does not interact with the use-cases *Create New Account* and *Delete Existing Account*. Therefore, use-case diagrams also serve as a mechanism to define the scope of the system. Frequently use-cases and scenarios have been used interchangeably in the literature. However, a subtle difference does exist. A scenario is one instance of execution of the use-case, whereas a use-case is an abstract entity; therefore, one use-case can typically represent several scenarios of use. Also, one use-case may include one or more requirements for the software system.

The actual sequence of activities is described in the use-case narrative, an example of which is given in Fig. 5. The narrative of Fig. 5 captures the different ways the user *Bank Manager* can provide inputs to the system and receive outputs from the system, and it captures the detailed sequence of activities undertaken by the system in response to user inputs. Figure 5 is just one way of writing the use-case narrative—different formats are possible, including some that capture exceptional events, alternative events, and responses to multiple users (or, more appropriately, roles). Also, the details in the use-case narrative may be achieved iteratively over time—that is, the first version of the narrative may have only the intent of the use-case, the second version may have more details, the third even more details, and so on till all minutiae are uncovered in the final versions of the narrative. The details in a use-case may be modeled using UML activity diagrams during the modeling task.

Use Case Name	Create Bank Account	
Use Case ID	UC001	
Actor	Bank Manager	
Typical Course of Events	**Actor Action**	**System Response**
	1. The actor informs the system that a new account is to be created.	2. The system asks the user the type of account to be created.
	3. The actor informs the system the type of account to be created.	4. The system asks the user the account details.
	5. The actor enters the account details.	6. The system creates the new account and returns with the account number.

Figure 5. Use-case narrative for the use-case *Create Bank Account* in Fig. 4.

User Stories

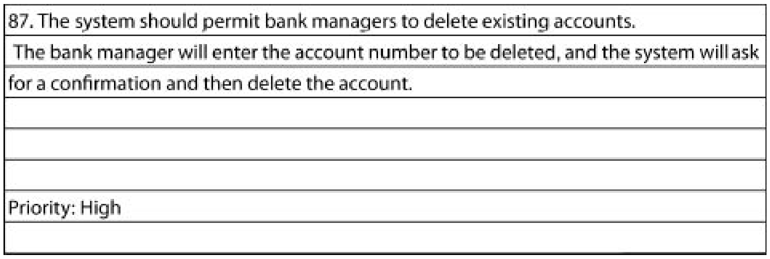
Another technique for capturing initial requirements is user stories (7)—a user story is a brief (usually about three sentences) description of an expected functionality written by the user. An example user story is given in Fig. 6. Each user story is usually captured on an index card and in many cases is written (actually handwritten) directly by the user or the domain expert using the language of the domain. That way it becomes a convenient medium to discuss business requirements with the users. Moreover, from a management perspective, each user story is given a unique number and a priority so that when the stack of user stories is collected, the scope of the system to be developed is well defined and the user stories may be prioritized for different iterations of the software system. In addition, since each user story is expected to take a fixed unit of time to implement, typically about a week, the schedule for the project can also be determined.

Decision Tables

Decision tables (1,8) are yet another technique for documenting requirements, although at a detailed level to capture business rules. Decision tables capture true or false values of conditions and their resultant effects on decisions. For example, if we have the following situation:

> *A bank account can be established only when the customer has both a social security number (SSN) and documented proof of residency; if the customer can electronically remit her salary to the account each month, then the monthly fee is waived; else a monthly fee is imposed.*

To make the decision whether a new account may be approved and, if so, whether a fee needs to be imposed, the decision table shown in Fig. 7 will be useful. In Fig. 7, *T* stands for a condition that is true, whereas *F* stands for a condition that is false, *X* represents an irrelevant condition (that is, it can be either true or false, but its value does not affect the decision), and the applicable decision is indicated with the *a* mark. Therefore, in only one situation can a new account be approved without a fee and that is when all three conditions (customer has SSN, customer has residency proof, and customer can remit salary electronically) are true; when the first two conditions are true, but the customer is unable to remit salary electronically, then a new account is approved but with a fee imposed; in all other cases, a new account is not approved.



87. The system should permit bank managers to delete existing accounts.

The bank manager will enter the account number to be deleted, and the system will ask for a confirmation and then delete the account.

Priority: High

Figure 6. Requirements capture using a user story.

TASK 4: ANALYZE REQUIREMENTS

The main purpose of requirements analysis is to refine the requirements collected during the requirements elicitation phase. Usually the requirements collected initially do not satisfy all the characteristics of being unambiguous, correct, consistent, verifiable, complete, and unique—the requirements may be considered "raw" that need to be refined and better understood for subsequent phases of software development. The chief technique for this purpose is reviewing the requirements. During reviews, teams of development personnel along with customers/users go over the SRS developed to determine errors and omissions. Any errors and omissions determined will have to be corrected before proceeding. Reviews can be used iteratively during this phase and can be applied to any artifact developed during the process of requirements specification and analysis. Other goals of requirements analysis include requirements prioritization, determining requirements dependencies, determination of versions, schedule formation, staff size determination, preliminary cost estimation, formulation of acceptance tests, and analyzing NFRs.

Requirements Prioritization

Even though at the end of the requirements specification phase we may have a list of all expected software requirements, it may not be feasible for the software development organization to implement all requirements in the very first version of the software, the chief reasons being the time constraints and the delays incurred because of the inevitable changes to software requirements. Therefore, only a subset of the requirements is usually delivered in the first version, and this subset includes only those requirements deemed most important by the customer and considered most feasible by the software development organization. Customers/users determine requirements priorities while their feasibilities are evaluated by the developers—thus, even though the customer, for example in Fig. 3, may state that requirements 1.1.1, 1.1.2, and 1.1.3 are more important than 1.1.4, the current state of hardware technology for the system may not incorporate a knob-based input; in this case, the developers may negotiate with the customer to provide only the requirements 1.1.1 and 1.1.2 with the first release and push requirements 1.1.3 and 1.1.4 to a subsequent release of the software system. An important aspect of negotiating with customers is to ensure that a "win–win" situation is reached for all concerned stakeholders.

Customer has SSN	T	F	F	T	T
Customer has residency proof	F	T	F	T	T
Customer can remit salary electronically	X	X	X	F	T
Bank account approved with fee				α	
Bank account approved without fee					α
Bank account not approved	α	α	α		

Figure 7. Documenting detailed business rules using decision tables.

Identifying Requirements Dependencies

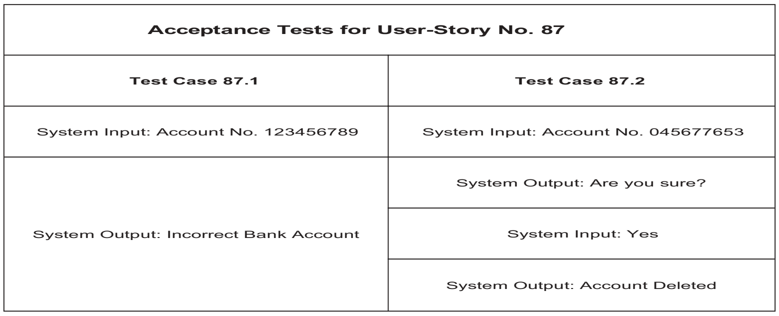
Requirements frequently have dependencies among themselves such that in order to fulfill one requirement, another requirement must have been fulfilled previously. For example, requirement 1.2.3 in Fig. 3 assumes that requirement 1.1.2 is already satisfied since front panel updating ability is a prerequisite for requirement 1.2.3; therefore, requirement 1.1.2 may need to be developed before requirement 1.2.3. Identification of such dependencies will also help with scheduling the software system development such that requirements upon which others are dependent are completed before the requirements that depend on them. When requirements are captured as use-cases, then the dependencies may exist among use-cases. Here the use-case upon which most other use-cases are dependent should be completed before the others.

Requirements Versioning

Prioritization of requirements and identification of their interdependencies can help identify the versioning requirements that determine the distribution of features (a feature is a use-case or a set of requirements) among the different versions of the software system. Therefore, for example, if a high-priority requirement is dependent on a low-priority one, it may be necessary to have the low-priority requirement along with the high-priority one as part of the first version of the system. Versioning assumes software development is based on an incremental process.

Schedule Formation

Versioning is closely tied to scheduling the software development project—thus, if the initial requirements identify the need for ten versions to accommodate all the features required by the customer, then by estimating the time needed to complete each version, an accurate schedule for the project may be developed. Since SRS frequently serves as a contract between the software development



Acceptance Tests for User-Story No. 87	
Test Case 87.1	Test Case 87.2
System Input: Account No. 123456789	System Input: Account No. 045677653
System Output: Incorrect Bank Account	System Output: Are you sure?
	System Input: Yes
	System Output: Account Deleted

Figure 8. Acceptance tests for the user story of Fig. 6 .

organization and the customer, determination of the schedule at this stage will allow the software organization to negotiate properly requirements and cost for the project. For example, using the process chosen for the project, if each version takes four months of development (design and implementation), then for ten versions and assuming sequential development, the project manager can quickly lay out a schedule lasting more than three years.

Staff Size Determination

Another objective of requirements analysis is to identify accurately the staffing requirements for the project—this will help the software organization quickly determine whether the expected schedule and cost can be satisfied or whether more negotiation with the customer is required. Since SRS frequently serves as a contract between the software development organization and the customer, it will be useful to consider staff size requirements at this stage itself. For determining staff size, if prior experience with the company's methodology indicates that a feature takes 10 person-days to design, implement, and test, then for ten features, 100 person-days will be required; based on versioning, which has already decided the approximate sequences of feature-sets to be delivered to the customer, the average staff size for development can be determined quickly.

Project Cost Estimation

Based on personnel requirements and expected length of the project, the project manager can develop or revise preliminary estimates for the budget so that the project benefits may be determined clearly. Again, since SRS frequently serves as a contract between the software development organization and the customer, it will be useful to identify or update project costs based on requirements so that, if needed, additional negotiation regarding requirements and schedule may be held between the two parties. For estimating project costs, past experience could serve as the guide; for example, if past experience indicates that each person-day costs $500, then a project requiring 100 person-days of effort will cost $50,000. However, cost estimation may also be performed using well-known models such as the COCOMO (Constructive Cost Models) that helps, especially COCOMO II (the latest version) (9), to calculate effort in terms of person-months during the requirements specification and analysis phase itself; therefore, if the cost per person-month is known, the project cost can be estimated.

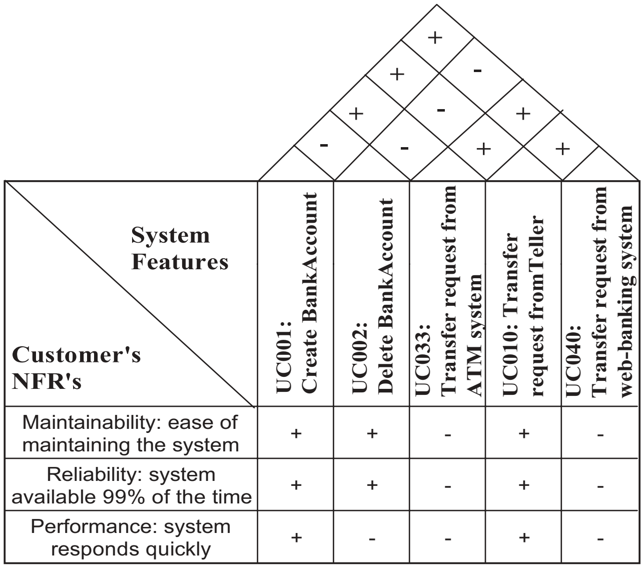
Formulating Acceptance Tests

Requirements analysis serves the very important purpose of developing acceptance tests that document the tests the user will execute on the final system for verifying that the system indeed satisfies the requirements—thus, for example, if we consider the user story shown in Fig. 6, the user may say that the acceptance tests shown in Fig. 8 will be executed to confirm the proper implementation of the user story. Development of acceptance tests during this phase will help clarify requirements and encourage proper design of software. For example, from the customer's acceptance tests, we can request the rules for identifying correct account numbers and, during design, develop a system that will first check that the entered account number is legal before continued processing.

Analyzing NFRs

Analyzing NFRs brings with it its own set of issues, chief among them being the inherent vagueness in defining NFRs. Although several techniques have been proposed, most seem to suffer from their own restricted definitions for the nonfunctional requirements they seek to analyze. However, from a wide applicability standpoint, two techniques stand out: the House of Quality and the NFR Framework. The House of Quality is part of the Quality Function Deployment process that was originally developed for the manufacturing industries in Japan and has been used for software requirements quality analysis as well. In its simplest form, the House of Quality is a matrix where rows represent the NFRs and the columns represent the technical aspects of the project that help achieve the NFRs, and at the intersection of each row and column, we identify the extent to which the technical aspects satisfy the customer's NFRs using qualitative measures such as strongly positive, positive, negative, and strongly negative. The "roof of the house" is the correlation between the technical factors themselves that captures the conflicting or synergistic interactions between the factors. The House of Quality may be extended with other aspects such as project planning and cost for achieving the NFRs. For example, for the bank account management system modeled in Fig. 4, if the user considers NFRs maintainability, reliability, and performance as being the most important, then an analysis of these NFR's using the House of Quality is presented in Fig. 9. In Fig. 9, the technical aspects considered are the two use-cases from Fig. 4, and the three additional use-cases identified for the system. In Fig. 9, we see that in the central part of the figure we denote how the use-cases affect the customer requirements—positive impacts are denoted by "+" whereas negative impacts are denoted by "–". Thus, for example, the use-case *UC033: Transfer Request From ATM System* has a negative influence on Performance, since two systems (the ATM system and the bank account management system) need to interface in order to complete this request, and such system interfaces are typically slower; however, the use-case *UC010: Transfer Request From Teller* has a positive influence on Performance, since only one system (the bank account management system) is involved and the required data are entered by the experienced teller. At the top of the figure, we indicate the trade-offs between the use-cases by using "+" to denote synergies and "–" to denote conflicts. Thus, *UC001: Create Bank Account* conflicts with *UC002: Delete Bank Account* since they are two opposing features with different constraints, whereas UC040 and UC010 synergize each other since developing one feature (UC040) helps in reducing dependence on the other (UC010). This information can be used for project planning, cost/effort determination, and use-case prioritization, and it can be captured in an extended version of Fig. 9.

Some drawbacks of the House of Quality technique include the inability to capture justifications and the flexibility to accommodate various definitions of NFRs. The NFR framework (10) addresses these drawbacks. The chief artifact of the NFR framework is the Softgoal Interdependency Graph (SIG), which captures all information pertaining to the achievement or otherwise of the NFRs. In a SIG, the NFRs are captured as NFR softgoals (depicted by a cloud-shape) that are to be achieved during the process of software development (we are concerned with the requirements specification and analysis), and three relationships are possible between NFR softgoals: AND (depicted by a single arc) means that the parent NFR softgoal is achieved only if all child softgoals of the AND-relationship are achieved; OR (depicted by a double arc) means that the parent NFR softgoal is achieved if even one of the child softgoals in the OR-relationship is achieved; and EQUAL relationship relates (by a line) one child to a parent and the parent is achieved if the child is achieved. Even though we used the word "achieved," the actual NFR framework term is satisfied, which means relative satisfaction and not absolute satisfaction. The system features are captured in the SIG by means of operationalizing softgoals (depicted by a dark-bordered cloud-shape), and the contributions of the operationalizing softgoals to the NFR softgoals are captured by contributions that come in four flavors: strongly positive (++), positive (+), negative (–), and strongly negative (– –). The contributions can be propagated up the SIG using well-defined propagation rules of the NFR framework. Finally, the reasons for the contributions (and, in fact, any element of the SIG) can be captured by means of argumentation softgoals (depicted by dashed-bordered cloud-shapes). The information in the House of Quality of Fig. 9 regarding two use-cases (UC001 and UC033) is captured easily by the SIG of Fig. 10—in this figure, the use-cases have been represented as operationalization softgoals (operationalizations refer to the artifacts currently under consideration that help achieve or deny NFR softgoals, and during the requirements specification and analysis phase, the artifacts could be requirements, use-cases, user stories, or decision tables). In Fig. 10, it may be noted that the argumentation softgoals capture justifications, the decomposition of the NFR softgoals (the AND–OR–EQUAL relationships) capture the definitions of the NFRs, and the propagation rules permit the determination of whether the NFRs are achieved, and, more importantly, they help identify the reasons in either case.



System Features / Customer's NFR's	UC001: Create BankAccount	UC002: Delete BankAccount	UC033: Transfer request from ATM system	UC010: Transfer request fromTeller	UC040: Transfer request from web-banking system
Maintainability: ease of maintaining the system	+	+	-	+	-
Reliability: system available 99% of the time	+	+	-	+	-
Performance: system responds quickly	+	-	-	+	-

Legend: "+" means positive influence (synergistic)
"–" means negative influence (conflicting)

Figure 9. Application of House of Quality to analyze nonfunctional requirements (NFRs).

TASK 5: MODEL REQUIREMENTS

Modeling helps to clarify requirements with the customers and users, so that the SRS is unambiguous, complete, and captures all essential requirements. Modeling is based on the premise that "a picture is worth a thousand words" and that information can be captured concisely using pictures. Modeling may be accomplished using modeling notations or formal logic.

Modeling Notations

Common modeling approaches include unified modeling language (UML) diagrams (11), data flow diagrams (DFDs) (1), and entity relationship diagrams (ERDs) (1). The common diagrams used for requirements analysis in UML include use-case model diagrams (as in Fig. 4), activity diagrams, class diagrams, sequence diagrams, and state charts. Noun-parsing of use-case narratives helps discover potential classes for developing class diagrams. DFDs are useful for modeling business processes and for capturing the interaction between manual and computerized processes (8). DFDs can be hierarchical with each level expand-

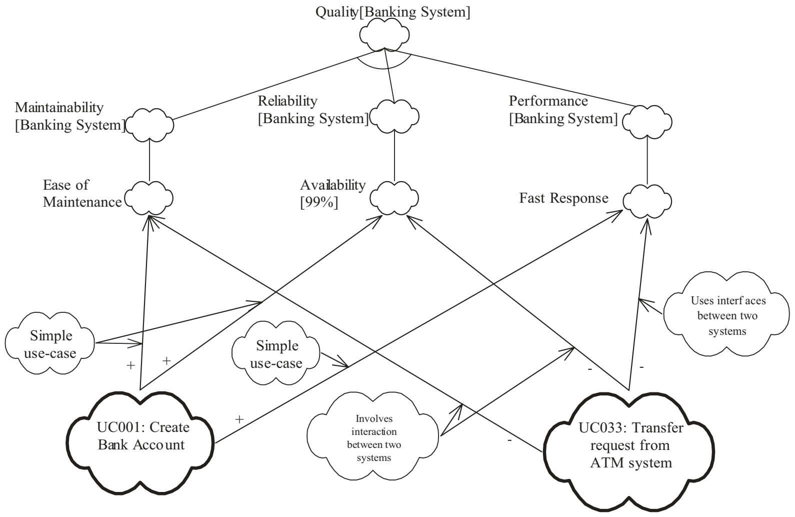


Figure 10. SIG of the NFR framework.

ing on the processes at a higher level—level 0 is the highest level and is called the context diagram, which considers the entire system as one black box; level 1 is the lower level that expands the black box of level 0; level 2 is the next lower level; and so on. A rule of thumb is to ensure that a higher level process is decomposed into at least five detailed processes at the lower level. ERDs can capture relationships between real-world entities in the problem domain and serve as an excellent technique to model data requirements for the domain. Petri nets have also been used to model requirements, but their semantics may be difficult for the average user to understand. Petri nets help to model concurrent activities where the order of occurrence is not important (12).

Formal Methods

Formal techniques for modeling requirements include first-order logic, temporal logic, Object Constraint Language, Z language, algebraic specifications, Specification and Description Language, and Software Cost Reduction (12). Formal techniques attempt to avoid problems associated with misinterpreting semantics of a model or inherent ambiguity in using natural languages (such as, for example, English) for documenting requirements. Again without going into details, it may be safely mentioned that practicality and scalability of these methods is limited even though claims have been made that for mission-critical systems these techniques are useful.

REQUIREMENTS MANAGEMENT

One of the major issues with requirements analysis is what to do when an error is discovered in the requirements—the error may be an incorrect requirement, ambiguous requirement, or even a missing requirement. The processes by which changes to the requirements specifications are handled fall under the category of requirements management. An important aspect of requirements management is ensuring that all stakeholders during the requirements phase always refer to the most updated set of requirements. Frequently a committee, called the Change Control Board, is formed to manage requirements, and all requests for changes need to be forwarded to this committee, which then decides on the necessity for the changes and, if required, updates the requirements and disseminates change notices.

Another facet to requirements management is the development of traceability tables where each traceability table relates requirements to an aspect of the system. Thus, for example, there could be a features traceability table, source traceability table, dependency traceability table, subsystem traceability table, and interface traceability table. The features traceability table relates each requirement to a feature (or functionality) of the system, the source traceability table relates each requirement to its source, the dependency traceability table indicates how requirements are dependent on each other, the subsystem traceability table relates each requirement to the subsystem the requirement applies to, and the interface traceability table captures the relationship between requirements and system interfaces.

TOOL SUPPORT FOR REQUIREMENTS SPECIFICATION AND ANALYSIS

The most important software tools for requirements specification are a word processor or spreadsheet software. Both of them help capture software requirements—the advantage with a word processor is that diagrams and detailed descriptions can be captured, whereas the main advantage of a spreadsheet software is that the requirements can be represented in tabular forms that help to add, delete, and modify requirements easily. Also, both tools help capture categorized or nested requirements. Some modeling environments such as IBM's Rational Requisite Pro (13) and Telelogic's System Architect (14) help draw use-cases and determine requirements dependencies. Drawing applications such as Smartdraw (15) and Microsoft Visio (16) have libraries for modeling requirements using UML, ERD, and

DFD notations. In order to capture different versions of requirements a configuration management tool such as open-source Concurrent Versions System (17) or IBM's Rational ClearCase (13) may be useful. Very often the versioning system is integrated with a Web-based system that is usually developed for each project so that interfaces for viewing and uploading the latest requirements are readily available. Sometimes, the Web-based repository serves as a bulletin board service as well so that all update notifications are issued at one place—the Web-based system then becomes a central repository for all data pertaining to the project including requirements.

DOORS (Dynamic Object Oriented Requirements System) (18) marketed by Telelogic is a tool for collaborative requirements management that helps capture, view, and update the latest requirements for a project. In addition, DOORS can help with change management and can be integrated with third-party application tools.

The Automated Requirements Measurement (ARM) (19) tool from NASA can evaluate requirements specified in the English language for words, including imperatives, directives, continuances, options, and weak phrases. These constructs are measured in terms of size, specification depth, readability, and text structure.

RESEARCH PROBLEMS IN REQUIREMENTS SPECIFICATION AND ANALYSIS

As stated, one of the important problems with the requirements specification and analysis phase is that users are frequently unable to articulate their needs—the roots for this problem may lie in an interdisciplinary area overlapping the fields of neuroscience, psychology, and cognition. A team of interdisciplinary researchers need to analyze the causes of this problem so that tools—both hardware and software—to better elicit requirements from people may be developed. Another problem is that of modeling requirements for communication among stakeholders (20)—current communication tools that use natural language such as English are prone to ambiguities, whereas a modeling language such as UML requires training for proper usage. Another issue that requires additional study is that of analysis paralysis, which refers to the problem of being in a state of perpetual analysis with no end to the phase in sight—the answer to the question "*When has enough analysis been done?*" is of considerable interest to practitioners (21). A very useful development in this area would be identification of conceptual patterns of user needs so that these concepts may be used as building blocks for developing requirements for new systems; in addition, these concepts may be used as a mechanism for communicating with users. For example, concepts such as human–machine interaction, data access, and networking seem to be common for most systems, and these could be designated as patterns that can be used for building new systems. Yet another research area is quantification of requirements quality (10) so that practitioners know when they are done—currently this field is in its infancy with tools such as ARM providing mainly syntactic metrics, but it needs to be developed to include semantic or conceptual levels.

BIBLIOGRAPHY

1. IEEE Std 830-1998, *Recommended Practice for Software Requirements Specifications*, June 25, 1998.
2. J. L. Whitten and L. D. Bentley, *Systems Analysis and Design Methods*, 7th ed. New York: McGraw-Hill, 2007.
3. R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed. New York: McGraw-Hill, 2005.
4. S. R. Schach, *Object-Oriented & Classical Software Engineering*, 6th ed. New York: McGraw-Hill, 2005.
5. S. L Pfleeger and J. M. Atlee, *Software Engineering: Theory and Practice*, 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 2006.
6. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Boston, MA: Kluwer Academic Publishers, 2000.
7. R. Denney, *Succeeding with Use Cases: Working Smart to Deliver Quality*. Reading, MA: Addison-Wesley Professional, 2005.
8. M. Cohn, *User Stories Applied: For Agile Software Development*. New York: Wiley, 2004.
9. J. Wetherbe and N. P. Vitalari, *Systems Analysis and Design: Traditional, Best Practices*, 4th ed. St. Paul, MN: West Publishing, 1994.
10. B. W. Boehm, C. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, Cost models for future life cycle processes: COCOMO 2.0, *Ann. Soft. Eng.*, **1**(1): 57–94, 1995.
11. L. Chung and N. Subramanian, Process-oriented metrics for software architecture adaptability, *Proc. International Symposium on Requirements Engineering*, IEEE Computer Press, Aug–Sep. 2001, pp. 310–311.
12. A. van Lamsweerde, Requirements engineering in the year 00: a research perspective, *Proc. 22nd International Conference on Software Engineering (ICSE'00)*, Limerick, Ireland, June 5–9 2000, pp. 5–19.
13. P. Zave and M. Jackson, Four dark corners of requirements engineering, *ACM Trans. Soft. Eng. Methodology*, **6**(1): 1–30, 1997.
14. www.uml.org
15. http://satc.gsfc.nasa.gov/tools/arm/
16. http://www.telelogic.com/corp/products/doors/index.cfm
17. http://www-306.ibm.com/software/rational/
18. http://www.nongnu.org/cvs/
19. www.smartdraw.com
20. http://office.microsoft.com/en-us/visio/default.aspx
21. http://www.telelogic.com/products/systemarchitect/index.cfm

NARAYANAN SUBRAMANIAN
University of Texas at Tyler
Tyler, Texas

SOFTWARE AGING AND REJUVENATION

INTRODUCTION

Several studies have now shown that outages in computer systems are more due to software faults than due to hardware faults (1,2). Recent studies have also reported the phenomenon of "software aging" (3,4) in which the state of the software degrades with time. The primary causes of this degradation are the exhaustion of operating system resources, data corruption, and numerical error accumulation, which eventually may lead to performance degradation of the software, crash/hang failure, or both. Some common examples of "software aging" are memory bloating and leaking, unreleased file-locks, data corruption, storage space fragmentation, and accumulation of round-off errors (3). Aging has not only been observed in software used on a mass scale but also in specialized software used in high-availability and safety-critical applications (4). This type of aging in operational software systems is different from code decay in software systems caused by maintenance (5,6). The former results in performance problems, system slow downs, and crashes, whereas the latter results in unrunnable or invalid software and maintenance-induced bugs.

As aging leads to transient failures in software systems, environment diversity, a software fault-tolerance technique, can be employed proactively to prevent degradation or crashes, which involves occasionally stopping the running software, "cleaning" its internal state or its environment and restarting it. Such a technique known as "software rejuvenation" was proposed by Huang et al. (4,7,8), [1] which counteracts the aging phenomenon in a proactive manner by removing the accumulated error conditions and freeing up operating system resources. Garbage collection, flushing operating system kernel tables, and reinitializing internal data structures are some examples by which the internal state or the environment of the software can be cleaned.

Software rejuvenation has been implemented in the AT&T billing applications (4). An extreme example of a system-level rejuvenation, proactive hardware reboot, has been implemented in the real-time system collecting billing data for most telephone exchanges in the United States (9). Occasional reboot is also performed in the AT&T telecommunications switching software (10). On reboot, called *software capacity restoration*, the service rate is restored to its peak value. On-board preventive maintenance in spacecraft has been proposed and analyzed by Tai et al. (11), which maximizes the probability of successful mission completion by the spacecraft. These operations, called *operational redundancy*, are invoked whether or not faults exist. Proactive fault management was also recommended for the Patriot missiles' software system (12,13). A warning was issued saying that a very long running time could affect the targeting accuracy. This decrease in accuracy was evidently due to overflow in the counter keeping track of time, during conversion from integer to real numbers. The longer the system ran continuously, the larger the error became. The warning, however, failed to inform the troops how many hours "very long" was and that it would help if the computer system was switched off and on every eight hours, which exemplifies the necessity and the use of proactive fault management even in safety critical systems. More recently, rejuvenation has been implemented in cluster systems to improve performance and availability (14–17). Two kinds of policies have been implemented taking advantage of the cluster failover feature. In the periodic policy, rejuvenation of the cluster nodes is done in a rolling fashion after every deterministic interval. In the prediction-based policy, the time to rejuvenate is estimated based on the collection and statistical analysis of system data. The implementation and analysis are described in detail in Refs. 14 and 15. A software rejuvenation feature known as process recycling has been implemented in the Microsoft IIS 5.0 web server software (18). The popular web server software Apache implements a form of rejuvenation by killing and recreating processes after a certain numbers of requests have been served (19,20). Software rejuvenation has been proposed for specialized transaction processing servers (21), cable and DSL modem gateways (22), in Motorola's Cable Modem Termination System (23), and in middleware applications (24) for failure detection and prevention. Automated rejuvenation strategies have been proposed in the context of self-healing and autonomic computing systems (25). Recently, recursive restarts and micro-reboot has been proposed to increase availability (26). Software rejuvenation (preventive maintenance) incurs an overhead (in terms of performance, cost, and downtime), which should be balanced against the loss incurred due to unexpected outage caused by a failure. Thus, an important research issue is to determine the optimal times to perform rejuvenation.

Here, we present two approaches for analyzing software aging and studying aging-related failures. The rest of this article is organized as follows: The next section describes various analytical models for software aging and to determine optimal times to perform rejuvenation. Measurement-based models are dealt with next, followed by discussion of the implementation of a software rejuvenation agent in a major commercial server, various approaches and methods of rejuvenation. The article concludes with pointers to future work.

[1] Although we use the by-now-established phrase "software aging," it should be clear that no deterioration of the software system per se is implied but rather the software appears to age due to the gradual depletion of resources (8). Likewise, "software rejuvenation" actually refers to rejuvenation of the environment in which the software is executing.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

ANALYTIC MODELS FOR SOFTWARE REJUVENATION

The aim of the analytic modeling is to determine optimal times to perform rejuvenation that maximizes *availability* or minimizes the *probability of loss* or minimizes the *mean response time of a transaction* (in the case of a transaction processing system), which is particularly important for business-critical applications for which adequate response time can be as important as system uptime. The analysis is done for different kinds of software systems exhibiting varied failure/aging characteristics.

The accuracy of a model-based approach is determined by the assumptions made in capturing aging. In Refs. 4,11,27–29, only the failures causing unavailability of the software are considered, whereas in Ref. 30 only a gradually decreasing service rate of a software that serves transactions is assumed. Garg et al. (31), however, consider both these effects of aging together in a single model. Models proposed in Refs. 4,27,28 are restricted to hypo-exponentially distributed time to failure. Those proposed in Refs. 11,29,30 can accommodate general distributions but only for the specific aging effect they capture. Generally distributed time to failure, as well as the service rate being an arbitrary function of time are allowed in Ref. 31. It has been noted (2) that transient failures are partly caused by overload conditions. Only the model presented by Garg et al. (31) captures the effect of load on aging. Existing models also differ in the measures being evaluated. In Refs. 11 and 29, software with a finite mission time is considered. In Refs. 4,27,28,31, measures of interest in a transaction-based software intended to run forever are evaluated.

Bobbio et al. (32) present fine-grained software degradation models, where one can identify the current degradation level based on the observation of a system parameter. Optimal rejuvenation policies based on a risk criterion and an alert threshold are then presented. Dohi et al. (33,34) present software rejuvenation models based on semi-Markov processes. The models are analyzed for optimal rejuvenation strategies based on cost as well as steady-state availability. Given a sample data of failure times, statistical non-parametric algorithms based on the total time on test transform are presented to obtain the optimal rejuvenation interval.

Basic Model for Rejuvenation

Figure 1 shows the basic software rejuvenation model proposed by Huang et al. (4). The software system is initially in a “robust” working state, 0. As time progresses, it eventually transits to a “failure-probable” state, 1. The system is still operational in this state, but can fail (move to state 2) with a non-zero rate. The system can be repaired and brought back to the initial state, 0. The software system is also rejuvenated at regular intervals from the failure probable state 1 and brought back to the robust state 0.

Huang et al. (4) assume that the stochastic behavior of the system can be described by a simple homogeneous continuous-time Markov chain (CTMC) (35). The CTMC is then analyzed and the expected system downtime and the expected cost per unit time in the steady state are computed. An optimal rejuvenation interval that minimizes expected downtime (or expected cost) is obtained.

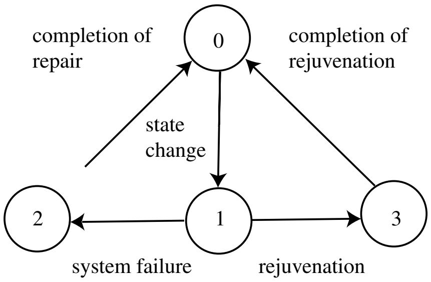


Figure 1. State transition diagram for rejuvenation.

It is not difficult to introduce the periodic rejuvenation schedule and to extend the CTMC model to the general one. Dohi et al. (33,34) developed semi-Markov models with the periodic rejuvenation and general transition distribution functions. Garg et al. (27) have developed a Markov Regenerative Stochastic Petri Net (MRSPN) model where rejuvenation is performed at deterministic intervals assuming that the failure probable state 1 is not observable.

Software Rejuvenation in Transactions-Based Software Systems

In Ref. 31, Garg et al. consider a transaction-based software system whose macro-states representation is presented in Fig. 2. The state in which the software is available for service (albeit with decreasing service rate) is denoted as state *A*. After failure, a recovery procedure is started. In state *B*, the software is recovering from failure and is unavailable for service. Lastly, the software occasionally undergoes rejuvenation, denoted by state *C*. Rejuvenation is allowed only from state *A*. Once recovery from failure or rejuvenation is complete, the software is reset to state *A* and is as good as new. From this moment, which constitutes a renewal, the whole process stochastically repeats itself.

The system consists of a server-type software to which transactions arrive at a constant rate. The effect of aging in the model may be captured by using decreasing service rate and increasing failure rate, where the decrease or the increase respectively can be a function of time, instantaneous load, mean accumulated load, or a combination of the above.

Two policies that can be used to determine the time to perform rejuvenation are considered. Under policy I, which is purely time-based, rejuvenation is initiated after a constant time δ has elapsed since it was started (or restarted). Under policy II, which is based on instantaneous load and

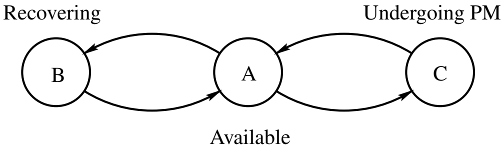


Figure 2. Macro-states representation of the software behavior.

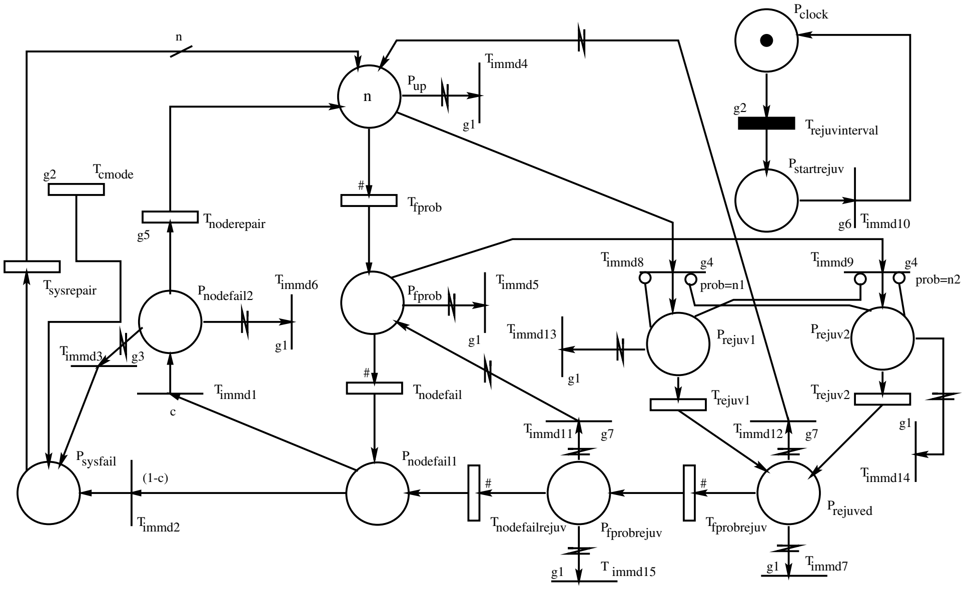


Figure 3. SRN model of a cluster system employing simple time-based rejuvenation.

time, a constant waiting period δ must elapse before rejuvenation is attempted. After this time, rejuvenation is initiated if there are no transactions in the system. Otherwise, the software waits until the queue is empty upon which rejuvenation is initiated. The goal of the analysis is to determine optimal values of δ (rejuvenation interval under policy I and rejuvenation wait under policy II) different objective functions such as the availability, the loss probability, and the mean response time.

Software Rejuvenation in a Cluster System

Software rejuvenation has been applied to cluster systems (14,16), which significantly improves cluster system availability and productivity. The Stochastic Reward Net (SRN) model of a cluster system employing simple time-based rejuvenation is shown in Fig. 3. The cluster consists of n nodes, which are initially in a "robust" working state, P_{up}. The aging process is modeled as a two-stage hypoexponential distribution (increasing failure rate) (35) with transitions T_{fprob} and $T_{noderepair}$. Place P_{fprob} represents a "failure-probable" state in which the nodes are still operational. The nodes then can eventually transit to the fail state, $P_{nodefail1}$. A node can be repaired through the transition $T_{noderepair}$, with a coverage c. In addition to individual node failures, there is also a common-mode failure (transition T_{cmode}). The system is also considered down when there are a ($a \leq n$) individual node failures. The system is repaired through the transition $T_{sysrepair}$.

In the simple time-based policy, rejuvenation is done successively for all the operational nodes in the cluster, at the end of each deterministic interval. The transition $T_{rejuvinterval}$ fires every δ time units depositing a token in place $P_{startrejuv}$. Only one node can be rejuvenated at any time (at places P_{rejuv1} or P_{rejuv2}). Weight functions are assigned such that the probability of selecting a token from P_{up} or P_{fprob} is directly proportional to the number of tokens in each. After a node has been rejuvenated, it goes back to the "robust" working state, represented by place $P_{rejuved}$, which is a clone place for P_{up} in order to distinguish the nodes that are waiting to be rejuvenated from the nodes that have already been rejuvenated. A node, after rejuvenation, is then allowed to fail with the same rates as before rejuvenation even when another node is being rejuvenated. Clone places for P_{upb} and P_{fprob} are needed to capture this result. Node repair is disabled during rejuvenation. Rejuvenation is complete when the sum of nodes in places $P_{rejuved}$, $P_{fprobrejuv}$, and $P_{nodefail2}$ is equal to the total number of nodes, n. In this case, the immediate transition T_{immd10} fires, putting back all the rejuvenated nodes in places P_{up} and P_{fprob}. Rejuvenation stops when there are $a-1$ tokens in place $P_{nodefail2}$, to prevent a system failure. The clock resets itself when rejuvenation is complete and is disabled when the system is undergoing repair. Guard functions ($g1$ through $g7$) are assigned to express complex enabling conditions textually.

For the analysis, the following values are assumed. The mean times spent in places P_{up} and P_{fprob} are 240 hrs and 720 hrs, respectively. The mean times to repair a node, to rejuvenate a node, and to repair the system are 30 mins, 10 mins, and 4 hrs, respectively. In this analysis, the common-mode failure is disabled and node failure coverage is assumed to be perfect. All the models were solved using the SPNP (Stochastic Petri Net Package) tool (36). The measures computed were expected downtime and the expected cost incurred over a fixed time interval. It is assumed that the cost incurred due to node rejuvenation is much less than the cost of a node or system failure since rejuvenation can be done at predetermined or scheduled times. In our analysis, we fix the value for $cost_{nodefail}$ at \$5,000/hr and the $cost_{rejuv}$ at \$250/hr. The value of $cost_{sysfail}$ is computed as the number of nodes, n, times $cost_{nodefail}$.

Figure 4 shows the plots for an 8/1 configuration (8 nodes including 1 spare) system employing simple time-based

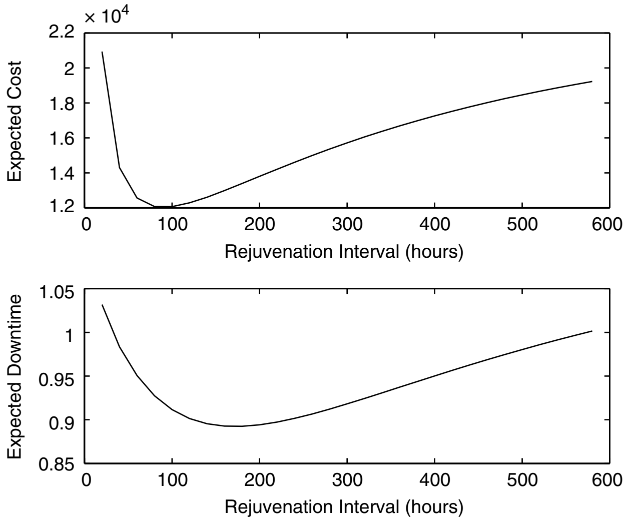


Figure 4. Results for an 8/1 cluster system employing time-based rejuvenation.

rejuvenation. The upper plot and lower plots show the expected cost incurred and the expected downtime (in hours), respectively, in a given time interval, versus rejuvenation interval (time between successive rejuvenation) in hours. If the rejuvenation interval is close to zero, the system is always rejuvenating and thus incurs high cost and downtime. As the rejuvenation interval increases, both expected downtime and cost incurred decrease and reach an optimum value. If the rejuvenation interval goes beyond the optimal value, the system failure has more influence on these measures than rejuvenation. The analysis was repeated for 2/1, 8/2, 16/1, and 16/2 configurations. For time-based rejuvenation, the optimal rejuvenation interval was 100 hours for the 1-spare clusters, and approximately 1 hour for the 2-spare clusters.

MEASUREMENT-BASED MODELS FOR SOFTWARE REJUVENATION

While all the analytical models are based on the assumption that the rate of software aging is known, in the measurement-based approach, the basic idea is to monitor and collect data on the attributes responsible for determining the health of the executing software. The data is then analyzed to obtain predictions about possible impending failures due to resource exhaustion.

In this section, we describe the measurement-based approach for detection and validation of the existence of software aging. The basic idea is to periodically monitor and collect data on the attributes responsible for determining the health of the executing software, in this case the UNIX operating system. Garg et al. (3) propose an approach for detection and estimation of aging in the UNIX operating system. An SNMP-based distributed resource monitoring tool was used to collect operating system resource usage and system activity data from nine heterogeneous UNIX workstations connected by an Ethernet LAN at the Department of Electrical and Computer Engineering at Duke University. A central monitoring station runs the manager program, which sends *get* requests periodically to each of the agent programs running on the monitored workstations. The agent programs, in turn, obtain data for the manager from their respective machines by executing various standard UNIX utility programs like *pstat*, *iostat*, and *vmstat*. For quantifying the effect of aging in operating system resources, the metric *Estimated time to exhaustion* is proposed.

In the time-based estimation method presented by Garg et al. (3), data was collected from the UNIX machines at intervals of 15 minutes for about 53 days. Time-ordered values for each monitored object are obtained, constituting a time series for that object. The objective is to detect aging or a long-term trend (increasing or decreasing) in the values. Only results for the data collected from the machine *Rossby* are discussed here.

First, the trends in operating system resource usage and system activity are detected using *smoothing* of observed data by *robust locally weighted regression*, proposed by Cleveland (3). This technique is used to get the global trend between outages by removing the local variations. Then, the slope of the trend is estimated in order to do prediction. Figure 5 shows the smoothed data superimposed on the original data points from the time series of objects for Rossby. Amount of *real memory free* (plot 1) shows an overall decrease, whereas *file table size* (plot 2) shows an increase. Plots of some other resources not discussed here

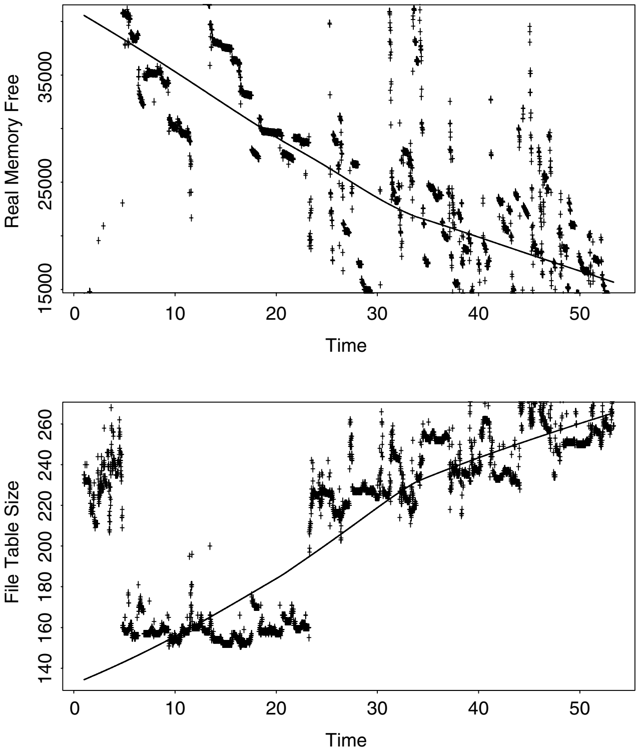


Figure 5. Non-parametric regression smoothing for Rossby objects.

Table 1. Estimated slope and time to exhaustion for Rossby, Velum, and Jefferson objects

Resource Name	Initial Value	Max Value	Sen's Slope Estimation	95% Confidence Interval	Estimated Time to Exh. (days)
Rossby					
Real Memory Free	40814.17	84980	−252.00	−287.75 : −219.34	161.96
File Table Size	220	7110	1.33	1.30 : 1.39	5167.50
Process Table Size	57	2058	0.43	0.41 : 0.45	4602.30
Used Swap Space	39372	312724	267.08	220.09 : 295.50	1023.50
Jefferson					
Real Memory Free	67638.54	114608	−972.00	−1006.81 : −939.08	69.59
File Table Size	268.83	7110	1.33	1.30 : 1.38	5144.36
Process Table Size	67.18	2058	0.30	0.29 : 0.31	6696.41
Used Swap Space	47148.02	524156	577.44	545.69 : 603.14	826.07

also showed an increase or decrease, which corroborates the hypothesis of aging with respect to various objects.

The seasonal Kendall test (3) was applied to each of these time series to detect the presence of any global trends at a significance level, α, of 0.05. With Z_α= 1.96, all values are such that the null hypothesis (H_0) that no trend exists is rejected for the variables considered. Given that a global trend is present and that its slope is calculated for a particular resource, the time at which the resource will be exhausted because of aging only is estimated. Table 1 refers to several objects on Rossby and lists an estimate of the slope (change per day) of the trend obtained by applying Sen's slope estimate for data with seasons (3). The values for real memory and swap space are in Kilobytes.

A negative slope, as in the case of *real memory*, indicates a decreasing trend, whereas a positive slope, as in the case of *file table size*, is indicative of an increasing trend. Given the slope estimate, the table lists the estimated time to failure of the machine due to aging only with respect to this particular resource. The calculation of the time to exhaustion is done by using the standard linear approximation $y = mx + c$.

The method discussed in Ref. 3 assumes that accumulated depletion of a resource over a time period depends only on the elapsed time. However, it is intuitive that the rate at which a resource is depleted is dependent on the current workload. In Refs. 37 and 38, a measurement-based model to estimate the rate of exhaustion of operating system resources as a function of both time and the system workload is discussed. The SNMP-based distributed resource monitoring tool described previously was used for collecting operating system resource usage and system activity parameters (at 10 min intervals) for over 3 months. Only results for the data collected from the machine Rossby are discussed here. The *longest* stretch of sample points in which no reboots or failures occurred were used for building the model. A semi-Markov reward model (39) is constructed using the data. First, different workload states are identified using statistical cluster analysis and a state-space model is constructed. Corresponding to each resource, a reward function based on the rate of resource exhaustion in the different states is then defined. Finally, the model is solved to obtain trends and the estimated exhaustion rates and time to exhaustion for the resources.

A methodology based on time-series analysis to detect and estimate resource exhaustion times due to software aging in a web server while subjecting it to an artificial workload is proposed in Ref. 19. The experiments are conducted on an Apache web server running on the Linux platform.

The analysis can be done using two different approaches: (1) building a univariate model for each of the outputs or, (2) building only one multivariate model with seven outputs. In this case, seven univariate models are built and then combined into a single multivariate model. First, the parameters are determined to determine their characteristics and build an appropriate model with one output and four inputs for each parameter—connection rate, linear trend, periodic series with a period of one week, and periodic series with a period of one day. The autocorrelation function (ACF) and the partial autocorrelation function (PACF) for the output are computed. The ACF and the PACF help us decide the appropriate model for the data (40). For example, from the ACF and PACF of *used swap space*, it can be determined that an autoregressive model of order 1 [AR(1)] is suitable for this data series. Adding the inputs to the AR(1) model, we get the ARX(1) model for used swap space:

$$Y_t = aY_{t-1} + b_1X_t + b_2L_t + b_3W_t + b_4D_t \tag{1}$$

where Y_t is the used swap space, X_t is the connection rate, L_t is the time step that represents the linear trend, W_t is the weekly periodic series, and D_t is the daily periodic series. After observing the ACF and PACF of all the parameters, we find that all of the PACFs cut off at certain lags. So all the multiple input single output (MISO) models are of the ARX type, only with different orders, which gives great convenience in combining them into a multiple input multiple output (MIMO) ARX model, which is described later.

In order to combine the MISO ARX models into a MIMO ARX model, we need to choose the order between different outputs, which is done by inspecting the CCF (cross-correlation function) between each pair of the outputs to find out the leading relationship between them. If the CCF between parameter A and B gets its peak value at a positive lag k, we say that A leads B by k steps and it might be possible to use A to predict B. In our analysis, there are

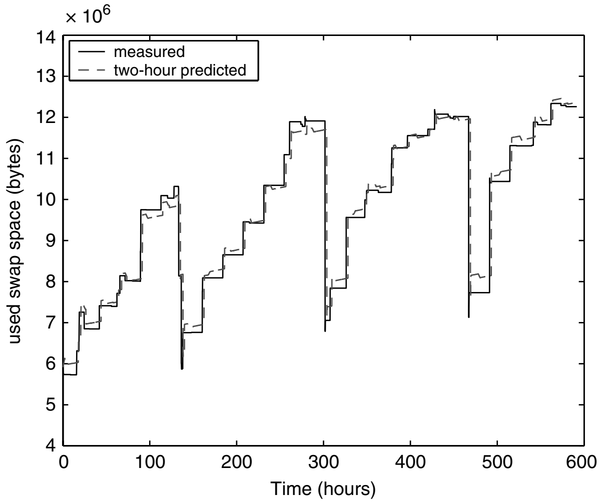


Figure 6. Measured and two-hour-ahead predicted used swap space.

21 CCFs that need to be computed. And, in order to reduce the complexity, we only use the CCFs that exhibit obvious leading relationship with lags less than 10 steps. The next step after determination of the orders is to estimate the coefficients of the model by the least squares method. The first half of the data is used to estimate the parameters and the rest of the data is then used to verify the model. Figure 6 shows the two-hour-ahead (24-step) predicted used swap space, which is computed using the established model and the data measured up to two hours before the predicted time point. From the plots, we can see that the predicted values are very close to the measured values.

In Ref. 8, a model is developed to account for the gradual loss of system resources, especially the memory resource. In a client-server system, for example, every client process issues memory requests at varying points in time. An amount of memory is granted to each new request (when there is enough memory available), held by the requesting process for a period of time, and presumably released back to the system resource reservoir when it is no longer in use. A memory leak occurs when the amount of allocated memory is not fully released. The available memory space is gradually reduced as such resource leaks accumulate over time. As a consequence, a resource request that would have been granted in the leak-less situation may not be granted when the system suffers from memory resource leaks. This model accommodates both the leak-free case and the leak-present case. The model relates system degradation to resource requests, releases or resource holding intervals, and memory leaks. These quantities can be monitored and modeled directly from obtainable data measurements (19).

Avritzer and Weyuker (10) monitor production traffic data of a large telecommunication system and describe a rejuvenation strategy that increases system availability and minimizes packet loss. Cassidy et al. (21) have developed an approach to rejuvenation for large online transaction processing servers. They monitor various system parameters over a period of time. Using pattern recognition methods, they come to the conclusion that 13 of those parameters deviate from normal behavior just before a crash, providing sufficient warning to initiate rejuvenation.

IMPLEMENTATION OF A SOFTWARE REJUVENATION AGENT

The first commercial version of a software rejuvenation agent (SRA) for the IBM xSeries line of cluster servers has been implemented with our collaboration (14–16). The SRA was designed to monitor consumable resources, estimate the time to exhaustion of those resources, and generate alerts to the management infrastructure when the time to exhaustion is less than a user-defined notification horizon. For Windows operating systems, the SRA acquires data on exhaustible resources by reading the registry performance counters and collecting parameters such as available bytes, committed bytes, non-paged pool, paged pool, handles, threads, semaphores, mutexes, and logical disk utilization. For Linux, the agent accesses the /proc directory structure and collects equivalent parameters such as memory utilization, swap space, file descriptors and inodes. All collected parameters are logged on to disk. They are also stored in memory preparatory to time-to-exhaustion analysis.

In the current version of the SRA, rejuvenation can be based on elapsed time since the last rejuvenation or on prediction of impending exhaustion. When using timed rejuvenation, a user interface is used to schedule and perform rejuvenation at a period specified by the user. It allows the user to select when to rejuvenate different nodes of the cluster, and to select "blackout" times during which no rejuvenation is to be allowed. Predictive rejuvenation relies on curve-fitting analysis and projection of the use of key resources, using recently observed data. The projected data is compared with prespecified upper and lower exhaustion thresholds, within a notification time horizon. The user specifies the notification horizon and the parameters to be monitored (some parameters believed to be highly indicative are always monitored by default), and the agent periodically samples the data and performs the analysis. The prediction algorithm fits several types of curves to the data in the fitting window. These different curve types have been selected for their ability to capture different types of temporal trends. A model-selection criterion is applied to choose the "best" prediction curve, which is then extrapolated to the user-specified horizon. The several parameters that are indicative of resource exhaustion are monitored and extrapolated independently. If any monitored parameter exceeds the specified minimum or maximum value within the horizon, a request to rejuvenate is sent to the management infrastructure. In most cases, it is also possible to identify which process is consuming the preponderance of the resource being exhausted, in order to support selective rejuvenation of just the offending process or a group of processes.

APPROACHES AND METHODS OF SOFTWARE REJUVENATION

Software rejuvenation can be divided broadly into two approaches as follows:

- **Open-loop approach:** In this approach, rejuvenation is performed without any feedback from the system. Rejuvenation, in this case, can be based just on elapsed time (periodic rejuvenation) (4,27) or instantaneous/cumulative number of jobs on the system (31).
- **Closed-loop approach:** In the closed-loop approach, rejuvenation is performed based on information on the system "health." The system is monitored continuously (in practice, at small deterministic intervals) and data is collected on the operating system resource usage and system activity. This data is then analyzed to estimate time to exhaustion of a resource that may lead to a component or an entire system degradation/crash. This estimation can be based purely on time and workload-independent (3,14), or it can be based on both time and system workload (37,38).

  The closed-loop approach can be further classified based on whether the data analysis is done offline or online. Offline data analysis is done based on system data collected over a period of time (usually weeks or months). The analysis is done to estimate time to rejuvenation. This offline analysis approach is best suited for systems whose behavior is fairly deterministic (37,38). The online closed-loop approach, on the other hand, performs online analysis of system data collected at deterministic intervals (14). Another approach to estimate the optimal time to rejuvenation could be based on system failure data (34).

This classification of approaches to rejuvenation is shown in Fig. 7.

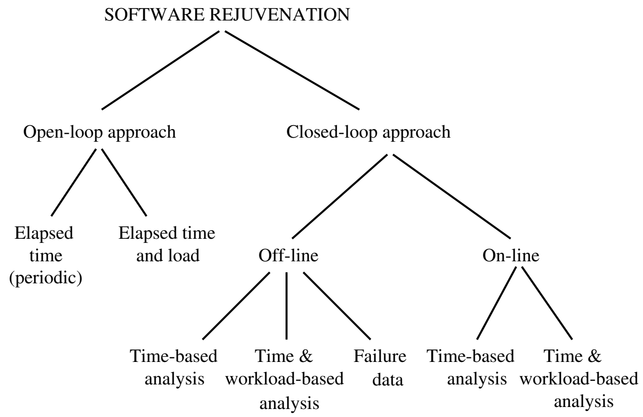


Figure 7. Approaches to software rejuvenation.

Rejuvenation is a very general proactive fault management approach and can be performed at different levels—the system level or the application level. An example of a system-level rejuvenation is a hardware-reboot. At the application level, rejuvenation is performed by stopping and restarting a particular offending application, process, or a group of processes, also known as a *partial rejuvenation*. The above rejuvenation approaches when performed on a single node can lead to undesired and often costly downtime. Rejuvenation has been recently extended for cluster systems, in which two or more nodes work together as a single system (14,16). In this case, rejuvenation can be performed by causing no or minimal downtime by failing over applications to another spare node.

CONCLUSIONS

In this article, various analytical models for software aging and to determine optimal times to perform rejuvenation were described. Measurement-based models based on data collected from operating systems were also discussed. The implementation of a software rejuvenation agent in a major commercial server was then briefly described. Finally, various approaches to rejuvenation and rejuvenation granularity were discussed.

In the measurement-based models presented in this article, only aging due to each individual resource has been captured. In the future, one could improve the algorithm used for aging detection to involve multiple parameters simultaneously, for better prediction capability and reduced false alarms. Dependencies between the various system parameters could be studied. The best statistical data analysis method for a given system is also yet to be determined.

BIBLIOGRAPHY

1. J. Gray and D. P. Siewiorek, High-availability computer systems, *IEEE Computer*, 1991, pp. 39–48.
2. M. Sullivan and R. Chillarege, Software defects and their impact on system availability – A study of field failures in operating systems, *Proc. 21st IEEE Int'l. Symposium on Fault-Tolerant Computing*, 1991, pp. 2–9.
3. S. Garg, A. Van Moorsel, K. Vaidyanathan, and K. Trivedi, A methodology for detection and estimation of software aging, *Proc. of 9th Int'l. Symposium on Software Reliability Engineering*, Paderborn, Germany, 1998, pp. 282–292.
4. Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, Software rejuvenation: Analysis, module and applications, *Proc. of 25th Symposium on Fault Tolerant Computing, FTCS-25*, Pasadena, California, 1995, pp. 381–390.
5. S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, Does code decay? Assessing the evidence from change management data, *IEEE Trans. Software Eng.*, **27** (1): 1–12, 2001.
6. D. L. Parnas, Software Aging, *Proc. 16th Int'l. Conf. on Software Engineering*, Sorrento, Italy, 1994, pp. 279–287.
7. Available: http://www.software-rejuvenation.com.
8. Y. Bao, X. Sun, and K. Trivedi, A workload-based analysis of software aging and rejuvenation, *IEEE Trans. Reliability*, **54** (3): 541–548, 2005.
9. L. Bernstein, Text of Seminar Delivered by Mr. Bernstein. *University Learning Center*, George Mason University, January 29, 1996.
10. A. Avritzer and E. J. Weyuker, Monitoring Smoothly Degrading Systems for Increased Dependability. *Empirical Software Eng. J.*, **2** (1): 59–77, 1997.

11. A. T. Tai, S. N. Chau, L. Alkalaj, and H. Hecht, On-board preventive maintenance: Analysis of effectiveness and optimal duty period, *3rd Int'l. Workshop on Object Oriented Real-time Dependable Systems*, Newport Beach, CA, 1997.

12. L. Bernstein and C. M. R. Kintala, Software Rejuvenation. CrossTalk – J. Defense Software Eng., August 2004.

13. E. Marshall, Fatal error: How patriot overlooked a scud, *Science*, 1347, 1992.

14. V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. Zeggert, Proactive management of software aging, *IBM J. R&D*, **45** (2): 2001.

15. IBM Netfinity Director Software Rejuvenation – White Paper, Research Triangle Park, NC:IBM Corp., Jan. 2001.

16. K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi, Analysis and implementation of software rejuvenation in cluster systems, *Proc. of the Joint Int'l. Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS 2001/Performance 2001*, Cambridge, MA, 2001.

17. W. Xie, Y. Hong, and K. S. Trivedi, Software rejuvenation policies for cluster systems under varying workload, *Proc. of Tenth Int'l. Pacific Rim Dependable Computing Symp., PRDC 2004*, Papeete, Tahiti, French Polynesia, 2004.

18. Available: http://www.microsoft.com/technet/prodtechnol/windows2000serv/technologies/iis/ default.mspx.

19. M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, Analysis of software aging in a web server, *IEEE Trans. Reliability*, **55** (3): 411–420, 2006.

20. Available: http://www.apache.org.

21. K. Cassidy, K. Gross, and A. Malekpour, Advanced pattern recognition for detection of complex software aging in online transaction processing servers, *Proc. of DSN 2002*, Washington D.C., 2002.

22. C. Fetzer and K. Hostedt, Rejuvenation and failure detection in partitionable systems, *Proc. of the Pacific Rim Int'l. Symposium on Dependable Computing, PRDC 2001*, Seoul, South Korea, 2001.

23. Y. Liu, Y. Ma, J. J. Han, H. Levendel, and K. S. Trivedi, Modeling and analysis of software rejuvenation in cable modem termination system, *Proc. of the Int'l. Symp. on Software Reliability Engineering, ISSRE 2002*, Annapolis, MD, 2002.

24. T. Boyd and P. Dasgupta, Premptive module replacement using the virtualizing operating system, *Proc. of the Workshop on Self-Healing, Adaptive and Self-Managed Systems, SHAMAN 2002*, New York, NY, 2002.

25. Y. Hong, D. Chen, L. Li, and K. S. Trivedi, Closed loop design for software rejuvenation, *Proc. of the Workshop on Self-Healing, Adaptive and Self-Managed Systems, SHAMAN 2002*, New York, NY, 2002.

26. G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, Microreboot, A technique for cheap recovery, *Proc. 6th Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, CA, 2004.

27. S. Garg, A. Puliafito, and K. S. Trivedi, Analysis of software rejuvenation using markov regenerative stochastic petri net, *Proc. of the Sixth Int'l. Symposium on Software Reliability Engineering*, Toulouse, France, 1995, pp. 180–187.

28. S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi, Time and load based software rejuvenation: Policy, evaluation and optimality, *Proc. of the First Fault-Tolerant Symposium*, Madras, India, 1995.

29. S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi, Minimizing completion time of a program by checkpointing and rejuvenation, *Proc. 1996 ACM SIGMETRICS Conference*, Philadelphia, PA, 1996, pp. 252–261.

30. A. Pfening, S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, Optimal rejuvenation for tolerating soft failures, *Perform. Eval.*, **27** & **28**: 491–506, 1996.

31. S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, Analysis of preventive maintenance in transactions based software systems, *IEEE Trans. Comput.*, **47**(1): 96–107, 1998.

32. A. Bobbio, A. Sereno, and C. Anglano, Fine grained software degradation models for optimal rejuvenation policies, *Perform. Eval.*, **46**: 45–62, 2001.

33. T. Dohi, K. Goseva–Popstojanova, and K. S. Trivedi, Analysis of software cost models with rejuvenation, *Proc. of the 5th IEEE International Symposium on High Assurance Systems Engineering, HASE 2000*, Albuquerque, NM, 2000.

34. T. Dohi, K. Goseva–Popstojanova, and K. S. Trivedi, Statistical Non-Parametric Algorithms to Estimate the Optimal Software Rejuvenation Schedule, *Proc. of the 2000 Pacific Rim International Symposium on Dependable Computing, PRDC 2000*, Los Angeles, CA, 2000.

35. K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, 2nd ed., New York: Wiley, 2001.

36. C. Hirel, B. Tuffin, and K. S. Trivedi, SPNP: Stochastic Petri Net Package. Version 6.0. B. R. Haverkort et al. (eds.), TOOLS 2000, Lecture notes in computer science 1786, Heidelberg: Springer-Verlag, 2000, pp. 354–357.

37. K. Vaidyanathan and K. S. Trivedi, A comprehensive model for software rejuvenation, *IEEE Trans. on Dependable and Secure Computing*, **2** (2): 124–137, 2005.

38. K. Vaidyanathan and K. S. Trivedi, A comprehensive model for software rejuvenation, *IEEE Trans. on Dependable and Secure Computing*, Apr. 2005 (in press).

39. K. S. Trivedi, J. Muppala, S. Woolet, and B. R. Haverkort, Composite performance and dependability analysis, *Perform. Eval.*, **14**(3–4): 197–216, 1992.

40. R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications*, New York: Springer-Verlag, 2000.

FURTHER READING

E. Adams, Optimizing Preventive Service of the Software Products, *IBM J. R&D*, **28** (1): 2–14, 1984.

Kishor S. Trivedi
Duke University
Durham, North Carolina
Kalyanaraman Vaidyanathan
Scalable Systems Group,
Sun Microsystems, Inc.
San Diego, California

SOFTWARE ARCHITECTURE

INTRODUCTION

During the 1990s, architectural design emerged as an important subfield of software engineering. Practitioners have come to realize that having a good architectural design is a critical success factor for complex system development. A good architecture can help ensure that a system will satisfy key requirements in such areas as performance, reliability, portability, scalability, and interoperability. A bad architecture can be disastrous.

Practitioners have also begun to recognize the value of making explicit architectural choices and leveraging past architectural designs in the development of new products. Today, there are numerous books on architectural design, regular conferences and workshops devoted specifically to software architecture, a growing number of commercial tools to aid in aspects of architectural design, courses in software architecture, major government and industrial research projects centered on software architecture, and an increasing number of formal architectural standards. Codification of architectural principles, methods, and practices has begun to lead to repeatable processes of architectural design, criteria for making principled tradeoffs among architectures, and standards for documenting, reviewing, and implementing architectures.

THE ROLES OF SOFTWARE ARCHITECTURE

What exactly is meant by the term “software architecture?” If we look at the common uses of the term “architecture” in software, we find that it is used in different ways, often making it difficult to understand what aspect is being addressed. Among the uses are: (1) the architecture of a particular system, as in “the architecture of system S contains components $C_1 \dots C_n$”; (2) an architectural style, as in “system S adopts a client-server architecture”; and (3) the general study of architecture, as in “there are many books on software architecture.”

Within software engineering, however, most uses of the term focus on the first of these interpretations. A typical definition is:

> The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them (1).

Although numerous similar definitions of software architecture exist, at the core of all of them is the notion that the architecture of a system describes its gross structure using one or more views. The structure in a view illuminates a set of top-level design decisions, including things such as how the system is composed of interacting parts, where the main pathways of interaction are, and what the key properties of the parts are. Additionally, an architectural description ideally includes sufficient information to allow high-level analysis and critical appraisal.

Software architecture typically plays a key role as a bridge between requirements and code (see Fig. 1).

By providing an abstract description (or model) of a system, the architecture exposes certain properties while hiding others. Ideally, this representation provides an intellectually tractable guide to the overall system, permits designers to reason about the ability of a system to satisfy certain requirements, and suggests a blueprint for system construction and composition.

For example, an architecture for a signal processing application might be constructed as a dataflow network in which the nodes read input streams of data, transform that data, and write to output streams. Designers might use this decomposition, together with estimated values for input data flows, computation costs, and buffering capacities, to reason about possible bottlenecks, resource requirements, and schedulability of the computations.

To elaborate, software architecture can play an important role in at least six aspects of software development.

1. *Understanding:* Software architecture simplifies our ability to comprehend large systems by presenting them at a level of abstraction at which a system’s design can be easily understood (2–4). Moreover, at its best, architectural description exposes the high-level constraints on system design, as well as the rationale for specific architectural choices.
2. *Reuse:* Architectural design can support reuse in several ways. Current work on reuse generally focuses on component libraries. Architectural design supports, in addition, both reuse of large components (or subsystems) and also frameworks into which components can be integrated. Such reusable frameworks may be domain-specific software architectural styles (5,6), component integration standards (7), and architectural design patterns (8).
3. *Construction:* An architectural description provides a partial blueprint for development by indicating the major software components and dependencies between them. For example, a layered view of an architecture typically documents abstraction boundaries between parts of a system’s implementation, clearly identifying the major internal system interfaces, and constraining what parts of a system may rely on services provided by other parts (2).
4. *Evolution:* Software architecture can expose the dimensions along which a system is expected to evolve. By making explicit the “load-bearing walls” of a system, system maintainers can better understand the ramifications of changes, and thereby more accurately estimate costs of modifications. Moreover, architectural descriptions separate concerns about the functionality of a component from the ways in

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

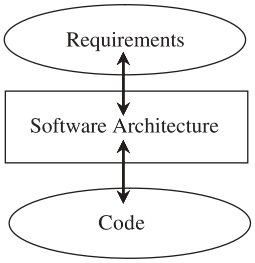


Figure 1. Software architecture as a bridge.

which that component interacts with other components, by clearly distinguishing between components and mechanisms that allow them to interact. This separation permits one to more easily change connection mechanisms to handle evolving concerns about performance and reuse.

5. *Analysis:* Architectural descriptions provide new opportunities for analysis, including system consistency checking (9,10), conformance to constraints imposed by an architectural style (11), conformance to quality attributes (12), dependence analysis (13), and domain-specific analyses for architectures built in specific styles (14–16).
6. *Management:* Experience has shown that successful projects view achievement of a viable software architecture as a key milestone in an industrial software development process. Critical evaluation of an architecture typically leads to a much clearer understanding of requirements, implementation strategies, and potential risks (17).
7. *Communication:* An architectural description often serves as a vehicle for communication among stakeholders. For example, explicit architectural design reviews allow stakeholders to voice opinions about relative weights of features and quality attributes when architectural tradeoffs must be considered (12).

PRECURSORS

The notion of providing explicit descriptions of system structures goes way back. In the 1960s and 1970s there were active debates about criteria on which to base modularization of software (18,19). Programming languages began to provide new features for modularization and the specification of interfaces.

In 1975, DeRemer and Kron (20) argued that creating program modules and connecting them to form larger structures were distinct design efforts. They created the first module interconnection language (MIL) to support that connection effort. In an MIL, modules import and export resources, which are named programming-language elements such as type definitions, constants, variables, and functions. A compiler for an MIL ensures system integrity using intermodule-type checking. Since DeRemer and Kron's proposal, other MILs have been developed for specific programming languages such as Ada and Standard ML, and have provided a base from which to support software construction, version control, and system families (21,22). Enough examples are available to develop models of the design space (23).

These early efforts to develop good ways to talk about system structures and to provide criteria for software modularization focused primarily on the problem of code organization and relationships between the parts based on interactions such as procedure call and simple data sharing. The key question was how to partition the software into units that could be implemented separately by software developers, and that would provide downstream benefits in support of extensibility, maintenance, and system understandability.

Today's view of software architecture builds on the insights and concepts from the early days of software structuring, but goes much further by also considering architectural representations that capture a system's run-time structures and behavior. By representing architectures as interacting components (viewed as actual run-time entities), these representations more directly facilitate reasoning about system properties such as performance, security, and reliability. Additionally, modern views of software architecture provide a much richer notion of interaction (than procedure call and simple data sharing), permitting new abstractions for the "glue" that allows components to be composed.

A NEW DISCIPLINE EMERGES

Initially, architectural design was largely an ad hoc affair. Architectural definitions relied on informal box-and-line diagrams, which were rarely maintained once a system was constructed. Architectural choices were made in an idiosyncratic fashion—typically by adapting some previous design, whether or not it was appropriate. Good architects, even if they were classified as such within their organizations, learned their craft by hard experience in particular domains and were unable to teach others what they knew. It was usually impossible to analyze an architectural description for consistency or to infer nontrivial properties about it. There was virtually no way to check that a given system implementation faithfully represented its architectural design.

However, despite their informality, architectural descriptions were central to system design. As people began to understand the critical role that architectural design plays in determining system success, they also began to recognize the need for a more disciplined approach. Early authors began to observe certain unifying principles in architectural design (24), to call out architecture as a field in need of attention (4), and to establish a working vocabulary for software architects (3). Tool vendors began thinking about explicit support for architectural design. Language designers began to consider notations for architectural representation (25).

Within industry, two trends highlighted the importance of architecture. The first was the recognition of a shared repertoire of methods, techniques, patterns, and idioms for structuring complex software systems. For example, the box-and-line-diagrams and explanatory prose that typically accompany a high-level system description often refer to such organizations as a "pipeline," a "blackboard-oriented design," or a "client-server system." Although these terms were rarely assigned precise definitions, they permitted designers to describe complex systems using abstractions that make the overall system intelligible. Moreover, they provided significant semantic content about the kinds of properties of concern, the expected paths of evolution, the overall computational paradigm, and the relationship between this system and other similar systems.

The second trend was the concern with exploiting commonalities in specific domains to provide reusable frameworks for product families. Such exploitation is based on the idea that common aspects of a collection of related systems can be extracted so that each new system can be built at relatively low cost by "instantiating" the shared design. Familiar examples include the standard decomposition of a compiler (which permits undergraduates to construct a new compiler in a semester), standardized communication protocols (which allow vendors to interoperate by providing services at different layers of abstraction), fourth-generation languages (which exploit the common patterns of business information processing), and user interface toolkits and frameworks (which provide both a reusable framework for developing interfaces and sets of reusable components, such as menus and dialog boxes).

Much has changed in the past two decades. Although there is wide variation in the state of the practice, broadly speaking, architecture is much more visible as an important and explicit design activity in software development. Job titles now routinely reflect the role of software architect; companies rely on architectural design reviews as critical staging points; and architects recognize the importance of making explicit tradeoffs within the architectural design space.

In addition, the technological basis for architectural design has improved dramatically. Three of the important advancements have been the development of architecture description languages and tools, the emergence of product line engineering and architectural standards, and the codification and dissemination of architectural design expertise.

ARCHITECTURE DESCRIPTION LANGUAGES AND TOOLS

The informality of most box-and-line depictions of architectural designs leads to a number of problems. The meaning of the design may not be clear. Informal diagrams cannot be formally analyzed for consistency, completeness, or correctness. Architectural constraints assumed in the initial design are not enforced as a system evolves. There are few tools to help architectural designers with their tasks.

To alleviate these problems, there have been a number of important developments. First has been the emergence of practitioner guidelines (2) and published standards for architectural documentation (26,27), which have helped to codify best practices and provide some uniformity to the way architectures are documented.

A second development has been the creation of formal notations for representing and analyzing architectural designs. Sometimes referred to as "Architecture Description Languages" or "Architecture Definition Languages" (ADLs), these notations usually provide both a conceptual framework and a formal language for characterizing software architectures (25,28). They also typically provide tools for parsing, displaying, compiling, analyzing, or simulating architectural descriptions.

Examples of ADLs include AADL (29), Acme (30), Adage (14), C2 (31), Darwin (16), Rapide (10), SADL (32), UniCon (33), Meta-H (34), and Wright (9). Although all of these languages are concerned with architectural design, each provides certain distinctive capabilities: AADL supports the design and analysis of real-time and embedded computer systems; Acme supports checking of conformance to architectural styles; Adage supports the description of architectural frameworks for avionics navigation and guidance; C2 supports the description of user interface systems using an event-based style; Darwin supports the analysis of distributed message-passing systems; Meta-H provides guidance for designers of real-time avionics control software; Rapide allows architectural designs to be simulated and has tools for analyzing the results of those simulations; SADL provides a formal basis for architectural refinement; UniCon has a high-level compiler for architectural designs that supports a mixture of heterogeneous component and connector types; and Wright supports the formal specification and analysis of interactions between architectural components.

Although these languages (and their tools) differ in many respects, a number of key insights have emerged through their development.

The first insight is that good architectural description benefits from multiple views, each view capturing some aspect of the system (2,26,27,35). Two of the more important classes of view are:

- **Code-oriented views**, which describe how the software is organized into modules and what kinds if implementation dependencies exist between those modules. Class diagrams, layered diagrams, and work breakdown structures are examples of this class of view; and
- **Execution-oriented views**, which describe how the system appears at run time, typically providing one or more snapshots of a system in action. These views are useful for documenting and analyzing execution properties such as performance, reliability, and security.

A second insight is that architectural description of execution-oriented views, as embodied in most of the ADLs mentioned earlier, requires the ability to model the following as first-class design entities:

- **Components** represent the computational elements and data stores of a system. Intuitively, they correspond to the boxes in box-and-line descriptions of software architectures. Examples of components include clients, servers, filters, blackboards, and databases. Components may have multiple interfaces, each interface defining a point of interaction between a component and its environment. A component may have several interfaces of the same type (e.g., a server may have several active http connections).

- **Connectors** represent interactions among components. They provide the "glue" for architectural designs, and they correspond to the lines in box-and-line descriptions. From a run-time perspective, connectors mediate the communication and coordination activities among components. Examples include simple forms of interaction, such as pipes, procedure call, and event broadcast. Connectors may also represent complex interactions, such as a client-server protocol or an SQL link between a database and an application. Connectors have interfaces that define the roles played by the participants in the interaction.
- **Systems** represent graphs of components and connectors. In general, systems may be hierarchical: Components and connectors may represent subsystems that have their own internal architectures. We will refer to these as *representations*. When a system or part of a system has a representation, it is also necessary to explain the mapping between the internal and external interfaces.
- **Properties** represent additional information (beyond structure) about the parts of an architectural description. Although the properties that can be expressed by different ADLs vary considerably, typically they are used to represent anticipated or required extra-functional aspects of an architectural design. For example, some ADLs allow one to calculate system throughput and latency based on performance estimates of the constituent components and connectors. In general, it is desirable to be able to associate properties with any architectural element in a description (components, connectors, systems, and their interfaces). For example, a property of an interface might describe an interaction protocol.
- **Styles** represent families of related systems. An architectural style typically defines a vocabulary of design element types as a set of component, connector, port, role, binding, and property types, together with rules for composing instances of the types. We will describe some of the more prominent styles later in this article.

To illustrate the use of these modeling constructs, consider the example shown in Fig. 2. The system defines an execution-oriented view of a simple string-processing application that extracts and sorts text. The system is described in a pipe-filter style, which provides a design vocabulary consisting of a filter component type and pipe connector type, input and output interface (port) types, and a single binding type. In addition, there would likely be constraints (not shown) that ensure, for example, that the reader/writer roles of the pipe are associated with appropriate input/output ports. The system is described hierarchically: *MergeAndSort* is defined by a representation that is itself a pipe-filter system. In complementary documentation, properties of the components and connectors might list, for example, performance characteristics used by a tool to calculate overall system throughput.

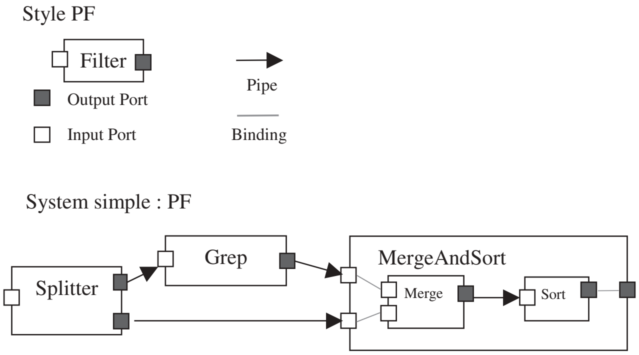


Figure 2. A system in the pipe-filter style.

PRODUCT LINES AND ARCHITECTURAL STANDARDS

As noted earlier, an important trend has been the desire to exploit commonality across multiple products. Two specific manifestations of that trend are improvements in our ability to create product lines within an organization and the emergence of domain-specific architectural standards for cross-vendor integration.

With respect to product lines, a key challenge is that a product line approach requires different methods of development. In a single-product approach, the architecture must be evaluated with respect to the requirements of that product alone. Moreover, single products can be built independently, each with a different architecture.

However, in a product line approach, one must also consider requirements for the *family* of systems and the relationship between those requirements and the ones associated with each particular instance. Figure 3 illustrates this relationship. In particular, there must be an up-front (and ongoing) investment in developing a reusable architecture that can be instantiated for each product. Other reusable assets, such as components, test suites, tools, and so on, typically accompany this approach.

Although product line engineering is not yet widespread, we are beginning to have a better understanding

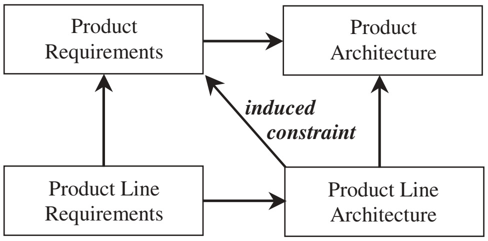


Figure 3. Product line architectures.

of the processes, economics, and artifacts required to achieve the benefits of a product line approach. A number of case studies of product line successes have been published (36,37). Moreover, organizations such as the Carnegie Mellon University's Software Engineering Institute are well on their way toward providing concrete guidelines and processes for the use of a product line approach (38).

Like product line approaches, domain-specific architectural standards for cross-vendor integration provide frameworks that permit system developers to configure a wide variety of specific systems by instantiating those frameworks. But more importantly, such standards support the integration of parts provided by multiple vendors. A number of these have been sanctioned as formal international standards (such as those sponsored by the Institute of Electrical and Electronics Engineers, Incorporated (IEEE) or the International Standards Organization (ISO)), whereas others are ad hoc or de facto standards promoted by one or more industrial leaders.

A good example of a formal standard is the High-Level Architecture (HLA) for Distributed Simulation (5). Initially proposed by the U.S. Defense Modeling and Simulation Office as a standard to permit the integration of simulations produced by many vendors, it now has become an IEEE Standard (IEEE P1516.1/D6). The HLA prescribes interface standards defining services to coordinate the behavior of multiple semi-independent simulations. In addition, the standard prescribes requirements on the simulation components that indicate what capabilities they must have, and what constraints they must observe on the use of shared services.

An example of an ad hoc standard is Sun's Enterprise Java-Beans (EJB) architecture (6). EJB is intended to support distributed, Java-based, enterprise-level applications, such as business information management systems. Among other things, it prescribes an architecture that defines a vendor-neutral interface to information services, including transactions, persistence, and security. It thereby supports component-based implementations of business processing software that can be easily retargeted to different implementations of those underlying services.

CODIFICATION AND DISSEMINATION

One early impediment to the emergence of architectural design as an engineering discipline was the lack of a shared body of knowledge about architectures and techniques for developing good ones. Today, the situation has improved, due in part to the publication of books on architectural design (1,8,24,26,36,39) and courses (40).

A common theme in these books and courses is the use of standard architectural *styles*. An architectural style typically specifies a design vocabulary, constraints on how that vocabulary is used, and semantic assumptions about that vocabulary (2,11). For example, a pipe-filter style might specify vocabulary in which the processing components are data transformers (filters) and the interactions are via order-preserving streams (pipes). Constraints might include the prohibition of cycles. Semantic assumptions might include the fact that pipes preserve order and that filters are invoked non-deterministically.

Other common styles include blackboard architectures, client-server architectures, event-based architectures, and object-based architectures. Each style is appropriate for certain purposes, but not for others. For example, a pipe-and-filter style would likely be appropriate for a signal processing application, but not for an application in which there is a significant requirement for concurrent access to shared data (41). Moreover, each style is typically associated with a set of associated analyses. For example, it makes sense to analyze a pipe-filter system for system latencies, whereas transaction rates would be a more appropriate analysis for a repository-oriented style.

The identification and documentation of such styles (as well as their more domain-specific variants) enables others to adopt previously defined architectural patterns as a starting point. In that respect, the architectural community has paralleled other communities in recognizing the value of established, well-documented patterns, such as those found in Ref. 42.

While recognizing the value of stylistic uniformity, realities of software construction often force one to compose systems from parts that were not architected in a uniform fashion. For example, one might combine a database from one vendor, with middleware from another, and a user interface from a third. In such cases, the parts do not always work well together—in large measure because they make conflicting assumptions about the environments in which they were designed to work (43), which has led to a recognition of the need to identify architectural strategies for bridging mismatches. Although we are far from having well-understood ways of detecting such a mismatch, and of repairing it when it is discovered, a number of techniques have been developed, some of which are illustrated in Fig. 4 (due to Mary Shaw).

RELATED AREAS

There are a number of closely related areas.

Software Development Methods

One of the hallmarks of software engineering progress has been the development of methods and processes for

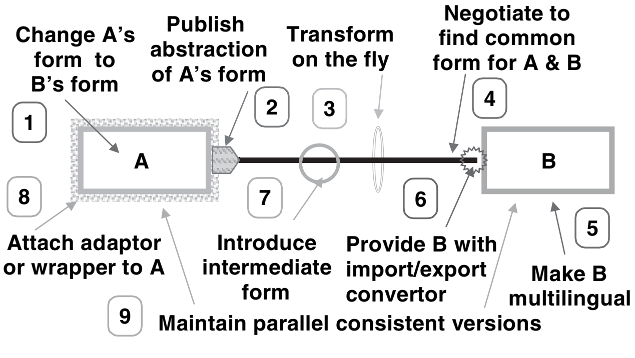

Change A's form to B's form
Publish abstraction of A's form
Transform on the fly
Negotiate to find common form for A & B
1
2
3
4
A
B
8
7
6
5
Attach adaptor or wrapper to A
Introduce intermediate form
Provide B with import/export convertor
Make B multilingual
9
Maintain parallel consistent versions

Figure 4. Some mismatch repair techniques.

software development. Like software architecture, methods attempt to provide a path from requirements to code that eliminates some of the ad hoc development practice of the past.

Methods complement software architecture: The former attempt to provide a set of regular steps for software development, whereas the latter attempts to provide a basis for developing and analyzing certain design models along that path.

To the extent that they support conceptual design of systems, they also address architectural concerns. On the other hand, most methods tend to favor a particular architectural style. For example, object-oriented methods naturally favor architectural designs based on interacting objects, whereas other methods favor other styles.

Object-Oriented Design and Modeling

There are a number of parallels between the evolution of object-oriented design techniques and the trends of software architecture, outlined above.

- **Description Languages and Tools:** Object-oriented systems have long had design languages and tools to support their use. The Unified Modeling Language (UML) has emerged as a standard notation, unifying many of its predecessors (44). Increasingly, vendors are developing tools that take advantage of this technological standardization.
- **Product Lines and Standards:** Object-oriented frameworks have long been an important point of leverage in system development. In particular, component-oriented integration mechanisms, such as CORBA, .NET, and JavaBeans have played an important role in supporting integration of object-oriented parts. In other more domain-specific ways, frameworks like J2EE, VisualBasic, and MFC, have helped improve productivity in specific areas.
- **Codification and Dissemination:** There has been considerable work and interest in object-oriented patterns, which serve to codify common solutions to implementation problems (42).

Given these similarities, it is worth asking the following question: What are the important differences between the two fields? To shed light on the issue, it is helpful to view the relationship between architecture and object-oriented methods from at least three distinct perspectives.

1. *Object-oriented design as an architectural style:* This perspective treats the part of object-oriented development that is concerned with system structure as the special case of architectural design in which the components are objects and the connectors are procedure calls (method invocation). Some ADLs support this view, providing built-in primitives for inter-component procedure call.
2. *Object-oriented design as an implementation base:* This perspective treats object-oriented development as a lower-level activity, more concerned with implementation. Viewed this way, object modeling becomes one viable implementation target for any architectural design.
3. *Object-oriented design as an architectural modeling notation:* This perspective treats a notation such as UML as a suitable notation for all architectural descriptions (8,35,45). Proponents of this perspective have advocated various ways of using object modeling, including class diagrams, collaboration diagrams, and package diagrams (36,46,47). From this perspective, architecture is viewed as a sub-activity of object-oriented design.

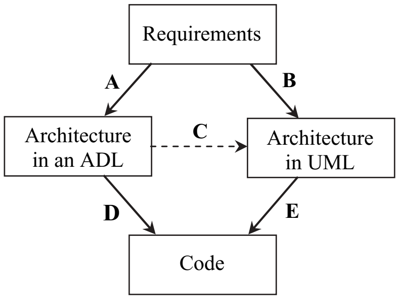


Figure 5. ADLs versus object modeling.

Elaborating on the relationship between ADLs and object-oriented modeling notations, such as UML, Fig. 5 shows some of the paths that might be followed. Path A-D is one in which an ADL is used as the modeling language. Path B-E is one in which UML is used as the modeling notation. Path A-C-E is one in which an architecture is first represented in an ADL, but then transformed into UML before producing an implementation.

Using a general-purpose modeling language such as UML has the advantages of providing a notation that practitioners are more likely to be familiar with and providing a more direct link to object-oriented implementations and development tools. But general-purpose object languages suffer from the problem that the object conceptual vocabulary may not be ideally suited for representing architectural concepts, and there are likely to be fewer opportunities for automated analysis of architectural properties.

Component-Based Systems

Component-based systems are closely related to object-oriented systems insofar as both are based on the construction of systems from encapsulated entities that provide well-defined interfaces to a set of services. However, most component-based systems have a strong intrinsic architectural flavor in that they are usually coupled with an integration framework that prescribes what kinds of interfaces the components must have and ways in which components can interact at run time (7).

From an architectural perspective, component-based systems such as .NET, CORBA, and J2EE define architectural styles that are predominantly object-oriented. In addition, they may support other forms of interaction such as event publish-subscribe. However, component integration standards typically go beyond architectural modeling by providing run-time infrastructure and (in many cases) considerable support for generating code from more abstract descriptions.

FUTURE PROSPECTS

The field of software architecture is one that has experienced considerable growth since the 1990s, and it promises to continue that growth for the foreseeable future. As architectural design matures into an engineering discipline that is universally recognized and practiced, there are a number of significant challenges that will need to be addressed. Many of the solutions to these challenges are likely to occur as a natural consequence of dissemination and maturation of the architectural practices and technology that we know about today. Other challenges develop because of the shifting landscape of computing and the needs for software: These challenges will require significant new innovations. This article has attempted to provide a high-level overview of the terrain, illustrating where software architecture has come over the past few years and outlining relationships between software architecture and other aspects of software engineering.

ACKNOWLEDGMENTS

The author would like to acknowledge a number of colleagues and students who have helped clarify his ideas on software architecture, including Barry Boehm, Dewayne Perry, John Salasin, Mary Shaw, Dave Wile, Alex Wolf, and past and present members of the ABLE research group at Carnegie Mellon University.

BIBLIOGRAPHY

1. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed., Reading, MA: Addison-Wesley, 2003.
2. P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architecture: Views and Beyond*, Reading, MA: Addison-Wesley, 2002.
3. D. Garlan and M. Shaw, An Introduction to software architecture, in *Advances in Software Engineering and Knowledge Engineering*, Singapore: World Scientific Publishing Company, 1993, pp. 1–39.
4. D. E. Perry and A. L. Wolf, Foundations for the study of software architecture, *ACM SIGSOFT Software Eng. Notes*, **17** (4): 40–52, 1992.
5. F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Englewood Cliffs, NJ: Prentice Hall, 2000.
6. V. Matena and M. Hapner, Enterprise JavaBeans, Palo Alto, CA: Sun Microsystems, Inc., 1998.
7. C. Szyperski, *Component Software*: *Beyond Object-Oriented Programming*, Reading, MA: Addison-Wesley, 1998.
8. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern Oriented Software Architecture: A System of Patterns*, New York: Wiley, 1996.
9. R. Allen and D. Garlan, A formal basis for architectural connection, *ACM Trans. Software Engin. Methodol.*, July 1997.
10. D. C. Luckham, L. M. Augustin, J. J. Kenny, J. Veera, D. Bryan, and W. Mann, Specification and analysis of system architecture using Rapide, *IEEE Trans. Software Eng.*, **21** (4): 336–355, April 1995.
11. G. Abowd, R. Allen, and D. Garlan, Using style to understand descriptions of software architecture, in *Proc. SIGSOFT'93: Foundations of Software Engineering*, ACM Press, December 1993.
12. P. Clements, L. Bass, R. Kazman, and G. Abowd, Predicting software quality by architecture-level evaluation, in *Proc. Fifth International Conference on Software Quality*, Austin, TX, 1995.
13. J. A. Stafford, D. J. Richardson, and A. L. Wolf, Aladdin: A Tool for Architecture-Level Dependence Analysis of Software, University of Colorado at Boulder, Boulder, C.O., Technical Report CU-CS-858-98, April 1998.
14. L. Coglianese and R. Szymanski, DSSA-ADAGE: An environment for architecture-based avionics development, in *Proc. AGARD'93*, 1993.
15. D. Garlan, R. Allen, and J. Ockerbloom, Exploiting style in architectural design environments, *Proc. SIGSOFT'94: The 2nd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Press, December 1994, pp. 170–185.
16. J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, Specifying distributed software architectures, in *Proc. Fifth European Software Engineering Conference, ESEC'95*, September 1995.
17. B. Boehm, P. Bose, E. Horowitz, and M. J. Lee, Software requirements negotiation and renegotiation aids: A theory-W based spiral approach, *Proc. 17th International Conference on Software Engineering*, 1994.
18. E. W. Dijkstra, The structure of the "THE" – multiprogramming system, *Comm. ACM*, **11** (5): 341–346, 1968.
19. D. Parnas, On the criteria to be used in decomposing systems into modules, *Comm. ACM*, **15** (12): 1053–1058, 1972.
20. F. DeRemer and H. H. Kron, Programming-in-the-Large versus Programming-in-the-Small, *IEEE Trans. Software Eng.*, **SE-2** (2): 80–86, June 1976.
21. D. L. Parnas, Designing software for ease of extension and contraction, *IEEE Trans. Software Eng.*, **5**: 128–138, 1979.
22. L. W. Cooprider, *The representation of software families*, PhD Thesis, Technical Report CMU-CS-79-116. Carnegie Mellon University, Pittsburgh, PA: 1979.
23. D. E. Perry, Software interconnection models, *Proc. 9th International Conference on Software Engineering*, IEEE Computer Society Press, 1987.
24. E. Rechtin, *Systems Architecting: Creating and Building Complex Systems*, Englewood Cliffs, NJ: Prentice Hall, 1991.
25. N. Medvidovic and R. N. Taylor, A Classification and comparison framework for software architecture description languages, *IEEE Trans. Software Eng.*, **26** (1): 70–93, 2000.
26. International Organization for Standardization, *ISO/IEC 10746 1–4 Open Distributed Processing–Reference Model (Parts 1–4)*, July 1995. ITU Recommendation X. 901–904.
27. *IEEE Std. 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems*, Piscataway, NJ: IEEE Standards, October 2000.

28. D. Garlan and D. Perry, Introduction to the special issue on software architecture, *IEEE Trans. Software Eng.*, **21** (4), 1995.

29. Society of Automotive Engineering, SAE AADL Information Site, Available: http://www.aadl.info.

30. D. Garlan, R. T. Monroe, and D. Wile, G. T. Leavens and M. Sitaraman (eds.), Acme: Architectural description of component-based systems, *Foundations of Component-Based Systems*, Cambridge UK: Cambridge University Press, 2000, pp. 47–68.

31. N. Medvidovic, P. Oreizy, J. E. Robbins, and R. N. Taylor, Using object-oriented typing to support architectural design in the C2 style, *Proc. 4th ACM Symposium on the Foundations of Software Engineering, SIGSOFT'96*, New York: ACM Press, 1996.

32. M. Moriconi, X. Qian, and R. Riemenschneider, Correct architecture refinement, *IEEE Trans. Software Eng., Special Issue on Software Architecture*, **21** (4): 356–372, 1995.

33. M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnick, Abstractions for software architecture and tools to support them, *IEEE Trans. Software Eng.*, **21** (4): 314–335, 1995.

34. P. Binns and S. Vestal, Formal real-time architecture specification and analysis, *10th IEEE Workshop on Real-Time Operating Systems and Software*, May 1993.

35. P. B. Kruchten, The 4+1 view model of architecture, *IEEE Software*, November, 1995, pp. 42–50.

36. C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*, Reading, MA: Addison-Wesley, 2000.

37. P. Donohoe, (ed.), *Software Architecture: TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, Boston, MA: Kluwer Academic Publishers, 1999.

38. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Boston, MA: Addison-Wesley Longman, 2001.

39. M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Englewood Cliffs, NJ: Prentice Hall, 1996.

40. D. Garlan, M. Shaw, C. Okasaki, C. Scott, and R. Swonger, Experience with a course on architectures for software systems, *Proceedings of the Sixth SEI Conference on Software Engineering Education*, New York: Springer Verlag, LNCS 376, October 1992.

41. M. Shaw and P. Clements, A field guide to boxology: Preliminary classification of architectural styles for software systems, *Proc. of COMPSAC 1997*, August 1997.

42. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Design*, Reading, MA: Addison-Wesley, 1995.

43. D. Garlan, R. Allen, and J. Ockerbloom, Architectural mismatch: Why reuse is so hard, *IEEE Software*, **12** (6): 17–28, 1995.

44. J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, Reading, MA: Addison-Wesley, 1999.

45. S. Mellor, K. Scott, D. Weise, and A. Uhl, *Mda Distilled: Principles of Model-Driven Architecture*, Reading, MA: Addison-Wesley, 2004.

46. D. Garlan and A. J. Kompanek, Reconciling the needs of architectural description with object-modeling notations, *Proc. Third International Conference on the Unified Modeling Language*, 2000.

47. N. Medvidovic and D. S. Rosenblum, Assessing the suitability of a standard design method for modeling software architectures, *Proc. First Working IFIP Conference on Software Architecture (WICSA1)*, San Antonio, TX, 1999.

DAVID GARLAN
Carnegie Mellon University
Pittsburgh, Pennsylvania

SOFTWARE COMPONENT REPOSITORIES

INTRODUCTION

In the past, reuse has primarily been the result of opportunistic success, where one program was able to take advantage of the efforts of another. A paradigm shift is needed from current software engineering and development practices to a software engineering process in which software reuse is institutionalized and becomes an inseparable part of the software development process. Reuse should be systematic, driven by a demand for software components identified as a result of domain analysis and architecture development. Reuse needs to be treated as an integral part of engineering and acquisition activities. It is essential that an organizational infrastructure be implemented to manage domains, define products and standards, establish ownership criteria, allocate investment resources, and direct the establishment and population of reuse repositories. An effective infrastructure will guide reuse activities to avoid duplication of effort, impose necessary standardization, and ensure repository population is user demand-driven.

WHAT IS SOFTWARE COMPONENT REPOSITORY?

A component repository system that supports software reuse (3) by helping programmers locate, comprehend, and modify components has three parts: a repository that contains components, an indexing and retrieval mechanism, and an interface for user interaction.

Usually, component repository capabilities include the following:

- Automated repository system with a graphical user interface (GUI) for browsing, searching and retrieval;
- Standard component description framework (e.g., to include purpose, functional description, certification level, key environmental constraints, historical results of usage, and legal restrictions);
- Effective classification scheme for each domain; and
- Thorough system and component documentation.

REPOSITORY MECHANISM

Each repository system should provide as much automated support to users as possible on identification, comparison, evaluation, and retrieval of similar reusable components. Support for adapting, transforming, and specializing components is desirable. It must also provide a range of support to users in locating and comparing the relative reusability of individual repository components. Furthermore, the system must be readily available to system developers if it is to be used, and it must support access from a variety of platforms. As the repository acquires a significant number of reusable software components (RSCs), an automated search and retrieval system becomes indispensable (4–6). Whatever tool is used, the repository must have a way to classify RSCs so that a user can quickly find what is wanted without frustration and delay. Sophisticated, expert system, knowledge-based approaches and new technologies for high-speed text search are the subjects of current research efforts.

Standard component description frameworks help ease the process of comprehension and comparison of similar components, and they include data such as relative numeric measures for reusability, reliability, maintainability and portability (7). Inclusion of testing and component documentation provides additional information to help the potential user gauge the effort required to tailor the component for reuse.

Effective classification schemes are essential to assist the user in locating and comparing repository components and to speed the process of identifying appropriate components for the task at hand (8,9). Finally, system and component documentation complete the cycle of evaluation and enable the reuser to determine which components have reuse potential with regard to specific requirements and to fully comprehend the process of obtaining components for reuse in a new application.

In addition, other equally important requirements have been identified that require resolution to support cohesive, wide reuse, including [1] integration of repository capabilities and procedures within the system development and acquisition process; [2] identification and support of specific requirements associated with the security and integrity of reusable components implementing trusted computing base (TCB) or other security capabilities; and [3] intercommunication and interoperabllity among diverse repository systems. Experience has shown that these requirements can only be resolved through the combination of developing new technologies, standard procedures, and evolution or revision of existing policies.

REPOSITORY RETRIEVAL

Component retrieval is a fundamental issue in software reuse. The retrieval process involves finding a component matching the desired functionality and making sure that the component satisfies required nonfunctional properties such as timing and resource constraints.

Precision and recall are two measures that have traditionally been used to evaluate methods for retrieving software components. Let Q be the set of items that should be returned in answer to the query and let R be the choice set actually returned. Then precision can be defined as $|R \cap Q|/|Q|$, which measures the ability to return only the relevant components. Recall can be defined as $|R \cap Q|/|R|$, which measures the ability not to miss relevant components. Desirable retrieval techniques should yield high precision and recall.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Classic Retrieval Approaches

The most classic approach to retrieval is to classify items by keywords and then search for items that have certain given keywords (10). Experience shows that this approach works poorly for retrieving software components from even moderately large repositories. One problem is that the user must be familiar with both the classification scheme and the particular repository. Also, it is very difficult to get both high precision and high recall. This situation suggests that, for ranked filtering, it would be most appropriate to use a small number of keywords.

Another classic approach is browsing. Browsing systems depend on links among the items to be searched and on the user following those links to find the desired item. Experience shows that browsing through large structures can be very frustrating and time-consuming. The problem is that the structure of the links often does not match the needs of most users and that different users may need different structures.

The Facet Approach

Prieto-Diaz (11) proposed using *facets*, which are groups of related terms in a subject area. For example, a facet to describe the functions performed by components might use terms chosen from *find, compare, sort, update, send, receive*, and so on. This approach provided a better description of UNIX components than a pure keyword approach because of its standardized structure. However, it still relies on an informal description of components, using a limited set of facets and terms. Facets also suffer from the same problem as the keyword approach.

AI Approaches

Artificial intelligence (AI) based approaches use a knowledge base and statistical information to retrieve reusable components, based on a keyword search from texts describing the components (12–14). However, because the characterization of the component behavior is completely informal, the behavior is unpredictable.

The Ontology-based Approach

Yen (15) used an ontology-based approach to facilitate browsing and effective search in a repository for embedded software. They developed a merging and echoing technique, which converts the ontology into a hierarchy suitable for browsing, but without the loss of critical semantics of the ontology. A search result categorization approach was developed to eliminate the problem of obtaining a large number of search results without reducing the recall factor. As the ontology approach can typically have many views, it may not be suitable for navigation and browsing.

Specification-based Approaches

Specification-based approaches use semantics for software component retrieval (16,17). The primary aim was to check that retrieved components yield the behavior specified in the user's query, therefore increasing the precision of retrieval. Zaremski and Wing (18) focused on specification matching, using the Larch/ML interface language to express pre- and post-conditions in first-order logic and the Larch prover to verify that candidate components satisfy these conditions. Various senses of matching are defined, but neither ranking nor partial semantic matching is considered.

In general, using formal specifications as search keys has two main problems. The first problem is practical: Not all users are sophisticated enough to write formal specifications, much less correct ones. The second problem is that semantic matching is very time-consuming, because some form of theorem proving must be done. As theorem proving requires unbounded time, practical implementations must impose time limits, which reduce recall.

Automated Retrieval

Luqi et al. (19) proposed an automated retrieval approach. In this approach, search is organized as a series of increasingly stringent filters on candidate components. They first filter components by comparing their signatures with that of the query, which is accomplished by signature matching, which looks for maps that translate the type and function symbols of the query into corresponding type and function symbols of candidate components. A first stage of signature filtering can compare pre-computed syntactic profiles of components with the profile of the query. These profiles are special data structures that support an efficient approximation of signature matching. Signature matches can be partial, in that only part of the functionality the user seeks may actually be available. Profile matching should be followed by full signature matching. To achieve high recall, filters in the early stages must eliminate only those components that are definitely not compatible with the query.

Finally, semantic filters rank components by how well they satisfy the equations in the query. In this process, equations that are logical consequences of the query specification are translated through the signature matches into equations whose proof is attempted in the candidate specifications. For greatest efficiency, it is desirable to restrict queries to be ground equations; these ground equations correspond to test cases and enable semantic matching to be efficiently decidable. The candidates in the choice set are ranked according to their likelihood of success. If the closest match is partial, the user will need to modify the closest matching component. This whole process can be made iterative.

Figure 1 shows the multi-level filtering architecture of the automated retrieval approach; the top line is to indicate user modification of the query in light of the final filtering results.

In comparing with other approaches, the automated retrieval approach has the following merits:

1. It can simultaneously achieve high precision and high recall (20).
2. It compares formal specifications of components using ground equation test cases as queries.
3. Users do not need to deal with formal specification notation, but instead can express queries in a

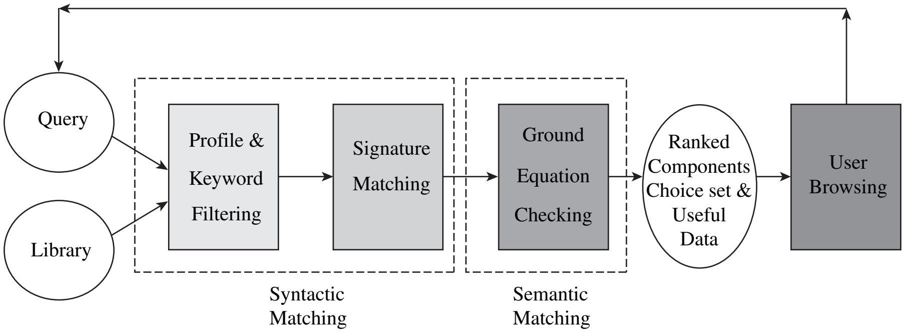


Figure 1. An organization model for software component search.

standard programming notation, which is automatically translated into algebraic notation.

4. It seeks to achieve both efficiency and effectiveness by imposing a series of increasingly stringent filters that use both syntactic and *partial* semantic information about components.
5. A rank is provided on components in the choice set, measuring how well they fit the user's query and enabling sorting by relevance.
6. Generic modules are allowed in the software base.
7. It addresses structuring the software base to facilitate search.
8. Users can give *selection criteria* to control the search and display of retrieved components.
9. Besides returning the ranked components, it also reports information to help the user reformulate the query in case no suitable component was found.

SOME REUSABLE SOFTWARE COMPONENT REPOSITORIES

Microsoft Repository

Microsoft Repository is composed of two major components: a set of ActiveX interfaces that a developer can use to define open information models and a repository engine that is the underlying storage mechanism for these information models (21). The repository engine sits on top of an SQL Server and Microsoft JET database system.

A tool developer can use Microsoft Repository to share and reuse components. To share components effectively, it is useful to share not only the executable image of a component but also descriptive information about the component and its configuration.

Microsoft Repository uses an SQL Server database to store object and relationship data. The repository engine is scalable from desktop-based to server-based solutions. The repository exposes information models by way of ActiveX objects and uses an SQL Server as a storage and query provider.

+1Reuse Repository

The +1Reuse system supports reuse repositories created and maintained by the user, project-wide "filtered" repositories under strict quality controls, and selective reuse. Selective reuse enables reuse of any submodel from an existing or re-engineered +1Environment projrect. In a sense, every +1Environment project is a reuse repository. Selective reuse significantly improves a user's ability to reuse all source code and documentation from all previous projects and at any granularity, which, to the best of our knowledge, is currently the only system to support this feature.

The +1Reuse system supports reuse of design, documentation, source code, header files, test cases, test shell scripts, expected test results, and modeling information.

The +1Reuse system was developed by +1 Software Engineering Co. in California (22). It is now running on Sun Workstation platforms under Solaris. The GUI is based on OpenWindows, Motif, and CDE.

ComponentSource Repository

ComponentSource provides a web-based repository for "off-the-shelf" reusable software components (23). It uses a taxonomy to structure components to facilitate retrieval. This taxonomy provides an effective way of locating generic components (domain-independent) that are well known to programmers and corresponds well with their intuition. ComponentSource has spent many years accumulating the world's largest online repository of quality software components and development tools. Every product that is listed on the site has first gone through a commercial and technical assessment process, to review such things as software and documentation quality, sample code, likely support issues, and the ongoing viability of the supplier.

Agora

The Component-Based Systems (CBS) Initiative at the Software Engineering Institute (SEI) developed the Agora software prototype to investigate the integration of search technology with component introspection to create a distributed, worldwide component repository.

Agora is a prototype component repository being developed by the SEI at Carnegie Mellon University (24,25). The objective of this work is to create an automatically generated, indexed, database of software components classified by component models (e.g., JavaBean, ActiveX, CORBA, Enterprise JavaBean). Agora combines introspection with Web search engines to reduce the costs of bringing software components to, and finding components in, the software marketplace.

The Agora search engine enhances existing but rudimentary search capabilities for Java applets. By using Java introspection, the Agora search engine can maintain a more structured and descriptive index that is targeted to the type of content (the component model) and the intended audience (application developers) than is supported by existing search engines. For example, information about component properties, events, and methods can be retrieved from Agora.

CAPS Software Reusable Component Repository

CAPS (Computer-Aided Prototyping System) is a research project developed by the Software Engineering Group at the Naval Postgraduate School (26). Initial implementation of CAPS software base was first explored in 1988 (27). An implementation of the software base was accomplished in 1991 by using ONTOS, an object-oriented database management system that provides an interface to C++ for customization and flexibility (28). The CAPS software base has been changing to a software component repository since 1998 (19). The CAPS component repository supports two critical functions: component storage and component retrieval. Much effort has been made to improve the component retrieval method (20,29). To the best of our knowledge, CAPS Repository is the only one that supports profile matching and signature matching. It simultaneously provides high precision and recall retrieval. The CAPS repository is still under construction. A prototype has been developed to verify the performance of the retrieval methods.

CONCLUSION

Web-based reuse is the current trend of software component repositories. Usually, the aim is to provide a service within a domain, organization, or area. This kind of repository is used in a wide scope. Another trend of component repositories is to be a part of an integrated CASE environment. The aim is to provide an integrated CASE environment for a software development organization. This kind of repository is generally used in a relatively narrow scope.

As suggested by Weisert (30), to guarantee successful software reuse, the software development organization must support a component repository, to which programmers contribute new components as byproducts of their projects, and from which other programmers will draw existing components for use in their projects.

BIBLIOGRAPHY

1. B. Fischer, M. Kievernagel, and W. Struckmann, VCR: A VDM-based software component retrieval tool, *Proc. ICSE-17 Workshop on Formal Methods Application in Software Engineering Practice*, Seattle, WA, 1995.
2. A. Mili, R. Mili, and R. Mittermeir, Storing and retrieving software components: A refinement based system, *Proc. 16th Int'l Conf. on Software Engineering*, Sorrento, Italy, 1994, pp. 91–100.
3. G. Fischer, S. Henninger, and D. Redmiles, Cognitive tools for locating and comprehending software objects for reuse, *Proc. 13th International Conference on Software Engineering*, Austin, TX, 1991, 318–328.
4. J. Penix, P. Baraona, and P. Alexander, Classification and retrieval of reusable components using semantic features, *Proc. 10th Knowledge-Based Software Engineering Conference*, Boston, MA, pp. 131–138, 1995.
5. A. M. Zaremski, Signature and Specification Matching, PhD thesis, Carnegie Mellon University, Pittsburgh, PA 1996.
6. A. M. Zaremski and J. M Wing, Specification matching of software components, *3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, New York, 1995.
7. J. Penix and P. Alexander, Design representation for automating software component reuse, *Proc. First International Workshop on Knowledge-Based Systems for the (re)Use of Program Libraries*, Sophia Antipolis, France, 1995.
8. R. McDowell and J. Solderitsch, The Reusability Library Framework, *Proc. 3rd Unisys Defense Systems Software Engineering Symposium*, 1990.
9. R. McDowell and K. Cassell, The RLF librarian: A reusability librarian based on cooperating knowledge-based systems, *Proc. 4th Annual Rome Air Development Center Knowledge-Based Software Assistant Conference*, Utica, N.Y., 1989.
10. Y. Matsumoto, A software factory: An overall approach to software production, in P. Freeman, (ed.), *Tutorial on Software Reusability*, 1987, pp. 155–178.
11. R. Prieto-Diaz, Implementing faceted classification for software reuse, *Comm. ACM*, 34(5):89–97, 1991.
12. E. Ostertag, J. Hendler, R. Prieto-Diaz, and C. Braun, Computing similarity in a reuse library system, *ACM Tran. Softw. Eng. Methodol.*, 1(3):205–228, 1992.
13. S. Henninger, Using iterative refinement to find reusable software. *IEEE Software*, 11(5):48–59, 1994.
14. G. Fischer, S. Henninger, and D. Redmiles, Cognitive tools for locating and comprehending software objects for rescue, *Proc. 13th International Conference on Software Engineering*, Austin, TX, 1991.
15. I-L. Yen, J. Goluguri, F. Bastani, L. Khan, and J. Linn, A component-based approach for embedded software development, *5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Washington, D.C., 2002.
16. Luqi, Normalized specifications for identifying reusable software, *Proc. 1987 Fall Joint Computer Conference, IEEE*, Dallas, TX, 1987, pp. 46–49.
17. A. Mili, R. Mili, and R. Mittermeir, Storing and retrieving software components, *Proc. 16th International Conference on Software Engineering*, Sorrento, Italy, 15–19, 1994.
18. A. Moormann Zaremski, and J. M. Wing, Specification matching of software components, *ACM Trans. Softw. Eng. Methodol*, **6**(4): 333–369, 1997.
19. Luqi and J. Guo, Toward automated retrieval for a software component repository, *Proc. IEEE International Conference and Workshop on the Engineering of Computer Based Systems (IEEE ECBS)*, Nashville, T.N., 1999, pp. 99–105.
20. D. Nguyen, An Architectural Model for Software Component Search, PhD Dissertation, Naval Postgraduate School, Monterey, C.A., 1995.
21. Microsoft Repository. http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dnarrepos/html/msdnr/eposwp.asp.

22. +1 Software Engineeering Corporate Mission, http://www.plus-one.com/company.html.

23. Component Source. Available: http://www.componentsource.-com/CS/Default.asp.

24. R. Seacord, S. Hissam, and K. Wallnau, Agora: A search engine for software components, *IEEE Internet Computing*, **2**(6):62–70, 1998.

25. R. Seacord, S. Hissam, and C. Wallnau, Agora: A search engine for software components. Technical report, *CMU/SEI-98-TR-011*, 1998.

26. Luqi and M. Ketabchi, A Computer-aided prototyping system, *IEEE Trans. Software Eng*, 5(2):66–72, 1988.

27. R. Steigerwald, Luqi, and J. McDowell, CASE Tool for reusable software component storage and retrieval in rapid prototyping, *Inf. Softw Techn*, 38(9):698–705, 1991.

28. S. Dolgoff, Automated Interface for Retrieving Reusable Software Components, Master's Thesis, Naval Postgraduate School, Monterey, C.A., 1992.

29. J. Herman, Improving Syntactic Matching for Multi-Level Filtering, M.S. Thesis, Naval Postgraduate School, Monterey, C.A., 1997.

30. C. Weisert, Reusable Components—Decades of Misconceptions, Guidelines for Success. Available: http://www.idinews.-com/reUse.html. 29 February 2004.

LUQI
Naval Postgraduate School
Monterey, California
LIN ZHANG
Beijing University of
Aeronautics and Astronautics
Beijing, China

SOFTWARE CYBERNETICS

INTRODUCTION

Separately, the concepts of software and cybernetics are well known and found as in the definitions extracted from an online dictionary.

Software (1). The programs and procedures required to enable a computer to perform a specific task, as opposed to the physical components of the system (hardware).

Cybernetics (1). The study of communication and control, typically involving regulatory feedback, in living organisms, in machines, and in organizations and their combinations, for example, in sociotechnical systems, computer-controlled machines such as automata, and robots.

As we can see from the definition above, cybernetics includes software that implements a control system but does not include the control of software itself, much less the control of the software development process. For example, the design of the software for a cruise control system of an automobile is considered part of cybernetics, but the design of a software/technique to regulate the behavior of another software system is not addressed in the scope of cybernetics. A new area is therefore needed to develop control systems with this purpose. It should be noted that the definition of control systems (as seen below) used in this new area remains unchanged.

Control Systems (1). A control system is a device or set of devices that manage the behavior of other devices. Some devices or systems are not controllable. A control system is an interconnection of components connected or related in such a manner as to command, to direct, or to regulate itself or another system. A control loop is a collection of instruments and control algorithms arranged in such a fashion as to regulate a variable at a setpoint or a reference value. The loop part of the name refers to the fact that most control loops make use of feedback in a continuous loop. These loops are referred to as closed-loop control. An open-loop controller does not directly make use of feedback. The most common control loop uses a feedback or PID controller.

Control theory has been successfully applied to solve problems in areas such as biology, management, and social sciences. The successful application of the same concepts to control/regulate the behavior of software systems and/or of the software development process (2) in all its aspects is what we now refer to as software cybernetics. Software cybernetics also includes principles and theories in software engineering that can be applied to control engineering. In this article we provide a definition of software cybernetics and delineate its scope. Research in software cybernetics has been applied to many distinct areas such as software development, adaptive software, network security, and fault tolerance; a brief description of these applications appears in this article.

Software cybernetics has been consistently expanding since its inception and the community is organizing itself mainly through the International Workshop on Software Cybernetics (IWSC). However, skepticism continues to exist as some reject the feasibility/use of regulating software systems or its development process by mathematical laws. Also, a belief exists that control is purely a continuous approach and overlooks the fact that discrete counterparts do exist for all continuous techniques. Moreover, even when a continuous approach is used to model the behavior of the object under consideration, a control technique can be applied at discrete time intervals. The difficulties in applying feedback control to software processes, for example, have been delineated by Lehman et al. (3). One of the difficulties they point out is the immaturity of software processes. They state, "...we need research to establish appropriate theories from which to derive necessary control mechanisms and experimentations to establish their settings and effects." Although this has not yet been fully accomplished, results from research on software cybernetics have moved a few steps toward this goal. Another aspect contributing to the slow development and adoption of control-theoretic concepts for software is the paucity of control-system researchers involved in software engineering. Again, software cybernetics offers an environment where collaboration among both areas can flourish.

DEFINITION AND SCOPE

According to Norbert Wiener (4), cybernetics refers to control and communication in man and humans in machines. Accordingly, one may define software cybernetics as control and communication in the software, its processes, and systems. However, this definition is not satisfactory for two reasons. First, it implies that software cybernetics should cover various ad hoc control activities in software engineering and various communication mechanisms in concurrent computing. It would be difficult to distinguish software cybernetics from the existing discipline of software engineering and the theories of concurrent computing. Second, the above definition rules out the possibility that the principles and theories of software engineering can be applied to control theories and engineering.

It is important to note that control is a well-established discipline, of which feedback and optimization are two central themes. In addition, a solid theoretical foundation has yet to be established for the existing discipline of software engineering. Therefore, we define software cybernetics as an emerging discipline that explores the *theoretically justified* interplay between software and control. More importantly, according to Cai et al. (5), software cybernetics addresses issues and questions that relate to (*1*) the formalization and quantification of feedback

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

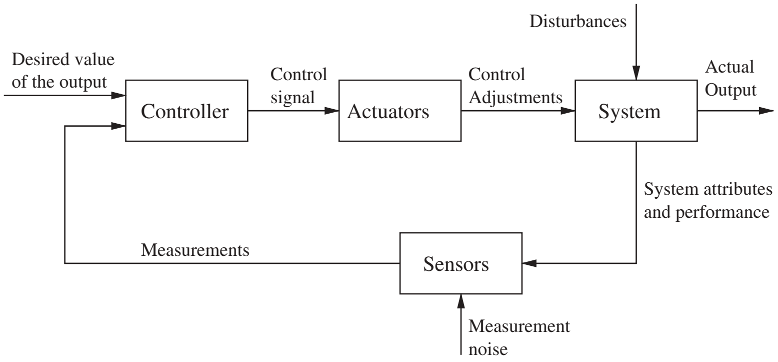


Figure 1. Typical feedback loop (5).

mechanisms in software processes and systems, (*2*) the adaptation of control theoretic principles to software processes and systems, (*3*) the application of the principles of software theories and engineering to control systems and processes, and (*4*) the integration of the theories of software engineering and control engineering. In response to these issues and questions, research subareas of software cybernetics can be divided into four classes: fundamental principles, cybernetic software engineering, cybernetic autonomic computing, and software-enabled control.

Figure 1 presents the structure of a typical closed feedback control loop (6). Although it is oriented towards physical systems, the same structure can be used to control software systems and software processes. The block labeled "System" in Fig. 1 can be mapped to a software system or software process as the object to be controlled, and the actuators could be exemplified as the process manager executing suggested changes (in case the object to be controlled is a software process), or the operating system allocating additional memory when the object under control is a software application. Clearly, the application of control requires quantification/qualification of the output variables to be controlled and the input values used to control them. This means that any measurable quantities/qualities with respect to software products and processes can be potentially controllable. For example, consider the control system proposed for the software system test phase by Cangussu et al. (7); the controller uses the model parameters and two inputs corresponding to the test manager's vectors to induce changes in the process. The control technique explicity requires that the user be able to quantify the values of the control inputs, and it implicitly requires—through the use of the model parameters—the data from which the model parameters were calibrated. This example illustrates that the data requirements of a particular application of software cybernetics are primarily dictated by the underlying model, with a small additional requirement in the specification of the expected model inputs.

It should be clear that software cybernetics is not the only overlap of software systems/software engineering with control theory. The implementation of a controller for a boiler system represents this overlap (software being used to implement a control mechanism). However, software cybernetics is more comprehensive and can be seen more as the mapping of software systems/software engineering concepts to concepts in control systems. For example, the use of control theory to regulate the amount of memory reserved for a cache system represents this mapping.

Fundamental Principles

The research area of fundamental principles (FP) is concerned with the fundamental questions and theoretical foundation of software cybernetics. Such questions are as follows: Can the software behavior be controlled? What role can feedback play in software processes and systems? How can software behavior be modeled in the framework of software cybernetics? Three specific topics should be addressed in this research area: modeling formalism, controllability and bisimulation, and feedback complexity and bisimulation.

Modeling Formalism. To address the question of whether software behavior can be controlled in a theoretically justified manner, it is important to examine how software behavior can be modeled. Three modeling formalisms have been proposed: formal models, dynamical system models, and controlled Markov chains. A typical formal model is the extended finite state machine (EFSM), which has been widely adopted to describe communication software behavior (8). It is interesting to note that an EFSM can be reformulated as a closed-loop control system comprising a controlled object and a controller (9). This implies that most software systems can be modeled as a control system.

Linear dynamic system models have been proposed to describe software testing process (7) and software service behavior (10). In the continuous-time domain, the model is in the form of Equation (1).

$$\begin{cases} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{cases} \tag{1}$$

where $x(t)$, $u(t)$, and $y(t)$ are state vector, control input, and output, respectively, and A, B, C, and D are matrices of appropriate dimension. The discrete-time counterpart of Equation (1) is given by Equation (2)

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases} \tag{2}$$

where k denotes the kth sampling instant.

Finally, controlled Markov chains have been proposed to describe the software testing process (11). The states in a controlled Markov chain are defined by software variables of interest such as the number of defects remaining in the software under test. As distinct testing actions are applied to the software under test, the software state transitions take place in accordance with a Markov law whose probability distributions depend on the applied testing actions.

Controllability and Bisimulation. Controllability is a fundamental concept in modern theories of control. Generally, a dynamic system is said to be controllable if a control input transforms the system from an arbitrary state to the zero state in a finite length of time (12). Furthermore, a (formal) language of a discrete-event system is said to be controllable if the prefix closure is invariant under the occurrence of uncontrollable events (13). On the other hand, bisimulation is a fundamental concept in process algebra and concurrent computing (14). It determines whether two processes in a computing system or two states in a state space are equivalent in some sense of actions and state transitions. Three classes of research have been delineated within the topic of controllability and bisimulation. The first class reveals that inherent relationships exist between controllability and bisimulation (15,16). This fact is surprising and supports the suggestion that control theories and computing theories may be put into a unified theoretical framework.

The second class of research is devoted to the introduction of bisimulation relations for conventional dynamic systems (17). It is shown that the abstract notion of bisimulation in the context of open maps may characterize the equivalence relations for discrete event systems, for conventional dynamic systems, as well as for hybrid systems (18,19). The third class of research is concerned with how to control a system so that the resultant closed-loop system bisimulates or approximately bisimulates another system in some sense (20).

Feedback Complexity and Limitation. It is well known in the control community that feedback is not a panacea for achieving control goals. There are limitations on the role that feedback can play (21). Normally, a closed-loop control system is composed of controller and a controlled object, which communicate with each other in a collaborative manner to achieve a given control objective. Feedback between the controller and the controlled object defines a kind of communication. On the other hand, the communication complexity theory is a well-established area in computer science, which is concerned with why communication is necessary for two collaborative agents to complete a given task (22). A natural question arises: Can feedback limitation be formulated in the context of communication complexity theory? This research topic remains largely unexplored.

Cybernetic Software Engineering

Cybernetic software engineering (CSE) treats software development as a control problem and applies control theoretic principles to guide software process improvements and quality assurance (23). Because a software development process is often divided into several phases such as requirement analysis, design, and testing, control theoretic principles are applied to individual phases.

Software Requirement Acquisition. Software requirement acquisition is an interactive process between the software development personnel and the user, and feedback is an intrinsic feature of the process. It is argued that the requirement acquisition process be treated as feedback requirements process control (RPC) system, where the requirements specification of the application serves as the object to be controlled (24). The RPC perspective can then be applied to assess the quality of the requirement acquisition process and to guide the corresponding process improvement. This research area is in its early stages of development.

Software Synthesis. Because most software systems can be treated as a control system, it is natural to raise the question whether control theories can help guarantee the correctness of software design solutions. A modest amount of research has been devoted to address this question where the control theories of discrete-event systems are applied. In the work of Marchand and Samaan (25), and Wang et al. (26) the software under synthesis, modeled by a polynomial dynamical system (PDS) serves as the required controller, whereas the operating environment serves as the controlled object. Sridharan et al. (27,28) applies the theory of supervisory control to synthesize the safety controllers for ConnectedSpaces, which is a collection of one or more devices, each described by its Digital Device Manual and reachable over a network. On the other hand, it is shown that software fault-tolerance can be treated as a robust supervisory control problem and the traditional idea of diverse redundancy can be avoided (29).

Software Test Management. A control mechanism has been used to regulate the behavior of the system test process. A mathematical model capturing the dominant behavior of the process has been developed (7). The parameters of the model are calibrated using data from the ongoing process by means of a least squares approach. A parametric control approach is then used to regulate the process and to correct problems such as schedule slippage. The approach has been statically validated using a Kronecker product to conduct a sensitivity analysis (30) and has also been successfully applied to a series of projects from large corporations (31).

Adaptive Testing. Adaptive testing is the software testing counterpart to adaptive control and is the outgrowth of the controlled Markov chain (CMC) approach to software testing. In the CMC approach, the software under test serves as the controlled object and is modeled by a controlled Markov chain, whereas the test strategy serves as the corresponding controller. The software under test and the test strategy make up a closed-loop feedback control system. Although the test strategy uses the test history to generate or select the next set of test cases to be applied to the software under test, adaptive testing implies that the underlying parameters in the test strategy be also updated online during testing by using the test history. It is shown that adaptive testing can be applied not only for software reliability improvements by removing the detected defects (11,32), but also for software reliability assessment by freezing the code of the software under test (33).

Cybernetic Autonomic Computing

Autonomic computing (referred to later as *CAC)* is an initiative launched by IBM (34). It is aimed at making computing systems self-managing, which implies that computing should be self-configuring, self-optimizing, self-healing, and self-protecting. By cybernetic automatic computing we refer to autonomic computing achieved by applying control-theoretic or cybernetic principles and methods. Moreover, computing systems are treated as a feedback closed-loop control system and thus should be self-stabilizing. Several research topics have been addressed in the literature as follows.

Software-Aging Control. Aging is a phenomena widely studied in hardware reliability community. Hardware systems tend to age because of physical deterioration such that the corresponding failure rate function behaves as a nondecreasing function of the operating time. In 1990s software systems also suffered from the aging phenomena (35,36). Software aging emerges as a result of computing resource contention, memory leakage, file-space fragmentation, and so on. It causes the computing systems to demonstrate performance degradation and then to hang, panic, and crash. This is particularly true for Web servers in the Internet environment. Three questions can be raised for software aging. First, what are the underlying aging mechanisms (37,38)? Second, how can software aging be modeled (39,40)? Finally, how can software aging be forecasted and controlled (41,42)? Software-aging control is an interesting research area deserving of additional investigation.

Adaptive Software. The environments where software products are executing today have considerably increased in complexity. The number of simultaneous users on distinct platforms with different resource constraints and the dynamic interaction among all of these elements constitute the basis for this complex and often open environment such as the Internet. Adaptive software operating in the environment of this kind identifies the changes in the environment, adjusts its internal architecture and/or parameters, and assesses its behavior to provide satisfactory quality of service. Feedback is an imperative kind of activity for adaptive software and a crucial problem is how to formulate, to quantify, and to optimize the underlying feedback mechanism. Three questions have been partially addressed in the literature. First, what architectures should be adopted for adaptive software (43,44)? Second, what control algorithms should be developed for adaptive software (45–47)? Finally, what quality attributes should be employed to assess adaptive software (48)? Adaptive software is a research topic in which control-theoretic principles and methods can play a fundamental role.

Self-Stabilizing Software. The notion of self-stabilization was introduced by Dijkstra in 1974 (49) and a many self-stabilizing computing algorithms has been reported in the literature (50,51). A system is said to be self-stabilizing if, regardless of its initial state, it is guaranteed to arrive at a legitimate state in a finite number of steps. This implies that self-stabilizing software can resume normal operation in the presence of transient software faults. Although it is observed that self-stabilization shares some concepts with self-management in autonomic computing (52), research on self-stabilizing software is still in early stages. This topic is important for two reasons. First, existing mainstream mechanisms for software fault-tolerance lack solid theoretical foundation (29) and self-stabilizing software may be considered as a new kind of fault-tolerant software that is theoretically justified. Second, the notion of self-stabilization is different from the traditional notion of Lyapunov's stability, and self-stabilizing software may lead to a new theory of stability.

Autonomic Computing Prototyping. Although autonomic computing systems are claimed to mimic the autonomic neurons in biological systems (53,54), the field of autonomic computing does not currently provide researchers with a clear idea of what is required to develop an autonomic computing system (48). What are the fundamental concepts and principles of autonomic computing? What are the qualitative and quantitative goals of autonomic computing? What are the architectures that autonomic computing systems should adopt? What are the computing models and algorithms that autonomic computing should be based on? All these questions should be addressed in the research works on autonomic computing. By autonomic computing prototyping we mean that these questions are addressed and examined by developing various prototyping systems. These systems can be observed in the various works presented at the international conferences on autonomic computing (55,56).

Software-Enabled Control

Software-enabled control (SEC) was initially a research program launched by the U.S. Defense Advanced Research Projects Agency (57,58). The motivation for SEC is twofold. First, conventional control systems, including adaptive control systems, and robust control systems, are often over-designed based on simplified models of system dynamics and well-defined operational environments. This leads to underperformance in normal environments and control vulnerabilities that arise in extreme environments such as damaged control surfaces for modern aircrafts. Second, developments in software technology have enabled new apparatus for control systems, including device networks, smart sensors, programmable actuators, and systems-on-a-chip. A challenge is how to design control and software, or how to exploit software and computation to achieve new control capabilities. This requires a new perspective of system dynamics. Besides conventional accounts of parameter uncertainty, noise, and disturbance, the new system dynamics should also take into account dynamic tasking, sensor and actuator reconfiguration, fault detection and isolation, and structural changes in plant model and dimensionality. It should also treat software as a dynamic system which has an internal state, time scales, transients and saturation points, responds to inputs, and produces outputs. SEC is now an established research area that shares the general idea of software cybernetics (57,58). In the following we identify three research topics that relate to SEC.

Control Software Architecture. Research in control software architecture is concerned with particular types of software architectures that fit well the implementation of complex control algorithms. There are two primary concerns. First, the control software of concern is mostly real-time embedded software. This is particularly true for airborne software of modern aircrafts. Second, monolithic structures should be avoided to reflect the closed-loop feedback feature of control systems (57,58). Overall, the control software architecture should follow the new perspective of system dynamics. An outgrowth of this research topic is the so-called open control platform (OCP) for Unmanned Aerial Vehicles (UAVs), which is an object-oriented software infrastructure that allows seamless integration of cross-platform software and hardware components in any control system architecture.

Software-Enabled Control Synthesis. The endeavor for software-enabled control was carried out mainly for modern aircrafts in general, and UAVs in particular. How to put flight control, flight management, task management, and software constraints into a unified framework is a grand challenge for computer science, control theory, and software engineering. For example, the distributed controller enabled by emerging real-time middleware support can consist of hierarchical systems, integrated subsystems, or independent confederated systems, such as multi-vehicle systems. It is unclear how to synthesize the required coordinated control algorithms.

Control Software Validation. Conventional software validation assumes that the underlying algorithms implemented by the software under validation are given a priori and are not adjusted online. For example, conventional software testing assumes that a test oracle is given a priori. However, this assumption is not true for software-enabled control that follows the new perspective of system dynamics. Control algorithms may be adjusted or even reconfigured in response to sensor/actuator failures or unexpected conditions. This may trigger the reconfiguration of control software to adopt an alternative software architecture. On the other hand, software reconfiguration in response to software component failures may require that control algorithms are updated to guarantee flight safety. There is a dynamic process of interactions between control algorithms and software systems. This imposes a challenge to software validation (57,58).

APPLICATIONS OF SOFTWARE CYBERNETICS

In this section we highlight applications developed within the research areas presented in Section 2. Since the majority of the research work falls within more than one of the areas from Section 2, we have organized the works by application area. The research topics are then individually categorized based on the four main software cybernetics research areas defined in Section 2 and are also listed below.

- Fundamental Principles (*FP*)
- Cybernetic Software Engineering (*CSE*)
- Cybernetic Autonomic Computing (*CAC*)
- Software-Enabled Control (*SEC*)

Process Management

By *process management* we refer to the work directed toward the task of bringing control-theoretic approaches to bear on the perennial problems of software process improvement and control. Xu et al. (24) have mapped the 66 key practice areas from the Requirements Engineering Good Practice Guide (59) to the corresponding parts of a typical control system (i.e., to actuators, sensors, etc.) and sketch an overview of the process of building a Requirements Process Controller. Their work falls in the *CSE* area.

Management of the construction phase of incremental software development is addressed by Miller (60), where a state-model of the construction phase is proposed, and an outline given of a control strategy based on model predictive control (MPC) (61). The control attempts to minimize the deviation between the actual progress and the schedule while balancing the cost of the control resources with the cost of schedule deviation. These projects can be characterized as part of FP and CSE areas.

Modeling and control of the system test phase (STP) of software development within a control-theoretic formalism has been addressed by Cangussu et al. (7,30) where a state-model is constructed for the STP and a partial eigenvalue-assignment control technique is proposed. The control technique presents a test manager with a set of options which will likely achieve the quality objectives by the schedule deadline. Miller et al. propose (62) a controller for the STP model based on MPC. This work can also be characterized as part of FP and *CSE* areas.

Buy and Darabi build upon their work on time-extended Petrinets (63,64) to construct a supervisory controller capable of enforcing constraints on a class of workflow processes. Workflows can be used to describe many processes, including those in software development. This work is best characterized by the area of CSE.

Padberg has studied the link between software process modeling and Markov Decision Theory (65), where a model of software development is proposed as a Markov Decision Process, and an optimal schedule is derived using a dynamic programming approach. CSE again is the best characterization of this work.

Software Development

Although body of work proposes architecture and design elements to support software cybernetic implementations, these elements are out-of-scope for the present survey.

Instead we focus on the *actual usage* of control-theoretic techniques and ideas.

The task of designing software can be aided by supervisory control techniques which commonly augment existing systems to impose constraints. Examples of design synthesis via supervisory control are given in Sections 3.1 and 3.4.

Software testing has also received considerable attention from the software cybernetics community (11). Cai et al. (32) view the software under test as a controlled object which is modeled by a controlled Markov chain. The testing strategy is synthesized as an optimal controller of the software under test. This body of work falls within the CSE area.

Research has also focused on the application of adaptation techniques to improve random testing. Chan et al. (66) propose an adaptive *center of gravity* constraint to pure random testing to improve the input domain coverage with fewer tests. Cai et al. (67) propose a dynamic partitioning of the input domain for random testing to improve the test selection process. As above, *CSE* characterizes this work.

Adaptive Software

Control-theoretic foundations for the construction of adaptive software have been studied (45–47). For example, a system identification technique is used to capture the behavior of a software application with respect to a specified resource usage (47). The derived model is then used to predict constraint violations and the software is adapted accordingly to avoid such violations. An increase in software robustness is achieved with this adaptation. The work on adaptive systems is better characterized by the *CAC* research area.

Safety

Software cybernetics has been used to address the enforcement of safety policies in collaborative environments. Sridharan et al. (27,28) propose a safety enforcement environment called “ConnectedSpaces” with a formalism for describing and exchanging the safety policy of a device within a ConnectedSpace. A form of supervisory control is then applied to achieve online generation of a safety controller for the ConnectedSpace that adapts as devices enter and exit the ConnectedSpace. This research project involves both areas of *FP* and *CSE*.

Adaptive software introduces its own safety concerns: Is it possible for the adaptation to fail and leave the system in a dangerous state? This issue is addressed by Liu et al. (68) where a stability monitor is constructed based on Lyapunov Stability Theory (69). The stability monitor determines whether the current data will prevent the adaptation process from converging (i.e., because of due to abnormal or incorrect data) and, if so, it prevents the data from reaching the adaptation routine. FP and CAC characterize these projects.

Networking

Techniques based on control theory have been applied to common problems in networking as well. Moerdyk et al. (70) propose a hybrid optimal controller based on MPC which achieves load-balancing in a cluster of computer nodes.

Tan et al. (71) propose a technique for handling high-bandwidth traffic aggregates (e.g., DOS attacks) by installing rate throttles in upstream routers and building control-theoretic algorithms to adaptively, robustly, and fairly set the throttle rates at the routers. CAC can be used to characterize the projects in this section.

Fault Tolerance

Software rejuvenation refers to the idea that software can repair its internal state to prevent a more severe future failure. A framework for adaptive software rejuvenation is proposed by Bao et al. (42) with examples given for monitoring and for adapting the rejuvenation schedule in response to resource loss (e.g., memory leaks.)

A self-stabilizing program is one which guarantees the arrival at a legitimate state after a finite number of steps, regardless of the initial state (49). Gouda and Herman (72) relate program adaptation to self-stabilization in the context of fault tolerance. The research areas of *CAC* and *SEC* characterize these projects.

Information Security

Venkatesan and Bhattacharya (73) propose a threat-adaptive security policy in which a trust model is developed as a finite state machine from a set of rules specifying how trust is to be adjusted. Depending on the level of trust, the system requires varying levels of authentication; the intent is to improve performance while retaining control over the access of untrusted users.

An approach to security quantification is proposed by Griffin at al. (74). A state-space representation of security is proposed, followed by a stochastic attack model. Analysis of the pair yields estimates of mean time-to-failure, the probability of reaching a particular fail-state, and a method of optimizing the security policy. *FP* and *CSE* are a good characterization for these work.

FUTURE DIRECTIONS

Many areas have already benefited from the use of control theoretical aspects but much more needs to be explored. There is always the alternative of increasing the number of areas where software cybernetics can be applied. This is naturally occurring as more and more researchers embrace the benefits of applying control techniques and theories within their research areas. The diverse areas surveyed in this article provide a clear indication of this phenomena. However, most of the research on software cybernetics can be considered to be in their preliminary stages and much more detailed solutions with a full body of results/concepts must be in place before software cybernetics can achieve a reasonable level of maturity.

For example, the work on software project management has been almost restricted to the final phases of the development process or more specifically to the testing phases. This occur because later phases are easier to quantify and less subjective than early phases of the development process. The lack of better and/or more precise quantification mechanisms to represent early phases are not only a matter of their subjectivity but also a matter of the immaturity of software engineering (3). Clearly, software cybernetics research in this area has to move toward all the phases of the development process until a full body of models and control mechanisms are in place to regulate the entire development process.

Another aspect to be considered is how software cybernetics will require the development of new techniques (or the adjustment of existing ones) used in control. For example, the quantification of noise is well understood and used when controlling physical systems. However, noise in software is difficult to quantify. Though most researchers make assumptions about noise (for example, assuming a white Gaussian (Normal) random noise (75), $\omega[n] \sim N[0, \sigma^2]$, of zero mean and variance σ^2 at an instant n to represent noise), we believe that there has not been a detailed study about the quantification of noise for software systems. The availability of such a study may lead to the development of new techniques to handle the specifics of software systems.

CONCLUDING REMARKS

Control theory is a tool that can be used to regulate systems and processes. Software systems do not operate in a static environment and many aspects of software (either at operating system, networking, or application level) need to be regulated to improve performance, increase reliability, and even regulate resource usage. Similarly, the same concepts can be applied to regulate the development process of a software product. Software cybernetics is therefore an area that brings together researchers from both the software and the control systems communities to develop solutions for these problems. The initial results of research projects on software cybernetics are an indication of the benefits of molding this new exciting area. Developments targeting network security, safety, testing process, and fault tolerance, among others, demonstrate this potential.

BIBLIOGRAPHY

1. Wikipedia. Available: http://en.wikipedia.org/wiki/Main_Page.
2. L. Osterweil, Software processes are software too, *Proceedings of the 9th International Conference on Software Engineering,* 1987, pp. 2–13
3. M. M. Lehman, D. E. Perry, and W. M. Turski, Why is it so hard to find feedback control in software process?*Proc. of 19th International Australasian Computer Science Conference, Melbourne, Australia, 1996*, pp. 107–115.
4. N. Wiener, *Cybernetics: or Control and Communication in the Animal and the Machine*, New York: John Wiley, 1948.
5. K. Y. Cai, J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, An overview of software cybernetics, *Proc. of the 11th International Workshop on Software Technology and Engineering Practice,* 2003, pp. 77–86.
6. G. C. Goodwin, S. F. Graebe, and M. E. Salgado, *Control system design*. Upper Saddle River, NJ: Prentice Hall, 2001.
7. J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, A formal model of the software test process, *IEEE Transactions on Software Engineering*, **28**(8): 782–796, 2002.
8. Introduction to SDL 88. Available: http://www.sdlforum.org/sdl88tutorial/index.html, 2002.
9. P. Wang and K. Y. Cai, Representing extended finite state machines for SDL by a novel control model of discrete event systems, *Proceedings of the Sixth International Conference on Quality Software, IEEE Computer Society Press,* 2006.
10. C. Lu, R. ZhangY. LuT. F. Abdelzaher, and J. A. Stankovic, Feedback performance control in software services, *IEEE Control Systems Magazine*, **23**(3): pp. 74–90, 2003.
11. K.-Y. Cai, Optimal software testing and adaptive software testing in the context of software cybernetics, *Informat. Softw. Technol.*, **44**: 841–855, 2002.
12. C. T. Chen, *Linear System Theory and Design*, New York: CBS College Publishing, 1984.
13. P. J. Ramadge and W. M. Wonham, The control of discrete event systems, *Proc. IEEE*, **77**: 81–98, 1989.
14. R. Milner, *Communication and Concurrency*, Englewood Cliffs: Prentice-Hall, 1989.
15. G. Barrett and S. Lafortune, Bisimulation, the supervisory control problem and strong model matching for finite state machines, *Discrete Event Dynamic Sys: Theory and Applicat.*, **8**: 377–429, 1998.
16. J. J. M. M. Rutten, Coalgebra, concurrency, and control. CWI, SEN-R9921, 1999.
17. A. J. van derSchaft, Equivalence of dynamical systems by bisimulation, *IEEE Trans. Auto. Control*, **49**(12): 2160–2172, 2004.
18. E. Haghverdi, P. Tabuada, and G. J. Pappas, Bisimula-tion relations for dynamical, control, and hybrid systems, *Theoret. Comp. Sci.*, 2005.
19. A. Joyal, M. Nielsen, and G. Winskel, Bisimulation from open maps, *Informat. Comput.*, **127**(2): 164–185, 1996.
20. C. Zhou, R. Kumar, and S. Jiang, Control of non-deterministic discrete-event systems for bisimulation equivalence, *IEEE Trans. Auto. Control*, **51**(5): 754–765, 2006.
21. M. M. Seron, J. H. Braslavsky, and G. C. Goodwin, *Fundamental Limitations in Filtering and Control*, New York: Springer, 1997.
22. E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge UK: Cambridge University Press, 1997.
23. K. Y. Cai, T. Y. Chen, and T. H. Tse, Towards research on software cybernetics, *Proc. 7th IEEE International Symposium on High Assurance Systems Engineering,* 2002, pp. 240–241.
24. H. Xu, P. Sawyer, and I. Sommerville, Requirement process establishment and improvement: from the viewpoint of cybernetics, *Computer Software and Applications Conference, 2005. 29th Annual International*, 2005, pp. 89–92.
25. H. Marchand and M. Samaan, Incremented design of a power transformer station controller using a controller synthesis methodology, *IEEE Trans. Softw. Enginee.*, **26**(8): 729–741, 2000.
26. X. Y. Wang, Y. C. Li, and K. Y. Cai, On the polynomial dynamical system approach to software development, *Science in China (Series F)*, **47**(4): 437–457, 2004.
27. B. Sridharan, A. P. Mathur, and K.-Y. Cai, Using supervisory control to synthesize safety controllers for connnectedspaces, *Proceedings of the 3rd International Conference on Quality Software, IEEE Computer Society Press,* 2003, pp. 186–193.

28. B. Sridharan, A. P. Mathur, and K.-Y. Cai, Synthesizing distributed controller for safe operation of connected spaces, *Proceedings of the IEEE International Conference on Pervasive Computing and Communication,* 2003, pp. 452–459.
29. K. Y. Cai and X. Y. Wang, Towards a control-theoretical approach to software fault-tolerance, *Proc. the 4th International Conference on Quality Software,* 2004, 198–205.
30. J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, Using sensitivity analysis to validate a state variable model of the software test process, *IEEE Trans. Softw. Engineer.*, **29**(5): 430–443, 2003.
31. J. W. Cangussu, R. M. Karcich, R. A. DeCarlo, and A. P. Mathur, Software release control using defect based quality estimation, *Pro. of 15th International Symposium on Software Reliability Engineering, Saint-Malo, Bretagne, France,* 2004.
32. K.-Y. Cai, Y. C. Li, and K. Liu, Optimal software testing in the setting of controlled markov chains, *Euro. J. Operat. Res.*, **162**(2): 552–579, 2005.
33. K. Y. Cai, Y. C. Li, and K. Liu, Optimal and adaptive testing for software reliability assessment, *Informat. Softw. Technol.*, **46**: 989–1000, 2004.
34. Autonomic computing: IBM perspective on the state of information technology. Available: http://www.ibm.com/research/autonomic.
35. Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, Software rejuvenation: Analysis, module, and applications, *Proc. The 25th International Symposium on Fault-Tolerant Computing*, 1995, pp. 381–390.
36. E. Marshall, Fatal error: How patriot overlooked a scud, *Science*, **255**: 1347, 1992.
37. K. C. Gross, V. Bhardwaj, and R. Bickford, Proactive detection of software aging mechanisms in performance critical computers, *Proc. the 27thAnnual NASA Goddard/IEEE Software Engineering Workshop,* 2003.
38. M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, Software aging and multifractality of memory resources, *Proc. the 2003 International Conference on Dependable Systems and Networks,* 2003.
39. S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, Analysis of preventive maintenance in transactions based software systems, *IEEE Trans. Comp.*, **47**(1): 96–107, 1998.
40. L. Li, K. Vaidyanathan, and K. S. Trivedi, An approach to estimation of software aging in a web server, *Proc. International Symposium on Empirical Software Engineering,* 2002.
41. T. Dohi, K. Goseva-Popstojanova, K. Vaidyanathan, K. S. Trivedi, and S. Osaki, Preventive software rejuvenation - theory and applications, In H. Pham, *Springer Handbook of Reliability*, New York: Springer-Verlag, 2002.
42. Y. Bao, X. Sun, and K. S. Trivedi, Adaptive software rejuvenation: Degradation model and rejuvenation scheme, *Proceedings 2003 International Conference on Dependable Systems and Networks,* 2003. pp. 241–248.
43. C. Dellarocas, M. Klein, and H. Shrobe, An architecture for constructing self-evolving software systems, *Proc. the Third International Software Architecture Workshop,* 1998, pp. 29–32.
44. P. Oreizy, M. Gonlick, R. Taylor, D. Heilaignel, G. Johnson, N. Medvidov, et al.An architecture-based approach to self-adaptive software, *IEEE Intell. Sys.*, **14**(3): 54–62, 1999.
45. J. Palsberg, C. Xiao, and K. Lieberherr, Efficient implementation of adaptive software, *ACM Transactions on Programming Languages and Systems*, **17**(2): 264–292, 1995.
46. Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. E. Kaiser, and D. Phung, A control theory foundation for self-managing computing systems, *IEEE J. Selected Areas in Communications*, **23**(12): 2213–2222, 2005.
47. J. W. Cangussu, K. Cooper, and C. Li, A control theory based framework for dynamic adaptable systems, *Proc. of the 19th Annual ACM Symposium on Applied Computing,* 2004.
48. P. Lin, A. MacArthur, and J. Leaney, Defining automatic computing: A software engineering perspective, *Proc. the 2005 Australian Software Engineering Conference.* 2005.
49. E. W. Dijkstra, Self-stabilizing systems in spite of distributed control, *Commun. ACM*, **17**(11): 643–644, 1974.
50. M. Schneider, Self-stabilization, *ACM Computing Surveys*, **25**(1): 45–67, 1993.
51. S. Dolev, *Self-Stabilization*, Cambridge MA: The MIT Press, 2000.
52. K. Herrmann, G. Muhl, and K. Geihs, Self management: The solution to complexity or just another problem, *IEEE Distrib. Syst. Online*, **6**(1), 2005.
53. A. G. Ganek and T. A. Corbi, The dawning of the auto-nomic computing era, *IBM Sys. J.*, **42**(1): 5–18, 2003.
54. J. O. Kephart and D. M. Chess, The vision of au-tonomic computing, *IEEE Computer*, **36**(1): 41–50, 2003.
55. *Proc. of the International Conference on Autonomic Computing.* 2004.
56. *Proceedings of the international conference on auto-nomic computing. IEEE Computer Society,* 2005.
57. T. Samad and G. Balas, ed. *Software-Enabled Control: Information Technology for Dynamical Systems*, Hoboken: IEEE Press, 2003.
58. J. S. Bay and B. S. Heck, Software-enabled control: An introduction to the special section, *IEEE Control Sys. Mag.*, **23**(1): 19–20, 2003.
59. I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, New York: John Wiley and Sons, Inc., 1997.
60. S. D. Miller, A control-theoretic aid to managing the construction phase in incremental software development, *30th Annual International Conference on Computer Software and Applications (COMPSAC),* 2006.
61. E. F. Camacho and C. Bordons, *Model Predictive Control*, New York: Springer Publication, 2004.
62. S. D. Miller, R. A. DeCarlo, A. P. Mathur, and J. W. Cangussu, A control-theoretic approach to the management of the software system test phase, *J. Sys. Softw.; Special section on Software Cybernetics*, **11**(79): 1486–1503, 2006.
63. U. Buy and H. Darabi, Deadline-enforcing supervisory control for time Petri nets, *CESA '2003 - IMACS Multiconference on Computational Engineering in Systems Applications, Lille, France,* 2003.
64. U. Buy and H. Darabi, Sidestepping verification complexity with supervisory control, *Proc. 2003 Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation - The Monterey Workshop Series, Chicago, Illinois,* 2003. Available: www.cs.uic.edu/shatz/SEES.
65. F. Padberg, Linking software process modeling with markov decision theory, *Computer Software and Applications Conference, 2004. 28th Annual International*, 2004, pp. 152–155.
66. F. T. Chan, K. P. Chan, T. Y. Chen, and S. M. Yiu, Adaptive random testing with eg constraint, *Computer Software and Applications Conference, 2004, Proceedings of the 28th Annual International*, 2004, pp. 96–99.

67. K.-Y. Cai, T. Jing, and C.-G. Bai, Partition testing with dynamic partitioning, *Computer Software and Applications Conference,* 2005, 113–116.

68. Y. Liu, S. Yerramalla, E. Fuller, B. Cukic, and S. Gu-rurajan, Adaptive control software: can we guarantee safety?*Computer Software and Applications Conference, 2004. Proceedings of the 28th Annual International*, 2004, pp. 100–103.

69. V. I. Zubov, Methods of A. M. Lyapunov and Their Application. U. S. Atomic Energy Commission, 1957.

70. B. Moerdyk, R. A. Decarlo, D. Bird-well, M. Zefran, and J. Chiasson, Hybrid optimal control for load balancing in a cluster of computer nodes, *Proceedings of the IEEE International Conference on Control Applications,* 2006.

71. C. Tan, D. Chiu, J. C. S. Lui, and D. K. Yau, Handling high-bandwidth traffic aggregates by receiver-driven feedback control, *Computer Software and Applications Conference, 2005*, 2005, pp. 143–145.

72. M. G. Gouda and T. Herman, Adaptive programming, *IEEE Trans. Softw. Engineer.*, **17**(9): 911–921, 1991.

73. R. M. Venkatesan and S. Bhattacharya, Threat-adaptive security policy, *Performance, Computing, and Communications Conference, 1997. IPCCC 1997*, 1997, pp. 525–531.

74. C. Griffin, B. Madan, and T. Trivedi, State space approach to security quantification, *Computer Software and Applications Conference, 2005. COMPSAC 2005, 29th Annual International*, 2005, pp. 83–88.

75. A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, 2nd ed., Reading, MA: Addison-Wesley Publishing Company Inc, 1994.

JOÃO W. CANGUSSU
University of Texas at Dallas
Richardson, Texas
KAI-YUAN CAI
Beijing University of
Aeronautics and Astronautics
Beijing, China
SCOTT D. MILLER
ADITYA P. MATHUR
Purdue University
West Lafayette, Indiana

S

SOFTWARE EFFORT PREDICTION

INTRODUCTION

Empirical modeling to predict software development effort has been an important topic in software engineering studies for over 30 years. The basic underlying premise is that historical data about past projects can be employed as a basis to predict effort for future projects. For both engineering and management purposes, accurate prediction of effort is of significant importance, and to improve estimation accuracy is an important goal of most software organizations. A continuing pursuit exists to derive better techniques and models to improve predictive performance.

Until recently, the emphasis has been on algorithmic models that postulate some functional relationship between effort and significant project characteristics, and that are derived from project data using statistical techniques such as regression analysis. Some of these models incorporate tuning parameters so that the model can be adapted to different development environments. In recent years, an increased emphasis has been on the use of machine learning and on neural network approaches. Some of these have produced promising results. An important goal of these studies is to characterize projects by the features that have the most impact on effort and use them as independent variables in prediction models. For example, size, complexity, functionality, and so on all potentially are relevant attributes that impact effort.

Formally, the development of a predictive model can be seen as finding a functional mapping between a project's features, called inputs, and between effort, called output, that satisfies some predictive performance criterion. The data (D) and the function (f) can be represented as follows:

$$D = \{\mathbf{x}_i, y_i, \mathbf{x}_i\ R^d, y_i\ R, i = 1, 2, , n\} \tag{1}$$

and

$$y = f(x; \beta) + e \tag{2}$$

Here x denotes the software project features, d is the number of features, y is the effort, n is the number of projects in the dataset, β is the parameter set, and e is an error term. The derived model can be seen as a particular realization of the function in Equation (2) obtained by some modeling technique. The predictive performance of a model is evaluated by some accuracy measure or measures. Two measures employed commonly in software effort research are the mean magnitude of relative error (MMRE) and prediction at level υ, Pred (υ), defined as:

$$MMRE_i = \frac{1}{n}\sum_{i=1}^{n}\frac{|Actual\,Effort - Predicted\,Effort\,|}{Actual\,Effort} \tag{3}$$

$$PRED(\upsilon) = \frac{Number\,of\,estimates\,within\,(y \pm \upsilon y)}{n} \tag{4}$$

As the definitions imply, models with low MMRE and high Pred(υ) are preferred.

This article describes the main effort prediction techniques and discusses some issues that develop during the modeling process. The next section presents an overview of the commonly used models and some representative comparative studies. The prediction modeling process is described in the following section. An illustrative example is then used in the next section to derive support vector models from several industrial projects. Finally, some concluding remarks are presented.

MODELS AND COMPARISONS

A large number of effort prediction models (1) have been proposed. This section presents model groupings and provides highlights of commonly used models. Key findings from some comparative studies also are presented. The following represents selected studies and is not intended to be a review of the literature on effort estimation.

Model Categories

Most currently used models use historical data for parameter estimation. Based on the technology employed for model development, current models can be grouped into the following categories:

- Statistical
- Neural networks
- Machine learning

Statistical models are derived from historical data using statistical methods, mostly regression analysis. The statistical models were some of the earliest to be developed. Examples of the statistical models are provided in Refs. 2 and 3. Bayesian models have also been developed (4,5).

Neural network models formulate the prediction problem as developing a trained net to approximate the input-output mapping that is used to derive effort estimates for new projects.

The machine learning techniques used for software effort prediction modeling are rule induction, case-based

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

reasoning (CBR), neuro-fuzzy logic, and so on. Currently, a popular CBR approach is based on the Angel tool (6,7).

Another possible category is expert judgment, where the models are generated by experts. Experts are individuals with knowledge about the development environment and have experience with similar systems. Such approaches clearly are nonrepeatable and are biased. However, for local use, expert judgment can be a very effective and accurate effort estimation technique.

Effort Models

Some of the models from the above categories are summarized below, roughly in a chronological order.

Walston and Felix (2) developed one of the earlier cost models, based on data from IBM systems. They used primarily regression analysis for model fitting. Their basic model was Effort = 5.2(size)**0.91. This simple model was the basis of several later studies.

Putnam (8) proposed a lifecycle manpower distribution model based on his observations for many defense systems. This model had previously been used for hardware projects. Its parameters were determined from previous projects, adjusted for the characteristics of the new projects and new environments. Its analytical form facilitates the derivation of many useful project-related performance measures to monitor and to control resource allocation.

Boehm (9) developed the COCOMO model that is currently one of the most widely used models. It was based on data from 63 projects using a statistical approach and expert judgment. The primary project feature is its size in delivered source instructions. Its basic form is man-months = a*(KDSI)**b. The parameters a and b depend on the software development mode. The man-month estimate is then adjusted by 15 cost drivers. Subsequent revisions (10) and updates have incorporated many current trends and techniques in software development.

Function Points. Most of the algorithmic models use lines of code as an important software feature. However, size must be estimated by other means, which affects adversely the prediction accuracy of the model. To overcome this difficulty, Albrecht and Gaffney (11) use function points as a substitute for size. Function points can be derived from the requirements and the specification documents thus becoming available early in the lifecycle. In these models, effort is related directly or indirectly to function points.

Analogy. Estimation by analogy is a form of case-based reasoning. The key phases of estimation by analogy are the identification of a problem as a new case, the retrieval of similar cases from a repository, the reuse of knowledge derived from previous cases, and the suggestion of a solution for a new case. Several different algorithmic approaches can be used to assess similarity between new case and cases in the repository, for example, the k-nearest neighbor and the fuzzy similarity. Several case studies (6) have documented experiences with the use of analogy.

Neural network. Such models have been proposed within the past 10 years. Wittig and Finnie (12) employed a back-propagation training approach for a multi-layer perceptron network. Shin and Goel (13) developed radial basis function models using data from Bailey and Basili (3), whereas Lim (14) employed support vector machines (SVMs) on many datasets for effort-prediction modeling.

Comparative Studies

Over the past 15 years, several comparisons of modeling techniques have been reported. Key findings from some of these reports are summarized below. However, general conclusions cannot be drawn from such studies because almost every technique requires subjective decisions during the modeling process, which influence the results invariably. Nevertheless, such studies provide some interesting insights into the strengths and the limitations of the various techniques.

Sheppard and Schofield (7) studied the performance of analogy, of stepwise multiple regression, and of multiple regression on a large number of industrial projects developed in different environments for many applications. They found that their analogy approach had a superior performance as judged by MMRE and Pred(25) measures.

Briand et al. (15) presented the results of a comprehensive comparison of common effort estimation techniques. They compared predictive performance on multi-organization and company-specific data and found no significant difference with respect to modeling techniques. However, on their data, regression models outperformed analogy-based estimation.

Dolado (16) evaluated regression, neural networks, and genetic programming for effort estimation using several datasets and concluded that the improvements by machine learning techniques were not impressive.

Lim (14) developed SVM-based effort models for several datasets, (6,11,17,18). Their performance was superior to the models derived by multiple regression, stepwise multiple regression, and analogy (6) in terms of MMRE and Pred(25).

MODEL DEVELOPMENT PROCESS

Developing a prediction model consists of finding some expression for the function in Equation (2), generally called “function approximation” in modeling literature. Various models described above are manifestations of this approximation. The process to derive these functions depends on the chosen technique, but, at an abstract level, a common underlying process exists that is described below and shown in Fig. 1.

Modeling Approach

The first issue is to determine a modeling approach. This decision is governed by the application and the development environment as well as by the available data and by modeling expertise. Requirements of predictive perfor-

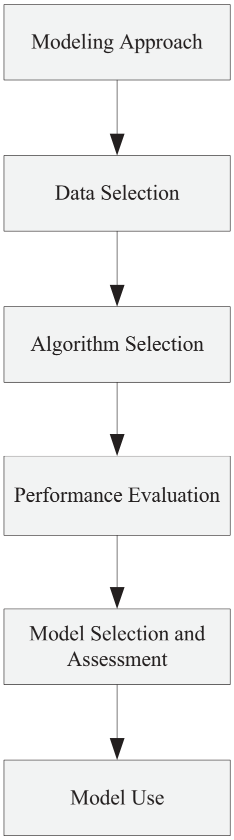


Figure 1. Model development process.

mance, ease of use, and model interpretability also affect this selection.

Data Selection

The modeling approach dictates data selection. For example, if a regression model is to be built, explicit data in the form of Equation (1) is required. The case is the same for neural network models. For case-based reasoning models, a case base and domain knowledge are required. Also, preprocessing of the data usually is necessary, which includes dealing with missing data, outliers, normalization, and data transformation, as well as with feature subset selection and with dimensionality reduction. Obviously, data selection has a major impact on model performance.

Algorithm Selection

An algorithm to determine model parameters is chosen to satisfy some desired speed and efficiency constraints. If a linear regression model is chosen, the algorithm is straightforward. For neural network modeling, several algorithms are available. Other decisions include the approach to determine network architecture, error tolerance, and training parameters. For CBR models, issues of distance measures, adjustment mechanism, and so on, are to be addressed. This process is nontrivial, which requires much insight into the algorithms and their tradeoffs. As examples, see Ref. 13 for radial basis function models and Ref. 19 for SVMs.

Performance Evaluation

By using measures such as MMRE and Pred(υ), multiple models usually are developed and their performance is evaluated. Some model validation approaches are employed for model comparison. One approach is to divide the data into training, validation, and test sets. Models are derived from the training set. Their performance is evaluated on the validation set and usually the best performing model is selected. If the dataset is of limited size, k-fold cross-validation is more appropriate. The data are divided into k groups of almost equal size. A model is developed from (k-1) sets and its performance is evaluated on the kth set. This process is repeated k times and the average of the k values is a measure of model performance. In the special case when the dataset is small, k = 1 is used and is called the leave-one-out cross-validation approach.

Model Selection and Assessment

Model performance on the test set is considered to be a measure of its generalization performance (i.e., its predictive performance on future projects). Other factors that are employed generally for model assessment are ease of use, meaningfulness, interpretability, and so on. Model complexity is also an important concern. A parsimonious model is almost always preferred because, among other things, it requires fewer data features for model development and use.

Model Use

The selected model is employed to predict the effort of new systems. Lessons learned are used to improve future predictions.

SUPPORT VECTOR MACHINE PREDICTION

SVM is a relatively new approach for effort estimation. It possesses some very nice mathematical properties and has shown promising performance in fields such as bioinformatics, medicine, and banking. SVM is a new type of learning algorithm based on statistical learning theory. Specifically, it represents an approximate implementation of the method of structural risk minimization, which states that the generalization error of a learning machine is bounded by the sum of its training error and a term that depends on its Vapnik–Chervonenkis dimension (20). A brief description of SVM is given below, followed by an illustrative example.

Support Vector Machine

Originally, SVMs were employed for linearly separable pattern classification tasks (i.e., for data that are linearly separable in the input space). Even though the above is unrealistic for most real-world applications, its treatment is helpful to understand the SVM fundamentals. For realistic situations, a high dimensional feature space is constructed by using an inner-product kernel $K(x,x_i)$. The input data are mapped into this feature space and the linear classification problem is solved in the feature space. Recently, this approach has been extended to solve nonlinear regression problems such as effort estimation modeling. In this context, the function in Equation (2) is nonlinear in the parameters and can be written as

$$y = f(x;\beta) \equiv \sum_{i=1}^{n} y_i \alpha_i K(x, x_i) \tag{5}$$

where for the Gaussian kernel is

$$K(x;x_i) = exp\{-\frac{1}{2\sigma^2}\|x - x_i\|^2\} \tag{6}$$

The dual problem to be solved for the above model is, given the project data of Equation (1), find the Langrange multipliers *ai*'s such that the following objective function is maximized (21).

$$Q(\alpha_i, \alpha_i') = \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i') - \varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i') - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')K(\mathbf{x}_i, \mathbf{x}_j) \tag{7}$$

$$\text{subject to} \sum_{i=1}^{n}(\alpha_i - \alpha_i') = 0 \tag{8}$$

$$0 \le \alpha_i, \alpha_i' \le C, \quad i = 1, 2, , n \tag{9}$$

In the above, the kernel width σ, the penalty parameter C, and theVapnik loss function ε are to be specified by the user. These parameters are called the support vector hyperparamaters. Finally, the effort prediction model is

$$\hat{y} = \sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) \tag{10}$$

where the α_i's are between 0 and C and **x** are the features of the project for which effort is to be predicted. Thus, for specified σ, C, and ε, and for the software project data D represented by Equation (1), the necessary model parameters are derived by support vector regression, and Equation (10) is used to predict effort.

Illustrative Example

Data about some projects from Ref. 18 and reported in Ref. 6 are used here to develop support vector effort models. It consists of 77 projects, each with nine features (i.e., d = 9 and n = 77 for this data). The data was separated additionally into three groups of 44, 23, and 10 projects according to its development environment. The four datasets are labeled DE, DE1, DE2, and DE3, respectively.

The first step is to determine the support vector hyperparameters, which is an important problem and has been studied extensively in the literature (22). For this example, a new methodology proposed in Ref. 14 is used. It is called SVEG (Support Vector parameter selection using Experimental design-based Generating set search). SVEG combines ideas from the Design of Experiments and from the generated set of search areas. It consists of three steps, namely, to determine hyperparameter ranges, to define an experimental design in the hyperparameter space, and to conduct a generating set search. A central composite design is constructed based on the ranges of σ, C, and ε, determined according to the SVEG methodology. Then, a generating set is constructed and a generating set search is undertaken using the experimental design points as the initial settings for the search.

Performance measures MMRE and Pred(25) are obtained by the leave-one-out approach. Based on the values of these parameters, the support vector regression model is developed as described above. The predictive performance of the developed models is evaluated by generating a set search. Two models are selected, one with the smallest MMRE and another with the largest Pred(25). The average test errors of the selected models are listed in Table 1.

It is noted that the predictive performance improves if the data are from a homogeneous environment, as is the case for DE1, DE2, and DE3 compared with the DE performance. Also included in this table are the performance results from the models developed by stepwise multiple regression and the analogy reported in Ref. 6. From a comparative perspective, the SVEG has better performance on both measures for each dataset. Similar results were found for several other projects (14).

CONCLUDING REMARKS

This article addresses the issue of software effort prediction. To provide a perspective on the current state of practice and research, highlights about commonly used models and key findings from some comparative studies

Table 1. Prediction Performance for Example Data

	MMRE (%)			Pred(25) (%)		
Dataset	Regression	Analogy	SVEG	Regression	Analogy	SVEG
DE	66	64	52	42	36	46
DE-1	41	37	35	45	47	54
DE-2	29	29	24	48	47	64
DE-3	36	26	25	30	70	80

are presented. The modeling approach is illustrated using support vector machines for some industrial data.

BIBLIOGRAPHY

1. M. Jorgensen and M. Shepperd, A systematic review of software development cost estimation studies, *IEEE Trans. Soft. Engineer.*, **33**: 33–53, 2007.
2. C. E. Walston and C. P. Felix, A method of programming measurement and estimation, *IBM Syst. J.*, **16**: 54–73, 1977.
3. J. W. Bailey and V. R. Basili, A meta-model for software development resource expenditures, *Proc. 5th International Conference on Software Engineering*, 107–116. 1981.
4. S. Chulani, B. Boehm, and B. Steece, Bayesian analysis of empirical software engineering cost models. *IEEE Trans. Soft. Engineer.*, **25**: 573–583, 1999.
5. D. Zhang and J. J. P. Tsai, *Machine Learning Applications in Software Engineering*, Singapore: World Scientific, 2005.
6. C. Schofield, An Empirical Investigation into Software Effort Estimation by Analogy, Ph.D. Dissertation, Dorset, UK: Bournemouth University, 1998.
7. M. Shepperd, and C. Schofield, Estimating software project effort using analogies, *IEEE Trans. Soft. Engineer.*, **12**: 736–743, 1997.
8. L. H. Putnam, A general empirical solution to the macro sizing and estimating problem, *IEEE Trans. Soft. Engineer.*, **4**: 345–361, 1978.
9. B. W. Boehm, *Software Engineering Economics*. New York: Prentice-Hall, 1981.
10. B. W. Boehm, COCOMO II Experience and Plans, *ESCOM97*, Berlin, 1997.
11. A. J. Albrecht and J. R. Gaffney, Software function, source lines of code, and development effort prediction: A software science validation, *IEEE Trans. Soft. Engineer.*, **9**: 639–648, 1983.
12. G. Wittig and G. Finnie, Estimating software development effort with connectionist models, *Informat. Soft. Technol.*, **39**: 469–476, 1997.
13. M. Shin and A. L. Goel, Empirical data modeling in software engineering, *IEEE Trans. Soft. Engineer.*, **26**: 567–576, 2000.
14. H. Lim, Support Vector Parameter Selection Using Experimental Design Based Generating Set Search (SVEG) with Applications to Predictive Software Data Modeling, Ph.D Dissertation, Syracuse, N. Y., Syracuse University, 2004.
15. L. Briand, et al. A replicated assessment and comparison of common software cost modeling techniques, *Interna. Conf. Software Engineering*, 377–386, 2000.
16. J. Dolado, Limits to methods in cost estimation, Technical Report, Spain: University of Vasque County, 1999.
17. C. F. Kemerer, An empirical validation of software cost estimation models, *Commun. ACM*, **30**: 416–429, 1987.
18. J. M. Desharnais, Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction, *Masters Thesis*, Montreal: University of Quebec, 1988.
19. H. Lim and A. L. Goel, Support Vector Machines for Data Modeling with Software Engineering Applications, in H. Pham (ed.), *Springer Handbook of Engineering Statistics*, London: Springer Verlag, 2006.
20. V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley-Interscience, 1998.
21. S. Haykin, *Neural Networks – A Comprehensive Foundation*, 2nd ed., Upper Saddle River, N.J., Prentice Hall, 1999.
22. R. M. Rifkin, Everything Old is New Again: A Fresh Look at Historical Approaches in Machine Learning, Ph.D. Dissertation, Cambridge, MA: Massachusetts Institute of Technology, 2002.

HOJUNG LIM
Korea Electronics Technology Institute (KETI)
Sungnam, Korea
AMRIT L. GOEL
Syracuse University
Syracuse, New York

SOFTWARE INSTRUMENTATION

INTRODUCTION

Since the inception of computer systems, development of reliable (i.e., bug-free) and efficient (i.e., extracting maximum performance from the available hardware resources) software programs has been one of the major concerns for the developer/programmer community. Over time, the task of ensuring such reliability and efficiency has become progressively complex along with the software systems themselves. Consequently, to assist the individual coders, a set of tools and methods for program analysis, debugging, and profiling has been developed. Although such tools usually vary widely in their designs and intents, many of them are similar in that they employ a technique called *software instrumentation*, or simply *instrumentation*. Naturally, instrumentation has been an active area of research over the past few decades.

In its simplest form, instrumentation involves adding extra code to a program's text. The intent usually is to *monitor some kind of program behavior*—either for debugging or for optimization purposes. For example, a programmer who wants to optimize the execution time of a large program might first like to know the regions of code that are executed most of the time, and then go on to optimize those frequently executed sections (rather than naively trying to optimize the entire program–a large part of which might have been written to handle extremely rare error cases). To accomplish this task, the programmer at least needs to know how many times each *function* in the program has been invoked i.e., she/he needs to *monitor the function calling behavior of the program*. The simplest way to accomplish this is to employ manual instrumentation—that is, to add extra statements at the beginning of each function to increment a counter variable that corresponds to that function, and printing all the counters when the program exits.

The abovementioned manual strategy no longer remains viable as the size of the program grows (e.g., large programs with hundreds of functions) or as the complexity of the monitoring task increases (e.g., monitoring execution of each statement or each memory access rather than each function). Consequently, many program analysis tools have been developed over the years for automatic instrumentation of program code. Because manual instrumentation is employed mostly in ad-hoc basis without any general methodology or tool support, only automated instrumentation tools will be discussed in this article.

Before going into more details, it is necessary to define a few terms clearly that will be used throughout this article. As has been mentioned already, instrumentation means adding extra code to a program or modifying the program text—usually to monitor some program behavior for the purpose of performance analysis or debugging (some other uses of instrumentation will be introduced later in this article). The extra code added is termed as *instrumentation code*. When instrumentation code is added automatically by a tool, the tool is called an *instrumenter*. The application being modified through instrumentation is called the *target application*.

The rest of this article is organized as follows. In the next two subsections, the most important usages of instrumentation are introduced, and a classification of different instrumentation techniques is provided. Two major classes of instrumentation techniques are discussed in detail in the following two sections. Finally, the article is concluded with a brief summary.

Purposes of Instrumentation

Instrumentation techniques vary widely in their design and implementation. In the following, the different purposes for which instrumentation techniques are used will be introduced:

1. **Profiling, performance analysis, and program optimization.** Profiling and performance analysis involve identification of most computation-intensive sections of an application program, followed by the optimization of such sections by partial or complete rewriting. Computation-intensive sections of a program might correspond to coarse-grain program elements, such as complete *functions*, or might be more fine grained such as *loops, basic blocks*[1] or even individual *program statements* or instructions. The required information for performance analysis can be obtained by attaching counters to each program element of interest (i.e., functions or basic blocks) and incrementing them when the corresponding program element is executed. The extra code for initialization, incrementing, and printing of such counters can be added through instrumentation. A wide variety of profiling tools, such as UNIX prof and GNU gprof (2), MIPS Pixie (3) and QPT (4), use instrumentation for various profiling tasks such as *basic block counting* and *function call-graph profiling*. Instrumentation is also used in several other important program analysis tool such as *program tracing* (5) and *memory trace generation for cache simulation*. Tools like Intel Thread Profiler (6) extend performance profiling even to multithreaded applications.

   Apart from manual optimizations based on profiling, many compilers, such as GNU gcc (7), Microsoft's Visual C++ (8) and Intel C++ and Fortran compilers (9), can automatically use profiling data to

[1] Basic blocks are well-known abstract program elements. A basic block is defined as the maximum sequence of program statements that do not contain any control flow. For detailed definition of basic blocks please consult Ref. 1.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

optimize applications. This method is usually known as *profile guided optimization*. There are certain performance events, such as cache miss rate and branch prediction, which are difficult to track through static analysis. Instrumentation is also used to dynamically optimize programs by analyzing such events (10,11).

2. **Programming error detection and debugging.** Instrumentation is used heavily in many program error detection—especially memory access error detection—tools. Memory access errors (e.g., out-of-bound accesses from statically or dynamically allocated arrays, memory leaks, etc.) are the most common sources of program malfunctions or performance degradations. Several tools—such as IBM Rational Purify (12), Valgrind (13), and Insure++ (14)—use instrumentation for detecting memory-access errors. For example, IBM Rational Purify automatically modifies the object code of a program by inserting instrumentation code before every memory access instruction (i.e., load or store) and every invocation of dynamic memory allocation/deallocation routines (e.g., malloc and free). When the program is executed, the added instructions keep track of the memories allocated/deallocated during execution and signal error when access violations occur. The results of memory error detection process can also be used to eliminate bugs from the program. Other tools such as Intel Thread Checker (15) address issues in multithreaded programming (e.g., detection of race conditions).

3. **Virtualization.** Virtualization is a technique that emulates or simulates applications, operating systems (OSs), or whole platforms on a host environment (i.e., a single physical computer running a specific operating system). The emulation engine is usually called a *Virtual Machine* (VM). Virtual machines are frequently used for porting and testing of software on new or unsupported hardware/OS, or for migrating to a new hardware/OS. Another common usage of virtualization is to facilitate implementation of easily portable and interoperable languages [such as C# (16) or Java (17)], and efficient execution engines for scripting languages [such as Python (18) and Tcl (19)]. For example, programs written in Java are translated by the Java compiler to a platform-independent *bytecode* that, in turn, is executed by a platform specific implementation of the Java Virtual Machine (JVM) (17).

   Another application of virtualization is in building *Instruction-Set Simulators*[2] (ISSs) (20) that can imitate the behavior of Instruction Set Architectures (ISAs) other than that of the host machine. Apart from allowing programmers to debug, test, and optimize their applications on different ISAs, ISSs can also be used to evaluate the hardware performance while designing new processor architectures, or embedded Systems-on-Chip (SoCs).

   Instrumentation techniques are used in many abovementioned virtualization platforms. Tools such as VMWare (21), Virtual PC (VPC) (22), and PinOS (23) use a technique called *Dynamic Binary Instrumentation* (DBI) to emulate the behavior of a *guest OS* on the host computer. These tools have varying purposes—the most prominent being porting, testing, and migration to unsupported OSs. For a more-detailed discussion on virtualization, the interested reader may consult Ref. 24. Other tools, such as the novel *Virtual Co-Processor* (VCP) (25) use instrumentation to facilitate extremely quick ISA simulation and debugging for embedded system programmers. Tools such as Ref. 26 use instrumentation for evaluation of SoC performance in the virtualization environment described in Ref. 27.

4. **Software quality assurance and testing.** Instrumentation is often helpful in many tasks related to software quality assurance and testing. The uses of instrumentation in memory error detection has already been illustrated. Many memory errors—specially memory leaks—do not directly exhibit themselves as bugs but degrade program performance in the long run. Therefore, eliminating them is often a requirement for writing high-quality software.

   Another usage of instrumentation is in *code coverage* analysis. A piece of software is said to have, say 90% code coverage, if 90% of all *Lines-Of-Code* (LOC) are executed at least once for a set of given test data. A high percentage of code coverage usually indicates a highly tested software, whereas a low code coverage percentage indicates that either the software has many unreachable sections of code (and therefore is in need of cleanup), or the test suite is inadequate. Code coverage detection tools, such as Insure++ (14), Quantify (28), and gcov (29), usually insert instrumentation code at the end of each basic block to test whether that basic block has been executed at least once.

One characteristic of instrumentation is its *intrusiveness* (i.e., the instrumentation code almost always modifies the exact program behavior). For example, added code might increase the execution time of the program or might affect the cache behavior by changing the exact memory locations accessed. Such intrusive behavior might not be acceptable in many cases, especially for performance analysis, program error detection, and debugging.

Classification of Instrumentation Techniques

Traditionally, applications are written in human readable high-level languages (such as C, C++, FORTRAN, and

[2] Many of todays ISSs are built using a technique called binary translation, which dynamically translates one Instruction Set Architecture to another during program execution. Because, binary translation is not really instrumentation, it is kept out of discussion in this article.

Pascal), which are automatically translated to binary executables by the compiler tool chain. This binary executable is normally executed on a host machine. Instrumentation techniques are usually classified based on how and when the instrumentation code is added inside a program in this compilation-execution process. Tools that add instrumentation code before execution are called *static instrumenters*, whereas those modifying a program during execution are known as *dynamic instrumenters*. Some examples of prominent static and dynamic instrumentation techniques are provided in Fig. 1 along with pointers to the sections and subsections where they are discussed in more detail.

Many modern high-level languages such as Java and C# are often compiled to machine-independent intermediate formats that, in turn, are executed through interpretive or *Just-In-Time* (JIT) compilation techniques. The .NET framework (30) from Microsoft (Redmond, WA) extends this concept by allowing several different languages to be compiled to a single intermediate format and runs it on a VM called *Common Language Runtime (CLR)*(31). For the sake of brevity, these intermediate formats will be referred to as byte codes. For such languages and platforms, static instrumenters can only instrument the byte codes of corresponding applications, because no executables are available.

Some tools exist that support both static and dynamic instrumentation. Two examples of such tools are Insure++ (14) and Vulcan (10). Insure++ is a runtime memory analysis and error-detection tool for C and C++ which can perform instrumentation at three abstraction levels (1) source code instrumentation, (2) binary instrumentation at link time, and (3) dynamic instrumentation based on the tool Chaperon (part of Insure++ suite). Vulcan is an executable editing framework for instrumenting programs that are composed of components in IA-32, IA-64, and Microsoft Intermediate Language (MSIL) binaries. Vulcan supports both static and dynamic instrumentation, as well as facilities to instrument distributed components. However, it should be noted that tools that support both static and dynamic instrumentation are usually rare.

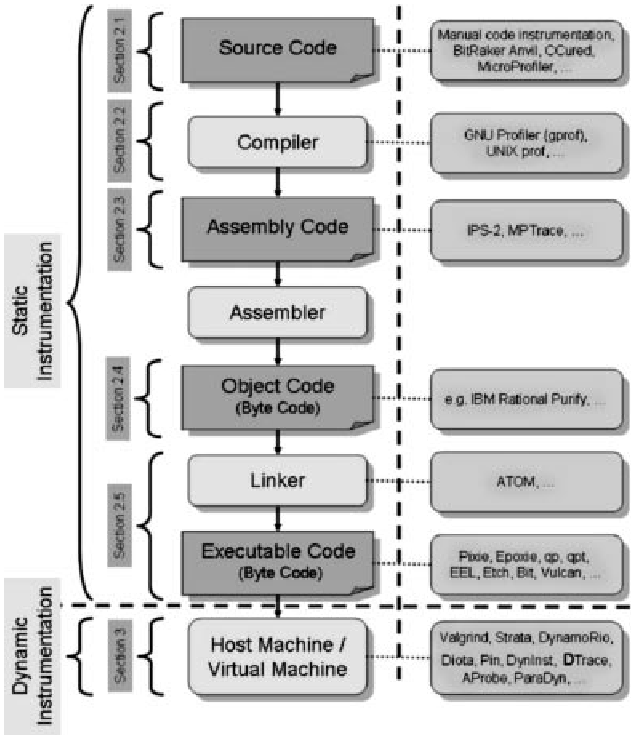

Static Instrumentation
Dynamic Instrumentation
Section 2.1
Section 2.2
Section 2.3
Section 2.4
Section 2.5
Section 3
Source Code
Compiler
Assembly Code
Assembler
Object Code (Byte Code)
Linker
Executable Code (Byte Code)
Host Machine / Virtual Machine
Manual code instrumentation, BitRaker Anvil, CCured, MicroProfiler, ...
GNU Profiler (gprof), UNIX prof, ...
IPS-2, MPTrace, ...
e.g. IBM Rational Purify, ...
ATOM, ...
Pixie, Epoxie, qp, qpt, EEL, Etch, Bit, Vulcan, ...
Valgrind, Strata, DynamoRio, Diota, Pin, DynInst, DTrace, AProbe, ParaDyn, ...

Figure 1. Instrumentation techniques in the compiler tool chain.

STATIC INSTRUMENTATION

As has been mentioned earlier, static instrumentation techniques generally insert instrumentation code inside a program *statically* i.e. during of after compilation *but definitely* prior to execution. Compiling the source code of a program to the linked binary executable generally involves three steps (as shown in Fig. 1):

1. **Compilation.** In this step, the source code written in a high-level programming language (such as C, C++, Java, or Pascal) is translated to the assembly language representation for a target machine.
2. **Assembling.** The assembly language representation of the program is converted to binary object code in this step. For languages like C# or Java, this means translation into some form of bytecode.
3. **Linking.** In this step, a set of separately compiled and assembled binary object files are linked together to produce the target executable. For languages like C# and Java, linking can mean merging of several bytecode files into one single bytecode that can be executed on a VM.

Instrumentation code can be inserted during each of the above three steps. It is also possible to instrument the source code before it is handed over to the compiler *(source-to-source instrumentation)*, or the linked executable produced by the compiler tool chain *(binary instrumentation / rewriting)*. In the following subsections, many such static instrumentation techniques are discussed in detail with a special focus on their relative merits and demerits. The instrumentation techniques have been categorized depending on the step in the compilation process (or, before and after it) where instrumentation code is inserted.

Source-to-Source Instrumentation

Source-to-source transformations can insert instrumentation code directly inside a program's text. Usually, such instrumentation strategies are applied to measure some abstract source-level behavior of a program, rather than testing, debugging on a target machine. The goal of such instrumentation can vary widely—from measuring potential parallelism in scientific/engineering programs (32) to evaluating high-level algorithmic complexity for multimedia and signal processing algorithms (33) and performance estimation during embedded SoC design (26). Naturally, the program instrumentation techniques also vary a lot depending on the final goal. For example, Ref. 32 directly rewrites source code of FORTRAN programs to measure parallelism, Ref. 33 translates C programs to C++ programs and uses overloaded operators to collect execution statistics, whereas Ref. 26 lowers standard ANSI-C code to a *Intermediate Representation* (IR) for instrumentation and then writes back the instrumented IR as low-level C code. Although useful in niche application areas (such as high-performance computing and embedded systems design), source-to-source instrumenters are not widely popular because of the *slow speed of instrumented programs* compared with other forms of static instrumentation, inability to track *library functions* and requirement of *reinstrumentation before compilation* each time a program's source is modified.

One source-to-source instrumentation framework, which is not bound to niche application areas and consequently can find wide acceptance, is CCured (34). CCured has been invented to detect memory access violations in C applications. It analyzes a program and inserts the smallest number of runtime checks needed to detect memory access errors. It claims to be an order of magnitude faster than another popular memory access detection tool purify (12).

Instrumentation During Compilation

During compilation, a compiler can introduce instrumentation code inside a program as it is compiled. Most common examples of such profilers are the UNIX prof and the GNU gprof (2) profilers. For example, the gprof profiler is tightly bound to the *gcc compiler tool chain*, which inserts a call to a monitoring routine at the prolog of each function being compiled. When the instrumented binary executes, the monitoring routine collects execution statistics and dumps it into a file. The self- and cumulative execution times of different functions, as well as the dynamic call graph of the program, are later reconstructed by an analysis program from the dumped information.

The advantages of compiler-based instrumentation are twofold. Firstly, the instrumenter can use the extensive program analysis done by the compiler to insert the instrumentation code intelligently, which reduce the overhead of profiling/tracing/measurement. Second, the instrumentation code can be made to report measurement statistics in human-readable terms such as procedure names, variable names, file names and line numbers by using the database of syntactic and semantic structures constructed by the compiler.

However, several drawbacks of compiler based instrumentation are outlined below:

1. Compiler based instrumentation techniques are not easily extensible. Because most compilers are extremely complex pieces of software, adding new instrumentation functionalities is very difficult.
2. A program may be built from source code written in different languages (which, in turn, might be compiled by different compilers) and precompiled libraries. Compiler-based instrumentation cannot be used in such cases, because each compiler might use a different instrumentation strategy, and the libraries might be compiled without using any instrumentation.
3. Compiler-based instrumentation requires a re-compilation of source files. This is often time consuming and annoying for a programmer who intends to quickly analyze or debug an application.

Instrumentation During Assembling

A compiler generates assembly code for a target machine, which is handed over to an assembler for producing the binary object code. One possible point for instrumentation is to rewrite the assembly code before passing it to the assembler. Two examples of such instrumentation systems are presented in Refs. 5 and 35. In contrast to compiler-based instrumentation, instrumenting the assembly code has the advantage that it can be extended easily to incorporate new functionalities. Still, assembly instrumentation is not very popular, because it suffers from other disadvantages similar to compiler based instrumentation (e.g., tracking library function calls and need to re-assemble before debugging and analysis).

Instrumentation of Object Code Before Linking

Many drawbacks of compiler-based and assembler-level instrumentation strategies can be overcome if instrumentation is performed on the assembled object code. This process of object code modification is called *Object Code Insertion* (OCI). One prominent user of this technique is the software testing and quality assurance tool IBM Rational Purify (12). Purify reads object files generated by existing compilers and adds instructions to track memory leaks and access errors without changing the symbol table or the program semantics. The major advantages of OCI techniques over compiler-based or assembly-level instrumentation are as follows:

1. OCI does not require recompilation of high-level language programs to object code level. Consequently, OCI is much faster than compiler-based instrumentation.
2. Several different languages can be supported easily at the object code level. For example, one major difference between object codes generated from C and C++ is that C++ objects contain *mangled names*. An OCI instrumenter for C object codes can be used easily for C++ objects with the *assistance of a demangler*.
3. OCI works well with all kinds of objects, which includes the ones derived from hand optimized assembly code, as well as precompiled libraries.

Link-Time and Post-Linking Executable Editing

One of the most common forms of static software instrumentation is *executable editing*.[3] Executable editing can be carried out during linking the different components (i.e., object modules and static libraries) of a program or by directly modifying the linked binary executable. Like object code instrumentation, executable editing eliminates most disadvantages of compiler-based and assembly-level instrumentation. Executable editors can be used to instrument programs without recompiling the sources, track statically linked library functions, and handle executables built using a collection of different source languages.

Information about the entire application, which includes the relocation information, is available during (or after) linking an executable. The executable editors usually use this global information for efficiently instrumenting binaries, which is impossible for compiler-based, assembly-level, and even object code-level instrumenters. Such global information is used by many binary tools not only to instrument but also to *optimize programs* (36). However, the focus of this section will be confined to instrumentation techniques based on binary tools.

One of the earliest examples of executable editors is the MIPS profiling and tracing tool Pixie (3), which instrumented fully linked executables. Following Pixie, several binary tools, such as QP, QPT (4), and Epoxie (37) appeared. The major problem with these tools were that they were designed to perform only one single type of instrumentation, typically program tracing and basic block counting. It was difficult to extend or alter the functionalities of these tools. Naturally, these fixed-functionality tools were followed by many *executable editing frameworks* that facilitated the construction of *customized program instrumentation and analysis tools*. The genesis and evolution of these frameworks are discussed in the next few subsections.

It is to be mentioned that several bytecode instrumentation tools also exist for languages like Java or C#. One example is the *Java Runtime Analysis Toolkit* (JRat) (38), which can keep track of timing statistics, function tracing, and so on. JRat currently allows five different kinds of Java application instrumentation, one of which is bytecode instrumentation.

Executable Editing Frameworks. Executable editing frameworks were first conceived as tool-construction systems that could be used to build a variety of instrumentation programs. They are built on the observation that several apparently different tasks such as basic block counting, cache simulation, memory access error detection, and program tracing can be performed by inserting instrumentation code at a few selected points in a program. Such selected insertion points may include, but are not limited to, the beginning and the end of a basic block, prologue and epilogue of a function, and before and after an instruction. An executable editing framework usually provides a common infrastructure that facilitates injection of *user-defined routines* inside a program's executable at some such *selected insertion points*.

To understand the philosophy of executable editing frameworks, a simple program analysis tool, which counts the number of times each conditional branch in the program is taken (39), can be considered. The building of such a program analysis tool can be divided into two phases:

1. **Instrumentation phase**. The instrumenter needs to examine each instruction of the edited executable and test whether this instruction is a conditional branch. If the instruction in question is indeed a conditional branch, then the instrumenter needs to insert a function call that increments a counter.
2. **Analysis phase**. The analysis phase is activated when the instrumented binary executes. The function calls, inserted during the instrumentation phase, increment required counters and finally, dumps the collected statistics into a file before the program exits.

The major hurdle in constructing the above analysis tool is to build a program representation that can be traversed one instruction at a time, examined to detect conditional branches, and modified by inserting instrumentation code. It is relatively simple to write the instrumentation and analysis routines that insert and collect the branch profiling data. This is the general case for many program analysis tools. The philosophy of executable editing frameworks is to let the user write the instrumentation and analysis routines, while a convenient, intuitive and modifiable representation of the program binary is supplied by the framework itself. This work flow reflected is in the Fig. 2 where the binary executable program (or, a prelinking object file) is translated to an internal representation inside the editing framework. A set of *custom instrumentation routines* can traverse and modify this representation using an API, which is also provided by the editing framework. The final instrumented binary may contain *custom analysis routines* inserted through instrumentation.

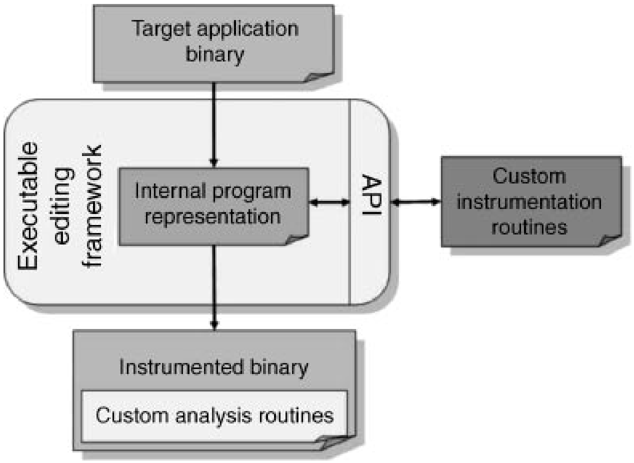

Target application binary
Executable editing framework
Internal program representation
API
Custom instrumentation routines
Instrumented binary
Custom analysis routines

Figure 2. General work flow of executable editing frameworks.

[3] Executable editing is also known by a variety of names, such as rewriting an executable file, and binary rewriting. Tools that perform executable editing are known as *binary tools or executable editors*. These terms will be used interchangeably during the rest of this article.

As an example, ATOM (39) can be considered as one of the first executable editing frameworks. The program representation used by ATOM was based on OM (40)—a link-time code optimization framework. OM modeled a program as a sequence of procedures. Each procedure consisted of a sequence of basic blocks, and each basic block, in turn, consisted of a sequence of instructions. Instructions were modeled using a simple *Register Transfer Level* (RTL) syntax which was machine independent and generic enough for a variety of RISC machines. The custom instrumentation routines were combined with OM to build custom instrumenters. During linking, the object modules and static libraries could be supplied to such a custom instrumenter that would first build up a symbolic representation of the whole program through OM. The custom instrumentation routines then interfaced with the symbolic representation to traverse and insert analysis code in the program.

Another instrumentation framework built on ATOM-like philosophy was EEL (41), which also constructed a program representation and provided means for user-defined routines to traverse and modify it. The major difference was that whereas ATOM was a link-time instrumentation system EEL worked on linked binary executables. However, the interfaces of EEL were lower level and more detailed than those of ATOM. The added details provided more control over the instrumentation process while complicating the writing of user routines.

One major innovation of EEL was that it provided ways to retarget the analysis tools for processing binaries of different target machines. This retargeting was done by providing a concise description of the target instruction set that helped EEL tools to parse binary instruction codings and understand the syntax and semantics of different machine instructions, typically jump, branch, and memory access instructions.

Following the footsteps of ATOM and EEL, several binary instrumentation frameworks, such as Etch, BIT, and Vulcan appeared. Etch (42) was designed to tackle Win32 binaries on x86 machines, whereas *Vulcan* has been designed to tackle components available in IA-32, IA-64, and MSIL binaries in Win32 environment.

Vulcan (10) deserves a special mentioning because, to the best of our knowledge, it is the only framework that not only permits both static and dynamic instrumentation/modification of binaries on a single machine, but also provides tools for dynamic modification of binaries that run on remote machines. It is a framework that has been specially designed to modify/analyze/instrument applications whose components might be distributed over several machines in a networked environment. For such applications, it is important that tasks like code generation or optimization are delayed until execution time, which is what Vulcan facilitates.

With the increasing popularity of languages like Java and C# and platforms like .NET, which support interoperable software development by compiling source code to some kind of intermediate bytecode, the number of frameworks targeting some form of bytecode instrumentation and manipulation has been also increasing. One of the first Java bytecode instrumentation frameworks is BIT (43). Two other important Java bytecode manipulation frameworks are JOIE (44) and BCEL (45) (BCEL is now part of the Apache project). All of them provide APIs for manipulation of Java bytecodes. Other similar instrumentation frameworks include SERP (46) and Javassist (47).

Several frameworks also exist for instrumentation of .NET bytecode. One of them is AbstractIL (48) from Microsoft, which provides methods to access .NET binaries in a convenient format which can be analyzed, transformed or instrumented. RAIL (49) is another framework, which claims to be a general purpose instrumentation library for .NET platform.

DYNAMIC INSTRUMENTATION

Historically, static instrumentation techniques appeared first; so far, they have been mainly used in program analysis, debugging, testing, and code-coverage tools. However, they suffer from some major limitations, which are sketched below:

1. Static instrumentation techniques modify the software executable. Running an extra tool for instrumentation before executing the code is often not convenient for programmers.
2. Static instrumentation can only cover code that is statically linked. It is not possible to track execution of dynamically linked code, shared libraries, or dynamically generated code through static instrumentation. This issue is the most severe with static instrumentation, because most executables nowadays contain a large section of code in dynamically linked libraries.
3. Static binary instrumentation is often difficult for binary formats that allow mixing of code and data, rather than separating them in well-defined sections (e.g., Win32 binaries).

These limitations have forced the development of dynamic instrumenters that perform instrumentation at runtime (i.e., when the program is executing). Unlike static instrumenters, dynamic instrumenters can only work on compiled executable binaries (or, bytecodes). Therefore, dynamic instrumentation is also called *Dynamic Binary Instrumentation* (DBI). Unlike static instrumenters that expect the instrumented code to run directly on the host machine, DBI tools insert themselves between the original application and the host machine to supervise the execution closely (Fig. 3 highlights this difference between DBI and static instrumentation). Such close monitoring can be performed in various ways. For example, some dynamic instrumenters, such as Pin (50) and DynInst (51) use UNIX/LINUX ptrace (ptrace is part of the Linux kernel (52)) command to obtain control of an application. ptrace allows the dynamic instrumenter to monitor the execution of each instruction, and accordingly add instrumentation code. Another method for dynamic instrumentation is to use the so-called library interposers (53). For instance, in UNIX systems, the LDPRELOAD environment variable forces the dynamic loader to load a shared library in an application's address space. This shared library can contain

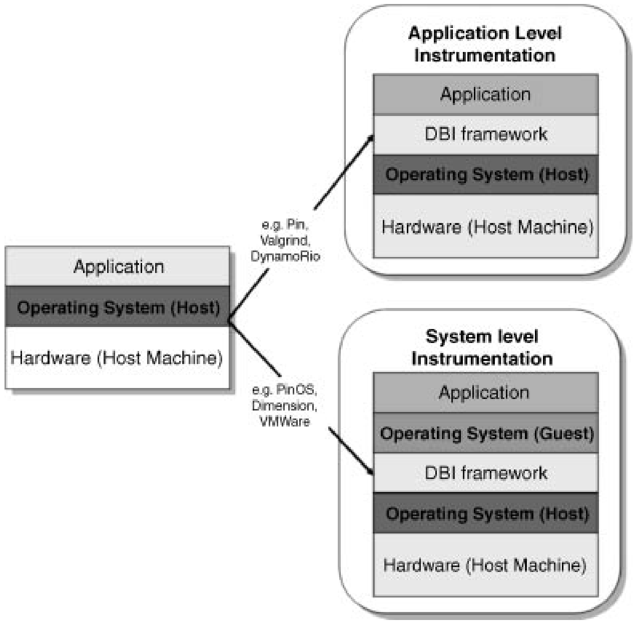


Figure 3. Overview of dynamic instrumentation techniques.

functions to monitor execution of the application and perform instrumentation. One user of this technique is DynamoRio (54).

The most important advantage of dynamic instrumentation is that any piece of code executed can be tracked and analyzed. Through dynamic techniques it is possible to instrument not only statically linked libraries and object files, but dynamically generated or linked code and shared libraries, too. Additionally, dynamic instrumenters do not need to parse through a program binary to detect code and data segments and build a complete program representation before instrumentation. Moreover, indirect branches are especially difficult to handle through static instrumentation. As the targets of indirect branches cannot be resolved at compile time, a static instrumenter must conservatively assume that each instruction is a potential branch target. Therefore, each instruction has to be instrumented, even for simple basic block counting. Through dynamic instrumentation, such cases can be easily handled. From a programmer's perspective, dynamic instrumentation is extremely convenient for debugging, because no extra pass for instrumentation is necessary.

Although dynamic instrumentation is advantageous in many ways, it is not without its limitations. The overall execution time of an application can be affected greatly if dynamic instrumentation is used on it in runtime. Therefore, both static and dynamic instrumentation frameworks suffer from intrusiveness, as they alter the timing of software execution. Additionally, dynamic instrumenters are rather difficult to implement compared with static ones.

Unlike static instrumenters, DBI frameworks cannot be categorized on the basis of the step (in the compilation process) where instrumentation code is added. Instead, the DBI frameworks are classified based on the *techniques* used for instrumentation. Figure 3 shows a possible classification of DBI tools based on those *techniques*. DBI techniques can be applied on application level for the use of profiling, performance measurement, and memory error detection. Several DBI tools, such as Pin (50), Valgrind (13) and DynamoRio (54), are used mostly (but not exclusively) for this purpose. Some other DBI tools can be applied on the system level. Examples of such tools are VMWare (21), Dimension (55), and PinOS (23). These tools mostly exploit instrumentation for running a virtual guest OS on top of a host OS and can even keep track of the OS kernel and its performance characteristics. Such a technique is typically referred to as virtualization and is discussed in depth in Ref. 24.

Based on the underlying techniques of instrumentation, all DBI tools can be classified under two major categories. The first one is called *probe-based* DBI. In this approach, code snippets, which are known as *trampolines*, are inserted in an application during its execution. Examples for such probe-based instrumentation frameworks are DynInst (51), DTrace (56), and Vulcan (10). The second technique is known as *dynamic binary transformation*. This technique can be subdivided into three main approaches—namely, *copy* & *annotate, copy* & *modify* and *Just-In-Time*(JIT)*-compilation* based approaches.

The *copy* & *annotate* approach copies code that is about to be executed, inserts the instrumentation code into this copy, and executes this annotated code. Example frameworks that use this techniques include PIN (50), Nirvana (57), and Dimension (55). A very similar binary transformation approach is *copy* & *modify*. However, the use cases of these two techniques differ significantly. Whereas *copy* & *annotate* tools are mostly used for profiling, error detection and debugging, *copy* & *modify* tools are mostly used for virtualization [VMWare (21) and Virtual PC (22)].

The second binary transformation approach is known as *Just-In-Time-compilation* based approach [e.g., Valgrind (58)]. Such DBI frameworks intercept a program element [usually a basic block (1)] that is about to be executed, and recompiles the application code JIT to insert instrumentation code. All binary transformation techniques are typically considered to be costly in terms of execution time. Therefore, these frameworks incorporate caching strategies to improve performance of DBI engines.

In the following subsections, the focus will be on dynamic instrumentation techniques and not on the instrumentation tools themselves. For example, the dynamic instrumentation tools Memcheck (13) and Cachegrind (13) target different problems in software development, nevertheless both are based on the underlying Valgrind (13) framework. So Valgrind, rather than Memcheck or Cachegrind, will be discussed in detail.

Probe-Based Approaches

Probe-based approaches dynamically analyze executing applications to identify *instrumentation points* that are, typically, either function boundaries, or the start/end of basic blocks. The instrumentation framework then inserts *probes* into the executing code at suitable instrumentation points. A probe usually overwrites the instruction at the corresponding instrumentation point, and adds a jump to a code segment usually called a *trampoline*. The overwritten instruction is usually relocated inside the trampoline. The

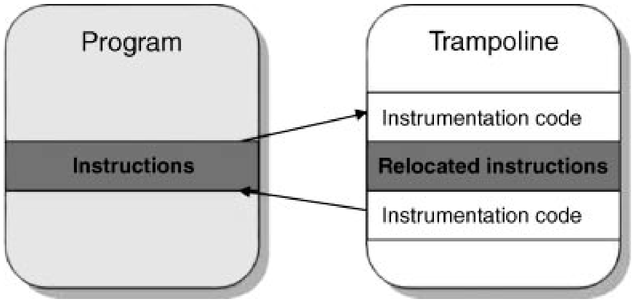


Figure 4. Probe-based instrumentation.

trampoline also contains additional instrumentation code that might increment a counter, a timer or do something else depending on the instrumentation objective. Figure 4 depicts the principle of probe-based dynamic instrumentation. It is to be noted that trampolines themselves can be complex and make calls to other trampolines. For the sake of simplicity, this information is omitted from the figure.

Examples of well-known probe-based instrumentation frameworks are DTrace (56), ParaDyn (59), which is based on DynInst (51), as well as Vulcan (10). Despite several advantages, probe-based instrumentation frameworks have been mostly replaced over time by binary transformation techniques because of the following disadvantages.

1. Trampolines are implemented using one or more levels of branches. This can have significant performance penalties.
2. In probe-based instrumentation, the instructions from the original code are overwritten and/or relocated by the jumps to the trampolines. Adding such jump instructions requires relocation of all other instructions after the overwritten instruction. Additionally, all jump targets, whether direct or indirect, have to be updated as the intended jump target might have been relocated. Such update can only be handled by a dynamic relinking and update step, which is, in principle, feasible, but rather difficult to implement in practice. Consequently, the execution of such code loses transparency, and the implementation task becomes extremely challenging.

Binary Transformation

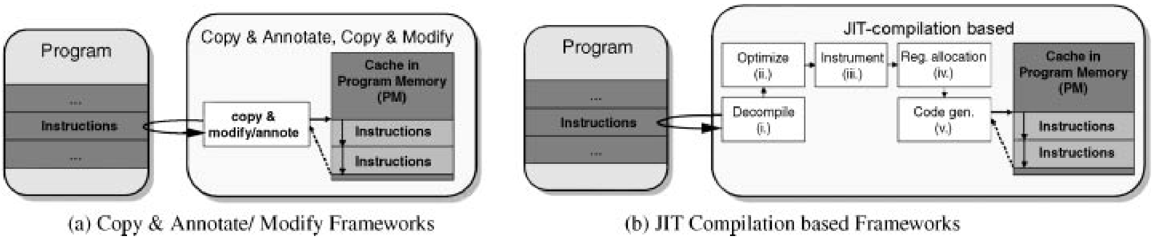
Figure 5 illustrates the common philosophy behind the binary transformation frameworks. As can be observed from the figure, in contrast to the probe-based frameworks, which insert instrumentation code in the original program text. A binary transformation tool copies the original code and generates a completely new set of instructions from it. The new piece of code may contain instrumentation code added inside it. The behavior of the original code, however, is preserved as far as possible. Usually, the copied code is put into a software cache inside the program memory for future execution to the same section of the application.

Binary transformation frameworks/tools such as the Pin (23, 50) and VMW are (21) make use of the *copy* & *annotate* / *modify* approach, whereas Valgrind (58) relies on the JIT-compilation based approach. *Copy* & *annotate* / *modify* frameworks are typically, but not necessarily, used for performance measurements and error detection. JIT-compilation based approaches are rather often used for more fine-grained software instrumentation.

Advantages of binary transformation compared to probe-based approaches are:

1. The original application is never executed in such frameworks and no code modifications are applied to the original application's code. Therefore separation of the instrumented and original code can be done safely (e.g., Pin separates both into two different threads). Moreover, there is no need for code relocation which makes the implementation simpler.
2. Optimized caching strategies of the copied code can improve performance of the instrumentation framework. Even jumps and branches can be removed dynamically by flattening the cached application's instructions.
3. Instrumentation can be incorporated at the very start of the application's code and not only after calling the entry point (usually the main function) of the application. For example, Valgrind can even monitor the loading process of an application and accordingly detect errors there.
4. Self-modifying and referential code can be better supported through binary transformation. For example, Valgrind supports the Ada programing language, which particularly uses self-modifying code (58).

Copy and Annotate/Modify Frameworks. The *copy* & *annotate* and *copy* & *modify* approaches both come under the category of binary transformation frameworks and are closely related from technical point of view. Whereas *copy* &



(a) Copy & Annotate/ Modify Frameworks (b) JIT Compilation based Frameworks

Figure 5. Binary transformation frameworks.

annotate frameworks, like Pin, focus on performance measurements as well as on error detection, the main purpose of *copy* & *modify* frameworks lies on virtualization, like VMWare (21) and VirtualPC (22).

Figure 5(a) depicts the basic principle of such DBI frameworks. The first step is the fetching of a piece of code from the application's text (which is characteristic of all binary transformation approaches), and a transformation process that directly inserts extra code into original text according to the objectives of instrumentation. The transformation process is typically very simple within *copy* & *annotate*/*modify* tools—it does not contain any transformations to and from an intermediate representation (which is characteristic of the JIT-compilation based tools). The resulting instructions are then executed on the host machine, and the procedure is repeated from the beginning.

JIT-Compilation-Based Frameworks. Just-in-time compiled DBI frameworks are more suitable for fine-grained instrumentation of executables. The key idea of such frameworks is to decompile and then to re-compile the binary code dynamically. Examples for such JIT-based instrumentation frameworks are Valgrind (58), Strata (60), DynamoRio (54), and Diota (61).

To understand the philosophy of JIT-based instrumentation frameworks, Valgrind can be considered as an example. Valgrind is an open-source framework that is used in many large projects [e.g. the Linux Desktop Environment KDE (62) and Firefox (63)]. Valgrind is the underlying infrastructure or core of a collection of assorted tools. Other softwares, like Memcheck (13) or Cachegrind (13), can build on this core framework to accomplish specific tasks by performing appropriate types of instrumentation.

Like other binary transformation frameworks, Valgrind first copies a set of instructions from the executable binary. The next step is to generate a new set of instructions that contain instrumentation code. Instead of directly inserting instrumentation code in the copied instructions as within the *copy* & *annotate*/*modify* technique, Valgrind follows a complex process as outlined below and in Fig. 5(b):

1. *Decompilation.* The IA-32 binary code of the application is dynamically disassembled into an intermediate representation named UCode.
2. *Optimization.* In this step, redundancy introduced by the previous phase is removed to improve execution of the instrumented code.
3. *Instrumentation.* Appropriate instrumentation code can be inserted by a tool in this step. This exactly is the phase when tools like Memcheck or Cachegrind interact with the Valgrind core framework to instrument the intermediate UCode.
4. *Register allocation.* Here registers are allocated for the virtual registers of the intermediate UCode.
5. *Code generation.* Finally, instrumented IA-32 code is generated from the register-allocated UCode. The generated code is then executed.

As is clear from this example, insertion of instrumentation by a JIT-compilation based approach introduces a lot of execution overhead. Consequently, most JIT-based tools are much slower than their *copy* & *annotate*/*modify* counterparts, which use direct techniques for instrumentation. For example, in Valgrind, the slowdown factor ranges between 20 and 50 (13) times of the original execution depending on the instrumentation used. This result is 4 times slower than Pin and 4.4 times slower than DynamoRio (58). Another problem of JIT based DBIs is that the recompilation process completely erases all links to the original code.

The advantage of JIT-based approaches is that they can go beyond performance analysis and simple OS virtualization. For example, JIT approaches can be used to emulate a completely different ISA on the host machine. This approach will be difficult to accomplish through *copy* & *annotate*/*modify* DBI frameworks, however, it is possible in principle.

SUMMARY

Software instrumentation is a technique that is widely applied in program analysis, debugging, performance optimization, and virtualization. The instrumentation techniques can be classified into static and dynamic software instrumentation. Historically, static approaches appeared first. They have been, and still are, used in the fields of performance analysis, error detection, and software quality assurance and testing. Dynamic approaches can be used for all of these, but additionally, they are used in several virtualization environments.

Static instrumentation frameworks/tools can be classified into five groups according to the compilation step in which instrumentation code is added. These are source-to-source instrumentation, instrumentation during compilation, instrumentation of the assembly, instrumentation of the object code before linking and finally, link-time and post-link instrumentation. Static instrumentation techniques suffer from the problem that dynamically generated or linked code cannot be tracked using them. Moreover, static instrumentation often involves an extra pass over the application, which adds instrumentation code. This code is often inconvenient for many programmers. Despite of these limitations, static approaches are often simple to implement and still find widespread usage in many profiling, program analysis, error detection, and testing tools.

In contrast to static instrumentation, which has to be applied before execution of the application, techniques based on dynamic instrumentation can be applied at runtime. They can track any piece of executed code, which includes shared and dynamically linked libraries and dynamically generated code. Moreover, they can be used directly on a compiled binary without any extra pass through a special tool. Two major approaches are used for dynamic instrumentation: probe-based approaches and binary transformation. Binary transformation approaches can be subclassified into three different techniques, which are copy & annotate, copy & modify, and JIT-compilation based. Copy & annotate and JIT-compilation based-approaches target primarily performance measurement and error detection, whereas copy & modify frameworks and tools are targeting the field of virtualization.

BIBLIOGRAPHY

1. S. S. Muchnick, *Advanced compiler design and implementation*, Morgan Kaufmann, San Francisco, CA: 1997.
2. S. L. Graham, P. B. Kessler, and M. K. McKusick, An execution profiler for modular programs. *Softw., Pract. Exper.*, **13** (8): 671–685, 1983.
3. MIPS Computer Systems, *RISCompiler and C Programmer's Guide*, 1989.
4. J. R. Larus and T. Ball, Rewriting executable files to measure program behavior, *Softw., Pract. Exper.*, **24** (2) 197–218, 1994.
5. S. J. Eggers, D. R. Keppel, E. J. Koldinger, and H. M. Levy, Techniques for efficient inline tracing on a shared-memory multiprocessor. In *SIGMETRICS '90: Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, New York, ACM Press, 1990.
6. Intel Threading Profiler Tools, Available: http://www.intel.com/support/performancetools/threadprofiler.
7. GNU Compiler Collection (GCC), Available: http://gcc.gnu.org/.
8. Microsoft Visual C++ Development Center, Available: http://msdn.microsoft.com/visualc/.
9. Intel Compiler, Available: http://www.intel.com/cd/software/products/asmona/eng/compilers/284132.htm.
10. A. Edwards, A. Srivastava, and H. Vo. Vulcan, Binary transformation in a distributed environment, *Microsoft Corporation, Technical Report MSR-TR-2001-50,* April 2001.
11. J. Chen, H. Lu, P. Yew, and W. Hsu, Design and implementation of a lightweight dynamic optimization system, 2004.
12. R. Hastings and B. Joyce, Purify: Easy detection of memory leaks and access errors. In *Proc. Winter Usenix Conference*, pp. 1–12, 1992.
13. N. Nethercote, *Dynamic Binary Analysis and Instrumentation*. PhD thesis, University of Cambridge, United Kingdom, November 2004.
14. Parasoft Insure++, Available: http://www.parasoft.com/.
15. Intel Threading Checker Tools, Available: http://www.intel.com/support/performancetools/threadchecker.
16. Visual CSharp Developer Center, Available: http://msdn.microsoft.com/vcsharp/.
17. Java Programming Language, Available: http://java.sun.com/.
18. Python Programming Language, Available: http://www.python.org/.
19. TCL Developer Xchange. http://www.tcl.tk/.
20. A. Nohl, G. Braun, A. Hoffmann, O. Schliebusch, H. Meyr, and R. Leupers, A Universal technique for fast and flexible instruction-set architecture simulation. *Proc. of the Design Automation Conference (DAC)*, New Orleans, June 2002.
21. VMWare Inc., Available: http://www.vmware.com/.
22. Microsoft VirtualPC, Available: http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx.
23. P. P. Bungale and C.-K. Luk, Pinos: A programmable framework for whole-system dynamic instrumentation. *VEE '07: Proc. of the 3rd international conference on Virtual execution environments*, New York, 2007, pp. 137–147.
24. J. E. Smith and R. Nair, *Virtual machines: versatile platforms for systems and processes*. 2005.
25. L. Gao, S. Kraemer, R. Leupers, G. Ascheid, and H. Meyr, A fast and generic hybrid simulation approach using C virtual machine. In *CASES '07: Compilers, Architecture and Synthesis for Embedded Systems*, 2007.
26. T. Kempf, K. Karuri, S. Wallentowitz, G. Ascheid, R. Leupers, and H. Meyr, A SW performance estimation framework for early System-Level-Design using fine-grained instrumentation. *Proc. of the Conference on Design, automation and test in Europe (DATE)*, 2006, pp. 468–473.
27. T. Kempf, M. Dorper, R. Leupers, G. Ascheid, and H. Meyr, T. Kogel, and B. Vanthournout, A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC platforms. *Proc. of the Conference on Design, Automation & Test in Europe (DATE)*, Munich, Germany, March 2005.
28. IBM Rational PurifyPlus (includes Quantify), Available: http://www.ibm.com/software/awdtools/purifyplus/.
29. Gcov Using the GNU Compiler Collection (GCC), Available: http://gcc.gnu.org/onlinedocs/gcc/gcov.html.
30. .NET Framework Developer Center, Available: http://msdn2.microsoft.com/en-us/netframework/default.aspx.
31. Common Language Runtime Overview, Available: http://msdn2.microsoft.com/en-us/library/ddk909ch(vs.71).aspx.
32. M. Kumar, Measuring parallelism in computation-intensive scientific/engineering applications, *IEEE Trans. Comput.*, **37** (9): 1088–1098, 1988.
33. M. Ravasi and M. Mattavelli, High-level algorithmic complexity evaluation for system design, *J. Sys. Architecture: the EUROMICRO J.*, **48** (13–15): 403–427, 2003.
34. G. C. Necula, J. Condit, M. Harren, S. McPeak, and W. Weimer, Ccured: Type-safe retrofitting of legacy software, *ACM Tran. Prog. Languages Sys. (TOPLAS)*, **27** (3): 477–526, 2005.
35. B. P. Miller, M. Clark, J. K. Hollingsworth, S. Kierstead, S.-S. Lim, and T. Torzewski, Ips-2: The second generation of a parallel program measurement system, *IEEE Tran. Parallel Distributed Sys.*, **1** (2): 206–217, 1990.
36. A. Srivastava and D. W. Wall, Link-time optimization of address calculation on a 64-bit architecture, *PLDI '94: Proc. of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, New York, 1994, pp. 49–60.
37. D. W. Wall, Systems for late code modification, in RobertGiegerich and Susan L.Graham, ed., *Code Generation - Concepts, Tools, Techniques*, New York: Springer-Verlag, 1992, p. 275–293.
38. JRat the Java Runtime Analysis Toolkit, Available: http://jrat.sourceforge.net/.
39. A. Srivastava and A. Eustace, Atom: A system for building customized program analysis tools, *Proc. of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, **39** (4): 528–539, 2004.
40. A. Srivastava and D. W. Wall, A practical system for intermodule code optimization at link-time, *J. Prog. Languages*, **1** (1): 1–18, 1992.
41. J. R. Larus and E. Schnarr, Eel: machine-independent executable editing, *PLDI '95: Proc. of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation*, New York, 1995, pp. 291–300.
42. T. Romer, G. Voelker, D. Lee, A. Wolman, W. Wong, H. Levy, B. N. Bershad, and J. B. Chen, Instrumentation and optimization of Win32/Intel executables using etch, in *USENIX Windows NT Workshop*, pp. 1–8, Aug. 1997.
43. H. B. Lee and B. G. Zorn, BIT: A tool for instrumenting java bytecodes, in *USENIX Symposium on Internet Technologies and Systems*, 1997.

44. G. A. Cohen, J. S. Chase, and D. L. Kaminsky, Automatic program transformation with joie, *ATEC'98: Proc. of the Annual Technical Conference on USENIX Annual Technical Conference, 1998*, Berkeley, CA, 1998, p. 14, USENIX Association.

45. Byte Code Engineering Library (BCEL), Available: http://jakarta.apache.org/bcel/.

46. SERP Bytecode Manipulation Framework, Available: http://serp.sourceforge.net/.

47. S. Chiba, Load-time structural reflection in Java, *Lecture Notes in Computer Science*, 1850, 2000.

48. D. Syme, A fast and generic hybrid simulation approach using C virtual machine, in *Microsoft Research*, Cambridge, September 2001, Available: http://research.microsoft.com/projects/ilx/.

49. B. Cabral, P. Marques, and L. Silva, Rail: Code instrumentation for .net. *Proc. OOPSLA '04: Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, New York, 2004, pp. 210–211.

50. C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, Pin: building customized program analysis tools with dynamic instrumentation. *PLDI '05: Proc. of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, 2005, pp. 190–200.

51. J. K. Hollingsworth, B. P. Miller, and J. Cargille, Dynamic program instrumentation for scalable performance tools, May 1994.

52. Linux Kernel, Available: http://www.kernel.org/.

53. Debugging and Performance Tuning with Library Interposers, Available: http://developers.sun.com/solaris/articles/lib_interposers.html.

54. G. T. Sullivan, D. L. Bruening, I. Baron, T. Garnett, and S. Amarasinghe, Dynamic native optimization of interpreters, *IVME '03: Proc. of the 2003 Workshop on Interpreters, Virtual Machines and Emulators*, New York, 2003, pp. 50–58.

55. J. Yang, S. Zhou, and M. L. Soffa, Dimension: An instrumentation tool for virtual execution environments. *VEE '06: Proc. of the 2nd international conference on Virtual execution environments*, New York, 2006, pp. 164–174.

56. B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal, Dynamic instrumentation of production systems. *ATEC'04: Proc. of the USENIX Annual Technical Conference 2004 on USENIX Annual Technical Conference*, Berkeley, 2004, p. 2.

57. S. Bhansali, W.-K. Chen, S. de Jong, A. Edwards, R. Murray, M. Drinic, D. Mihocka, and J. Chau, Framework for instruction-level tracing and analysis of program executions. *VEE '06: Proc. of the 2nd international conference on Virtual execution environments*, New York, 2006, pp. 154—163.

58. N. Nethercote and J. Seward, Valgrind: A framework for heavyweight dynamic binary instrumentation. *PLDI '07: Proc. of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, New York, 2007, pp. 89–100.

59. B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, The paradyn parallel performance measurement tool, *Computer*, **28** (11): 37–46, 1995.

60. K. Scott, N. Kumar, S. Velusamy, B. Childers, J. W. Davidson, and M. L. Soffa, Retargetable and reconfigurable software dynamic translation, *CGO '03: Proc. of the international symposium on Code generation and optimization*, Washington, D.C., 2003, pp. 36–47.

61. J. Maebe, M. Ronsse, and K. De Bosschere, DIOTA: Dynamic instrumentation, optimization and transformation of applications, *Proc. of the International Conference on Parallel Architectures and Compilation Techniques 2002 (PACT '02)*, September 2002.

62. K Desktop Environment (KDE), Available: http://www.kde.org/

63. Firefox, Available: http://www.firefox.org/.

TORSTEN KEMPF
KINGSHUK KARURI
LEI GAO
Institute for Integrated Signal Processing Systems
RWTH Aachen University
Aachen, Germany

SOFTWARE MODULE RISK ANALYSIS

INTRODUCTION

The practical goal of a software development team is to deliver the product within the allotted time and budget. The team strives to achieve the best possible software quality within the given resources. Assessing software module risk is generally a precursor to a software quality enhancement initiative. To improve software quality, development teams apply various techniques such as reengineering, extra reviews, and additional testing. Analyzing the risk associated with modules for a given system often involves *software quality estimation* and then applying quality improvement techniques to the modules that need it the most.

In software engineering practice, assessing the risk of a software module is often attained by either classifying it into risk-based classes, such as *fault-prone* (*fp*) or *not fault-prone* (*nfp*), or by predicting the number of *software faults* expected. In addition, a software module's risk can be assessed by obtaining a relative risk-based ordering of all software modules. The software attribute that represents software quality, i.e., risk-based class, number of faults, or another factor, is referred to as *software quality data*. Although other software quality models exist, we focus primarily on software quality classification, software fault prediction, and software module-order models. These models are more widely used in software engineering practice.

Software measurements play a vital role in developing software quality estimation models (1), because of the software engineering assumption that software metrics reflect the quality of a software product. A software metric is an attribute that quantifies or qualifies a certain aspect of the software module or program. For example, "lines of code" can represent the size of a program. Several types of software metrics exist in the literature (2) ranging from very basic attributes to more complex attributes. The use of specific software metrics for software quality estimation depends on the system under evaluation and other practical considerations.

The output of software quality estimation models are predictions that the development team can use to identify high-risk or low-quality modules in the software system. The need for identifying such modules is primarily from economic and practical needs. An ideal software development situation would be that all modules are targeted for inspection and quality improvement to maximize software reliability. However, in practice, a software development team generally has limited and finite resources allocated for quality improvement. Software quality estimation models provide practical assistance to the development team by isolating high-risk software modules for a targeted and cost-effective use of project resources.

We present three types of software quality estimation models, i.e., software quality classification, software fault prediction, and module-order modeling, all built using a case study of software measurement data from a large telecommunications system. A software quality classification model predicts the class membership of modules into predefined quality-based classes (3). A software fault prediction model predicts the number of faults expected in the modules (4). A module-order model predicts the relative risk-based ordering of the software modules (5). Our aim is to provide the reader with sufficient information of research on software module risk analysis, so as to promote further software quality and/or interdisciplinary research.

SOFTWARE QUALITY ESTIMATION MODELS

Software development is a human-intensive endeavor, and software quality is invariably affected by many factors that vary among development organizations. To achieve useful accuracy, software quality models must be built for each specific development environment. In addition, each software project team must decide on which software measurements are to be collected and recorded for software quality estimation. Software measurements have been used as quality predictors in software development. However, there is yet no consensus as to what software metrics are preferable for quality estimation.

Most literature on software metrics is typically aimed at demonstrating the efficacy of individual metrics. However, this does not directly relate to building useful software quality models. Our previous experience with software measurement data from industrial projects has indicated that a software quality model based on only one software metric does not have useful accuracy and robustness. A simple metric, such as lines of code, is not sufficient by itself. A more complex metric, such as McCabe's cyclomatic complexity (2), is also not enough by itself. A better approach is to employ multiple software metrics to build a software quality model instead of only one metric.

A given software development organization is often equipped with data collection tools, such as a software metric-analyzer tool. Hence, the relative cost of collecting many software metrics, instead of just a few, is not a practical problem. To determine software metrics that are better predictors of quality for a given system, a data mining approach should be taken instead of an arbitrary- or heuristic-based selection approach. Pragmatic considerations usually determine the set of available software metrics. We do not advocate the universal use of a given set of software metrics; instead, we prefer a data mining approach to selecting software metrics that are good quality predictors for the given software system.

To build a useful software quality estimation model, the following modeling steps are followed:

- Analysis of a previous system release or similar project, for which software quality data is known. Software measurement data are collected for this project.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

- The known software metrics and software quality data are used to build a software quality model. The model is then used to predict the quality of modules in the current project, for which software quality data are not known. We shall focus our discussion to three types software quality estimation models: software fault prediction, software quality classification, and software module-order modeling.
- The obtained predictions are used as a guide to allow a targeted software quality improvement initiative.

A software fault prediction model tries to find the relationship between the given set of software measurements of the modules and the number of faults expected in each module. A software fault is a defect in an executable product that causes a software failure. Faults are a result of mistakes or omissions made by the developers. A software quality classification model tries to find the relationship between the given set of software measurements of the modules and their membership into predetermined risk-based classes, such as *fp* and *nfp*. An advantage of a fault prediction model is that it allows the development team to observe the quality of modules in a relative sense. However, a development team may only be interested in knowing which modules should be targeted for quality improvement, regardless of their relative quality—a classification model provides such a software quality model.

A classification model, however, does not provide the relative risk of the modules classified as *fp*, which makes it difficult to initiate software quality improvement toward the most high-risk modules. In addition, a fault prediction model may predict well for low-risk modules but predict poorly for high-risk modules. These disadvantages are overcome by a module-order model, in which the relative risk-based order of the software modules is predicted. More specifically, the output of such a model is a ranking of the modules according to a risk factor, such as number of faults expected. Such a model is attractive to the software quality assurance team because it provides valuable guidance for quality improvement without quantifying the quality of individual program modules. To obtain a cost-effective software quality improvement, the available resources can be applied to the modules starting from the most-risky and selecting additional modules according to the quality-based ranking of program modules until the available resources are exhausted.

A given software development team may choose to select any one of the three software quality models based on their organizational preference, expertise with the modeling approach, and available resources for carrying out the software quality modeling and analysis process. The performance of software quality models is often measured by their prediction accuracy. The prediction of a software quality classification model is often measured by its misclassification error rates—the lower the error rates, the better the model. In a two-group (*fp* and *nfp*) classification model, two types of errors can occur, Type I and Type II. A Type I error occurs when a *nfp* module is predicted as *fp*, whereas a Type II error occurs when a *fp* module is predicted as *nfp*. A Type II error is more costly than a Type I error, because it implies a lost opportunity to fix a fault before system operations. In contrast, a Type I error implies unnecessary inspection of a good quality module. In our studies (6), we have observed an inverse relationship between the Type I and Type II error rates—as one increases, the other decreases. Hence, the selection of the final model requires the knowledge of a preferred balance between the two error rates.

The prediction of a fault prediction model can be measured by the average absolute error (AAE) or the average relative error (ARE). The relative importance of these two performance measures is out of scope for this article and is an open research issue. The AAE and ARE performance measures are computed as follows:

$$\mathrm{AAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{1}$$

$$\mathrm{ARE} = \frac{1}{n}\sum_{i=1}^{n} \left|\frac{y_i - \hat{y}_i}{y_i + 1}\right| \tag{2}$$

where y_i is the actual number of faults in a module, $\hat{y}_i$ is the predicted number of faults in a module, and n is the number of modules in the target dataset. The denominator in ARE has a one added to avoid division by zero, because a software module could have zero faults associated with it. Our studies evaluate fault prediction models using both AAE and ARE.

In our study, the prediction of a module-order model is measured by evaluating, for a given cutoff percentage of modules, how accurately the model accounts for the actual number of faults for that cutoff point. A cutoff point reflects the percentage of modules that the software development team will choose to apply software quality improvement techniques. Subsequently, different projects will choose different cutoff points according to their quality improvement requirements. The precise evaluation procedure for a module-order model is discussed in more detail in the next section.

MODELING TECHNIQUES

In the literature one can find various techniques and methods for building software quality estimation models. Although most techniques are suited either for quality classification or fault prediction, not many can address both problems. Some techniques used for software quality classification include discriminant analysis (7), logistic regression (8,9), decision trees (6,10), artificial neural networks (11,12), genetic programming (13), belief networks (14), fuzzy logic (10), and case-based reasoning (15). Some techniques used for software fault prediction include multiple linear regression (16), artificial neural networks (16), case-based reasoning (17), and regression trees (18).

In this study, we present the logistic regression technique for building software quality classification models and the multiple linear regression technique for software fault prediction. These methods are commonly used in classification and regression problems, especially in the software quality engineering field. A good comparative study that compares seven software quality classification methods is presented in Ref. (3). A similar comparative study that compares five software fault prediction methods is presented in Ref. (16).

Logistic Regression

Logistic regression is a statistical modeling technique in which the dependent variable has only two possible values and the independent variables may be categorical, discrete, or continuous (8). In the context of classification with logistic regression, we designate a module being *fp* as an "event" (19). Therefore, if p is the probability of an event, i.e., a module is *fp*, $(\frac{p}{1-p})$ is the odds of an event. We denote x_j as the jth independent variable. A logistic regression model has the following form:

$$\log_e\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_j x_j + \ldots + \beta_m x_m \quad (3)$$

where m represents the number of independent variables and $\beta_0, \ldots, \beta_m$ are model coefficients.

In software engineering, most software metrics have a monotonic relationship with faults that are inherent in the underlying processes. In practice, not all independent variables may contribute to the logistic regression model. To exclude insignificant variables, the stepwise logistic regression method for *model selection* was adopted (19). The significance tests for each variable are based on the chi-squared statistic. We compute the maximum likelihood estimates of the parameters of the model, b_j, using the iteratively reweighted least-squares algorithm (20), where b_j is the estimated value of β_j. The estimated logistic regression model takes the following form:

$$\log_e\left(\frac{\hat{p}}{1-\hat{p}}\right) = b_0 + b_1 x_1 + \ldots + b_j x_j + \ldots + b_m x_m \quad (4)$$

To classify program modules as either *fp* or *nfp* with the logistic regression model, $(\frac{\hat{p}}{1-\hat{p}})$ is computed using the above equation. A module x_i's class, *Class*($\mathbf{x}_i$), is assigned as *nfp* if $(\frac{1-\hat{p}}{\hat{p}}) > \zeta$ and *fp*, otherwise. The term ζ is a modeling parameter that can be empirically varied to obtain the preferred classification model (19). The desired balance between the error rates is project dependent and is largely influenced by the software quality improvement needs of the development team and the disparity between the respective costs of misclassifications.

Multiple Linear Regression

The multiple linear regression technique provides a statistical means of estimating or predicting a dependent variable as a function of known independent variables. The model is in the form of an equation where the response or dependent variable is expressed in terms of predictors or independent variables. The general form of a multiple linear regression (MLR) model can be given by

$$\hat{y} = a_0 + a_1 x_1 + \ldots + a_m x_m \quad (5)$$

$$y = a_0 + a_1 x_1 + \ldots + a_m x_m + e \quad (6)$$

where $x_1, \ldots, x_m$ are the m independent variables' values, $a_0, \ldots, a_m$ are the parameters to be estimated, $\hat{y}$ is the dependent variable to be predicted, y is the actual value of the dependent variable, and $e = y - \hat{y}$ is the error in prediction.

The data available are initially subject to statistical analysis, with the aim to remove any correlation existing between independent variables and to remove insignificant independent variables, not accounting for the dependent variable. The process of determining the variables that are significant is known as *model selection*. Several methods of model selection exist. They are forward elimination, stepwise selection, and backward elimination (20). Here, stepwise regression is used.

Stepwise regression selects an optimal set of independent variables for the model. In this process, variables are either added or deleted from the regression model at each step of the model building process. Once the model is selected, the parameters $a_0, \ldots, a_m$ are then estimated using the *least-squares* method. The values of the parameters are selected such that they minimize $\sum_{i=1}^{N} e_i^2$, where N is the number of observations in the *fit* dataset.

Module-Order Modeling

A module-order model (MOM) can be defined as a software metrics-based quality estimation model, which is used to predict the prioritized rank-order of modules according to a predetermined software quality factor. The choice of the quality factor is dependent on the project management team; however, it should be a good representation of the actual quality of the module. A good example would be the number of faults (as defined by the project) expected in a software module during system test or operations. A MOM predicts the relative quality of each program module, especially those that are the most faulty.

A MOM comprises the following three components: (*1*) an underlying software quality prediction model; (*2*) a ranking of modules according to the quality factor predicted by the underlying model; and (*3*) a procedure for evaluating the accuracy and effectiveness of the predicted ranking. In the context of a MOM, a software metrics-based underlying software quality prediction model may be considered as a function of a vector of software measurements $\mathbf{x}_i$, predicting a quality factor F_i, for module i; i.e., $F_i = f(\mathbf{x}_i)$. Generally speaking, any prediction technique may be selected as an underlying quality prediction model. We use the prediction obtained by the multiple linear regression technique.

When obtaining the quality-based rankings of software modules, the following notations are used. Let $\hat{F}(\mathrm{x}_i)$ be an estimate of F_i by the underlying prediction model $\hat{f}(\mathrm{x}_i)$. R_i is the perfect ranking of the observation i according to F_i, whereas $\hat{R}(\mathrm{x}_i)$ is the same ranking but according to $\hat{F}(\mathrm{x}_i)$. In module-order modeling, the emphasis is on whether a module falls above a certain cutoff percentile that indicates the proportion of modules that are to be targeted for reliability enhancements. All modules that fall above the cutoff percentile will be subjected to quality improvement. According to the allocated software quality improvement resources, the project management team will select a certain cutoff percentile and apply quality enhancement processes to all modules that fall within that cutoff value.

Once the quality-based rankings are determined, the following steps illustrate the evaluation procedure for a module-order model (5). Given a model and a test dataset with software modules indexed by i:

1. Management will choose to enhance modules in a priority-based order, beginning with the most faulty. However, the rank of the last module that will be enhanced is uncertain at the time of modeling. Determine a range of percentiles that covers management's options for the last module (from the rank order), based on the schedule and resources allocated for a software quality improvement. Choose a set of representative cutoff percentiles c from that range.
2. For each cutoff percentage value of interest c, define the number of faults accounted for by the modules above the percentage c. This process is done for both the perfect and the predicted ranking of the modules: $G(c)$ is the number of faults accounted for by the modules that are ranked (perfect ranking) above the percentile c, and $\hat{G}(c)$ is the number of faults accounted for by the modules that are predicted as falling above the percentile c.

$$G(c) = \sum_{i:R \geq c} F_i \tag{7}$$

$$\hat{G}(c) = \sum_{i:\hat{R}(\mathbf{X}_i) \geq c} F_i \tag{8}$$

3. Calculate the percentage of faults accounted for by each ranking, namely, $\frac{G(c)}{G_{tot}}$ and $\frac{\hat{G}(c)}{G_{tot}}$, where G_{tot} is the total number of actual faults of all program modules in the given dataset.
4. Calculate the performance of the module-order model $\phi(c) = \frac{\hat{G}(c)}{G(c)}$, which indicates how closely the faults accounted for by the model ranking match those of the perfect module ranking. In the context of accuracy of a MOM at a given c value, the performance of the model, i.e., $\phi(c)$, should be as close to 1 (or 100%) as possible. After evaluating the accuracy and robustness of a MOM, it is ready for use on a current similar project or subsequent release. Determine the predicted ranking, by ordering modules in the current dataset according to $\hat{F}(\mathbf{x}_i)$.

In practice, a manager is interested in the accuracy of a MOM only at the preferred cutoff percentile value. As all modules that fall above the preferred cutoff point get the same reliability enhancement treatment, the distance of the predicted rank-order from the actual is not an appropriate measure of model accuracy. We are therefore not interested in the accuracy of the rank-order within the enhanced group, which consists of modules that are subjected to quality improvements. However, we do want $\phi(c)$ to be close to 100% for the c of interest.

EMPIRICAL CASE STUDY

System Description

The case study data were collected over two successive releases, from a very large legacy telecommunications system, abbreviated as LLTS. The software system is an embedded-computer application that included finite-state machines. Using the procedural development paradigm, the software was written in a high-level language and was maintained by professional programmers in a large organization. The releases considered in our study are labeled as Release 1 and Release 2 and do not represent the first two chronological releases of the system.

A software module was considered as a set of related source-code files. Faults attributed to a software module were recorded only if their discovery resulted in changes to the source code of the respective module. Software fault data were collected at the module level by the problem reporting system and comprised post-release faults discovered by customers during system operations. A problem reporting system is an information system for managing software faults from initial discovery through distribution of fixes. Preventing the occurrence of software faults after deployment was a high priority for the developers, because visits to customer sites involved extensive consumption of monetary and other resources.

Configuration management data analysis identified software modules that were unchanged from the prior release. A configuration management system is an information system for managing multiple versions of artifacts produced by software development processes. Fault data collected from the problem reporting system was tabulated into problem reports, and anomalies were resolved. Because of the nature of the system being modeled, i.e., a high-assurance system, the modules associated with post-release faults were very few as compared with modules with no faults. Two clusters of modules were identified: unchanged and updated. The updated modules comprised those that were either new or had at least one update to their source code since the prior release. Among the unchanged modules, almost all (over 99%) of them had no faults and, therefore, were not considered for modeling purposes.

We selected updated modules with no missing data in the relevant variables. These updated modules had several million lines of code, with a few thousand of these modules in each system release. The number of updated modules (that remained after unchanged modules or those with missing data were removed) that were considered for the two releases are 3649 for Release 1 and 3981 for Release 2. In the case of the software quality classification study, a module was considered as *nfp* if it had no post-release faults and *fp* otherwise. The proportion of modules with no faults among the updated modules of Release 1 was $\pi_G = 0.937$, and the proportion with at least one fault was $\pi_R = 0.063$. Such a small set of *fp* modules is often difficult for a software quality modeling technique to identify.

The set of available software metrics is usually determined by pragmatic considerations. A data mining approach is preferred in exploiting software metrics

Table 1. LLTS **Software Product Metrics**

Symbol	Description
	Call Graph Metrics
CALUNQ	Number of distinct procedure calls to others.
CAL2	Number of second and following calls to others. *CAL2* = *CAL* − *CALUNQ* where *CAL* is the total number of calls.
	Control Flow Graph Metrics
CNDNOT	Number of arcs that are not conditional arcs.
IFTH	Number of non-loop conditional arcs, i.e., if–then constructs.
LOP	Number of loop constructs.
CNDSPNSM	Total span of branches of conditional arcs. The unit of measure is arcs.
CNDSPNMX	Maximum span of branches of conditional arcs.
CTRNSTMX	Maximum control structure nesting.
KNT	Number of knots. A "knot" in a control flow graph is where arcs cross due to a violation of structured programming principles.
NDSINT	Number of internal nodes (i.e., not an entry, exit, or pending node).
NDSENT	Number of entry nodes.
NDSEXT	Number of exit nodes.
NDSPND	Number of pending nodes, i.e., dead code segments.
LGPATH	Base 2 logarithm of the number of independent paths.
	Statement Metrics
FILINCUQ	Number of distinct include files.
LOC	Number of lines of code.
STMCTL	Number of control statements.
STMDEC	Number of declarative statements.
STMEXE	Number of executable statements.
VARGLBUS	Number of global variables used.
VARSPNSM	Total span of variables.
VARSPNMX	Maximum span of variables.
VARUSDUQ	Number of distinct variables used.
VARUSD2	Number of second and following uses of variables. *VARUSD2* = *VARUSD* − *VARUSDUQ*, where *VARUSD* is the total number of variable uses.

data, by which a broad set of metrics are analyzed rather than limiting data collection according to a predetermined set of research questions. Data collection for this case study involved extracting source code from the configuration management system. The available data collection tools determined the number and selection of the software metrics. Software measurements were recorded using the EMERALD (Enhanced Measurement for Early Risk Assessment of Latent Defects) software metrics analysis tool, which includes software-measurement facilities and software quality models. Another project might collect and consider a different set of software metrics for modeling purposes (21–23).

Preliminary data analysis selected metrics that were appropriate for our modeling purposes. Another software project may consider (depending on availability) a different set of software metrics as more appropriate. Software metrics collected included 24 product metrics, 14 process metrics, and 4 execution metrics. The 14 process metrics were not used in our empirical evaluation, because this study is concerned with the software quality estimation of program modules after the coding (implementation) phase and before system tests. Therefore, the case study consists of 28 independent variables (Tables 1 and 2) that were used to predict the respective dependent variable: *Class* (*fp* or *nfp*) for software quality classification, and *Faults* for software fault prediction. The predicted number of faults are used to rank the modules for building the module-order model.

The software product metrics in Table 1 are based on call graph, control flow graph, and statement metrics. An example of call graph metrics is the number of distinct procedure calls. A module's control flow graph consists of nodes and arcs depicting the flow of control of the program. Statement metrics are measurements of the program statements without implying the meaning or logistics of the statements. The problem reporting system maintained records on past problems. The proportion of installations that had a module, *USAGE*, was approximated by deployment data on a prior system release. Execution times in Table 2 were measured in a laboratory setting with different simulated workloads.

It should be noted that software quality estimation models based on source code metrics that are available

Table 2. LLTS **Software Execution Metrics**

Symbol	Description
USAGE	Deployment percentage of the module.
RESCPU	Execution time (microseconds) of an average transaction on a system serving consumers.
BUSCPU	Execution time (microseconds) of an average transaction on a system serving businesses.
TANCPU	Execution time (microseconds) of an average transaction on a tandem system.

relatively later (such as software product metrics) in the software development process may have some drawbacks. For example, it may be difficult to relate the model with software engineering issues related to the requirements and specifications development phase. In addition, some early software design quality issues may not be completely reflected by the model. The ideal scenario would be to evaluate software quality in a progressive manner as the project develops, using techniques suited for the given development phase. However, limited software project funds often restrict implementing such an ideal scenario.

Software Quality Classification Model

The logistic regression model formed after the stepwise logistic regression procedure for identifying significant independent variables (at an $\alpha = 0.15$ significance level) for the LLTS case study is shown below:

$$\begin{aligned}\log\left(\frac{\hat{p}}{1-\hat{p}}\right) \\ &= -6.0473 + 0.0327\,\mathit{FILINCUQ} + 1.9610\,\mathit{USAGE} \\ &\quad + 0.0230\,\mathit{LGPATH} + 0.0145\,\mathit{LOP} \\ &\quad + 0.0002\,\mathit{VARSPNMX} - 0.0032\,\mathit{STMCTL} \\ &\quad + 0.0079\,\mathit{NDSPND} + 0.0031\,\mathit{IFTH} \qquad (9)\end{aligned}$$

where $\hat{p}$ is the estimated value of p, and the respective software metrics are described in Tables 1 and 2. This model was built using the Release 1 software modules as a training dataset. The modules of Release 2 were used as a test dataset to evaluate the prediction accuracy of the model.

In our case study, we varied the value of the parameter ζ to obtain the preferred balance between the error rates. The misclassification rates for the logistic regression models based on the different values of ζ are presented in Table 3. Other values for ζ were also considered; however, we have presented a representative set in the table. An inverse relationship between the Type I and Type II error rates is observed. A very high value of ζ yielded a very low Type II error and a very high Type I error. On the other hand, a very low value of ζ yielded a very high Type II error and a very low Type I error. For example, when ζ = 50, the corresponding fitted (Release 1) model yielded a Type I error rate of 67% and a Type II error rate of 3.5%. Moreover, when ζ = 1, the corresponding model yielded a Type I error rate of 0.4% and a Type II error rate of 91.7%.

In our previous studies with high assurance systems such as the legacy telecommunication system presented, a preferred balance of equality between the Type I and Type II error rates and Type II being as low as possible was chosen as the model selection criterion. Such a model selection criterion is representative of a software project that has very few *fp* modules in comparison with the *nfp* modules, and when the cost of misclassifying a *fp* module is much greater than the cost of misclassifying a *nfp* module. In addition, such a model selection strategy provides a practical software quality classification. If a model with the lowest Type II error rate is selected as the final model, it will have a very high Type I error rate. This implies that a large number of modules will be predicted as *fp*, with many of them actually being *nfp*. Such a model is not practical for cost-effective software quality improvement.

Therefore, based on our model-selection strategy, the preferred balance was obtained when ζ = 16. We observe that for ζ = 16 the two error rates are approximately equal, with the Type II error rate being low. The performance of this model for the test dataset, i.e., Release 2, is fairly reasonable and is not too overfitted. An overfitted model is one that performs very well on the fit dataset but performs very poorly on the test dataset. We observe that, for Release 2, the model with ζ = 16 maintains the preferred balance between the two error rates reasonably well.

Table 3. Logistic Regression Classification Models

	Release 1		Release 2	
ζ	Type I	Type II	Type I	Type II
50	67.00 %	3.50 %	64.79 %	4.23 %
30	49.80 %	8.70 %	46.70 %	11.64 %
25	42.90 %	13.50 %	39.90 %	14.81 %
20	34.90 %	19.70 %	31.59 %	21.69 %
18	31.00 %	21.00 %	28.11 %	24.87 %
16	**27.00 %**	**24.90 %**	**25.00 %**	**29.60 %**
10	15.50 %	38.90 %	14.61 %	41.80 %
5	6.20 %	60.30 %	6.20 %	67.20 %
1	0.40 %	91.70 %	0.40 %	91.53 %
0.067	0.00 %	100.00 %	0.00 %	100.00 %

Software Fault Prediction Model

The multiple linear regression technique with stepwise regression selected seven software metrics at a significance level of α = 0.05. A significance level represents the statistical degree of importance for the independent variables. The selected metrics are *FILINCUQ*, *CNDNOT*, *NDSENT*, *NDSEXT*, *NDSPND*, *NDSINT*, and *STMDEC*. The model parameters were estimated, and the following model was obtained:

$$\begin{aligned}\mathit{Faults} &= 0.0143\mathit{FILINCUQ} - 0.0035\mathit{CNDNOT} \\ &\quad + 0.0238\mathit{NDSENT} - 0.009\mathit{NDSEXT} \\ &\quad + 0.017\mathit{NDSPND} + 0.0066\mathit{NDSINT} \\ &\quad - 0.0031\mathit{STMDEC}\end{aligned}$$

The values of average absolute and average relative errors obtained from the model are presented in Table 4. The table also shows the standard deviation of the two error measures. We observe that, with respect to AAE, the performance of the model for the test dataset is better than the

Table 4. Fault Prediction Model Performance

Dataset	AAE	SDAE	ARE	SDRE
Release 1	1.007	1.534	0.550	0.545
Release 2	0.890	1.091	0.571	0.610

fit dataset, which is generally not expected. The trained model has an AAE of about 1 fault, with its predictive performance of AAE = 0.890. With respect to the ARE performance measure, the trained model has an error rate of 0.55. The performance of the model for the test dataset is very similar with ARE = 0.571. When considering both performance measures, we note that the multiple linear regression model does not show overfitting tendencies.

A comparison of the software metrics used by the classification and fault prediction model reveals that the *FILINCUQ* and *NDSPND* metrics are common to the two models. This finding may suggest that these two metrics are useful (among others) software quality predictors for the legacy telecommunications system considered in this study. The problem of feature selection or attribute selection is very important in software quality estimation, because it could reduce data collection efforts and improve the robustness of the software quality models.

Software Module-Order Model

The software fault prediction obtained by the multiple linear regression model is used as the underlying prediction model for the module-order model. More specifically, for the given dataset, the modules are ranked according to their predicted number of faults, starting with the highest. The results of the model are summarized in Table 5, which evaluates the model at different cutoff values, i.e., c values. We note that, for Release 1, 100 % of software faults are accounted for at c = 0.50; i.e., $\frac{G(c)}{G_{tot}} = 1.000$. Similarly, for Release 2, 100 % of software faults are accounted for at c = 0.60.

The fourth and seventh columns in the table represent the performance $\phi(c)$ of the module-order model for the Release 1 and Release 2 datasets, respectively. We observe that $\phi(c)$ generally increases with a decrease in c, which is expected because as c is decreased more modules are subjected to inspection and hence more software faults will be accounted for. The performance of the model is generally lower for the test dataset than the fit dataset, which suggests that the module-order model may be prone to some overfitting. However, it should be noted that the ranking of the model is obtained by the predictions of the underlying multiple-linear regression model. This finding suggests that the efficiency of the underlying prediction model is likely to affect the performance of the subsequent module-order model.

Table 5. Module-Order Model Based on Multiple Linear Regression

	Release 1			Release 2		
c	$\frac{G(c)}{G_{tot}}$	$\frac{\hat{G}(c)}{G_{tot}}$	$\phi(c)$	$\frac{G(c)}{G_{tot}}$	$\frac{\hat{G}(c)}{G_{tot}}$	$\phi(c)$
0.95	0.381	0.239	0.627	0.368	0.204	0.554
0.90	0.541	0.361	0.667	0.539	0.330	0.613
0.85	0.645	0.458	0.709	0.671	0.421	0.628
0.80	0.743	0.522	0.703	0.748	0.489	0.654
0.75	0.794	0.577	0.726	0.814	0.539	0.662
0.70	0.843	0.624	0.740	0.880	0.591	0.672
0.65	0.891	0.670	0.752	0.946	0.632	0.668
0.60	0.940	0.706	0.751	1.000	0.672	0.672
0.55	0.988	0.737	0.746	1.000	0.709	0.709
0.50	1.000	0.771	0.771	1.000	0.740	0.740

THREATS TO VALIDITY

Controlled experiments to evaluate the usefulness of empirical models are not practical because of the many human factors that affect software quality. Hence, we adopted a case study approach to demonstrate the usefulness of the software quality estimation models in a real-world setting. To be credible, the software engineering community demands that the subject of an empirical study be a system with the following characteristics (24). The subject of an empirical case study must be developed

- By a group and not by an individual
- By professionals and not by students
- In an industry/government organization and not in a laboratory
- As large as industry projects and not a toy problem

We note that our case study fulfills all of the above criteria through collaborative arrangements with the development organization.

CONCLUSION

The task of delivering a software product with good quality is daunting, especially in the presence of limited budget and time constraints. Software development teams apply various techniques to improve software quality, so as to maximize software quality within the given project resources. Software quality estimation models predict the quality of software modules can be used to provide a targeted software quality improvement initiative.

A software quality classification model predicts the class membership of modules into risk-based classes, such as fault-prone and not fault-prone. A software fault prediction model estimates the number of faults in a software module. A software module-order model predicts the relative risk-based order of the modules. These software quality models are built using software measurements and quality data obtained from a previous system release or similar project. The trained models are then used to predict the software quality of the modules currently under development. These software quality models have successfully been used in software engineering practice toward software quality improvement of real-world projects.

This article presents the useful principles of building software quality classification, software fault prediction, and software module-order models. The logistic regression technique was used to build the classification model, whereas the fault prediction model was built by using the multiple linear regression technique. The module-order model was obtained by ranking the modules according to the predictions of the fault prediction model. A case study of software measurement data obtained from a telecommunications system was used to build the respective models. Several other software systems have also been explored in other related studies, and have demonstrated the effectiveness of software quality estimation models toward cost-effective software quality improvement.

Some modeling tools that can be used for building software quality models include SAS (www.sas.com), S-Plus (www.mathworks.com), CART (www.salford-systems.com), MATLAB (www.mathsoft.com), SMART (www.cse.fau.edu/esel), IBM Intelligent Data Miner (www.ibm.com), and WEKA (25). Not all tools have a wide selection of modeling techniques. For example, CART predominantly implements a decision- and regression-tree-based modeling approach. In contrast, WEKA provides a collection of modeling techniques, such as C4.5 decision tree and instance-based learning.

ACKNOWLEDGMENTS

We express our gratitude to Dr. Kehan Gao for her patient reviews of the manuscript. In addition, we thank the various current and previous members of the Empirical Software Engineering Laboratory, Florida Atlantic University, for their assistance with empirical and modeling analysis.

BIBLIOGRAPHY

1. N. F. Schneidewind, Body of knowledge for software quality measurement, *IEEE Comput.*, **35** (2): 77–83, 2002.
2. N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. Boston, MA: PWS Publishing Company, 1997.
3. T. M. Khoshgoftaar and N. Seliya, Comparative assessment of software quality classification techniques: An empirical case study, *Empirical Softw. Eng. J.*, **9** (3): 229–257, 2004.
4. A. R. Gray and S. G. MacDonell, Software metrics data analysis: Exploring the relative performance of some commonly used modeling techniques, *Empirical Softw. Eng. J.*, **4**: 297–316, 1999.
5. T. M. Khoshgoftaar and E. B. Allen, Ordering fault-prone software modules, *Softw. Quality J.*, **ll** (l): 19–37, 2003.
6. T. M. Khoshgoftaar, X. Yuan, and E. B. Allen, Balancing misclassifica-tion rates in classification tree models of software quality, *Empirical Softw. Eng. J.*, **5**: 313–330, 2000.
7. P. Runeson, M. C. Ohlsson, and C. Wohlin, A. classification scheme for studies on fault-prone components, *Lecture Notes Comput. Sci.*, **2188**: 341–355, 2001.
8. K. El Emam, W. Melo, and J. C. Machado, The prediction of faulty classes using object-oriented design metrics, *J. Syst. Softw.*, **56** (1): 63–75, 2001.
9. N. F. Schneidewind, Investigation of logistic regression as a discriminant of software quality, *Proc. 7th Int. Softw. Metrics Symp.*, London, UK, 2001, pp. 328–337.
10. A. Suarez and J. F. Lutsko, Globally optimal fuzzy decision trees for classification and regression, *Pattern Anal. Mach. Intell.*, **21** (12): 1297–1311, 1999.
11. M. Reformat, W. Pedrycz, and N. J. Pizzi, Software quality analysis with the use of computational intelligence, *Proc. IEEE Int. Conf. Fuzzy Syst.*, Vol. 2, Honolulu, HI, 2002, pp. 1156–1161.
12. Z. Xu and T. M. Khoshgoftaar, Software quality prediction for high assurance network telecommunications systems, *Comput. J.*, **44** (6): 557–568, 2001.
13. T. M. Khoshgoftaar, Y. Liu, and N. Seliya, Genetic programming-based decision trees for software quality classification, *Proc. 15th International Conference on Tools with Artificial Intelligence*, Sacramento, 2003, pp. 374–383.
14. L. Guo, B. Cukic, and H. Singh, Predicting fault prone modules by the dempster-shafer belief networks, *Proc. 18th International Conference on Automated Software Engineering*, Montreal, Quebec, Canada, 2003, pp. 249–252.
15. K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, Comparing case-based reasoning classifiers for predicting high-risk software componenets, *J. Syst. Softw.*, **55** (3): 301–320, 2001.
16. T. M. Khoshgoftaar and N. Seliya, Fault prediction modeling for software quality estimation: Comparing commonly used techniques, *Empirical Softw. Eng. J.*, **8** (3): 255–283, 2003.
17. K. Ganesan, T. M. Khoshgoftaar, and E. B. Allen, Case-based software quality prediction, *Int. J. Softw. Eng. Knowl. Eng.*, **10** (2): 139–152, 2000.
18. S. S. Gokhale and M. R. Lyu, Regression tree modeling for the prediction of software quality, in H. Pham, (ed.), *Proc. 3rd International Conference on Reliability and Quality in Design*, Anaheim, CA, 1997, pp. 31–36.
19. T. M. Khoshgoftaar and E. B. Allen, Logistic regression modeling of software quality, *Int. J. Reliability Quality Safety Eng.*, **6** (4): 303–317, 1999.
20. R. H. Myers, *Classical and Modern Regression with Applications*, Boston, MA: PWS-KENT, 1990.
21. L. C. Briand, W. L. Melo, and J. Wust, Assessing the applicability of fault-proneness models across object-oriented software projects, *IEEE Trans. Softw. Eng.*, **28** (7): 706–720, 2002.
22. M. C. Ohlsson and P. Runeson, Experience from replicating empirical studies on prediction models, *Proc. 8th International Software Metrics Symposium*, Ottawa, Ontario, Canada, 2002, pp. 217–226.
23. Y. Ping, T. Systa, and H. Muller, Predicting fault-proneness using OO metrics: An industrial case study, in T. Gyimothy and F. B. Abreu, (eds.), *Proc. 6th European Conference on Software Maintenance and Reengineering*, Budapest, Hungary, 2002, pp. 99–107.
24. C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*: *An Introduction*, Boston, MA: Kluwer Academic Publishers, 2000.
25. I. H. Whitten and E. Frank, *Data Mining*: *Practical Machine Learning Tools and Techniques with JAVA Implementations*. San Francisco, CA: Morgan Kaufmann, 2000.

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida
Naeem Seliya
University of Michigan—
Dearborn
Dearborn, Michigan

SOFTWARE PERFORMANCE EVALUATION

INTRODUCTION

Performance and quality of service (QoS) aspects of modern software systems are crucially important for their successful adoption in the industry. Most generally, the *performance* of a software system indicates the degree to which the system meets its objectives for timeliness and the efficiency with which it achieves this. Timeliness is normally measured in terms of meeting certain response time or throughput requirements and scalability goals. Response time refers to the time required to respond to a user request, for example a Web service call or a database transaction, and throughput refers to the number of requests or jobs processed per unit of time. Scalability, on the other hand, is understood as the ability of the system to continue to meet its objectives for response time and throughput as the demand for the services it provides increases and resources (typically hardware) are added.

Numerous studies, for example, exist in the areas of e-business, manufacturing, telecommunications, military, health care, and transportation that have shown that a failure to meet the performance requirements can lead to serious financial losses, loss of customers and reputation, and in some cases even to loss of human lives. To avoid the pitfalls of inadequate QoS, it is important to evaluate the expected performance characteristics of systems during all phases of their lifecycle. The methods used to do this are part of the discipline called *software performance engineering* (SPE) (1,2). Software performance engineering helps to estimate the level of performance a system can achieve and provides recommendations to realize the optimal performance level (3).

However, as systems grow in size and complexity, estimating their performance becomes a more and more challenging task. Modern software systems are often composed of multiple components deployed in highly distributed and heterogeneous environments. Figure 1 shows a typical architecture of a multitiered distributed component-based system (4). The application logic is partitioned into components distributed over physical tiers. Three tiers exist: presentation tier, business logic tier, and data tier. The presentation tier includes Web servers hosting Web components that implement the presentation logic of the application. The business logic tier includes a cluster of application servers hosting business logic components that implement the business logic of the application. Middleware platforms such as Java EE (5), Microsoft .NET (6), or CORBA (7) are often used in this tier to simplify application development by leveraging some common services typically used in enterprise applications. The data tier includes database servers and legacy systems that provide data management services. The inherent complexity of such architectures makes it difficult to manage their end-to-end performance and scalability. To avoid performance problems, it is essential that systems are subjected to rigorous performance evaluation during the various stages of their lifecycle. At every stage, performance evaluation is conducted with a specific set of goals and constraints. The goals can be classified in the following categories, some of which partially overlap:

Platform selection: Determine which hardware and software platforms would provide the best scalability and cost/performance ratio. Software platforms include operating systems, middleware, database management systems, and so on. Hardware platforms include the type of servers, disk subsystems, load balancers, communication networks, and so on.

Platform validation: Validate a selected combination of platforms to ensure that taken together they provide adequate performance and scalability.

Evaluation of design alternatives: Evaluate the relative performance and scalability of alternative system designs and architectures.

Performance prediction: Predict the performance of the system for a given workload and configuration scenario.

Performance tuning: Analyze the effect of various deployment settings and tuning parameters on the system performance and find their optimal values.

Performance optimization: Find the components with the largest effect on performance and study the performance gains from optimizing them.

Scalability and bottleneck analysis: Study the performance of the system as the load increases and more hardware is added. Find which system components are most utilized and investigate whether they are potential bottlenecks.

Sizing and capacity planning: Determine the amount of hardware that would be needed to guarantee certain performance levels.

Two broad approaches help conduct performance evaluation of software systems: *performance measurement* and *performance modeling*. In the first approach, load testing tools and benchmarks are used to generate artificial workloads on the system and to measure its performance. In the second approach, performance models are built and then used to analyze the performance and scalability characteristics of the system. In both cases, it is necessary to characterize the workload of the system under study before performance evaluation can be conducted. The *workload* can be defined as the set of all inputs that the system receives from its environment during a period of time (3).

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

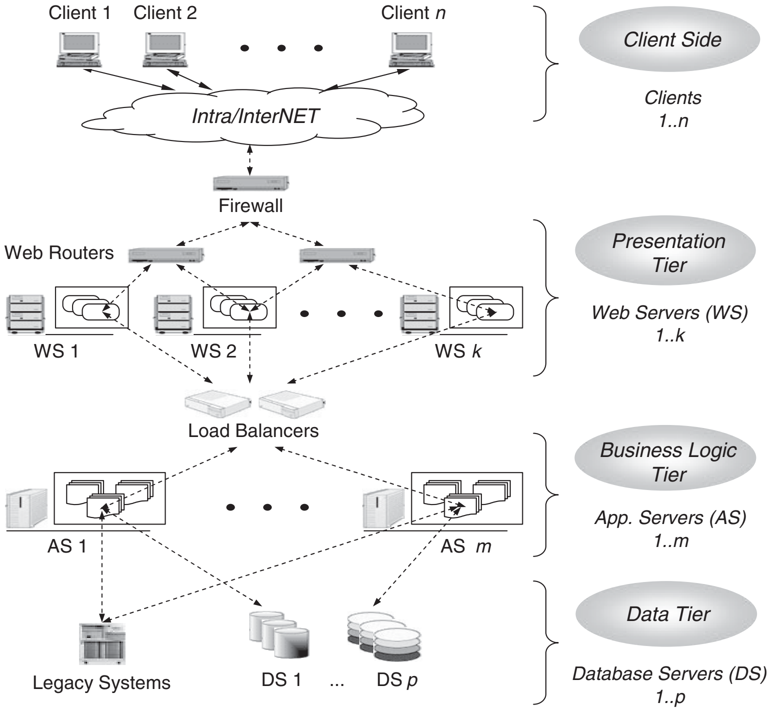


Figure 1. A multitiered distributed component-based system.

In performance evaluation studies normally workload models are used that are representations of the real system workloads.

WORKLOAD MODELS

A *workload model* is a representation that captures the main aspects of the real workload that have effect on the performance measures of interest. We distinguish between *executable* and *nonexecutable* models. Executable models are programs that mimic real workloads and can be used to evaluate the system performance in a controlled environment. For example, an executable model could be a set of benchmark programs that emulate real users sending requests to the system. Nonexecutable models, on the other hand, are abstract workload descriptions normally used as input to analytical or simulation models of the system. For example, a nonexecutable model could be a set of parameter values that describe the types of requests processed by the system and their load intensities. Workload models are aimed to be as compact as possible and at the same time representative of the real workloads under study.

As shown in Fig. 2, workload models can be classified into two major categories: *natural models* and *artificial models* (3). Natural models are constructed from real workloads of the system under study or from execution traces of real workloads. In the former case, they are called *natural benchmarks,* and in the latter case they are called *workload traces*. A natural benchmark is a set of programs extracted from the real workload such that they represent the major characteristics of the latter. A workload trace is a chronological sequence of records describing specific events that were observed during execution of the real workload. For example, in the three-tier environment described earlier, the logs collected by the servers at each tier (Web servers, application servers and database servers) can be used as workload traces. Although traces usually exhibit good representativeness, they have the drawback that they normally consist of huge amounts of data and do not provide a compact representation of the workload.

Unlike natural models, artificial workload models are not constructed using basic components of real workloads as building blocks, however, they try to mimic the real workloads. Artificial models can be classified into *synthetic benchmarks, application benchmarks*, and *abstract*

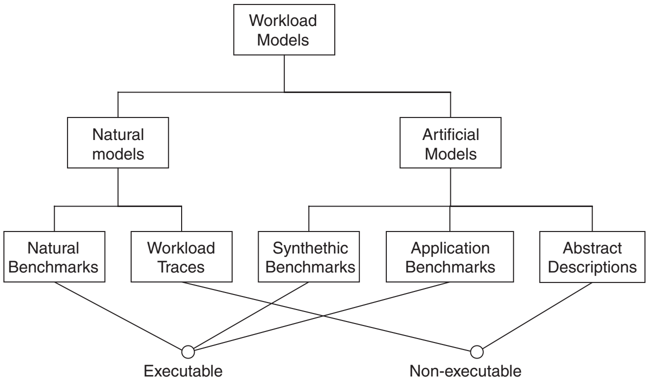


Figure 2. Taxonomy of workload models.

workload descriptions. Synthetic benchmarks are artificial programs carefully chosen to match the relative mix of operations observed in some class of applications. They usually do no real, useful work. In contrast, application benchmarks are complete real-life applications. They are normally designed specifically to be representative of a given class of applications. Finally, abstract workload descriptions are nonexecutable models composed of a set of parameter values that characterize the workload in terms of the load it places on the system components. Such models are typically used in conjunction with analytical or simulation models of the system. Depending on the type of workload, different parameters may be used, such as transaction/request types, times between successive request arrivals (interarrival times), transaction execution rates, transaction service times at system resources, and so on. As an example, an e-commerce workload can be described by specifying the types of requests processed by the system (e.g., place order, change order, cancel order), the rates at which requests arrive, and the amount of resources used when processing requests, that is, the time spent receiving service at the various system resources such as central processing units (CPUs), input–output (I/O) devices, and networks. For additional examples and details on executable and nonexecutable workload models, the reader is referred to Refs. 8 and 9, as well as 3 and 10, respectively.

PERFORMANCE MEASUREMENT

The measurement approach to software performance evaluation is typically applied in three contexts:

– *Platform benchmarking:* Measure the performance and scalability of alternative platforms on which a system can be built and/or deployed.

– *Application profiling:* Measure and profile the performance of application components during the various stages of the development cycle.

– *System load testing:* Measure the end-to-end system performance under load in the later stages of development when a running implementation or a prototype is available for testing.

In all three cases, executable workload models are used. In this section, we briefly discuss the above three contexts in which performance measurements are done. A more detailed introduction to performance measurement techniques can be found in Refs. 1, 8, 9, 11 and 12. The *Proceedings of the Annual Conference of the Computer Measurement Group* (CMG) are an excellent source of recent publications on performance measurement tools, methodologies, and concepts.

Platform Benchmarking

While benchmarking efforts have traditionally been focused on hardware performance, over the past 15 years, benchmarks have increasingly been used to evaluate the performance and scalability of end-to-end systems including both the hardware and *software* platforms used to build them (9). Thus, the scope of benchmarking efforts has expanded to include software products like Web servers, application servers, database management systems, message-oriented middleware, and virtual machine monitors. Building on scalable and efficient platforms is crucial to achieving good performance and scalability. Therefore, it is essential that platforms are validated to ensure that they provide adequate level of performance and scalability before they are used to build real applications. Where alternative platforms are available, benchmark results can be used for performance comparisons to help select the platform that provides the best cost/performance ratio. Two major benchmark standardization bodies exist, the Standard Performance Evaluation Corporation (SPEC) (13) and the Transaction Processing Performance Council (TPC) (14). Many standard benchmarks have appeared in the last decade that provide means to measure the performance and scalability of software platforms. For example, SPECjAppServer2004 and TPC-App for application servers, SPECjbb2005 for server-side Java, TPC-W and SPECweb2005 for Web servers, TPC-C, TPC-E and TPC-H for database management systems, and SPECjms2007 for message-oriented middleware. Benchmarks such as these are called *application benchmarks* because they are designed to be representative of a given class of real-world applications.

Although the main purpose of application benchmarks is to measure the performance and scalability of alternative platforms on which a system can be built, they can also be used to study the effect of platform configuration settings and tuning parameters on the overall system performance (9,15,16). Thus, benchmarking not only helps to select platforms and validate their performance and scalability, but also helps to tune and optimize the selected platforms for optimal performance. The *Proceedings of the Annual SPEC Benchmark Workshops* are an excellent source on the latest developments in benchmarking methodologies and tools (17).

Application Profiling

Application profiling is conducted iteratively during the system development cycle to evaluate the performance of components as they are designed and implemented. Design and implementation decisions taken at the early stages of system development are likely to have a strong impact on the overall system performance (1). Moreover, problems caused by poor decisions taken early in the development cycle are usually most expensive and time-consuming to correct. Therefore, it is important that, as components are designed and implemented, their performance is measured and profiled to ensure that they do not have any internal bottlenecks or processing inefficiencies. Software profilers are normally used for this purpose.

Software profilers are performance measurement tools that help to gain a comprehensive understanding of the execution-time behavior of software components. They typically provide information such as the fraction of time spent in specific states (e.g., executing different subroutines, blocking on I/O, running operating system kernel code) and the flow of control during program execution. Two general techniques are normally used to obtain such information, *statistical sampling* and *code instrumentation* (11,12). The statistical sampling approach is based on interrupting the program execution periodically and recording the execution state. The code instrumentation approach, on the other hand, is based on modifying the program code to record state information whenever a specified set of events of interest occur. Statistical sampling is usually much less intrusive; however, it only provides statistical summary of the times spent in different states and cannot provide any information on how the various states were reached (e.g., call graphs). Code instrumentation, on the other hand, is normally more intrusive; however, it allows the profiler to precisely record all the events of interest as well as the call sequences that show the flow of control during program execution. For example, in CPU time profiling, statistical sampling may reveal the relative percentage of time spent in frequently-called methods, whereas code instrumentation can report the exact number of times each method is invoked and the calling sequence that led to the method invocation.

System Load Testing

Load testing is typically done in the later stages of system development when a running implementation or a prototype of the system is available for testing. Load-testing tools are used to generate synthetic workloads and measure the system performance under load. Sophisticated load-testing tools can emulate hundreds of thousands of "virtual users" that mimic real users interacting with the system. While tests are run, system components are monitored and performance metrics (e.g., response time, throughput and utilization) are measured. Results obtained in this way can be used to identify and isolate system bottlenecks, fine-tune system components, and measure the end-to-end system scalability (18). Unfortunately, this approach has several drawbacks. First of all, it is not applicable in the early stages of system development when the system is not available for testing. Second, it is extremely expensive and time-consuming because it requires setting up a production-like testing environment, configuring load testing tools, and conducting the tests. Finally, testing results normally cannot be reused for other applications.

PERFORMANCE MODELING

The performance modeling approach to software performance evaluation is based on using mathematical or simulation models to predict the system performance under load. Models represent the way system resources are used by the workload and capture the main factors that determine the system behavior under load (10). This approach is normally much cheaper than load testing and has the advantage that it can be applied in the early stages of system development before the system is available for testing. A number of different methods and techniques have been proposed in the literature for modeling software systems and predicting their performance under load. Most of them, however, are based on the same general methodology that proceeds through the steps depicted in Fig. 3 (1, 3, 19–21).

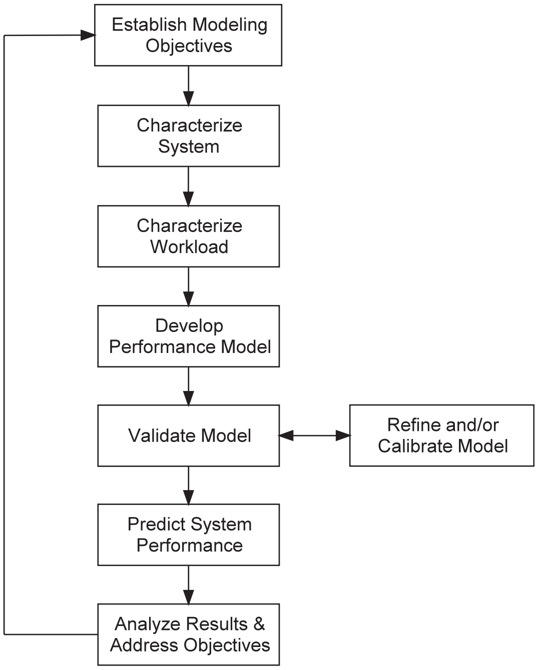

Establish Modeling Objectives
Characterize System
Characterize Workload
Develop Performance Model
Validate Model
Refine and/or Calibrate Model
Predict System Performance
Analyze Results & Address Objectives

Figure 3. Performance modeling process.

First, the goals and objectives of the modeling study are specified. After this, the system is described in detail in terms of its hardware and software architecture. The aim is to obtain an in-depth understanding of the system architecture and its components. Next, the workload of the system is characterized and a workload model is built. The workload model is used as a basis to develop a performance model. Before the model can be used for performance prediction, it has to be validated. This is done by comparing performance metrics predicted by the model with measurements on the real system. If the predicted values do not match the measured values within an acceptable level of accuracy, then the model must be refined and/or calibrated. Finally, the validated performance model is used to predict the system performance for the deployment configurations and workload scenarios of interest. The model predictions are analyzed and used to address the goals set in the beginning of the modeling study. We now take a closer look at the major steps of the modeling process.

Workload Characterization

Workload characterization is the process of describing the workload of the system in a qualitative and quantitative manner (20). The result of workload characterization is a nonexecutable workload model that can be used as input to performance models. Workload characterization usually involves the following activities (1, 22):

- The basic components of the workload are identified.
- Basic components are partitioned into workload classes.
- The system components/resources used by each workload class are identified.

- The inter-component interactions and processing steps are described.
- Service demands and workload intensities are quantified.

In the following, we discuss each of these activities in turn.

The Basic Components of the Workload are Identified. *Basic component* refers to a generic unit of work that arrives at the system from an external source (19). Some examples include HTTP requests, remote procedure calls, Web service invocations, database transactions, interactive commands, and batch jobs. Basic components could be composed of multiple processing tasks, for example client sessions that comprise multiple requests to the system or nested transactions (open or closed). The choice of basic components and the decision how granular they are defined depend on the nature of the services provided by the system and on the modeling objectives. Because, in almost all cases, basic components can be considered as some kind of requests or transactions processed by the system, they are often referred to as *requests* or *transactions*[1].

Basic Components are Partitioned into Workload Classes. To improve the representativeness of the workload model, the basic components are partitioned into classes (called *workload classes)* that have similar characteristics. The partitioning can be done based on different criteria, depending on the type of system modeled and the goals of the modeling effort (19, 23). The basic components should be partitioned in such a way that each workload class is as homogeneous as possible in terms of the load it places on the system and its resources.

The System Components and Resources Used by Each Workload Class are Identified. For example, an online request to place an order might require using a Web server, application server, and backend database server. For each server, the concrete hardware and software resources used must be identified. It is distinguished between active and passive resources (10). An *active resource* is a resource that delivers a certain service to transactions at a finite speed (e.g., CPU or disk drive). In contrast, a *passive resource* is needed for the execution of a transaction, but it is not characterized by a speed of service delivery (e.g., thread, database connection or main memory).

The Intercomponent Interactions and Processing Steps are Described. The aim of this step is to describe the processing steps, the inter-component interactions, and the flow of control for each workload class. Also for each processing step, the hardware and software resources used are specified. Different notations may be exploited for this purpose, for example client/server interaction diagrams (20), execution graphs (1), communication-processing delay diagrams (19), as well as conventional UML sequence and activity diagrams (24).

Service Demands and Workload Intensities are Quantified. The goal is to quantify the load placed by the workload components on the system. *Service-demand parameters* specify the average total amount of service time required by each workload class at each resource. Most techniques for obtaining service-demand parameters involve running the system or components thereof and taking measurements. Some techniques are also available that can be used to estimate service-demand parameters in the early stages of system development before the system is available for testing (25). *Workload-intensity parameters* provide for each workload class a measure of the number of units of work (i.e., requests or transactions), that contend for system resources. Depending on the way workload intensity is specified, it is distinguished between open and closed classes. For open classes, workload intensity is specified as an arrival rate, whereas for closed classes it is specified as average number of requests served concurrently in the system.

The product of the workload characterization steps described above (i.e., the workload model) is sometimes referred to as *software execution model* because it represents the key facets of software execution behavior (1).

Performance Models

A performance model is an abstract representation of the system that relates the workload parameters with the system configuration and captures the main factors that determine the system performance. Performance models can be used to understand the behavior of the system and predict its performance under load. Figure 4 shows the major types of performance models that are available in the literature for modeling computer systems. Note that this model classification is not clear cut because some model types partially overlap. Performance models can be grouped into two main categories: *simulation models* and *analytical models*. One of the greatest challenges in building a good model is to find the right level of detail. A general rule of thumb is: “Make the model as simple as possible, but not simpler!” Including too much detail might render the model intractable, on the other hand, making it too simple might render it unrepresentative.

Simulation Models. Simulation models are software programs that mimic the behavior of a system as requests arrive and get processed at the various system resources. Such models are normally stochastic because they have one or more random variables as input (e.g., the request interarrival times). The structure of a simulation program is based on the states of the simulated system and events that cause the system state to change. When implemented, simulation programs count events and record the duration of time spent in different states. Based on these data, performance metrics of interest (e.g., the average time a request takes to complete or the average system throughput) can be estimated at the end of the simulation run. Estimates

[1] The term transaction here is used loosely to refer to any unit of work or processing task executed in the system.

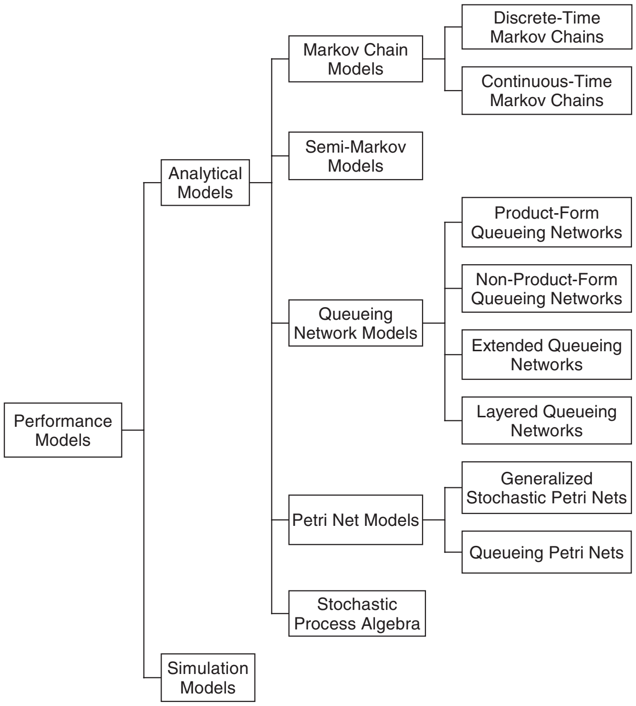

Performance Models
Analytical Models
Markov Chain Models
Discrete-Time Markov Chains
Continuous-Time Markov Chains
Semi-Markov Models
Queueing Network Models
Product-Form Queueing Networks
Non-Product-Form Queueing Networks
Extended Queueing Networks
Layered Queueing Networks
Petri Net Models
Generalized Stochastic Petri Nets
Queueing Petri Nets
Stochastic Process Algebra
Simulation Models

Figure 4. Major types of performance models.

are provided in the form of confidence intervals. A confidence interval is a range with a given probability that the estimated performance metric lies within this range. The main advantage of simulation models is that they are very general and can be made as accurate as desired. However, this accuracy comes at the cost of the time taken to develop and run the models. Usually, many long runs are required to obtain estimates of needed performance measures with reasonable confidence levels.

Several approaches to developing a simulation model (22) exist. The most time-consuming approach is to use a general purpose programming language such as C++ or Java, possibly augmented by simulation libraries (e.g., CSIM or SimPack). Another approach is to use a specialized simulation language such as GPSS/H, Simscript II.5, or MODSIM III. Finally, some simulation packages support graphical languages for defining simulation models (e.g., Arena, Extend, SES/workbench). A comprehensive treatment of simulation techniques can be found in Refs. 26 and 27.

Analytical Models. Analytical models are based on mathematical laws and computational algorithms used to derive performance metrics from model parameters. Analytical models are usually less expensive to build and more efficient to analyze compared with simulation models. However, because they are defined at a higher level of abstraction, they are normally less detailed and accurate. Moreover, for models to be mathematically tractable, usually many simplifying assumptions need to be made impairing the model representativeness.

Queueing networks and generalized stochastic Petri nets are perhaps the two most popular types of models used in practice. Queueing networks provide a very powerful mechanism for modeling hardware contention (contention for CPU time, disk access, and other hardware resources) and scheduling strategies. A number of efficient analysis methods have been developed for a class of queueing networks called *product-form queueing networks*, which enable models of realistic size and complexity to be analyzed (28). The downside of queueing networks is that they are not expressive enough to model software contention accurately (contention for processes, threads, database connections, and other software resources), as well as blocking, simultaneous resource possession, asynchronous processing, and synchronization aspects. Even though extensions of queueing networks, such as *extended queueing networks* (29) and *layered queueing networks* (also called stochastic rendezvous networks) (30–32), provide some support for modeling software contention and synchronization aspects, they are often restrictive and inaccurate.

In contrast to queueing networks, generalized stochastic Petri net models easily can express software contention, simultaneous resource possession, asynchronous processing, and synchronization aspects. Their major disadvantage, however, is that they do not provide any means for direct representation of scheduling strategies. The attempts to eliminate this disadvantage have led to the emergence of *queueing Petri nets* (33–35), which combine the modeling power and expressiveness of queueing networks and stochastic Petri nets. Queueing Petri nets enable the integration of hardware and software aspects of system behavior in the same model (36, 37). A major hurdle to the practical use of queueing Petri nets, however, is that their analysis suffers from the state space explosion problem limiting the size of the models that can be solved. Currently, the only way to circumvent this problem is by using simulation for model analysis (38).

Details of the various types of analytical models shown in Fig. 4 are beyond the scope of this article. The following books can be used as reference for additional information (3, 12, 28, 35, 39–42). The *Proceedings of the ACM SIGMETRICS Conferences* and the *Performance Evaluation Journal* report recent research results in performance modeling and evaluation. Further relevant information can be found in the *Proceedings of the International Conference on Quantitative Evaluation of SysTems (QEST)*, the *Proceedings of the Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), the *Proceedings of the International Conference on Performance Evaluation Methodologies and Tools*, (VALUETOOLS) and the *Proceedings of the ACM International Workshop on Software and Performance* (WOSP).

Model Validation and Calibration

Before a model can be used for performance prediction, it has to be validated. We assume that the system modeled or a prototype of it is available for testing. The model is said to be *valid* if the performance metrics (e.g., response time, throughput, and resource utilization) predicted by the model match the measurements on the real system within a certain acceptable margin of error (3). As a rule of thumb, errors within 10% for utilization and throughput, and

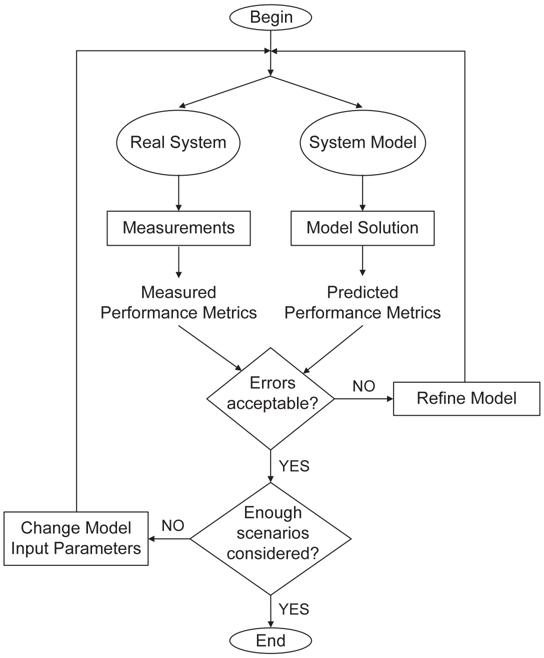

Begin
Real System
System Model
Measurements
Model Solution
Measured Performance Metrics
Predicted Performance Metrics
Errors acceptable?
NO
Refine Model
YES
Enough scenarios considered?
NO
Change Model Input Parameters
YES
End

Figure 5. Model validation and refinement process.

within 20% for response time are considered acceptable (10). Model validation is normally conducted by comparing performance metrics predicted by the model with measurements on the real system. This testing is performed for several different scenarios varying the model input parameters. If the predicted values do not match the measured values within an acceptable level of accuracy, then the model must be refined. Otherwise, the model is deemed valid and can be used for performance prediction. The validation and refinement process is illustrated in Fig. 5. It is important that the model predictions are verified for several scenarios under different transaction mixes and workload intensities before the model is deemed valid. The model refinement process usually involves the following activities:

- The model input parameters are verified.
- Assumptions and simplifications made when building the model are revisited.
- The system is monitored under load to ensure that all critical aspects of its behavior are captured by the model.
- It is considered to increase the level of detail at which the system is modeled.

If after refining the model, predicted metrics still do not match the measurements on the real system within an acceptable level of accuracy, then the model has to be calibrated. *Model calibration* is the process of changing the model to force it to match the actual system (43). This is achieved by changing the values of some model input or output parameters. The parameters may be increased or decreased by an absolute or percentage amount. Normally, input parameters are changed (e.g., service demands); however, in certain cases, also output parameters might be changed. If an output parameter is altered when calibrating the baseline model, then it must be altered in the same manner whenever the model is used for performance prediction. After the model is calibrated, the validation procedure must be repeated to make sure that the calibrated model now accurately reflects the real system and workload. For a detailed discussion of model calibration techniques, the reader is referred to Refs. 10 and 44.

The extent to which a model can be validated quantitatively as described above depends on the availability of an implementation of the system components. In the initial phases of system development when no implementation is available, model validation would be limited to revisiting the assumptions made when building the model. If a system or a prototype with a similar architecture to the one modeled is available, then it could be used to provide some rough measurement data for quantitative validation.

Software Performance Engineering

Over the last 15 years, a number of approaches have been proposed for integrating performance evaluation and prediction techniques into the software engineering process. Efforts were initiated with Smith's seminal work pioneered under SPE (45). Since then many meta-models for describing performance-related aspects (46) have been developed by the SPE community, the most prominent being the UML SPT profile and its successor the UML MARTE profile, both of which are extensions of UML as the de facto standard modeling language for software architectures. Other proposed meta-models include SPE-MM (47), CSM (48), and KLAPER (49). The common goal of these efforts is to enable the automated transformation of design-oriented software models into analysis-oriented performance models, which make it possible to predict the system performance. A recent servey of model-based performance prediction techniques was published in Ref. 50. Many techniques that use a range of different performance models have been proposed including standard queueing networks (3, 25, 47, 51), extended queueing networks (49, 52, 53), layered queueing networks (48), stochastic Petri nets (54, 55), and queueing Petri nets (4, 21). In recent years, with the increasing adoption of component-based software engineering, the performance evaluation community has focused on adapting and extending conventional SPE techniques to support component-based systems. For a recent survey of performance prediction methodologies and tools for component-based systems, refer to Ref. 56.

OPERATIONAL ANALYSIS

An alternative approach to performance evaluation known as *operational analysis* is based on a set of basic invariant relationships between performance quantities (57). These relationships, which are commonly known as *operational laws*, can be considered as consistency requirements for the values of performance quantities measured in any particular experiment. We briefly present the most important operational laws. Consider a system made up of K resources (e.g., servers, processors, disk drives, network links). The system processes transactions requested by clients. It is assumed that during the processing of a transaction, multiple resources can be used and at each point in time the transaction is either being served at a resource or waiting for a resource to become available. A resource might be used multiple times during a transaction, and each time a request is sent to the resource, we will refer to this as the transaction *visiting* the resource. The following notation will be used:

V_i the average number of times resource i is visited during the processing of a transaction.

S_i the average service time of a transaction at resource i per visit to the resource.

D_i the average total service time of a transaction at resource i.

U_i the utilization of resource i (i.e., the fraction of time the resource is busy serving requests).

X_i the throughtput of resource i (i.e., the number of service completions per unit time).

X_0 the system throughput (i.e., the number of transactions processed per unit time).

R the average transaction response time (i.e., the average time it takes to process a transaction including both the waiting and service time in the system).

N the average number of active transactions in the system, either waiting for service or being served.

If we observe the system for a finite amount of time T, assuming that the system is in steady state, then the following relationships can be shown to hold:

Utilization Law:

$$U_i = X_i \times S_i$$

Forced Flow Law:

$$X_i = X_0 \times V_i$$

Service Demand Law:

$$D_i = U_i / X_0$$

Little's Law:

$$N = X_0 \times R$$

The last of the above relationships, Little's Law, is one of the most important and fundamental laws in queueing theory. It can also be extended to higher moments (58). If we assume that transactions are started by a fixed set of M clients and that the average time a client waits after completing a transaction before starting the next transaction (the client think time) is Z, then using Little's Law, the following relationship can be easily shown to hold:

Interactive Response Time Law:

$$R = \frac{M}{X_0} - Z$$

Although operational analysis is not as powerful as queueing theoretic methods for performance analysis, it has the advantage that it can be applied under much more general conditions because it does not require the strong assumptions typically made in stochastic modeling. For a more detailed introduction to operational analysis, the reader is referred to Refs. 3, 10, and 22.

SUMMARY

In this article, an overview of the major methods and techniques for software performance evaluation was presented. First, the different types of workload models that are typically used in performance evaluation studies were considered. Next, an overview of common tools and techniques for performance measurement, including platform benchmarking, application profiling, and system load testing, was given. Then, the most common methods for workload characterization and performance modeling of software systems were surveyed. The major types of performance models used in practice were considered and their advantages and disadvantages were discussed. An outline of the approaches to integrating model-based performance analysis into the software engineering process was presented. Finally, operational analysis was introduced briefly as an alternative to queueing theoretic methods.

BIBLIOGRAPHY

1. C. U. Smith and L. G. Williams, *Performance Solutions - A Practical Guide to Creating Responsive, Scalable Software*, Reading, MA: Addison-Wesley, 2002.
2. R. R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, eds. *Performance Engineering, State of the Art and Current Trends, Vol. 2047 of Lecture Notes in Computer Science*, New York: Springer, 2001.
3. D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design*, Englewood Cliffs, NJ: Prentice Hall, 2004.
4. S. Kounev, *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction*, Herzogenrath, Germany: Shaker Verlag, 2005.
5. Sun Microsystems, Inc. Java Platform, Enterprise Edition (Java EE), 2007. http://java.sun.com/javaee/.
6. Microsoft Corp. Microsoft .NET Framework, 2007. http://msdn.microsoft.com/netframework/.
7. Object Management Group (OMG). Common Object Request Broker Architecture (CORBA), 2007. http://www.corba.org/.
8. L. K. John and L. Eeckhout, eds., *Performance Evaluation and Benchmarking*. Boca Raton, FL: CRC Press, 2006.
9. R. Eigenmann, ed., *Performance Evaluation and Benchmarking with Realistic Applications*. Cambridge, MA: The MIT Press, 2001.
10. D. A. Menascè, V. A. F. Almeida, and L. W. Dowdy, *Capacity Planning and Performance Modeling - From Mainframes to Client-Server Systems*, Englewood Cliffs, NJ: Prentice Hall, 1994.
11. D. Lilja, *Measuring Computer Performance: A Practitioner's Guide*, Cambridge, U.K., Cambridge University Press, 2000.

12. R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, New York: Wiley-Interscience, 1991.

13. Standard Performance Evaluation Corporation (SPEC). http://www.spec.org/.

14. Transaction Processing Performance Council (TPC). http://www.tpc.org/.

15. S. Kounev and A. Buchmann, Improving data access of J2EE applications by exploiting asynchronous processing and caching services, *Proc. of the 28th International Conference on Very Large Data Bases - VLDB2002*, Hong Kong, China, 2002.

16. S. Kounev, B. Weis, and A. Buchmann, Performance tuning and optimization of J2EE applications on the JBoss platform, *J. Comput. Res. Manage.*, **113**: 2004.

17. SPEC Benchmark Workshop Proceedings. http://www.spec.org/events/.

18. B. M. Subraya, *Integrated Approach to Web Performance Testing: A Practitioner's Guide*, Hershey, PA: IRM Press, 2006.

19. D. Menascé and V. Almeida, *Capacity Planning for Web Performance: Metrics, Models and Methods*, Upper Saddle River, NJ: Prentice Hall, 1998.

20. D. Menascé, V. Almeida, R. Fonseca, and M. Mendes, A methodology for workload characterization of e-commerce sites, *Proc. of the 1st ACM Conference on Electronic Commerce*, Denver, CO, 1999, pp. 119–128.

21. S. Kounev, Performance modeling and evaluation of distributed component-based systems using queueing Petri nets, *IEEE Trans. Soft. Engineer.*, **32**(7): 486–502, 2006.

22. D. Menascé and V. Almeida, *Scaling for E-Business - Technologies, Models, Performance and Capacity Planning*, Upper Saddle River, NJ: Prentice Hall, 2000.

23. J. Mohr and S. Penansky, A forecasting oriented workload characterization methodology, *CMG Trans.*, **36**: 1982.

24. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Reading, MA: Addison-Wesley, 1999.

25. D. A. Menascé and H. Gomaa, A method for desigh and performance modeling of client/server systems, *IEEE Trans. Soft. Engin.*, **26**(11): 2000.

26. A. Law and D. W. Kelton, *Simulation Modeling and Analysis*. 3rd ed. New York: Mc Graw Hill Companies, Inc., 2000.

27. J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 3rd ed. Upper Saddle River, N.J: Prentice Hall, 2001.

28. G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, 2nd ed. New York: John Wiley & Sons, Inc., 2006.

29. E. A. MacNair, An introduction to the research queueing package, *WSC '85: Proc. of the 17th Conference on Winter Simulation*, New York, NY, 1985, pp. 257–262.

30. M. Woodside, Tutorial Introduction to Layered Modeling of Software Performance, 3rd ed., 2000. Available: http://www.sce.carleton.ca/rads/lqn/lqn-documentation/tutorialg.pdf.

31. P. Maly and C. M. Woodside, Layered modeling of hardware and software, with application to a LAN extension router, *Proc. of the 11th International Conference on Computer Performance Evaluation Techniques and Tools - TOOLS 2000*, Motorola University, Schaumburg, *Ill*, 2000.

32. M. Woodside, J. Neilson, D. Petriu, and S. Majumdar, The stochastic rendezvous network model for performance of synchronous client-server-like distributed software, *IEEE Trans. Comput.*, **44**(1): 20–34, 1995.

33. F. Bause, Queueing Petri nets - A formalism for the combined qualitative and quantitative analysis of systems, *Proc. of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, 1993.

34. F. Bause and P. Buchholz, Queueing Petri nets with product form solution, *Perform. Eval.*, **32**(4): 265–299, 1998.

35. F. Bause and F. Kritzinger, *Stochastic Petri Nets - An Introduction to the Theory*, 2nd ed. New York: Vieweg Verlag, 2002.

36. F. Bause, P. Buchholz, and P. Kemper, Integrating software and hardware performance models using hierarchical queueing Petri nets, *Proc. of the 9. ITG / GI - Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, (MMB'97)*, Freiberg, Germany, 1997.

37. S. Kounev and A. Buchmann, Performance modelling of distributed e-business applications using queuing Petri nets, *Proc. of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software - ISPASS2003*, Austin, TX, 2003.

38. S. Kounev and A. Buchmann, SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation, *Perform. Eval.*, **63**(4-5): 364–394, 2006.

39. K. S. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science Applications*, 2nd ed. New York: John Wiley & Sons, Inc., 2002.

40. R. Sahner, K. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems - An Example-Based Approach Using the SHARPE Software Package*, Dordrecht, The Netherlands: Kluwer Academic Publishers, 1996.

41. K. Begain, G. Bolch, and H. Herold, *Practical Performance Modeling - Application of the MOSEL Language*, Dardrecht: The Netherlands, Kluwer Academic Publishers, 2001.

42. J. Hillston, *A Compositional Approach to Performance Modelling*, Cambridge, U.K., Cambridge University Press, 1996.

43. P. J. Buzen and A. W. Shum, Model calibration. in *Proc. of the 1989 International CMG Conference, Reno, Nevada*, 1989, pp. 808–811.

44. J. Flowers and L. W. Dowdy, A comparison of calibration techniques for queuing network models, *Proc. of the 1989 International CMG Conference*, Reno, Nevada, 1989 pp. 644–655.

45. C. U. Smith, *Performance Engineering of Software Systems*, Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1990.

46. V. Cortellessa, How far are we from the definition of a common software performance ontology? *WOSP'05: Proc. of the 5th international Workshop on Software and Performance*, 2005, pp. New York, NY195–204.

47. C. U. Smith, C. M. Llad, V. Cortellessa, A. Di Marco, and L. G. Williams, From UML models to software performance results: an SPE process based on XML interchange formats, *WOSP '05: Proc. of the 5th International Workshop on Software and Performance*, New York, NY, 2005, pp 87–95.

48. D. Petriu and M. Woodside, An intermediate metamodel with scenarios and resources for generating performance models from UML designs, *Soft. Sys. Mode.*, **6**(2): 163–184, 2007.

49. V. Grassi, R. Mirandola, and A. Sabetta, Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach, *J. Sys. Soft.*, **80**(4): 528–558, 2007.

50. S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, Model-based performance prediction in software development: A survey, *IEEE Trans. Soft. Engineer.*, **30**(5): 295–310, 2004.
51. V. S. Sharma, P. Jalote, and K. S. Trivedi, *A Performance Engineering Tool for Tiered Software Systems*. Los Alamitos, CA: IEEE Computer Society, 2006, pp. 63–70.
52. V. Cortellessa and R. Mirandola, Deriving a queueing network based performance model from UML diagrams, *WOSP '00: Proc. of the 2nd International Workshop on Software and Performance*, New York, NY, 2000, pp. 58–70.
53. A. D'Ambrogio and G. Iazeolla, Design of XMI-based tools for building EQN models of software systems, in P. Kokol (ed.), *IASTED International Conference on Software Engineering, part of the 23rd Multi-Conference on Applied Informatics*, Innsbruck, Austria, Calgary, Alberta, Canada: IASTED/ACTA Press, 2005 pp. 366–371.
54. J. P. Lopez-Grao, J. Merseguer, and J. Campos, From UML activity diagrams to stochastic Petri nets: application to software performance engineering, *SIGSOFT Softw. Eng. Notes*, **29**(1): 25–36, 2004.
55. S. Bernardi and J. Merseguer, QoS assessment via stochastic analysis, *IEEE Inter. Comput.*, **10**(3): 32–42, 2006.
56. S. Becker, L. Grunske, R. Mirandola, and S. Overhage, Performance prediction of component-based systems: A survey from an engineering perspective, in R. H. Reussner, J. Stafford, and C. Szyperski, (eds.), *Architecting Systems with Trustworthy Components*, Vol. 3938 of *LNCS*, New York: Springer, 2006, pp. 169–192.
57. P. J. Denning and J. P. Buzen, The operational analysis of queueing network models, *ACM Comput. Surv.*, **10**(3): 225–261, 1978.
58. W. Witt, A review of L = lambda-W and extensions, *Queueing Sys.*, **9**(3): 235–268, 1991.

Samuel Kounev
University of Cambridge
Cambridge, United Kingdom

SOFTWARE PRODUCT CERTIFICATION

INTRODUCTION

Here, we examine the process and challenges of certifying software-intensive systems. We will first review the interpretation of the term, *certification*. We then review different types of certification processes, and discuss challenges to forming software certification procedures. And finally, we discuss a strategy for gaining the maximum benefit from performing certification by formulating the correct procedures that are needed to roll out a certification plan by recommending appropriate guidelines to those developers whose products will be judged.

DIFFERENT TYPES OF CERTIFICATION (WHO, WHAT, WHY?)

Software certification is simply the process of generating a *certificate* that supports claims such as (1) the software was developed in a certain manner, (2) the software will exhibit some set of desirable run-time characteristics (also termed attributes), (3) the software version is the authentic one, or (4) the software has some other static characteristic embedded into it (for example, there are no faults of type *X*). The reason for having certificates in the first place is to be able to make predictions about how successfully the software will operate over time. Missions, environments, hardware, threats, personnel, and many other factors change during the software's lifetime, and the purpose for creating certificates is to reduce the uncertainty as to what impact all of these changes will have on the software.

So, for example, the certificate could simply state that a particular type of testing was applied and to what degree of thoroughness that testing was performed. A certificate can state that the software should be successfully composed with any other software component that contains a certain set of predefined characteristics. Or a certificate can state that the software will never fail more than once in 10,000 hours. Furthermore, any certificate created for (2) must have very specific assumptions about the environment, mission, and threat space that the software will experience during operational deployment. Thus, a *certificate* is simply the end product of this accreditation process.

Note that the information content in a certificate must be carefully written. Failure to remove ambiguities or inconsistencies in certificates essentially renders them as dangerous. For example, if a certificate that is based on test results only claims that, for 10 distinct test cases that were used, the output from the software was always a positive integer, that should not be interpreted as a guarantee that all other test cases not tried, the output will always be a positive integer. Therefore, key facet to software certification is to not over claim aspects of how the software will behave in the future. And without a proper scoping of the bounds of the claims in the certificate, the certificate will be flawed.

Two key questions that arise before certification occurs are as follows: (1) *Who* does the certification? and (2) *what* is being certified?

Let's first turn to the question of who performs the certification. The process of creating a certificate is typically performed by one of three parties: (1) either by the vendor of the software, (2) by a user, or (3) by an independent *third party* that is performing the service independent of the vendor or users. These key approaches roll a certification process out after the appropriate *standards* (i.e., criteria for certificate judgment and scope) are formed. Note, however, that first-party certification is always viewed suspiciously, and second-party certification adds an additional burden on the user of the software that may be a burden for which the user cannot perform. Therefor, third-party certification is generally viewed as preferable.

Numerous examples of third-party certification occur in industries such as electronics (e.g., Underwriters Laboratory), aviation [e.g., DERs (designated engineering representatives)], and, consumer products (e.g., *Consumer Reports* magazine). In industrial applications, organizations such as TUV dominate the market for the assessment of programmable controllers used in the process industry, and they are also active in railways and nuclear power plant assessments. In aviation, DERs are employee representatives of the company building the aircraft or components who have sworn their allegiance to a regulatory organization such as the FAA or CAA. And in terms of consumer products, a plethora of consumer advocate organizations do everything from rate the safety of toys for children to rating the quality of automobiles.

Many quality-of-service (QOS) attributes of software can be certified: reliability, safety, security, performance, etc. In this effort, we are focusing on two attributes: interoperability and safety.

Certification approaches are not a totally new idea. For example, The Open Group certifies that Linux implementations are indeed Linux and that X-windows conforms to their standards. Other examples are protocol testing by telecoms laboratories and compiler testing to demonstrate conformance with a language definition. (The rather misnamed Ada validation suite tests for certain functional aspects of the compiler but not for all required QOS attributes such as *reliability*.) However, the uniqueness of this effort stems from the following hypothesis: it is better to certify products that have negative safety consequences if they fail than to only certify how those products were developed.

We begin from the premise that interoperability can only be successfully achieved if the following characteristics are considered: (1) composability, (2) predictability, (3) attribute measurement, (4) QoS attribute trade-off analysis (economic and technical), (5) fault tolerance and

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

non-interference analysis, (6) requirements traceability, (7) access to prequalified components, and finally, (8) precise bounding of the software's mission, environment, and threat space. If we have access to information concerning these 8 considerations, we contend that a plausible and scientifically sound approach to software certification can be formulated. Furthermore, information concerning human factors must be included in the definition concerning the assumed, target environment.

This discussion then brings us to the issue concerning the difference between an (1) original certification, (2) decertification, (3) miscertification, and (4) recertification. As mentioned, the original certification is based on specific assumptions about the environment, mission, and threat space that the software will encounter during operational usage. In most systems over time, these assumptions will change more quickly than the code itself can be modified.

Therefore, some events require that an existing certificate (or parts of it) be nullified. This process is a decertification. To ensure that the software can handle changing assumptions, additional certification activities will be required. This process is a recertification, and it need not necessarily be a complete start from scratch effort, provided that assurances included in the original certificate still hold. And a miscertification is simply the problem of creating the wrong certificate or no certificate at all (when one should have been created). Miscertifications are of grave concern, and the goodness of the certification process and how well it is adhered to in order to avoid this event are of much importance.

Before leaving the topic of miscertifications, we must stress that, ultimately, any certification program must ensure *fairness*. Vendors of products generally view regulation and certification suspiciously, where vendor A believes that vendor B got a better deal when going through the process. All steps must be considered to ensure that a fair hearing of all evidence occurs. This process does not mean that mistakes will not be made, but if a certification program is perceived as a coin toss, where the outcomes made are lacking repeatable, scientific, and statistical processes, the certification program will ultimately die.

Certificates are typically done as a follow-up check to ensure that certain processes were followed during development; these "process certificates" are the first type of certification that will be discussed here. The second type of software certification often mentioned refers to the licensing of software engineering professionals; this is still referred to as software certification but should more appropriately be referred to as "professional licensing." And the third and most important but most difficult claim that a certificate can make is the determination of how the software will behave in use. This process is referred to as "product certification," and that is our focus here.

Note that the three key messages that a certificate can convey are not necessarily the same:

1. Compliance with standards *vs*.
2. Fitness for purpose *vs*.
3. Compliance with the requirements.

Compliance with the standards simply means that the standards that were required during development were indeed followed, but that does not mean that the product itself is fit for the purpose that the user needed for it to be, and that does not necessarily mean that the software complies with the requirements.

The key difference between (2) and (3) is that those two are only equivalent if the requirements accurately and completely defined what the user needed the software to do, and thus, it is possible that the software meets the requirements but does not perform as the user needs. Note that (1) deals with process certification and that (2) and (3) deal with product certification.

The beauty of having a certificate with correct information is that it allows for a common language (a.k.a., mutual recognition agreements that support *interoperability*) that can be employed to discuss relatively abstract notions. As we know, software is somewhat amorphous in that, unlike a hardware entity, it is hard to get an understanding about the QoS properties of entities that are so abstract.

So from that standpoint, certification standards can be beneficial. One classic example is the recent adoption by many nations of the Common Criteria, which is simply a process certificate that defines various security levels and the processes that must be employed to demonstrate those levels. And furthermore, some accreditation agencies (e.g., NIST) now certify third-party companies that actually perform the work and generate the certificates.

But with the exception of a few other process certificates (e.g., IEC 61508 and RTCA DO-178B), not many organizations perform third-party certifications that are product-focused. In fact, organizations such as Microsoft and Netscape have been accused of violating user privacy, by spying on their users during product usage and sending information back to the companies that allow those companies to not only collect accurate *operational profiles* (operational profiles are part of what we consider as the environment) but also to perform a "quasi" first-person product certification.

Note that software certification is similar to Independent Verification and Validation (IV&V), a technique that has long been used in the NASA and DoD communities, and so we now should explain how those two term relate. To begin, Barry Boehm defines *validation* as making sure that you are building the right product, and *verification* involves assuring that you build the product right (i.e., correctly). And although these terms may seem confusing, they are both closely related.

Furthermore, little "i" Validation and Verification (V&V) is simply performing first-person V&V on a system, and big "I" V&V requires independence, from either a second party or third party (third party is typically preferred). How IV&V relates to certification is as follows: They are the same, provided that the type of certification being performed is either a process certification or product certification. However, one caveat here: Certification typically results in a certificate, and IV&V does not necessarily result in a certificate.

BENEFITS AND THREATS

The development, deployment, and integration of systems developed and perhaps certified to a wide range of military standards is an inescapable part of the problem of interest here. There are, however, benefits and threats from attempting this, as follows.

Potential Benefits

- Certification can influence the vendor space from which the DoD acquires components. That is to say that a certification program can force a minimum set of requirements that all software should satisfy. This processs has the potential to raise the general level of dependability or to reduce costs to the users.
- Certification as a basis for gaining assurance over time, as a form of trend analysis that shows that a system is improving during development or maturing during usage.
- Risk transfer from the user or vendor to the certification authority.
- Guarantees of some minimum level of behavior (both functional and nonfunctional) for products. Note that nonfunctional behaviors generally include safety, security, availability, performance, fault tolerance, maintainability, survivability, and sustainability. And also note that *quantitative* and *qualitative metrics* such as mean-time-to-failure, mean-time-to-repair, mean-time-to-hazard, up-time, and performance can all be collected as part of the evidence needed to create a software certificate.

Potential Threats

- Adding unnecessary costs and delays to projects
- Giving unwarranted confidence in system behavior as a result of miscertification
- Preventing flexibility, innovation, and interoperability, as certificates can be quite narrowly defined
- Reducing the ability of user to undertake an examination of a product (why bother if organization XYZ has already done it?)

Although it is unclear as to what the current world market is in software certification, we can look at the recent NIST report that said that, in the United States alone, the United States lost around $60B as a result of inadequate software testing in 2001. Had the United States had access to a certification organization such as Underwriter's Laboratory to provide independent assessments on the quality of the software before its release, it is not surprising to assume that that number could have been decreased.

Note that there are two differing "political" camps on the ethics of having certification processes: there are those that believe any "bar" that people are forced to cross is better than no bar at all. And there are those that believe that any bar lulls those that produce products into a false sense of security: So long as they do "just enough" to cross the bar, then they have done enough to satisfy minimum industry standards. In practice, any certification approach will have an impact on the market and the behavior of suppliers, and so the issues are not solely technical, and any strategy must be cognizant of the perhaps subtle interplay of technical, social, and market forces.

The future systems are likely to be heterogeneous, dynamic coalitions of systems of systems (SoS), and as such, they will have been built and assessed to a wide variety of differing standards and guidelines. Our main recommendations are that the certification of SoS should be based around the concept of interoperability cases, generalizing the current requirement for safety and reliability cases. Examples of "reliability cases" can be found in British Def Stan 00-42 Part 3, which deals with system reliability and maintainability; Part 2 of that standard deals specifically with building software reliability cases. Safety cases are required in the British military's Def Stan 00-55 and more recently in the United Kingdom's CAA Safety Regulation of air traffic management systems and its proposals[1].

Ultimately, all certificates are not warranties or guarantees but evidence, arguments, and claims. Thus, all information to support creation of a certificate should be based on a *claims-arguments-evidence*[2] framework with the following components:

1. A goal-based view of that expresses certification requirements in terms of a set of claims about the system and its QoS attributes
2. Evidence that support the claims
3. An explicit set of argument that provides a link from the evidence to the claims
4. For critical systems, the underlying assumptions and concepts used to support and formulate the goals and claims should be described in terms of a series of models (e.g., at system, architecture, design, and implementation levels)

Such a framework should provide a technical basis that allows for interworking of standards such as IEC 61508 , UL 1998, and DO178[3] IEC 61508 is process and system safety focused, UL 1998 is component safety and product oriented, and DO178B is system and reliability focused. Thus, harmonizing three such standards into a single approach can only be accomplished using an approach such as the claims-evidence-arguments perspective because standard is attempting to convey a different definition for what is or is not trustworthy.

Although this goal-based approach moves from safety to other dependability areas (e.g., to interoperability), it needs

[1] CAP670 SW01, "*Requirements for Software Safety Assurance in Safety Related ATS Equipment*."

[2] An introduction to safety cases on which these idea are built can be found at http://www.adelard.com/ that hosts the guidance for the IEE Functional Safety portal on this topic.

[3] Need a cross-reference and consistency with the standards section of the report.

supporting technical work and the development of a body of practice. Although there is considerable experience with this approach for safety applications worldwide, it may be new to some organizations, and the deployment of the approach could be facilitated by:

1. Guidance on strategies and arguments for demonstrating claims and on how claims might be derived, including guidance on what are useful certifiable and measurable properties.
2. Guidance on how evidence is generated by validation and vertifcation techniques. This is not found in any existing standards, and we have begun to elaborate on how evidence is generated throughout the lifecycle. It should be noted that it is extremely difficult to provide high-fidelity certificates for highly critical systems. To obtain substantially improved mean-time-to-failure measures (beyond those of commercial software), one order of magnitude in fault density reduction is generally required.
3. Guidance on pragmatics such as feasibility, scalability, and tool support.
4. Guidance on the relationship to (and the interface with) frequently used standards.

CONCLUSIONS

Disparate forms of certification exist, and they vary widely in their objectives, the level of detail information they provide, and their ability to effectively reduce project and system risk.

Certification is an attempt to transfer a product's risk to the certifier, but in most safety situations, the user of the system remains responsible for the product's safety. However, in other situations, the risk transfer can be real, thus making it highly attractive to the user. However, for software, the fear of liability that stems from potentially miscertifying software has prevented much commercial interest in creating laboratories such as UL.

All military organzations, both in the United States and abroad, should encourage certification as a tool for supporting interoperability and hence confidence that their systems are behaving as anticipated. Process certification should not be totally dismissed and can support confidence in the evidence collected (and normally be a prerequisite to baseline standards such as ISO9001), but the overall emphasis of certification should be on the product. We propose that a claim-argument-evidence-based approach should be adopted as best practice.

At this stage, we are cautious about recommending the actual arrangements for rolling out a certification program, and whether self-certification, second-party, or independent certification is optimal. For critical systems, third party is generally regarded as best practice. And for less critical systems, the benefits of independent oversight should be assessed, and only those activities that add value to prejects should be identified.

FURTHER READING

J. Voas and C. Vossler, Defective software: An overview of legal remedies and technical measures available to consumers, *Adv. Comput.*, **53**: 451–497, 2001.

J. Voas, *Software Certificates and Warranties: Ensuring Quality, Reliability, and Interoperability*. New York: Wiley, 2004.

J. Voas, Certification: Reducing the hidden costs of poor quality, *IEEE Software*, **16** (4): 22–25, 1999.

R. Bloomfield, J. Cazin, D. Craigen, N. Juristo, E. Kesseler, and J. Voas, Final Report of the NATO Research Task Group IST-027/RTG-009 on the Validation, Verification, and Certification of Embedded Systems, 2004.

J. Voas, Certifying off-the-shelf software components, *IEEE Computer*, **31** (6): 53–59, 1998. (Translated into Japanese and reprinted in *Nikkei Computer* magazine).

J. Voas, Toward a usage-based software certification process, *IEEE Computer*, **33** (8): 32–37, 2000.

J. Voas, Certifying software for high assurance environments, *IEEE Software*, **16** (4): 48–54, 1999.

JEFFREY VOAS
SAIC
Arlington, Virginia

SOFTWARE QUALITY CLASSIFICATION MODELS

INTRODUCTION

Software quality is an important attribute for the successful operation of high-assurance systems. Many measures of quality exist, fault content being the most common. An ideal system will have no faults, but given the current state-of–the-art of software engineering, no large system can be guaranteed to be fault-free, and the goal is to minimize the presence of faults in the delivered product. Toward this objective, a variety of tools and techniques are employed throughout the development lifecycle. Among these, testing is considered to be the most effective technique. Because testing resources are limited and expensive, it is highly desirable to identify and target potentially fault-prone components early for efficient allocation of testing and other quality assurance efforts, where the term "fault-prone" is application-dependent, which requires objective and dependable models that relate component metrics to fault-proneness. Such models can be used to classify components into two classes, fault-prone or not fault-prone. However, the software development process is not understood well enough to derive theory-based models. Therefore, such models are mostly obtained from experimental or historical data using one of many available techniques after an empirical modeling process. The conjecture is that new components that are similar to the old ones likely are to have equivalent fault-proneness.

Formally, this modeling activity can be described as developing an input–output mapping pattern on the basis of limited evidence about its nature. The evidence available is a set of labeled data about n components, denoted as

$$D = \{(\mathbf{x}_i, y_i) : x_i \in R^d, y_i \in R, i = 1, 2, \ldots, \mathrm{n}\} \tag{1}$$

Here, $\boldsymbol{x}_i$ are the component metrics and y_i are the class labels. This article addresses various issues that occur during the model development and evaluation process.

Classification models are employed in many disciplines (e.g., astronomy, medicine, computer science, social sciences, and engineering). A vast body of literature exists that deals with the development of such models using techniques from statistics, machine learning, neural networks, and so on. In software engineering, the component classification problem has been addressed using techniques such as principal component analysis, logistic regression, neural networks, classification trees, and case-based reasoning. Some of the reported studies provide a comparative assessment of various classification techniques to gain insight into their applicability and predictive performance in varying software development environments.

The remainder of this article is organized as follows. A brief technical description of some component classification techniques is presented, as is an overview of selected representative studies. A generic model development and evaluation process is then presented, which includes a description of several important issues that occur during this process. The radial basis function (RBF) model, a recent technique used for software component classification, is also described. A case study is presented to illustrate the modeling process using RBF and NASA software data. Some concluding remarks are presented to finalize the article.

CLASSIFICATION TECHNIQUES

This section provides a brief description of some commonly used techniques reported in the literature for software classification applications. Details about their theoretical underpinnings and model fitting algorithms can be found in Han and Kamber (1) and in Hastie et al. (2).

Logistic Regression

Logistic regression is used to determine the posterior probability of a class as a linear function of predictor variables. In the simple case of two classes, the probability, p, of a component being fault-prone in terms of metrics $x_1, x_2, \ldots, x_d$ is given by

$$\log\left(\frac{p}{1-p}\right) = \beta_o + \sum_{i=1}^{d} \beta_i x_i \tag{2}$$

The parameters β's are usually obtained by the maximum likelihood estimation method. Let the class of a component be y_i, $y_i = 0$ or 1, where 0 is not fault-prone and 1 is fault-prone, and suppose there are n components in the dataset. Then treating y_i's as n independent Bernoulli trials, with success probability p, the likelihood function is obtained from Equation (2) based on the data from Equation (1). The function is then solved using the Newton Raphson algorithm. The metrics values of a new component are then used to determine its class using the fitted model obtained from Equation (2).

Classification Trees

Tree-based classification models partition the metrics space according to some specified criterion. The model is represented graphically as a spanning tree that is traversed according to component metrics and their values. The leaf nodes represent component classes. The tree is generated recursively in a top-down fashion using one of many algorithms. Quinlan's (3) in a popular algorithm that starts with a single node (root node) that represents the metric that best discriminates between component classes

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

using an entropy-based criterion. Subtrees are then generated recursively using the remaining metrics.

Principal Component Analysis

In most practical applications, the metric values are correlated so that the resulting models are hard to interpret and are not stable. Principal component analysis (PCA) is a statistical technique that generates new metrics, called the principal components, that are weighted sums of the original metrics. These metrics are orthogonal to each other, but not independent. Generally, small number of principal components can capture the effect of all the metrics. A classification model is developed in terms of these principal components, which leads to a reduction in the dimensionality of the model, a desirable feature of a classifier. However, all the metrics values of a new component are still needed to determine its class.

Case-Based Reasoning

This technique generates classification models based on previous similar cases in the database, called the case base. The similarity of a component in terms of its metrics is determined by a specified distance measure. The rationale behind this technique is that components that are similar to each other are likely to have a similar fault-proneness property. Many issues need to be resolved while designing a case-based classifier, such as choice of distance measure, number of neighbors to use for determining fault-proneness, and weights to assign to the component metrics.

Neural Networks

A neural network consists of one input, one output, and one or more hidden layers, each responsible for data processing. The component metrics are presented to the input layer. Their weighted values are processed by the nodes (neurons) in the first hidden layer according to a specified activation function, weights, and bias values. The outputs are then presented to the next layer for processing. This process continues until the output layer evaluates the component class. The architecture and other parameter values of the network are determined by using one of many algorithms, such as the popular back-propagation algorithm. During training, the discrepancy between the network generated and the true output values is back-propagated through the network to update the weight and bias parameters. The training process is repeated for a specified number of iterations or when a desired accuracy is achieved.

SOFTWARE CLASSIFICATION STUDIES

Several classification models have been developed and are employed in software engineering applications. Many studies during the past 15 years have described the results of classification investigations using domain-specific data from specific development environments and one or more classification techniques. As a result, it is not practical to draw general inferences about the relative merits of various techniques. Yet, they provide insights into their applicability and use for different environments and applications. Some selected representative studies are summarized below. The objective here is to provide a perspective on the state of research and of practice rather than a complete literature review.

An empirical comparison of several classification techniques was reported in Lanubile and Visaggio (4) based on small business applications written by students. They considered discriminant analysis, PCA, logistic regression, classification trees, and so on. Their results indicated that no model was able to discriminate effectively between fault-prone and non-fault-prone components. Denaro et al. (5) applied logistic regression and cross validation by using many metrics combinations from systems produced in industrial environments. Their results are optimistic but cautionary about the cost-effectiveness and practical applicability of such models. Briand et al. (6) studied subsets of metrics, called optimized set reduction, to characterize objects. These subsets form patterns that are used to classify new objects. Tree-based models were presented by Khoshgoftaar and Seliya in Ref. 7. Classification trees were used by Selby and Porter (8) to analyze fault-proneness of components from several NASA software systems. Fenton and Neil (9) provided a critical evaluation of many defect detection models. They suggested the use of Bayesian belief networks for improved performance.

Case-based reasoning classification models were compared by El Eman et al. (10). They evaluated many combinations of the parameters that need to be specified for such models. The data source was a large real-time system, written by professional programmers in a commercial environment. They found that the classification performance was not sensitive to the choice of parameters.

Khoshgoftaar and Seliya (11) give the results of a comparative assessment of logistic regression, case-based reasoning, classification trees, and so on by using data from four releases of a large telecommunication system. They also presented tree-based methods in Ref. 7. They found that the predictive performance of the techniques was significantly different across releases. This observation is consistent with some other studies, indicating that prediction of fault-proneness is influenced by system and data characteristics.

Various studies have investigated the fault-proneness issue for classes in object-oriented development based on class metrics. A study by Basili et al. (12) compared the relative predictive performance of design-based object-oriented metrics. Recently, Zhou et al. (13) reported that design metrics were able to predict low severity faults better than high severity ones in fault-prone classes. Nagappan et al. (14) studied post-release defects in some Microsoft systems and their statistical correlation to complexity metrics. They employed principal component regression to predict the likelihood of defects in the field for new entities. They also noted that the predictors from one project can be useful for similar new projects.

Ma et al. (15) analyzed fault data from five NASA projects for software quality prediction. They employed balanced random forests, classification trees, k-nearest neighbors, and several other machine learning and statistical techniques to compare performance using six measures, including accuracy.

Neural network-based classification models have also been employed by many authors [e.g., Zhang and Tsai (16)]. Two recent studies using radial basis functions, a special type of neural network, are reported by Goel and Shin (17) as well as Shin and Goel (18).

MODEL DEVELOPMENT PROCESS

Classification model development can be seen as an iterative multistep process as shown in Fig. 1. Most techniques follow such a process even though they differ in implementation details. The modeling process requires a thorough understanding of the data, the techniques, and the application environment. The first step involves the selection of metrics data to be used and the undertaking of data preprocessing. In the second step, statistical or other algorithms are used to determine parameters and to evaluate model performance. Multiple candidate models are usually considered. The third step involves model selection and its assessment. Some pertinent issues that occur during the modeling process are briefly described below. Details can be found in Goel and Shin (19), Han and Kamber (1), and Hastie et al. (2).

Data Selection and Preprocessing

Selection of data to be used for modeling is a nontrivial task that requires an understanding of the development environment and the application domain. Using too many or too few metrics is undesirable. Sometimes data about many metrics are available, but all may not be necessary, which can happen when some metrics have interdependencies or when some are irrelevant. Subset selection techniques can be used to reduce the number of metrics for modeling. Preprocessing of data (e.g., data cleansing, normalization, grouping, and transformation) are undertaken at this stage.

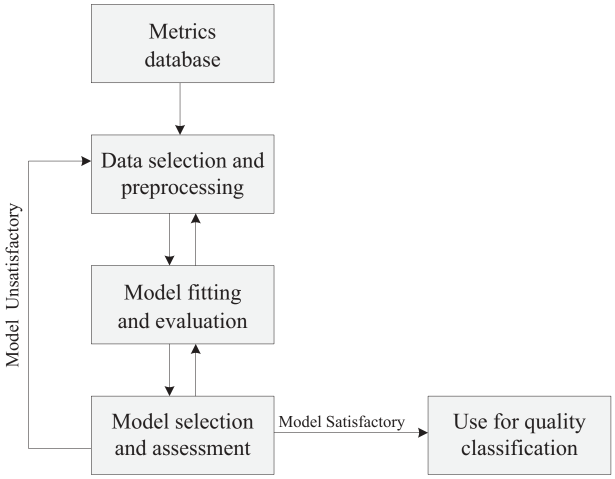

Metrics database
Data selection and preprocessing
Model fitting and evaluation
Model selection and assessment
Model Satisfactory
Use for quality classification
Model Unsatisfactory

Figure 1. Model development process.

Model Fitting and Evaluation

Often, multiple algorithms are available for determining model parameters. Efficiency and accuracy considerations generally dictate the choice of the algorithm to use. The performance of the model is evaluated using criteria such as Type I and Type II errors, overall classification accuracy, and so on. Several models are generally considered and their performance measures are evaluated. A commonly used approach is to divide the data into three groups called training, validation, and test data. Training data is used to fit candidate models. Their performance is evaluated on the validation data and generally the model with the lowest validation error is preferred. Some other common approaches are cross-validation, leave-one-out, and bootstrap (2). If the models are not good, different datasets or modeling techniques are selected.

Model Selection and Assessment

In this step, a preferred model is determined, based on several considerations. In addition to the error measures, model complexity is also an important issue. Generally, a parsimonious model is preferred. Yet, too simple a model may not fit the data well, whereas a complex model may fit it too well and will fail to produce good results on new components. This phenomenon is well known as the underfitting-overfitting or bias-variance dilemma. Sensitivity analyses are performed to evaluate multiple models. The adequacy of the model from its use and applicability perspectives is assessed at this stage. Issues such as comprehensibility and reasonableness are also considered. An appropriate model is then selected based on the above considerations. The performance of the selected model on the test data is used as a measure of its predictive accuracy and usually is considered to be an important criterion to evaluate the usefulness and applicability of a model. If no model is satisfactory or alternate models are desired, the iterative process is repeated.

RADIAL BASIS FUNCTION MODEL

This section presents a recently introduced new technique (17,18) for software quality evaluation using radial basis functions (RBF), a class of advanced mathematical models for function approximation or classification. They have been employed in a wide variety of disciplines, from signal processing to medical diagnosis. The RBF is a nonlinear model that consists of two mappings. In the first, inputs are transformed nonlinearly by the basis functions; in the second, the transformed outputs are weighted linearly to produce the output. Formally, for a mapping $f : R^d \rightarrow R$, the RBF network model can be described as

$$\mathrm{f}(\mathbf{x}) = \sum_{\mathrm{j}=1}^{\mathrm{m}} \mathrm{w_j} \varphi_{\mathrm{j}}(\mathbf{x}) = \sum_{\mathrm{j}=1}^{\mathrm{m}} \mathrm{w_j} \varphi(\|\mathbf{x} - \mu_{\mathrm{j}}\|/\sigma_{\mathrm{j}}) \tag{3}$$

where $\mathbf{x} \in \mathrm{R^d}$ is the input vector and m is the number of basis functions. Also, $\mu_\mathrm{j} \in \mathrm{R^d}$ is the jth basis function center, σ_j's are basis function widths, w_j's are weights, and $\| \cdot \|$ denotes the Euclidean distance. The basis functions $\phi(\cdot)$ here play the role of transfer functions in traditional neural networks, except for the unique feature that their responses to the input vectors are monotonically decreasing or increasing with distance from the centers, and hence the name. In practice, the Gaussian is the most popular basis function because it has attractive mathematical properties of universal and best approximation and its hill-like shape is easy to control with the parameter σ. The mathematical form of the mapping in Equation (3) for the Gaussian case becomes

$$\mathrm{f}(\mathbf{x}) = \sum_{\mathrm{j}=1}^{\mathrm{m}} \mathrm{w_j} \exp(-\|\mathbf{x} - \mu_\mathrm{j}\|^2/2\sigma_\mathrm{j}^2) \qquad (4)$$

A diagrammatic representation showing the details of the above mappings is given in Fig. 2. Here, the input layer takes the metrics data **X** to be used for classification modeling. The hidden layer performs the nonlinear transformations on these data via the *m* basis functions. The output layer produces the classification results y by the weighted responses from the basis functions.

The RBF model is defined completely by the parameter set $P = (m, \sigma, \mu, w)$. Therefore, the RBF design problem is to determine its three m parameters, namely, *m* centers, *m* widths, and *m* weights. Many algorithms have been proposed in the literature to determine these parameters. In a new algorithm, called the SG (Shin–Goel) algorithm, the modeling problem is formulated as follows (17,18,20). First, the algorithm selects the number of basis functions for a given global σ that satisfies a specified measure of model complexity, called representational capability. For a selected (*m*,σ) pair, the corresponding centers are determined next. Finally, the linear parameters *w* are determined by the pseudo-inverse method. This algorithm is purely algebraic and employs only matrix computations. In particular, it avoids random iterations and leads to a consistent and reproducible design.

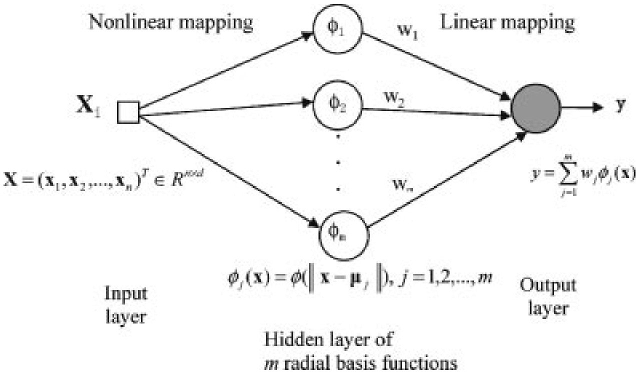

Nonlinear mapping
Linear mapping
ϕ_1
ϕ_2
ϕ_m
w_1
w_2
w_m
$\mathbf{X}_1$
y
$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^T \in R^{n \times d}$
$y = \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$
$\phi_j(\mathbf{x}) = \phi(\|\mathbf{x} - \boldsymbol{\mu}_j\|), j = 1,2,\ldots,m$
Input layer
Output layer
Hidden layer of
m radial basis functions

Figure 2. Radial basis function model structure.

ILLUSTRATIVE EXAMPLE

Data from selected NASA software systems, SEL (21), are used in this section to illustrate the classification model development, evaluation, and selection process. The database contains project and product measure from space-related systems and has been used in many other studies [e.g., Selby and Porter (8), Selby (22), and Shin and Goel (18)]. The RBF model and the SG algorithm described above are employed for developing the classification models in this example.

Data Selection and Preprocessing

The database contains values of several metrics and the number of faults for each component in the form of Equation (1). Only three metrics, the design metrics, are used in this simple example. These metrics are listed in Table 1 along with two statistics, the average and the standard deviation. A module is defined to be fault-prone if the number of defects exceeds five. The design metrics become available early in the software development lifecycle and, therefore, classifiers based on these can be used for early identification of potential fault-prone components. However, as mentioned earlier, metrics selection is an important issue but is not discussed further in this example.

Next, the three metrics were normalized to be in the range [0,1] as follows.

$$Normalized\,Value = \left(\frac{Original\,value - Minimum\,value}{Maximum\,value - Minimum\,value}\right)$$

For model development, evaluation, and selection, half of the components were randomly selected to form the training set. The other half were randomly divided into two equal sets to form the validation and test sets.

Model Fitting and Evaluation

The algorithm described in the previous section was used to develop RBF classifiers. Six fitted models and their classification errors on the training, validation, and test sets are listed in Table 2. It is noted that the training error tends to decrease with increasing model complexity (m) but not monotonically. Because of noise in real-world data, the training error can sometimes increase with m, as is the case here. However, the trend will show decreasing training error as m increases, and the value eventually will become very small when the model complexity gets very high. The validation and test errors first decrease with m

Table 1. List of Metrics

Variable	Description	Average	Standard Devidation
X_1	Function calls from this component	9.51	11.94
X_2	Function calls to this component	3.91	8.45
X_3	Input/Output parameters	27.78	23.37

Table 2. RBF Models and Classification Errors

			Classification Error (%)		
Model	σ	Complexity (m)	Training	Validation	Test
A	0.2	21	23.86	24.62	26.62
B	0.4	10	23.37	24.62	25.13
C	0.6	7	23.12	24.12	25.63
D	0.8	4	23.62	25.63	26.13
E	1.0	4	23.87	26.13	26.13
F	1.2	4	23.62	26.13	26.63

and then increase. Again, the values are not likely to follow a monotonic pattern for data from actual systems, although the trend will be as mentioned above . Plots of these errors versus m and σ as well as versus m are shown in Figs. 3 and 4, respectively. These plots provide useful insight into the behavior of training, validation, and test errors, even for this limited set of data and models.

Model Selection and Assessment

In practice, the model with the smallest validation error is selected from the candidate models. From the data in Table 2 and the plots in Figs. 3 and 4, model C would seem to be the preferred choice. However, other models may also be acceptable, depending on the application. When evaluated on test data, the classification error of this model is 25.63%. This indicates that if model C is used to determine the fault-proneness of new, yet unseen, components, it is likely to make about 25.63% erroneous classifications. From the software engineering perspective, this value is respectable and the model could be considered to be satisfactory.

Sensitive Analysis

The above classifiers were obtained from the design metrics only. To evaluate the effect of metrics selection, classifiers were also developed based on three coding metrics (size, comment lines, and number of decisions) as well as for all six metrics. The test errors for these cases were 24.92% and 23.86%, respectively. From a software engineering viewpoint, they are all satisfactory values. Other error measures could be used for model evaluation and selection, as well as different sensitivity analyses could also be pursued. Such choices depend on the software development and application environment.

CONCLUDING REMARKS

This article discusses the importance of classification models for early identification of fault-prone software components to increase testing efficiency and effectiveness. Some commonly used modeling techniques and selected classification studies are summarized. A generic modeling process is presented and some issues that develop during this process are highlighted. A recently introduced classification technique based on radial basis functions is described. A case study is discussed to illustrate the model development, evaluation, and selection process. In conclusion, it should be noted that modeling from data is a difficult problem, which is especially true in software engineering applications.

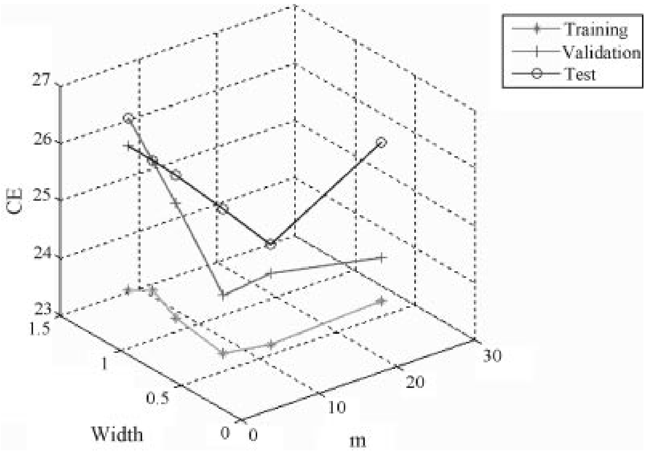


Figure 3. Classification errors versus (m-σ).

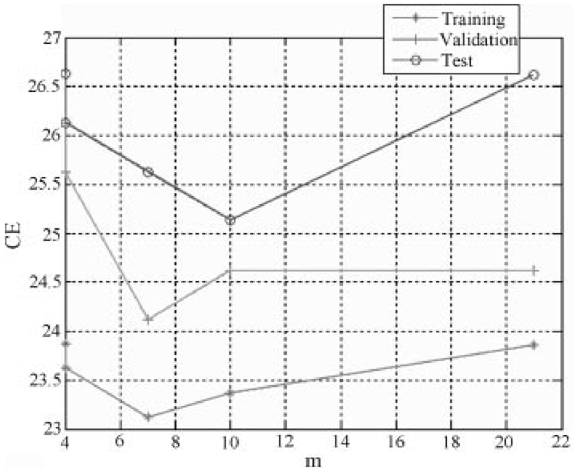


Figure 4. Classification errors versus m.

BIBLIOGRAPHY

1. J. Han and M. Kamber, *Data Mining*, 2nd ed., San Francisco, CA: Morgan Kauffman, 2006.
2. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Interference*, New York: Springer, 2001.
3. J. R. Quinlan, *C4.5 Programs for Machine Learning*, San Francisco, CA: Morgan Kaufmann, 1993.
4. F. Lanubile and G. Visaggio, Evaluating predictive quality models derived from software measures: lessons learned, *J. Syst. Softw.,* **38**: 225–234, 1997.
5. G. Denaro, M. Pezze, and S. Morasca, Towards industrial relevant fault-proneness models, *Internat. J. Soft. Engineer., Knowledge Engineer.,* **14**: 1–23, 2003.
6. L. Briand, V. Basili, and C. Hetmanski, Developing interpretable models with optimized set reduction for identifying high-risk components, *IEEE Trans. Soft. Enginee.,* **19**: 1028–1044, 1993.
7. T. M. Khoshgoftaar and N. Seliya, Tree-based software quality estimation models for fault prediction, *METRICS 2002, IEEE Computer Society*, pp.203–214, 2002.
8. R. W. Selby and A. A. Porter, Learning from examples: generation and evaluation of decision trees for software resource analysis, *IEEE Trans. Soft. Engineer.* **14**(12): 1743–1757, 1998.
9. N. Fenton and M. Neil, A critique of software defect prediction models, *IEEE Trans. Soft. Engineer.,* **25**: 675–689, 1999.
10. K. El Eman, et al., Comparing case-based reasoning classifiers for predicting high risk software components, *J. Syst. Soft.,* **55**: 301–320, 2001.
11. T. Khoshgoftaar and N. Seliya, Comparative assessment of software quality classification techniques: an empirical case study, *Empirical Soft. Engineer.,* **9**: 229–257, 2004.
12. V. R. Basili, L. C. Briand and W. L. Melo, A validation of object-oriented design metrics as quality indicators, *IEEE Trans. Soft. Engineer.,* **22**: 751–761, 1996.
13. Y. Zhou et al. Empirical analysis of object-oriented design metrics for predicting high and low severity faults, *IEEE Trans. Soft. Engineer.*, **32**: 771–789, 2006.
14. N. Nagappan et al., Mining metrics to predict component failures, *Proc. 28th International Conference on Software Engineering*, Shanghai, China, pp. 452–460, 2006.
15. Y. Ma, L. Guo, and B. A. Cukic, Statistical framework for the prediction of fault-proneness, in D. Zhang, and J.J. Tsai, (eds.), *Advances in Machine Learning Applications in Software Engineering*, Hershey, PA: Idea Group, 2007.
16. D. Zhang and J. J. P. Tsai, *Machine Learning Applications in Software Engineering*, Singapore: World Scientific, 2005.
17. A. Goel and M. Shin, Radial basis functions: an algebraic approach (with Data Mining Applications), *Tutorial at the 15th European Conference on Machine Learning, (ECML 2004)*, Pisa, Italy, 2004.
18. M. Shin and A. L. Goel, Modeling software component criticality using a machine learning approach, in T. G. Kim (ed.), *Artif. Intell. Simulat.*, Lecture Notes in Computer Science, New York: Springer, pp. 440–448. 2004.
19. A. L. Goel and M. Shin, Tutorial on software models and metrics, *Internat. Conf. on Software Engineering*, Boston, MA, 1997.
20. M. Shin and A. L. A. Goel, Empirical data modeling in software engineering using radial basis functions, *IEEE Trans. Soft. Engineer.,* **28**: 567–576, 2000.
21. Software Engineering Lab. Database Organization and User's Guide, SEL-81-102, NASA/GSFC, Greenbelt. MD, 1983.
22. R. W. Selby, Enabling reuse – based software development for large – scale systems, *IEEE Trans. Soft. Engineer.,* **31**: 495–510, 2005.

MIYOUNG SHIN
Kyungpook National University
Daegu, Korea
AMRIT L. GOEL
Syracuse University
Syracuse, New York

SOFTWARE QUALITY MODELING AS A RELIABILITY TOOL

INTRODUCTION

Software engineering practitioners agree that a software quality- and reliability-centric focus is ideal for most software project development efforts. A specific definition of software quality, however, is often a point of debate because "software quality" encompasses many different attributes of software with varying relative importance (1). For example, one may argue that conformance to requirements and specifications must govern software quality, whereas another may argue that ease of maintenance, low complexity, and good documentation are equally, if not more, important. Although software quality can often have a qualitative or subjective evaluation, software reliability is typically measured using objective criteria. In essence, software reliability is one of the multiple software quality factors, such as understandability, maintainability, usability, and testability.

The general aim of software quality and reliability methods is to obtain a defect-free, dependable, and functional software system. This article provides a brief overview of software quality and reliability tools[1] and presents a detailed case study on software quality modeling by using decision tree models. Software quality is the burden of all stakeholders involved in the project, including management, developers, and customers. Hutton (2) defines "must haves" for a successful outcome of a software quality improvement initiative, which include (*1*) a compelling reason for change, (*2*) suitable sponsors, (*3*) an informed commitment of sponsors, and (*4*) a change agent or "champion." Having effective software quality tools has been the subject of core software engineering research and practice, and suggested standards such as CMMI (3,4) and ISO 9001 (5) explicitly state requirements that must be met for a high quality software product. Some commonly used software quality improvement methods include formal inspections, walkthroughs, defect tracking, software measurement-based analysis, and testing (1,6). The advent and acceptance of agile software engineering has introduced new issues in software quality engineering, some of which are aptly discussed by Ming et al. (7) and Mnkandla et al. (8).

Software reliability engineering has been introduced with various models and tools over the last two decades (9–11). The growth of software reliability engineering research is about 35% per year (12), attributed largely to technical conferences and journals dedicated to this maturing field. Operational profiles and their use for software reliability and testing was introduced initially by Musa (12). Different types of software reliability models are options depending on available project data and the development phase. A good overview of different types of software reliability models is provided by Asad et al. (9), in which the authors lay out a logical approach to selecting a specific software reliability model among the over-thirty models surveyed. The different models were classified based on early prediction models, software reliability growth models, input domain-based models, architecture-based models, and hybrid reliability models. The selection criteria used by Asad et al. [9] included attributes such as required model input, required user output, structure and nature of project, data validity, and the development and testing processes. The popularity of object-oriented development and increasing use of service-oriented architectures has fueled research in reliability analysis exhaustive literature review is out of scope for this article. based on software architecture. A recent study provides good insight into the state of the art of software architecture-based reliability analysis (10).

Our research focus has included software quality models as effective tools for targeted and resource-aware software quality improvement. A software quality estimation model is typically trained with known software measurement and defect data of a previously developed system release or similar project (13–16). The model then is applied to predict unknown defect data of program modules currently under development. Based on such a quantitative analysis, software quality modeling can generally be achieved by two commonly used approaches: software defect prediction and software quality classification. The former involves estimating the expected number of defects in the software, whereas the latter involves grouping similar program modules based on their expected quality, for example, fault-prone (*fp*) or not-fault prone (*nfp*). We focus on the software quality classification models because they are more appealing from a practical point of view. In addition to software quality modeling, classification models have been studied for other software engineering problems, such as requirements engineering (17) and component-based software engineering (18). Although different methods have been investigated for software quality classification, decision trees models have gained popularity among practitioners because of their comprehensible model structure, ease in model interpretation, and quick rule extraction for database retrievals (19–23).

The empirical case study presented compares five decision tree methods for software quality modeling, and they include, Classification and Regression Trees (*CART*)(21), *S-PLUS* (*SPL*), Sprint-Sliq (*SPT*) (22), C4.5 (23), and Treedisc (*TD*) (24). The software measurement and defect data is obtained from multiple releases of a very large telecommunications software system (21). The different models are evaluated for decision tree structure, model complexity, and software quality prediction performance. In addition, the decision trees are evaluated for their expected costs of misclassification at different cost ratios. A relative ranking of the five models is obtained based on statistical testing using ANOVA models and

[1] As exhaustive literature review is outside the scope of this article.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

multiple- pairwise hypothesis testing. It is shown that with respect to expected costs of misclassification, the *CART* model yielded better results at lower cost ratios, where as the *SPT* model yielded better results at higher cost ratios.

The remainder of this article summarizes the decision tree models investigated in our study and details our empirical case study, including software system description, modeling approach, and empirical results. Finally, we summarize our work and include some directions for future work.

DECISION TREE-BASED CLASSIFICATION MODELS

The *Classification and Regression Trees* (*CART*) (25) system is a widely used decision tree model with many applications to data mining and data preprocessing. The algorithm searches for questions that split tree nodes into relatively homogeneous child nodes, such as a high-(or low-)-risk group. The nodes become more homogeneous as the tree evolves, identifying important segments. The initial tree is grown to the maximum possible size and then is pruned backward using cross validation and cost complexity. Certain model parameters, such as number of observations in leaf nodes and depth of tree, can be controlled to yield different models (21).

The *SPL* system combines an intuitive graphical user interface with an extensive data analysis environment for advanced data mining, including regression tree models (26). At the core of the *SPL* system is *S,* the language designed specifically for data visualization and exploration, statistical modeling, and programming with data. Limited to processing only numerical data, the regression tree is constructed by recursive binary partitioning of the training data. A generalized classification rule is used to label each leaf node after the regression tree is built. Model parameters, such as minimum node size and minimum node deviance, can be controlled to obtain the desired pruned tree (20).

The *C4.5* learner is an inductive supervised learning system, which employs decision trees to represent a quality model (27). It consists of four principal programs: decision tree generator, production rule generator, decision tree interpreter, and production rule interpreter. In contrast to *CART* and *SPL*, the *C4.5* classifier can entertain non-numeric data types during modeling and provide a non binary partitioning of data at decision nodes. Tree pruning can be controlled by modeling parameters, including minimum node size and pruning percentage (23).

The *TD* algorithm is an SAS macro implementation of the modified chi-square automatic interaction detection algorithm (19). A predictor variable is selected to be the variable that is most significantly associated with the dependent variable according to a chi-squared test of independence in the contingency table. The algorithm requires special preprocessing of the data sets and is suitable for ordinal predictors with many categories. A decision tree is built by recursively partitioning the data set until a stopping criterion is satisfied, such as minimum node size and p-value.

The *SPT* algorithm the (scalable parallelizable induction of decision trees-supervised learning in quest algorithm also known as the *sprint-Sliq* algorithm) can be used to build classification trees that can analyze both numeric and categorical attributes (28,29). As a modified version of the *CART*, it uses a pruning technique based on the minimum description length principle. The IBM Intelligent Data Miner that implements the *SPT* algorithm, was used in our study (22). The *Gini* index is the default for evaluating goodness-of-fit of possible splits at a decision node.

EMPIRICAL CASE STUDY

Software System and Measurements

The case study data was collected from a very large legacy-embedded telecommunications system (abbreviated as LLTS) over four successive releases. Using the procedural development paradigm, the software was written in a high-level language similar to Pascal and was maintained by professional programmers in a large organization. A software module was considered as a set of related source-code files. Fault data was collected at the module level by the problem reporting system and consisted of post release faults discovered by customers during system operations. Faults were recorded only if their discovery resulted in changes to the source code of the respective module. Preventing discovery of faults after deployment was a high priority for the developers because visits to customer sites involved extensive consumption of monetary and other resources.

Configuration management data analysis identified software modules that were unchanged from the prior release. Fault data collected from the problem reporting system were tabulated into problem reports, and anomalies were resolved. Because of the nature of the system being modeled, for example, a high-assurance system, the number of modules associated with post release faults were very few compared with modules with no faults. Two clusters of modules were identified: unchanged and updated. The updated modules consisted of those that were either new or had at least one update to their source code since the prior release. Among the unchanged modules, almost all of them had no faults and, therefore, were not considered for modeling purposes.

We selected updated modules with no missing data in relevant variables. These updated modules had several million lines of code, with a few thousand of these modules in each system release. The number of updated modules (that remained after unchanged modules or those with missing data were removed) that were considered for each of the four releases are: 3649 for Release 1, 3981 for Release 2, 3541 for Release 3, and 3978 for Release 4. A module was considered as *nfp* (non fault-prone) if it had no post release faults and *fp* (fault-prone)otherwise.

The proportions of modules with no faults among the updated modules of Release 1, Release 2, Release 3, and Release 4 were respectively 93.7%, 95.3%, 98.7%, and 97.7%. The corresponding proportions of modules with one or more faults were 6.3%, 3.9%, 1.0%, and 2.1%. The number of modules with no faults generally increased over successive releases, as is expected for a legacy system in an environment similar to LLTS. This increase is because the development process for such a high-assurance software system generally tends to improve with time. However, for another system, other factors such as new development staff or changes in system functionality could increase the number of faults and introduce additional *fp* modules. Software quality models for LLTS were trained with Release 1 program modules, whereas the latter three releases were used as test data for model evaluation.

Software data collection for LLTS involved recording measurements using the EMERALD software metrics analysis tool (21). Preliminary analysis selected metrics appropriate for our modeling purposes and included 24 product metrics, 14 process metrics, and 4 execution metrics. The process metrics were not used in this study because we focus on software quality modeling and prediction after the coding (implementation) phase and before system tests. The case study, hence, consists of 28 independent variables that are used to predict a program module as *fp* or *nfp*. In Table 1, the top 24 attributes are product metrics based on call graph, control flow graph, and statement metrics, whereas the bottom four attributes are execution metrics. The *USAGE* metric was approximated by deployment data on a previous system release, whereas execution times were measured in a laboratory setting with different simulated workloads.

Modeling and Evaluation

The decision tree models are trained using a common model building and selection process. The Release 1 program modules are used as training data, whereas the Release 2 data set is used for model selection. For each decision tree method, different models are built by varying the respective model parameters. In the case of *CART*, model selection was based on 10-fold cross validation with Release 1. However, a close inspection of the different *CART* models indicated that the same model would be selected if Release 2 was used for model selection. This situation alleviates any threat to empirical validity from a non uniform model-selection process. We note that cross validation was not explicitly available for the other four decision tree classifiers. A generalized classification rule is used with all five methods for classifying modules as *fp* or *nfp*(19–22).

A Type I error occurs when a *nfp* module is misclassified as *fp,* whereas a Type II error occurs when a *fp* module is misclassified as *nfp*. Although it is beneficial to detect, before operations, as many *fp* modules as possible, the usefulness of a software quality classification model is affected by its Type I error rate. A model with a very low Type II error rate and a very high Type I error rate is not practical because given the large number of modules predicted as *fp* (many of them are actually *nfp*), deploying the limited quality improvement resources will pose a difficulty. The model-selection strategy of our study consists of finding the preferred balance of equality between the two error rates, with Type II being as low as possible. Such a strategy was used based upon the recommendation of the software quality engineers and the project management team of the system being modeled. A different application

Table 1. Software Metrics

Symbol	Description
CALUNQ	Number of distinct procedure calls to others.
CAL2	Number of second and following calls to others
CNDNOT	Number of arcs that are not conditional arcs
IFTH	Number of non loop conditional arcs, i.e., if–then constructs
LOP	Number of loop constructs
CNDSPNSM	Total span of branches of conditional arcs. The unit of measure is arcs
CNDSPNMX	Maximum span of branches of conditional arcs
CTRNSTMX	Maximum control structure nesting
KNT	Number of knots, which is where arcs cross because of a violation of structured programming principles
NDSINT	Number of internal nodes, i.e., not an entry, exit, or pending node
NDSENT	Number of entry nodes
NDSEXT	Number of exit nodes
NDSPND	Number of pending nodes, i.e., dead code segments
LGPATH	Logarithm (base two) of the number of independent paths
FILINCUQ	Number of distinct include files
LOC	Number of lines of code
STMCTL	Number of control statements
STMDEC	Number of declarative statements
STMEXE	Number of executable statements
VARGLBUS	Number of global variables used
VARSPNSM	Total span of variables
VARSPNMX	Maximum span of variables
VARUSDUQ	Number of distinct variables used
VARUSD2	Number of second and following uses of variables
USAGE	Deployment percentage of the module
RESCPU	Time of an average transaction on a system serving consumers
BUSCPU	Time of an average transaction on a system serving businesses
TANCPU	Time of an average transaction on a tandem system

may consider another preferred balance between the error rates.

The cost of a Type I misclassification, C_I, is the effort wasted (ineffective reviews) on an *nfp* module, and the cost of a Type II misclassification, C_{II}, is the lost opportunity to correct faults early for an *fp* module. The disparate costs of misclassifying an *fp* and *nfp* module poses a problem in evaluating competing software quality models. Hence, a unified cost metric, normalized expected cost of misclassification (NECM), is proposed and used in our case study (30). The NECM metric is a function of the two error rates, the prior probabilities of the two groups, and the cost ratio $\frac{C_{II}}{C_I}$. The cost ratio used during modeling depends on the project type and application domain, for example, a mission-critical software will have a higher C_{II} compared with an in-house business application. Moreover, the actual cost ratio for a project cannot be determined until after development and deployment. Hence, for a given application domain, one may consider a range of cost ratios that include representative and basic values. We compute the NECM values for our case study system at different cost ratios. If π_{fp} is the prior probability of *fp* modules, π_{nfp} is the prior probability of *nfp* modules, $Pr(fp|nfp))$ is the proportion of *nfp* classified as *fp*, and $Pr(nfp|fp)$ is the proportion of *fp* classified as *nfp*, then NECM is given by $Pr(fp|nfp)\pi_{nfp} + \frac{C_{II}}{C_I} Pr(nfp|fp)\pi_{fp}$.

In a comparative study such as ours, it is important to know whether performance differences between models are statistically significant. We use the two-way ANOVA (analysis of variance) randomized complete block experimental design with five factor treatments (i.e., decision tree classifiers) and three blocks (i.e., Release 2, 3, and 4 test data sets) (31). We are interested in observing if the different tree models and the different system releases are significantly different from their respective counterparts. The response variable is the NECM metric of the five decision trees computed for the three test data releases at a given cost ratio. The p-values obtained by the ANOVA design models indicate the significance of the differences among the five models and among the three test data sets. Multiple-pairwise comparison methods facilitate more detailed insight into the differences of means and provide a statistical technique for comparing two methods at a time. We use the Bonferroni's multiple-pairwise comparison equation (32) in conjunction with hypothesis testing using the p-vsdue approach. For two competing models A and B, the null (H_o) and alternate (H_A) hypotheses for the multiple-pairwise comparison are $H_0 : NECM_A \geq NECM_B$ and $H_A : NECM_A < NECM_B$.

Results and Discussion

The five decision tree models are compared with respect to their *complexity, robustness*, and *prediction* performances. For a given tree model, the number of leaf nodes (*TN*) and the number of unique software attributes used (*IVUT*) express the model's complexity. Robustness is evaluated in terms of model stability, for example, how consistently the preferred balance (between Type I and Type II error rates) of equality is maintained across the different releases. A robust model is important because the software quality assurance team typically prefers a "no surprise" software reliability engineering policy. Classification performance is evaluated in terms of misclassification error rates and NECM values. In our study, we consider a large range of cost ratios, for example, $\frac{C_{II}}{C_I} = \{1, 2, 3, 4, 5, 10, 15, 20, 25, 50, 100, 200, 350, 500\}$,to compute NECM values. However, for a high-assurance system such as LLTS, a range of 20 to 100 is most practical and likely.

Given the limited space considerations, we graphically illustrate only two of the five decision trees, for example, *CART* (Fig. 1) and *SPT* (Fig. 2). The other three models are presented in our respective prior works (19,20,23). The *CART* model ($TN = 3; IVUT = 2$) depicted lowest complexity, whereas the *SPT* model ($TN = 15; IVUT = 11$) showed highest complexity. Both models selected FILINCUQ, the number of unique include files, as the most important (i.e., root node) software attribute for LLTS. The five models would be ranked in descending order as *CART*, *SPL*, *TD*, *C4.5*, and *SPT* if based solely on model complexity. In our study, we observed that *TN* and *IVUT* were generally proportional, for example, a larger tree used more unique software attributes.

The misclassification error rates and normalized expected costs of misclassification for the five models are shown in Tables 2 and 3, respectively. The error rates are typical for the given system. The *NECM* values are only shown for cost ratios 1, 2, 5, 10, 25, 50, and 100, again because of limited space considerations. The *SPT* and *TD* models provide the best consistency in the preferred balance between the Type I and Type II error rates, whereas the *CART* model yielded the poorest consistency in the preferred balance. The average balance across Releases 2, 3, and 4 for *SPT*, *TD*, and *CART* are 2.84%,

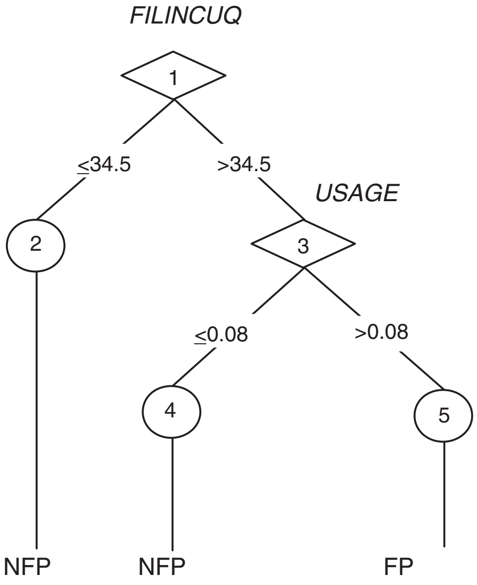


Figure 1. *CART* classification tree model.

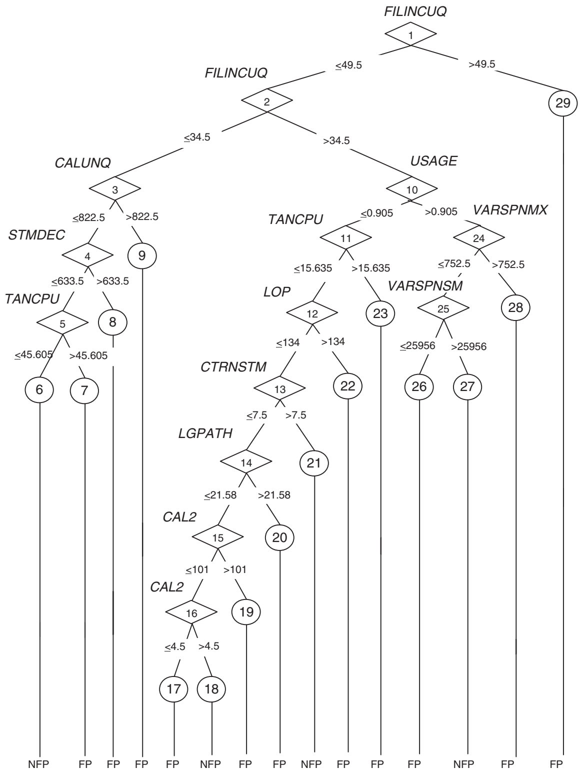


Figure 2. *SPRINT-SLIQ* classification tree model.

Table 2. Misclassification Rates for Models

Method	Rel.	Type 1 %	Type 11 %	Overall %
	2	24.47	26.84	24.58
SPT	3	25.38	20.83	25.32
	4	28.48	26.88	28.44
	2	31.67	23.28	31.27
CART	3	30.30	14.89	30.10
	4	35.64	22.82	35.34
	2	25.08	26.46	25.15
SPL	3	26.67	21.28	26.60
	4	32.24	21.74	32.00
	2	25.08	29.10	25.27
TD	3	28.91	25.53	28.87
	4	28.33	28.26	28.33
	2	22.94	29.10	23.23
C4.5	3	25.44	21.27	25.39
	4	31.34	28.26	31.27

Table 3. NECM Values for Models

		Cost Ratio, $\frac{C_{II}}{C_I}$						
Method	Rel.	1	2	5	10	25	50	100
	2	0.246	0.263	0.314	0.398	0.652	1.075	1.920
SPT	3	0.251	0.264	0.303	0.369	0.566	0.894	1.550
	4	0.284	0.301	0.352	0.436	0.690	1.114	1.960
	2	0.311	0.326	0.370	0.443	0.663	1.030	1.763
CART	3	0.293	0.303	0.331	0.378	0.518	0.753	1.222
	4	0.348	0.363	0.406	0.478	0.693	1.053	1.772
	2	0.252	0.268	0.318	0.402	0.652	1.068	1.902
SPL	3	0.263	0.277	0.317	0.384	0.585	0.920	1.591
	4	0.316	0.329	0.371	0.439	0.644	0.987	1.672
	2	0.253	0.272	0.327	0.418	0.693	1.152	2.068
TD	3	0.287	0.303	0.351	0.432	0.673	1.075	1.879
	4	0.283	0.301	0.354	0.443	0.711	1.156	2.046
	2	0.233	0.252	0.307	0.398	0.673	1.132	2.048
C4.5	3	0.252	0.265	0.305	0.372	0.573	0.908	1.578
	4	0.311	0.329	0.383	0.472	0.739	1.184	2.074

2.49% and 12.21%, respectively. These values were computed by averaging the absolute differences between the Type I and Type II error rates of a given model across the three test data sets. For *SPT*, this average is computed as $\{|24.47 - 26.84| + |25.38 - 20.83| + |28.48 - 26.88|\}/3 = 2.84$. In terms of model robustness, the five models would be ranked in descending order as *TD*, *SPT*, C4.5, *SPL*, and *CART*.

A graphical illustration of the influence of cost ratios on the significance of the differences between the methods is shown in Fig. 3. The figure plots *p*-values against the different cost ratios. The details of our ANOVA models, including degrees of freedom, mean squares, sums of squares, and *F* statistic, are not shown to maintain simplicity. The ANOVA models revealed that for all cost ratios considered, the system releases are significantly different (at $\alpha = 0.05$) from each other. In addition, the five decision tree methods were significantly different (at $\alpha = 0.10$) from each other for most cost ratios, except 10 (*p*-value = 0.3070), 15 (*p*-value = 0.3640), 20 (*p*-value = 0.2520), and 25 (*p*-value = 0.1570). The spike in the middle portion of the curve (Fig. 3) indicates poor significance of the difference in NECM values of the tree models.

In our multiple-pairwise comparisons, each tree model is compared with the other four models using a one-tailed pairwise comparison. For example, *CART* is compared with *SPL*, *SPT*, *TD*, and *C4.5* individually. Thus, for each pair of methods A and B, we have two comparisons: *"Is A better than B?"* and *"Is B better than A?"*. The *p*-values computed from the multiple-pairwise comparisons for the limited set of cost ratios is shown in Table 4. The table can be viewed as a matrix, for example, each pair of methods forms a comparison; methods in the second column are compared with the methods listed in the headings of the tables. Methods are not compared to themselves, and this situation is indicated by a "*" in the table. Because five methods comparise our comparative study, 20 comparisons exist for each cost ratio.

To infer the performance rank order of the methods, we present the details only for cost ratio 2. The following steps break down the inference process of how the rank order for the five models is obtained.

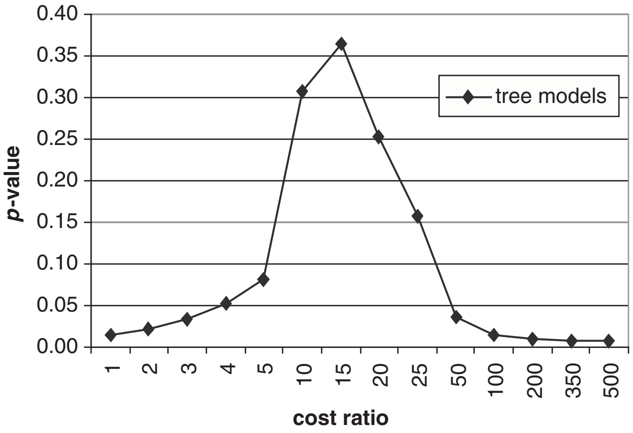


Figure 3. *p*-values.

Table 4. Multiple-pairwise Comparison *p*-values

$\frac{C_{II}}{C_I}$		*SPT*	*CART*	*SPL*	*TD*	*C4.5*
1	*SPT*	*	0.0041	0.4611	0.5983	1.0000
	CART	1.0000	*	1.0000	1.0000	1.0000
	SPL	1.0000	0.0259	*	1.0000	1.0000
	TD	1.0000	0.0195	1.0000	*	1.0000
	C4.5	1.0000	0.0071	0.7972	1.0000	*
2	*SPT*	*	**0.0062**	**0.5347**	**0.5115**	**1.0000**
	CART	**1.0000**	*	**1.0000**	**1.0000**	**1.0000**
	SPL	**1.0000**	**0.0348**	*	**1.0000**	**1.0000**
	TD	**1.0000**	**0.0365**	**1.0000**	*	**1.0000**
	C4.5	**1.0000**	**0.0118**	**0.9749**	**0.9383**	*
5	*SPT*	*	0.0228	0.8109	0.3399	1.0000
	CART	1.0000	*	1.0000	1.0000	1.0000
	SPL	1.0000	0.0877	*	1.0000	1.0000
	TD	1.0000	0.2329	1.0000	*	1.0000
	C4.5	1.0000	0.0581	1.0000	0.7937	*
10	*SPT*	*	0.1838	1.0000	0.2165	0.9172
	CART	1.0000	*	1.0000	1.0000	1.0000
	SPL	1.0000	0.3489	*	0.4075	1.0000
	TD	1.0000	1.0000	1.0000	*	1.0000
	C4.5	1.0000	0.5778	1.0000	0.6669	*
25	*SPT*	*	1.0000	1.0000	0.1434	0.7397
	CART	1.0000	*	1.0000	0.0763	0.4249
	SPL	1.0000	1.0000	*	0.0858	0.4729
	TD	1.0000	1.0000	1.0000	*	1.0000
	C4.5	1.0000	1.0000	1.0000	0.5868	*
50	*SPT*	*	1.0000	1.0000	0.1376	0.7013
	CART	0.2453	*	0.7125	0.0101	0.0527
	SPL	0.9530	1.0000	*	0.0431	0.2408
	TD	1.0000	1.0000	1.0000	*	1.0000
	C4.5	1.0000	1.0000	1.0000	0.5983	*
100	*SPT*	*	1.0000	1.0000	0.1439	0.6982
	CART	0.0762	*	0.3442	0.0037	0.0168
	SPL	0.7108	1.0000	*	0.0315	0.1673
	TD	1.0000	1.0000	1.0000	*	1.0000
	C4.5	1.0000	1.0000	1.0000	0.6249	*

1. From comparisons *SPT* vs. *CART* ($p = 0.0062$), *CART* vs. *SPT* ($p = 1.0000$), *SPT* vs. *SPL* ($p = 0.5347$), *SPL* vs. *SPT* ($p = 1.0000$), *SPT* vs. *TD* ($p = 0.5115$), *TD* vs. *SPT* ($p = 1.0000$), *SPT* vs. *C4.5* ($p = 1.0000$), and *C4.5* vs. *SPT* ($p = 1.0000$), we observe that *SPT* is better than all methods except *C4.5*. Therefore, *SPT* will be ranked either first or second.
2. From comparisons *C4.5* vs. *CART* ($p = 0.0118$), *CART* vs. *C4.5* ($p = 1.0000$), *C4.5* vs. *SPL* ($p = 0.9749$), *SPL* vs. *C4.5* ($p = 1.0000$), *C4.5* vs. *TD* ($p = 0.9383$), *TD* vs. *C4.5* ($p = 1.0000$), *C4.5* vs. SPT ($p = 1.0000$), and *SPT* vs. *C4.5* ($p = 1.0000$), we observe that *C4.5* is better than all the methods except *SPT*. Therefore, *C4.5* will be ranked either first or second.

   Given the intermediate results from Steps 1 and 2, we can conclude that the first two in the rank order are *SPT* and *C4.5*
3. From comparisons *TD* versus *SPL* ($p = 1.0000$) and *SPL* versus *TD* ($p = 1.0000$), we conclude that *SPL* and *TD* are similar to each other. However, looking at comparisons *SPT* versus *SPL* ($p = 0.5347$), *SPT*

versus *TD* ($p = 0.5115$), *C4.5* versus *SPL* ($p = 0.9749$), and *C4.5* versus *TD* ($p = 0.9383$), we observe that *SPL* performs relatively better than *TD* when both are compared against *SPT* and *C4.5*. Hence, *SPL* will be placed before *TD*.

4. Based on comparisons *SPL* versus *CART* ($p = 0.0348$), *CART* versus *SPL* ($p = 1.0000$), *TD* versus *CART* ($p = 0.0365$), and *CART* versus *TD* ($p = 1.0000$), *SPL* and *TD* are both better than *CART*.

   Given the intermediate results from Steps 3 and 4, we can conclude the following partial rank order: *SPL*, *TD*, and *CART*.
5. Combining the partial rank order deduction in Step 4 with the knowledge of the first two (deduction of Step 2) in the rank order, the final order for cost ratio 2 is *SPT*, *C4.5*, *SPL*, *TD*, and *CART*.

The rank orders for different cost ratios are shown in Table 5, where the notation $\geq$ signifies that the method on the left side is either better than or similar to the method on right side. At lower cost ratios ($\frac{C_{II}}{C_I} \leq 15$), *SPT* performs the best, whereas *CART* performs the worst. For cost ratio 20, *SPL* performs the best, whereas *TD* performs the worst. For cost ratios between 25 and 500, *CART* performs the best, whereas *TD* performs the worst. The *SPT* model is more robust to changes in the cost ratios. Thus, based on the performance order presented in Table 5, we conclude that for lower cost ratios *SPT* should be preferred, whereas for higher cost ratios *CART* should be preferred.

In empirical software engineering, threats to internal validity are unaccounted influences that may affect case study results. To be credible, the subject of an empirical study must be a system that is (33): (*1*) developed by a group rather than an individual, (*2*) developed by professionals rather than students, (*3*) developed in an industrial environment rather than an artificial setting, and (*4*) large enough to be comparable to real industry projects. The software system investigated in this study meets all these requirements. In the context of this study, poor estimates can be caused by a wide variety of factors, including measurement errors while recording software metrics, errors in model selection, and the presence of outliers and noise in the training data. Measurement errors are inherent to the data collection effort. A common model-building and model-selection approach has been adopted, and the statistical analysis was performed by only one skilled person to keep modeling errors to a minimum. Threats to external validity are conditions that limit generalization of case study results. As our results are based on analysis of the LLTS data set, another system may yield different conclusions. However, the comparative modeling approach would be the same.

Table 5. Performance Order of Tree Models

$\frac{C_{II}}{C_I}$	Performance Order
1	$SPT \geq C4.5 \geq TD \geq SPL \geq CART$
2	$SPT \geq C4.5 \geq SPL \geq TD \geq CART$
5	$SPT \geq C4.5 \geq SPL \geq TD \geq CART$
10	$SPT \geq SPL \geq C4.5 \geq TD \geq CART$
25	$CART \geq SPL \geq SPT \geq C4.5 \geq TD$
50	$CART \geq SPL \geq SPT \geq C4.5 \geq TD$
100	$CART \geq SPL \geq SPT \geq C4.5 \geq TD$

CONCLUSION

An important requirement for a high-quality and dependable software product is the availability and application of effective tools for software quality and reliability improvement. In this article, software quality estimation models have been used to provide insight into the reliability of software systems and, thereby, control the software development process to minimize software failures. Many software applications involve the use of software in safety-critical situations, which warrant the need to develop and quantify effective measures and models of software quality. The timely prediction of faulty components, before their operations, can enable software quality assurance teams to target enhancement efforts only to the needed areas.

Tree-based classification modeling is a simple yet efficient technique that facilitates such a prediction. We compared existing tree-based classification techniques with respect to their performance accuracy, model structure, and model complexity. The modeling methods we compared are *CART*, *S-PLUS*, *Sprint-Sliq*, *C4-5*, and *Treedisc*. These methods were used to build software quality models that predict program modules as either *fp* or *nfp*. The case study used data collected over four successive releases of a very large telecommunications system. A common model building, selection, and evaluation process is used for all the decision tree classifiers. A two-way ANOVA randomized block experimental design analysis examines (*1*) whether the decision trees perform significantly differently than each other and (*2*) whether the different system releases were different from each other.

We observe that in the context of classification accuracy, obtained preferred balance, and model stability, the *Sprint-Sliq* classification model is generally a better method as compared with the other four decision tree models. However, with respect to model interpretation and model complexity, *CART* and *S-PLUS* are generally better methods. We note that results of this study may not be applicable to another software system because the models were built using software metrics and defect data of the LLTS software. However, the modeling and evaluation process generally remain unchanged when applied for software quality modeling of another system. Hence, future work may include investigating a similar comparative study with software metrics data from another software system.

REFERENCES

1. P. A. Jansma, When management gets serious about managing software, *Proc. IEEE Aerospace Conference*, March 2005, pp. 4366–4382.
2. D. H. Hutton, *The Change Agents Handbook: A Survival Guide for Quality Improvement Champions*. Milwaukee, WI: ASQ Quality Press, 1994.

3. M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement*, 2nd ed.The SEI Series in Software Engineering. Addison Wesley, 2006.
4. B. Curtis, W. E. Hefley, and S. A. Miller, *The People Capability Maturity Model: Guidelines for Improving the Workforce*, 1st ed., The SEI Series in Software Engineering. Addison Wesley, 2001.
5. M. C. Paulk, A comparison of iso 9001 and the capability maturity model for software, Technical Report CMU/SEI-94-TR-12, ESC-TR-94-12. Carnegie Mellon University, Software Engineering Institute, July 1994.
6. J. A. McDonough, Template for software quality management for department of defense programs, *Proc. IEEE National Aerospace and Electronics Conference, NAECON*, Vol. 3, Dayton, OH, May. 1990, pp. 1281–1283.
7. H. Ming, J. Verner, Z. Liming, and M. A. Babar, Software quality and agile methods, *Proc. 28th International Computer Software and Applications Conference, COMPSAC*, Vol. **1**, 2004, pp. 520–525.
8. E. Mnkandla and B. Dwolatzky, Defining agile software quality assurance, *Proc. International Conference on Software Engineering Advances*, Tahiti, Oct. 2006, pp. 36–43.
9. C. A. Asad, M. I. Ullah, and M. J. Rehman, An approach for software reliability model selection, *Proc. 28th International Computer Software and Applications Conference, COMPSAC*, vol. **1**, 2004, pp. 534–539.
10. S. S. Gokhale, Architecture-based software reliability analysis: Overview and limitations, *IEEE Tran. on Dependable and Secure Computing*, **4**(l): 32–40, 2007.
11. H. Okamura and T. Dohi, Building phase-type software reliability models, *Proc. of 17th International Symposium on Software Reliability Engineering, ISSRE'06*, Raleigh, NC, Nov. 2006, pp. 289–298.
12. J. D. Musa, Introduction to software reliability engineering and testing, *Proc. 8th International Symposium on Software Reliability Engineering, ISSRE'97*, Albuquerque, NM, Nov. 1997, pp. 3–12.
13. T. M. Khoshgoftaar. and N. Seliya, Comparative assessment of software quality classification techniques: An empirical case study, *Empirical Software Engineering Journal*, **9**(3): 229–257, 2004.
14. N. Nachiappan, L. Williams, J. Hudepohl, W. Snipes, and M. Vouk, Preliminary results on using static analysis tools for software inspection, *Proc. 15th International Symposium on Software Reliability Engineering, ISSRE'04*, St. Malo, France, Nov. 2004, pp. 429–439.
15. A. P. Nikora and J. C. Munson, Developing fault predictors for evolving software systems, *Proc. 9th IEEE International Software Metrics Symposium*, Pasadena, CA, Sept. 2003, pp. 338–350.
16. N. F. Schneidewind, Body of knowledge for software quality measurement, *IEEE Computer*, **35**(2): 77–83, February 2002.
17. A. P. Nikora, Classifying requirements: Towards a more rigorous analysis of natural-language specifications, *Proc. 16th IEEE International Symposium on Software Reliability Engineering*, Chicago, IL, Nov. 2005, pp. 291–300.
18. G. Kotonya, I. Sommerville, and S. Hall, Towards a classification model for component-based software engineering research, *Proce. 29th Euromicro Conference*, Chicago, IL, Sept. 2003, pp. 43–52.
19. T. M. Khoshgoftaar and E. B. Allen, Controlling overfitting in classification-tree models of software quality, *Empirical Software Engineering Journal*, **6**(l): 59–79, 2001.
20. T. M. Khoshgoftaar, E. B. Allen, and J. Deng, Using regression trees to classify fault-prone software modules, *IEEE Tran. on Reliability*, **51**(4): 455–462, 2002.
21. T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, Classification tree models of software-quality over multiple releases, *IEEE Trans. on Reliability*, **49**(1): 4–11, 2000.
22. T. M. Khoshgoftaar and N. Seliya, Software quality classification modeling using the SPRINT decision tree algorithm, *Proc.: l4th International Conference on Tools with Artificial Intelligence*, Washington, DC, November 2002, pp. 365–374.
23. V. Ponnuswamy, Classification of software quality with tree modeling using C4.5 algorithm. Master's Thesis, Boca Raton, FL: Florida Atlantic University, December 2001.
24. T. M. Khoshgoftaar, X. Yuan, and E. B. Allen, Balancing misclassification rates in classification tree models of software quality, *Empirical Software Engineering Journal*, **5**: 313–330, 2000.
25. D. Steinberg and P. Colla, *CART: A supplementary module for SYSTAT*. San Diego, CA: Salford Systems, 1995.
26. L. A. Clark and D. Pregibon, Tree-based models, in J. M. Chambers and T. J. Hastie (eds.), *Statistical Models in S*. Pacific Grove, CA: Wadsworth International Group, 1992, pp. 377–419.
27. J. R. Quinlan, *C4.5. Programs For Machine Learning: Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.
28. M. Melita, R. Agarwal, and J. Rissanen, SLIQ: A fast scalable classifier for data mining, IBM White Paper Series. Available: http: www.almaden.ibm.com/cs/quest, March 1996.
29. J. Shafer, R. Agarwal, and M. Mehta, SPRINT: A scalable parallel classifier for data mining, IBM White Paper Series, Available: http://www.almaden.ibm.com/cs/quest, September 1996.
30. T. M. Khoshgoftaar, E. B. Allen, and J. Deng. Controlling overfitting in software quality models: Experiments with regression trees and classification, *Proc. 7th International Software Metrics Symposium*, London, UK, April 2001, pp. 190–198.
31. J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman, *Applied Linear Statistical Models*. Tom Casson, 1996.
32. G. P. Beaumont, *Statistical Tests: An Introduction with Minitab Commentary*. Prentice Hall, 1996.
33. L. G. Votta and A. A. Porter, Experimental software engineering: A report on the state of the art, *Proc. the 17th International Conference on Software Engineering*, Seattle, WA, April 1995, pp. 277–279.

NAEEM SELIYA
University of Michigan—Dearborn
Dearborn, Michigan

TAGHI M. KHOSHGOFTAAR
Florida Atlantic University
Boca Raton, Florida

SOFTWARE SAFETY

INTRODUCTION

Safety is a property of a system such that it will provide hazard-free operation and, thus, will not endanger human life, jeopardize property, nor harm the environment. Alternatively, safety relates to those activities that seek either to minimize or to eliminate hazardous conditions that can cause bodily injury (1). *Merriam-Webster* (2) defines safety as "the condition of being safe from undergoing or causing hurt, injury, or loss." Leveson (3) states: "Safety is freedom from accidents or losses." Safety is a relative term as it depends on the perspective of an individual of what is risky; some would pass on a rock climbing or parachute jumping opportunity. Additionally, as a the state of a system changes in time, its level of safety also may change: Consider the airplane in a hangar and one cruising at 30,000 feet.

In addition, to provide desired functionality, performance, and quality of service, system designers strive to prevent *accidents, mishaps,* and *incidents*. An accident is defined by safety engineers as "an unwanted and unexpected release of energy" (4). This incomplete definition is expanded: A mishap is defined as "... an unplanned event or series of events resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment" (5). An incident is an event that involves no loss but a potential for loss under a different circumstance. All three terms exemplify a *safety violation*. With more systems relying on software, spectacular mishaps (e.g., accidents of Ariane-5, Therac-25 radiation therapy machines, the Mars Climate Orbiter mission, the Patriot Missile problem, the London Ambulance Service failure, the Osprey crash, etc.) have been attributed to software (6,7).

For modern software-intensive systems, safety-critical software functions are those that directly or indirectly can cause or allow a hazardous system state to exist, which in turn, may lead to a safety violation. However, software is unique: it is an abstract concept similar to music. Software is represented by a set of computer instructions like music is represented by a set of notes on a music sheet. By itself, software can do nothing and therefore, obviously, it is not hazardous. Similar to music, which must be played on an instrument, software must be executed on hardware in order to do anything useful. And if the hardware is part of a system that can lead to injury, death, destruction, loss of property, or damage to the environment, then system safety, including software safety, is paramount. Such systems will be called *safety-critical*. With more of our ubiquitous technology being controlled by software, a significant portion of the risk we face is in the hands of software engineers.

Software is considered safety-critical if it meets one of the three following criteria (8):

1. Resides in a safety-critical system, as determined by the system hazard analysis, AND (at least one of the following):
   a. causes or contributes to an identified hazard
   b. provides control or mitigation for identified hazards
   c. controls safety-critical functions
   d. processes safety-critical commands or data
   e. detects and reports, or takes corrective action, if the system reaches a specific hazardous state
   f. mitigates damage if a hazard occurs
   g. shares the processor with safety-critical software
2. Processes data or analyzes trends that lead directly to the safety decisions of the operator
3. Provides full or partial verification or validation of safety-critical systems, including hardware or software subsystems

Modem microprocessors and their flexible software have replaced the physical constraints of the electromechanical components of earlier systems. The real-time nature of modern application adds an additional layer of complexity and indeterminism (9,10). Adding functionality is an attractive proposition, and the design may expand beyond capabilities of the developers to properly analyze it. Because of software flexibility, the number of interactions increases to the point that they cannot be properly understood or planned. Inability to consider all possible system states or operator actions increases the likelihood for system mishaps. Additionally, computers introduce failure modes that cannot be handled easily by traditional hardware-based methods (e.g., redundancy), where the failures or random nature are caused by individual components rather than by systemic interactions between multiple components (11).

FAULTS AND FAILURES

Nothing is perfect, and thus mistakes are expected to be made by people in all phases of system development: specification, design, coding, manufacturing, and so forth. A mistake can cause a fault, which may be a defect within the hardware or a software bug introduced by the developer. Another source of fault can be a component or a tool deficiency. External disturbance also may trigger a fault by changing the system operating conditions.

IEEE standards define a *fault* as "an incorrect step, process, or data definition in a computer program" (12). A system or a component is said to have *failure* if it cannot perform its required functions within specified performance requirements. A symptom of the failure is observable as the system "incorrect" behavior at the system boundary, which represents an external view of the system not meeting the requirements.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

Software professionals use the term *error* to designate the result of a fault that leads to failure—resulting not only from the fault but also from the current system state and possibly a combination of events. However, the error often is understood as a defect, bug, or flaw in a program. The term *software anomalies* have been used (13) classifying the items that are missing *(omission errors)* and items that are incorrect *(commission errors)*. Therefore, differentiating between faults and errors is extremely difficult. For fault-tolerant systems, it is essential to define faults and failures. Other terms like error, defect, or bug can be only a source of confusion. Thus, it is recommended to substitute the term *fault* anywhere *error* or a similar term is used.

All faults are internal and may lay *dormant,* either being *latent* (not discovered) or *detected*. Alternatively, faults can be *active*, propagating other faults but still latent or resulting in failures observable outside the system boundary. Thus, a fault can lead to other faults or to a failure or to neither. Effectively, fault and failure are equivalent, except the boundary of the relevant system or subsystem is different: A failure of a subsystem may be considered a fault in the parent system.

There are numerous reasons for system faults to occur. Hardware may exhibit faults because of component defects and external disturbances like electromagnetic interference, radiation, temperature, pressure, and a physical damage. Operator mistakes constitute yet another reason for faults to occur. The reasons attributed mostly to software are:

- Specification mistakes in a form of wrong software requirements, incorrect algorithms, bad architectural decisions, ill-selected hardware platform, flawed tools, etc.
- Implementation mistakes, including poor design, sloppy construction, inadequate testing, wrong component selection, misunderstanding system software and low-level interfaces, faulty interfaces, imperfect tools, coding defects, etc.

Faults can be categorized considering their *nature, duration,* and *extent*. The *nature of fault* can be *random* or *systematic*. Random faults typically are limited to hardware, because of the wear-out of a component or a loose interconnect. Systematic faults can occur both in hardware and software because of mistakes in specification, design, and implementation (coding and manufacturing) (4).

The *duration of a fault* can be categorized as *permanent* or *intermittent*. A permanent fault remains until a corrective action is taken. The typical systemic causes of software permanent faults include mistakes in specification (incorrectness, ambiguity, inconsistency, and incompleteness), conceptual or architectural error in design, logical errors in algorithmic calculations, coding issues like improper parameter passing, location of synchronization constructs, instruction side-effect, stack problems (in terms of overflow, underflow, and memory leak), incorrect initialization of variables, range excess, and so forth. Another category of fault duration is the intermittent or transient fault, which tends to appear and then disappear after a short time. It may appear once more at a later time when appropriate conditions occur again. Typical causes of software intermittent faults are because of real-time conditions, concurrency, and resource contention (e.g., deadlock, live-lock, and priority inversion). However, the resulting failure may persist even though the fault seemingly disappeared. The *extent of a fault* can be either *localized*, for example, limited to one software module by means of partitioning and protection to prevent fault propagation, or *global*, with system-wide effects.

Techniques for fault management include requirement definition methodologies, architectural and design solutions, testing strategies, and use of mathematical formalism in the system requirements and design.

HAZARDS ANALYSIS

A *hazard* is a situation in which a potential source exists for danger or harm to people or the environment (14). Risk of human life, destruction of environment, and financial loss are the primary considerations in safety assessment. The issue is the gravity of a mishap, for example, the consequence of the operation rather than the actual operation outcome: Risk is evaluated as a combination of the *likelihood* of a mishap and the *severity* of the mishap consequences. A hazard is a set of conditions, or a state, that could lead to a mishap, given the right environmental trigger or set of events. A mishap (or even worse, an accident) is the actual realization of the negative potential inherent in a hazard. As presented in Fig. 1, three components of a “hazard triangle” must exist for a hazard to occur: hazardous element, initiating mechanism, and target and threat (15).

Often hazards and the related mishaps can result from a mismatch between a model of the process used to create the software and the actual physical process the software is controlling. The mismatch may be caused by an incorrect or incomplete software model of the process or by lack of accurate information about the system state. One goal of the software safety analysis then is to verify that the model (representing the requirements) specifies sufficiently safe behavior in all circumstances (16). The specification is analyzed with respect to known hazards by using the state machine concept and searching. Safety analyses often need to consider what can be termed as a negative property (i.e., “bad things do not happen”), as opposed to functional requirements defined in terms of positive properties (i.e., “good things do happen”) (17).

Some accidents could have been avoided by considering potential hazards. Building safe systems requires performing detailed analysis of the software with respect to system hazards. Specific hazards must be identified early in the development lifecycle, safety-critical requirements must be identified, and the likelihood and the criticality of a potential resulting accident need to be assessed. Special precautions need to be taken, and the resulting software should be designed in such a way that these potential hazards can either be avoided or controlled and that the risks associated with these hazards can be mitigated.

A variety of system-level hardware mechanisms, external to the computer, can be used to handle hazardous

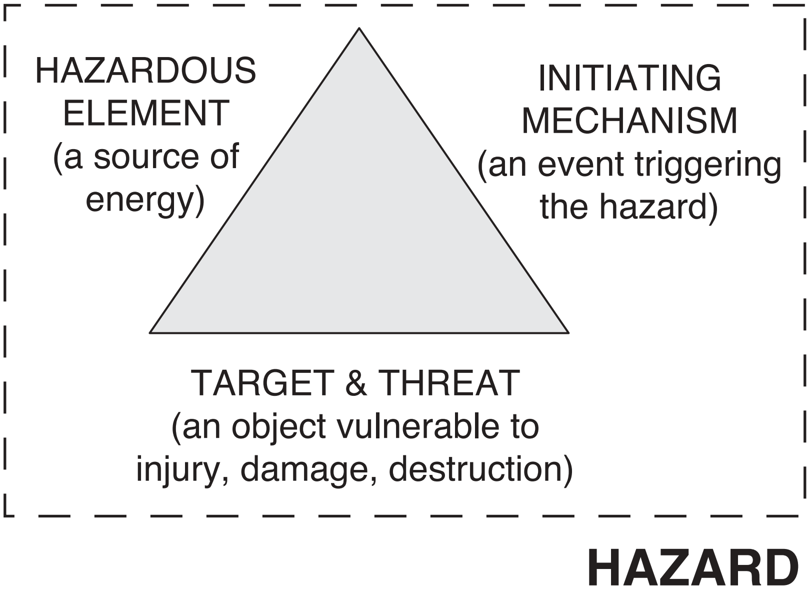

HAZARDOUS ELEMENT (a source of energy)
INITIATING MECHANISM (an event triggering the hazard)
TARGET & THREAT (an object vulnerable to injury, damage, destruction)
HAZARD

Figure 1. Hazard triangle.

conditions. They often have an equivalent representation in software. Requiring depressing the brake pedal before turning the ignition key limits a sequence of events that might permit a little child to start an engine, this example illustrates *interlock*. In software, interlocks are implemented by designing the execution of specific code elements in a sequence (using synchronization mechanisms, for example, semaphores) or by inhibiting execution until certain conditions are met. To prevent the system from entering into an unsafe state (or someone from entering a dangerous area), *lockouts* are used. In software, lockouts are techniques to control access to safety-critical code or data, implemented by a variety of access-right mechanisms (e.g., monitors and safety kernels). To enforce continuation in a safe state, *lockins* are used, for example, rejecting an input that would transition the system to an unsafe state (3).

A variety of techniques have been applied successfully to software modules and to components used to analyze the software behavior and its impact on the overall system operations. Analysis can be *static* in the early development phase, focusing on prevention, and *dynamic*, requiring actual execution of the program. Another categorization is between *functional* (including traditional testing, formal inspections, Cleanroom, and code/scenario analysis) and *logical* (including special cases of testing: boundary, fault injection, and structural) (18).

Of particular value to software safety have been well-established techniques common to both system and hardware: Fault Tree Analysis (FTA), Event Tree Analysis (ETA), Failure Modes and Effects Analysis (FMEA), Failure Modes, Effects and Criticality Analysis (FMECA), Markov Chains (MKV), Bayesian Belief Networks (BBN), Petri Nets (PN), Hazard and Operability Analysis (HAZOP), Cause Consequence Analysis (CCA), Operational/Support Hazard Analysis (OSHA), and Sneak Circuit Analysis (SCA) (15).

RISK ANALYSIS

The reality is that hazards do exist. The related risks need to be assessed and mitigated by considering three steps of building from a hazard to a mishap as presented in Fig. 2.

When building safety-critical software, the developers need to ensure an acceptable level of risk. At each of the three steps, the objective is to eliminate the risks where possible and to reduce the risks that are unavoidable. The criterion for risk acceptance is based on the decision of whether the risk is "As Low As Reasonably Practicable" (ALARP). The approach is to identify clearly two distinct and separated regions: acceptable and unacceptable regions of risk. Between these two regions, a gray area exists where the risk is tolerable only if further reduction is impractical or the cost of reduction is disproportional to the gained improvement (4).

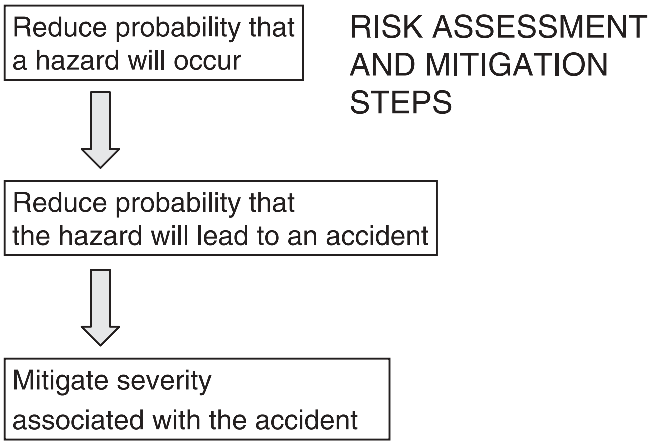

RISK ASSESSMENT AND MITIGATION STEPS
Reduce probability that a hazard will occur
Reduce probability that the hazard will lead to an accident
Mitigate severity associated with the accident

Figure 2. Progression from hazard to mishap.

Table 1. Risk categorization

	AVIATION: RTCA DO-178B	MILITARY: Mil Std 882D
Severity	• Catastrophic—prevents flight continuation • Hazardous—large reduction of flight safery and possible accident • Major—significant safety reduction of flight safely and possible injuries • Minor—slight reduction of flight safety and discomfort • No effect—no influence on safety	• Catastrophic—deaths, total disability, \$1M loss, irreversible environmental damage • Critical—pennanent multiple injuries resulting in partial disability, \$200K loss, reversible environmental damage • Marginal—injury or illness, \$10K loss, environmental damage easy to mitigate • Negligible—minor injury or illness, \$2K loss, minimal environmental damage
Likelihood	• Probable (frequent and reasonably probable): l0^{-4}–10^{0} • Improbable (remote and extremely remote): 10^{-9}–10^{-5} • Extremely improbable: $< 10^{-9}$	• Frequent—likely to occur often; $\sim 10^{-1}$ • Probable—fikely to occur several times: 10^{-2}—10^{-1} • Occasiional—likely to occur some times: 10^{-3}—10^{-2} • Remote—unlikely but possible to occur: 10^{-6}—10^{-3} • Improbable—extremely unlikely to occur: $< 10^{-6}$

Risk is a measure associated with hazard, representing the relative importance of the hazard and defining a possibility of something undesirable to occur. Risk is a composite of the probability and the severity of a mishap. Typically, the risk is expressed as a combination of likelihood (frequency) and severity (consequence).

Table 1 presents an example of risk categorization for civil aviation, according to the Radio Telecommunication Committee for Aviation DO-178B (19), and for military applications, following Department of Defense MIL STD 882D (5).

Considering the categorization of the likelihood and severity, the overall mishap risk can be classified using a risk assessment matrix in the order of decreasing risks level (e.g., letters A to D, Roman numerals I to IV, High to Low). Software may be categorized regarding its impact on the system operation. Table 2 shows an example of such categorization based on MIL STD 882C (20).

Using the defined software control category and the related system hazard criticality, a hazard criticality matrix can be built to identify the hazard risk index. This system-level assessment allows the software developers to apply varying levels of rigor and managerial approval.

THE NATURE OF SOFTWARE

Software does not degrade or change over time. Software does not fail by itself, and it always works the same way. Unlike when manufacturing physical entities, software can

Table 2. Mil Std 882C software control categories

MilStd 882C Software Control Categories	Description
I	Software exercises autonomous control over potentially hazardous hardware systems, subsystems, or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazard's occurrence.
IIa	Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.
IIb	Software item displays information requiring immediate operator action to mitigate a hazard. Software failures ill allow or fail to prevent the hazard's occurrence.
IIIa	Software item issues commands over potentially hazardous hardware systems, subsystems, or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.
IIIb	Software generates information of a safety-critical nature used to make safety-critical decisions. There are several, redundant, independent safety measures for each Hazardous event.
IV	Software does not control safety-critical hardware systems, subsystems, or components and does not provide safety-critical information.

be duplicated bit by bit without any variations. It seems to be easy to modify and change. However, physical change of a line of code may wreck havoc in the logical structure of the program. Leveson writes: "...while natural constraints enforce discipline on the design, construction, and modification of a physical machine, these constraints do not exist for software" (3). The unique and typically complex nature of software, combined with its tremendous flexibility, makes it difficult to analyze all the possible ways software performs (or occasionally fails to perform) the desired function. Despite recent progress in component-based design, software is not standardized to the extent that hardware is. Software may break immediately after installation, because of environmental or usage conditions not considered by the developers. It may fail intermittently, because of sporadic environmental conditions related to the timing sequence of external events. It may perform reliably until a certain unexpected combination of events or user inputs. It may work well for years until specific operating conditions change. Hardware fails because of physical stress, time, wearing out, and environmental factors. In contrast, software fails by human error during requirements, design, code, test, or maintenance. Factors affecting software failure rate, and thus software reliability, include complexity, methodologies and tools, process, schedule, and computing platforms (21).

The advantages of software, because of its flexibility, easy upgrade, high level of sophistication, and added functionality, outweigh potential risk associated with the software use in safety-critical systems. Software-controlled devices can collect information, interpret information, perform diagnostics, or present elegant interfaces to the user, typically at a more acceptable cost than can their hardware counterparts.

According to Brooks (22), the "essential" properties of software represent inherently its nature. To build safe software, these properties need to be considered and dealt with:

- *Complexity:* Computer programs have an extremely large number of execution paths and binary data that can create a number of combinations exceeding the number of hardware states in most complex integrated circuits.
- *Error Sensitivity:* Small errors may have a huge impact on the output; for example, flipping one bit totally may change the outcome of computations, resulting in a catastrophic failure.
- *Difficulty with Testing:* Exhaustive testing is not achievable for a program exceeding a few hundreds lines of code; it is particularly difficult in reactive systems, where sporadic external events may have impact on execution flow.
- *Correlated Failures:* Most failures in software do not result from wear-out but rather from developer-inserted defects; redundant systems may duplicate the original error, whereas the alternative designs may introduce new defects.
- *Lack of Professional Standards:* Anyone writing computer code can be called a "software engineer," and no objective way exists to argue with that, given the increasing dependency on software, the idea of licensing software engineers is ever more attractive.

Software is just another system component. The defects in software can cause hazardous events in the hardware it is controlling. A close collaboration between the system, safety, and software engineers is essential to identify potential causes of hazards. Software must be evaluated for its contribution to the safety of the system during the concept and planning phases and before its development or acquisition.

There are two aspects to be considered for software in safety-related systems. The first aspect is to design software in such a way to help mitigate the known hazards to the system. The second aspect is to construct the software in such way that it will not contribute to additional hazards because of undesired or incorrect operation.

SAFETY: TERMS AND CONCEPTS

The term *software safety,* or better *software system safety,* relates to the features and procedures (18) that ensure that the software is designed so that (a) the system performs predictably under normal and abnormal conditions and (b) the likelihood of unplanned events is minimized; their consequences are controlled and contained. The discipline of software safety is the systematic approach to identifying, analyzing, and tracking software mitigation and control of hazards and hazardous functions to ensure safe enough (ALARP) software operation within a system.

Software safety is an integral component of the system development. A software specification inaccuracy, design defect, or the lack of appropriate safety-critical requirements can contribute to a system failure or unsafe human decision. To achieve an acceptable level of safety for software used in critical applications, software safety methodology must be used not only in the requirements definition and the conceptual design, but also throughout the development and operational lifecycle of the system.

Despite visible progress since Leveson (3) made these observations in the early 1990s, often only marginal or superficial connection exists between software engineers and system/safety engineers. The former treat the computer as a stimulus–response subsystem and do not consider the system hazards or effects of software on system safety. The latter often ignore software and treat the computer as a black box, not giving consideration to hazards that it can mitigate or introduce.

Safety engineers identify different modes of safe operation of a *fault-tolerant system*. A *fail-safe system* is one that in case of failure will revert to a non operating state that will cause no mishap. A *fail-operate system* is a system that will continue to operate and will remain in a safe, possibly degraded state (23). The assessment of the system hazards and risk is the base for determination of what mode of fault-tolerant system will be necessary.

The *Software System Safety Handbook*(24) defines fundamental goals for software to ensure system safety. They are adapted here as “the ten commandments” of software safety:

1. Identify, evaluate, and eliminate hazards associated with the system and its software and reduce the risk to an acceptable level throughout the lifecycle.
2. Design safety into the software in a timely, cost-effective manner.
3. Address failure modes in the design of the software, including hardware, software, human, and system.
4. Minimize the number and complexity of safety-critical interfaces.
5. Minimize the number and complexity of safety-critical computer software components.
6. Apply sound human engineering principles to the design of the software user interface to minimize the probability of human error.
7. Minimize reliance on administrative procedures for hazard control.
8. Use sound software engineering practices and documentation in the development of the software.
9. Address safety issues as part of the software testing effort at all levels of testing.
10. Design software for ease of maintenance and modification or enhancement.

RELIABILITY VERSUS SAFETY

One important characteristic of a system is reliability. A widely accepted definition of reliability is the probability that the system will perform its function at a given time (25,26). Software reliability is the probability of failure-free software operation for a specified period of time in a specified environment. To assess reliability, developers and analysts take a bottom-up approach, placing the focus on the sources of the failures. In contrast, safety is a top-down paradigm, concentrating on system hazards, for example, on how the system contributes to endangering people; destruction of the system; and its environment (3). Because both reliability and safety assessment practices use the same methods and tools, they often are misconstrued as equivalent. Reliability engineering can complement safety. However, even a highly reliable system may be unsafe and, conversely, an unreliable (or simply not working) system may be perfectly safe. To make the point, consider a loaded unsecured gun in the hand of a child (reliable but not safe) and an aircraft with a non working engine sitting in a hangar (not reliable but safe). In fact, accidents may happen without evident system failure. Most accidents result from a combination of procedural or operator mistakes, environmental events, and system faults. A safety analysis considers the possibility of reaching a hazardous state when components operate without evident failure, concentrating on the interaction between the components in various states and environmental conditions.

According to Leveson (11), with respect to the requirements, the produced software may be reliable and correct, but still unsafe when:

- The software correctly implements its requirements, but the specified behavior is unsafe from a system perspective.
- The requirements do not specify a particular behavior required for the safety of the system.
- The software has unintended (and unsafe) behavior beyond what is specified in the requirements.

Nearly all software-related accidents can be traced either to (a) incomplete or incorrect assumptions about the operation of the system or the operations required from the computer, or to (b) environmental conditions or system states remaining unhandled. Failure of the software may cause a situation that leads to the system getting out of control and endangering or harming the operator and/or public. The complexity and apparent “easy” modifiability of software is the major obstacle to achieving system safety.

SOFTWARE FAULT TOLERANCE

The handling of software faults takes place in the context of overall system fault tolerance (23,27–29). Because software faults are expected to exist, they need to be managed. Fault management (in the form of avoidance, removal, evasion, and tolerance) depends on the phase of the project. In the development phase the concept of *fault avoidance* is applied, for example, the reduction of faults through carefully selected design and implementation methodologies, the adherence to a rigorous development process, verification, and validation (including testing, inspections, and application of formal methods). Another complementary approach is *fault removal*, which is based on identification and removal of faults found through testing. In the operational phase, *fault evasion* is based on the detection and mitigation of fault effects before they occur.

The ability of a system or component to continue normal operation regardless of presence of hardware or software faults is known *as fault tolerance*. The objective of fault tolerance is to design a system in such a way that faults do not result in a system failure and a related safety violation. Fault-tolerant systems have an ability to continue with normal operation even though a fault has occurred. Fault tolerance evidently improves reliability, availability, and safety.

Fault-tolerant software is capable of mitigating the impact of errors before they cause the failure of the system. In software-intensive systems, the issue is to handle requirements and design deficiencies where most of software faults reside. The fault-tolerance process consists of four stages:

- *Detection*—identification of the problem, for example the erroneous state.
- *Diagnosis*—evaluation of the potential damage and determination of the causes.
- *Containment*—prevention of the damage propagation to other system modules.
- *Recovery*—replacement of the erroneous system state with a correct state to continue operation.

A fault detection mechanism is based on checking the functionality of the components. The specific techniques for detecting data inconsistency include parity, checksum, residue, and cyclic code. Built-in testing (BIT) either in a form of predefined pattern generators to check the output against expected result or in a form of progress monitoring watchdogs/timeouts to detect illegal sequences, deadlock, or system inactivity are other popular techniques. Use of predefined reference devices to monitor correctness of the input or a device self-test at startup is often used in embedded dependable systems. At the system level, two basic fault detection mechanisms are the *acceptance test* (with either a known value or by comparison) and *voting* (30). The quality of such decision elements (adjudicators) is vital to the dependability of a system.

Recovery can be executed either *backward or forward*. In backward recovery, upon detection of error, the system is rolled back to the previously saved checkpoint, which is an error-free known state. Backward recovery typically is an application-independent scheme that can be used as long as the previous error-free state is known. However, the scheme may require that the system stops its operation in the process of recovery. Another disadvantage is that significant resources may be required to save the checkpoints periodically and to roll back when the error is detected. In more complex systems, a domino effect may occur when interacting processes are not properly synchronized. One process rolls back to a checkpoint, which causes the interacting process to roll back, which in turn may cause the first process to roll back further, and so forth. In contrast, the forward recovery is based on either a transition to a predefined state from which the system operation can be continued or a compensation for the detected discrepancy from the desired value. Forward recovery is application-specific and requires knowledge of the system state. Typically, the forward recovery scheme is faster because it does not require time-consuming rollback (31).

Software fault-tolerance techniques allow a system to tolerate software faults that may remain after system delivery. Software fault-tolerance techniques are based on the concepts of diversity in three areas: *design, data*, and *temporal*. Diversity introduces overhead that requires additional resources in terms of time, space, or both. The techniques depend on the type of computing software environment, A common misconception exists that the system must be redundant to be safe. A simplex system is one that does not employ redundancy and still can be fail-safe (but not necessary fail-operate). Redundant systems, with more than one computer, are employed in situations that require high dependability to ensure operation in presence of failure (23). In either case, a system may have one or more versions of the software. In a single version software environment, the techniques are limited to monitoring, assuring atomicity of operations, verifying decisions, and handling exceptions. Multiple version software environments, with independently developed software versions, provide for more assurance and complete recovery. The techniques used include (a) *recovery blocks* where the critical block of execution is checked for acceptance and, in case of failing the acceptance test, an alternate version is executed sequentially, (b) *N-version programming* where separate, independently developed software versions are operating in parallel on redundant computer resources and the output is decided based on the voting or based on the application-specific acceptance test, and (c) *N-self-checking programming* where the outputs from each concurrently operating component are compared and accepted only when they agree. Multiple data representation environments use diverse representation of input data. Example techniques are (a) *retry blocks* where an acceptance test is used to determine if the retry should be activated, and (b) *N-copy programming* using two or more copies of the program and an appropriate data re-expression algorithm (31). For more detailed description, see the article Fault Tolerant Software.

SOFTWARE SAFETY IMPLEMENTATION

Application of coding standards and the need to create readable, easy-to-maintain code often contradicts smart programming focused on reducing the execution time or/and saving memory. Safety-critical software often implements defensive programming guards against run-time errors. Each module has an initial code segment, which checks assumptions and data validity before the algorithm execution. For object-oriented programming, a class should be instantiated only one time as an object declared outside the block with a static lifetime. Using pointers is prohibited explicitly in much safety-critical application code. Also, to avoid potential memory leaks a dynamic allocation and de-allocation of memory should be avoided. Use of software-based checksums (e.g., cyclic redundancy check) also is recommended.

Evaluation and assessment of software safety is a rather difficult proposition. The approaches include safety requirement coverage, checklists, meeting specific process objectives, and so forth (32–34). Formal methods, based on discrete mathematics, provide a rigorous mechanism for describing both system and software during the development lifecycle. This description can be analyzed to verify the system behavior. The methods can prove certain properties of the system (safety, liveness, and reachability) and demonstrate that the computer program transfers its pre-conditions into desired post-conditions. In highly dependable applications, the use of formal methods has been encouraged and occasionally mandated (35–37).

A popular approach is to annotate a program with specification constructs to support formal verification. A proof of correctness is a formal verification approach based on enforcing a strict subset of the programming language thus applicable to the new development of safety- and mission-critical software rather than on trying to improve the quality of legacy code. The goal of another formal verification approach is the detection of a substantial proportion of defects rather than proving program correctness. Both approaches have been developing a variety of tools supporting formal analysis. Some of these tools forgo completeness and soundness of a new language in order to be more useful on un annotated or lightly-annotated legacy code.

Language Subsets

Programming languages need to be tailored before they can be used in safety-critical software. This tailoring is done by identifying a subset of the language that excludes certain features that are hard to use and verify (38).

An example of a formal verification approach that focuses on providing proof of correctness is SPARK Ada (39), a commercially-supported notation that has been used to develop some sizeable critical systems. SPARK uses an ordinary Ada 95 compiler. However, SPARK is more than an Ada subset because of the addition of annotations and the availability of the SPARK examiner tool that checks for adherence to the subset. The annotations provide design information about the usage of variables that would not be present in conventional Ada code. SPARK is really a design tool using the concept of correctness by construction (40).

The Motor Industry Software Reliability Association (MISRA) has developed coding guidelines for the C programming language intended to improve intelligibility of the programs and the predictability of the code behavior for safety-critical applications (41). A major difference from SPARK Ada is that no specific tool is associated with the guidelines. However, the industry has developed a variety of tools that check for the selected subset. The MISRA guidelines are formulated as set of rules based on two principles: (a) promotion of a common programming style and (b) avoidance of language features that are suspected to lead to program failure (with or without appropriate failure data). The first principle is supporting code maintainability; the second is related more to the actual hazards originating from the coding practices, and these rules can be the base for the selection of a safe language subset to prevent the occurrence of common mode failures. For example, the C language has a rich set of operators that can be combined without multiple levels of tedious parentheses. Unfortunately, problems are reported with the precedence of the default operators and the side effects, which can be avoided by adhering to specific rules. Some development groups in other industries have adopted the MISRA C guidelines, which can be enforced by performing static code analysis on application source code. However, MISRA-C rules and the related static checking do not guarantee predictable execution, for example, an array index range only can be checked dynamically (42).

The Ravenscar profile was established as a model for building safe and reliable real-time systems. The profile was defined in 1997 at an Ada 95 workshop convening at the village of Ravenscar in northern England (43). The workshop defined a set of tasking features compatible with a realistic size of application but defined to be implemented efficiently and be reasonably easy to certify. Such a safe Ada 95 subset includes tasking that is restricted by preventing local declaration of tasks, dynamic allocation of tasks, and asynchronous transfer of control. Memory allocation is allowed only once at program elaboration time. Deallocation is disallowed, simplifying run-time system. Task rendezvous is not allowed, and tasks can communicate only via protected objects. The tasks are dispatched in FIFO manner, with a ceiling-locking protocol priority. A single global handler handles all exceptions. The restrictions support a deterministic model of computation required for safety critical applications, which could be certifiable to the highest integrity levels.

Java has not been considered a suitable programming language for safety-critical applications because of its automatic garbage collection, complex object-oriented programming features, and inadequate support for real-time multi threading. The Real-Time Specification for Java has introduced features that help in the real-time domain. However, the complex programming model and the resulting supporting real-time virtual machine complexity prevent confident use of Java in high-integrity systems. The proposed Ravenscar-Java profile (44) concentrates on reliability, robustness, traceability, and maintainability, with an objective to ensure predictability in three areas: memory use, timing, and control and data flows. The profile allows for concurrent execution of schedulable objects (threads and event handlers) based on pre emptive priority-based scheduling. Schedulable objects have to be either periodic or sporadic with minimum inter-arrival times, and the priority ceiling protocol is required to be implemented in the runtime system. This profile facilitates the use of off line schedulability analysis, which is associated with fixed priority scheduling (for example, deadline- or rate-monotonic analysis)

Partitioning and Firewalls

For highly critical applications in regulated industries, depending on the criticality of the application as defined by system safety assessment, object code analysis on the target level often is required. To reduce potential errors, the development methodology calls for *independence* and *diversity*. For independence, it is understood that the verification/validation is performed by an independent entity. Depending on the criticality of the module, it can be a different organization, a different unit within the same company, or a different person not previously engaged in the development. Diversity in development of the same module, independently by two or more teams, is a mechanism to reduce the likelihood of similar human mistakes. Despite wide use in highly critical applications (nuclear, aviation, and space), the approach has been considered controversial, because common specification defects or the selection of the same approach in a difficult part of the implementation by the teams may compromise the diversity concept (45).

Certainly, the additional safety code required for checking, monitoring, redundancy, voting, and acceptance tests adds complexity to the software and thus inherently increases the safety risk. Rigorous software verification by means of testing, analysis, and inspection always is required (46). The recommended option to reduce the impact of potential safety violations and fault propagations includes the use of firewalls for safety-critical modules. Such firewalls can be accomplished by placing the source in separate translation units (separate files) and declaring all external functions and data inside the module as static (private in C++), with the only not-static functions being external ports (public in C++).

Partitioning is a fundamental concept used to implement differing levels of protection in systems combining both critical and non critical software. Partitions are used in fault-tolerant systems that require high availability, redundancy, or dynamic re-configuration (47). Traditionally, partitioning has been implemented via memory address space to ensure that the non critical and critical codes do not use the same physical resources. A microprocessor supervisor/ user mode can be a vehicle to implement such partition. The increasing demands on reliability and dynamic reconfiguration require the use of an explicit *spatial firewall* often implemented as a memory management unit. Additionally, a *temporal firewall* implemented by the run-time executive may be used. Such a solution may include fixed time-slice round-robin scheduling of all partitions or implementation of a partition priority scheme such that the critical partition gets as much CPU time as it needs.

Some modern real-time operating systems (RTOS) support partitioning adhering to standards such as the ARINC 653 Application Executive (APEX) for integrated modular avionics (48). The partitioned system executive consists of two major parts: (a) the operating system kernel that controls the scheduling of and the communications between the partitions that are resident on that processor and (b) the partition operating system that supports the internal scheduling of each of the partition threads and the mechanism to detect and respond to error situations within the application partition.

Safety Kernels

The software in safety-related systems must have certain behavioral properties to be considered safe. These properties can be expressed as predicates that the software must maintain with respect to its inputs, outputs, and states. Showing that the software for a particular system satisfies a given set of predicates is a verification problem, which typically is carried out via combination of analyses, tests, and inspections. More formal analysis for code of a significant size is difficult and expensive. One useful strategy is to design the software so that the most rigorous analysis can be applied to a relatively small portion of the entire software, with conventional methods applied to the remainder of the software.

Safety kernels are small, relatively simple units, which ensure desired particular behavior for the overall system without making any assumptions about the proper functioning of the remaining software. Kernels have a successful history in operating systems as a means of protecting access to computing resources and in security applications as a means of preventing unauthorized information flow. Because of the small size of the kernel, it can be thoroughly tested, analyzed, and verified. To enforce safety, two conditions must hold: (a) All predicates describing safe operation at the system level must be under the kernel control, and (b) an arbitrary behavior external to the kernel cannot impact the predicates (17).

SAFETY IN THE LIFECYCLE

Successful software safety can be implemented only as part of an overall system safety program. A continuous coordination and open communication between systems engineers, system safety personnel, software developers, software assurance personnel, and project management is the key to success. The critical steps to achieve software safety is first to perform rigorous safety analysis of the system, identifying the role and impact of software on safety. The identification of hazards and failure modes attributed to software allows developers to trace the safety requirement to software components and to appropriate testing procedures.

For software developers building safety-critical systems, an ideal and practically impossible objective to realize is to develop complete and correct requirements, a defect-free design, and a fault-free software implementation. Thus, the accepted feasible approach is to develop fault-tolerant designs, which will detect and compensate for software faults while the system is operating.

Depending on their placement in the system development lifecycle, the hazard analyses may have variety of forms (15):

- Preliminary Hazard List (PHL), to identify potential hazards/mishaps in the conceptual phase of developing a system
- Preliminary Hazard Analysis (PHA), to analyze the hazards and establish the initial system safety requirements
- Subsystem Hazard Analysis (SSHA), concentrating on the system components to identify hazards causal factors, effects, risks, and mitigation measures
- System Hazard Analysis (SHA), focusing on integration to ensure that overall system risk is known and accepted
- Operating/Support Hazard Analysis (OSHA), with a focal point on the procedures and human interface to assess the safety of operations
- Safety Requirements/Criteria Analysis (SRCA), to ensure that all identified hazards match their respective safety requirements and that they can be validated

IEC 61508 standard (49) identifies two types of safety requirements: safety function requirements and safety integrity requirements. The *safety function requirements* define the input/output sequences that perform the safety-critical operation. For example, a boiler could have a pressure sensor (input) that can reach a maximum value (algorithm) before the gas is shut off (output) to the burner. The *safety integrity requirements* define diagnostics and fail-safe mechanisms used to ensure that failures of the system are detected and that the system goes to a safe state if it is not capable of performing a safety function. Examples of integrity elements in the boiler would be a current-range diagnostic on the pressure sensor or a watchdog timer. If either of these elements detected a failure, they could force the system to a safe state.

The *Software System Safety Handbook* (24) divides safety requirements into two categories: generic and specific. The *generic software safety requirements* are domain independent and are applicable to common safety problems.

Generic requirements address such issues as the need for detection, isolation, and recovery from any failure of a safety-critical software function, checking the prerequisite conditions, status, and handling of software inhibits by the modules and assuring return to safe state/mode; they also use self-tests, unused code, configuration, fault containment, exception handling, error propagation, and so forth. The complete list of generic software safety requirements can be found in the Appendix E of the *Handbook* (50). The *specific, software safety requirements* include application-specific, system-unique constraints that can be identified in three ways (50):

(a) Top-down analysis of system requirements to identify system hazards and to specify which system functions are safety-critical. The entire safety organization participates in the mapping of these requirements to the software.

(b) Preliminary hazard analysis considers whether system hazards are mapped to the software. Software hazard-control features are identified and specified as requirements.

(c) Bottom-up analysis (e.g., flow diagrams, failure modes, effect analyses, fault trees) where the design solutions are analyzed and new hazard causes can be identified.

STANDARDS AND CERTIFICATION

A safety standard is a systematic approach to assure safety that is codified by a regulatory authority organization. The purpose of such a standard is to improve the safety of a critical system, to specify minimum standards of design and development techniques within the relevant industry, to encourage a structure of professional responsibility, to promote uniformity of approach between different teams and industries, as well as to provide legal basis in the case of a dispute.

Certification (or approval) is a process of getting a formal approval from a statutory authority (government or industry) to use the product. Depending on the determined system safety integrity level, an independent verification and validation (IV&V) effort may be required to get the system certified (from another developer, department, organization, or governmental agency). Collection of appropriate supporting evidence, particularly safety-related, in a format defined by standards and guidelines is the basis for the certification. Certification indicates a conformance to standards or guidelines and can be applied to individuals, organizations, tools, methods, systems, or products.

In other terms, certification is a legal recognition by the authority that an entity (product, service, organization, or person) complies with the applicable requirements. The objective of certification is to improve safety of the product by (51):

- enforcing minimum safety standards,
- increasing the awareness of safety,
- improving organizational structure, and
- encouraging professional responsibility.

The developer needs, therefore, to prepare and present the safety case that proves adherence to standards, which typically is a huge investment of time and resources.

Table 3 represents selected standards used in the development of safety-critical systems. The focus is on the standards directly related to the systems with significant software components.

The Functional Safety: The Safety Related Systems IEC61508 standard (49) was issued by the International Electrotechnical Commission in 1995. This standard is accepted in Europe as a generic standard for the functional safety of a programmable electronic system with a focus on product functionality in terms of the entire system, not only the software. Part 3 of the document (IEC 61508-03) is dedicated to software safety. Meeting the requirements of IEC 61508 for software development involves a systematic process, which emphasizes requirements traceability, criticality analysis, and validation.

Three major guidelines for ensuring safety of software intensive systems are: RTCA DO-178B (aviation), NASA-STD-8719.13A (aerospace), and MIL STD 882C/D (military).

Aviation

In 1980, the Radio Technical Commission for Aeronautics, now RTCA, Inc., convened a special committee to establish guidelines for developing airborne systems. After two revisions, DO-178B was published in 1989 (19). The FAA Advisory Circular AC20-115B mandates use of DO-178B for the development of software in airborne systems. The FAA Order 8110.49 compiles a variety of guidelines related to the use of software in airborne systems (52). Chapter 10 of the *FAA System Safety Handbook* (53) addresses issues of software in airborne system development.

DO-178B addresses the issue of lack of software visibility at the system level by describing ihe system aspect of software development, software lifecycle, planning, development, verification, configuration management, quality assurance, certification, required data, and additional considerations. The document amplifies the notion that safety assessment is a hierarchical process including functional hazard analysis on the aircraft and at the system level, common cause analysis, preliminary system safety assessment, and system safety assessment (SSA). The SSA documentation includes a system description, event probabilities, and classification and analyses of failure conditions. DO-178B defines safety and reliability categories for airborne equipment (catastrophic 10^{-9}, hazardous 10^{-7}, major 10^{-5}, minor 10^{-3}, and no effect) and their relation to the development assurance levels from the highest A to the lowest E. The main element of DO-178B are ten tables of objectives related to lifecycle processes described in the guide. Each table includes entries for an objective (identifier, description, and document section reference), applicability by software level, required artifacts, and control category. The objectives can be satisfied either with or without independence, conditional on the assurance level. The focus is on the lifecycle transition criteria and traceability.

Table 3. Selected safety/software standards

Domain	Standards/Guidelines	Organizations Involved
Aerospace	• RTCA/DO-178B • NASA-STD-8719.13A • NASA GB-1740.13-96 • ECSS-Q-80A	• FAA : Federal Aviation Administration, • NASA : National Aeronautic and Space Agency • EUROCAE : European Organisation for Civil Aviation Equipment • RTCA : Radio Telecommunication Committee for Aviation • ESA - European Space Agency
Military	• MIL-STD-882C/D • DEF STAN 00-55	• DoD : Department of Defense • MoD: Ministry of Defence
Nuclear	• EC 60880 Ed. 2,2006 • AECL CE-1001 rev 2 • IEC 62138, 2004	• CANDU : Canada Deuterium Uranium
Transportation	• EN 50128 • IEC 62279, 2002 • MISRA Guidelines	• MISRA : Motor Industry Software Reliability Association
Biomedical	• IEC 601-1-4 (1996-06) • ANSI/AAMI SW68 • ANSIUL1998 • AAMI TIR32:2004	• FDA : Food and Drug Administration
Generic	• IEC 61508-3, 1998 • IEC 61511-1,2003 • AS 61508.3,1999 • IEEE 1228-1994	• IEC - International Electrotechnical Commission • IEEE - Institute of Electrical and Electronic Engineers • ANSI - American National Standards Institute • CENE - Comité Européen de Normalisation Electrotechnique

Software development processes artifacts defined in the DO-178B include the plan for software aspect of certification (PSAC), development, verification, configuration management and quality assurance plans, design and code standards, requirements and design specification data, source and executable code, verification procedures and test cases and their results, life cycle configuration index, problem reports, configuration management and quality assurance records, and the software accomplishment summary (SAS). The PSAC and SAS are obligatory, whereas others need to be either submitted to the certifying authority for review or made available when requested, depending on the assessed criticality of the system.

Aerospace

The *NASA Software Safety Guidebook* NASA-STD-8719.13A (54) replaces the older NSS1740.13. The standard defines whether the software is safety-critical and describes the activities necessary to ensure that safety is designed into the software. This standard also specifies the software safety requirements, activities, data, and documentation necessary for the acquisition or development of software in a safety-critical system. The standard describes the general purpose of a safety process and the minimal requirements for a safety process (as a list of "shall" statements for specific stages of the software lifecycle) with an emphasis on IV&V.

For a software quality analysis process, the following activities are prescribed:

- Evaluation of standards and procedures
- Audits of management, engineering, and assurance processes
- Reviews of project documentation
- Monitoring of formal inspections and reviews
- Monitoring/witnessing of formal acceptance-level software testing

For software quality engineering process, the following activities are recommended:

- Analysis, identification, and detailed definition of quality requirements
- Evaluations of standards, design, and code
- Collection and analysis of metric data pertaining to quality requirements

Military

System Safety Program Requirements MIL-STD-882C was released in 1993 (20) and the updated version Standard Practice for System Safety MIL-STD-882D in 2000 (5). The standard focus is on the entire system rather than on the software specific components. In version 882C entire sections are dedicated to the software safety, and the appendix identifies specific tasks associated with management and engineering. In contrast, version 882D mentions software only three times, and the specific tasks are not described clearly. The standard identifies acronyms and definitions, general requirements (approach, hazards and risks mitigation, and hazard reduction), and the detailed requirements—delegated to appendix as guidelines. The main objective of the standard is to reduce or eliminate hazards via the following mitigation measures:

- Hazard elimination through design selection
- Incorporation of safety devices (with periodic functional checks of the devices)
- Use of warning devices to detect the hazard condition and to produce warning signals
- Development of operating procedures and training for personnel
- Reduction of risk to acceptable level
- Performing verification and risk assessment review
- Tracking hazards

The STD-882D defines hazard severity categories (I–IV: catastrophic, critical, marginal, and negligible) and probability levels (A–E: frequent, probable, occasional, remote, and improbable). It also provides the related definition of risk levels (high, serious, medium, and low), their impact, approaches, and mitigation measures, as well as the system safety design order of precedence (i.e., the order to be followed for satisfying system safety requirements and reducing risks). The standard also provides the definition of system safety planning (objective, organization, milestones, reporting, approach, and methodology), safety performance requirements (quantitative and standards), and safety design requirements.

BIBLIOGRAPHY

1. Encyclopedia Britannica Available: http://www.britannica.com/eb/article-9064709/safety [2007 June 12].
2. Merriam-Webster. Available: http://www.merriam-webster.com/dictionary/safety [2007 June 12].
3. N. Leveson, *Safeware—System Safety and Computers*. Reading, MA: Addison Wesley, 1995.
4. N. Storey, Safety-Critical Computer Systems. Reading, MA: Addison Wesley Longman, 1996.
5. Department of Defense *Standard Practice for System Safety*, DoD Std 882D, Feb 2000. Available: http://safetycenter.navy.mil/instructions/osh/milstd882d.pdf [2007 June 12]
6. C. M. Knutson and S. Carmichael, Safety first: Avoiding software mishaps, *Embedded Systems Programming*. Available: http://www.embedded.com/2000/0011/001lfeatl.htm [2007 June 12].
7. A. Kornecki and J. Lewis, Software tragedies: Case studies in software safety, *Proc. 21st International System Safety Conference*, System Safety Society, Montreal, 2003, pp. 896–905.
8. Goddard Space Center *Information and Tools for Saftware Assurance Practitioners in the NASA Community*. Available: http://sw-assurance.gstc.nasa.gov/disciplines/safety/index.php [2007 June 12].
9. A. Burns and J. Mc Dermid, Real-time safety critical systems: Analysis and synthesis, *Software Engineering Journal*, **9** (6), l994.
10. T. Anderson and J. Knight, A framework for software fault tolerance in real-time systems, *IEEE Trans. Software Engineering*, SE-**9** (3) 355–364, 1983.
11. N. Leveson, System safety in computer-controlled automotive systems, *Proc. SAE Congress*, 2000. Available: http://sunnyday.mit.edu/papers/sae.pdf [2007, June 12].
12. IEEE Standard Board. *Standard Glossary of Software Engineering Terminology*, Std. 610.12, 1990. Available: http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990 desc.html deschfail [2007, June 12].
13. *Guide to Classification for Software Anomalies*, IEEE Std.1044.1–1995.
14. *Safety Aspects—Guidelines for Their Inclusion in Standards,* ISO/IEC Guide 51, 1999. Available: http://webstore.iec.ch/preview/info_isoiecguide51%7Bed2.0%7Den.pdf [2007, June 12].
15. C. Ericson, *Hazard Analysis Techniques for System Safety*. New York: 2005.
16. M. Jaffe, N. Leveson, M. Heimdahl and B. Melhart, Software requirements analysis for real-time process control systems, *IEEE Trans. Software Engineering*, **17** (3): 241–258, 1991.
17. J. Rushby, Kernels for safety, in T. Anderson (ed.), *Safe and Secure Computing Systems*. Blackwell Scientific Publications, 1989.
18. D. Herrmann, *Software Safety and Reliability*. IEEE Computer Society, 1999.
19. Radio Technical Commission for Aeronautics, *Software Considerations in Airborne Systems and Equipment Certification*, RTCA DO-178B, RTCA SC-167, 1992. Available for purchase: Radio Technical Commission for Aeronautics site, www.rtca.org.
20. Department of Defense, *System Safety Program Requirements*, DoD Std 882C, Jan 1993. Available: http://www.wbdg.org/cch/FEDMIL/ms882c.pdf [2007 June 12]
21. A. Kornecki and J. Erwin, Characteristics of safety critical software, *Proc. 22nd International System Safety Conference*, System Safety Society, Providence, RI, 2004.
22. F. Brooks, *The Mythical Man Month*, 20th anniversary ed. Reading, MA: Addison-Wesley, 1995.
23. W. Dunn, *Practical Design of Safety-Critical Computer Systems*. Reliability Press, 2002.
24. Joint Services Software Safety Committee, *Software System Safety Handbook*, December 1999. Available http://www.egginc.com/dahlgren/files/ssshandbook.pdf [2007 June 12].
25. M. Friedman and J. Voas, *Software Assessment: Reliability, Safety, Testability*. New York: Wiley, 1995.
26. M. Lyu, *Handbook of Software Reliability Engineering*. New York: McGraw Hill, 1996.
27. *A Conceptual Framework for Systems Fault Tolerance*, SEI, March, 1995. Available: http://hissa.ncsl.nist.gov/chissa/SEI_Framework/framework_6.html [2007 June 12].
28. J-C. Laprie, Definition and analysis of hardware and software fault tolerant architectures, *IEEE Computer*, **23** (7): 39–51, 1990.
29. D. Pradhan, *Fault Tolerant Computer System Design*. Englewood Cliffs, NJ: Prentice Hall, 1996.
30. S-T. Levi and A. Agrawala, *Fault Tolerant System Design*. New York: McGraw Hill, 1994.

31. L. Pullum, *Software Fault Tolerance Techniques and Implementation*. New York: Artech House, 2001.

32. D. Parnas, A. Van Schouven, A. Po Kwan, Evaluation of safety critical software, *Communications of the ACM*, pp. 636–648, June 1990.

33. A. Kornecki, Assessment of software safety via catastrophic events coverage, *Proc. 21st IASTED International Conference on Software Engineering (SE'2003)*, February 2003.

34. R. de Lemos, A. Saeed and T. Anderson, Analyzing safety requirements for process control systems, *IEEE Software*, May 1995.

35. J. Rushby, *Formal Methods and Their Role in Certification of Critical Systems*, Technical Report, CSL-95-l, SRI International, 1995.

36. J. Bowen and M. Hinchey, *High Integrity System Specification and Design*. New York: Springer1999.

37. N. Platt, J. Van Katwijk and H. Toetenel, Application and benefits of formal methods in software development, *Software Engineering Journal*, **7** (5) 335–346, 1992.

38. P. V. Bhansali, A systematic approach to identifying a safe subset for safety-critical software, *ACM SIGSOFT Software Engineering Notes*. ACM Press, **28** (4), 2003

39. J. Barnes, *High Integrity Software: The SPARK Approach to Safety and Security*. Reading, MA: Addison Wesley, 2003.

40. A. Hall and R. Chapman, Correctness by construction: Developing a commercial secure system, *IEEE Software*, pp. 18–25, Jan/Feb 2002.

41. MISRA C guidelines *(1998) ISBN 0-9524156-9-0*, Available for purchase: Motor Industry Software Reliability Association site, www.misra.org.uk.

42. L. Hatton, Safer Language Subsets: An overview and a case history, MISRA C, *Information and Software Technology*, **46** (7): 465–472, 2004.

43. B. Dobbing and A. Burns, The Ravenscar profile for real-time and high integrity systems, *CrossTalk—The Journal of Defense Software Engineering*, Nov 2003.

44. J. Kwon, A. J. Welling and S. King, Predictable memory utilization in the Ravenscar–Java Profile, *Proc. IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2003. Available: http://citeseer.1st.psu.edu/kwon03predictable.html [2007 June 12].

45. J. Knight and N. Leveson, An experimental evaluation of the assumption of independence in multiversion programming, *IEEE Trans. Software Engineering*, **SE-12**, (1): 96–109, 1986.

46. S. Gardiner, *Testing Safety Related Software—A Practical Handbook*. New York: Springer, 1998.

47. B. Dobbing, Building partitioning architectures based on the Ravenscar profile, *Special Issue: Presentations from SIGAda 2000*, **XX** (4), 2000.

48. *ARINC653 Avionics Application Software Standard Interface, Part 1: Required Services, Part 2: Extended Services, Part 3: Conformity Test Specification*. Available for purchase: Aeronautical Radio, Inc. site, www.arinc.com.

49. *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems—Part 3: Software Requirements*, IEC 61508-3 (1998–12) Available: for purchase: International Electrotechnical Commission site www.iec.ch/61508.

50. Directorate for Safety, US Army, Communication-Electronics Life Cycle Management Command, *Software System Safety* AMSEL-SF. Fort Monmouth, NJ. Available: http://www.monmouth.annv.mil/cecom/safety/sys_service/softwar-e.htm [2007 June 12].

51. J. Voas, Certifying software for high assurance environments, *IEEE Software*, pp. 48–54, Jul/Aug. 1999.

52. U.S. Department of Transportation, Federal Aviation Administration,*Software Approval Guidelines*, FAA Order 8110.49, 2003. Available: http://www.airweb.faa.gov [2007 June 12].

53. Federd Aviation Administration (FAA System Safety Handbook. Available: http://www.faa.gov/librarv/manuals/aviation/nsK_management/ss_handbook/ [2007 June 12].

54. NASA Glenn Research Center Safety and Assurance Directorate *NASA Software Safety Guidebook*, NASA STD-8719.13 A. Available: http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf [2007 June 12].

Andrew J. Kornecki
Embry Riddle Aeronautical University
Daytona Beach, Florida

SOFTWARE SECURITY

Software security is a branch of computer security that addresses enforcement of secure behavior of development and operation of software systems. As shown in Fig. 1, computer security consists of four layers: crypto, protocols, systems and languages, and applications (1). Although the crypto and protocol layers often involve software and operating systems and programming languages are in essence software, software security is primarily concerned with security issues at the application layer. It is built upon the crypto, protocols, and systems and languages layers. The goal is to prevent, detect, and recover from software attacks that violate application-specific security requirements with respect to confidentiality, integrity, and availability (CIA). Confidentiality, integrity and availability refer to the concealment, trustworthiness, and desired use of information and resources (2).

Software security has been a major source of cyber security risks for many reasons. First, many attacks against a company's network come at the application layer. They bypass the traditional security mechanisms at the crypto, protocols, and systems layers (e.g., firewall and intrusion detection, to name a few). However, protection of software applications is in general beyond the capabilities of these mechanisms because they lack knowledge of application semantics. For example, a firewall cannot distinguish between the traffic of an attempted attack and the traffic of legitimate use of an application through the open ports. Second, software security is often added as an afterthought near the end of software development. It is well-known that "penetrate-and-patch" is a common yet dumb practice for software security. Although secure coding is important, software security requires a lot more beyond coding. For example, a large portion of the security flaws uncovered during Microsoft's "security push" in 2002 were closely related to design-level problems (3). Principled use of assurance techniques is yet seldom adopted throughout software development processes (4). Last, a lack of well-trained workforce exists for secure software development. Software security has not gained much attention until recently.

It is well accepted that software security should be addressed throughout software development processes. The rest of this article discusses software security from the software engineering perspective. It serves as an overview of secure software engineering.

THE ADVERSARY'S PERSPECTIVE OF SOFTWARE SECURITY

As mentioned before, the goal of software security is to prevent, detect, and recover from software attacks. Attacks are misuses and anomalies of the intended system functionality. They violate the security goals of information, resources, and services. Consider a web-based shopping cart application. One of its intended functions is to allow customers to purchase goods online. A potential misuse of this intended function is that the customers may purchase goods at reduced prices by unexpected modification to the price data. This misuse violates integrity of the price data. The adversary's perspective of software security refers to the idea that thinks of software security from the standpoint of how an adversary would attack or exploit a software application. A software application should be considered insecure until demonstrated to be resistant to all potential attacks. Strictly speaking, no software is absolutely secure. We refer to the potential attacks as security threats.

As shown in Fig. 2, the adversary's perspective of software security advocates that security threats should be identified according to the security objectives imposed on the intended functions, services, and related information and resources. The information and resources that a software system must protect from attacks are called assets. They are potential targets of attacks. The security threats then suggest what, where, and how security features for threat mitigation should be applied. Thus, the intended functions and threat mitigations together form a system that is secured from the identified security threats. The adversary perspective of software security can be applied throughout the software development processes, which include from requirements analysis, design, implementation, testing, and deployment.

Amplification of Vulnerabilities in Software Development Processes

Security threats are caused by security vulnerabilities in software artifacts (e.g., design and code). A vulnerability is a security flaw that represents a valid way for an adversary to realize one or more threats. An unfortunate fact is that more vulnerabilities and hence more security threats may emerge when the development moves from one phase to the next (e.g., from design to implementation). We call this phenomenon amplification of vulnerabilities in software development processes, as shown in Fig. 3 where each shaded circle represents a vulnerability. On one hand, one vulnerability in the artifact of an earlier software development phase can result in multiple vulnerabilities in a later development phase. For example, a missing or incomplete security requirement can lead to several vulnerabilities in the design and code. On the other hand, design and implementation may introduce new security holes that are specific to the design and implementation choices. For example, the design of a web-based shopping cart application that relies on client-side input validation may leave a vulnerability that can be exploited to purchase goods at modified and reduced prices. Using C as the implementation language may result in C-specific buffer overflow vulnerabilities. Amplification of security

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

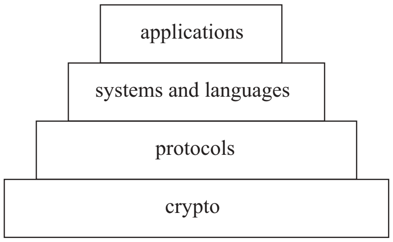


Figure 1. Layers of computer security (1).

vulnerabilities requires that threat modeling, risk assessment, threat mitigation, and risk control should be conducted in each development phase. The earlier the vulnerabilities are found, the cheaper the cost to fix them.

Threat Modeling

To better understand security requirements and solutions, it is important to identify, specify, evaluate, prioritize, and mitigate security threats. Threat modeling is the process of producing a simplified, abstract description of how an adversary would perform potential attacks or pose security threats to the system. It can be conducted at various levels of abstraction and granularity or in various software development phases—requirements, design, implementation, and even testing.

Several notations have been proposed for threat modeling, such as misuse cases (based on use case modeling), threat trees (a variation of fault trees for safety analysis), threat nets (based on Petri nets), anti-goals (based on goal-oriented requirements analysis), and threat traces (based on sequence diagrams or activity diagrams). Among these notations, threat trees and misuse cases have been widely applied. Threat trees [or attack trees (6)] represent security threats in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes. Figure 4 shows a sample threat tree. The goal of the threat is to obtain another person's username and password. To achieve this goal, the adversary must obtain a valid username AND a valid password for the username. To obtain a valid username, the adversary can either use an error string from the login page to determine username validity OR trick a user to disclose her username. Similarly, to obtain a valid password of the username, the adversary can either use the login page to guess the password by brute force OR trick the user to disclose her password. In general, AND nodes represent different steps toward achieving the same goal, whereas OR nodes are alternatives. A threat tree implies attack paths through which the threat can be accomplished. Each attack path is a route from a leaf to the root, inclusive of AND nodes. For example, the threat in Fig. 4 has four attack paths. It is also possible to compose threat trees. A threat tree can represent a node in another threat tree. In addition, threat trees can be documented in a textual form. Figure 5 shows the textual description of the threat tree in Fig. 4. Threat trees are simple and easy to understand. For rigorous modeling, however, formal methods such as Petri net can be applied (5).

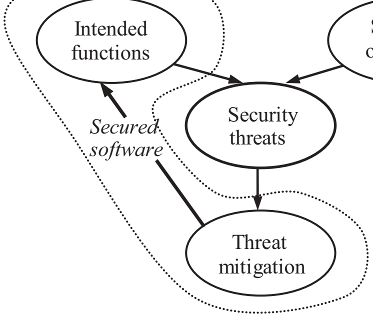


Figure 2. The adversary's perspective of software security (5).

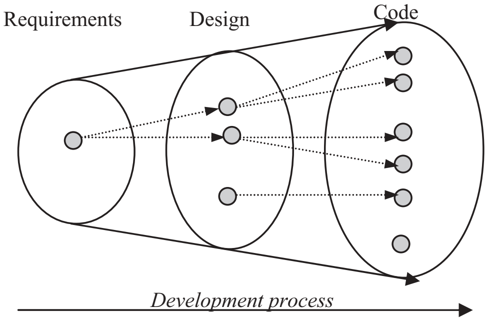


Figure 3. Amplification of security vulnerabilities in software development processes.

From the adversary's perspective of software security shown in Fig. 2, threat modeling should not be independent of system modeling. System modeling should deal with both intended functions and threat mitigations (security requirements or mechanisms). A threat makes no sense if it is irrelevant to the intended functions. For example, the node 1.1.1 in Fig. 4 indicates that the system offers a login page. Threat modeling and system modeling may exploit different modeling notations. For example, the Microsoft's threat modeling approach (7,8) uses data flow diagrams for system modeling and threat trees for threat modeling, Xu and Nygard's approach (5) uses Petri nets as a unified formalism for system modeling and threat modeling.

To select appropriate notations for threat modeling and system modeling, the following issues should be taken into consideration:

- The notations for threat modeling and system modeling must be able to express the desired level of abstraction. Security threats make sense only when they are defined with respect to the intended functions at the same level of abstraction. For example, misuse cases are suitable for capturing security threats at the requirements phase because they are based on the use case modeling approach to requirements analysis. However, they are probably inappropriate for modeling threats at the level of detailed design.
- The notation for system modeling must be able to capture the information, resources, and services that

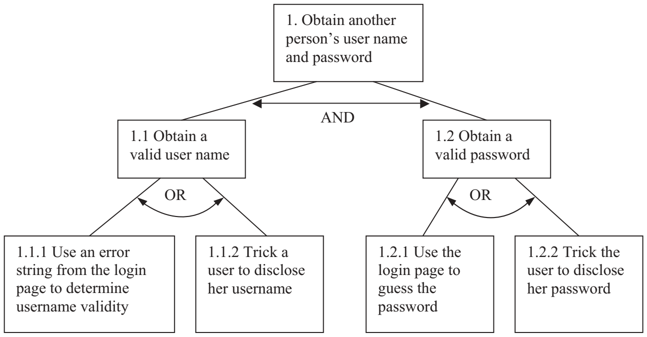

1. Obtain another person's user name and password
AND
1.1 Obtain a valid user name
1.2 Obtain a valid password
OR
OR
1.1.1 Use an error string from the login page to determine username validity
1.1.2 Trick a user to disclose her username
1.2.1 Use the login page to guess the password
1.2.2 Trick the user to disclose her password

Figure 4. A sample threat tree.

should be protected but can be threatened. A threat is meaningless unless it involves some asset or target that is of interest to the adversary.

- The notation for threat modeling must be able to capture the interaction between the threat model and the system model. A threat is meaningless unless it involves some way through which the adversary can interact with the system. In other words, threat models must be traceable—it can be determined which part of the intended functionality is relevant to a given threat and which part of the system mitigates the threat if the threat is claimed to be mitigated.

Threat identification is often a challenging task before threat modeling can be done. In general, we can identify security threats in terms of security objectives (e.g., CIA), threat catalogs, and threat elements (assets of interest to the adversary and way of interaction with the system).

- Identifying threats with respect to security objectives. We can check each intended function and related information and resources to see if it can be misused to violate confidentiality, integrity, or availability. For example, purchasing goods online is an intended function in a shopping cart application. To elicit security threats, we can ask such questions: Could the adversary purchase goods without payment or at reduced price? Could the adversary put the system out of service?
- Identifying threats using threat catalogs or checklists. We can use a security catalog (a list of threat types) to elicit security threats. For example, Shirey (14) categorizes security threats as disclosure (unauthorized access to assets and services), deception (acceptance of false data), disruption (interruption or prevention of correct operation), and usurpation (unauthorized control of some part of a system). Howard and LeBlanc's STRIDE catalog (7) includes spoofing (an impersonation of one entity by another), tampering (unauthorized change of data), repudiation (a false denial that an entity sent or created something), information disclosure (unauthorized access to information or exposure of protected data), denial of service (a long-term inhibition of service), and elevation of privilege (access to assets with a higher security context than intended). Different threat catalogs can be overlapping. For example, spoofing is a form of both deception and usurpation; tempering is a form of deception, disruption, and usurpation; repudiation is a form of deception; and denial of service is a form of usurpation.
- Identifying threats in terms of assets and system interactions.
  A meaningful threat must involve assets of interest to the adversary and a way of interacting with the system. We can identify threats by inspecting whether the information and resources (e.g., patient information) and the user interface (e.g., input of user name and password) can be misused or exploited.

Assessment of Security Risks

Security threats are unwanted risks that have negative consequences. After security threats have been identified and specified, we can evaluate their risks quantitatively and decide how to control the risks. As security threats are ubiquitous, the quantitative evaluation helps prioritize the threats. With limited budget and resources, for example, we can choose to mitigate the threats that are of highest risks.

Risk exposure is a traditional approach to quantifying the effects of risks. The risk exposure of a threat is the multiplication of the risk impact (i.e., the loss associated with the threat) by the risk probability (i.e., the likelihood that the event will occur). Consider the disclosure of patient information as an example. If the likelihood of this threat is 0.4 and the loss caused by such an attack is $100,000, then the risk exposure of the threat is $40,000. Although threat mitigation reduces security risks, it may not be possible or practical to completely remove the risks. In particular, a threat mitigation technique, (e.g., SSL) may have its own

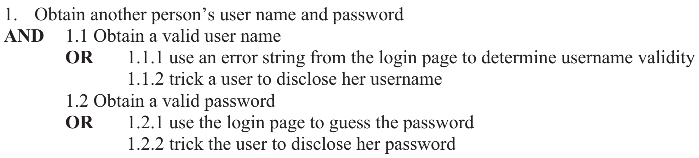

```
1.   Obtain another person's user name and password
AND     1.1 Obtain a valid user name
        OR       1.1.1 use an error string from the login page to determine username validity
                 1.1.2 trick a user to disclose her username
        1.2 Obtain a valid password
        OR       1.2.1 use the login page to guess the password
                 1.2.2 trick the user to disclose her password
```

Figure 5. The textual description of a threat tree.

Table 1. An Example of DREAD Risk Rating

Threat Description	Attacker Manipulates Patient Information
Threat target	Patient data
Threat category	Tempering with data and potentially information disclosure
Risk	Damage potential: 10 Reproducibility: 7 Exploitability: 8 Affected users: 10 Discoverability: 10 Overall: 9
Comments	This threat concerns accessing the patient information as it travels across the network.

threats. We must take into account the cost of threat mitigation and the security risks after the mitigation. To do so, we can measure the risk leverage of threat mitigation. It is the difference in risk exposure divided by the cost of mitigating the threat. Formally, risk leverage is equal to (risk exposure before mitigation-risk exposure after mitigation)/ cost of mitigation.

If the leverage value of the threat mitigation technique is not high enough to justify the action, then we can look for other mitigation techniques.

Microsoft's DREAD is another way to evaluate the risks of security threats. For each threat, the damage potential, reproducibility, exploitability, affected users, and discoverability are rated on a scale of one to ten. The bigger the rating, the greater the risk the threat poses to the system. The overall risk is determined by averaging the five ratings. As an example, Table 1 shows the DREAD ratings of threat "attacker manipulates patient information."

Threat Mitigation

Different strategies are used for dealing with the risks of security threats. Primary examples include (*1*) assuming the risks without taking any action, (*2*) avoiding the risks by changing functional or performance requirements such that the threats no long exist, and (*3*) adopting security requirements or techniques to mitigate the threats. To reduce the threat of spoofing, for example, we can add an authentication requirement. To meet this requirement, several security techniques are available, such as digest, Kerberos, and passport. To mitigate the threats of tempering with data, we may choose from authorization, message authentication codes, digital signatures, and temper-resistant protocols, e.g., SSL/TLS. A partial list of mitigation requirements and techniques for the STRIDE threats can be found in (7). In addition, security design patterns can also serve as general mitigation techniques.

A mitigation requirement or technique may countermeasure multiple threats. For example, authentication is useful for mitigating both spoofing and denial of service. One threat (e.g., information disclosure) may entail multiple mitigation techniques. When a mitigation requirement or technique is applied, the threat model and system model should be updated to reflect the decision. This model will improve the traceability of threats and mitigations within a development phase and across different development phases.

Appropriate applications of mitigation techniques can significantly reduce the risks of security threats. it does not mean that the software will be absolutely secure. Security threats are ubiquitous. Even if it is possible to identify all threats, mitigation of all threats is in general beyond the availability of required resources. Moreover, mitigation requirements and techniques are subject to their own threats. It is not realistic to expect the prevention of all potential attacks, although the process of threat modeling, risk assessment, and threat mitigation within a development phase is often repetitive. The process would stop when, for the limited resources available, the level of security has become acceptable, and the unmitigated threats are believed to be sufficiently expensive for an adversary to accomplish. Prevention of attacks is critical, but it is not the only goal of software security. Security-intensive software should also address issues that pentain to the detection of and recovery from attacks. Auditing such as logging and monitoring is a common approach to this end. Auditing is also critical to forensics, the goal of which is to investigate when and how an attack has happened and who is responsible.

REQUIREMENTS ANALYSIS OF SECURE SOFTWARE

The requirements analysis process for building a software system aims at capturing precisely what to build. The requirements of a software system include functional requirements, nonfunctional requirements, design constraints, and process constraints (9):

- A functional requirement describes required behavior in terms of required activities, such as reactions to inputs.
- A nonfunctional requirement, or quality requirement, describes some quality characteristic that the software solution must possess, such as fast response and ease of use.
- A design constraint is a design decision, such as choice of platform or interface components, that has already been made and that restricts the set of solutions.

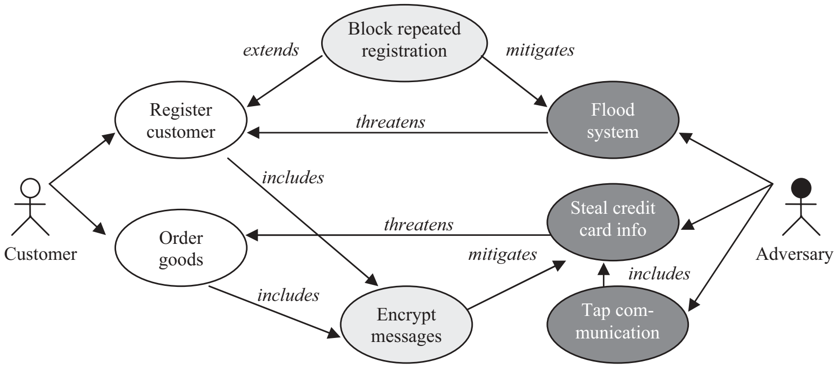


Figure 6. A use/misuse/mitigation use case diagram.

- A process constraint is a restriction on the techniques or resources that can be used to build the system.

In general, security requirements are nonfunctional requirements. However, they can entail functional requirements and additional constraints on the development process. For example, user registration can be considered as a functional requirement that is raised by the need of authentication. Security requirements are not independent of functional requirements, design constraints, or process constraints. Security requirements analysis should therefore be an integral part of the requirements analysis process. Modeling notations for security requirements are usually extensions to those for functional requirements. For example, the misuse case approach (10–15) is based on the use case approach, the anti-goal approach (16) is based on the goal-oriented requirements analysis, and the abuse frame approach (17) extends the problem frames.

Misuse Cases

The misuse case approach centers a round an interactive modeling of use cases, misuse cases, and mitigation use cases. From the adversary's perspective, use cases, misuse cases, and mitigation use cases (or security use cases) describe the intended behaviors (i.e., functional requirements), security threats, and security mitigations (i.e., security requirements), respectively. In the interactive modeling, we first identify use cases; each use case defines the interaction between an external actor (role played by a person or thing) and the system to accomplish a goal. Then, we identify potential misuse cases with respect to the use cases; each misuse case defines the interaction between an adversary (or mis-actor) and the system to accomplish the goal of attacking the system by exploiting the use cases. For example, "Steal credit card information" is a misuse case that threatens the use case "Order goods" in an online shopping application. Finally, mitigation use cases are suggested as countermeasures of the misuse cases. Each mitigation use case defines the requirement for mitigating the misuse cases. For example, "Encrypt messages" is a mitigation use case for the misuse case "Steal credit card information." The interactive modeling is often iterative, starting with a small set of use cases and dealing with more use cases in the next iteration. As mitigations are subject to misuses, misuses of mitigation use cases should also be examined in each cycle.

Use/misuse case diagrams and textual descriptions are often used to document use cases, misuse cases, and mitigation use cases and the relationships among them. Figure 6 shows some use cases ("Register customer" and "Order goods"), misuse cases ("Flood system", "Steal credit card information", "Tap communication"), and mitigation use cases ("Block repeated registration," "Encrypt messages") in an online shopping application. As in the traditional use case approach, "includes," "extends," and "generalization" can be used to label the relationships between misuse cases. For example, "Steal credit card information" includes "Tap communication." These relationships may also exist between mitigation use cases and regular use cases. "threatens" can be used to label the link from a misuse case to the corresponding use case(s), and "mitigates" to depict the relationship between a mitigation use case and the corresponding misuse case.

A use/misuse/mitigation use case diagram provides a high-level view of the intended functions and security requirements. To better understand the system, however, many details need to be described. In Fig. 6, for example, "Steal credit card information" is a misuse case with regard to the use case "Order goods." It makes no sense unless the use case requires the customer to provide credit card information. To complement a use/misuse/mitigation use case diagram, textual descriptions are often used to capture the detailed use cases, misuse cases, and mitigation use cases. A use case template in the traditional use case approach, although no standard, exists usually includes case number, case name, goal, actors, preconditions, main flow of events, alternative flows, related business logic, and postconditions. This template can be used as a basis for describing use cases, misuse cases, and mitigation use cases. To keep track of the relationships between use cases, misuse cases, and mitigation use cases, we can add a new entry for enumerating misuse cases and mitigation use cases to the template for use cases and mitigation use cases. Table 2 shows the textual description of the use case UC 11 "Order goods." In Step 6 of the main flow of events, the customer submits credit card information. This is where

Table 2. Textual Description of the Use Case "Order Goods"

Case Number	UC11
Case name	Order goods
Goal	To order goods from the system
Actors	Customer, system
Preconditions	1. The customer is registered. 2. The customer has logged on to the system.
Main flow of events	1. The customer enters order goods section. 2. The system displays the customer's account detail. 3. For each product that the customer wishes to order, the customer enters its identity. 4. The customer provides delivery details. 5. The system calculates and displays the price of the goods ordered. 6. The customer submits credit card information. 7. The system confirms the result of transaction. 8. The system collects the detail of the order. 9. The system processes the order.
Postconditions	1. The order and its detail are entered in the system and the order is processed.
Misuse cases and mitigation cases	MC10 Steal credit card info (Step 6 in the main flow of events) SC5 Encrypt messages

the use case is threatened by the misuse case MC 10 "Steal credit card information." Therefore, the last entry of the table lists the misuse case (MC10) and the corresponding mitigation use case (SC 5).

Anti-Goals

The anti-goal approach (16) is based on the goal-oriented requirements analysis, in which goals prescribe intended behaviors. Goals may refer to the intended services (functional goals) or to quality of service (nonfunctional goals). They are organized in AND/OR refinement-abstraction hierarchies in which higher-level goals are strategic and coarse grained, whereas lower-level goals are technical and fine grained. AND-refinement links relate a goal to a set of subgoals whereas OR-refinement links may relate a goal to a set of alternative refinements. Goal refinement ends when every subgoal is realizable. A requirement is a terminal goal under responsibility of an agent (active component such as a human and device).

To produce requirements for more robust systems, obstacle analysis (18) takes a pessimistic view at the goals. It tries to identify and resolve as many ways of obstructing the goals as possible. An obstacle to some goal is a condition that prevents the goal from being achieved. In the context of security, threats are obstacles or anti-goals intentionally set up by attackers. Threat trees are built systematically through anti-goal refinement until leaf nodes are either software vulnerabilities or anti-requirements. New security requirements are then obtained as countermeasures by application of threat resolution operators to the specification of the anti-requirements and vulnerabilities.

SECURE SOFTWARE DESIGN

Secure software design is a process of transforming the software security problem into a solution. It must meet the security requirements and avoid potential vulnerabilities caused by the design decisions.

Principles of Secure Design

In addition to the principles of traditional software design, such as abstraction and modularization, the design of secure software should also follow the general principles for security design. Six principles of secure software design are discussed below. Although they all originate from the general design principles of computer and information security (19), the discussion below places an emphasis on software applications. When these principles are applied to a specific application, they may need to be scoped and revisited to resolve potential conflicts. More design principles can be found in Refs. 20–22.

Least Privilege. The principle of least privilege states that a subject (user or software component) should be given only those privileges necessary to complete a task. If a subject does not need access to an object to perform its task, then it should not be granted that access privilege. If a subject needs extra privileges required of a specific action, then the extra privileges should be relinquished immediately on completion of the action. For example, if a web server is only responsible for serving marketing pages, then it should only be given access to the exact set of files it serves to its clients.

Simplicity. The principle of simplicity is also known as the principle of economy of mechanism. It states that security design should be as simple as possible. Complex design is more difficult to understand, test, and maintain. It is likely to have more possibilities for errors. For example, the UNIX sendmail utility that routes mail from a sender to a recipient is a complicated program with many security holes. Complex design often makes assumptions about the system and environment. Security problems may develop when these assumptions are made incorrectly. To achieve simplicity and reduce the number of security holes, a secure design should try to localize security checks. For example, choke points and aspect-orientation can serve this purpose. A choke point is a centralized component through which all control must pass. Aspect-orientation can modularize security concerns into separate components.

Secure Defaults. The principle of secure defaults states that a user or software component should be denied access to an object unless it is given explicit access to that object or the access request is found to be consistent with security policy. When the software is designed, it should, by default, be optimized for security wherever possible. The principle of secure defaults is of particular importance for software deployment—the initial configuration should not turn on every possible feature and make every service available to the user by default.

Secure Failure. The principle of secure failure states that a failure or exception in a function or mechanism should not lead to violation of security policy. When the failure or exception occurs, the software should be kept in a state that denies rather than grants access. When a user or software component has access to an object but cannot complete its task, it should still ensure some level of security. It should undo the changes that it had made in the secure state of the system before it terminates. It should not report error messages that reveal useful information to malicious users. For example, a basic trick of Structured Quens Language (SQL) injection attacks is to first determine the names of data fields in a database by exploiting the error messages for maliciously crafted SQL queries. Some SQL servers by default include the names of data fields in the error messages.

Defense in Depth. The principle of defense in depth states that a secure design should manage security risk with diverse or hierarchical strategies rather than rely on any one defense. As mentioned before, mitigation techniques or secure mechanisms cannot prevent all potential attacks because they are subject to their own threats. To protect sensitive information on a server, for example, one can integrate a hierarchy of defense strategies, such as a firewall, authentication, encrypted storage, and encrypted messages. The defense in depth would significantly increase the difficulty and costs of a full breach. This defense also contributes to secure failure—the software can continue securely even when one strategy has failed. As mitigation techniques are not perfect, a secure design should also consider mechanisms that help to detect and recover from attacks.

Usability. The principle of usability states that security mechanisms should not make the software more difficult to use than if the security mechanisms were not present. The extra burden on users added by the security mechanisms should be minimal and reasonable. From the usability study of a program for sending and receiving encrypted email, Whitten and Tygar (23) observed that most users were not able to successfully send or receive encrypted e-mail, even though the user interface seemed "reasonable." According to Whitten and Tygar, secure software is usable if the users: "(1) are reliably made aware of the security tasks they need to perform; (2) are able to figure out how to successfully perform those tasks; (3) don't make dangerous errors; and (4) are sufficiently comfortable with the interface to continue using it."

Security Design Patterns

As many security problems are shared by various distributed applications, security design patterns can provide reusable security solutions from application to application. A security design pattern is a generic, well-understood solution to a recurring security problem. For example, passwords are a widely accepted approach to remote user authentication. To mitigate the common threat of password cracking via brute force (e.g., guessing), one can block repeated password attempts through the security pattern "account lockout." This pattern imposes on a limit on the number of incorrect password attempts. More patterns can be found in Refs. 26 and 27.

Beyond design patterns, security patterns can also be used to improve the development process of secure software. For example, the Security Patterns Repository (Version 1.0) (24) describes 13 patterns (called procedural patterns) that often impact the organization or management of a software development project.

SECURE CODING

Secure design does not automatically guarantee a secure implementation. Secure coding must not only correctly implement all security policies and mechanisms (e.g., authentication and authorization) of a secure design, but also it must avoid potential implementation-level vulnerabilities caused by the chosen programming languages, compilers, tools (e.g., database systems and commercial software packages), and platforms (e.g., operating systems). Relatively more work has occurred on secure coding than any other aspects of secure software development. A great variety of implementation-level vulnerabilities and countermeasures has been identified. A root cause for most vulnerabilities is inadequate validation of inputs from users and software components (of course, adequate input validation is not as easy as one might think). This section introduces a few of the most common problems (buffer overflow, format string, SQL injection, and cross-site scripting). Some good practices of secure coding are also suggested. For further reading, we refer the reader to Refs. 3, 7, 21, 22, 27 and 28.

Buffer Overflow

Buffer overflow has been and likely continues to remain the principal method for remotely injecting malicious code into target software. A buffer overflow, or buffer overrun, is an anomalous condition in which a program attempts to store data beyond the boundaries of a buffer (e.g., stack and heap) because of insufficient bounds checking. For example, it can happen if a string of characters is copied from one buffer to another when the length of the string is greater than the size of the target buffer. As a consequence, adjacent memory is overwritten by the extra code and data that may cause a program to crash or produce incorrect results. In particular, the extra code and data can be injected maliciously to make the program operate in an unintended way.

The techniques to exploit buffer overflow fall into two categories: stack overflow and heap overflow. An exploit of stack overflow attempts to manipulate a program by (*1*) overwriting a local variable that is near the buffer in memory on the stack to change the behavior of the program, (*2*) overwriting the return address in a stack frame, or (*3*) overwriting a function pointer or exception handler, which is subsequently executed. An exploit of heap overflow attempts to corrupt a heap to cause the program to overwrite the internal structures such as linked list pointers. A heap is an area of memory that is dynamically allocated by a program at run-time. The canonical heap overflow technique overwrites dynamic memory allocation linkage and uses the resulting pointer exchange to overwrite a function pointer.

Various techniques exist for detecting or preventing buffer overflows with various tradeoffs (http://en.wikipedia.org/wiki/Buffer_overflow):

- Choosing a type-safe programming language can avoid many buffer overflow problems. It requires a careful consideration of tradeoff between safety and performance costs. Among the most popular languages, C and C++ provide no built-in protection against accessing or overwriting data in any part of memory. The Java and .NET bytecode environments are relatively safer.
- Correct use of safe libraries of an unsafe language can prevent most buffer overflow vulnerabilities. For example, strings and arrays are the two main building-block data types in C and C++ in which buffer overflows commonly occur. When C/C++ is chosen for the implementation, safe library implementations for these datatypes should be used.
- Stack-smashing protection can detect the most common buffer overflows by checking that the stack has not been altered when a function returns. If it has been altered, then the program exits with a segmentation fault.
- Executable space protection can prevent execution of code on the stack or the heap. Any attempt to execute that code will cause an exception.
- Address space layout randomization randomizes the virtual memory addresses at which functions and variables can be found. This method can make exploits of buffer overflows more difficult, but not impossible.
- Deep packet inspection attempts to block packets which have the signature of a known attack or have a long series of no-operation instructions.

Format String

The format string problem stems from the use of unfiltered user input as the format string parameter in certain C/C++ functions that perform formatting, such as printf(). A malicious user may use the %s and %x format tokens to print data from the stack or possibly other locations in memory or use the %n format token to write data or code to the memory. This error would lead to the same problems as buffer overflows: crashing the program or executing the maliciously injected code. Format string vulnerabilities develop because C/C++'s argument passing conventions are not type safe. Many compilers can check format strings and report warnings for dangerous or suspect formats.

SQL Injection

SQL is a standard programming language for retrieving information from and updating a database. SQL injection is a technique that injects SQL statements into an input (e.g., user-supplied identity and password in a web page) for malicious purposes (e.g., obtaining other users' passwords and unauthorized access to a database). It exploits security vulnerability in the database layer of an application.

Primary forms of SQL injection vulnerabilities (http://en.wikipedia.org/wiki/SQL_injection) are as follows:

- Incorrectly filtered escape characters. User input is passed into a SQL statement without being filtered for escape characters. It may allow for potential manipulation of the statements performed on the database by the end user.
- Incorrect type handling: SQL injection may occur when a user supplied field is not strongly typed or is not checked for type constraints.
- Vulnerabilities inside the database server, such as MySQL server's real_escape_chars() functions.

Techniques for avoiding SQL injection include:

- Input sanitization. User inputs are sanitized to ensure that they contain no dangerous code.
- Security privileges. Setting security privileges on the database to the least required. For example, the delete rights to a database for end users are seldom required.
- Disabling literals. SQL injection can be avoided if the database engine supports a feature called disabling literals, where text and number literals are not allowed as part of SQL statements.

Cross-Site Scripting

Cross-site scripting (XSS) is a technique that injects code (e.g., HTML code and client-side scripts) into the web pages viewed by other users. An XSS vulnerability can be exploited to bypass access controls and produce phishing attacks and browser exploits.

Three forms of XSS vulnerabilities exist (http://en.wikipedia.org/wiki/Cross-site_scripting):

- Vulnerability in a client-side script. For example, if a piece of JavaScript accesses a URL request parameter and uses this information to write some HTML to its own page, and it this information is not encoded using HTML entities, then an XSS hole will likely be present, because this written data will be reinterpreted by browsers as HTML that could include additional client-side script.
- Non-persistent or reflected vulnerability. It occurs when data provided by a web client is used immediately by server-side scripts to generate a page of results for that user. If user-supplied data is not validated but included in the resulting page without HTML encoding, this will allow client-side code to be injected into the dynamic page.
- Stored or persistent or second-order vulnerability (also referred to as HTML injection) can occur when data provided to a web application by a user is first stored persistently on the server (in a database, file system, or other location), and it is later displayed to users in a web page without being encoded using HTML entities.

Techniques for avoiding XSS exploits, among others, include encoding of all user-supplied HTML special characters so as to prevent them from being interpreted as HTML and input validation of all potentially malicious data sources.

Good Secure Coding Practices

The principles of secure design in previous section also apply to detailed code design. Besides these principles, the Top 10 Secure Coding Practices by Robert Seacord's (https://www.securecoding.cert.org/) include the following:

- Validate input:

  Validate input from all untrusted data sources. Proper input validation can avoid most implementation-level vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files.

- Heed compiler warnings:

  Compile code using the highest warning level available for the compiler and eliminate warnings by modifying the code.

- Sanitize data sent to other systems:

  Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf components. Attackers may invoke unused functionality in these components through the use of SQL, command, or other injection attacks. It is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem.

- Adopt a secure coding standard:

  Develop and/or apply a secure coding standard for the chosen language and platform. The CERT Secure Coding Standards for C and C++ can be found at https://www.securecoding.cert.org/.

VALIDATION AND VERIFICATION FOR SECURITY ASSURANCE

Security assurance provides justification that security mechanisms, as implemented and operated, meet security requirements through assurance evidence and approvals based on evidence (2). Many problem sources exist in the development of secure software, such as omissions and mistakes in requirements specifications, design flaws, and implementation errors. Assurance addresses each of these problem sources. For instance, design assurance is the evidence establishing that a design is sufficient to meet the requirements of the security policy. In the following, we discuss several common approaches to security assurance. It should be noticed, however, that no silver bullet exists for security assurance. Various techniques are often used throughout software development to improve the level of assurance.

Review for Security

Review is a common practice of software development. Review of specification, design, and code can identify and eradicate many problems at various development phases, such as omissions and inconsistency in requirements specification, flaws in design, and programming errors (e.g., format string and buffer overflow vulnerabilities). Review practices include formal inspections and lightweight reviews. A formal inspection involves a careful and detailed process with multiple participants and multiple phases. A lightweight review requires less overhead than formal inspections. Review approaches include over-the-shoulder (one developer looks over the author's shoulder as the latter walks through the documentation or code), pair programming, email pass-around, and tool-assisted review (e.g., anonymous CVS allows groups of individuals to collaboratively review code).

Assurance Arguments and Formal Methods

Assurance arguments fall into two main categories: property-based and threat-based arguments. A property-based argument is a statement about a security property or constraint [e.g., if a user is authorized by the given security policy to perform operations on a resource, then the system must grant the access request (29)]. A threat-based argument is a statement about whether a threat exists (e.g., an adversary can intercept payment information). Property-based and threat-based arguments imply different strategies for proof or disproof. To verify a property-based argument, one can check to observe whether the specification or code *always* satisfies the property or constraint. To verify a threat-based argument, one can check to observe whether a chance *exists* for the threat to be realized according the specification or code. However, both approaches can take advantage of formal methods for specifying and reasoning about software security. Formal methods are mathematical tools for rigorous specification and (partially) automated verification. In the property-based approach, properties are often formalized as logical formulas and then proved via either theorem-proving or model-checking. In the threat-based approach, security can be assured through refutation. A threat-driven security design (5), for example, is said to be secured from the anticipated security threats under two conditions: (*1*) It is possible for the security threats to occur in the intended functions alone—the functions are insecure and the security threats are identified correctly; and (*2*) the security threats will never take place after the security features are applied—they are indeed absent from the security design unless the security features themselves are compromised. As a matter of fact, this style of refutation for assurance can be applied to threat modeling, no matter whether the modeling notation is formal (5) or informal (12,30).

Static Code Analysis for Security

Static code analysis is the analysis of source code or a compiled form of the program (e.g., object code or byte code) that is performed without actually executing the code (31). Often supported by automated tools, static analysis can help to identify and eradicate many coding problems, including security vulnerabilities, before a program is released. The basic idea of static analysis is to look for a predefined set of patterns or rules in the code that indicate possible security vulnerabilities. As the results of static analysis are not perfect, they require additional evaluation. For example, a static analysis tool can produce false negatives or false positives (32). False negatives mean that the program contains vulnerabilities that the tool does not report, whereas false positives refer to reported vulnerabilities that the program does not contain. False negatives are very dangerous because they lead to a false sense of security.

Static analysis tools often take advantage of compiler technology. For example, some tools are based on lexical analysis. They preprocess and tokenize source files and then match the resulting token stream against a library of vulnerable constructs such as gets(&buf) in the C language. Annotations can be used to help the analysis. For example, CQual (33) requires a C programmer to annotate a few variables as either tainted or untainted and then uses type inference rules to propagate the qualifiers. Once the qualifiers are propagated, type checking can reveal format string vulnerabilities in the C program. Splint (33) allows developers to add annotations for finding abstraction violations, unannounced modifications to global variables, and possible use-before-initialization errors. Computer-aided verification techniques have also been applied to static analysis. The Eau Claire tool (31) uses a theorem-prover to create a general specification-checking framework for C programs. It can help find common security problems like buffer overflows, file access race conditions, and format string bugs. MOPS (35) takes a model-checking approach to look for violations of temporal safety properties. Developers can model their own safety properties, and some have used the tool to check for privilege management errors, incorrect construction of chroot jails, file access race conditions, and ill-conceived temporary file schemes.

Software Testing for Security

Different from static code analysis for security, software testing for security is the process of executing a program with the intent of finding vulnerabilities. Software testing exercises a program with not only positive test cases for verifying that the system does what it is supposed to do (e.g., access is granted to authenticated and authorized users), but also negative or dirty test cases for verifying that the system does not do what it is not supposed to do (e.g., access is not granted to unauthorized users). Although both positive and negative tests are required of testing of any software, testing for security must pay more attention to negative tests because by nature they verify whether the system can be misused. This finding indicates that security testing is hard because too many possible negative tests can occur. It needs to test the "presence of an intelligent adversary bent on breaking the system"(36), which is the major difference between software security and software safety.

Security testing of software applications is different from network and system testing for security, such as network scanning, vulnerability scanning, password cracking, log reviews, file integrity checkers, virus detectors, and war driving for identifying unauthorized modems or wireless access points. When feasible, however, it can extend or take advantage of existing system security testing techniques, particularly penetration testing. A penetration test is a traditional method of evaluating the security of a computer system or network by simulating an attack by an adversary, known as a cracker. This testing is conducted from the position of a potential attacker and can involve active exploitation of security vulnerabilities. The underlying idea is aligned with the adversary's perspective of software security and applicable to security testing of software applications. Specifically, one can derive security tests as attack attempts from threat models (e.g., threat trees, threat nets, misuse cases) and threat mitigations. Then the code is exercised with the derived tests to determine whether the software is free from the attacks. This strategy is an example of black-box testing, which views the software under test as a black box without knowledge of internal workings of the code. As security threats are pervasive and software testing is highly labor-intensive, security testing should be based on risk assessment and threat prioritization (called risk-based testing),

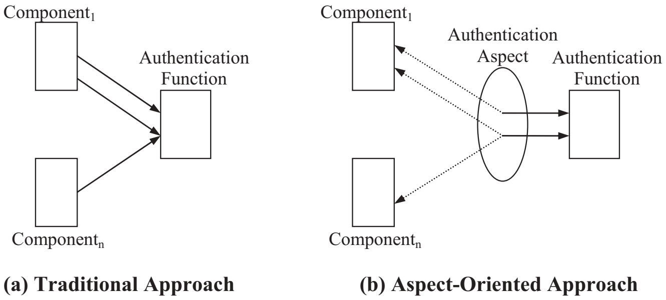


Figure 7. Tradition approach versus aspect-oriented approach.

which is usually an integral part of the threat modeling process. Black-box testing closely simulates the actions of an actual cracker and tests whether security mechanisms are properly implemented. It is yet weak at detecting the attacks that require some implementation-level knowledge of the software. At the other end, white-box testing takes advantage of the knowledge of the internal workings from the source code. It is often very effective in finding programming errors. According to the amplification phenomenon of vulnerabilities in software development processes, all different types of vulnerabilities (e.g., omissions in specification, flaws in design, and implementation specific faults) might accumulate in the code. To detect and reduce these vulnerabilities, various testing strategies are often required.

ADVANCED TOPICS: ASPECT-ORIENTED DEVELOPMENT OF SECURE SOFTWARE

Separation of concern is one important software engineering principle. Aspect-oriented programming (AOP) (37) is a new paradigm for separating concerns that crosscut multiple system components. The representative AOP language is AspectJ, which is an extension to the Java programming language. AOP modularizes crosscutting concerns into aspects with the advice invoked at the specified points of program execution. An aspect-oriented program consists of base components (or classes) and aspects that can be woven into an executable whole. The base classes in an aspect-oriented program can also be executed independently. This model allows for incremental or separate development of aspects. An aspect is an encapsulated entity of inter-type declarations, pointcuts, and advice. Inter-type declarations introduce new members such as instance variables and methods to the base classes. A pointcut is a collection of join points, and each join point is a well-defined point (e.g. method-call) in the flow of a program execution. A piece of advice for a pointcut defines additional code to be executed at each of the join points picked out by the pointcut. Originated from AOP, aspect-oriented software development (AOSD) aims to deal with crosscutting concerns at different development phases, (e.g., requirements analysis, architectural design, and detailed component design).

Security is in natural a crosscutting concern that involves many components of a software application. For example, authentication is often required of many system components in an online shopping application. With a traditional approach, each of the places that require authentication would first need to call an authentication method, as shown in Fig. 7(a). With the aspect-oriented approach, however, the authentication concern is separated from these components and modularized into an authentication aspect, as shown in Fig. 7(b). These components, when first developed, are not concerned about authentication. The authentication aspect will collect all places that are required of authentication (i.e., join points) and define what authentication method is applied (i.e., advice). In the traditional approach, security concerns are scattered across many components. This approach may require frequent, close interaction between nonsecurity and security personnel on the development team. Generally speaking, this method would make the development of secure software ineffective and inefficient because of the common shortage of security experts available. AOSD also fits in the adversary's perspective of software security—both security threats and threat mitigations have a nature of crosscutting. Notice that AOSD itself is not a security feature. It can help to address software security issues primarily because it caters for the principle of simplicity through improved modularization.

Aspect-Oriented Security Requirements

Aspect-orientation offers a better way to structure security threats and mitigation requirements. In the misuse case approach, one can easily identify these issues: (*1*) a single misuse case can threaten a use case at different steps of its main or alternate flows of events, (*2*) a single misuse case can threaten multiple use cases, and (*3*) a single step in the main or alternate flows of events of a use case can be threatened by multiple misuse cases. For example, step 6 "the customer submits credit card information" in the main flow of events of the use case "Order goods" in Table 2 is subject to such threats as block system, replay of message, modification of messages, and tap communication. The same issues exist for mitigation use cases. As such, the relationships among use cases, misuse cases, and mitigation use cases can be very complex. If a single use/misuse case diagram is used to depict all use cases, misuse cases, mitigation use cases, and their relationships, then the diagram likely becomes unreadable.

In the aspect-oriented misuse case approach developed by Xu et al.(38), misuse cases are called threat aspects, whereas mitigation use cases are called mitigation aspects. A point in a use case section (e.g., step 6 in the main flow of events of use case “Order goods”) can be referenced as a join point. If a join point is a point at which a security threat can happen, then it is called a threat point (a point of vulnerability). The various join points that are threatened by a particular threat are grouped together in a pointcut. That is, a threat pointcut is a collection of join points that are vulnerable to a common threat. Thus, each threat aspect consists of a group of pointcuts and their advice (how the threats can happen at corresponding threat points). After crosscutting threats are specified, one can identify the requirements for mitigating the threats. A mitigation aspect consists of mitigation pointcuts and advice. Each mitigation pointcut includes one or more threat pointcuts, which means that a mitigation can apply to one or more threats. A piece of mitigation advice contains the mitigation logic required to countermeasure the identified threats. Therefore, each mitigation aspect indicates a security concern. The above aspect-oriented organization can help different stakeholders understand requirements specifications by better clarifying the following questions: (*1*) For a given use case, what are the identified misuse cases and mitigation use cases? (*2*) For a given misuse case, what use cases does it threaten and what mitigation use cases are identified? (*3*) For a mitigation use case, to what use cases and misuse cases does it apply?

Aspect orientation can also be integrated into other approaches to security requirements analysis. Based on the notion of problem frames for elicitation of functional requirements, for example, Haley et al.(39) have developed away to derive security requirements from crosscutting threat descriptions.

Aspect-Oriented Design for Security

Aspect orientation can contribute to the principle of simplicity for secure design by separating crosscutting security policies and mechanisms from functional components. For example, enforcing access control policies in a secure design may spread the access control concern across many design modules. The interference of the access control concern with other application behavior can make it difficult to understand, analyze, and evolve the access control concern (40). Using an aspect-oriented modeling approach to addressing the pervasive access control concern of an application, one can localize the access control concern in an aspect. Aspect orientation also fits in the adversary’s perspective of secure software design. One can first identify security threats with respect to the components of intended functions and then specify mitigation mechanisms as aspects (5).

AOP for Security

An obvious application of AOP to security is separation of crosscutting security code from functional code for better modularization (41). Another application is to improve the security of legacy applications. According to Viega et al. (42), a large portion of commercial applications have significant security problems that are present in the design phase and persist through to implementation. Penetrate-and-patch is the traditional approach to resolving security problems—after vulnerabilities are revealed, the source code is modified, and then the revised version is compiled and released. With AOP, however, some security problems of a legacy application can be addressed without direct modification to the legacy code. To this end, aspects can be used to insert code before or after points of interest or replace the code at the points of interest. Sample scenarios are replacing insecure function calls with secure replacements, performing error-checking on security-critical calls, and logging data that are relevant to security.

AOP also offers a potential for better reuse of security solutions. Various core security requirements are common to many distributed software applications. Although these applications can follow security design patterns for respective security requirements, the implementation of the security patterns often vary from application to application. With AOP, a security design pattern can be realized as an abstract aspect that encodes the security solution in advice. When an application adopts a security design pattern, it only needs to extend the abstract aspect of the pattern and provide concrete pointcuts. The concrete pointcuts collect the points of program execution at which the security solution should be applied.

BIBLIOGRAPHY

1. J. Wing, A symbiotic relationship between formal methods and security, *Proc. of NSF Workshop on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solution*, 1998, pp. 26–38.
2. M. Bishop, *Computer Security: Art and Science*, Reading, MA: Addison-Wesley, 2003.
3. G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*, Reading, MA: Addison-Wesley, 2004.
4. P. G. Neumann. Principled assuredly trustworthy composable architectures, *Project Report*, Computer Science Laboratory, Menlo Park, CA: SRI International, 2004.
5. D. Xu and K. E. Nygard, Threat-driven modeling and verification of secure software using aspect-oriented Petri nets, *IEEE Trans. Softw. Engin.***32** (4): 265–278, 2006.
6. B. Schneier, Attack trees, *Dr. Dobb’s J. Softw. Tools***24** (12): 21–29, 1999.
7. M. Howard and D. LeBlanc,*Writing Secure Code*, 2nd ed. Redmode WA: Microsoft Press, 2003.
8. F. Swiderski and W. Snyder, *Threat Modeling.*, Redmond, NA: Microsoft Press, 2004.
9. S. L. Pfleeger, and J. M. Atlee, *Software Engineering: Theory and Practice*, 3rd Ed., Englewood Cliffs, NJ: Prentice Hall, 2006.
10. I. Alexander. Misuse cases: Use cases with hostile intent,*IEEE Soft.*, **20**: 58–66, 2003.
11. D. G. Firesmith. Security use cases, *J. Object-Technol.*, **2** (3): 53–64, 2003.
12. J. McDermott, Abuse-case-based assurance argumentss, *Proc. of the 17th Annual Computer Security Application Conference (ACSAC’01),*2001, pp. 366–374.
13. J. McDermott and C. Fox, Using abuse case models for security requirements analysis, *Proc. of the 15th Annual Computer Security Application Conference (ACSAC’99)*, 1999, pp. 55–66.

14. R. W. Shirey, Security architecture for Internet protocols: A guide for protocol designs and standards. *Internet Draft: draft-irtf-psrg-secarch-sect1 -00*, November 1994.

15. G. Sindre and A. L. Opdahl, Eliciting security requirements by misuse cases, *Proc. of TOOLS Pacific*, 2000, pp. 120–131.

16. A.van Lamsweerde, Elaborating security requirements by construction of intentional anti-models, *Proc. of ICSE'04*, pp. 148–157, 2004.

17. L. Lin, B. A. Nuseibeh, D. C. Ince, M. Jackson, and J. D. Moffett, Analyzing security threats and vulnerabilities using abuse frames. Technical Report No. 2003/10, Millon Keynes Uk: The Open University.

18. A. van Lamsweerde and E. Letier, Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Softw. Engin.***26** (10): 978–1005, 2000.

19. J. Saltzer and M. Schroeder, The protection of information in computer systems, *Proc. of the IEEE*, **63** (9): 1278–1308, 1975.

20. T. V. Benzel, C. E. Irvine, T. E. Levin, G. Bhaskara, T. D. Nguyen, and P. C. Clark,Design principles for security, *SecureCore Technical Report*, ISI-TR-605 and NPS-CS-05-010, Monterey and Los Angeles, CA: Naval Postgraduate School and University of Southern California, 2005.

21. N. Daswani, C. Kern, and A. Kesavan, *Foundations of Security: What Every Programmer Needs to Know*, Berkeley CA: Apress, 2007.

22. J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems in the Right Way*, Reading, MA: Addison Wesley, 2002.

23. A. Whitten and J. D. Tygar, Why Johnny can't encrypt: A usability evaluation of PGP 5.0, *Proc. of the 8th USENIX Security Symposium*, 1999, pp. 169–183.

24. D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt, *Security Patterns Repository,* Version 1.0. Available: http://www.scrypt.net/~celer/securitypatterns/

25. M. Hafiz, A collection of privacy design patterns, *Pattern Languages of Programs (PLoP) Conference*, 2006.

26. B. Blakely and C. Health, *Security Design Patterns*, Berkshire, UK: The Open Group, 2004.

27. C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*, Englewood chiffs, NJ: Prentice Hall, 2005.

28. M. Howard, D. LeBlanc, and J. Viega, *19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*, New York: The McGraw-Hill Companies, 2005.

29. Y. Deng, J. Wang, J. Tsai, and K. Beznosov, An approach for modeling and analysis of security system architectures, *IEEE Trans. Knowl. Data Engineer.* **15** (5): 1099–1119, 2003.

30. D. Xu and J. Pauli, Threat-driven design and analysis of secure software architectures, *J. Informat. Assur. and Secur.*, **1** (3): 171–180, 2006.

31. B. Chess, Improving computer security using extended static checking, *Proc. IEEE Symp. Security and Privacy*, 2002, pp. 118–130.

32. B. Chess and G. McGraw, Static analysis for security, *IEEE Secur. Priv.*, **2**: 76–79, 2004.

33. J. Foster, T. Terauchi, and A. Aiken, Flow-sensitive type qualifiers, *Proc. of ACM Conf. Programming Language Design and Implemtation (PLDI2002)*, 2002, pp. 1–12.

34. D. Larochelle, and D. Evans, Statically detecting likely buffer overflow vulnerabilities, *Proc. 10th Usenix Security Symp (USENIX'01)*, 2001, pp. 177–189.

35. H. Chen, D. Dean, and D. Wagner, Model checking one million lines of C code, *Proc. of the 11th Annual Network & Distributed System Security Symp. (NDSS)*, 2004, pp. 171–185.

36. B. Potter, B. Allen, and G. McGraw, Software security testing, *IEEE Security & Priv.*, **2**: 81–85, 2004.

37. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, and J. Irwin, Aspect-oriented programming, *Proc. of the European Conference on Object-Oriented Programming (ECOOP'97)*, 1997, pp. 220–242.

38. D. Xu, V. Goel, K. Nygard, and W. E. Wong, Aspect-oriented specification of threat-driven security requirements, *Internat. J. Comp. Applicat. Technol.*, In press.

39. C. B. Haley, R. C. Laney, and B. Nuseibeh, Deriving security requirements from crosscutting threat descriptions, *Proc. of the International Conference on Aspect-Oriented Software Development (AOSD'04)*, 2004, pp. 112–121.

40. I. Ray, R. France, N. Li, and G. Georg, An aspect-based approach to modeling access control concerns, *Informat. Softw. Technol.*, **46** (9): 575–587, 2004.

41. B. DeWin, B. Vanhaute, and B. De Decker, Security through aspect-oriented programming, *Proc of the First Annual Working Conference on Network security:* Advances in Network and Distributed Systems Security, 2001, pp. 125–138.

42. J. Viega, J. T. Bloch, and P. Chandra, Applying aspect-oriented programming to security, *Cutter IT J.*, **14** (2) 2001, pp. 31–39.

DIANXIANG XU
North Dakota State University
Fargo, North Dakota

SOFTWARE TESTING: TESTING NEW SOFTWARE PARADIGMS AND NEW ARTIFACTS

INTRODUCTION TO THE ARTICLE

The purpose of software testing is to *improve the quality* of the software product by detecting and removing as many failures as possible, so to increase the developer confidence in the proper functioning of the software. For this purpose, software testing is not meant to be an exhaustive technique for software verification and validation. It can show that software defects are present, while not being able to show their absence (1).

Software testing is centered on the concept of selecting *some* (fault enabling) inputs from a possibly infinite input domain, to perform system runs driven by such inputs, and to compare the expected results with the real ones. If expected and observed behaviors differ, then a failure is manifested because of a system fault that needs to be fixed. A successful test is the one that uncovers undiscovered errors.

By taking a look at the evolution of software testing in the past decades, we can recognize that two (among the others) can be considered the main sources of advances in software testing: *(1)* testing techniques have been applied over new software development paradigms, languages, and applications, and (*2*) testing techniques have been applied over new artifacts (other than the source code).

This article presents current testing techniques along those two directions.

Testing New Software Paradigms

Over the course of the past 50 years, the way software has been produced is changed greatly. We have moved from procedural code, toward object-oriented systems, to component-based development (as pictorically described in Fig. 1); from thousands to millions of lines of code with real-time, reliability, safety, and performance requirements. Software testing has had to change accordingly. Traditional techniques, adopted for testing procedural code, had to be extended for testing object-oriented (OO) software: inheritance, polymorphism, dynamic binding, and other OO characteristics required new testing features. With the introduction of component-based systems, component-based testing has been introduced for testing the components in isolation or the assembly.

Testing New Artifacts

In traditional approaches to software testing, specific methodologies are used to select test cases based on the source code of the program to be tested (2). The main practical drawback related to (purely) source code testing is that because the code is produced at the latest step in the software production process, testing activities are left to the end of the software lifecycle. In consequence, schedule slippage, time-to-market pressures, and cost-constraints result in neglected testing.

Nowadays, source code is no longer the single source for selecting test cases, and we can apply testing techniques all along the development process, by basing test selection on different pre-code artifacts (3); the test selection phase can be based on system specifications (formally or informally defined), on architectural high-level design, or on component-based or object-oriented specifications.

All these testing techniques deserve consideration. Recent studies have shown that different test selection techniques target different classes of faults (e.g., Ref. 4), and that the combined use of diverse techniques is always more effective than concentrating the effort on only one technique (even though proved to be the most efficacious) (5).

This Article

The goal of this article is to provide a view on software testing techniques from this specific perspective, as graphically described in Fig. 2. It discusses code-based testing with a specific focus on testing object-oriented software. It covers component-based testing and distinguishing between component-testing and component-based testing. It analyzes specification-based testing (with particular attention to formal testing, model-based testing, and software-architecture-based testing).

This article provides the most comprehensive treatment of the testing subject from the specific perspective presented above, which describes the state of the art and provides references for further readings and expected directions for future work. As the article is oriented to a wide audience of readers, a glossary of testing terms has been provided in Appendix A to facilitate the reading to inexperienced readers. As the study is limited in size, references are provided to those best sources an interested reader may refer to for detailing the concepts outlined in this work. The focus has also been oriented to mature research, to practices of current use, and to current trends, while limiting the discussion on the latest speculative research that, although promising, might never work out. Moreover, this article focuses on systematic testing techniques, while not considering purely experience-driven, not repeatable, testing techniques.

The remaining sections are structured according to the following template: problem statement, information on the main achievements, links to tool support, and references to further readings.

Further Readings

M.J. Harrold's (6) ICSE 2000 Future of Software Engineering track paper on software testing provides a clear and wide roadmap on fundamental research topics to be further investigated to obtain practical testing methods, tools, and processes to develop high-quality software. A. Bertolino's (7) ICSE 2007 Future of Software Engineering track paper

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

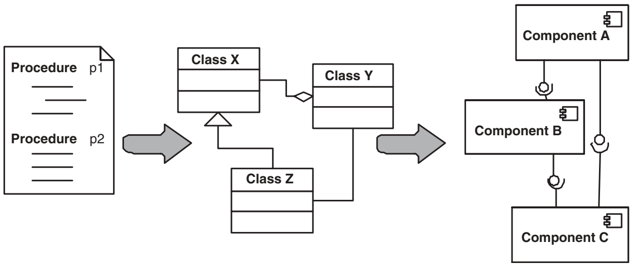


Figure 1. From procedural programming to component-based development.

on software testing provides a roadmap that structures and classifies software testing into three main aspects: achievements (where we are today), dreams (where researchers would like to go), and challenges (as a way to move from achievements to dreams). M. Pezzé and M. Young (8) authored a recent book on Software Testing and Analysis, which covers both the basic concepts and techniques, testing methods, and processes. *ISSTA* (9), *ICST* (10) and *TESTCOM* (11) are among the main conferences on software testing.

CODE-BASED TESTING: TESTING THE SOURCE CODE

Problem Statement

Code-based testing (also known as structural testing or white box testing) assumes visibility into internal data and structures of the system implementation. Test cases are selected according to the structure of the program and executed to exercise program statements.

The code structure is (typically) represented through a *graph:* Traditional white box analysis techniques use a control flow or data flow graph representation of a program. In *control flow* graphs, nodes correspond to sequentially executed statements whereas edges represent the flow of control between statements. A *data flow* graph extends the content of a control flow graph by adding information on variables accessed and modified by program statements.

Achievements

The *test selection* phase in code-based testing consists of selecting the minimum number of test cases that cover as much as possible of the flow graph. Designers have been applying white box techniques for a long time, and several coverage criteria are available today. Some criteria are based on the control flow, whereas others are based on the data flow.

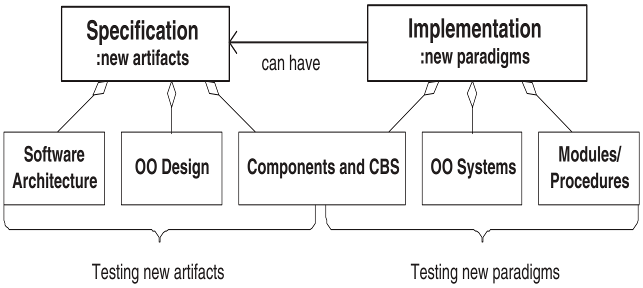


Figure 2. Testing new software paradigms and new artifacts.

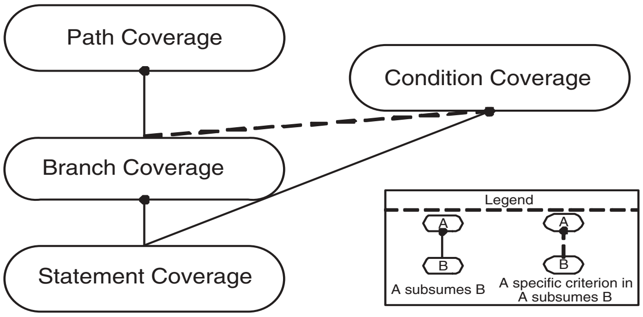


Figure 3. *Subsume* relationship among the analyzed coverage criteria.

Statement Coverage. This criterion reports whether *each executable statement* is encountered. This coverage selects a test set T such that, by executing a program P for each test in T, each elementary statement of P (i.e., each node in the graph) is executed at least once. The chief characteristic of this measure is that it provides the weakest coverage: If compared with other coverage criteria (see Fig. 3), then statement coverage is subsumed by any of the structural coverage criteria.

Branch Coverage. This criterion measures the coverage of *all blocks* and *case statements* that affect the control flow. Boolean expressions are evaluated for both true and false conditions. This criterion selects a test set T such that, by executing P for each test in T, each of P's control flow graph is traversed at least once (i.e., each branch of the control flow graph has to be executed by at least one test case). This measure has the advantage of simplicity, but it may ignore branches within Boolean expressions (see condition coverage below), or relevant combinations of branches (considered in path-based criteria).

Condition Coverage. This criterion measures the *subexpressions* independently of each other, allowing for a better analysis of the control flow. It forces the exploration of possible conditions of a boolean expression in a branch, which covers *different combinations of the individual conditions* in a compound boolean expression. This coverage criterion selects a test set T such that, for each test in T, each edge of P's control flow is traversed at least once and all possible values of the constituents of compound conditions are exercised at least once. Many variations of the condition coverage criterion exist (see Ref. 8 chapter 12 for a detailed explanation). Here, it is worth mentioning the modified condition/decision coverage (MCDC), which is required by many certification agencies: It enhances the condition/ decision coverage criterion requiring that each condition independently affecting the outcome of the corresponding decision has to be tested (12). MCDC is an attractive compromise between number of required test cases and thoroughness of the test, since with a complexity of $n+1$[1] (in

[1] $n+1$ test cases for a decision with n inputs

the best case), it achieves many of the benefits of the multiple-condition testing, which has a complexity of 2^n.

Path Coverage. It measures the coverage of all paths. Path coverage is similar to condition/decision coverage, but it handles multiple sequential decisions. As the number of paths soon becomes unfeasible, several variations of this criterion are considered to limit the number of loops. This technique selects a test set T such that, executing P for each test in T, all paths that lead from the initial to the final node of the P's control flow graph are traversed at least once.

Data-Flow Coverage. *Data-flow testing* (13–15) offers a family of criteria for unit testing of programs represented as *data-flow graphs*. It improves control-flow testing by identifying how the execution of a certain statement can affect the system computation. Data flow testing criteria pair *variable definitions* with *uses,* which ensures that each computed value is actually used, and thus selecting from among many execution paths a set that is more likely to propagate the result of erroneous computation to the point of an observable failure (8). Many coverage criteria are based on data flow, the most important being: *all DU pairs* adequacy criterion (it requires each DU pair to be exercised in at least one program execution), *all DU paths* adequacy criterion (extends the all DU pairs criterion by requiring each simple DU path to be traversed at least once) and *all definitions* adequacy criterion (which requires pairing each definition with at least one use).

Let P be the following program:

```
function P return INTEGER
begin
    X, Y:INTEGER;
    READ(X); READ(Y);
    while (X > 10) loop
        X := X - 10;
        exit when X = 10;
    end loop;
    if (Y < 20 and then X mod 2 = 0) then
        Y := Y + 20;
    else
        Y := Y - 20;
    end if;
    return 2 * X + Y ;
end P;
```

The corresponding control flow graph is shown in Fig. 4.

Some structural tests follow:

All-statement coverage: Inputs: (x = 20, y =10) and (x = 20, y = 30), where (x = 20, y = 10) covers nodes <1,2,3,4,5,6,8,9>, and (x = 20, y = 30) covers nodes <1,2,3,4,5,7,9>.

All-branches coverage: Inputs: (x = 20, y = 10), (x = 15, y = 30), and (x = 20, y = 15) where (x = 20, y = 10) covers nodes <1,2,3,4,5,6,8,9> and braches in <2T,4T,5T,6T>, (x = 15, y = 30) covers nodes <1,2,3,4,2,5,7,9> and braches in <2T,4F,2F,5F>, and (x = 20, y = 15) covers nodes <1,2,3,4,5,6,7,9> and braches in <2T,4T,5T,6F>.

It is important to remark that a 100% coverage is typically unfeasible because it may require the execution of unreachable code (referred as "infeasibility" problem).

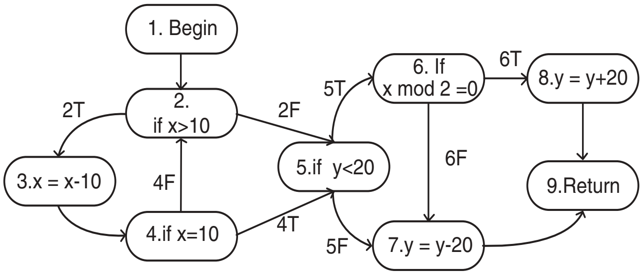


Figure 4. The control flow graph of the P program above.

Many other coverage criteria are used in practice. For more details, please refer to Refs. 16–19. How coverage criteria have been adapted to concurrent programs may be found in Ref. 20. An analysis of coverage criteria costs may be found in Ref. 21. A comparison of structural testing criteria can be found in Ref. 8.

Tool Support

As far as concerns tool support, the Open Source Testing Tools (22) website presents a list of open source testing tools (for white-box testing), whereas the Software QA Testing and Test Tool Resources (23) website presents a list of software quality-assurance and testing tools.

Further Readings

M. Pezzé and M. Young (8) authored a chapter (ch.12), entitled "Structural Testing," which presents the various structural testing techniques with examples. Chapter 13, which is entitled "Data Flow Testing," presents theory and examples on data flow testing.

Testing Object-Oriented Software: Testing the Source Code

Problem Statement. Testing OO software resembles testing procedural software, in which the focus moves from unit, to integration and system testing, and regression testing (24). However, OO software has some specific characteristics that make testing OO software diverse from other testing strategies and requires specialized techniques. In particular, as remarked in Ref. 25 and thoroughly discussed in Refs. 8 and 26, the unique facets of testing OO software are that each method must be tested in the context of its class and inherited features; objects are stateful and need to be tested in the different states; and encapsulation, dynamic binding, inheritance, and exceptions make the system behavior unpredictable so that an apparently harmless modification may affect different portions of the program.

Achievements. In code-based testing of OO software,[2] new criteria are introduced (extending the classic statements, branches, conditions, and decisions coverage) to cope with state-dependent behavior.

[2] Although this section will focus on structural testing of OO software, model-base testing techniques will be described in the Specification-Based Testing Section.

When considering *unit testing of OO code,* a unit is the class, and unit testing aims at testing the functions that belong to a class. As noticed in Ref. 8, although an individual method might be considered as a unit, testing a single method can become unpractical, because methods (in a class) interact by modifying object states and affect other methods' behavior. For this reason, unit testing of OO software is usually referred as *intraclass testing*. When performing intraclass testing of the source code, a graph model [typically, a control flow graph (CFG)] is built for each method, and CFGs of the various methods in a class are joined together to specify intraclass method calls. This joined graph is usually referred to as a class control flow graph (CCFG) (27). CCFGs are then used to compute def-use pairs, by first annotating the CCFG with definitions and uses and then traversing it to compute all pairs.

When different classes are integrated, *inter-class testing* checks interactions among objects of different classes. Souter and Pollock (28) define a *contextual def-use pair (d, u)* of a variable *o* to be used in interclass testing. Buy, et al. (29) combine dataflow analysis with symbolic execution and automatic deduction to generate test cases that satisfy data flow criteria. In Refs. 30 and 31, it is recognized that a representation based on simple CFGs is inadequate for interclass testing of Java programs. To handle all Java language constructs and features adequately (such as inheritance, polymorphism and dynamic binding, and exception handling), a Java interclass graph (JIG) representation is provided. A JIG extends the CFG to handle variable and object type information, internal or external methods, interprocedural interactions through calls to internal or external methods from internal methods, interprocedural interactions through calls to internal methods from external methods, and exception handling. The JIG is used in Refs. 30 and 31 for regression testing of Java programs.

Data flow testing criteria for computing definitions and uses of object attributes have been proposed in the context of unit and integration testing of OO software (as discussed in Ref. 26).

Tool Support. As far as tool support for testing object-oriented code, a recent study on tool support for white-box testing of OO software seems to be missing. Other than the popular JUnit unit testing framework for Java code (32), many other tools (typically, for structural intra class testing of Java or C++ programs) can be found in References 33–36.

Further Readings. D.C. Kung, et al. (37) authored a book that initially identifies difficulties and challenges in testing object-oriented software, then focuses on unit, integration, regression, and object state testing. J.D. McGregor and D.A. Sykes (24) cover the entire process in testing OO software, focusing on the what, why, how, who, and when. This text specifically focuses on testing classes, testing interactions, class hierarchies, and distributed objects. M. Pezzé and M. Young (8), chapter 15, entitled "Testing Object-Oriented Software" focuses on interclass and intraclass testing. It also focuses on testing in case of polymorphism, dynamic binding, inheritance, genericity, and exceptions. In L. Mariani and M. Pezzé (26), both specification-based and structural-based approaches for intra-class and inter-class testing of OO software are discussed.

COMPONENT-BASED TESTING

Roughly speaking, a component-based software system is an assembly of reusable components that are designed to meet the quality attributes identified during the architecting phase (38), whereas a component is defined as "a unit of composition with contractually specified interfaces and explicit context dependencies only" (39). Components are specified, designed, and implemented with the intention to be reused, and they are assembled in various contexts to produce a multitude of software systems. The main peculiarities in component-based software development are that components can be white-box or black-box (e.g., if components are off-the-shelf, their implementation is not accessible), they can be produced in house or bought from external vendors, they can be written in different programming languages and run in different execution environments, a detailed specification may exist or not.

Problem Statement

Component-based testing consists of two main tasks: testing the individual components (i.e., component testing), and testing the component-based systems built by assembling components.

In *component testing,* the goal is to detect software errors and to validate the quality of the software components considered as the smallest test unit (40). Testing of components becomes extremely important for the success of the entire component-based system (CBS) because the quality of the entire system is strongly affected by the quality of each single component. In *testing component-based systems,* instead, the main goal is to test the assembly of heterogeneous components, possibly written in different programming languages, distributed across the network and executed in various platforms.

Although many issues are specific to component testing and testing component-based software (see Ref. 41 chapters 2.3 and 4.2, and Ref. 42) two issues can be considered the most relevant, which are common to both aspects of component-based testing:

– *Availability of information about components:* The basic ingredient of any testing strategy is the availability of information/representation that describes relevant aspects of the component to be tested. Although in traditional software development processes the organization has a full control on the system under development (from requirements down to implementation), in component-based development the source code is in the general case not available. In the specific context of commercial-off-the shelf (COTS) components, components are retrieved from the market and come to the users without the source code. This lack of information poses particularly difficult questions to the final user for what concerns testing.

- *Presence of different stakeholders:* Different stakeholders take place in the development of a component-based system, which belongs to even unrelated organizations. Each of these stakeholders has to define and follow a specific testing process while pursuing, in general, different objectives. In this article we mainly differentiate between two main roles: the *component developer* (who implements software components to be easily and widely reused) and the *component user* (who integrates the component in a final system).

  The *component developer* starts the development of a component having in mind possible reuse scenarios that will make the component worthy to be bought by a possible user. In general, the process adopted will follow a "traditional" approach, and the testing process will not show major novelties.

  The *component user* is the developer of a complex system that decided to manage the complexity by reusing existing components. In case of COTS components, the unavailability of the source code makes all the traditional code-based techniques not applicable. Moreover, the scarcity of information makes also black-box approaches more difficult to apply.

Therefore, developers and users of components are completely different actors from a testing point of view, in which the former has complete view of the component internal details but little information on the final deployment environment, and the latter has little information on the inner parts of the component but a precise understanding of the final deployment environment.

In summary, component-based testing strongly relies on the information available to any possible stakeholder interested in carrying on a testing campaign on a software component or a set of them. At the same time, it is necessary to understand how the available information can be fruitfully used for testing purposes.

Achievements

In the following part of this section, existing component-based testing approaches are classified according to two main criteria: *who defines the test to be executed,* and *which information are sued to define the test cases.* Alternative classifications can be found in Refs. 41,43,44 and 45.

Following the proposed classification, three main classes have been identified:

- Developer-defined test cases;
- Developer-defined component models suitable for analysis and testing derivation by the user;
- User-defined test cases.

In developer-defined test cases, test cases are defined by the developers and provided as they are to the component user, who can just run them (i.e., the user has no control on the test selection phase).

Built in test (BIT) approaches and *testable architectures-based CB testing* are typical examples of approaches in this category. Built in testing is a generic approach to testing where the component is augmented with executable test cases that are built into the component, together with its normal functions. BIT requires component developers to embed tests in software component implementation to support self-testing. By running the embedded test cases, the component user can thus validate in the final environment the hypotheses made by the component developer. Several techniques have implemented this philosophy. Edwards (46) has proposed a framework to provide BIT wrappers for component testing, using the specification language RESOLVE as an example. Wang et al. (47) make use of the BIT for enhancing component-based software maintainability. They build tests in component source code as extra member functions. Other BIT techniques are used in Refs. 48 and 49. More details on BIT approaches can be found in Refs. 41 and 43.

The *testable architectures-based CB testing* approach can be recognized as a special case of BIT in which component developers equip the component with a specific testable architecture that allows the component user to execute the test cases easily. The test information is appended by the developer in the form of specifications, instead of enclosing them in the component itself. One of the most known approaches is the one presented by Momotko and Zalewska (50) which propose a testable architecture based on the Component+ built in testing (C+ BIT) technology for testing component interactions with the environment at run-time.

In *developer-defined component models suitable for analysis and testing derivation by the user,* the component developer provides users with (behavioral) models to use for test case selection. All forms of additional information appended to the software component (either by the developer, the user, or a third-party tester, so as to simplify software testing) can be regarded as forms of *meta-data.*

Wu et al. (51) define a test model called component interaction graph (CIG) for integration testing. As a first step, test elements are identified to build a CIG, and test cases are generated for each test element in the CIG. Another interesting approach, likewise relying on the definition of a particular framework for component testing, has been proposed by Atkinson and Gross (52). The framework is not intended for the execution of generic test cases, but rather focuses on providing the component user with specific test cases derived from contract specifications.

In *user-defined test cases*, the user has full control on the test selection phase. Three main approaches can be assigned to this class: *interface probing* (53), in which the component user derives a set of test cases, executes the component in accordance with them, and analyzes the outputs produced; the *component deployment testing (CDT) approach* where Bertolino and Polini (54) propose an integration testing framework for easing the execution of test cases derived from user architectural specifications; the *Behavior Capture and Test Framework* proposed by Mariani and Pezzé (55) in which the saving and monitoring of test results in one environment can be useful for evaluating component behavior in another environment.

Further Readings

M.J. Harrold et al.'s (42) paper focuses on issues and challenges in analysis and testing of CBS, from both the component developer and component user perspectives. J. Z. Gao, et al. (41) published a complete book on component testing and testing component-based systems and quality assurance. S. Beydeda and V. Gruhn (56) provide a collection of papers that cover the topics of built-in testing, metadata, software product line, testability, and integration testing. M. Jaffar-ur Rehman et al. (43) published a recent survey on issues and solutions in component-based testing.

SPECIFICATION-BASED TESTING

A software specification describes the expected behavior and characteristics of a software product as outlined by Poston (57).

In *specification-based testing* (also called functional testing or black box testing), the internal structure and behavior of the program is not known (i.e., the program is considered as a black box, because we cannot observe what internally happens). What is assumed to have is a *model of the system under test* and the objective is to analyze whether the system behaves correctly with respect to the specifications, that is, to find discrepancies between the actual behavior of the implemented system's functions and the desired behavior, as described in the functional specifications. Test sets are derived from the system specification, and the test selection phase can start even before the code is available (although test execution still needs the existence of the system implementation).

Manifold are the motivations for making use of a specification-based testing approach. (*1*) Code unavailability: If source code is not available (not yet developed or not publicly released), then test cases can be produced out of a specification. (*2*) *Scalability:* Even if source code is available, it becomes rapidly unfeasible to provide complete coverage of complex systems. Being specifications more abstract than source code, specification-based testing is certainly more scalable. (*3*) *Accuracy:* specification-based testing complements code-based testing techniques; although source code testing is typically structural and at a syntactic level (i.e., source code structure coverage), specification-based testing is more functional and at a semantic level (i.e., system functions coverage). (*4*) *Effectiveness:* A specification-based testing plan can start before the code is available, which avoids having to leave the testing activities to the end of the software lifecycle, when schedule slippage, time-to-market pressures, and cost-constraints may result in neglected testing.

The rest of this section will cover the topics of *model-based testing based on formal specifications* (when the model is generated from *a formal specification* or the formal specification itself is used as the test input), and *model-based testing based on diagrammatic specifications* (which refers to UML-based testing and testing based on models designed through *diagrammatic tools).* Next, this article presents existing work on software architecture-based testing (to be considered as a form of specification-based testing).

Model-Based Testing Based on Formal Specifications

Problem Statement. A formal specification can be regarded as "the expression in some formal language and at some level of abstraction, of a collection of properties some system should satisfy" (58). A formal specification language consists of a clearly defined syntax (the notation), fully specified semantics (the specifiable objects), and some satisfied relation (the semantics associated to the syntax). Although a formal specification can be of difficult use (if compared with informal specifications), it has precise, unambiguous semantics, which enables mathematical precision of the analysis of systems and the reasoning about them (59). Moreover, a specification written in a formal language can be processed by automated tools, and formal specification-based testing guarantees higher accuracy, objectivity, and repeatability than ad-hoc test derivation from informal specifications.

Achievements. Testing from formal specifications has received much attention in the last decades. The first approaches for specification-based testing were proposed in the 1980s. Since then, many specification-based testing approaches have been proposed, based on formal languages such as Z, VDM, CSP, CCS, LOTOS, SDL, and Petri Nets (60). More recently, conformance testing approaches based on Labeled Transition Systems or Finite State Automata (or their variations) have been proposed (61–63). Here, we restrict our attention to test selection and execution from such formal specifications, for which a mature theory of conformance testing now exists, together with automated tool support.

The aim of a formal conformance testing framework is to define a *conformance relation* between the implementation under test *IUT* and the (formal) specification *S*. Such a relation precisely establishes when *IUT* is a correct implementation of *S* and is based on the *test hypothesis,* that is, the *IUT* can be modeled by a formal object *MOD,* such that all the observations that we can make on the *IUT* and on *MOD* (along the executions of all defined test cases) cannot be distinguished (64). In such a way, we can formally define an "implementation relation" *(imp)* that correlates *S* with *MOD*. *IUT* conforms to *S* iff *MOD* is *imp*-correct with respect to *S*.

Figure 5. graphically summarizes how a conformance testing framework works.

In Tretmans' approach (62,65), both the specification *S* and the model *MOD* of the implementation *I* are expressed

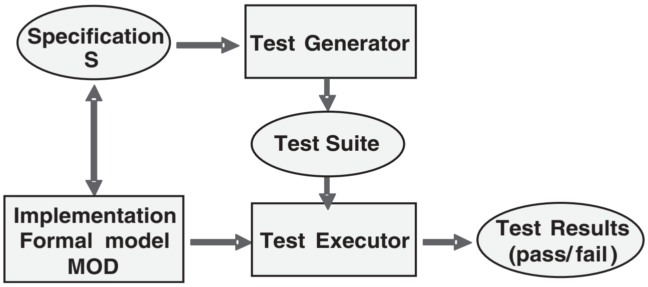


Figure 5. The conformance testing framework.

using input/output transition systems (IOTSs), which is an extension of the classic LTS model, in which the set of actions are partitioned into the input actions and the output actions. The implementation relation *imp* used is of the form **ioco**, that is a relation holding when *MOD* can never produce an output that could not have been produced by *S* after the same sequence of actions, or *trace*(66).

Similar derivations are obtained by von Bochmann et al. (63) who use finite state machines to derive the system specification, its implementation and "conformance relations" (equivalence, quasi-equivalence, reduction) to generate test sequences. Fernandez et al. (67), use IOLTS in their approach for an automatic, on-the-fly generation of test cases.

Tools. TorX, TGV, and TVEDA are well-known tools for automating the test selection and execution process in formal testing.

TorX (68) allows an on-the-fly test generation and execution (i.e., test derivation and test execution occur simultaneously). The input language is LOTOS or Promela, and the test selection can be random or manual. In TGV (61) the tester can use his/her knowledge of the implementation under test and of the context to guide the test selection through the notion of a *test purpose* (61,66) (which permits to focus on testing relevant interactions, while hiding the others). The TGV input languages are LOTOS or SDL, and test cases are derived in TTCN (69). TVEDA (70) derives TTCN tests from SDL specifications.

Differently from TGV and TVEDA, TorX allows test derivation and execution in an integrated manner.

Further Readings. *C. Jard and T. Jeron* (71) provide one of the most recent papers on TGV; it describes the tool and the theory behind it. Reference (72) points to the TGV website. *J. Tretmans* (59) covers the theory behind conformance testing in an easy-to-understand and complete way.

Model-Based Testing Based on Diagrammatic Specifications

Problem Statement. A model-based testing approach accepts two main inputs, a model of the software under test and a set of test generation directives (which guide the test cases selection), and outputs a test specification (which includes a set of stimuli the tester should introduce in the system together with expected responses) (73). This section will focus on model-based specifications in the Unified Modeling Language (UML) (74).

Achievements. Binder surveys how each and every UML diagram could be tested (Reference 25, ch. 8). Although UML structural diagrams are usually employed for testing design consistency (gray-box testing), behavioral diagrams are used for functional testing, precisely, interaction diagrams for integration testing (interactions among objects), and state diagrams for functional testing of objects.

This section will focus especially on a representative subset of (mostly academic) scenario-based and state-based testing approaches. Figure 6 [taken from Navlasky (75)]

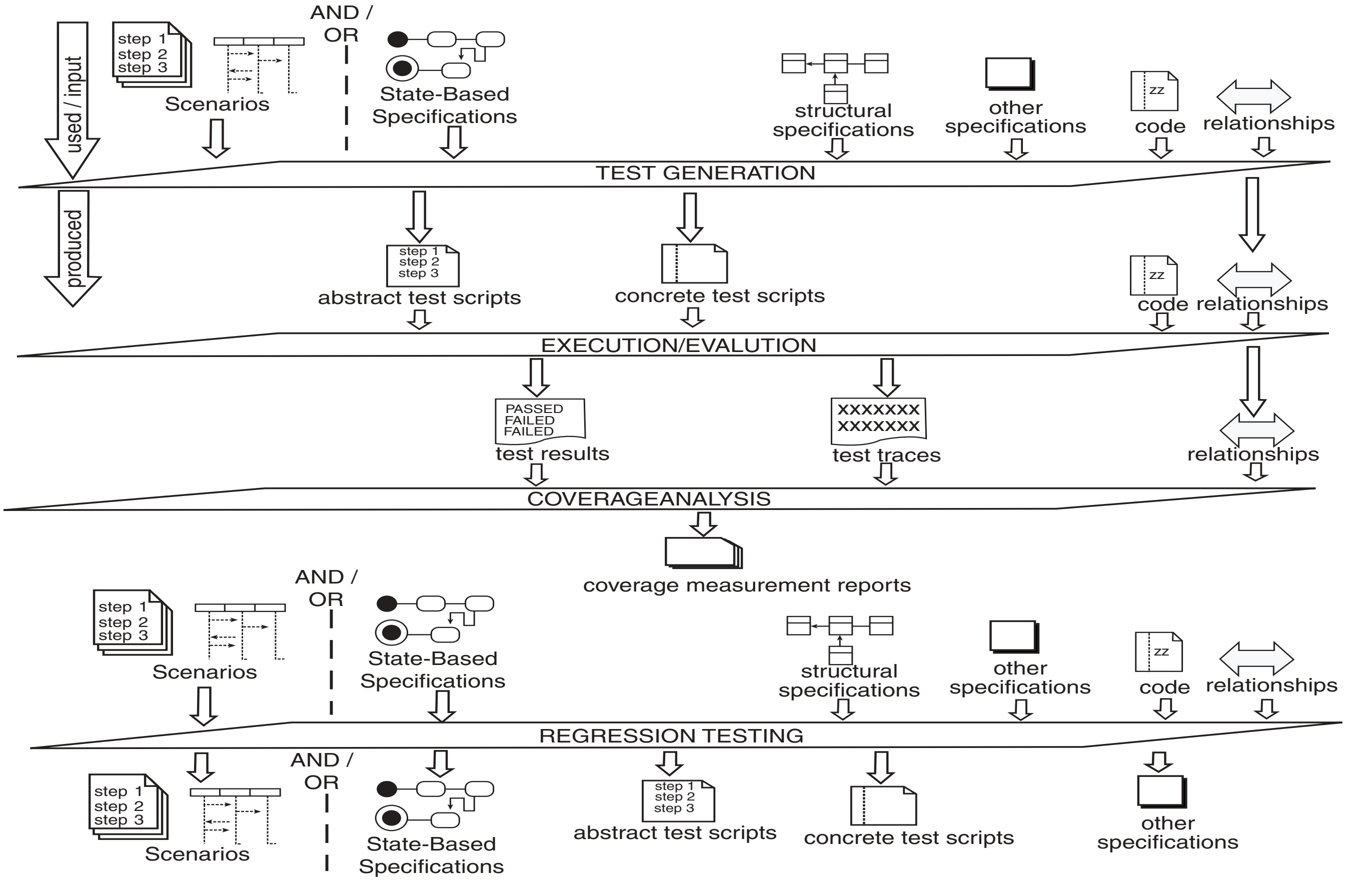


Figure 6. Steps in model-based testing.

summarizes the test generation, test execution and evaluation, coverage analysis, and regression testing activities implemented by automated model-based testing approaches.

The *test selection* activity receives models as input and produces test scripts as output. The inputs are textual or graphical scenarios, state-based, or other specifications (structural specifications, such as class or component diagrams and activity diagrams). Sometimes, the code itself becomes an input of the test generation activity. The outputs are *test scripts* (sometimes referred to as test sequences), and comprise the following: a set of steps to be followed when testing a program, as well as input and output values. Test scripts are either abstract or concrete. Abstract test scripts describe the steps a tester should follow when using the system, the inputs to provide, and the outputs to expect. Concrete test scripts can be compiled and automatically executed. They consist of calls to methods in the code, the inputs to provide, and the outputs to expect. The evaluation is either made manually by the tester or automatically by an oracle.

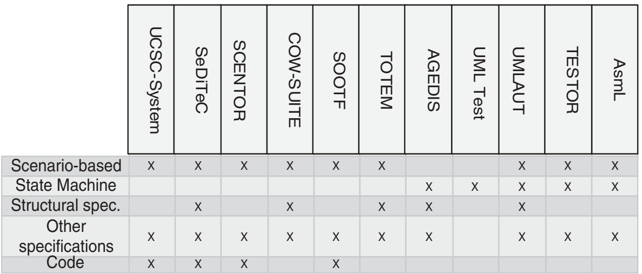
The *test execution and result evaluation* activity receives as input abstract or concrete test scripts outputted by the test generation activity. Test execution runs test scripts over the source code. The observed behaviors are evaluated against expected results, as expressed by the specification models.

Coverage analysis can be automatically performed to evaluate how much of the code has been covered. *Regression testing* can be applied to re-test changed models.

In the study conducted in Ref. 75 and summarized in Ref. 76, 11 automated model-based testing approaches were evaluated. Six of them are based on scenarios, two on state-based models, two on both scenarios and state machines, and one on scenarios or state machines. Figure 7 summarizes the input artifacts used by the 11 surveyed model-based testing techniques.

By taking a look at other surveys on the topic, Utting et al. (77) place model-based testing approaches into a seven-dimension orthogonal taxonomy. The dimensions characterize the approaches with regard to the nature of the model used (e.g., what is modeled, notation used), to the nature of the test generation techniques used (e.g., test selection coverage) and to the nature of the test execution (e.g. on-line, or off-line).

In their work, Prasanna et al. (78) survey test case generation approaches. They classify these approaches into two categories: specification-based approaches and model-based approaches.



	UCSC-System	SeDiTeC	SCENTOR	COW-SUITE	SOOTF	TOTEM	AGEDIS	UML Test	UMLAUT	TESTOR	AsmL
Scenario-based	x	x	x	x	x	x			x	x	x
State Machine							x	x	x	x	x
Structural spec.		x		x		x	x		x		
Other specifications	x	x	x	x	x	x	x		x	x	x
Code	x	x	x		x						

Figure 7. The main artifacts used in model-based testing.

Tools. Hartman (73) presents a survey on model-based test generation tools. He distinguishes between test generators and model-based input generators. He also distinguishes between test generators and test automation framework, where the automation framework executes the test sequences without human supervision. The objective of Hartman's survey is to place the AGEDIS (79) project and tool in relation to other tools. He groups the tools into academic and commercial and succinctly describes each of them. It is therefore a good reference to a list of tool supported approaches.

In their book chapter (80), Belinfante et al. provide an in-depth evaluation of test case generation tools. The book, however, is on model-based testing for reactive systems, where models are described in their majority with formal specification languages.

Further Readings. R. V. Binder (25) mostly describes model-based test design techniques (based on combinational models, state machines and the UML) for OO systems and test design patterns. M. Broy et al. (81) provide a collection of papers presented at the 2004 Dagstuhl seminar on "Model-Based Testing of Reactive Systems." It covers testing of finite state machines, testing of labeled transition systems, model-based test case generation, tools and case studies, standardized test notation, and execution architectures. L. Mariani and M. Pezzé (26) discuss both specification-based and structural-based approaches for intra-class and inter-class testing of OO software. The Model Based Testing Forum (82) is a useful forum on model-based testing.

Software Architecture-based Testing

A software architecture (SA) is the first design artifact that can be produced during the software development. It breaks down the system into several cooperating elements (components, connectors, data elements). Although components represent the core artifacts, a software architecture consists in the study of how such components can be integrated in order to satisfy desired functional and non functional requirements.

According to Stafford and Bosch (38), software architecture represents an integral part of any component-based software system; software architecture and components are two sides of the same coin. A component-based design can be (*1*) component-driven, if existing components are integrated via a component framework, according to a certain architecture; or (*2*) architecture-driven, if components are produced or acquired in order to satisfy architectural requirements.

Problem Statement. A growing interest in software architectures characterizes the last decade. Although most initial effort has been on how to specify an architecture, how to select the right architecture has become one of the most relevant challenges in recent days. In particular, the importance of the role of SA in testing and analysis is becoming evident (83).

When referring to software architecture and testing, we need to distinguish between SA-based testing and testing

SAs. In *SA-based testing*, the goal is to use the SA specification as a reference model to extract suitable test classes for the higher levels of testing and to refine them into concrete tests at the code level (84). In *testing SA,* instead, the main goal consists in assessing the SA quality with respect to expected requirements it shall meet.

Two major issues are to be solved in architecture-based testing: *traceability* and *test execution*.

Traceability consists of "relating the abstract values of the specification to the concrete values of the implementation" (85). Because test cases are selected from the (intentionally abstract) architectural model, they have to be refined into more concrete test sequences to execute these tests on the code. Similar problems can be found in specification-based testing (previously discussed), but they are typically solved by making use of detailed and lower-level specifications.

Test execution entails forcing the implementation under test (IUT) to execute the specific sequence of events that has been selected. However, a problem develops with concurrent programs that, starting from the same input, may exercise different sequences of interactions (among several concurrent processes) and produce different results. This problem has already been analyzed in the literature, and nondeterministic and deterministic-testing (86) approaches have been proposed.

Achievements. Several authors have advocated the use of architectural models to drive testing, and in particular to select relevant behaviors of interactions between system components based on the early SA specification. In Ref. 87, the authors analyze the advantages in using SA-level testing for reuse of tests and to test extra-functional properties. In Ref. 88, the authors define six architecture-based testing criteria, which adapt specification-based approaches. Those two papers represent the fist attempts in software testing based on an architectural specification.

In Ref. 89, the authors present an architecture-based integration testing approach that takes into consideration architecture testability, simulation, and slicing. In Ref. 90, Harrold presents an approach for effective software architecture regression testing, and in Ref. 6 she also discusses the use of software architecture for testing. In Ref. 91, Rosenblum adapts his strategy for component-based systems testing to SAs. The testing approach is based on architectural models, which could be simulated or executed or used to guide integration or regression testing of the implemented system. The author also shows how formal models of test adequacy can be used in conjunction with architectural models to guide testing.

In Ref. 92 the authors propose a technique to test data flow and control flow properties at the architectural level. Six architecture relations among architecture units are defined and then used to define architecture testing paths. Five architecture-level testing criteria are proposed. However, to the best of our knowledge, the approach in Ref. 84 is still the only comprehensive attempt to tackle the whole cycle of SA-based testing.

		Bertolino Inverardi '96 Ref. [87]	Richardson Wolf '96 Ref. [88]	Richardson Stafford Wolf '97 Ref. [89]	Harrold '98 Ref. [90]	Rosenblum '98 Ref. [91]	Jin Offutt '01 Ref. [92]	Muccini Bertolino Inverardi '04 Ref. [84]	Muccini, Dias, Richardson '06 Ref. [93]
Introduction to the topic		++	++	++	+	+	+		
Testin g Activities	Test case Selection		+	+	++		+	++	++
	Coverage Criteria		++	++	+		++	+	
	Adequacy criteria					++		++	
	Test Execution							++	++
	Results Evaluation			+					+
Testing Phases	Unit	++							
	Integration	++						++	+
	System								
Testing Goal	SA-based Testing		+	+	+		+	++	+
	SA assessment		+	+					
Other	Testability		+	++	++				
	SA styles			++					
	Regression Testing	+			++	+			++
	Traceability								

Figure 8. Software architecture-based testing techniques.

Reference 93 presents a framework for SA-based regression testing, coping with two main types of evolution: architectural evolution and code evolution.

Figure 8 shows a synthetic table of SA-based testing techniques, pointing out the specific goals of each given paper/approach.

Further Readings. H. Muccini, et al. (84) present a comprehensive attempt to tackle the whole cycle of SA-based testing. H. Muccini et al. (93) propose an approach for SA-based regression testing.

ONGOING RESEARCH IN SOFTWARE TESTING

Although previous sections have discussed consolidated testing techniques based on different development paradigms and based on different precode artifacts, this section introduces some ongoing research that, while not consolidated, is being used in academic and practical domains and are being considered key technologies. We here focus on testing service-oriented applications, on the UML testing profile, and on product line testing.

Testing Service-Based Applications

Service oriented computing aims to provide the basis for building software by assembling independent, loosely coupled services. Services are understood as autonomous, platform-independent computational entities that can be described, published, categorized, discovered, and dynamically assembled for developing massively distributed, interoperable, evolvable systems and applications. These characteristics pushed service-oriented computing toward its present widespread success, which is demonstrated by the industrial investments on this technology.

Problem Statement. Web services represent a concrete instantiation of the service oriented paradigm, where web services are distributed and integrated via Web standards. Because web services are considered a key technology in software production today, how to test them is becoming a relevant task in many industrial organizations.

The main issues and challenges related to testing web services resemble those related to component-based testing (as already analyzed previously and remarked in Ref. 95), with some extensions as in the following:

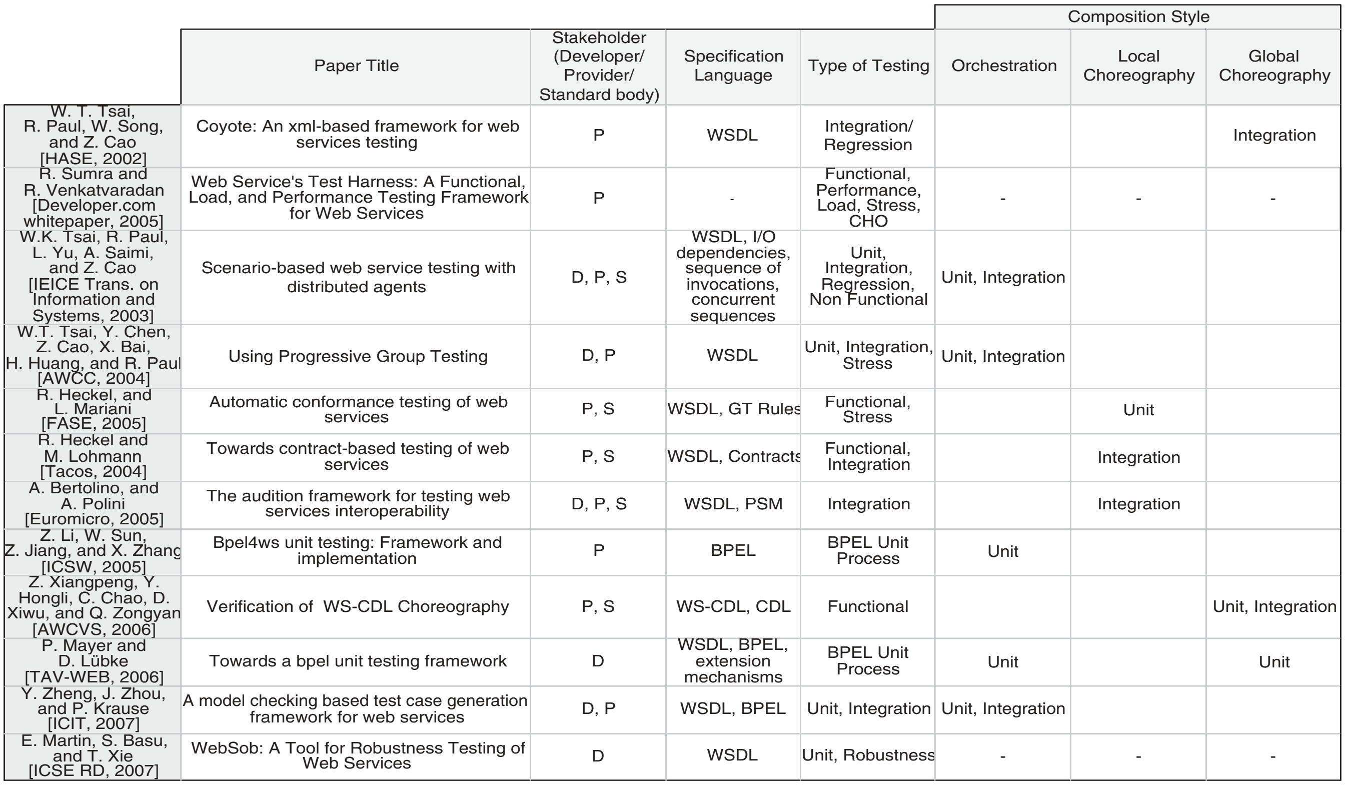
- *Lack of control:* As in component-based development, services can be produced by different stakeholders. However, although component-based applications can be produced by a single organization by integrating reusable components, service oriented systems are never under the full control of a single organization.
- *Different Stakeholders:* Although two main stakeholders are involved in component-based testing, five main actors might be concerned with testing of services: service developers, service providers, service integrators, third-party certification authorities, and final users. Each can thus require specific testing techniques, which are related to the level of knowledge/control they do have on the service and on the testing purpose.
- *Different stages:* A web service can be tested at three different stages: before deployment, at registration time, and at execution time.

  A service can be tested *before its deployment* by using off-line testing. At this time, the service developer or provider tests the service by simulating possible usage scenarios. Successively, at *registration time*, the test can be executed in a real execution environment to provide more realistic results. The discovery and directory services can then be reinforced with testing functionalities. Eventually, at *execution time*, the system can be monitored so as to discover run-time service failures.
- *Different ways of integrating services:* Services can be composed by following two complementary views: orchestration and choreography. In the orchestration, an orchestrator defines the interaction order among components. A choreography, instead, specifies the interaction among all the services that take place in the system. Different interaction primitives and languages are used by the two views, which impose clear distinctions among choreography-based and orchestration-based testing (96).

Achievements. Many approaches have been proposed so far for testing web services.

The survey study in Ref. 97 takes into consideration 12 web services testing frameworks and classify them according to many parameters, the most relevant being the following: the testing web service architecture, the stakeholders involved in the testing activity, the web services specification languages, and the testing strategies. The table in Fig. 9 summarizes the survey main results.

In Reference 95 , a comparison among three web services testing tools (Parasoft SOATest, Mindreef SOAPScope, and PushToTest TestMaker) is reported according to the following (main) parameters: support for functional testing; support for custom scripting, type of licence, and type of interface; support for simulation and recording; and support for extra-functional testing.



					Composition Style		
	Paper Title	Stakeholder (Developer/ Provider/ Standard body)	Specification Language	Type of Testing	Orchestration	Local Choreography	Global Choreography
W. T. Tsai, R. Paul, W. Song, and Z. Cao [HASE, 2002]	Coyote: An xml-based framework for web services testing	P	WSDL	Integration/ Regression			Integration
R. Sumra and R. Venkatvaradan [Developer.com whitepaper, 2005]	Web Service's Test Harness: A Functional, Load, and Performance Testing Framework for Web Services	P	-	Functional, Performance, Load, Stress, CHO	-	-	-
W.K. Tsai, R. Paul, L. Yu, A. Saimi, and Z. Cao [IEICE Trans. on Information and Systems, 2003]	Scenario-based web service testing with distributed agents	D, P, S	WSDL, I/O dependencies, sequence of invocations, concurrent sequences	Unit, Integration, Regression, Non Functional	Unit, Integration		
W.T. Tsai, Y. Chen, Z. Cao, X. Bai, H. Huang, and R. Paul [AWCC, 2004]	Using Progressive Group Testing	D, P	WSDL	Unit, Integration, Stress	Unit, Integration		
R. Heckel, and L. Mariani [FASE, 2005]	Automatic conformance testing of web services	P, S	WSDL, GT Rules	Functional, Stress		Unit	
R. Heckel and M. Lohmann [Tacos, 2004]	Towards contract-based testing of web services	P, S	WSDL, Contracts	Functional, Integration		Integration	
A. Bertolino, and A. Polini [Euromicro, 2005]	The audition framework for testing web services interoperability	D, P, S	WSDL, PSM	Integration		Integration	
Z. Li, W. Sun, Z. Jiang, and X. Zhang [ICSW, 2005]	Bpel4ws unit testing: Framework and implementation	P	BPEL	BPEL Unit Process	Unit		
Z. Xiangpeng, Y. Hongli, C. Chao, D. Xiwu, and Q. Zongyan [AWCVS, 2006]	Verification of WS-CDL Choreography	P, S	WS-CDL, CDL	Functional			Unit, Integration
P. Mayer and D. Lübke [TAV-WEB, 2006]	Towards a bpel unit testing framework	D	WSDL, BPEL, extension mechanisms	BPEL Unit Process	Unit		Unit
Y. Zheng, J. Zhou, and P. Krause [ICIT, 2007]	A model checking based test case generation framework for web services	D, P	WSDL, BPEL	Unit, Integration	Unit, Integration		
E. Martin, S. Basu, and T. Xie [ICSE RD, 2007]	WebSob: A Tool for Robustness Testing of Web Services	D	WSDL	Unit, Robustness	-	-	-

Figure 9. Comparing testing web services frameworks.

Further Readings. L. Baresi and E. Di Nitto (98) book contains contributions from leading academic and industrial research groups on static analysis, testing, monitoring, and nonfunctional analysis of web services.

MDA-Based Testing: The UML Testing Profile

Problem Statement. With the widespread adoption of software models and model-based testing techniques, new challenges arise. A multitude of different model-based testing approaches are available today (as outlined previously), each one focusing on different UML models, different level of abstractions, and different test selection/execution/coverage criteria. As soon as a model evolves, related models and the test themselves shall evolve to keep the specification and test specification consistent. Explicit support for models' alignment is rarely available. Moreover, it is not uncommon during evolution that only the low-level design and implementation are changed to meet tight deadlines, while the model specification is not updated to track the changes being made to the implementation (99). As soon as the specification "drifts" out of conformance with the implementation, it is no longer representative of the expected behavior.

Explicit relationships need to be devised between specifications and their implementations (sometimes called mapping), and between specifications and test results (sometimes called traces) (as initially discussed in Ref. 85).

Achievements. To handle such problems and to bridge the gap between models and implementation, in 2001 the object management group (OMG) has released the UML testing profile request for proposal. As a result, the UML testing profile (UTP) (100) has become an official OMG standard since late 2005, with the main goal of allowing a conformance testing activity to take place in parallel with the system development. The main objectives of the UTP are to provide an ontology of testing-related artifacts needed for testing purposes and to model transformation algorithms to move toward different models. The UTP implements the MDA philosophy by providing a common framework for system development and testing. The UTP profile extends the UML 2.0 meta-model to introduce test-specific concepts. The analogy to MDA has been illustrated in Fig. 10. The PIT is the platform independent test, which is generated through model-to-model transformation techniques from the platform independent model (PIM). Following the MDA philosophy, as the PIM is transformed into a platform specific model (PSM), the PIT is transformed into a platform specific test (PST), i.e., a test executable on the PSM. Although vertical transformations permit to move from the PIT, to the PST and test code, horizontal transformations permit to move back and forth from development to testing.

The main goals and principles of UTP are:

- **To interweave software development and testing**. The main principle implemented in the UTP is that software testing needs to be carried out during the entire software development, and it should be connected as much as possible with the system development models. The UTP thus provides a unique notation based on the UML to be used by both software developers and software testers, and it provides the ability to move from the modeling domain to the testing one.
- **Conformance Testing**. The UTP has been proposed to provide an answer to the need of solid conformance testing, certification, and branding.
- **Structure + behavior Modelling**. The UTP takes into consideration both structural and behavioral diagrams. Structural diagrams are used to specify the testing components. Behavioral diagrams are used for specifying test cases and for identifying the test cases execution order.
- **Black-box testing**. UTP focuses on black-box conformance testing. For this reason, the profile itself does not consider any information on the internal structure of the system under test.

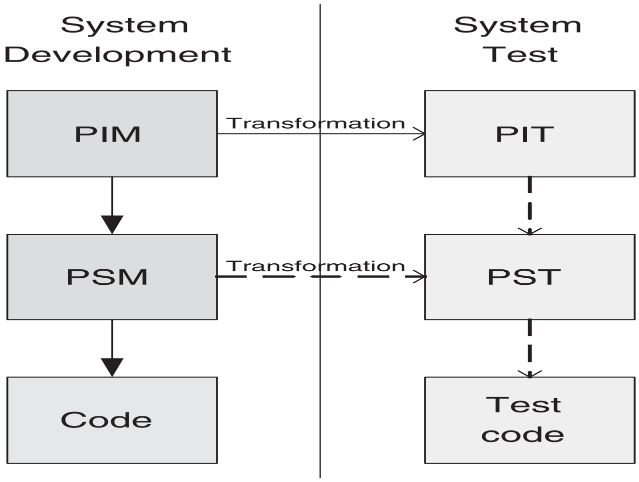


Figure 10. The UTP contextualization in MDA.

The UTP contributes to the aforementioned goals in two different ways; it provides an *ontology of model-based testing concepts* to produce a testing model, and it describes some *transformation rules* for moving from development models (PIM and PSM) to testing models (PIT and PST). The ontology of concepts has been categorized into four conceptual packages: test architecture, test behavior, test data, and time. Each package has a specific goal and contains meta-classes that identify the UTP ontology. Each package is described in detail in Ref. 100.

Further Readings. The UML Testing Profile (100) provides the OMG official specification for the UML Testing Profile, v 1.0. von P. Baker and colleagues (101) authored the first book on the UML Testing Profile. Z. Ru Dai et al. (102,104) present three papers on the UTP and its relation with TTCN-3.

Testing Software Product Lines

Software product lines (SPLs) represent a more strategic approach to reuse, where components and other assets are identified in relation with specific application domains, architectures, and scopes (opposed to software reuse schemes which try to create assets as general as possible). Quoting Clements and Northrop (105), “a software product line is a set of software-intensive systems sharing a common managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.” The idea behind a system–family approach is to build a new system or application from a common set of assets (e.g., component, known requirements or design elements, models, artifacts) in the same domain.

The main concepts that characterize an SPL are those of:

- *Domain-specific core assets:* Instead of creating libraries of general purpose components, hoping they will be used in future development, SPLs make a predictive analysis of such artifacts that may be reused in the development of a specific domain-related product (106).
- *Commonality and variability:* Commonality defines what different products have in common and guides the production of domain-specific core assets. Variability is the ability to change or customize a software system (107) (i.e., to distinguish one product from another in the SPL), and it requires to delay design decisions such that many decisions are left open until a certain product is selected out of the product line. A variation point identifies one or more locations at which the variation will occur.
- *Domain engineering and application engineering:* Taking a look at an SPL software process, two main phases are always contemplated: domain engineering and application engineering. In the former phase, commonality and variability of the entire family are analyzed, whereas during application engineering an individual application, member of the SPL, is selected.

When testing an SPL, traditional testing techniques have to be adapted so as to take into consideration the SPL peculiarities described above. The main challenge is in investigating *how the entire SPL can be used to generate testing information automatically that may be effectively reused to test each derivable product*, as discovered in Ref. 108.

Further Readings. J.D. McGregor (109) provides principles and techniques in testing software product lines. A. Tevanlinna et al. (110) describe the state-of-the art of product family testing techniques.

FUTURE DIRECTIONS IN SOFTWARE TESTING

Harrold (6) and Bertolino (7) in their ICSE 2000 and ICSE 2007 (respectively) Future of Software Engineering (FOSE) papers on software testing, highlight several working directions and challenges in software testing. While sharing most of them, I would like to briefly discuss here those areas I personally believe will be of major relevance in next generation software testing techniques.

Testing Evolving at Run-Time Software

An increasingly important requirement for software-intensive systems is the ability of changing over time, because of the need to add/remove new features dynamically, the need to provide a more dependable system, or the need to protect the system from run-time incoming attacks or deficiencies. This newly acquired level of dynamicity, while enhancing the flexibility of a software system that can become self-adaptable, imposes new degrees of complexity when certain qualities of the system must be guaranteed. Continuously evolving systems become difficult to be analyzed with techniques used over traditionally static systems. In the context of such highly dynamic systems, the focus moves from validating the static configuration to validating the changing over-time design. Thus, whereas in traditional static systems the verification can be done once before system deployment, for dynamic architectures the validation becomes a perpetual activity to be performed during system execution. On-line testing and monitoring techniques are currently being analyzed for testing such types of software (see for example Refs. 95 and 111).

Automation in Software Testing

Practitioners regard software testing as the central means for ensuring that a system behaves as expected. The introduction of automated testing techniques has strongly facilitated the introduction of software testing into practice; automation has reduced the amount of effort spent on technical testing activities and has also increased the precision of activities, like result evaluation, often performed by humans and thus more error prone. However, testing in industrial projects can be effective only when the testing effort is “affordable”: the testing approaches should support automatic creation of test plans sooner, they should automate most testing activities, and testing and development shall proceed hand-by-hand being automated in the same development and testing framework.

As a side effect, software testing automation would also improve technology transfer of testing and analysis research. Technology transfer is fundamental for making research useful in industrial domains and for bridging the gap among academic research and industrial needs, as remarked by Boshernitsan (112) in his ISSTA 2006 invited speak. Hartman (113) describes his experience as well as the challenges and dreams in making model-based testing an industrial practice. Test automation is remarked as one of the main means to make model-based testing an industrial practice.

Integration of Testing and Analysis Techniques. Integration of analysis techniques is a topic that has recently been receiving some attention in the software engineering community (e.g., Ref. 114). This problem can be addressed from two different viewpoints: integrating analysis techniques during the development process (from requirement-based to source code analysis) and integrating different analysis techniques at the same stage in software development (e.g., design-based testing and model-checking).

Antimodel-Based Testing

Specification-based testing is certainly useful and effective; however, there can be several reasons why such an approach *cannot be applied* or is *too expensive* for deployment in a specific context. One of the main obstacles relies on the assumption that a complete and consistent model of the software system exists, whereas for certain classes of systems (e.g., component-based systems), we cannot assume a priori that a specification or the source code are available. In such cases, model-based testing is not applicable, or would be too costly.

This is the rationale for an *"antimodel-based testing approach":* whereas model-based testing starts from an a-priori established model and tries to execute some sequences derived from this model, in "antimodel based" testing the opposite direction is taken: the implementation is executed on some sample executions, and by observing the traces of execution an abstract model of the system is inferred/synthesized a posteriori.

Self-Testing

An increasingly important requirement for software systems is the ability to self-manage, (i.e., to autonomically manage themselves). According to Kephart and Chess (115), the four main autonomic areas are self-protection, self-configuration, self-optimization, and self-healing. Other colleagues describe other forms of self* -techniques, such as self-adaptivity, and self-organization. In the context of self*- systems, self-testing seems to be a natural way to move forward, in which the system has the ability to test (and repair) itself autonomically.

CONCLUDING REMARKS

This article has tried to provide an ample view on software testing from two different viewpoints: the development technology to be tested and the artifacts to be used for testing purposes. A detailed list of references on high-quality research work has been provided to help the reader acquire a more advanced knowledge on each area covered.

ACKNOWLEDGMENTS

I wish to thank the anonymous reviewers, Mauro Pezzé and Antonia Bertolino for their useful feedback on a preliminary version of this paper. I thank you Andrea Polini for his contribution and help within the component-based testing section of this article, and Damien Andrew Tamburry for is thorough proofreading.

APPENDIX A: A GLOSSARY OF SOFTWARE TESTING TERMS

This section provides an explanation of the main terms used in software testing. In order to make the explanation more practical, some of the described concepts will be applied to the *BuyOnline* fictitious web application, which allows users to register, to select products from a list, to visualize their descriptions and information, and to buy them online.

Test Case: A test case is a set of inputs, execution conditions, and a pass / fail criterion [116]. A test case thus includes not only input data but also any relevant execution conditions and procedures, and includes a way of determining whether the program has passed or failed the test on a particular execution [8].

A single test case *tc1* for the BuyOnline application might be: select product with id 563AD, put it in the chart, buy it by credit card.

Test Suite: A test suite is a collection of test cases.

Further test cases can be generated to test the BuyOnline application (by changing the selected products, by removing a product from the chart, or by changing the payment type). All together they will form a test suite for the BuyOnline system.

Test Process: The testing process consists of different activities, the most important being:

- *Test Selection: it consists in selecting a suitable and finite set of test cases from the possibly infinite set;*
- *Test Execution and Evaluation: it consists in executing the code accordingly to the selected test cases and comparing real and expected results.*

The *test selection* activity provides guidelines on how to select test cases. It is driven by a "test criterion" and has to produce "suitable" test cases:

Test Criterion: A test criterion provides the guidelines, rules, and strategy by which test cases are selected. In general, a test criterion is a means of deciding which shall be a "good" set of test cases [117] .

Suitability: A test case is suitable if it contributes to discovering as many failures as possible, according to a test criterion.

Test cases for the BuyOnline application can be selected randomly (i.e., by randomly selecting products, payments types, and functions to be run) or through a more systematic approach, like code coverage (i.e., test cases are selected so to cover all the functions, or the code statements/ branches/paths.)

The *test execution* activity consists in executing the code according to test case inputs. Typically, it describes how to bring the software system in a state so that the test input can be given.

When executing the BuyOnline application according to test case *tc1,* the user has to be registered first, then the test case inputs can be provided.

The *test oracle* evaluates whether the observed outputs comply to expected behaviors. The test oracle can be the tester herself: based on her knowledge of the expected system output, she can predict the expected behavior and thus compare the oracle with the real behavior. Otherwise, the oracle can be automatically generated e.g., from an existing specification or from a previous version of the system.

The test oracle for *tc1* may state that if product 563AD is the one selected, it is the one delivered and the credit card is charged for its correct amount.

The *test adequacy* criteria permit to judge whether the test campaign is sufficient. In general, the test adequacy should state that a test suite is adequate when it allows the identification of any failures. However, since the number of failures is not known a priori, we need to approximate the intuitive concept of adequacy. Thus, a test criterion can be used to define a stopping rule for evaluating the adequacy of the selected test suite.

The testing campaign of the BuyOnline is adequate as soon as the selected test suite covers, for example, the amount of source code defined in the test criterion.

Further Readings

A. Bertolino et al. (3) in their chapter, part of the Guide to the Software Engineering Body of Knowledge, provide a compendium and guide to the body of knowledge on software testing as developed in the past four decades.

BIBLIOGRAPHY

1. E. W. Dijkstra, *Chapter I: Notes on Structured Programming*, 1972, pp. 1–82.
2. S. Rapps and E. J. Weyuker, Selecting software test data using data flow information, *IEEE Trans. Soft. Engin.*, **4**: 367–375, 1985.
3. A. Bertolino and E. Marchetti, Software testing, in: *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Piscataway, NJ: IEEE 2004.
4. J. R. Horgan and A. P. Mathur, Software testing and reliability, in handbook of software reliability and system reliability. (1996) 531–566.
5. B. Littlewood, P. T. Popov, L. Strigini, and N. Shryane, Modeling the effects of combining diverse software fault detection techniques. *IEEE Trans. Softw. Eng.* **26**: 1157–1167, 2000.
6. M. J. Harrold, Testing: a roadmap, in A. Finkelstein, (ed.), ACM ICSE 2000, *The Future of Software Engineering*, 2000, pp. 61–72.
7. A. Bertolino, Software testing research: achievements, challenges, dreams, in L. Briand, A. Wolf, (eds.), ACM ICSE 2007, *Future of Software Engineering*, 2007, pp. 85–103.
8. M. Pezzé and M. Young, *Software Testing and Analysis: Process, Principles and Techniques*. New York: Wiley, 2007.
9. ISSTA: International Symposium on Software Testing and Analysis. Available: http://issta08.rutgers.edu/.
10. ICST: International Conference on Software Testing, Verification and Validation. Available: http: //www.cs. colostate. edu/icst2008/.
11. TESTCOM: International Conference on Testing of Communicating Systems. Available: http://www-higashi.ist.osaka-u.ac.jp/TESTCOM-FATES08/history.html.
12. K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson, A Practical Tutorial on Modified Condition/Decision Coverage. Technical Report NASA/TM-2001-210876, National Aeronautics and Space Administration (NASA), 2001.
13. J. W. Laski, B. Korel, A data flow oriented program testing strategy. *IEEE Trans. Softw. Eng.***9**: 347–354, 1983.
14. P. G. Frankl and E. J. Weyuker, An applicable family of data flow testing criteria. *IEEE Trans. Softw. Eng.***14**: 1483–1498, 1988.
15. B. Beizer, *Software Testing Techniques*, 2nd ed. Boston, MA: *International Thomson Computer Press,* 1990.
16. S. Cornett, Code Coverage Analysis. Available: http://www.bullseye.com/coverage.html.
17. Wikipedia: Code Coverage. Available: http://en.wikipedia.org/wiki/Code_coverage.
18. A. H. Watson and T. J. McCabe: Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. NIST Special Publication 500-235, 1996.
19. W. G. Bently and E. F. Miller: Ct coverage initial results. *Softw. Qual. J.* **2**: 29–47, 1993.
20. R. N. Taylor, D. L. Levine, and C. D. Kelly, Structural testing of concurrent programs. *IEEE Trans. Softw. Eng.***18**: 206–215, 1992.
21. B. Marick: Experience with the Cost of Different Coverage Goals for Testing. Available: http://www.testing.com.
22. Tools: Open source testing tools. Available: http://www.opensourcetesting.org/.
23. Tools: Software QA Testing and Test Tool Resources. Available: http://www.aptest.com/resources.html.
24. J. D. McGregor and D. A. Sykes: *Practical Guide to Testing Object-Oriented Software*, 1st ed. Reading, MA: Addison-Wesley Professional, 2001.
25. R. V. Binder: *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Reading, MA: The Addison-Wesley Object Technology Series, 1999.
26. L. Mariani and M. Pezzé, Testing object oriented software, in A. D. Lucia, F. Ferrucci, G. Tortora, and M. Tucci, (eds.), *Emerging Methods, Technologies and Process Management in Software Engineering*. New York: Wiley-Interscience, 2008.
27. M. J. Harrold and G. Rothermel, Performing data flow testing on classes, in SIGSOFT '94: *Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of Software Engineering*. New York: ACM Press, 1994 pp. 154–163.
28. A. L. Souter and L. L. Pollock: The construction of contextual def-use associations for object-oriented systems, *IEEE Trans. Softw. Eng.* **29**: 1005–1018, 2003.
29. U. Buy, A. Orso, and M. Pezzé, Automated testing of classes, in *ISSTA '00: Proceedings of the 2000 ACM SIGSOFT international Symposium on Software Testing and Analysis*. New York: ACM Press, 2000, pp. 39–48.
30. M. J. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. A. Spoon, and A. Gujarathi, Regression test selection for java software, Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2001), Tampa Bay, FL, 2001, pp. 312–326.
31. A. Orso, N. Shi, and M. J. Harrold, Scaling regression testing to large software systems, Proceedings of the 12th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 2004), Newport Beach, CA: 2004, pp. 241–252.
32. Junit: The JUnit unit testing framework. Available: www.junit.org/,http://sourceforge.net/projects/junit/.
33. SourceForge: Sourceforge website. Available: http://sourceforge.net/.
34. Eclipse: Eclipse plugins. Available: http://eclipse-plugins.2y.net/eclipse/index.jsp.
35. Tigris: Tigris.org: Open Source Software Engineering Tools. Available: http://www.tigris.org/.
36. Wikipedia: Wikipedia: List of unit testing frameworks. Available:http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks.

37. D. C. Kung, P. Hsia, and J. Gao: *Testing Object-Oriented Software (Practitioners)*, 1st ed. Piscataway, NC: Wiley-IEEE Computer Society Press, 1998.

38. J. Stafford, and J. Bosch, Architecting Component-Based Systems, in *Building Reliable Systems from Components*. Norwood, MA: Artech House Publishers, 2002.

39. C. Szyperski, *Component Software. Beyond Object Oriented Programming*. Reading, MA: Addison Wesley, 1998.

40. B. Beizer, Black-Box Testing: *Techniques for Functional Testing of Software and Systems*. New York: Wiley, 1995.

41. J. Z. Gao, H. S. J. Tsao, and Y. Wu, *Testing and Quality Assurance for Component-Based Software*. Norwood, MA: Artech House Computer Library, 2003.

42. M. J. Harrold, D. Liang, and S. Sinha, An approach to analyzing and testing component-based systems, in: Proceedings of the First International ICSE Workshop on Testing Distributed Component-Based Systems, 1999.

43. M. Jaffar-ur Rehman , F. Jabeen, A. Bertolino, and A. Polini, Testing software components for integration: a survey of issues and techniques, *Softw. Test. Verif. Reliab.***17**: 95–133, 2007.

44. S. Beydeda and V. Gruhn, State of the art in testing components, in: QSIC '03: Proceedings of the Third International Conference on Quality Software, Washington, DC: IEEE Computer Society, 2003, pp. 146.

45. A. M. R. Vincenzi, J. C. Maldonado, M. E. Delamaro, E. S. Spoto, and W. E. Wong, Component-based software: an overview of testing. in: Component-Based Software Quality. Volume Lecture Notes in Computer Science. Berlin: Springer, 2003, pp. 99–127.

46. S. H. Edwards: A framework for practical, automated black-box testing of component-based software. *Softw. Test., Verif. Reliab.***11**: 97–111, 2001.

47. Y. Wang, G. King, and H. Wickburg, A method for built-in tests in component-based software maintenance, in: *CSMR '99: Proceedings of the Third European Conference on Software Maintenance and Reengineering*, Washington, DC: IEEE Computer Society, 1999, pp. 186.

48. E. Martins, C. M. Toyota, and R. L. Yanagawa, Constructing self-testable software components, in: *DSN '01: Proceedings of the 2001 International Conference on Dependable Systems and Networks*, Washington, DC: IEEE Computer Society, 2001, pp. 151–160.

49. F. Barbier and N. Belloir: Component behavior prediction and monitoring through built-in test, in: 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03), 2003, pp. 17–22.

50. M. Momotko and L. Zalewska, Component+ built-in testing - a technology for testing software components. Foundations of Computing and Decision Sciences, *Inst. Comput. Sci.* Poznan University Technol. **29**(1-2), 133–148, 2004.

51. Y. Wu, D. Pan, and M. H. Chen: Techniques for testing component-based software, in: ICECCS '01: Proceedings of the Seventh International Conference on Engineering of Complex Computer Systems, Washington, DC: IEEE Computer Society, 2001, pp. 222–232.

52. C. Atkinson and H. Gross, Built-in Contract Testing in Model-Driven Component-Based Development, Proceedings of Workshop on Component-Based Development Processes. 2002.

53. B. Korel, Black-box understanding of cots components, in: IWPC '99: *Proceedings of the 7th International Workshop on Program Comprehension*, Washington, DC: IEEE Computer Society, 1999, pp. 92+.

54. A. Polini and A. Bertolino, A user-oriented framework for component deployment testing, in: *Testing Commercial-off-the-Shelf Components and Systems*. Berlin: Springer Verlag, 2004.

55. L. Mariani and M. Pezzé, Behavior capture and test: Automated analysis of component integration. In: ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), Washington, DC: IEEE Computer Society, 2005, pp. 292–301.

56. S. Beydeda and V. Gruhn, *Testing Commercial-off-the-Shelf Components and Systems*. Berlin: Springer, 2005.

57. R. M. Poston, *Automating Specification-Based Software Testing*. Piscalaway, NJ: Institute of Electrical & Electronics Engineer, 1996.

58. A. van Lamsweerde, Formal specification: a roadmap, in: ACM ICSE 2000, The Future of Software Engineering, A. Finkelstein, 2000.

59. J. Tretmans, Testing Techniques. Technical report, University of Twente, The Netherlands, 2002.

60. M. R. Donat, Automating formal specification-based testing, in: TAPSOFT '97: Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development, London, UK: Springer-Verlag, 1997, pp. 833–847.

61. C. Jard and T. Jéron TGV: Theory, Principles and Algorithms, Conf. IDPT 2002, Pasadena, CA, 2002.

62. J. Tretmans: Test generation with inputs, outputs and repetitive quiescence, *Softw. Conc. Tools* **17**: 103–120, 1996.

63. G. von Bochmann and A. Petrenko, Protocol testing: Review of methods and relevance for software testing, In: ISSTA, 1994, pp. 109–124.

64. G. Bernot, Testing against formal specifications: a theoretical view, In: TAPSOFT '91: Proceedings of the international joint conference on theory and practice of software development on Advances in distributed computing (ADC) and colloquium on combining paradigms for software development (CCPSD), Vol. 2. New York: Springer-Verlag, 1991, pp. 99–119.

65. J. Tretmans, Testing concurrent systems: a formal approach, In: CONCUR'99. Volume LNCS 1664, 1999, pp. 46–65.

66. R. G. de Vries and J. Tretmans, Towards formal test purposes, In: Proc. FATES'01, Aalborg, Denmark, 2001.

67. J. C. C. Fernandez, C. Jard, T. Jeron, L. Nedelka , and C. Viho: An experiment in automatic generation of test suites for protocols with verification technology, *Sci. Comp. Progr.* **29** (1997) 123–146.

68. TorX: TorX Test Tool Information, Available: http://fmt.cs.utwente.nl/tools/torx/introduction.html.

69. TTCN: Testing and test control notation. Available: http://www.ttcn-3.org/.

70. R. Groz and N. Risser, Eight years of experience in test generation from fdts using tveda, In: FORTE X/PSTV XVII '97: *Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE X) and Protocol Specification, Testing and Verification (PSTV XVII)*, London, UK: Chapman & Hall, Ltd., 1998, pp. 465–480.

71. C. Jard and T. Jeron, TGV: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Int. J. Softw. Tools Technol. Transf.* **7**: 297–315, 2005.

72. TGV: Test Generation and Verification. Available: http://www-verimag.imag.fr/async/TGV/index.shtml.en.

73. A. Hartman, Model Based Test Generation Tools. Technical report, (AGEDIS project).

74. Object Management Group, OMG/Unified Modelling Language(UML) V2.1.2, 2007.

75. L. Naslavsky, D. J. Richardson, and H. Ziv: Scenario-based and State Machine-based Testing: An Evaluation of Automated Approaches. Technical report, University of California, Irvine, ISR Technical Report UCI-ISR-06-13, 2006.

76. L. Naslavsky, H. Muccini, and D. Richardson: Scenario-based and state machine-based testing: an evaluation of automated approaches, In: MTOOS 2006, First Int. Workshop on Model-based Testing and Object Oriented Systems, co-located with OOPSLA, 2006.

77. M. Utting, A. Pretschner, and B. Legeard, A taxonomy of model-based testing. Technical report 04/2006, Department of Computer Science, The University of Waikato, New Zealand, 2006.

78. M. Prasanna, S. N. Sivanandam, R. Venkatesan, and R. Sundarrajan, A Survey on Automatic Test Case Generation. Acad. Open Internet J. 2005.

79. AGEDIS: AGEDIS Project on Automated Generation and Execution of Test Suites for DIstributed Component-based Software. Available: http://www.agedis.de/index.shtml.

80. A. Belinfante, L. Frantzen, and C. Schallhart, Tools for test case generation, In: Model-Based Testing of Reactive Systems. Springer LNCS, 2005, pp. 391–438.

81. M. Broy, B. Jonsson, J. P. Katoen, M. Leucker, and A. Pretschner, eds.: Model-Based Testing of Reactive Systems: Advanced Lectures. Springer-Verlag Berlin and Heidelberg GmbH & Co. K (2005).

82. Forum: Model-based testing forum. Available: http://tech.groups.yahoo.com/group/model-based-testing/.

83. H. Muccini and M. Vieira, eds., ROSATEA 2007: 3rd Int. Workshop on The Role Of Software Architecture in Testing and Analysis, Co-located with CBSE and QOSA, 2007.

84. H. Muccini, A. Bertolino, and P. Inverardi:Using software architecture for code testing. *IEEE Trans. Softw. Engin.***30**: 160–171, 2003.

85. J. Dick and A. Faivre, Automating the generation and sequencing of test cases from model-based specifications, in: J. C. P. Woodcock and P. G. Larsen (eds.), *FME'93: Industrial-Strenght Formal Methods, LNCS 670*. Berlin: Springer Verlag, 1993, pp. 268–284.

86. R. H. Carver and K. C. Tai, Use of sequencing constraints for specification-based testing of concurrent programs, *IEEE Trans. Softw. Engin.* **24**(6): 471–490, 1998.

87. A. Bertolino and P. Inverardi: Architecture-based software testing, *Proc. ISAW96*. 1996.

88. D. J. Richardson and A. L. Wolf, Software testing at the architectural level, *ISAW-2, in Joint Proc. of the ACM SIGSOFT '96 Workshops*, 1996, pp. 68–71.

89. D. J. Richardson, J. Stafford, and A. L. Wolf, A Formal Approach to Architecture-based Software Testing. Technical report, University of California, Irvine, 1998.

90. M. J. Harrold Architecture-based regression testing of evolving systems, *Proc. Int. Workshop on the Role of Software Architecture in Testing and Analysis - ROSATEA 98*, 1998, pp. 73–77.

91. D. Rosenblum, Challenges in exploiting architectural models for software testing, *Proc. Int. Workshop on the Role of Software Architecture in Testing and Analysis-ROSATEA*. 1998.

92. Z. Jin and J. Offutt, Deriving tests from software architectures, *ISSRE*, 2001, pp. 308–313.

93. H. Muccini, M. Dias, and D. J. Richardson, Software architecture-based regression testing, *Int. J. Syst. Soft.*, **79**(10): 1379–1396, 2006.

94. H. Muccini, Software Architecture for Testing, Coordination and Views Model Checking. PhD thesis, University of Rome, La Sapienza, 2002.

95. PLASTIC: PLASTIC IST project, Deliverable D4.1: Test Framework Specification and Architecture. 2007. Available: http://www.ist-plastic.org.

96. A. Bucchiarone, H. Melgratti, and F. Severoni, Testing service composition, *Proceedings of the 8th Argentine Symposium on Software Enginnering (ASSE 2007)*, 2007.

97. F. Severoni, Studio Comparativo sui Framework di Testing per Web Services (in italian). Master's thesis, University of L'Aquila, Italy, 2007.

98. L. Baresi and E. D. Nitto, eds. *Test and Analysis of Web Services*, 1st ed. New York: Springer, 2007.

99. D. E. Perry and A. L. Wolf, Foundations for the study of software architecture, *ACM SIGSOFT Software Engineering Notes* **17**(4): 40–52, 1992.

100. UTP: Omg: Uml 2.0 testing profile. final adopted specification, 2005.

101. von, Paul Baker, Z. R. Dai, and J. Grabowski, Model-Driven Testing: Using the UML Testing Profile. Berlin, Springer, 2007.

102. I. Schieferdecker, Z. R. Dai, J. Grabowski, and A. Rennoch, The UML 2.0 testing profile and its relation to TTCN-3, *Proceedings of the 15th IFIP International Conference on Testing of Communicating Systems*, 2003.

103. Z. R. Dai, J. Grabowski, H. Neukirchen, and H. Pals: From design to test with UML-applied to a roaming algorithm for bluetooth devices, *Proceedings of the 16th IFIP International Conference on Testing of Communicating Systems* (TestCom2004), Oxford, UK: Lecture Notes in Computer Science 2978, 2004.

104. J. Zander, Z. R. Dai, I. Schieferdecker, and G. Din, From U2TP models to executable tests with TTCN-3 - an approach to model driven testing, *Proceedings of the 17th IFIP International Conference on Testing of Communicating Systems* (TestCom2005), 2005.

105. P. Clements and L. M. Northrop, *Software Product Lines: Practices and Patterns*, 1st ed. Reading MA: Addison-Wesley Pub Co, 2001.

106. C. W. Krueger, Introduction to software product lines. Available:http://www.softwareproductlines.com/introduction/introduction.html.

107. M. Jaring and J. Bosch, Representing variability in software product lines: a case study. SPLC, 2002, pp. 15–36.

108. H. Muccini and A. van der Hoek, Towards testing product line architectures, *Proc. ETAPS 2003 workshop on "Test and Analysis of Component Based Systems"* (Tacos). Warsaw, Poland, 2003.

109. J. D. McGregor, Testing a Software Product Line. Technical report, CMU/SEI-2001-TR-022, 2001.

110. A. Tevanlinna, J. Taina, and R. Kauppinen, Product family testing: a survey. *SIGSOFT Softw. Eng. Notes* **29**(2): 12–12, 2004.

111. H. Muccini, A. Polini, F. Ricci, and A. Bertolino, Monitoring architectural properties in dynamic component-based systems, In: *10th International ACM SIGSOFT Symposium*

on Component-Based Software Engineering. Volume Lecture Notes in Computer Science, LNCS 4608, 2007.

112. M. Boshernitsan, Technology transfer of testing and analysis research: observations from the receiving end, Invited Speaker at ISSTA 2006, the Int. Symposium on Software Testing and Analysis, 2006.
113. A. Hartman, Model Based Testing: What? Why? How? and Who cares? Invited Speaker at ISSTA 2006, the Int. Symposium on Software Testing and Analysis, 2006.
114. RISE 07, 4th Int. Workshop on Rapid Integration of Software Engineering techniques. (2007).
115. J. O. Kephart and D. M. Chess, The vision of autonomic computing. Computern **36**(1) (2003) 41–50.
116. IEEE Std 829-1998: IEEE standard for software test documentation, 1998.
117. A. Bertolino, Software testing research and practice, ASM 2003, Invited Presentation. Volume LNCS 2589. Taormina, Italy, 2003.

Henry Muccini
University of L' Aquila
L' Aquila, Italy

SOFTWARE VERIFICATION AND VALIDATION

Software verification and validation are software quality assurance activities that aim to ensure that the software system is developed according to a development process and meets the customer's needs (1). In other words, verification is about "are we building the product right," and validation is about "are we building the right product" (2). Validation is divided even more into static validation and dynamic validation. Static validation checks the correctness of the software product without executing the software system or a prototype, whereas dynamic validation executes the software system or a prototype. Software testing is one form of dynamic validation.

To explain, verification is concerned about the process to produce the product. That is, are we building the product in the right way? This process includes two aspects: (1) the right process and (2) correctly following the right process. As a minimum condition, the "right process" must require that a lower level artifact satisfy the requirements stated in the higher level artifact. Unlike verification, which is concerned about the "correctness" of the process, validation is concerned about the correctness of the product. That is, are we building the correct product? We need both verification and validation because either of them alone is not sufficient. For example, an implementation may satisfy the specification, but the specification may be incorrect. Moreover, the implementation may satisfy the specification and the specification is also correct, but the code may be hard to understand, test, and maintain. Customer review and/or expert review of requirements specifications would detect the former, whereas code review and code inspection would detect the latter. Therefore, a "right" (which means "preferred") software development process should include these verification and validation activities.

In the following sections, we first provide definitions of commonly encountered verification and validation concepts, followed by verification and validation in the software lifecycle, formal verification, and software testing techniques.

DEFINITIONS

This section presents the definitions of commonly used terminologies in software verification and validation.

Bug: A defect in the program code.

Desk checking: Examination of software artifact, typically the source code, by the developer to detect bugs, anomalies, and other potential problems.

Error: An unanticipated condition that puts the system into an incorrect state.

Failure: A result produced by the software under test does not satisfy the expected outcome.

Fault: A defect in the software system.

Inspection: A step-by-step checking of the software artifact/product against a predefined list of criteria, called a *checklist*.

Peer review: Evaluating the software artifacts by peers who are required to answer a list of questions to assess the artifact and provide improvement suggestions, if any.

Regression test: Rerun some test cases to ensure that the modified software system still delivers the functionality as required.

Software attribute: Property or characteristic of software.

Software metrics: Measurements of software (attributes).

Software quality assurance: Activities to ensure that the software under development or modification will meet desired quality requirements.

Test driver: Code to invoke the component under test and to check the outcome of the component under test.

Test harness: Test driver and test stub.

Test script: Code to test functionality of software system.

Test stub: Code to replace the module or procedure that is invoked by the component under test so that the component under test can be executed.

Testing: Executing a program with the intent to uncover bugs.

Verification and validation: According to Barry Boehm, verification is "are we building the product right?" And validation is "are we building the right product?"

Walk-through: Manually reviews the software artifact by following the described logic step by step with a certain scenario of operating the system and/or on certain input data as the test case. The artifact reviewed could be requirements specifications, high-level design, detailed design and source code, and so on. The walk-through of the source code is often performed by manually executing the software with test data to simulate machine execution of the software.

VERIFICATION AND VALIDATION IN THE LIFECYCLE

This section presents verification and validation in the software lifecycle. That is, what is checked in each lifecycle phase and who performs the checking. Moreover, what techniques are used to perform the checking.

Verification and Validation for the Requirements Phase

Verification and validation in the requirements phase detects errors in the requirements specification and

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

the analysis models. The techniques used include requirements reviews, inspection, walk-through, and prototyping. Requirements reviews include customer/user reviews, technical reviews, and expert reviews.

Customer/user reviews are performed by involving the customer and/or users of the system. Customer/user reviews should examine the requirements specification and look for problems in the following areas:

1. The correspondence between requirements and the real world. That is, the requirements specification correctly describes the functional requirements of the application for which the system is to be built or extended.
2. The user interfaces, which includes the appearance, the look and feel, sequence of interaction, input/output data and formats, and the GUI implementation technologies.
3. Nonfunctional requirements. That is, whether the nonfunctional requirements, including performance requirements, security requirements, and user friendliness requirements, are stated correctly.
4. Constraints. That is, whether application-specific constraints are stated correctly. Application-specific constraints may include constraints on the operating environment, political constraints, technological constraints, and so on.

Technical review is an internal review performed by the technical team. Technical review techniques include peer review, inspection, and walk-through:

- In peer review, the requirements specification and the analysis models are reviewed by peers, who are guided by a list of review questions designed to assess qualitatively the quality of the product being reviewed. Answers to the questions by the peers may vary drastically because the answers represent the reviewer's opinion about the product under review and heavily depend on the reviewer's knowledge, experience, background, and criticality. This process is similar to the product review reports published in consumer magazines. The review reports by different writers may differ drastically. The review meeting usually runs about two hours and takes place one to two weeks after the review assignments, which allows the developer and the peers to discuss the feedback and to identify action items to address the issues.
- Inspection checks the requirements and the analysis models against a list of items that are found to be error prone or problematic (3,4). Unlike reviews, inspection looks for more specific problems and the answers can be more objective. This process is similar to car inspection in which the inspector checks the engine, brake, lights, and so on. to see whether each of them is working properly. Since it is more well defined, a computer can perform a car inspection nowadays.
- Walk-through is carried out by explaining and examining the requirements (5). In particular, the analyst who wrote the requirements specification explains each requirement to the peers who would raise questions and stimulate doubts. The analyst would answer the questions and address the doubts. In addition, each of the analysis models is examined carefully. That is, the analyst who drafted the model leads the peers through the model and provides necessary clarification while the peers may ask questions and raise doubts. This process is similar to the new car salesperson at a car dealer who demonstrates a new car to potential buyers by showing various kinds of operations of the car. The buyers usually would raise questions and concerns during the demonstration, and the salesperson would address the questions and concerns.

The above verification and validation activities should aim to reveal the following problems:

- Incompleteness, which includes
  1. Definition incompleteness; for example, some application-specific concepts are not defined.
  2. Internal incompleteness; for example, some requirement expression has an "if part" but does not have the "else part." Another example is that a decision tree or decision table has not considered all possible combinations of the conditions used to construct the decision tree or the decision table.
  3. External incompleteness; that is, cases exist in the real-world application but are not included in the requirements specification. For example, a decision table or decision tree does not include a condition that should have been included.
- Inconsistency, which includes
  1. Type inconsistency; that is, an inconsistent specification is provided of one or more data types in the requirements or analysis model.
  2. Logical inconsistency; that is, contradictory conclusions can be inferred from the specification.
  3. View inconsistency; that is, inconsistency exists between views of the system by different user groups. For example, the perception of user group A is contradictory to the perception of user group B.
- Ambiguity, which includes
  1. Ambiguity in the definition of application specific concepts.
  2. Ambiguity in the formulation of requirements.
- Redundancy, which includes
  1. Duplicate definitions of the same concept.
  2. Duplicate formulations of the same requirement or constraint.
  3. Unnecessary concepts or constraints.
- Intractability, which means the high-level requirements do not correspond to the lower level requirements. If the system is being developed using the object-oriented paradigm, then the technical review must ensure that the use cases are tractable to the requirements and vice versa. It is often facilitated by constructing a requirements–use cases tractability matrix during the analysis phase. The matrix shows which requirement is to be realized by which use cases.

The review should ensure at a minimum that each requirement is realized by some use cases and that each use case serves to realize some requirements.

- Infeasibility in terms of performance, security, and cost constraint. That is, can the development team deliver the functional capabilities as stated in the requirements specification with the expected performance and security within the cost and schedule constraints?
- Unwanted implementation details. Implementation details must not be mentioned in the requirement specification because this limits the design space. Examples include mentions of pointers, physical data structures, and use of pseudocode or programming language statements.

Expert review in the requirements phase means review of the requirements specification by domain experts, looking for

1. Incorrect or inaccurate formulation of domain-specific laws, rules, behaviors, policies, standards, and regulations.
2. Incorrect, inaccurate, inappropriate, or inconsistent use of jargons.
3. Incorrect perception of the application domain.
4. Other potential domain-specific problems or concerns.

In the requirements phase, prototyping or rapid prototyping can take many different forms. The main purpose is to construct quickly a prototype of the system and to use it to acquire customer/user feedback. That is, prototyping is used as a validation technique in the requirements phase to help ensure that the team understands whether the customer/user requirements are captured correctly.

The simplest prototype could be a set of drawings that illustrate the user interfaces of the future system. The most sophisticated prototype could be a partially implemented system that the users can experiment with to gain hands-on experience. The most commonly observed prototype is one in which the team can demonstrate the functionality and user interfaces of the system to the customer or end users. Which type of prototype to use is an application-dependent issue. For instance, applications that are concerned mostly with mission-critical operations would benefit from prototypes that demonstrate the functionality and behavior. Applications that are end user oriented would benefit from prototypes that demonstrate the user interfaces.

The requirement phase is ideal for preparing system test cases to be used to validate the system before deployment. If use cases or scenarios have been used in requirements analysis, then they can be used to prepare system test cases. First, for each use case or scenario, the user input parameters are identified. Next, the possible input values of each input parameter are determined, which can be done as follows. For each input parameter, at least three possible cases can be considered (*1*) valid value is used, (*2*) an invalid value is used, and (*3*) the input parameter is not applicable or not available. A more refined approach will consider other partitions of the input parameter according to the application at hand. In addition to equivalence partitioning, boundary values for each input parameter can also be used. A table with the columns representing the input parameters and the rows representing the test cases can then be constructed and used during the system testing phase.

Verification and Validation for the Design Phase

Verification and validation in the design phase assess the correctness, consistency, and adequacy of the design with respect to the requirements and analysis models. Verification and validation activities in the design phase use review, inspection, walk-through, formal verification, and prototyping techniques. Depending on who performs these activities, we have peer review, customer review, and expert review. Peer review, inspection, walk-through, and formal verification are performed by the development team. They are mostly verification activities, although some of them may be concerned with design validation.

Peer review, inspection, walk-through, and formal verification check the design documentation to ensure

1. Correct use of the design language, which includes:
   - The notions and notations of the design specification language are used correctly.
   - The design specification expresses clearly and correctly the design of the proposed system.
2. Adequacy. The design specification prescribes a solution that is implemented that will satisfy the requirements of the proposed system. It can be done as follows:
   - The high-level verification ensures that each requirement is realized by some modules in the design and that each module in the design is necessary for satisfying some requirements.
   - The detail-level verification aims at ensuring that the capability stated in each requirement can actually be delivered when the system is implemented according to the design specification. It can be accomplished by a design traversal to demonstrate how the requirement can be satisfied.
3. Nonredundancy, which includes:
   - The design does not include items that are not necessary for satisfying the requirements or for significantly improving design quality. (For instance, design patterns may introduce additional classes, but proper use of design patterns significantly improves design quality.)
   - The design does not contain items that are already covered by another part of the design. For instance, a rule in a decision table may already be covered by other rule(s).
4. Consistency, which includes:
   - Logical consistency. That is, the various portions of the design specification do not contain contradictory design descriptions. For example, the decision table or decision tree are commonly used in the design phase to describe a process logic for modules. A

decision table or decision tree is inconsistent if two or more rules have the same condition combination but different action sequences. When the design is represented in a modeling language such as UML, it may contain several diagrams to represent the system from different views and/or at different levels of abstraction. These diagrams must also be checked for consistency across the diagram.

- Definition-use consistency. That is, the use of a component, class, data structure, data element, or function corresponds to the respective definition and interaction sequence. For example, the invocation of a function must correspond to the definition of the function signature and return type. A commonly observed inconsistency in object interaction design or sequence diagramming is an object calling another object, but the called function is not defined.
- Design/specification consistency. That is, the design specification is consistent with the models constructed in the analysis phase.

5. Internal completeness. Checking internal completeness is to ensure that the design has covered all possible combinations of a given set of conditions. For example, if a decision table has three binary conditions, then it must contain eight independent rules to cover the eight possible combinations of the three binary conditions.
6. Design principle compliance. The design follows well-known design principles such as separation of concerns, high cohesion, and low coupling. This process can be facilitated by computing and analyzing a set of design quality metrics, such as cohesion, coupling, scope of effect, scope of control, fan-in, fan-out, class size, height of inheritance tree, and design complexity metrics. For example, the class size metric is the number of methods in a class. If the class size less the number of getters and setters is large, then the class may have been assigned too many responsibilities. It then may signify that the cohesion of the class will be low. The reviewers can then focus their effort on examining such classes.
7. Module interface. That is, communication between modules is explicit and easy to understand. Moreover, no hidden assumptions should be provided for invoking a module.

Although peer review, inspection, and walk-through are concerned mostly with design verification, customer review and prototyping are concerned mainly with design validation. They are usually performed jointly by the development team and the customer (or customer representative, including the system analyst). As a design validation activity, customer review and prototyping aim at detecting mismatches, omissions, or inconsistencies between the design and the customer's interpretation of the requirements, including

1. Mismatch between designed functionality and/or behavior and the functionality and/or behavior as expected by the customer/users.
2. Mismatch between system states, events, and cases and the actual states, events, and cases in the business domain. It includes checking of external completeness.
3. Mismatch between the system's user interface design and what is expected by the customer/users.
4. Mismatch between the system's interfaces to other systems and the required interfaces in the real world.

Another validation activity in the design phase is the preparation of functional test cases, behavioral test cases, and integration test cases. The design phase is ideal for the preparation of these test cases because all needed information is contained in the design documents. For example, if decision tables have been used in the design phase to express process logic, then each rule of the decision table is a cause–effect test case. If state machines have been used in the design phase to describe state-dependent behaviors, then the state machines can be used to derive transition sequences to test the implemented state-dependent behaviors.

Integration test cases can be derived from structured charts (also called routine diagrams) using a preorder traversal in top-down integration and postorder traversal in bottom-up integration. If the system is being developed using an object-oriented approach, then the integration test cases can be derived from sequence diagrams or collaboration diagrams, that is, deriving test cases that will exercise message passing paths according to the coverage criteria selected.

Verification and Validation for the Implementation Phase

Verification and validation for the implementation phase ensure that the source code complies with the organization's coding standards; implements the required functionality, satisfies performance, real-time, and security requirements; and properly handles exceptional situations. Desk checking, code review, inspection, and walk-through are referred commonly to as verification and static validation methods, whereas testing is referred commonly to as the dynamic validation method. All these process are used in the implementation phase.

In desk checking, the programmer checks the program written by him/her. The programmer may use a pencil, a calculator, and/or other devices. It is an informal process, and hence, the effectiveness and efficiency depends on the individual programmer. In code review, the program is reviewed by peers who are required to comment on the quality of the code and answer a set of questions. Code inspection checks the code against a list of problems or defects that are commonly found in programs. The most famous code inspection method was proposed by Fagan and is called the Fagan inspection method (3,4). In walk-through, the reviewers use test data or a specific scenario in the operation of the software and manually follow step by step the logic described in the artifact under review to understand how the system operates and then to detect errors (5). For example, when the artifact under review is a piece of source code, the reviewers manually execute the program by following the control flow between the statements and expressions in the code. Finally, testing is actually executing the program with test cases derived using ad hoc or systematic test case generation methods.

Desk checking, code review, code inspection, and walkthrough are aimed to detect problems such as follows:

1. Incorrect/inadequate implementation of functionality.
2. Mismatch of implementation and design.
3. Mismatch of module interfaces.
4. Coding standards are not followed.
5. Poor code quality as measured by various code quality metrics such as cyclomatic complexity (e.g., some companies require this to be no more than 10), information hiding, cohesion and coupling, and modularity.
6. Improper use of the programming language.
7. Incorrect/improper implementation of data structures or algorithms.
8. Errors/anomalies in the definition and use of variables such as variables or objects are defined but not used, not initialized, or not initialized correctly.
9. Infinite loop.
10. Incorrect use of logical, arithmetic, or relational operators.
11. Incorrect invocation of functions.
12. Inconsistencies caused by concurrent updates to shared variables.

Desk checking, code review, code inspection, and walkthrough are effective in detecting errors and anomalies if applied properly. In particular, ordinary testing methods may not detect problems as described by items 4–6 and 8–12. On the other hand, testing is distinct and indispensable because testing can detect performance bottlenecks and incorrect interface. These issues usually cannot be detected by the static validation methods.

Verification and Validation for Integration Phase

In the integration phase, the software modules are integrated to form a complete software system. Dynamic validation or testing is the main activity of this phase. The purpose of integration testing is to detect errors in the interfaces between the software modules. These errors include

1. Incorrect assignment of actual parameters to formal parameters.
2. Incorrect assignment of values to variables in one module and/or incorrect use of the variables in another module.
3. Incorrect interaction between modules. For example, incorrect sequence of function calls or module invocations.
4. Incorrect state behavior resulting from module interactions

Integration testing can be carried out by using one or more of the following strategies. These strategies assume that the architectural design has a tree or lattice structure with a top-level module that invokes second-level modules, which invoke third-level modules, and so on:

1. Top-down strategy. Integration testing begins with testing the interfaces between the top-level module that corresponds to the overall system and modules that are invoked by the top-level module. Lower level modules that are invoked by modules being integrated are replaced by test stubs. A test stub is a module that is constructed specifically to provide the output values as expected by the higher level module. We need to use test stubs because we have not tested the interfaces between the modules being integrated and the modules being replaced; if any of these interfaces is incorrect, then the error may propagate up and affect the integration testing result at the higher level.
2. Bottom-up strategy. Integration testing begins with testing the interfaces between the lowest level modules and their parent module and progresses up the hierarchy. A test driver is needed to invoke the parent module because the interface between the parent module and its parent module has not been tested.
3. Hybrid strategy. As the name suggests, integration testing may proceed using both of the above strategies in various combinations.
4. Criticality-based strategy. Integration testing begins with integrating critical modules of the system first as long as the modules are available. This strategy allows the critical modules to be exercised more often and hopes to detect more errors in these modules.
5. Availability-based strategy. Integration testing is carried out incrementally by adding modules that are ready to be integrated into the software system.
6. Monolithic strategy. Integration testing is performed by integrating all modules of the system at once.

Verification and Validation for System Testing

During the system testing phase, the software system is integrated with other systems and tested against the software/system requirements. System testing is usually performed in the development environment. The end product of system testing is a system that is ready for deployment and acceptance testing in the customer's target environment.

As indicated, system testing is performed against the software/system requirements, including functional and nonfunctional requirements. The objective is to ensure that the system satisfies the functional and nonfunctional requirements. In addition, the system must also satisfy the constraints stated in the requirements specification. System testing with respect to functional requirements can be carried out using one or more of the following approaches:

- Use case-based testing. As described in the section entitled "Verification and Validation for the Requirements Phase," if system use cases have been derived from the requirements, then system testing can be performed by testing that the system satisfies each use case. Please see this section for more detail.

- Random testing. Test data are selected randomly to test the system against the requirements. This process may or may not use an input data distribution profile, which can be obtained from existing or similar systems' usage log.

In addition to functional testing, performance and stress testing are also performed during the system testing phase. Performance testing includes testing the throughput and response time according to the predefined workload, and stress testing is concerned with system throughput and response time under a workload that is multiple times or even ten folds of the normal workload.

Verification and Validation for Acceptance Testing

During the acceptance testing phase, the analyst or a consultant hired by the customer will conduct or direct the testing of the system in the customer's target environment to ensure that the system operates properly in that environment. Because the difference between system testing and acceptance testing is the environment, acceptance testing can be carried out by executing a subset of the test cases used during system testing. Clearly, test cases selection should be guided by changes to environment parameters, such as system configuration, run conditions, and network configurations.

Verification and Validation for Maintenance

Once the system is installed and operational in the target environment, the maintenance phase begins. Therefore, the operation and maintenance phases are in fact one combined, indivisible phase. Because of system dynamics (6), continual changes are made to the system once it is released to field operation. Changes or enhancements performed on the system are called collectively maintenance activities, which include:

- Corrective maintenance to correct errors in the system.
- Enhancements to add additional capabilities to the system.
- Improvements to the system, including performance, response time, user friendliness, and other quality aspects.
- Migration to new hardware, new technologies, or new operating environment.
- Preventive maintenance to prepare the system for possible problems such as virus attack.

The verification and validation techniques such as review, inspection, walk-through, and testing can still be used in the maintenance phase to verify and validate the changes. However, several issues must be considered during the maintenance phase:

- Change impact analysis. Changes can affect other parts of the system, and the impact must be identified and analyzed before the changes are made. This process is described usually in the Engineering Change Proposal along with change cost and schedule and is evaluated by a Change Control Board. This topic is beyond the scope of this article and is covered by the article "Software Configuration Management."
- Review, inspection, and walk-through may be conducted for new, changed, and affected modules.
- New test cases must be designed to test the newly introduced modules.
- The changed and affected modules must be retested using existing test cases to ensure that no undesired side effect has been introduced. This process is commonly called regression testing.

FORMAL VERIFICATION

Formal verification is a means to verify a specification or a design mathematically. Two main approaches to formal verification exist.

The first approach is based on theorem-proving methods (7–9). We call this approach the proof-theoretical approach. In this approach, a system specification consists of a set of declarative statements or declarative sentences. These statements typically specify properties of real-world and/or system entities or objects, their behaviors, and their relationships. In mathematical terms, the set of statements is called a theory and is assumed to be true at all times because the statements state what are about the system. In computer science and software engineering, the statements are called nonlogical axioms because they are not logically true but assumed to be true according to laws of the real-world application. For example, "every customer has an account" and "every account is owned by a customer" cannot be proved to be true logically, but they could be true for some bank application. Formal verification in the proof-theoretical approach is to prove that desired system properties or constraints are logical consequences of the nonlogical axioms. That is, desired properties or constraints can be derived logically from the nonlogical axioms.

Consider, for example, an overly simplified formal specification of a stack:

1. Maximal size of stack.

$$\mathrm{MAX} = 2$$

2. Initial size of stack.

$$\mathrm{size(S0)} = 0$$

where S0 denotes the initial state.

3. Operation "push" (we focus only on the size but nothing else).

$$\mathrm{size(S)} = \mathrm{s}\ \&\ \mathrm{s} < \mathrm{MAX} \rightarrow \mathrm{size(push(S)} = \mathrm{s} + 1$$

(If stack size in state S is s and s is less than MAX, then stack size in the state resulting from pushing a element onto the stack is s+1.)

4. Operation "pop,"

$$\text{size(S)} = \text{s} \,\&\, \text{s} > 0 \rightarrow \text{size(pop(S))} = \text{s} - 1$$

Now suppose we want to prove the desired property stating that "there is some state in which the stack size will be MAX.-formally

5.

$$(\exists \text{S})\text{size(S)} = \text{MAX}$$

That is, a state S exists in which the size of the stack is MAX.

We will illustrate the proof using the resolution proof technique proposed by Robinson (10). To prove that Q is a logical consequence of P1, P2, . . ., Pn, we prove that $\sim$Q, P1, P2, . . ., Pn cannot be true at the same time, where Q, P1, P2, . . ., Pn are statements.

Aresolution proof begins with the set of statements {Q, P1, P2, . . ., Pn}, and each resolution tries to deduct a statement called resolvent from two statements using the logical inference rule "A & $(\text{A} \rightarrow \text{B}) \Rightarrow \text{B}$" or equivalently "A & $(\sim \text{A} \vee \text{B}) \Rightarrow \text{B}$." That is, from statement "A" and statement "$\sim \text{A} \vee \text{B}$," we can deduct "B." Clearly, each resolution step takes two statements and produces one new statement. If the set of statements can be deduced to produce the nil statement, denoted by "□" and representing a contradiction, then the theorem is proved. The proof of our stack example is shown in Fig. 1.

Figure 1 is a special case because it does not use the so-called "frame axiom" originally proposed by McCarthy and Hayes (11). In their effort to construct the first question answering system using logical inference, McCarthy and Hayes discovered that the specification of the effect of an operation like items 3 and 4 in the above stack specification example is not enough. The specification must also state that everything that is not changed by the operation remains true in the new state resulting from the operation. This process is commonly referred to as the "frame axiom." Fortunately, nothing is not changed by the operations push and pop; therefore, our simple example does not have to use the frame axiom.

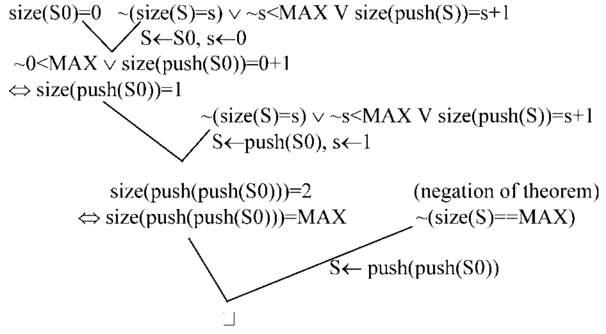

size(S0)=0
~(size(S)=s) ∨ ~s<MAX V size(push(S))=s+1
S←S0, s←0
~0<MAX ∨ size(push(S0))=0+1
⇔ size(push(S0))=1
~(size(S)=s) ∨ ~s<MAX V size(push(S))=s+1
S←push(S0), s←1
size(push(push(S0)))=2
⇔ size(push(push(S0)))=MAX
(negation of theorem)
~(size(S)==MAX)
S← push(push(S0))
□

Figure 1. Resolution proof of the simplified stack specification.

Now suppose we want to prove another desired property that states "the size of the stack is always greater than or equal to zero." Formally

6.

$$(\forall \text{S})\text{size(S)} > \,= 0$$

The reader will soon discover that applying resolution to prove this property is extremely difficult (almost impossible). A proof technique that is commonly used to prove theorems that state properties true for all cases like this is the proof by induction technique. Using an induction proof, the property is proved for the basis case, and then it is assumed to be true for all cases up to a number k; finally, the property is proved for the k+1 case. We illustrate this in the following. We use op(S) to denote either push(S) or pop(S) and op^k(S) a sequence of k push or pop operations applied in S.

The basis step. Since size (S0) = 0 is given in item 2 in the specification, this implies size(S0)>=0. Therefore, property is true in S0.

The hypothesis step. Now assume that size(op^k(S0))>=0 for all sequences of k push or pop operations applied in the initial state.

The induction step. We need to prove size(op^{k+1}(S0))>= 0. Since there are only two operations, size(op^{k+1}(S0)) can only be size(push(op^k(S0))) or size(pop(op$^{\text{k}}$(S0))). Since size(push(op^k(S0))) = size(op^k(S0))+1 according to item 3) and size(op^k(S0))>=0 from to hypothesis, size(push(-op^k(S0))) > 0, and hence size(push(op^k(S0))) >=0. Moreover, since size(op^k(S0))>=0 and pop can only be applied in state op^k(S0) if size(op^k(S0))>0. Thus, size(push (op^k(S0))) >=0. Therefore, size(op^{k+1}(S0))>=0.

A property regarding a software system that is true in all states, like the above example, is called an invariants.

The second approach is called model checking(12–15). This approach can also be called the model-theoretical approach. In this approach, the system is represented by an operational model, which typically depicts the system behavior. The commonly used operational model for model checking is a state machine consisting of vertices representing system states and directed edges representing system behaviors that cause state transitions. Each system state is specified by a logical or conditional statement. That is, the system is in that state if and only if the condition is evaluated to true using system attributes. Formal verification in the model-checking approach begins with the initial system state and generates the states by applying the operations. The desired properties or constraints are checked against each state generated, and violations are reported.

Consider a simplified thermostat example consisting of only a season switch, an AC relay and a furnace relay as

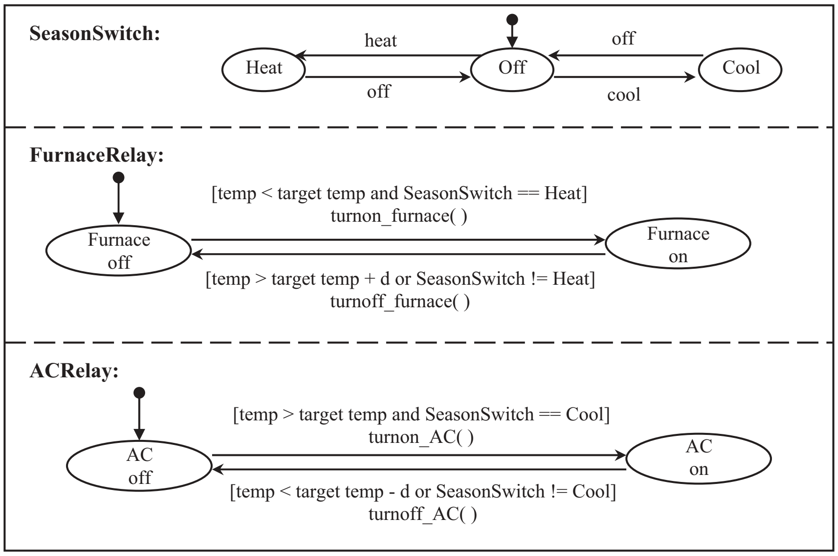

SeasonSwitch:
Heat
Off
Cool
heat
off
off
cool
FurnaceRelay:
Furnace off
Furnace on
[temp < target temp and SeasonSwitch == Heat]
turnon_furnace( )
[temp > target temp + d or SeasonSwitch != Heat]
turnoff_furnace( )
ACRelay:
AC off
AC on
[temp > target temp and SeasonSwitch == Cool]
turnon_AC( )
[temp < target temp - d or SeasonSwitch != Cool]
turnoff_AC( )

Figure 2. Thermostat specification.

shown in Fig. 2. The desired properties for the thermostat could be as follows:

C1. Not (SeasonSwitchOff and (FurnaceOn or ACOn))

C2. Not (FurnaceOn and ACOn)

C3. Not (SeasonSwitchCool and FurnaceOn)

C4. Not (SeasonSwitchHeat and ACOn)

Applying the operations of the thermostat results in the tree as shown in Fig. 3. A system state is represented by a triple (S1, S2, S3), where S1 denotes the state of the season switch, S2 denotes denotes the state of the furnace relay, and S3 denotes the state of the AC relay. The figure shows that starting in the initial state, the thermostat can enter into a state in which the season switch is at cool and the furnace and AC are both on. This state violates constraint C2 and constraint C3. In practice, model checking can be used to check not only static constraints like C1–C4 but also temporal constraints that involve sequences of states rather than a single state. This process is also true for the theorem-proving approach. Furthermore, the model checker could explore millions of state rather than only a few states as shown in Fig. 3.

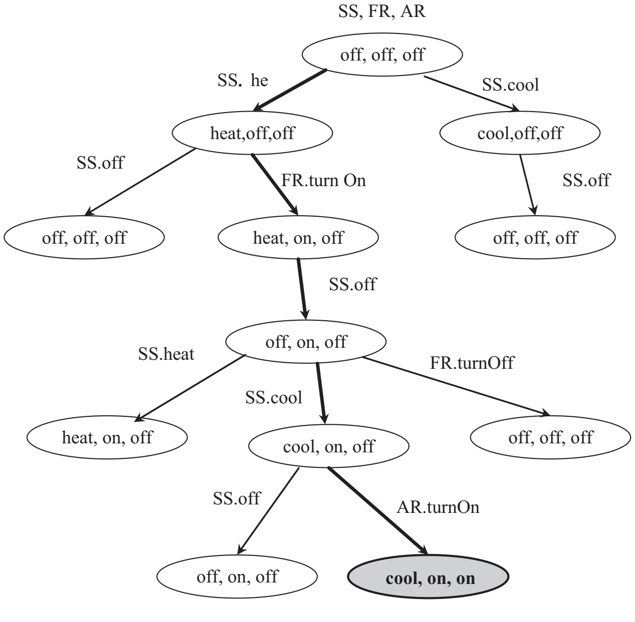


Figure 3. Partial state space of the thermostat example.

In practice, the state machine models are converted into the specification language of the model checker. Using SPIN (14), this would be the Promela language, which is a subset of the C programming language. The property to be verified is expressed as a temporal logic expression. The checker will explore the state space and verify the property.

In recent years, model checking has been applied to checking code or implementation rather than to checking the specification (16–18). This process has been termed “software model checking.” In software model checking, the model is constructed from code or implementation rather than from the specification. The construction can be manual or semiautomatic.

SOFTWARE TESTING TECHNIQUES

This section gives a brief introduction to well-known software testing techniques and methods.

Software Testing Processes

Generally speaking, software testing is an iterative process that involves several technical and managerial activities. In this section, we will focus on the technical aspects. As shown in Fig. 4, the main technical activities in the software testing process include planning, generating, and selecting test cases; preparing a test environment; testing the program under test; observing its dynamic behavior; analyzing its observed behavior on each test case; reporting test results; and assessing and measuring test adequacy.

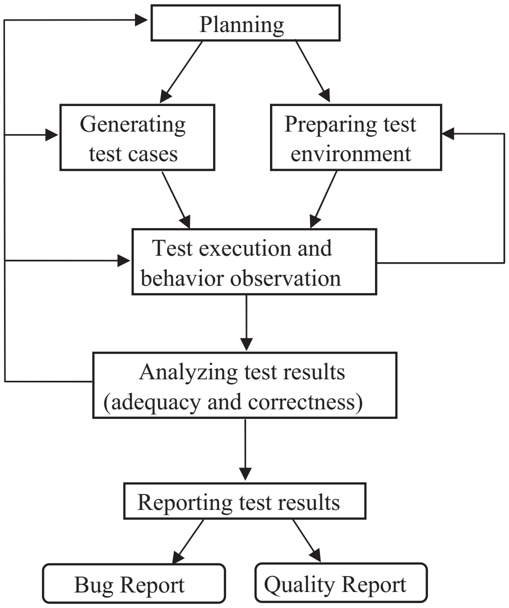

Planning
Generating test cases
Preparing test environment
Test execution and behavior observation
Analyzing test results (adequacy and correctness)
Reporting test results
Bug Report
Quality Report

Figure 4. Illustration of activities in software testing process.

In software testing practice, testers are confronted with questions like: Which test cases should be used? How to determine whether a testing is adequate? Or when can a testing process stop? These questions are known as the *test adequacy* problem (19). They are the central issues in software testing, and the most costly and difficult issues to address. A large number of test criteria have been proposed and investigated in the literature to provide guidelines to answer these questions. Some of them have been used in software testing practice and are required by software development standards. A great amount of research has been reported to assess and compare their effectiveness and efficiency.

The observation of dynamic behavior of a program under test is essential for all testing. Such observations are the basis of validating a software's correctness. The most often observed software behaviors are the input–output of the program during testing. However, in many cases, observation of the internal states, the sequences of code executed, as well as other internal execution histories are necessary to determine the correctness of the software under test. Such internal observations are often achieved by inserting additional code into the program under test, which is known as *software instrumentation*. Automated tools are available for the instrumentation of programs in various programming languages. Behavior observation can also be a very difficult task, for example, in the testing of concurrent systems because of nondeterministic behavior in testing component-based systems because of the unavailability of source code, in testing real-time systems because of their sensitiveness to timing and load, in testing systems that are history sensitive such as machine learning algorithms where the reproduction of a behavior is not always possible, in testing of service-oriented systems because of the lack of control of third-party services, and so on.

Checking the correctness of a program's output as well as other aspects of dynamic behavior is known as the test *oracle problem*. A test oracle is a piece of program that simulates the behavior of the program under test. It could be as simple as a person or a program that judges the output of the program under test according to the given input. If a formal specification of the system is available, then the output can be judged automatically, e.g., by using algebraic specifications (20–22). A recent development in the research on the metamorphic software testing method enables testers to specify relationships between outputs of a program on several of test cases and to check whether the relationships held during testing (23).

Testing methods

Testing activities, especially test case selection and generation and test adequacy assessment, can be based on various types of information available during the testing process. For example, at the requirements stage, test cases can be selected and generated according to the requirements specification. At the design stage, test cases can be generated and selected according to the architectural design and detailed design of the system. At the implementation stage, test cases are often generated according to the source code of the program. At the maintenance stage, test cases for regression testing should take into consideration the part of the system that has been modified, either the functions added or changed or the parts of the code that are modified. In general, software testing methods can be classified as follows[1].

- *Specification-based testing methods*. In a specification-based testing method, test results can be checked against the specification, and test cases can be generated and selected based on the specification of the system. For example, test cases can be generated from algebraic specifications (24), derived from specifications in Z (25, 26), or using model checkers to automatically generate test cases from state machine specifications (27, 28).
- *Model-based testing methods*. A model-based testing method selects and generates test cases based on diagrammatic models of the system, which could be a requirements model or design model of the system. For example, in traditional structured software development, test cases can be derived from data flow, state transition, and entity-relationship diagrams (29). For testing object-oriented software systems, techniques

[1] Traditionally, testing methods were classified into *white-box* and *black-box* testing. *White-box* testing was defined as testing according to the details of the program code, whereas black-box testing does not use the internal knowledge of the software. Many modern testing methods are difficult to classify either as black box or as white box. Thus, many researchers now prefer a more sophisticated classification system to better characterize testing methods.

and tools have been developed to generate test cases from various UML diagrams (30, 31) .

- *Program-based testing methods*. A program-based testing method selects and generates test cases based on the source code of the program under test. Tools and methods have been developed to generate test cases to achieve statement, branch, and basis path coverage. Another program-based testing method is the so-called *decision condition testing* method, such as the *modified condition/decision coverage* (MC/DC) criterion (32) and its variants (33), which focus on exercising the conditions in the program that determine the directions of control transfers.
- *Usage-based testing methods*. A usage-based testing method derives test cases according to the knowledge about the usage of the system. For example, a random testing method uses the knowledge about the probability distribution over the input space of the software, such as the operation profile. Another commonly used form of usage-based testing is to select test cases according to the risks associated with the functions of the software.

It has been recognized for a long time that testing should use all types of information available rather than just rely on one type of information (34). In fact, many testing methods discussed here can be used together to improve test effectiveness.

Testing Techniques

Several software testing techniques have been developed to perform various testing methods. These testing techniques can be classified as follows.

- *Functional testing techniques*. Functional testing techniques thoroughly test the functions of the software system. They start with the identification of the functions of the system under test. The identification of functions can be based on the requirements specification, the design, and/or the implementation of the system under test. For each identified function, its input and the output spaces and the function in terms of the relation between the input and the output are also identified. Test cases are generated in the function's input/output spaces according to the details of the function. The number of test cases selected for each function can also be based on the importance of the function, which often requires a careful risk analysis of the software application. Usually, functions are classified into high risk, medium risk, or low risk according to the following criteria

1. The cost and the consequences that a failure of the function may cause.
2. The frequency with which the function will be used.
3. The extent to which the whole software systems' functionality and performance depends on the function's correctness and performance.
4. The likelihood that the implementation of the function contains faults, say because of high complexity, the capability, and maturity of the developers, or any priori knowledge of the system.

A heuristic rule of functional testing is the so-called 80–20 rule, which states that *80% of test efforts and resourses should be spent on 20% of the functions of the highest risks*.

An advantage of functional testing techniques is that various testing methods can be combined. For example, functions can be identified according to the requirements specification. If additional functions are added during design, they can also be identified and added into the list of functions to be tested. An alternative approach is to identify functions according to the implementation, such as deriving from the source code. When assigning risks to the identified functions, many factors mentioned in the above criteria can be taken into consideration at the same time. Because some factors are concerned with users' requirements and some are related to the design and implementation, it naturally combines requirements-based with design and implementation-based methods. The main disadvantage is that functional testing techniques are largely manual operations, although they are applicable to almost all software applications.

- *Structural testing techniques*. Structural testing techniques regard a software system as a structure that consists of a set of elements of various types interrelated to each other through various relationships. They intend to cover the elements and their interrelationships in the structure according to certain criteria. Typical structural testing techniques include control flow testing and data flow testing techniques and various techniques developed based on them.

  Control flow testing techniques represent the structure of the program under test as a flow graph that is a directed graph where nodes represent statements and arcs represent control flows between the statements. Each flow graph must have a unique entry node where computation starts and a unique exit node where computation finishes. Every node in the flow graph must be on at least one path from the entry node to the exit node. For instance, the following program that

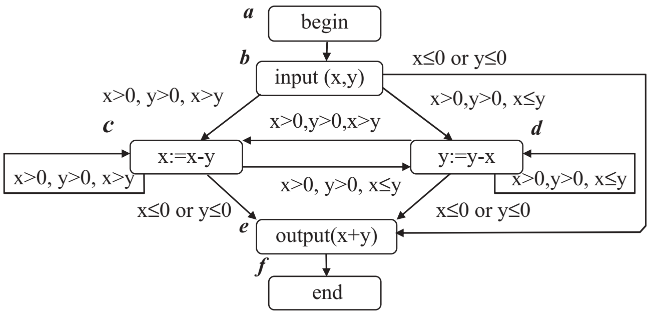


Figure 5. Flow graph of the Greatest Common Divisor program.

computes the greatest common divisor of two natural numbers using Euclid's algorithm can be represented as a flow diagram shown in Fig. 5.

```
 Procedure Greatest-Common-Divisor;
 Var x, y: integer;
 Begin
    input (x,y);
    while (x>0 and y>0) do
       if (x>y)
              then x:= x-y
              else y:= y-x
    endif
    endwhile;
    output (x+y);
end
```

As a control flow testing method, statement testing requires the test executions of the program on test cases exercise all the statements, i.e. nodes, in the flow graph. For example, paths $p = (a, b, c, d, e, f)$ in Fig. 5 cover all nodes in the flow graph; thus, the test case t_1= (x=2, y=1) that causes the path p to be executed is adequate for statement testing. Obviously, adequate statement testing may not execute all the control transfers in the program. Branch testing requires the test cases to exercise all the arcs in the flow graph, i.e. all the control flows, thus the branches, of the program. The test case t_1 is therefore inadequate for branch testing. Various path testing techniques require test executions cover various types of paths in the flow graph, such as all paths of length-N for certain fixed natural number N, all simple paths (i.e., the paths that contain no multiple occurrences of any arcs), all elementary paths (i.e., paths that contain no multiple occurrences of nodes), and so on.

Data flow testing techniques focus on how values of variables are assigned and used in a program. Each variable occurrence is therefore classified to be either a definition occurrence or a use occurrence:

– *Definition occurrence*: Where a value is assigned to the variable.

– *Use occurrence* (also called *reference occurrence*): Where the value of the variable is referred to. Use occurrences are also classified into *computation uses (c-use) and predicate uses (p-use).*

Predicate use: Where the value of a variable is used to decide whether a predicate is true for selecting an execution path.

Computation use: Where the value of a variable is used to compute a value for defining other variables or as an output value.

For example, in the assignment statement y:= x_1 − x_2, variables x_1 and x_2 have a computation use occurrence, whereas variable y has a definition occurrence. In the if-statement *if* x=0 *then goto L endif*, variable x has a predicate use occurrence. Figure 6 shows the flow graph with data flow information of the program given in Fig. 5.

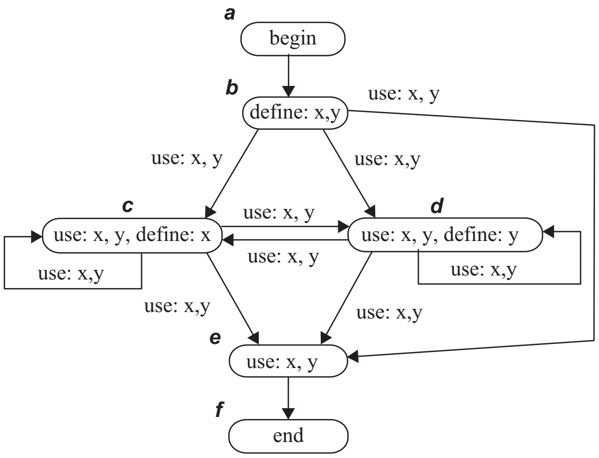
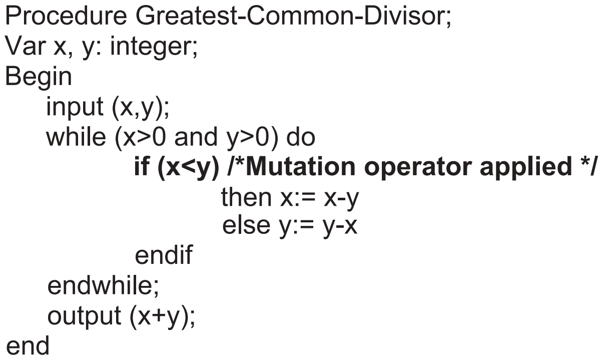


Figure 6. Flow graph with data flow information.

Using such data flow information, the data flow in a program can be expressed by the paths from a node where a variable x is defined to a node where the variable is used, but no other definition occurrence of the same variable x on the path (which is called the *definition-clear path of* x). Such a path is called a *definition-use association*. The principle underlying all data flow testing is that the best way to test whether an assignment to a variable is correct is to check it when its assigned value is used. Therefore, data flow test criteria are defined in the form of exercising definition-use associations or various compositions of the relation. For example, a data flow test criterion in Weyuker–Rapps–Frankl's data flowing testing techniques require testing all definition-use associations(35, 36) . Other data flow testing techniques include Laski and Korel's *definition context coverage criteria* (37), and Ntafos's *interaction chain coverage criteria* (38).

- *Fault-based testing techniques*. Fault-based testing techniques detects all faults of certain kinds in the software. For example, *mutation testing* detects all the faults that are equivalent to mutants generated by a set of mutation operators (39,40). In general, a mutation operator is a transformation that modifies the software with a single small change and preserves the software's syntax to be well formed. For example, a typical mutation operator changes a greater than symbol > in an expression to be the less than symbol <. When this mutation operator is applied to the



```
Procedure Greatest-Common-Divisor;
Var x, y: integer;
Begin
   input (x,y);
   while (x>0 and y>0) do
         if (x<y) /*Mutation operator applied */
               then x:= x-y
               else y:= y-x
         endif
   endwhile;
   output (x+y);
end
```

Figure 7. A mutant of the Greatest Common Divisor program.

Table 1. Levels of mutation analysis

Level	Goal	Method (Mutation operators)
Interface Analysis	Ensure interfaces between software components are correct and adequately tested in integration testing.	(1) Mutation operators are designed to model integration errors, (2) Tests only the connections between two modules, a pair at a time, and (3) Applies integration mutation operators only to module interfaces such as function calls, parameters, or global variables.

Language Specific Feature Analysis	Ensure language-specific features were used properly.	For example, for test of Java-specific features:
		Delete and insert *This* keyword;
		Delete and insert *Static* keyword;
		Delete member variable initialization; etc.

Polymorphism Analysis	Exercise all possible dynamic-type bindings to ensure the correctness of polymorphic behavior of object references.	Change the instantiation type of an object reference to a child or parent class; Delete, insert, or change type cast operator; Delete overloading method declarations; Change the parameters of overloading method calls.
Inheritance relationship Analysis	Ensure the inheritance relationships, including variable hiding, method overriding, uses of super, and definition of constructors, are correctly defined.	Delete or insert overriding methods and hiding variables; Change the calling position of overriding methods, Rename overriding methods; Delete and insert keyword 'Super'; Delete and insert parent constructor calls; etc.
Class Encapsulation Analysis	Ensure class declarations correctly use encapsulation facilities for various accessibility levels.	Change the access modifiers (i.e., private, protected, public, and unspecified) of the attributes and methods in class declarations.
Statement Analysis	Ensure that every branch is taken and the every statement is necessary.	Replace statement with CONTINUE; Replace logical and relational with true or false; Check labels on arithmetic IF statements for usage; Replace DO statements with FOR statements.
Predicate Analysis	Exercise predicate boundaries.	Alter predicate and DO loop limits sub-expressions by small amounts; Insert absolute value operators into predicate sub-expressions; Alter relational operators.r
Domain Analysis Coincidental Correctness Analysis	Exercise different data domains. Guard against coincidental correctness.	Change constants and sub-expressions by small amounts; Change data references and operators to other syntactically correct alternatives.

condition of the if-statement in the program given in Fig. 5, the mutant in Fig. 7 will be generated.

```
Procedure Greatest-Common-Divisor;
Var x, y: integer;
Begin
   input (x,y);
   while (x>0 and y>0) do
         if (x<y) /*Mutation operator applied*/
                  then x:= x-y
                     else y:= y-x
            endif
      endwhile;
      output (x+y);
   end
```

Each mutation operator represents a kind of error that could be made by software developers. If a test case enables the original software under test and the mutant to produce different outputs, we say that the mutant is killed by the test case or simply that the mutant is dead, which means that the modified part of the program has been executed and that the part actually affects the behavior of the system. Therefore, if the original program contains a fault at the location where the mutation operator is applied, the test case should be able to detect it. Otherwise, solely based on the test executions on the test cases, we would have no evidence to claim that the test cases are capable of differentiating the mutants from the original. In other words, if a fault exists, the test cases would not be able to detect it. Of course, there are two reasons that a mutant remains alive after testing on all test cases. First, the mutant is equivalent to the original. Thus, it cannot be killed. Second, the test cases were unable to kill it because of its inadequacy. The proportion of nonequivalent mutants that remain alive after testing, which is called *mutation score* in the software testing literature, gives a clear indication of the adequacy of the test set and serves as a test adequacy criterion.

Measuring the mutation score of a test set is, therefore, an analysis of the test adequacy. Different levels of mutation analysis can be done by applying mutation operators to the corresponding syntactical structures in the program (41–44). Table 1 summarizes the levels of mutation analysis and the methods to achieve the goals of the analysis.

Mutation testing tool such as Mothra for Fortran (42) and MuJava for Java (45) have been developed to generate automatically a large number of mutants from a program under test and to execute the program under test and the mutants and to collect the data about dead and alive mutants. Test cases can also be generated to kill a mutant (46). The equivalence of a mutant to the original is not decidable, but it can be determined automatically for a large proportion of mutants.

The idea of program mutation testing can also be extended to specification-based testing in which mutants of specifications are generated (47). A specification mutant is killed if the correctness of the output of the program under test is judged differently by the original specification.

More recently, the idea of mutation has also been applied to generate test data. A set of mutation operators designed so that when applied to a test case, they generate a set of test data that are of subtle differences from the original test case (48). This technique can be applied to test software systems whose test cases are of a complicated structure, such as modeling tools, and other test case generation techniques would have difficulties.

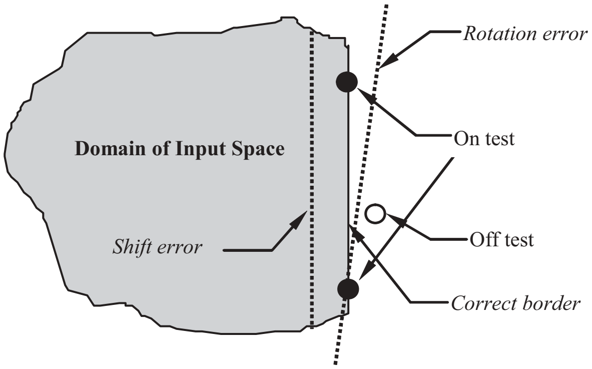


Figure 8. Illustration of boundary shift and rotation errors.

- *Error-based testing techniques*. Error-based testing techniques check all error-prone aspects of the system, where errors are mistakes made software developers. For example, test cases are often selected to test if division by zero error was processed properly by the program.

Among whether the well-established error-based testing techniques are the boundary analysis testing techniques, which select test cases on the boundary and near the boundary of an input space in order to make sure that the programmer has computed correctly the boundary, which has been recognized for a long time as error-prone. As illustrated in Fig. 8, two types of boundary errors have long been recognized as the most common programming errors. They are shift errors, in which the border of an input domain is shifted parallel to the correct border either toward the outside of the input domain or toward the outside of the domain, and rotation errors in which the border is rotated with respect to the correct border.

To detect shift errors of a border in N-dimensional input space, N test cases must be selected on the border and an additional test case must be selected nearby the border. If the input on the border belongs to the input domain, which are called on tests, the test case near the border must be selected outside the input domain, which is

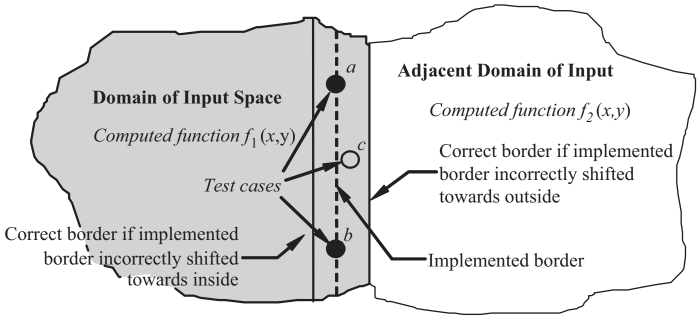

a
Domain of Input Space
Computed function $f_1(x,y)$
Adjacent Domain of Input
Computed function $f_2(x,y)$
c
Correct border if implemented border incorrectly shifted towards outside
Test cases
Correct border if implemented border incorrectly shifted towards inside
b
Implemented border

Figure 9. Selection of test cases using $N \times 1$ criterion.

called off test. Otherwise, if the inputs on the border do not belong to the valid input of the domain, the test case near the border should be selected inside the input domain. In this case, the test cases on the border are off tests, whereas the test case near the border is on test. As illustrated in Fig. 9 for two-dimensional input spaces, by selecting data according to this $N \times 1$ *criterion*, all shift errors can be detected provided that the computed functions in the input domain and outside the domain are different and the border is linear, e.g., a straight line in two dimensional space (49).

However, $N \times 1$ criterion cannot guarantee the detection of rotation errors. To detect rotation errors, in addition to the selection of *N* test cases on the border, N test cases must also be selected near the border in the same way as the $N \times 1$ criterion. It is the so-called $N \times N$ criterion (50).

ACKNOWLEDGMENT

We thank the anonymous reviewers for constructive comments and improvement suggestions.

BIBLIOGRAPHY

1. D. R. Wallace and R. U. Fujii, Software verification and validation: an overview, *IEEE Software*, **6** (3): 10–17, 1989.
2. B. W. Boehm, Software engineering economics, *IEEE Trans. on Software Eng.*, **10** (1) 4–21, 1984.
3. T. Gild, D. Gramham, *Software Inspection*. Reading, MA: Addison-Wesley, 1993.
4. D. A. Wheele, (ed.), *Software Inspection: An Industry Best Practice*, Piscataway, Nj: IEEE Computer Society Press, 1996.
5. E. Yourdon, *Structured Walkthroughs* 4th ed. Englewood Cliffs, Nj: Prentice-Hall International, 1989.
6. L. A. Belady and M. M. Lehman, A model of large program development, *IBM Systems J.*, **3**: 225–252. 1976.
7. M. Newborn, *Automated Theorem Proving: Theory and Practice*, Benlin: Springer, 2000.
8. C.A.R. , Hoare, An axiomatic basis for computer programming, *CACM* **12** (10): 576–583, 1969.
9. C. A. R. , Hoare, et al., Laws of programming, *CACM*, **30**(8): 672–687, 1987.
10. J. A. Robinson, A machine-oriented logic based on the resolution principle, *J. ACM*, **12**(1): 23–41, 1965.
11. J. McCarthy, and P. Hayes, Some philosophical problems from the stanpoint of artificial intelligence, in *Machine Intelligence*, no. 4, B. Meltzer and D. Michie (eds.), Edinburgh: Edinburgh University Press, 1969, 463–502.
12. OE. M. , Clarke, Jr. , Grumberg, and D.A. Peled, *Model Checking*, Cambridges, MA: The MIT Press, 1999.
13. E. M. Clarke, Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Trans. Programming Lang. Syst.*, **8** (2): 244–263, 1986.
14. G. J. Holzmann, The model checker SPIN, *IEEE Trans. Software Engineering*, **23** (5): 1997.
15. T. Henzinger, et al., Symbolic Model Checking for Real-Time Systems *Proceedings, The Seventh Annual IEEE Symposium on Logic in Computer Science*, 1992, pp. 394–406.
16. W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, Model Checking Programs, *Autom. Software Engineer. J.*, **10** (2): 2003.
17. M. B. , Dwyer, J. Hatcliff Robby, C. S. Pasareanu, W. Visser, Formal Software analysis emerging trends in software model checking, *Future Software Engineering*, 2007, pp. 120–136.
18. S. Chandra, P. Godefroid, C. Palm, Software model checking in practice: an industrial case study, *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 431–441.
19. H. Zhu, P. Hall, and J. May, Software unit test coverage and adequacy, *ACM Computing Surveys*, **29** (4): 366–427, 1997.
20. G. Bernot, M. C. Gaudel, and B. Marre, Software testing based on formal specifications: a theory and a tool,*Software Engineering J.*, 387–405, 1991.
21. H. Zhu, A note on test oracles and semantics of algebraic specifications, *Proc. of QSIC'03*, Dallas Tx, 2003, 91–99.
22. H. Y. Chen, T. H. Tse, and T. Y. Chen, TACCLE: a methodology for object-oriented software testing at the class and cluster levels, *ACM TSEM*, **10** (1): 56–109, 2001.
23. T. Y. Chen, T. H. Tse, and Z. Q. Zhou, Fault-based testing without the need of oracles, *Informat. Software Technol.*, **45** (1): 1–9, 2003.
24. L. Bouge, N. Choquet, L. Fribourg, and M. C. Gaudel, Test sets generation from algebraic specifications using logic programming, *J. Systems Software*, **6** (4): 343–360, 1986.
25. P. A. Stocks and D. A. Carrington, Test templates: a specification-based testing framework, *Proc. of ICSE'93*, 1993, pp. 405–414.
26. P. Ammann, and J. Offutt, Using formal methods to derive test frames in category-partition testing, *Proceedings of 9th Annual Conf. on Computer Assurance, IEEE*, Gaithersburg, MD, 1994, pp. 69–79,
27. P. Ammann, P. E. Black, and W. Majurski, Using model checking to generate tests from speci_cations, *Proc. of 2nd IEEE International Conference on Formal Engineering Methods (ICFEM'98)*, Brisbane, Australia, 1998, p. 46.
28. H. S. Hong, S. D. Cha, I. Lee, O. Sokolsky, and H. Ural, Data flow testing as model checking, *Proc. of ICSE'03*, Portland, or, 2003.
29. H. Zhu, L. Jin, D. Diaper, Software requirements validation via task analysis, *J. System Software*, **61** (2): 145–169, 2002.
30. J. Offutt, and A. Abdurazik, Using UML collaboration diagrams for static checking and test generation, *The Third International Conference on the Unified Modeling Language (UML '00)*, York, UK, 2000, pp. 383–395.
31. S. Li, J. Wang and Z. Qi, Property-oriented test generation from UML statecharts, *Proceedings of the 19th International Conference on Automated Software Engineering (ASE'04).*
32. RTCA/DO-178B. Software considerations in airborne systems and equipment certification, 1992.
33. J. Chilenski, An investigation of three forms of the modified condition decision coverage (MCDC) criterion, *Technical Report DOT/FAA/AR-01/18, FAA*, Washington, D. C., 2001.
34. J. B. Goodenough and S. L. Gerhart, Toward a theory of test data selection, *IEEE TSE*, **3**,1975.
35. S. Rapps and E. J. Weyuker, Selecting software test data using data flow information, *IEEE TSE*, **11** (4): 367–375, 1985.
36. P. G. Frankl and J. E. Weyuker, An applicable family of data flow testing criteria, *IEEE TSE*, **14** (10), 1483–1498, 1988.
37. J. Laski and B. Korel, A data flow oriented program testing strategy, *IEEE TSE*, **9**: 33–43, 1983.
38. S. C. Ntafos, On required element testing, *IEEE TSE*, **10** (6): 795–803, 1984.

39. R. A. DeMillo, R. J. Lipton, and F. G. Sayward, Hints on test data selection: help for the practising programmer, *Computer*, **11**, 34–41, 1978.

40. R. G. Hamlet, Testing programs with the aid of a compiler, *IEEE TSE*, **3** (4), 279–290, 1977.

41. T. A. Budd, Mutation analysis: ideas, examples, problems and prospects, in B. Chandrasekaran, and S. Radicchi, (eds.), *Computer Program Testing*, Amsterdam: North-Holland, 1981, 129–148.

42. K. N. King and A. J. Offutt, A FORTRAN language system for mutation-based software testing, *Softw.–Practice Exper.*, **21** (7): 685–718, 1991.

43. M. E. Delamaro, J. C. Maldonado, and A. P. Mathur, Integration testing using interface mutation, in *Proceedings of International Symposium on Software Reliability Engineering (ISSRE '96)*, 1996, 112–121.

44. S. Kim, J. Clark, and J. McDermid, Class mutation: mutation testing for object-oriented programs, *Proc. of Net.Object Days Conference on Object-Oriented Software Systems*, 2000.

45. Y. S. Ma, J. Offutt, and Y. R. Kwon, MuJava: an automated class mutation system, *Software Test. Verificat. Reliab.*, **15** (2): 97–133, 2005.

46. R. A. DeMillo and A. J. Offutt, Experimental results from an automatic test case generator, *ACM Trans. Soft. Engine. Methodol.*, **2** (2): 109–127, 1993.

47. M. R. Woodward, Errors in algebraic specifications and an experimental mutation testing tool, *SEJ*, 1993, pp. 211–224.

48. L. Shan and H. Zhu, Testing software modelling tools using data mutation, *Proc. of ICSE'06-AST'06*, Shanghai, China, 2006, ACM Press, pp. 43–49.

49. L. J. White and E. I. Cohen, A domain strategy for computer program testing, *IEEE TSE*, **6** (3): 247–257, 1980.

50. L. A. Clarke, J. Hassell, and D. J. Richardson, A close look at domain testing, *IEEE TSE*, **8** (4): 380–390, 1982.

DAVID KUNG
University of Texas at Arlington
Arlington, Texas

HONG ZHU
Oxford Brookes University
Oxford, United Kingdom

THE FINITE ELEMENT METHOD

INTRODUCTION

Finite element methods are now widely used to solve structural, fluid, and multiphysics problems numerically (1). The methods are used extensively because engineers and scientists can mathematically model and numerically solve very complex problems. The analyses in engineering are performed to assess designs, and the analyses in the various scientific fields are carried out largely to obtain insight into and ideally to predict natural phenomena. The prediction of how a design will perform and whether and how a natural phenomenon will occur is of much value: Designs can be made safer and more cost effective, while insight into and the prediction of nature can help, for example, to prevent disasters. Thus, the use of the finite element method greatly enriches our lives.

As with many other important scientific developments, it is difficult to give an exact date of the "invention" of the finite element method. Indeed, we could trace back the development of the method to the Greek philosophers and in modern times to physicists, mathematicians, and engineers (see the discussions in Refs. 2 and 3). However, the real impetus for the development of what is now referred to as the finite element method was provided by the need to analyze complex structures in aeronautical engineering and the availability of the electronic computer. Namely, when using the finite element method, large systems of algebraic equations need to be assembled and solved, and the computer provides the necessary means to accomplish this task.

The papers of Argyris and Kelsey (4) and Turner et al. (5) were seminal contributions in the 1950s. The name "finite element method" was coined by R.W. Clough in 1960 (6). The potential of the finite element method for engineering analysis was clearly foreseen, and henceforth the research on finite element methods started to accelerate in various research centers in Europe and in the United States. While significant papers and books were written in the next decade (see for example the references in Refs. 2 and 3), the development and rapidly increasing use of some computer programs clearly contributed in a major way to the acceptance and advancement of the method. Indeed, with no computer programs ever developed, the finite element method would have been a theoretical entity without much attention given to it.

The three finite element programs that had a major impact were ASKA, NASTRAN, and SAP (7–9). The NASTRAN and ASKA programs rapidly became major analysis tools in the aerospace and automotive industries. The SAP programs were largely used in civil and mechanical engineering industries and at universities. The availability and wide use of the source codes of SAP IV and NONSAP with the description of the techniques used therein, see Bathe et al. (9–12), had a seminal effect on the development of new algorithms, many finite element programs, and also finite element theory (3,13).

Today, finite element methods probably are used for the analysis of every major engineering design and probably in every branch of scientific studies. The method is now used primarily through the application of commercial finite element programs (see the section titled, "The Use of the Finite Element Method in Computer-Aided Engineering"). These programs are used on mainframes, workstations and PCs and are employed to solve very complex problems. While the finite element methods were used originally for the analysis of solids and structures, the procedures are now employed also for the analysis of multiphysics problems, including fluid flows with fluid–structure interactions (1–3).

The objective in the following sections is to briefly describe the finite element method and give some references that can be consulted for additional study. The description and references, of course, are by no means exhaustive. For the equations used, the notation of Ref. 3 is employed.

THE FORMULATION OF THE FINITE ELEMENT METHOD

Figures 1 to 3 show typical finite element meshes (or finite element assemblages) modeling some solid, structural, and fluid systems. In each case the finite elements are used to represent the volume of the system. The finite elements are connected at the nodal points located at the corners and along the sides and in the faces of the elements. However, nodal points can also be located within the volume of an element. An important feature is that the finite elements do not overlap geometrically but together fill the complete volume of the solid or fluid.

Considering the analysis of a three-dimensional solid, like the wheel in Fig. 1, the geometry and the displacements of each element (within each element including all element surfaces) are completely described by the geometric positions and displacements of the element nodal points. The element nodal coordinates prior to any displacements are known, of course, and the nodal displacements are the unknowns to be calculated. The basic step is to interpolate the element geometry and the element displacements by using the nodal values. Here the isoparametric element interpolation, a most important development due to Irons (14), is largely used. For an element with q nodal points we use

$$x = \sum_{i=1}^{q} h_i x_i;\ y = \sum_{i=1}^{q} h_i y_i;\ z = \sum_{i=1}^{q} h_i z_i$$

$$u = \sum_{i=1}^{q} h_i u_i;\ v = \sum_{i=1}^{q} h_i v_i;\ w = \sum_{i=1}^{q} h_i w_i \qquad (1)$$

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

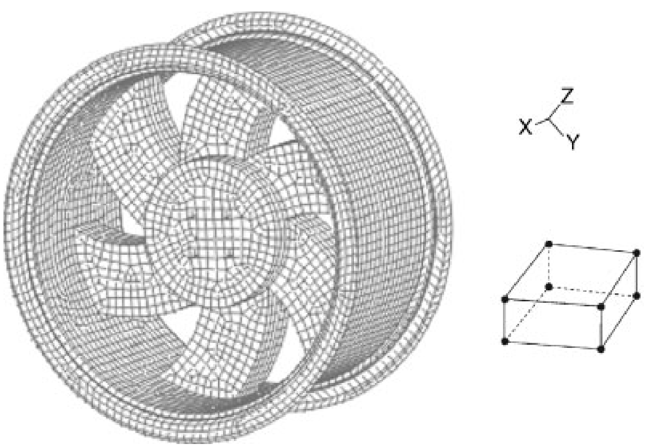


Figure 1. Finite element model of a wheel using three-dimensional brick elements, and a typical 8-node brick element (q=8).

where the h_i are the given interpolation functions, (x_i, y_i, z_i) are the known coordinates of nodal point i, and (u_i, v_i, w_i) are the unknown displacements of nodal point i. The h_i are functions of the element natural coordinates *(r, s, t)*, which relate uniquely to the global coordinates. In this assumption, the same interpolation is employed for the geometry and the displacements, and these equations are applicable to very general curved element configurations. However, in by far most analyses and certainly in nonlinear analyses, low-order linear, non-curved elements are preferred (4-node quadrilateral elements in the two-dimensional (2-D) analyses of solids, and 8-node brick elements in the three-dimensional (3-D) analyses of solids), but quadratic elements (9-node elements in 2-D analyses and 27-node elements in 3-D analyses) can be more effective (3). To obtain a more accurate solution, mostly the number of elements is simply increased; that is, the h-method is used. Alternatively, also the number and size of elements can be kept constant and the order of the geometry and displacement interpolations is increased; that is, the p-method is used. Of course, these approaches can also be combined and then an h/p-method is employed (15).

While some of the first finite elements for structural analyses were largely formulated based on physical insight and intuition, the development and use of a general theory based on the principle of virtual work (or virtual displacements) established a firm foundation and a general approach for the formulation of finite elements.

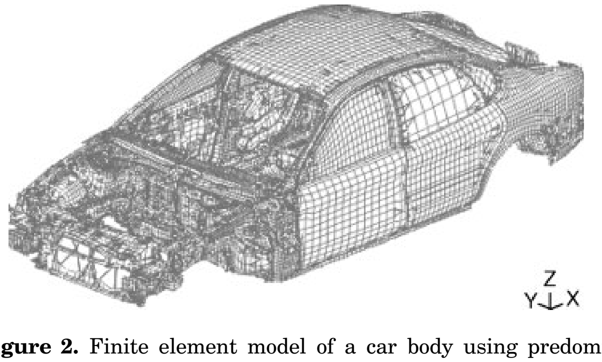


Figure 2. Finite element model of a car body using predominantly shell elements.

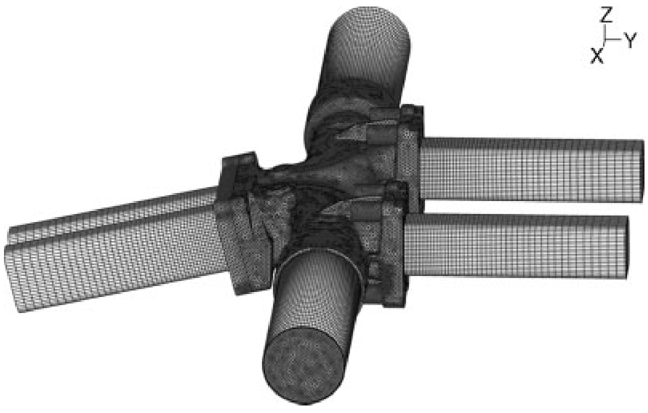


Figure 3. Finite element computational fluid dynamics (CFD) model of a manifold; FCBI elements, about 10 million equations solved in less than 1 hour on a single-processor PC.

Today, the principle of virtual displacements is the basic equation used to formulate displacement-based finite elements. Considering a general solid, this principle can be written as (3)

$$\int_V \bar{\boldsymbol{\varepsilon}}^{\mathrm{T}} \boldsymbol{\tau} dV = \int_{S_f} \bar{\mathbf{u}}^{S_f \mathrm{T}} \mathbf{f}^{S_f} dS + \int_V \bar{\mathbf{u}}^{\mathrm{T}} \mathbf{f}^{B} dV \tag{2}$$

where the unknown (real) stresses are $\boldsymbol{\tau}$, the known applied body forces are $\mathbf{f}^B$, the known applied surface tractions are $\mathbf{f}^{S_f}$, the virtual displacements are $\bar{\mathbf{u}}$, and the virtual strains that correspond to the virtual displacements are $\bar{\boldsymbol{\varepsilon}}$. Of course, the stresses need to satisfy the given stress-strain laws, and the strains (the virtual and real quantities) must satisfy the strain–displacement relationships. Also, the real displacements must satisfy the given displacement boundary conditions and the virtual displacements must be zero at the locations of the prescribed real displacements. The integrations are performed over the volume, V, and traction-loaded surface, S_f, of the solid considered.

The principle of virtual displacements, with these conditions satisfied, is totally equivalent to the differential formulation of the boundary value problem. The principle is referred to also as a variational formulation and a weak formulation (2,3).

In linear analysis, the displacements are assumed to be infinitesimally small, the material laws are assumed to be constant, and the volume and surfaces over which the integrations are performed are known and constant, that is, unaffected by the displacements. Substituting from Equation (1) into Equation (2) and invoking the principle of virtual displacements as many times as there are nodal displacement degrees of freedom, we obtain

$$\mathbf{KU} = \mathbf{R} \tag{3}$$

where $\mathbf{K}$ is the stiffness matrix, $\mathbf{U}$ is the displacement vector listing all unknown nodal point displacements (the u_i, v_i, w_i, for all nodes, i = 1, 2, ..., as applicable), and $\mathbf{R}$ is a vector of externally applied forces at the nodal-point displacement degrees of freedom. The solution of Equation (3) yields the nodal displacements for Equation (1) and hence the strains and stresses in all elements.

In dynamic analysis, the body forces include inertia and damping effects, and these can be included directly in the solution to obtain (3)

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{C}\dot{\mathbf{U}} + \mathbf{K}\mathbf{U} = \mathbf{R} \tag{4}$$

where $\mathbf{M}$ is the mass matrix (usually a consistent mass matrix), $\mathbf{C}$ is a damping matrix (frequently the Rayleigh damping matrix is used), and $\ddot{\mathbf{U}}$, $\dot{\mathbf{U}}$ denote the nodal accelerations and velocities, respectively. In structural vibration analyses, these dynamic equilibrium equations are solved by using mode superposition or implicit direct time integration (for example, by using the popular trapezoidal rule).

However, for wave propagation analyses, the dynamic equilibrium equations solved are generally (3)

$$\mathbf{M}\ddot{\mathbf{U}} = \mathbf{R} - \mathbf{F} \tag{5}$$

where $\mathbf{F}$ represents the nodal forces that correspond to the element stresses. Usually, a lumped (diagonal) mass matrix is used.

The above equations have been written for a three-dimensional solid, but also are applicable with appropriate modifications for the analysis of two-dimensional solids. Furthermore, the equations can be extended and then also are applicable for the analysis of structures modeled by beams, plates, and shells. The extension is reached by introducing midsurfaces, displacements and rotations, and the applicable kinematic and stress assumptions. For example, in the analysis of shells, the Reissner–Mindlin kinematic assumption—that plane sections originally plane and normal to the midsurface of the shell remain plane—is introduced, and the assumption that the stress normal to that midsurface is zero is imposed. Because the pure displacement-based finite elements are not effective for thin shells (and plates and beams), that is, they ‘lock’ in bending actions (see the section below titled, “The Analysis of Shells”), mixed and hybrid formulations are used to reach more effective elements. The construction of such elements in essence is based on an extension of the principle of virtual displacements, namely the use of the Hu–Washizu variational principle (2,3). Effective structural elements can then directly be employed in Equations (3–5) together with the displacement-based solid elements (but of course $\mathbf{U}$ now also includes nodal rotations).

Considering nonlinear analysis, the nonlinear effects can arise from nonlinear material behavior, like plasticity and creep; from geometric nonlinear effects, that is, large displacements and large strains; and from contact between bodies established as a result of the displacements. In these cases, the basic principle of virtual displacements and the extensions for structural elements still are directly applicable, but the appropriate stress and strain measures need be used and the equilibrium need be considered in the deformed geometry of the bodies. For such general nonlinear analyses, the total Lagrangian (TL) and the updated Lagrangian (UL) formulations provide a fundamental basis of solution (3,16) and are widely used. In these formulations, the nonlinear material behavior is introduced to establish the stresses for “given” deformations, that is, not using a rate formulation when the constitutive relations are rate-independent (see also Ref. 17). If required, contact conditions including frictional effects are imposed as constraints on the nodal displacements and forces over the contacting surfaces. Because the large deformations and nonlinear material conditions and the solution for the actual areas of contact introduce significant nonlinearities in the governing equations, iteration for solution in general is required (see below).

Considering a dynamic solution based on implicit direct time integration, and assuming that contact conditions are not present, the incremental nonlinear analysis is accomplished by solving at discrete times Δt apart

$$\mathbf{M}\,{}^{t+\Delta t}\ddot{\mathbf{U}} + \mathbf{C}\,{}^{t+\Delta t}\dot{\mathbf{U}} + {}^{t+\Delta t}\mathbf{F} = {}^{t+\Delta t}\mathbf{R} \tag{6}$$

where the equilibrium is considered at time $t + \Delta t$, and the nodal forces ${}^{t+\Delta t}\mathbf{F}$ and ${}^{t+\Delta t}\mathbf{R}$ correspond to the internal element stresses and the externally applied loads, respectively. In static analysis, simply the inertia and damping effects are neglected and the superscript $t =$ time denotes the load level (but is also an actual physical variable when a material law is time-dependent).

Because the nodal forces highly depend on the deformations and all the considered nonlinearities need be included in calculating these forces, an iteration is necessary for the solution of Equation (6). A Newton–Raphson iteration, or variant thereof, is commonly performed, with the governing equations

$$\begin{aligned} {}^{t+\Delta t}\hat{\mathbf{K}}^{(i-1)}\Delta\mathbf{U}^{(i)} &= {}^{t+\Delta t}\hat{\mathbf{R}}^{(i-1)} - {}^{t+\Delta t}\mathbf{F}^{(i-1)} \\ {}^{t+\Delta t}\mathbf{U}^{(i)} &= {}^{t+\Delta t}\mathbf{U}^{(i-1)} + \Delta\mathbf{U}^{(i)} \end{aligned} \tag{7}$$

where ${}^{t+\Delta t}\hat{\mathbf{K}}^{(i-1)}$ is the effective tangent stiffness matrix that corresponds to the linearization of the nodal force vectors with respect to the incremental displacements $\Delta\mathbf{U}^{(i)}$. The hat in ${}^{t+\Delta t}\hat{\mathbf{K}}^{(i-1)}$ and ${}^{t+\Delta t}\hat{\mathbf{R}}^{(i-1)}$ is used to signify that also mass and damping effects are included (3). In static analysis these effects of course are neglected. The iteration is continued until appropriate convergence criteria (based on forces, displacements, or energy) are satisfied. To reach fast convergence, and quadratic convergence when near the solution, it is important to use the linearization “consistent” with the stress integration procedures and the force updating used to evaluate the right-hand side of Equation (7) (3,17,18).

If contact conditions are to be included, then for example, for normal contact the following conditions need be satisfied (3):

$$ {}^{t+\Delta t}\lambda \geq 0; \quad {}^{t+\Delta t}g \geq 0; \quad {}^{t+\Delta t}\lambda \,.\, {}^{t+\Delta t}g = 0 \tag{8}$$

where ${}^{t+\Delta t}g$ represents the gap and ${}^{t+\Delta t}\lambda$ represents the normal contact force. To reach stable and accurate solutions, appropriate interpolations of contact tractions need be used (19). Of course, with the relations in Equation (8) imposed as conditions on Equation (6), penetration is prevented at the contacting surfaces. Frictional contact conditions are similarly solved for; see Ref. 3. These conditions to enforce contact between bodies enter into the iteration for equilibrium and compatibility; that is, Equation (7) needs to be amended to enforce the contact relations.

The use of Equation (7) (in general amended for contact conditions) for dynamic analysis of course implies that a transient solution with implicit time integration is pursued. If a short time duration or wave propagation analysis is to be performed (e.g., a crash simulation in the motor car industry), then an explicit time integration frequently is more effective. In this case, Equation (5), also amended to include contact conditions, is used with the nodal vectors continuously updated for all nonlinearities, which result from large deformations, contact and nonlinear material effects in the step-by-step solution. The solution is attractive because no iteration is performed like in Equation (7); however, the disadvantage is that the time step size in the forward integration needs to be very small (because the time integration is only conditionally stable) (3).

Equations (7) and (5), as discussed above, are applicable to the nonlinear analysis of solids and structures; however, for structural elements, a mixed formulation need be used (see the section titled, “The Analysis of Shells”). Also, in material nonlinear analysis, frequently (almost) incompressible conditions are encountered, as when modeling a rubber-like material or elasto-plastic conditions. In these cases, considering a solid, it is effective to use a mixed formulation in which the unknown displacements and pressure are interpolated; that is, the u/p formulation is employed (see the section titled, “The Reliable and Effective Analysis of Solids”).

The same basic approach also can be followed to establish finite element solutions of heat transfer problems (considering solids or fluids) and of incompressible fluid flows. In these cases, the temperature and the velocities (and pressure) need to be interpolated; in essence the “principle of virtual temperatures” and the “principle of virtual velocities” are employed. These principles in concept are similar to the principle of virtual displacements and hence of course also correspond to weak formulations. A major added difficulty arises in the use of the Eulerian formulation of fluid flows because of the convective effects. A pure interpolation of velocities and temperature results in a unstable solution. This instability can be circumvented, like in finite difference solutions, by introducing some form of upwinding (see the section titled, “Finite Element Discretizations for Incompressible Fluid Flows”).

The Lagrangian formulations for the solids and structures and the Eulerian formulations for fluid flows have been fully coupled by the use of arbitrary Lagangian–Eulerian formulations, referred to as ALE formulations for fluid–structure interactions. Such formulations are now used to solve reliably very complex multiphysics problems that involve displacements of solids and structures, temperature distributions, fluid flows, electromagnetic effects, and so on (see for example Refs. 1–3, and 20).

EFFECTIVE FINITE ELEMENT PROCEDURES

The first requirement of any analysis is of course the selection of an appropriate mathematical model to represent the physical problem (3). The next requirement is to solve this model with reliable and effective numerical methods. Because the use of reliable and effective finite element methods is very important, much research effort has been expended to reach such procedures.

The Reliable and Effective Analysis of Solids

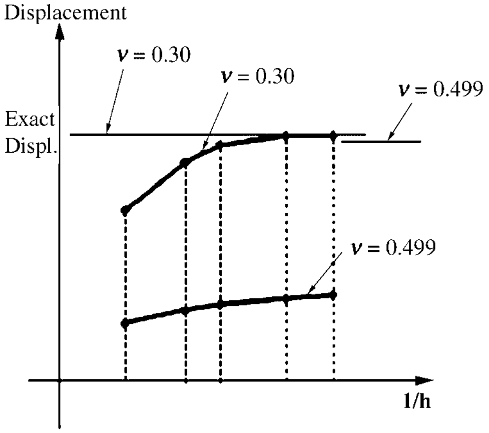
The analysis of solids is efficiently accomplished by using the pure displacement-based finite element method referred to above, unless incompressible or almost incompressible material conditions are considered, as in the analysis of rubber-like materials or in elasto-plastic response. In incompressible analysis, the pure displacement-based discretizations “lock,” and a mixed interpolation based on displacements and pressure is much more effective. However, the appropriate combination of displacement and pressure interpolations must be used.

The phenomenon of locking is of fundamental importance for the understanding of what we mean by a reliable finite element formulation. Figure 4 shows schematically the error in some norm between the finite element solution, $\mathbf{u}_h$, and the exact solution, $\mathbf{u}$, of the mathematical model to be solved, as the mesh is refined. Of course, the exact solution is in general not known, but we can think of a close approximation to the exact solution, obtained by a finite element analysis using a very fine mesh. If for almost incompressible analysis, a pure displacement-based finite element formulation is used, then the convergence curves in Figs. 4(a) and 4(b) are generally much below and above the respective curves obtained in the compressible analysis. Hence the error obtained for a given mesh depends on the bulk modulus and significantly increases as incompressible conditions are approached. Because the displacements become small measured on what they should be, the phenomenon is referred to as “locking”. Finite element discretizations that lock, like schematically referred to in Fig. 4, are not reliable because small changes in some data can cause large changes in the solution error.

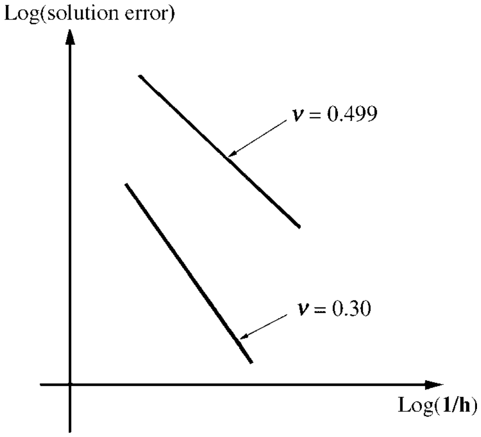
Hence a reliable finite element discretization displays the following convergence behavior in the appropriate norm:

$$\|\mathbf{u} - \mathbf{u}_h\| \simeq c h^k \tag{9}$$

where c is a constant (that is, independent of the bulk modulus), h is the generic element size, and k is the order of interpolation used. To reach such finite element discretizations in almost, or fully, incompressible analysis, the use of the displacement–pressure interpolation is effective where the displacement and pressure interpolations used should satisfy two conditions (3,21–23). First, the ellipticity



(a) Schematic solution curves for displacement when Poisson's ratio ν = 0.30 and ν = 0.499 showing locking; **h** = element size



(b) Schematic error curves (calculated in appropriate norm) showing locking; **h** = element size

Figure 4. Locking phenomenon (when using pure displacement interpolations in almost incompressible analysis).

condition must be satisfied

$$a(\mathbf{v}_h, \mathbf{v}_h) \geq \alpha \|\mathbf{v}_h\|^2 \quad \forall \mathbf{v}_h \in K_h(0) \tag{10}$$

where a (., .) is the (deviatoric strain energy) bilinear form of the problem, $K_h(0)$ represents all those finite element interpolations $\mathbf{v}_h$ over the mesh which satisfy the discretized zero divergence condition, and α is a constant greater than zero. And second, the inf-sup condition should be satisfied for any mesh, and in particular as $h \to 0$,

$$\inf_{q_h \in Q_h} \sup_{\mathbf{v}_h \in V_h} \frac{\int_{\mathrm{Vol}} q_h \mathrm{div}\, \mathbf{v}_h \, d\mathrm{Vol}}{\|\mathbf{v}_h\| \|q_h\|} \geq \beta > 0 \tag{11}$$

where, for the mesh, V_h is the space of displacement interpolations, Q_h is the space of pressure interpolation, appropriate norms are used, and β is a constant. Effective finite elements that satisfy these two conditions are given, for example, in Refs. 3 and 22. It is also possible to develop finite elements that bypass the inf-sup condition and do not lock, but these then depend on nonphysical numerical parameters (22,23).

The Analysis of Shells

The first developments of practical finite element methods were directed toward the analysis of aeronautical structures, that is, thin shell structures. Since these first developments, much further effort has been expended to reach more general, reliable, and effective shell finite element schemes. Shells are difficult to analyze because they exhibit a variety of behaviors and can be very sensitive structures, depending on the curvatures, the thickness t, the span L, the boundary conditions, and the loading applied (23). A shell may carry the loading largely by membrane stresses, largely by bending stresses, or by combined membrane and bending actions. Shells that only carry loading by membrane stresses can be efficiently analyzed by using a displacement-based formulation (referred to above); however, such formulations "lock" (like discussed above when considering incompressible analysis) when used in the analysis of shells subjected to bending. To establish generally reliable and effective shell finite elements (that can be used for any shell analysis problem), the major difficulty has been to overcome the "locking" of discretizations, which can be severe when thin shells are considered.

The principles summarized above for incompressible analysis apply, in essence, also in the analysis of plate and shell structures. The critical parameter is now the thickness to span ratio, t/L, and the relevant inf-sup expression ideally should be independent of this parameter. To obtain effective elements, the mixed interpolation of displacements and strain components has been proposed and is widely used in mixed-interpolated tensorial component (MITC) and related elements (3,23–25). The appropriate interpolations for the displacements and strains have been chosen carefully for the shell elements and are tied at specific element points. The resulting elements then only have displacements and rotations as nodal degrees of freedom, just as for the pure displacement-based elements. The effectiveness of the elements can be tested numerically to see whether the consistency, ellipticity, and inf-sup conditions (23) are satisfied. However, the inf-sup condition for plate and shell elements is much more complex to evaluate than for incompressible analysis (26), and the direct testing by solving appropriately chosen test problems is more straightforward (23,27,28). Some effective mixed-interpolated shell elements are given in Refs. 3,23 and 25.

For plates and shells, instead of imposing the Reissner–Mindlin kinematic assumption and the assumption of "zero stress normal to the midsurface," also three-dimensional solid elements without these assumptions can be used (29–31). Of course, for these elements to be effective, they also need to be formulated to avoid "locking;" that is, they also need to satisfy the conditions mentioned above and more (23).

Finite Element Discretizations for Incompressible Fluid Flows

Considering *all* fluid flow problems, in engineering practice most problems are solved by using finite volume and finite difference methods. Various CFD computer programs based on finite volume methods are in wide use for high-speed compressible and incompressible fluid flows. However, much research effort has been expended on the development of finite element methods, in particular for incompressible flows. Considering such flows and an Eulerian formulation, stable finite element procedures need to use velocity and pressure interpolations like those used in the analysis of incompressible solids (but of course velocities are interpolated instead of displacements) and also need to circumvent any instability that arises in the discretization of the convective terms. This requirement is usually achieved by using some form of upwinding (see for example Refs. 3, 32 and the references therein). However, another difficulty is that the traditional finite element formulations do not satisfy "local" mass and momentum conservation. Because numerical solutions of incompressible fluid flows in engineering practice should satisfy conservation locally, the usual finite element methods have been extended (see for example Ref. 33).

One simple approach that meets these requirements is given by the flow-condition-based interpolation (FCBI) formulation, in which finite element velocity and pressure interpolations are used to satisfy the inf-sup condition for incompressible analysis, flow-condition-dependent interpolations are used to reach stability in the convective terms, and control volumes are employed for integrations, like in finite volume methods (34). Hence stability is reached by the use of appropriate velocity and pressure interpolations, the conservation laws are satisfied locally, and, also, the given interpolations can be used to establish consistent Jacobian matrices for the Newton–Raphson type iterations to solve the governing algebraic equations (which correspond to the nodal conditions to be satisfied).

An important point is that the FCBI schemes of fluid flow solution can be used directly to solve "fully coupled" fluid flows with structural interactions (20,35). The coupling of arbitrary discretizations of structures and fluids in which the structures might undergo large deformations is achieved by satisfying the applicable fluid–structure interface force and displacement conditions (20). The complete set of interface relations is included in the governing nodal point equations to be solved. Depending on the problem and in particular the number of unknown nodal point variables, it may be most effective to solve the governing equations by using partitioning (36). However, once the iterations have converged (to a reasonable tolerance), the solution of the problem has been obtained irrespective of whether partitioning of the coefficient matrix has been used.

While the solution of fluid–structure interactions is encountered in many applications, the analysis of even more general and complex multiphysics problems including thermo-mechanical, electromagnetic, and chemical effects is also being pursued, and the same fundamental principles apply (1).

Solution of Algebraic Equations

The finite element analysis of complex systems usually requires the solution of a large number of algebraic equations; to accomplish this solution effectively is an important requirement.

Consider that no parallel processing is used. In static analysis, "direct sparse solvers" based on Gauss elimination are effective up to about one half of a million equations for three-dimensional solid models and up to about 3 million equations for shell models. The essence of these solvers is that first graph theory is used to identify an optimal sequence to eliminate variables and then the actual Gauss elimination (that is, the factorization of the stiffness matrix and solution of the unknown nodal variables) is performed.

For larger systems, iterative solvers possibly combined with a direct sparse solver become effective, and here in particular an algebraic multigrid solver is attractive. Multigrid solvers can be very efficient in the solution of structural equations but frequently are embedded in particular structural idealizations only (like for the analysis of plates). An "algebraic" multigrid solver in principle can be used for any structural idealization because it operates directly on the given coefficient matrix (37). Figure 5 shows a model solved by using an iterative scheme and gives a typical solution time.

Considering transient analysis, it is necessary to distinguish between vibration analyses and wave propagation solutions. For the linear analysis of vibration problems, mode superposition is commonly performed (3,38). In such cases, frequencies and mode shapes of the finite element models need be computed, and this is commonly achieved using the Bathe subspace iteration method or the Lanczos method (3).

For the nonlinear analysis of vibration problems, usually a step-by-step direct time integration solution is performed with an implicit integration technique, and frequently the trapezoidal rule is used (3). However, when large deformation problems and relatively long time durations are considered, the scheme in Ref. 39 can be much

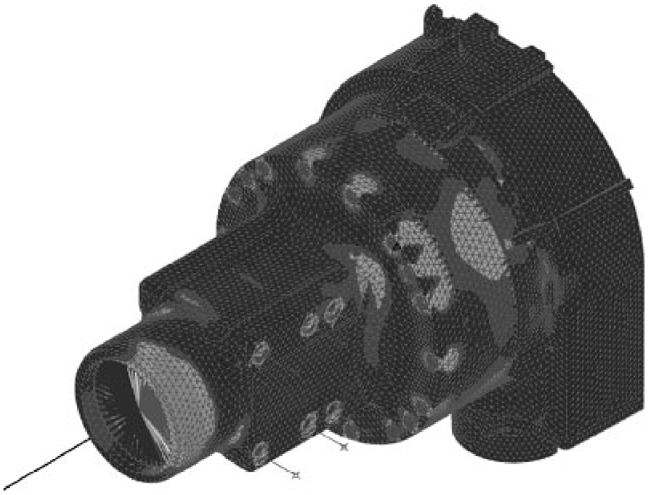


Figure 5. Model of a rear axle; about a quarter of a million elements, including contact solved in about 20 minutes on a single-processor PC.

more effective. The solution of a finite element model of one half of a million equations solved with a few hundred time steps would be considered a large problem.

Of course, the explicit solution procedures already mentioned above are used for short duration transient and wave propagation analyses (3,40).

In the simulations of fluid flows, the number of nodal unknowns usually is very large, and iterative methods need to be used for solution. Here algebraic multigrid solvers are very effective, but important requirements are that both the computation time and amount of memory used should increase about linearly with the number of nodal unknowns to be solved for. Figure 3 gives a typical solution time for a Navier–Stokes fluid flow problem. It is seen that with rather moderate hardware capabilities large systems can be solved.

Of course, these solution times are given merely to indicate some current state-of-the-art capabilities, and have been obtained using ADINA (41). Naturally, the solution times would be much smaller if parallel processing were used (and then would depend on the number of processors used, etc.) and surely will be much reduced over the years to come.

The given observations hold also of course for the solution of multiphysics problems.

MESHING

A finite element analysis of any physical problem requires that a mesh of finite elements be generated. Because the generation of finite element meshes is a fundamental step and can require significant human and computational effort, procedures have been developed and implemented that automatize the mesh generation without human intervention as much as possible.

Some basic problems in automatic mesh generation are (1) that the given geometries can be complex with very small features (small holes, chamfers, etc.) embedded in otherwise rather large geometric domains, (2) that the use of certain element types is much preferable over other element types (for example, brick elements are more effective than tetrahedral elements), (3) that graded meshes need be used for effective solutions (that is, the mesh should be finer in regions of stress concentrations and in boundary layers in fluid flows or the analysis of shells), and (4) that an anisotropic mesh may be required. In addition, any valuable mesh generation technique in a general purpose analysis environment (like used in CAE solution packages, see the section titled, “The Use of the Finite Element Method in Computer-Aided Engineering”) must be able to mesh complex and general domains.

The accuracy of the finite element analysis results, measured on the exact solution of the mathematical model, highly depends on the use of an appropriate mesh, and this holds true in particular when coarse meshes need be used to reduce the computer time employed for complex analyses. Hence, effective mesh generation procedures are most important.

Various mesh generation techniques are in use (42). Generally, these techniques can be classified into mapped meshing procedures, in which the user defines and controls the element spacings to obtain a relatively structured mesh, and free-form meshing procedures, in which the user defines the minimum and maximum sizes of elements in certain regions but mostly has little control as to what mesh is generated, and the user obtains an unstructured mesh. Of course, in each case the user also defines for what elements the mesh is to be generated. Mapped meshing techniques in general can be used only for rather regular structural and fluid domains and require some human effort to prepare the input but usually result in effective meshes, in the sense that the accuracy of solution is high for the number of elements used. The free-form meshing techniques in principle can mesh automatically any 3-D domain provided tetrahedral elements are used; however, a rather unstructured mesh that contains many elements may be reached. The challenge in the development of free-form meshing procedures has been to reach meshes that in general do not contain highly distorted elements (long, thin sliver elements must be avoided unless mesh anisotropy is needed), that do not contain too many elements, and that contain brick elements rather than tetrahedral elements. Two fundamental approaches have been pursued and refined, namely methods based on advancing front methodologies that generate elements from the boundary inwards and methods based on Delaunay triangularizations that directly mesh from coarse to fine over the complete domain. Although a large effort has already been expended on the development of effective mesh generation schemes, improvements still are much desired, for example to reach more general and effective procedures to mesh arbitrary three-dimensional geometries with brick elements.

Figure 1 shows a three-dimensional mapped mesh of brick elements, a largely structured mesh, for the analysis of a wheel, and Fig. 6 shows a three-dimensional mesh of tetrahedral elements, an unstructured mesh, for the analysis of a helmet. It is important to be able to achieve the grading in elements shown in Fig. 6 because the potential area of contact on the helmet requires a fine mesh.

Figure 7 shows another important meshing feature for finite element analysis, namely the possibility to glue in a “consistent manner” totally different meshes together (19). This feature provides flexibility in meshing different parts and allows multiscale analysis. The glueing of course is applicable in all linear and nonlinear analyses.

Because the effective meshing of complex domains still is requiring significant human and computational effort, some new discretization methods that do not require a mesh in the traditional sense have been developed (43). These techniques are referred to as meshless or meshfree methods but of course still require nodal points, with nodal point variables that are used to interpolate solid displacements, fluid velocities, temperatures, or any other continuum variable. The major difference from traditional finite element methods is that in meshfree methods, the discrete domains, over which the interpolations take place, usually overlap, whereas in the traditional finite element methods, the finite element discrete domains abut each other, and geometric overlapping is not allowed. These meshfree

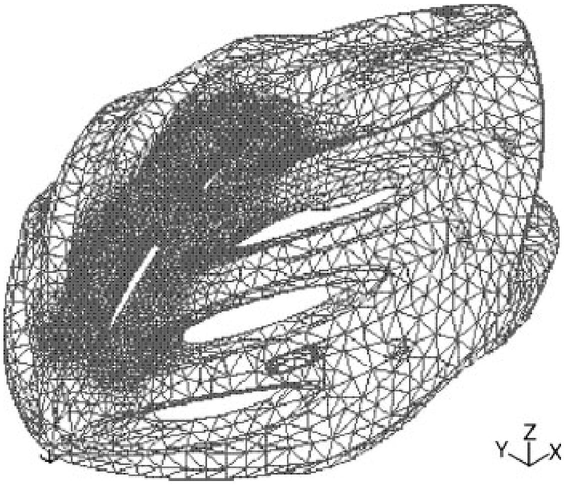


Figure 6. Model of a bicycle helmet showing mesh gradation.

methods require appropriate nodal point spacing, and the numerical integration is more expensive (43,44). However, the use of these procedures coupled with traditional finite elements shows promise in regions where either traditional finite elements are difficult to generate or such elements become highly distorted in geometric nonlinear analysis (43).

THE USE OF THE FINITE ELEMENT METHOD IN COMPUTER-AIDED ENGINEERING

The finite element method would not have become a successful tool to solve complex physical problems if the method had not been implemented in computer programs. The success of the method is clearly due to the effective implementation in computer programs and the possible wide use of these programs in many industries and research environments. Hence, reliable and effective finite element methods were needed that could be trusted to perform well in general analysis conditions as mentioned above. But also, the computer programs had to be easy to use. Indeed, the ease of use is very important for a finite element program to be used in engineering environments.

The ease of use of a finite element program is dependent on the availability of effective pre- and post-processing tools based on graphical user interfaces for input and output of data. The pre-processing embodies the use of geometries from computer-aided design (CAD) programs or the construction of geometries with CAD-like tools, and the automatic generation of elements, nodal data, boundary conditions, material data, and applied loadings. An important ingredient is the display of the geometry and constructed finite element model with the elements, nodes, and so on. The post-processing is used to list and graphically display the computed results, such as the displacements, velocities, stresses, forces, and so on. In the post-processing phase, the computed results usually are looked at first to see whether the results make sense (because, for example, by an input error, a different than intended finite element model may have been solved), and then the results are studied in depth. In particular, the results should give the answers to the questions that provided the stimulus for performing the finite element analysis in the first instance.

The use of finite element programs in computer-aided engineering (CAE) frequently requires that geometries from CAD programs need be employed. Hence, interfaces to import these geometries into the finite element pre-processing programs are important. However, frequently the computer-aided design geometry can not be used directly to generate finite element meshes and needs to be changed ("cleaned-up") to ensure well-connected domains, and by deleting unnecessary details. The effective importing of computer-aided design data is a most important step for the wide use of finite element analysis in engineering design.

As mentioned already, the purpose of a finite element analysis is to solve a mathematical model. Hence, in the post-processing phase, the user of a finite element program ideally would be able to ascertain that a sufficiently accurate solution has been obtained. This means that, ideally, a measure of the error between the computed solution and the "exact solution of the mathematical model" would be available. Because the exact solution is unknown, only estimates can be given. Some procedures for estimating the error are available (mostly in linear analysis) but need to be used with care (45,46), and in practice frequently the analysis is performed simply with a very fine mesh to ensure that a sufficiently accurate solution has been obtained. Figure 8 shows an example of error estimation. Here, the scheme proposed by Sussman and Bathe (47) is used in a linear, nonlinear, and FSI solution using ADINA.

In the first decades of finite element analysis, many finite element programs were available and used. However, the ensuing strive for improved procedures in these programs, in terms of generality, effectiveness, and ease of use, required a significant continuous development effort. The resulting competition in the field to further develop and

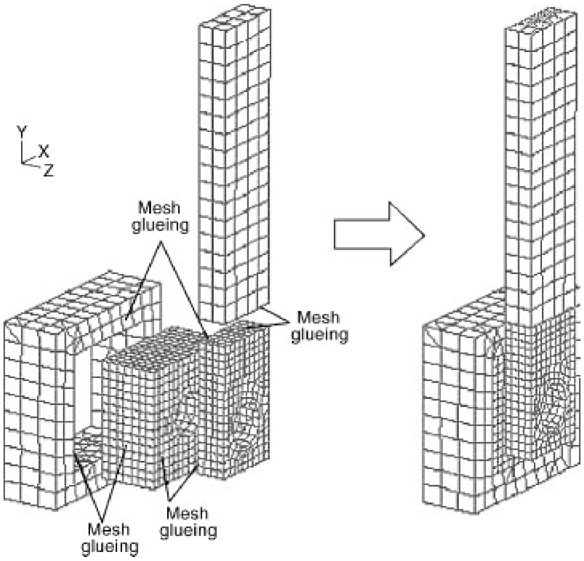


Figure 7. Glueing of dissimilar meshes.

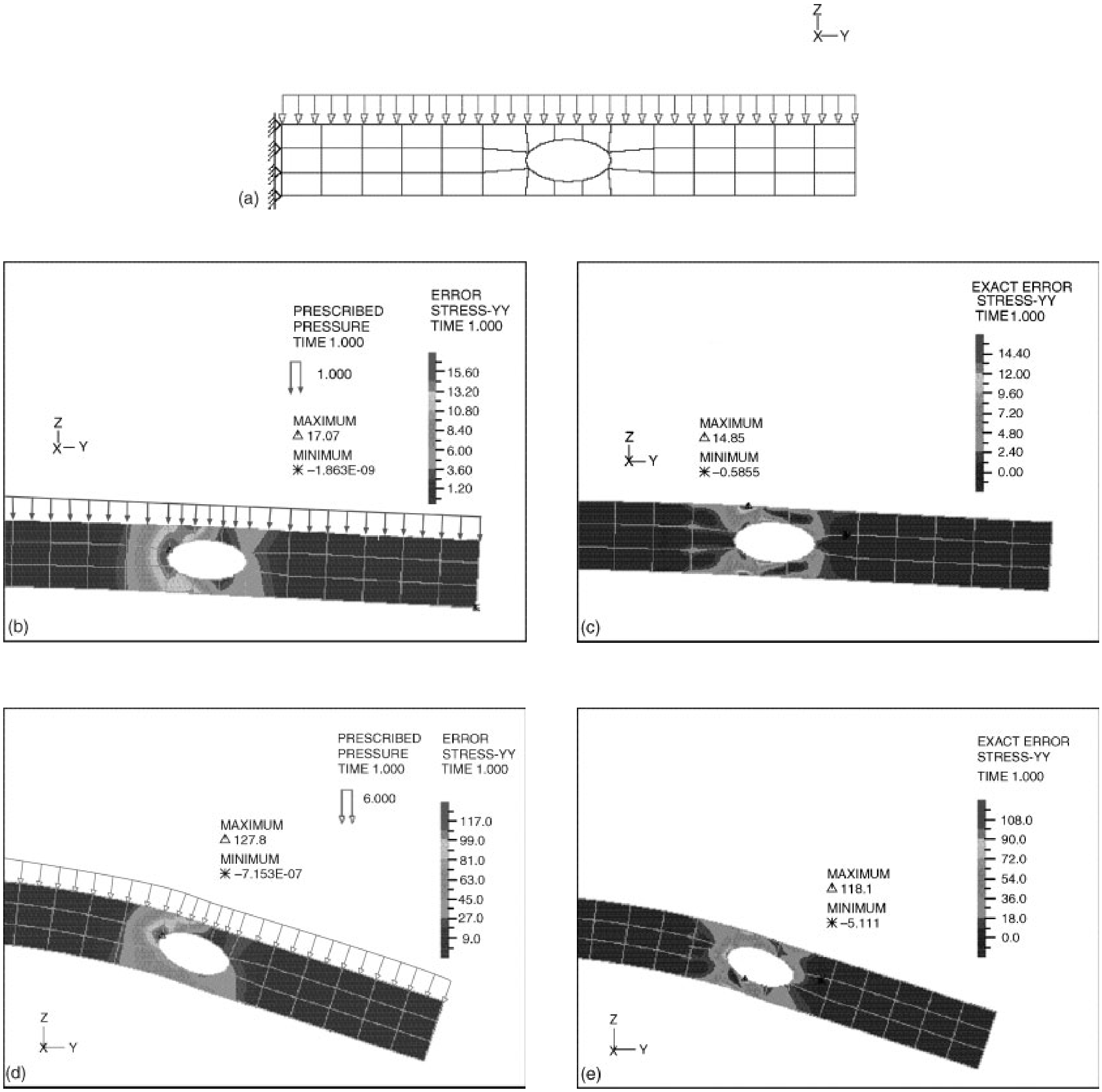


Figure 8. Error estimation example: analysis of a cantilever with a hole. (a) Mesh of 9-node elements used for analysis of cantilever (with boundary conditions). (b) Cantilever structure — linear analysis, estimated error in region of interest shown. (c) Cantilever structure — linear analysis, exact error in region of interest shown. (d) Cantilever structure — nonlinear analysis, estimated error in region of interest shown. (e) Cantilever structure — nonlinear analysis, exact error in region of interest shown. (f) Cantilever structure — FSI analysis, estimated error in region of interest shown. (g) Cantilever structure — FSI analysis, exact error in region of interest shown.

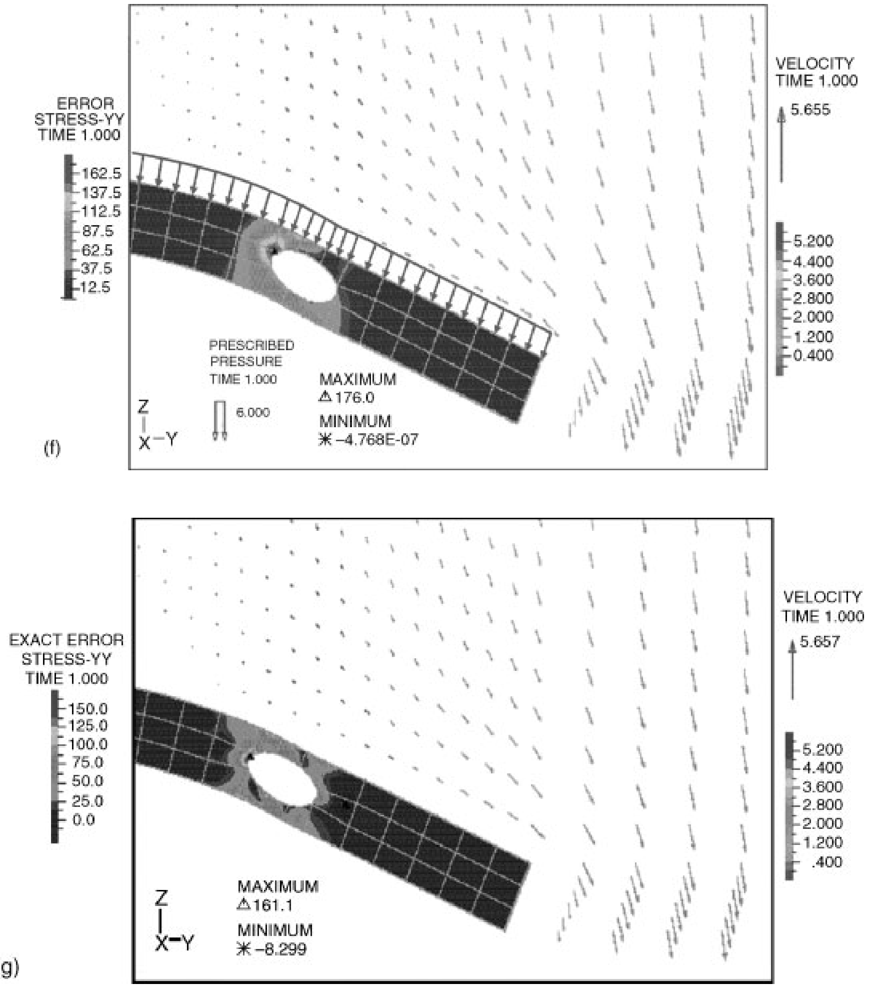


Figure 8. (*Continued*)

continuously support a finite element program reduced the number of finite element systems now widely used to only a few: MSC NASTRAN, NX NASTRAN, ANSYS, ABAQUS, ADINA, and MARC are the primary codes used for static and implicit dynamic analyses of structures, and LS-DYNA, RADIOSS, PAMCRASH are the primary codes used for explicit dynamic analyses. For multiphysics problems, notably fluid–structure interactions, primarily ADINA and ANSYS are used.

CONCLUSIONS AND KEY CHALLENGES

Since the first publications on the practical use of the finite element method, the field of finite element analysis has exploded in terms of research activities and applications. Numerous references are given on the World Wide Web. The method is now applied in practically all areas of engineering and scientific analyses. Since some time, the finite element method and related techniques frequently are simply referred to as "methods in computational fluid and solid mechanics."

With all these achievements in place and the abundant applications of computational mechanics today, it is surely of interest to ask, "What are the outstanding research and development tasks? What are the key challenges still in the field?" These questions are addressed in the prefaces of Ref. 1, see the 2003 and 2005 volumes.

Although much has been accomplished, still, many exciting research tasks exist in further developing finite element methods. These developments and the envisaged increased use of finite element methods not only will have a continuous and very beneficial impact on traditional engineering endeavors but also will lead to great benefits in other areas such as in the medical and health sciences.

Specifically, the additional developments of finite element methods should not be directed only to the more effective analysis of single media but also must focus on the solution of multiphysics problems that involve fluids, solids, their interactions, and chemical and electromagnetic effects from the molecular to the macroscopic scales, including uncertainties in the given data, and should also be directed to reach more effective algorithms for the optimization of designs.

Based on these thoughts, we can identify at least eight key challenges for research and development in finite element methods and numerical methods in general; these challenges pertain to the more automatic solution of mathematical models, to more effective and predictive numerical schemes for fluid flows, to mesh-free methods that are coupled with traditional finite element methods, to finite element methods for multiphysics problems and multiscale problems, to the more direct modeling of uncertainties in analysis data, to the analysis and optimization of complete lifecycles of designed structures and fluid systems, and finally, to providing effective education and educational tools for engineers and scientists to use the given analysis procedures in finite element programs correctly and to the full capabilities (1).

Hence, although the finite element method already is widely used, still, many exciting research and development efforts exist and will continue to exist for many years.

BIBLIOGRAPHY

1. K. J. Bathe, ed., *Computational Fluid and Solid Mechanics*. New York: Elsevier, 2001, 2003, 2005, 2007.
2. O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, Vols. 1 and 2, 4th ed., New York: McGraw Hill, 1989, 1990.
3. K. J. Bathe, *Finite Element Procedures*. Englewood Cliffs, NJ: Prentice Hall, 1996.
4. J. H. Argyris and S. Kelsey, Energy theorems and structural analysis, *Aircraft Engrg.*, Vols. 26 and 27, Oct. 1954 to May 1955.
5. M. J. Turner, R. W. Clough, H. C. Martin, and L. J. Topp, Stiffness and deflection analysis of complex structures, *J. Aeronaut. Sci.*, **23**: 805–823, 1956.
6. R. W. Clough, The finite element method in plane stress analysis, *Proc. 2nd ASCE Conference on Electronic Computation*, Pittsburgh, PA, Sept. 1960, pp. 345–378.
7. J. H. Argyris, Continua and discontinua, *Proc. Conference on Matrix Methods in Structural Mechanics*, Wright-Patterson A.F.B., Ohio, Oct. 1965, pp. 11–189.
8. R. H. MacNeal, A short history of NASTRAN, *Finite Element News*, July 1979.
9. K. J. Bathe, E. L. Wilson, and F. E. Peterson, SAP IV —A Structural Analysis Program for Static and Dynamic Response of Linear Systems, Earthquake Engineering Research Center Report No. 73–11, University of California, Berkeley, June 1973, revised April 1974.
10. K. J. Bathe, E. L. Wilson, and R. Iding, NONSAP—A Structural Analysis Program for Static and Dynamic Response of Nonlinear Systems, Report UCSESM 74–3, Department of Civil Engineering, University of California, Berkeley, May 1974.
11. K. J. Bathe, H. Ozdemir, and E. L. Wilson, Static and Dynamic Geometric and Material Nonlinear Analysis, Report UCSESM 74-4, Department of Civil Engineering, University of California, Berkeley, May 1974.
12. K. J. Bathe and E. L. Wilson, *Numerical Methods in Finite Element Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1976.
13. J. Mackerle, FEM and BEM in the context of information retrieval, *Comput. Struct.*, **80**: 1595–1604, 2002.
14. B. M. Irons, Engineering applications of numerical integration in stiffness methods, *AIAA J.*, **4**: 2035–2037, 1966.
15. I. Babuška and T. Strouboulis, *The Finite Element Method and its Reliability*. Oxford, UK: Oxford Press, 2001.
16. K. J. Bathe, E. Ramm, and E. L. Wilson, Finite element formulations for large deformation dynamic analysis, *Int. J. Numer. Methods Eng.,* **9**: 353–386, 1975.
17. F. J. Montáns and K. J. Bathe, Computational issues in large strain elasto-plasticity: an algorithm for mixed hardening and plastic spin, *Int. J. Numer. Methods Eng.*, **63**: 159–196, 2005.
18. M. Kojić and K. J. Bathe, *Inelastic Analysis of Solids and Structures*. Berlin: Springer, 2005.
19. N. El-Abbasi and K. J. Bathe, Stability and patch test performance of contact discretizations and a new solution algorithm, *Comput. Struct.*, **79**: 1473–1486, 2001.
20. K. J. Bathe and H. Zhang, Finite element developments for general fluid flows with structural interactions, *Int. J. Numer. Methods Eng.*, **60**: 213–232, 2004.
21. F. Brezzi and K. J. Bathe, A discourse on the stability conditions for mixed finite element formulations, *J. Comput. Methods Appl. Mechanics Eng.*, **82**: 27–57, 1990.
22. F. Brezzi and M. Fortin, *Mixed and Hybrid Finite Element Methods*. Berlin: Springer, 1991.
23. D. Chapelle and K. J. Bathe, *The Finite Element Analysis of Shells – Fundamentals*. Berlin: Springer, 2003.
24. K. J. Bathe and E. N. Dvorkin, A formulation of general shell elements — The use of mixed interpolation of tensorial components, *Int. J. Numer. Methods Eng.*, **22**: 697–722, 1986.
25. K. J. Bathe, A. Iosilevich and D. Chapelle, An evaluation of the MITC shell elements, *Comput. Struct.*, **75**: 1–30, 2000.
26. K. J. Bathe, The inf-sup condition and its evaluation for mixed finite element methods, *Comput. Struct.*, **79**: 243–252, 971, 2001.
27. D. Chapelle and K. J. Bathe, Fundamental considerations for the finite element analysis of shell structures, *Comput. Struct.*, **66**: no. 1, 19–36, 1998.
28. J. F. Hiller and K. J. Bathe, Measuring convergence of mixed finite element discretizations: An application to shell structures, *Comput. Struct.*, **81**: 639–654, 2003.
29. K. J. Bathe and E. L. Wilson, Thick shells, in *Structural Mechanics Computer Programs*, W. Pilkey, K. Saczalski and H. Schaeffer, eds. Charlottesville: The University Press of Virginia, 1974.
30. M. Bischoff and E. Ramm, On the physical significance of higher order kinematic and static variables in a three-dimensional shell formulation, *Int. J. Solids Struct.*, **37**: 6933–6960, 2000.
31. D. Chapelle, A. Ferent and K. J. Bathe, 3D-shell elements and their underlying mathematical model, *Math. Models & Methods Appl. Sci.*, **14**: 105–142, 2004.
32. J. Iannelli, *Characteristics Finite Element Methods in Computational Fluid Dynamics*. Berlin: Springer Verlag, 2006.
33. T. J. R. Hughes and G.N. Wells, Conservation properties for the Galerkin and stabilised forms of the advection-diffusion and incompressible Navier-Stokes equations, *J. Comput. Methods Appl. Mech. Eng.*, **194**: 1141–1159, 2005, and Correction **195**: 1277–1278, 2006.
34. K. J. Bathe and H. Zhang, A flow-condition-based interpolation finite element procedure for incompressible fluid flows, *Comput. Struct.*, **80**: 1267–1277, 2002.
35. K. J. Bathe and G. A. Ledezma, Benchmark problems for incompressible fluid flows with structural interactions, *Comput. Struct.*, **85**: 628–644, 2007.
36. S. Rugonyi and K. J. Bathe, On the finite element analysis of fluid flows fully coupled with structural interactions, *Comput. Model. Eng. Sci.*, **2**: 195–212, 2001.

37. K. Stuben, A review of algebraic multigrid, *J. Computat. Appl. Math.*, **128**(1–2): 281–309, 2001.

38. J. W. Tedesco, W. G. McDougal, and C. A. Ross, *Structural Dynamics*. Reading, MA: Addison-Wesley, 1999.

39. K. J. Bathe, Conserving energy and momentum in nonlinear dynamics: a simple implicit time integration scheme, *Comput. Struct.*, **85**: 437–445, 2007.

40. T. Belytschko and T.J.R. Hughes (eds), *Computational Methods for Transient Analysis*. New York: North Holland, 1983.

41. K. J. Bathe, ADINA System, *Encycl. Math.*, **11**: 33–35, 1997; see also: http:// www.adina.com.

42. B.H.V. Topping, J. Muylle, P. Iványi, R. Putanowicz, and B. Cheng, *Finite Element Mesh Generation*. Scotland: Saxe-Coburg Publications, 2004.

43. S. Idelsohn, S. De, and J. Orkisz, eds. *Advances in meshfree methods, Special issue of Comput. Struct.*, **83**, no. 17–18, 2005.

44. S. De and K. J. Bathe, Towards an efficient meshless computational technique: the method of finite spheres, *Eng. Computat.*, **18**: 170–192, 2001.

45. M. Ainsworth and J. T. Oden, *A Posteriori Error Estimation in Finite Element Analysis*. New York: Wiley, 2000.

46. T. Grätsch and K. J. Bathe, *A* posteriori error estimation techniques in practical finite element analysis, *Comput. Struct.*,**83**: 235–265, 2005.

47. T. Sussman and K. J. Bathe, Studies of finite element procedures — on mesh selection, *Comput. Struct.*, **21**: 257–264, 1985.

KLAUS-JÜRGEN BATHE
Massachusetts Institute of Technology
Cambridge, Massachusetts

ABSTRACT

The objective in this article is to give an overview of finite element methods that currently are used extensively in academia and industry. The method is described in general terms, the basic formulation is presented, and some issues regarding effective finite element procedures are summarized. Various applications are given briefly to illustrate the current use of the method. Finally, the article concludes with key challenges for the additional development of the method.

UNIFIED MODELING LANGUAGE (UML)

UML is a graphically oriented computer language used to *represent* software programs and related phenomena in a manner that allows its users to focus on the essential aspects of their software without being distracted by syntactic and technology-specific details found in traditional programming languages. In effect, a UML specification of a program is an example of an engineering model, that is, a reduced representation of an existing or planned design constructed to facilitate the assessment of key characteristics of that design. Given the complexity of modern software systems, the use of models and modeling as a complexity reduction technique is both obvious and necessary.

Since its adoption as a standard technology by the Object Management Group (OMG) in 1996 (1), UML has been widely adopted by software developers and researchers and is supported by numerous tool vendors. It is also an integral part of many computer science and software engineering curricula throughout the world. It is probably the most widespread software modeling language and has served as the foundation or inspiration for numerous domain-specific modeling languages and standards.

THE HISTORY OF THE DEVELOPMENT OF UML

The idea of modeling software dates back at least as far as classic flowcharts, which were used to depict algorithmic flow in an intuitive graphical form that captured the essence of the algorithm without the encumbrance of irrelevant detail. Since those early days, many other languages were proposed for modeling software systems. In particular, during the late 1980s and early 1990s, there was an explosion of new modeling languages and notations, stimulated by the resurgence of interest in object-oriented languages and methods (2). One of the characteristics of these technologies was that software based on them typically involved complex structural relationships (often mimicking structural relationships in the physical world), which became very difficult to discern in the text-based linear forms characteristic of traditional programming languages. Unfortunately, this diversity led to major fragmentation in terms of expertise and tooling. Users who wanted to take advantage of such methods found that they were forced to make a choice that would invariably lead them into isolation and an undesirable lock-in to those tools and vendors that supported their particular selection.

This was the setting in which UML was defined, first as a merger of two of the most widely known and most popular software modeling notations of the time—the Booch OO method (3) and the Object Modeling Technique (4) devised by Rumbaugh et al. . These were joined later by the Object-Oriented Software Engineering method and notation of Jacobson et al. (5) As this joint notation was being defined, it was also proposed as an industry standard, in response to a request for proposal issued in 1995 by the OMG, a representative consortium of software product vendors and users. At this point, other experts joined the original UML team of Jim Rumbaugh, Grady Booch, and Ivar Jacobson and the first version of standardized version of UML, UML 1.1, was adopted by the OMG in December 1996.

After the initial adoption, several lesser revisions of the language were produced. With two exceptions, these revisions did not add significant new features, comprising primarily minor fixes and clarifications. The first major addition, the concept of profiles, was introduced with UML version 1.4. Profiles provided a more structured facility for defining domain-specific variations of UML. The second major innovation was the introduction of a model of actions—a more precise definition of the run-time semantics of UML and a corresponding action language. The latter identified a set of basic primitive instructions for creating and manipulating run-time artifacts, such as objects and links, as well as instructions for inter-object communications. (Initially, the actions model was defined as a supplementary specification to the overall standard but has eventually integrated into the standard in version 1.5.)

The addition of the actions model to UML was caused by the growing pressure to extend it beyond being just a relatively informal documentation facility for supporting analysis and design activities. Instead—stimulated by experiences with several successful commercial products—there was a strong motivation to evolve UML into a fully fledged computer language that could be used to specify and even implement software systems. This style of software development in which high-level modeling languages play a primary role in analysis, design, and in some cases, implementation (as opposed to an optional support role) is often called *model-driven development (MDD).* The central idea behind MDD is that, because modeling languages are free of attention-diverting implementation-specific detail, they are better suited to coping with the complexity of modern software design than most common programming languages. Furthermore, if such languages are supported by powerful computer-based tools, which can unburden software developers from having to perform various time-consuming and mechanistic activities (such as translating the modeling specifications into corresponding programming language implementations), then MDD has the potential to improve significantly both the quality of software and the productivity of developers. In response to the growing interest and successes of MDD, the OMG defined a vision and a plan for a series of industry standards in support of MDD, which it named *Model-Driven Architecture (MDA)* (6).

For UML to be an effective MDD tool, it was necessary to provide a much tighter and more extensive specification of its semantics and its syntax. This, along with strong

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

pressure to add some new modeling capabilities, led to a major revision of UML, UML 2.0, which was adopted by the OMG in 2004 (7–11). Several further minor revisions of this standard were defined in subsequent years to fix lesser technical issues and inconsistencies and to add clarifications.

Distinguishing Characteristics of UML

As noted above, software modeling languages have a long history, starting with flowcharts, through to so-called structured analysis and structured design languages (inspired by the principles of structured programming), and on to a variety of object-oriented modeling languages and notations. UML builds on this tradition, reusing many of the proven ideas and methods of its predecessors. However, it is worth noting that modeling language design is still in its infancy, with no systematic design processes defined that are based on well-understood scientific and engineering principles. In other words, unlike programming languages, a sound and complete theory of modeling language construction has yet to emerge.

The principal features of UML 2 that drove its design are the following:

1. Object orientation
2. Visual concrete syntax
3. Separation of views
4. Single underlying model
5. Customizability

Object Orientation. UML emerged at the time of heightened interest in the object paradigm and object-oriented programming languages. Consequently, it was very much influenced by the dominant object-oriented programming languages of the time, in particular C++ and Smalltalk-80. It incorporates many, if not most, of the primary concepts and terminology of those languages, such as the notions of class, inheritance, encapsulation, polymorphism, and the like. Although it is possible to model non-object-oriented systems with UML, the underlying conceptual foundations are still based on the object paradigm. For example, the semantics of UML require that all run-time behaviors, regardless of whether they are in the form of state machines, activities, or actions, are the consequence of the actions of objects, even when such objects are not explicit in the user model. Furthermore, in UML 2. even behaviors are defined as kinds of objects—although this may be transparent to the modeler.

Note, however, that important elements of the relational paradigm as used in database theory are also included, particularly in the area of class modeling (i.e., UML class diagrams). Specifically, in the modeling of associations. UML 2 provides a capability to render an association as either the equivalent of a table entry, to accommodate the relational approach, or, alternatively, as a set of attributes belonging to multiple classes, in support of the object-oriented approach.

Visual concrete syntax. The use of diagrams and graphical forms started with the earliest software modeling languages. This is due to the synthetic nature of visual representations, which seem to appeal to human intuition and cognitive mechanisms. Thus, flowcharts, which provide a static view of dynamic phenomena (the execution of an algorithm), are typically more easily grasped than equivalent textual representations. Similarly, visual representations are generally preferred for depicting structural relationships such as network topologies and inheritance relationships. This is particularly useful in object-oriented systems, since their fundamental operational paradigm is that of a network of collaborating entities (objects). Consequently, UML provides graphical representations for many of its key concepts. These are usually supplemented with textual annotations to capture the more detailed aspects. Several textual versions of UML have been defined, but none of them have been broadly accepted, which confirm the intuitive appeal of visual languages.

Separation of Views. Software invariably reflects the complexity of the real world in which it operates, and this complexity can often overwhelm our cognitive abilities. A traditional means for dealing with this problem is to focus only on those aspects of the system that are of concern to the observer. For example, when trying to understand how a collection of objects collaborate to achieve some emergent system-level behavior, the details of the inheritance structure of the objects involved is not relevant and can be safely ignored. However, when the focus shifts to the implementation of these objects, their inheritance structures become of paramount importance, whereas the interactions become irrelevant. This has motivated many modeling language designers to partition their language into multiple different types of views, with each type of view defined such that it renders only those aspects that are relevant to its specific set of concerns. In such languages, the full system specification is represented by the combination of all individual views. UML too has adopted this approach and defines several different diagram types that describe the structure and behavior of the modeled software system as well as the structure of the model itself. These diagram types are as follows:

- Package diagrams
- Class diagrams
- Instance diagrams
- Structure diagrams (subdivided into collaboration diagrams and composite structure diagrams)
- Interaction diagrams (subdivided into sequence diagrams, interaction overview diagrams, communication diagrams, as well as interaction tables)
- Activity diagrams
- Statechart diagrams
- Use-case diagrams
- Deployment diagrams

The purpose, form, and meaning of these diagrams are explained below.

Single Underlying Model. One of the common problems in many older modeling languages that use multiple types of views is inconsistencies that can be introduced into the model when two or more types of views overlap. Namely, some elements or aspects of a system may be specified in more than one type of view. If the views are constructed independently, it can happen that an element defined in multiple views may have contradictory specifications. To avoid this problem, the diagrams in UML are defined as partial views of a single underlying model. This model is constructed according to the rules of UML as defined by its *metamodel*. The UML metamodel is a formal model, specified using the *Meta-Object Facility*, a standard OMG language used to define modeling languages (12). It specifies all the modeling concepts of UML as well as the rules for how these concepts can be combined to ensure semantic and syntactic consistency of UML models. Thus, when something is specified in any UML diagram, it can be checked against the metamodel rules, and violations of those rules that would lead to inconsistencies in the model can be identified and flagged. (It should be noted, however, that the UML metamodel does not guard against all possible inconsistencies, so that additional consistency checks may still be required.)

Customizability. UML is a general-purpose modeling language that covers a broad spectrum of application domains. This generality implies that UML must abstract out characteristics that may vary from one domain to another or from one technology to another. For instance, different domains may have very different multitasking and scheduling policies. Or, different programming languages might differ in their rules for type compatibility or forms of inheritance (e.g., Java only supports single inheritance, whereas C++ supports multiple inheritance). To cope with this diversity, UML incorporates numerous *semantic variation points,* where the well-formedness rules of the metamodel provide for domain-specific or technology-specific choices to be specified. This implies that standard UML cannot be used as an implementation language, since it is not sufficiently refined to produce a complete implementation. However, it does provide a customization capability, in the form of profile definition mechanisms, with which it is possible to produce a specialized variant or domain-specific interpretation of UML. Such specializations, known as *profiles*, can be taken to any desired degree of precision, including transforming UML into a domain-specific implementation language with all the necessary detailed semantics required to generate complete implementations directly from the model. One important feature of the profile mechanism is that a validly defined profile can be supported by any tool that supports standard UML, potentially eliminating or greatly reducing the need for developing and maintaining custom tooling.

UML Diagram Types

The following discussion is a brief overview of the purpose and form of individual UML diagram types. Not all diagram types are illustrated but only the ones that are most widely used. Furthermore, only the salient aspects of the concepts in those diagrams are described. Readers interested in more detail should refer to the standard itself or one of the references in the Bibliography.

In most UML tools, models are created through the construction of diagrams. Note that a given model element can appear in more than one diagram or diagram type. In each case, only those aspects that are relevant to that view need be shown.

Package Diagrams. Packages are different from most other UML concepts because they are typically not used to model anything (although they can and are sometimes used for that purpose). Instead, they are used to partition the model into convenient groupings. A UML *package* is a named container that houses a collection of related model elements, possibly including other packages. The top-level package that contains all other packages and model elements is called a *model*. UML does not impose or assume any specific grouping criteria for packages, leaving the choice up to the modeler (e.g., grouping for reuse, grouping by ownership, or grouping by functional cohesion).

A package diagram graphically shows packages, their contents, and their relationships to other packages. In Fig. 1, *ModelPackage* contains three subsystem packages and imports the contents of the *UtilitiesLibrary* package. Note that the subsystem packages could have been drawn within the *ModelPackage* graphical element to show that they are contained inside the *ModelPackage.*

Class Diagrams. Class diagrams are the most widely used diagrams in UML. They are based on classic entity-relationship diagrams from database theory, but they have been adapted to the needs of the object paradigm. *A class* in UML is a specification of an object type, including all of its structural features (called properties), and behavioral features (called *operations*), as well as their visibilities with respect to other objects (public, protected, private or package). When an instance of a class is created, the result is an object with all the features specified for the class.

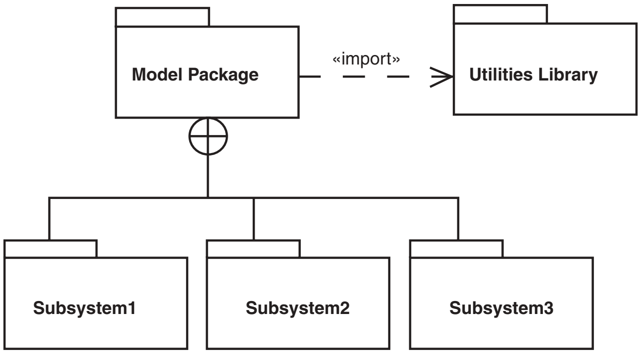


Figure 1. A package diagram.

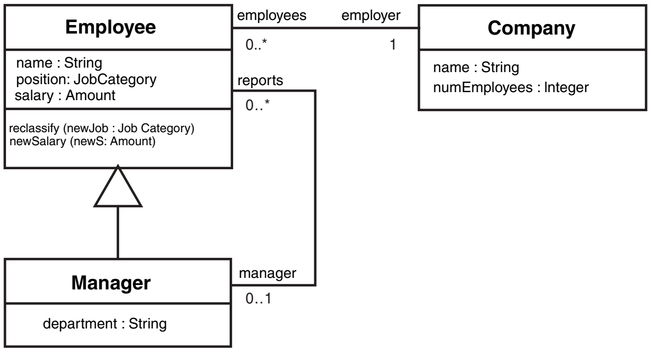


Figure 2. A class diagram.

A class is represented in a class diagram by a rectangle with the name of the class inside the rectangle. The diagrammatic representation may optionally include partial or complete lists of attributes and operations of the class. Each list is contained in its own subcompartment as shown in Fig. 2. The *Employee* class in the diagram is shown with three typed attributes and two operations, whereas the *Company* class is only shown with two attributes and no operations.

The class diagram in Fig. 2 also illustrates some important relationships that can exist between classes in UML. The arrow with the triangular arrowhead denotes *generalization*. That is, the class *Manager* is a special case of the more general class *Employee*. As such, *Manager* automatically inherits all the features (attributes and operations) of its parent class, but it may add its own additional features (e.g., the name of the *department* that the manager manages). The other lines in this diagram represent *associations*, which show how instances of the class at one end of the association relate to instances of the class at the other end. For example, the association between *Employee* and *Company* indicates that for each instance of *Employee* there is exactly one corresponding *Company* that is that employee's employer (the meaning of an association for a class is read at the far end), and that for each instance of the *Company* class, there is a *set* of zero or more (indicated by the "*" character) instances of the class *Employee* who are the employees of that company.

Instance Diagrams. These are also known as object diagrams, because they usually show how specific objects (class instances) relate to each other. Note that class diagrams abstract out individual object characteristics and only capture what is common across all instances of the classes shown in a diagram.

To distinguish more easily instances from classes, the names of instances in instance diagrams are underlined and the name of their type (class) is shown following a colon symbol. The instance diagram in Fig. 3 shows a set of object instances and is a particular case of the specification defined by the class diagram in Fig. 2. It depicts an instance of a company (named "Big Co.") that has four employees, one of which is the manager of the HR department who manages the other three employees. The lines that connect the objects are instances of corresponding associations and are called *links*.

Structure Diagrams. Whereas class diagrams specify what is common across all potential instances of a class independently of time, instance diagrams describe "snapshots" of a running system, showing specific instances at specific points in time. Structure diagrams belong in between these two extremes: Although they represent instances and their mutual relationships, they abstract away details of which particular instances are involved as well as the time of occurrence. This makes them useful for generic modeling of instances and their interconnection patterns. Figure 4 shows a *collaboration*, which is one kind of structure diagram that identifies a structural pattern of collaborating object instances (note that this diagram is a kind of generalization of the diagram in Fig. 3). The nodes represent generalized instances (generalized in the sense that their identities and attribute values have been abstracted out) and are called *roles*, whereas the lines represent generalized links and are called *connectors*.

Structure diagrams are also used to describe the implementation structure of complex classes that consist of collaborations of encapsulated objects. This internal structure is described by a collaboration structure drawn within a frame that represents the outer shell of the complex class (in contrast to the dashed oval frame used for collaborations).

Interaction Diagrams. UML interactions capture end-to-end scenarios that result from the collaborative actions of

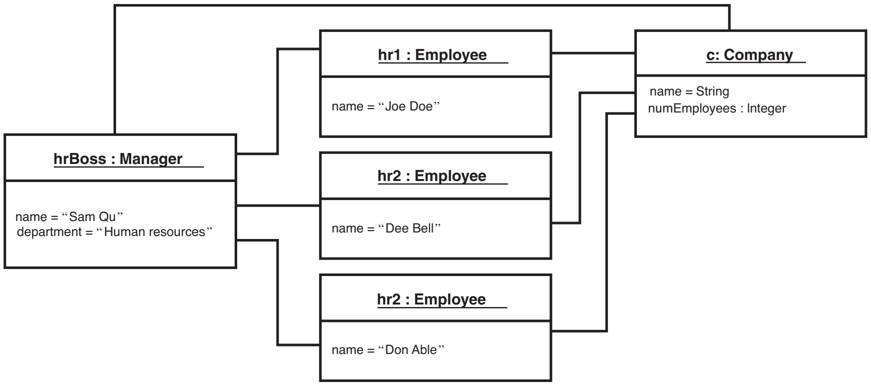


Figure 3. An instance diagram.

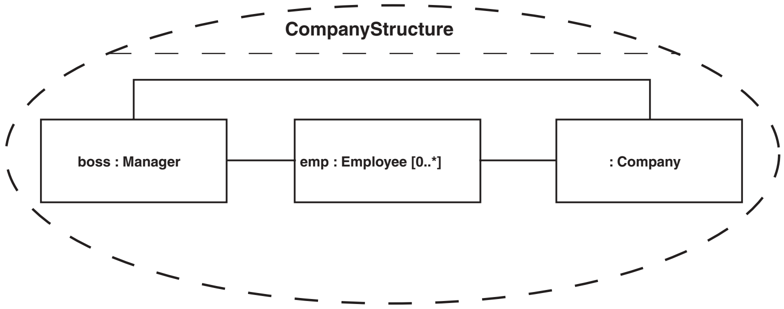


Figure 4. A structure diagram showing a collaboration.

multiple objects, which communicate with each other across links. Consequently, interactions are tightly coupled to collaboration structures, since these structures are used to identify both the participants of an interaction as well as the links (connectors) through which the communications takes place. The most common form for representing interactions in UML is the *sequence diagram*. These are diagrams in which the vertical axis represents the passage of time (although in some cases, it may run horizontally), whereas the horizontal axis identifies the participants in the interaction, shown as labeled vertical dashed lines called *lifelines*—as illustrated in Fig. 5. Communications between participants are represented by labeled lines with arrowheads that indicate the direction in which the messages are flowing. Several shorthand notations are defined to simplify complex interactions or to designate special semantics, such as the box labeled “loop” in Fig. 5, which indicates that the enclosed message sequence may be repeated multiple times. The thin rectangles overlapping the lifelines are called *execution occurrences* and model states during which the respective object is actively executing some behavior.

UML also provides other ways of representing interactions including communication diagrams, interaction overview diagrams, and interaction tables.

Activity Diagrams. Activity diagrams are an extended version of classic flowcharts and are used to represent algorithmic behaviors (e.g., business processes). The extensions include the ability to model the passing of data from one algorithmic step to the next (data flows) and to model concurrent execution.

The nodes in an activity graph represent either primitive actions or invocations of other activities (to support hierarchical functional decomposition). The directed arcs that join these nodes either represent the flow of data between nodes (*object flows*) or the passing of execution control from one node to another (*control flows*). A flow can be split into multiple concurrent flows, and conversely, concurrent flows can be reduced to a sequential flow. As an option, the individual nodes in an activity can be associated with the entities responsible for their execution, using a tabular format called *swimlanes*. An example of an activity diagram with swimlanes is shown in Fig. 6. When this activity starts, the *boss* role first waits for the arrival of a *start* signal and, when it comes, creates an order that is sent out to two employees (*emp[1]* and *emp[2]*). The employees then work on their respective copies of the order in parallel (the vertical bar with one incoming and two outgoing flows designates a concurrency fork). Note that the flows from *GetOrder* to the two *ProcessOrder* nodes are examples of object flows, which is signified by the labeled rectangle placed over the flow. The label identifies the type of data that flows between the nodes. Flows that do not have data placed over them are control flows, which means that execution control is passed from the source node to the destination node as soon as the source node completes. The activity then waits for both employees to complete the

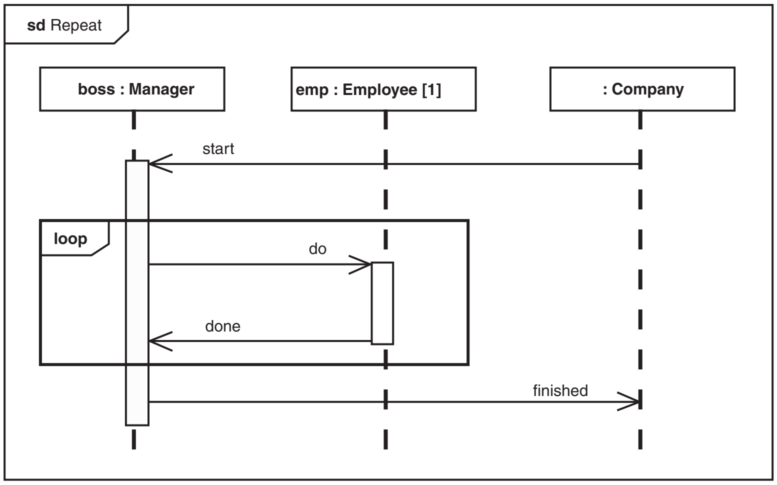


Figure 5. Sequence diagram.

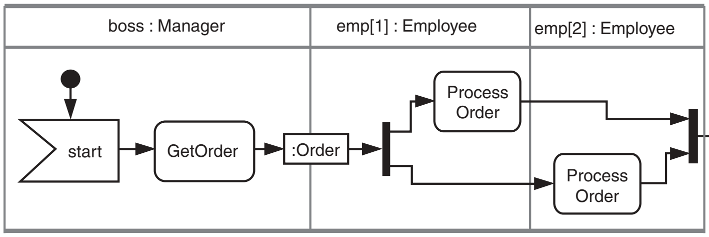


Figure 6. An activity diagram with swimlanes.

processing of their orders (indicated by the vertical bar with two incoming and one outgoing flow), and when that happens, the activity terminates.

Statechart Diagrams. State machines are used to describe event-driven behaviors in UML. An event can be the reception of a communications message, the expiry of some instant in time, or a change of state of some entity. The specific finite state machine formalism used in UML is a variant of David Harel's statecharts (13). Statecharts introduce several graphical shortcuts that enable the specification of some very complex behaviors in very concise ways. They also provide for hierarchical modeling so that a state at one level of abstraction can be decomposed into a statechart in its own right at the next lower level as illustrated by state *ProcessingOrder* in Fig. 7. Note also the transition from the edge of the *ProcessingOrder* state to the terminal state (indicated by a diagonal cross icon). This transition is taken when the *stop* event occurs regardless of which substate of the *ProcessingOrder* state is current at the time of the event. This is a shorthand representation for two separate transitions from the two inner states, each triggered by the same *stop* event.

Use-Case Diagrams. These diagrams are used to model requirements expressed in the form of use cases (i.e., ways in which the system under consideration is used to provide the desired functionality). Use case diagrams (see Fig. 8) identify the *actors*, who interact with the system to achieve the desired use case. They can also show various relationships between different use cases, such as when one use case incorporates another more primitive use case. In fact, use-case diagrams are merely a special form of class diagrams. However, these diagrams are not suitable for viewing the actual contents of use cases, which are typically captured as text.

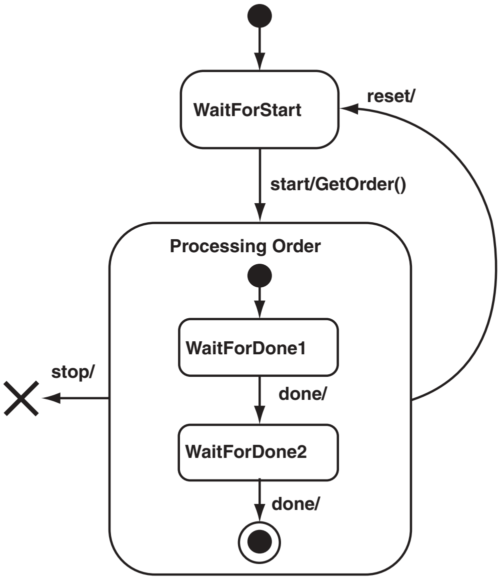


Figure 7. Statechart diagram.

Deployment Diagrams. Deployment diagrams are used to specify how software is distributed across an execution platform. In these diagrams, the software is represented by its physical manifestations, such as files, binary executables, deployment descriptors, and the like. Standard UML provides relatively rudimentary capabilities for modeling deployment, which is why this type of diagram is used less often in practice than any of the others. Where more sophisticated modeling of deployment is required, it may be necessary to define appropriate profiles. One such profile, adopted as an OMG recommendation, is the profile for *Modeling and Analysis of Real-Time and Embedded systems (MARTE)*(14).

UML Profiles

Profiles are a means for defining domain-specific interpretations of UML. Profiles usually consist of several refinements of standard UML concepts such that these refinements capture the specific characteristics of the concepts of a particular domain. For example, in the domain of concurrent programming, a mutual exclusion semaphore concept can be defined as a specialization of the general UML class concept. Such an extension would add suitable constraints and features that distinguish a semaphore from

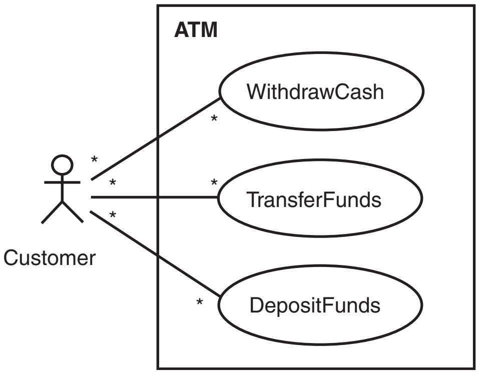


Figure 8. Use-case diagram.

other types of objects. These types of modeling concept extensions are called *stereotypes*, and once defined, they can be used like any other first-class UML language constructs. Since they are derived from standard UML concepts, they are compatible with any tool that supports the UML standard. Another important advantage of this approach to designing domain-specific modeling languages is that it reuses the design and validation efforts that went into standard UML.

Profile users have the choice to restrict their models to only use the domain-specific extensions defined in the profile, or alternatively, they can decide to combine the extensions with the general UML concepts. By selecting the first option, it is possible to derive a compact domain-specific language that is significantly smaller than the UML standard.

In addition to using profiles to construct domain-specific languages, it is possible to use them as domain-specific model filters. This is because of the ability for a UML profile to be dynamically applied to a UML model without corrupting the model. That is, when that profile is removed ("unapplied") at a later time, the original model, emerges unchanged. Such a "model filter" profile can be used to define a domain-specific interpretation of the original model. For example, it may be desired to determine the performance characteristics of a UML-based design. By applying a performance-based profile to the design model, it becomes possible to recast the UML model as a performance model, which can then be analyzed by appropriate methods and tools. Since the performance model is derived directly from the original model, the likelihood of translation errors from one formalism to another can be greatly reduced.

BIBLIOGRAPHY

1. Object Management Group, http://www.omg.org/.
2. I. Graham, *Object-Oriented Methods: Principles and Practice*, 3rd ed., Reading MA: Addison-Wesley, 2000.
3. G. Booch, *Object-Oriented Analysis and Design with Applications*. Benjamin-Cummings Publishing, 1993.
4. J. Rumbaugh, M. Blaha, W. Lorensen, F. Eddy, and W. Premerlani, *Object-Oriented Modeling and Design*. Englewood cliffs, NJ: Prentice-Hall, 1990.
5. I. Jacobson, *Object Software Engineering: A Use Case Driven Approach*. Reading MA: Addison-Wesley Professional, 1992.
6. Object Management Group, *Model Driven Architecture (MDA)*, OMG document ormsc/2001-07-01. Available: http://www.omg.org/docs/ormsc/01-07-01.pdf, July 2001.
7. Object Management Group, *The Unified Modeling Language: Infrastructure—version 2.1.2*, OMG document form/2007-11-04, Available: http://www.omg.org/docs/formal/07-11-04.pdf, November 2007.
8. Object Management Group, *The Unified Modeling Language: Superstructure—version 2.1.2*, OMG document form/2007-11-02, Available: http://www.omg.org/docs/formal/07-11-02.pdf, November 2007.
9. Object Management Group, *Object Constraint Language: Infrastructure—version 2.0*, OMG document form/2006-05-01. Available: http://www.omg.org/docs/formal/06-05-01.pdf, May 2005.
10. J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, 2nd ed., Reading, MA: Addison-Wesley, 2005.
11. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. 2nd ed., Reading, MA: Addison-Wesley, 2005.
12. Object Management Group, *Meta Object Facility Core Specification—version 20*, OMG document form/2006-01-01. Available: http://www.omg.org/docs/formal/06-01-01.pdf, January 2006.
13. D. Harel, A visual formalism for complex systems. in *Science of Computer Programming 8*. Amsterdam: North-Holland, 1987, pp. 231–274.
14. Object Management Group, *UML Model for Modeling and Analysis of Real-time and Embedded Systems (MARTE)*, OMG document ptc/07-08-04. Available: http://www.omg.org/docs/ptc/07-08-04.pdf, August 2007.

BRAN SELIC
Malina Software Corporation
Nepean, Ontario, Canada

VIENNA DEVELOPMENT METHOD

The Vienna Development Method (VDM) is one of the longest established *model-oriented formal methods* for the development of computer-based systems and software. It consists of a group of mathematically well-founded languages and tools for expressing and analyzing system models during early design stages, before expensive implementation commitments are made. The construction and analysis of the model help to identify areas of incompleteness or ambiguity in informal system specifications, and to provide some level of confidence that a valid implementation will have key properties, especially those of safety or security. VDM has a strong record of industrial application, in many cases by practitioners who are not specialists in the underlying formalism or logic. Experience with the method suggests that the effort expended on formal modeling and analysis can be recovered in reduced rework costs that develop from design errors.

VDM models are expressed in a specification language (VDM-SL) that supports the description of data and functionality. Data are defined by means of types built using constructors that define structured data and collections such as sets, sequences, and mappings from basic values such as Booleans and numbers. These types are very abstract, which allows the user to add any relevant constraints as data type invariants. Functionality is defined in terms of operations over these data types. Operations can be defined implicitly by preconditions and postconditions that characterize their behavior, or explicitly by means of specific algorithms. An extension of VDM-SL, called VDM++, supports object-oriented structuring of models and permits direct modeling of concurrency.

Because the modeling language has a formal mathematical semantics, a wide range of analyses can be performed on models, both to check internal consistency and to confirm that models have emergent properties. Analyses may be performed by inspection, static analysis, testing, or mathematical proof. To assist in this process, extensive tool support is available for building models in collaboration with other modeling tools, to execute and test models, to carry out different forms of static analysis, and to generate executable code in a high-level programming language.

The origins of VDM lie in work done in the IBM Laboratory at Vienna in the 1970s, where a formal specification language (Meta-IV) was developed to define the programming language PL/I. Meta-IV was subsequently used to define minimal BASIC, parts of FORTRAN and APL, ALGOL 60, Ada, and Pascal. The first description of this form of VDM, which was based on Meta-IV, was published in 1978 (1). Dines Bjørner and colleagues at the Dansk Datamatik Center developed the language definition capabilities of VDM that delivered the first European Ada compiler to achieve validation. Their modeling style, which emphasized explicit definition of functions, came to be known as the Danish School of VDM. Cliff Jones and colleagues, working at IBM Hursley, Oxford, and Manchester Universities subsequently developed the parts of VDM that were not specifically aimed at programming language definition into a more general modeling framework (2); their style, which emphasizes abstract modeling and validation by proof, became known as the English School of VDM. An account of the scientific decisions embodied in VDM can be found in Jones' summary (3).

The standardization of VDM-SL by the British Standards Institution and the International Organization for Standardization (ISO) sought to define a language that could accommodate the Danish and English Schools. It also provided an impetus for the development of tools to support the analysis of models written in the newly standardized language. At the same time, discussion surrounded the possibility of a "lightweight" application of formal modeling technology, which stressed the carefully targeted application of formal modeling, with strong industry-standard tool support. In spite of the 'Method' in its name, the use of VDM does not prescribe a particular development process or methodology. Instead, the components of the method may be used as developers see fit. This pragmatic approach (4) led to substantial advances in the application of VDM and later extended to the VDM++ language (5).

The major part of this article is an overview of the elements of the modeling language, validation technology, tool support, and industrial application. More advanced technical aspects of the semantics of the modeling language and its associated proof theory are briefly introduced. Current trends and open questions are identified. The leading source of current information is the VDM Portal (www.vdmportal.org), and the most current text on the tool-supported approach to system modeling and validation in VDM++ is Fitzgerald et al. (5). Texts on more detailed topics are cited in the body of the article.

SYSTEM MODELING IN VDM

The use of VDM involves the development and analysis of models to help understand systems and predict their properties. Good models exhibit abstraction and rigor. Abstraction is the suppression of detail that is not relevant to the purpose for which a model is constructed. The decision about what to include and what to omit from an abstract model requires good engineering judgment. A guiding principle in VDM is that only elements relevant to the model's purpose should be included; it follows that the model's purpose should be clearly understood and described. Rigor is the capacity to perform a mathematical

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

analysis of the model's properties to gain confidence that an accurate implementation of the modeled system will have certain key characteristics.

In computing systems development, modeling and design notations with a strong mathematical basis are termed *formal*. VDM is based on a formal specification language VDM-SL, the semantics of which are given mathematically in an ISO Standard (6). VDM models, although often expressed in an executable subset, are developed primarily for analysis rather than for final implementations.

Model Structure

In VDM, models consist of representations of the *data* on which a system operates and the *functionality* that is to be performed. Data includes the externally visible input/output and internal state data. Functionality includes the operations that may be invoked at the system interface as well as auxiliary functions that exist purely to assist in the definition of the operations. The ISO Standard (6) defines a (nonmandatory) module framework for VDM-SL models. This framework includes traditional import and export features as well as module parameterization and instantiation.

The VDM++ language extends VDM-SL with facilities for specification of object-oriented systems, and structures models into class definitions, each of which has similar elements to a single VDM-SL specification, with the state variables taking the role of instance variables and the operations playing the part of methods. The remainder of this section will restrict consideration to VDM-SL, with VDM++ considered at a later stage.

Modeling Data

Data models in VDM are founded on basic abstract data types together with a set of type constructors. A full account of VDM-SL data types and type constructors is provided in current texts (5).

Basic types include numbers (natural, integer, rational, and real) and characters. Note that, in accordance with VDM's abstraction principle, no predetermined maximum representable numbers or real number precisions exist. If a user wishes to specify these limits because they are relevant to the problem being modeled, then it is possible to do so explicitly by means of invariants. Invariants are logical expressions (predicates) that represent conditions to be respected by all elements of the data type to which they are attached.

Throughout this article, the ASCII syntax of the VDM-SL will be used. An alternative mathematical syntax is used in the older texts but, although both are permitted by the ISO Standard, the ASCII syntax is considered the more accessible for readers unfamiliar with the notations of discrete mathematics. Keywords are, by convention, shown in bold face. Consider, as a simple example, a system for monitoring the flight paths of aircraft in a controlled airspace. A simple data type definition that represents the Latitude of an aircraft would be given as follows:

```
Latitude = real
```

If it is desired to restrict the `Latitude` to the range of numbers from –90 to 90 inclusive, then an additional condition is added to the data type in the form of an *invariant*. This extended type definition is as follows:

```
Latitude = real
inv lat == lat >= -90 and lat <= 90
```

The invariant is an integral part of the data type. Thus, it is not possible to create a value of type `Latitude` that does not respect the invariant. The modeler must ensure that all functions and operations that create such elements respect the invariant.

More sophisticated data types are built using constructors. A record type constructor permits the definition of tuples with named fields. For example, assuming definitions of types that represent `Latitude`, `Longitude`, and `Altitude`, it is possible to define a type of values representing aircraft position, as follows:

```
Position :: lat :Latitude
            long:Longitude
            alt : Altitude
```

A value of type `Position` is a composite whose component values can be extracted by giving the field names. Thus, the `Longitude` component of a position `p` is given by `p.long`. VDM-SL also contains type constructors for building union and Cartesian product types.

Models are typically built around structured collections of values, so VDM-SL provides type constructors that support several collection types: sets (finite unordered collections), sequences (finite ordered collections or arrays), and mappings (finite functions). For example, one may wish to define a type to model the path of an aircraft as a finite sequence of positions. The corresponding definition is:

```
FlightPath = seq of Position
```

Thus, an element of type `FlightPath` is a finite sequence of position records. Given a value `fp` of type `FlightPath`, the initial `Altitude` is expressed as `fp(1).alt`. If modeling a flight control system that must manage several aircraft, then it would be appropriate to define a type that relates aircraft identifiers to their flight paths as a mapping:

```
FlightDetails = map AircraftId to FlightPath
```

A mapping in VDM is the abstract model of an associative array; individual associations are represented using an "arrow" notation. For example, '`{ 3 |-> "text1", 7 |-> "text2" }`' represents an association between numbers and character strings. In the flight details example, the mapping represents a finite collection of flight paths indexed by the aircraft identifier. Given a flight details mapping '`fd`' and an aircraft identifier '`a`', the following expression denotes the initial `Altitude` of '`a`':

```
fd(a)(1).alt
```

Several special basic types also facilitate abstraction. The *token* type is used to denote values whose representations are immaterial. Tokens can be compared for equality, but they have no internal representation so no other operators may be applied to them. Tokens are particularly useful for defining types that are necessary to a model but for which no individual elements are required. For example, if the air traffic model is concerned primarily with flight paths rather than call signs, then the modeler may choose not to give a detailed representation for the `AircraftId` type, preferring to use a token type:

```
AircraftId = token
```

Modeling Functionality

Functionality is described in terms of functions and operations that accept input values and deliver output values that belong to the types defined in the model. As with data, VDM-SL contains features to support abstraction of functionality.

Each basic type and type constructor has associated syntax allowing values to be expressed. For example, a sequence of four natural numbers might be expressed directly as follows:

```
[3, 7, 7, 2]
```

Comprehension notations allow more sophisticated constructions. For example, the following expression represents a sequence of all the squares of numbers up to 25:

```
[ n**2 | n in set{ 1,...,25} ]
```

The types are equipped with operators that allow complex expressions to be constructed. For example, given a value `s` that belongs to a sequence type, the expression **`len`** `s` denotes the length of the sequence. Two sequences `s1` and `s2` may be concatenated by an infix operator: `s1^s2`.

As in programming languages, some operators are *partial*, (i.e., undefined for certain values of their arguments.) For example, a sequence lookup such as the expression `s(i)` is undefined if the sequence `s` contains fewer than `i` elements. Such misapplications of partial operators correspond to potential run-time errors in a corresponding implementation. The behavior of a real computing system when such an error occurs is not usually predictable. An error message may be returned, or an infinite loop may be entered, for example. Because such behavior can rarely be known at modeling time, VDM treats them all as mathematically undefined in the semantics.

Functions may be described explicitly or implicitly. An explicit function definition is an expression that denotes the result to be returned in terms of input parameters. Returning to the air space management example, the modeler may wish to specify a function that adds a new position on to the end of a flight path. The function definition is given as follows:

```
AddPos: FlightPath * Position -> FlightPath
AddPos(fp,p) == fp^[ p]
```

Implicit function definitions provide an important abstraction capability in VDM. Although an explicit definition like the one shown above is concise, it may be considered to bias a reader implementing the model toward a particular implementation, for example using a corresponding concatenation operator built in to a programming language if present. An implicit definition describes a function purely in terms of the result to be delivered, with no direct reference to any algorithm to be used in the computation. This definition is given in terms of a logical (Boolean) expression that must be satisfied by the result. This expression is termed a *postcondition.* A classic example is a specification of a function for computing the square root `r` of a natural number `n`:

```
SQRT(n:nat)r:real
post r* r = n
```

Here the required result is merely characterized, with no bias toward any particular implementation. In particular, it will be noted that the postcondition does not constrain the result to be either positive or negative; the modeler has indicated that either result will suffice provided that it is a square root of the input `n`. Such implicit specification is valuable where the provision of an algorithmic description would obscure the meaning of the model. The disadvantage is that an implicit operation specification is not directly executable. In the airspace management example, an implicit specification might be used for a function to select a specific aircraft for landing, specified as follows, where the '**`in set dom`**' construction means that the result returned is present in the domain of the flight details mapping structure:

```
Select(fd:FlightDetails)a:AircraftId
post a in set dom fd
```

Both explicitly and implicitly defined functions may not always be applicable. For example, the function above cannot return a result if the flight details mapping `fd` were empty. The function description therefore cannot be satisfied for all valid inputs. Nonemptiness of the input `fd` is a *precondition* on the successful application of the function. Such preconditions are recorded explicitly in VDM. Thus, a satisfiable specification of the `Select` function would be as follows, where the precondition is shown in italics:

```
Select(fd:FlightDetails)a:AircraftId
pre dom fd <> { }
post a in set dom fd
```

Preconditions, like invariants, provide a means of recording constraints that are often left unrecorded in informal descriptions of computer-based systems. In the example above, the precondition is required to ensure that the function can return a correct result in accordance with the postcondition. An implicit specification can be considered a contract: An implementation of the operation promises to return a result that satisfies the postcondition provided the calling environment ensures that the precondition is satisfied. If the precondition is not satisfied, then no guarantees about behavior are made.

Modeling State and Operations

Many systems have persistent state variables that are read and modified by operations and that retain data between operation invocations. In VDM, such systems are modeled by defining a distinguished state variable of a defined type and operations that, like functions, deliver outputs from inputs but that may also have side effects on the state variables.

A state-based version of the airspace management system might have a single state variable of type `FlightDetails`, modeling the current state of the airspace:

```
state Airspace of
      fd: FlightDetails
end
```

An operation to add a new aircraft with a single position `p` in its flight path might be specified implicitly as follows. Note the use of the prefix '~' to denote the state variable's value before execution of the operation. This decorated version is required because the postcondition describes a mathematical relation between the preoperation and postoperation states. The `munion` operator used in the postcondition here forms the union of two mappings provided the two mappings do not disagree (any values in both domains map to the same range value).

```
New(a:AircraftId,p:Position)
ext wr fd: FlightDetails
pre a not in set dom fd
post fd = ~fd munion { a |-> [p] }
```

Operations may be specified explicitly as well as implicitly. Where state variables may be modified, the language for expressing such explicit operation definitions is close to that of a classic imperative programming language, albeit one with very abstract data types. For example, the following explicit definition of the `New` operation contains a single assignment to describe the updating of the `fd` state component. The signature of the operation, which is shown in the first line of the definition, shows only the visible input and output types. Because this operation produces no external output, but merely updates the state, the return type is empty, as indicated by the "()".

```
NewOp: AircraftId * Position ==> ()
NewOp(a,p) == fd := fd munion { a |-> [p] }
pre a not in set dom fd
```

Full details of implicit and explicit specification styles for both functions and operations can be found in the VDM literature (4, 5).

Modeling Object-Oriented and Concurrent Systems in VDM++

VDM++ provides facilities for the description of object-oriented systems. The elements of classic VDM-SL are all present, but the extended language provides for models based on class definitions in which each object's local state is represented as instance variables and operations are treated as methods. Information hiding and inheritance are also supported.

VDM-SL is limited to the description of sequential system models, although such models may be implemented in a parallel-computing framework. The challenge in modeling concurrent computation is that separate threads (independent sequences of computations) may communicate through shared variables and inconsistencies can develop when two or more independent threads access a shared variable simultaneously. Considerable research has been performed on handling shared variable concurrency in VDM, notably by extending the pre/postcondition framework with *rely* and *guarantee* conditions that state, respectively, the properties that an operation requires to be invariant and the properties that it guarantees to maintain during its execution (7).

The rely/guarantee approach has been a significant contribution to design methodologies for concurrent systems generally. VDM++ takes a rather pragmatic line. Here inconsistencies may develop through simultaneous access to shared objects by separate threads. These inconsistencies are avoided by providing synchronization constraints in the form of *permission predicates* that describe the conditions under which an operation may be carried out. A permission predicate may refer to an instance variable used as a flag to prevent other threads from an object being used in a critical way by another thread. It may also access special variables that represent the number of times each operation in an object has been requested, activated, or completed, or representing the number of currently active invocations of the current operation. Consider a simple model in which a sensor produces data, writes it to a buffer object, and then these data are consumed by a consumer object. The buffer object provides a data model of the buffer and methods (operations) to `Put` and `Get` data. The consumer object should only invoke the `Get` operation on the buffer when data is available to get. This restriction could be modeled by allowing a special value **`nil`** to indicate emptiness of the buffer, in which case the permission predicate (denoted by the keyword **`per`**) on the `Get` operation in the buffer object is of the form shown below:

```
per Get => data <> nil
```

If such a special **`nil`** flag is not available, one could count the number of completed `Put` and `Get` operations and permit a `Get` operation under the condition specified as follows:

```
per Get => #fin(Put) - #fin(Get) = 1;
mutex(Put,Get)
```

Here the expression '**#fin**`(op)`' represents the number of completed occurrences of the operation `op`. The `mutex` condition enforces mutual exclusion of the `Put` and `Get` operations.

It is worth emphasizing the difference between preconditions and permission predicates on operations. An operation's precondition records a fundamental assumption about the circumstances under which the operation will be invoked. If an operation is called in violation of its precondition, then no guarantees are given about the system's subsequent behavior (the modeling equivalent of a run-time error). In contrast, a permission predicate determines whether a request to perform an operation will be granted or denied. If permission is denied, then the request is blocked and another thread may be executed; a particular thread of computation may be held up when a permission predicate evaluates to false, but other threads can progress.

MODEL VALIDATION

Validation is the process of gaining confidence that the model describes the behavior that one would expect of an accurate representation of the system of interest. As a consequence of VDM-SL's formal definition, a particularly wide range of validation checks can be performed, with automated support, to identify potential static and run-time errors in a model as it is constructed. Most checks are described as *proof obligations* because they are observed as requirements on the modeler. Once the proof obligations have been discharged, the model's emergent behavior can be checked against expectations, which is described as *validation conjectures*.

Proof Obligations and Validation Conjectures

A range of basic proof obligations is common to all VDM models. For example, it should always be possible to tell whether a value belongs to a specified data type. Consequently, any invariants must be expressions that always return a value (true or false), given any element of the restricted supertype. For example, the definition of the `Latitude` data type defined earlier uses an invariant '`lat >= -90` **and** `lat <= 90`'. It must be possible to evaluate the invariant for any real number, so that it is possible to tell whether the number is a valid `Latitude`. This task seems trivial in this example, but it need not always be so. In particular, if an invariant makes use of a partial operator, then it is necessary to ensure that the partial operator is not applied outside its domain of definition. Consider a general type definition of the following form:

```
T = Rep
inv t == P(t)
```

The proof obligation for the invariant is shown below, where '**inv**-`T`' refers to the invariant as a Boolean function. The '**forall**' keyword is a quantifier over the type `Rep`, where `t` is the "bound variable" that ranges over the type, so this formula literally states that "for all values t of type Rep, the invariant yields a valid Boolean value" (the '&' symbol is merely used as a syntactic separator):

```
forall t:Rep & inv-T(t) : bool
```

One of the more significant and useful proof obligations is that of invariant preservation. When a function `f` returns a result of type `R` and the definition of `R` has an invariant, an obligation exists to show that the function respects the invariant on `R`. Formally, consider a function with the following signature (a signature is a summary of input and result types):

```
f: I -> R
```

In this case, the proof obligation is stated formally as follows:

```
forall i:I & pre-f(i) => f(i):R
```

For example, a function to increase a `Latitude` by adding an increment must be shown to take proper account of the invariants and behave correctly at the extremes of the range.

Versions of invariant preservation also exist for implicitly defined functions and particularly for implicitly specified operations, where the obligation is termed *satisfiability*. Consider an operation `Op` taking an input `i` of type `I` and operating on a state variable `s` of type `S`, returning a result `r` of type `R`. The operation is said to be satisfiable provided that, for all inputs satisfying the precondition, there exist a state after execution and result that together satisfy the postcondition. Formally, the obligation is to show the following:

```
forall i:I, ~s:S & pre-Op(i,s) =>
     exists s:S, r:R & post-Op(i,~s,s,r)
```

In practice, satisfiability is one of the most important checks to be performed on a model. Because invariants frequently embody safety constraints, checking that functionality respects them is one means of early detection of subtle defects in an implementation.

Validation Techniques

Once a model's proof obligations are satisfied, its other properties may be explored using a range of techniques. Again, because of the formality of the modeling language's semantics, analysis by mathematical proof is possible, and it is often done for critical applications. At a less rigorous level, models can also be explored by testing.

Explicitly defined functions and operations can be executed directly by means of an interpreter, provided they are expressed within an executable subset of the language. The expressiveness of modeling languages like VDM-SL is such that useful expressions can be written that are nevertheless not readily executable. One main cause of nonexecutability is quantification over unbounded data types. For example, the following function definition is not executable because it quantifies over the whole unbounded type of natural numbers.

```
is_square(i) == exists j:nat & i = j**2
```

In this example, an interpreter could be expected to iterate over the natural numbers until a suitable value for `j` is found. In the case where `i` is not a square, the iteration might not terminate unless some additional reasoning is performed, using information from the problem domain to halt it. The main existing VDM and VDM++ interpreters forbid attempts to execute formulae involving unbounded quantifications. Quantifications over other collections such as finite sets are permitted, however, so one might define the `is_square` function as follows:

```
is_square(i) == exists j in set{0,...,i} &
i = j**2
```

The disadvantage of building this extra knowledge into the model is that it is information added for the specific purpose of executing the model, and so may compromise abstraction (8,9).

Despite the caveats about executability, VDM models used in industrial applications are often built within the executable subset and validated mainly through testing. Tool support allows very efficient interpretation of the model, and additional tools facilities support batch-mode testing as well as test-coverage analysis. The level of confidence gained by model testing is limited by the quality of the test set. However, the level of abstraction means that validation tests used on a model provide a strong basis for designing tests used on subsequent system implementations.

Formal proof provides a much more general validation technique than testing. The formal semantics of VDM-SL has generated a well-documented proof theory (10), and the development of automated support for proof is the subject of research in the Overture project (www.overturetool.org). Experiments suggest that typically up to 90% of proof obligations can be automatically discharged by a theorem prover. However, user-guided proof, given adequate tool support, can be an effective means of exploring a model when an automated analysis simply identifies the presence of defect.

TOOL SUPPORT

VDM's recent industrial application has been closely tied to its tool support. Early tools for VDM, such as Adelard's SpecBox (11), (Adelard LLP, London, UK) were largely confined to basic static checking and pretty-printing of specifications. However, the availability of the ISO VDM-SL Standard (6, 12) gave impetus to the development of a toolset based on a parser, type-checker, and interpreter for the executable subset of the language. The product that ultimately resulted from this work, VDMTools (13), was primarily aimed at cost-effective industrial use rather than expressive completeness. VDMTools originated with the Danish company IFAD, but it is now maintained and further developed by the Japanese corporation CSK Systems.

A feature of successful tool support is that it should be capable of integration with existing tools in the development environment. VDMTools supports models written with the aid of any environment that can output a text file (or Microsoft Word (Microsoft corporation, Redmond, WA) file using a VDM style), but it has also been important to develop a link between the VDMTools environment and other object-oriented design tools. A bidirectional link exists with the IBM Rational Rose (IBM, Armonk, NY) toolkit that allows modelers to design the model structure as a UML class diagram and convert this directly to a VDM++ model. Subsequent changes to the VDM++ model can be reflected in the UML model and vice versa. Once a model has been written, it can be syntax checked and type checked. Proof obligations can be generated automatically and checked manually. This interplay between UML and VDM++ models, kept in sequence as a design evolves, is an instance of an approach sometimes referred to as "round-trip engineering."

To support validation, the VDMTools contain an interpreter for test-based analysis of specifications written within an executable subset. Testing is supported by a coverage analyzer. A CORBA-based application programmer interface (API) allows models to be executed on the interpreter but accessed through a graphical user interface, so that domain experts unfamiliar with the modeling language can explore the behavior described by the model by playing out scenarios or other test cases. The interpreter has a dynamic link library feature that allows external modules to be incorporated. Interpretation of an abstract model is typically slow compared with program code, so automatic code generation can be used to derive programming language implementations (e.g., in Java or C++) direct from the model.

The emphasis in VDM tool support has been on the provision of facilities that allow the formalism to be applied within existing development processes and in conjunction with existing tools. This emphasis has meant that the priority for tool support has been the provision of an efficient interpreter and the supporting features. Support for proof has been limited to research activity. A major spur to the development of a proof system for VDM was provided by the Mural tool (14), developed as part of the IPSE2.5 project in the 1990s. Mural provided an environment that supported manual construction of proofs by human users, with the tools managing the bookkeeping aspects of proof production. For example, the tool ensured that all the inference steps in a proof were sound applications of previously defined or proven rules, and it also managed the structuring of large numbers of rules into structured collections. Such a user-led proof is extremely time consuming if it all has to be done manually. However, advances in automated proof technology have allowed experimental use of theorem provers to discharge most proof obligations derived from VDM models. The expressiveness of the modeling language means that not all proof obligations or validation conjectures that are true can be proven completely automatically. Future extensions to VDMTools will support automatic proof of most obligations, but the provision of fully integrated manual and automatic proof support remains a research goal.

INDUSTRIAL APPLICATIONS

VDM has a strong record of industrial application in a wide variety of application domains. After the development of VDMTools, the approach advocated in industry concentrated on the construction of abstract models and test-based analysis. Several examples of industrial application give an indication of the ways in which the modeling technology is applied. More detailed analysis of these applications has been reported elsewhere (15).

The ConForm project (16) was an industrial case study rather than a commercial application. British Aerospace Systems and Equipment (BASE) studied two concurrent developments of a security-related software component using, in one stream, current best practice and, in the other stream, model-oriented specification using VDM-SL. Comparison of the two developments suggested a shift of effort from implementation to analysis phases in the system development and also indicated a shift in the volume and types of query raised against informal requirements. The project involved both systems and software engineers applying the modeling technology.

DustExpert (developed in 1995–1997 by Adelard LLP) is a safety-related knowledge-based system to advise on the construction of industrial plants that contain potentially explosive dusts. A VDM-SL model of 12kLOC is derived from about 450 pages of requirements formed the basis of manual proofs concerning safety properties as well as test-based analysis using the interpreter. Together these contributed to the product's safety case. The defect density of the completed product (implemented in Prolog and C++) was reported by the developers as lower than 1 defect/kLOC.

In 1996, Praxis reported on the development of the *Central control function Display Information System* (CDIS), an air traffic control support system for use in the London area (17). Several modeling approaches were used, including a modular variant of VDM. The operational software was about 197kLOC in size and exhibited 0.75 defects per kLOC, which made it one of the larger projects to have applied formal modeling techniques at the time.

TradeOne (developed between 2000 and 2002) is a back-office system for securities trading, developed by JFITS, now the CSK Systems part of CSK. Metrics from the development process were reported in 2005 (5). Two complex subsystems that handle tax exemptions and options were modeled in VDM++ prior to implementation in C++ and Java. The defect densities were reported as zero and 0.05 defects per kDSI in the two running subsystems. The effort profile for the development was also analyzed, which suggested an increase in the proportion of effort devoted to analysis. Overall development costs were compared extremely favorably against regular COCOMO (18) estimates.

Felica Networks Inc. in Japan have reported an application of VDM++ to the development of the firmware for a next generation mobile Integrated Circuit chip, which is based on contactless card technology (15). VDM++ models were developed alongside UML models as part of the process of improving requirements descriptions prior to implementation. The model was validated by performing over a high volume of tests to achieve very high path coverage. Most defects uncovered in requirements were attributed to the use of formal modeling. Of these, slightly more than half were attributed to issues raised during construction of the model, and the remainder was attributed to testing the model.

Industrial application of proof technology has so far been restricted largely to specialist high-integrity systems in areas such as the nuclear industry (19), although the variety of domains covered in the examples presented here suggests that model-oriented formalisms like VDM have the potential to be applied widely. However, a tendency exists to restrict the use of proof to areas in which high quality is the dominant concern or the complexity of functionality or data threatens the success of a development.

REIFICATION: FROM MODELS TO DESIGNS

Formal models often serve as abstract specifications of systems to be implemented in high-level programming languages. Each model typically contains representations of elements of the problem domain (for example, aircraft identifiers and flight plans), but an implementation must be expressed in terms of concepts present in the chosen programming language. Reification (more widely known as refinement) is the process of transforming an abstract model in VDM into a concrete one that includes representations of data and functionality that are expressed readily in the target programming language. The reification process is typically done in a series of steps, in which each introduces small changes to the model and involves verification that the modified (concrete) model still describes the same behavior as its more abstract predecessor. By composing several reification steps, a sufficiently concrete model may be reached. VDM pioneered the use of proof obligations to govern this stepwise reification process (2). Rules govern the reification of state definitions and operations.

Data Reification

In data reification, an abstract type representation is replaced by a more concrete counterpart. For example, an abstract set of aircraft (an unordered collection of objects) might be reified by a more concrete ordered sequence (array) structure. This structure might, in turn, be reified even more to a model of a linked list prior to translation into a programming language that supports linked list structures.

A data reification step is correct if every abstract state has a concrete counterpart. The correctness is typically checked by proposing a *retrieve function* that recovers the abstract counterpart from any concrete model. The retrieve function must be *total* and *adequate*. A total retrieve function is one that is defined on any possible concrete value. An adequate retrieve function is one that ensures every abstract value has a concrete counterpart (in mathematical terms, the function is "surjective"). Formally, given a concrete type `C`, an abstract type `A`, and a retrieve function `r`:

```
forall c:C & retr(c):A                    Totality
forall a:A & exists c:C & r(c)=a          Adequacy
```

In the example, the retrieve function simply gathers the elements of the array and generates the set that contains all those elements. Stated formally in VDM-SL, this appears as follows:

```
retr: seq of AircraftId -> set of AircraftId
retr(c) == elems c
```

Note that, in this example, the concrete type introduces redundancy because a sequence may contain duplicate values, whereas a set suppresses duplication. A set that contains a single aircraft identifier might be represented by a sequence that contains just one element or by a sequence that contains two or more repetitions of the same aircraft identifier.

When a model is subjected to data reification, the operation specifications that work on the abstract state must be adjusted to the new concrete form. Informally, the concrete operation may have a more liberal ("weaker") precondition than the abstract version, but it must have at least as restrictive a postcondition (i.e., one that is "stronger"). If an operation `OpA` is reified to a concrete counterpart `OpC`, then two conditions must be satisfied. First, `OpC` must be defined to operate on at least the states that `OpA` worked on (when viewed through the retrieve function). Second, `OpC` must define a relation between states that respects the relation between states defined by the abstract form (when viewed through the retrieve relation). These two conditions are expressed formally as the Domain Rule and Result Rule shown below.

```
forall c:C & pre-OpA(retr(c)) =>
pre-OpC(c)                                Domain Rule

forall ~c,c:C &                           Result Rule
    pre-OpA (retr(~c)) and post-OpC(~c,c) =>
       post-OpA(retr (~c),retr(c))
```

A full discussion of the data reification proof obligations is given by Jones (2).

Operation Decomposition

In reifying functionality, operations are broken down into simpler suboperations linked by control constructs such as sequential compositions, conditionals, or loops. Successive decomposition steps add control constructs and reduce the complexity of the suboperations. Rules governing such decomposition steps are given in a "triple" format. In such a format, the expression

$$\{P\}\ \mathrm{Op}\ \{Q\}$$

is written to indicate that the operation Op, when invoked in a state that satisfies the Boolean condition P, terminates and results in a condition satisfying the condition Q, where Q is a logical expression over both the before and after states. As a simple example, consider a rule for introducing conditionals. Specifically, suppose an operation with precondition *pre* and postcondition *post* is to be decomposed into a structure of the following form:

$$\text{if } \mathtt{test} \text{ then } \mathit{TH} \text{ else } \mathit{EL}$$

where *test* is a Boolean expression, and *TH* and *EL* are operations that correspond to the functionality to be executed in each limb of the conditional. The condition under which the decomposition is sound is given as follows:

$$\frac{\{pre\ \textbf{and}\ test\}TH\{post\};\quad \{pre\ \textbf{and not}\ test\}EL\{post\};\ pre \Rightarrow \delta(test)}{\{pre\}(\text{if}\ test\ \text{then}\ TH\ \text{else}\ EL)\{post\}}$$

Here the decomposition shown below the line is asserted to be sound under the conditions shown above the line. It is necessary to show that the postcondition is established in both the *TH* and *EL* branches of the conditional. The "δ(*test*)" condition is a requirement that the test actually terminates. All three conditions may be established under the assumption that the precondition *pre* holds. Other rules for operation decomposition are presented in Jones' book (2).

SEMANTICS AND FORMAL REASONING

VDM was one of the first modeling languages to have a mathematical semantics codified in an ISO standard (6,12). The semantics are denotational in style and give the meaning of models expressed in a core language into which full VDM-SL may be translated. The semantics is defined in terms of domains built on complete partial orders. The meaning of a VDM model is defined as a collection of possible environments that define domains for each of the defined data types. Operations are denoted as sets of valid relations over the domains that correspond to the types of the inputs and state (20).

As noted earlier, VDM has several features that support a level of abstraction in models that is not typically found in programming languages. These features lead to the more interesting aspects of the denotational semantics. One of the most significant features is the ability to use a style of loose specification. Loose specification develops where the model permits a level of choice. For example, an implicitly specified function or operation presents a choice over which of possibly many results satisfying the postcondition is to be returned. Loose specification is also supported by other constructs in the language, which allow choices to be arbitrarily from sets of values, for example. Two possible semantics are used for loose expressions. The first, termed underdeterminedness, interprets the construct as defining a set of possible different deterministic implementations. The second, termed nondeterminism, allows implementations that are themselves nondeterministic. The VDM-SL semantics treat looseness in function definitions as underdeterminedness, although operations are potentially nondeterministic. Functions that contain loose-choice operators are thus denoted as sets of possible deterministic functions, each representing a different choice. For operations, the possibility of nondeterminism is retained to allow the modeler to describe behavior that may be governed by factors outside the collection of inputs to the operation (21).

Logic and Proof in VDM

VDM has a well-established theory that supports the construction of proofs about models and reifications. One of the most distinctive features of VDM is the handling of undefinedness. Undefined terms can develop naturally in an abstract modeling framework as well as in program code. For example, an expression that looks up a value in a sequence has the possibility of yielding an undefined value if the index is out of range. The user can also define functions that may not terminate for certain inputs. The following simple example is regularly used in discussions about logics handling undefinedness. The `subp` function takes two integers `i` and `j` as inputs and returns `i−j` provided `i = j`, but otherwise its result is undefined:

```
subp: int* int -> int
subp(i,j) == if i=j then 0 else subp(i,j+1)
```

In reasoning about this function, the following assertion is plausible:

```
forall i,j:int & i < j or subp(i,j) = i−j
```

If `i < j`, then the right hand term using `subp` in this expression is undefined, and so the whole conjecture is meaningless in a classical logic.

The problem of how to reason about undefined values is a significant one in formal methods. Many approaches aim to use classical two-valued logic. Some approaches ban potentially undefined terms, and they generate proof obligations to enforce this. Other approaches allow undefinedness but insist that terms such as `subp(i,j)` always yield a valid (type correct) but unknown value. VDM's approach is unusual in that its logic is nonclassical, which allows undefined value terms and undefined logical values the logic of partial functions (LPF) (22) underpins reasoning about models and reification in VDM (23,24). LPF is in many respects similar to a classic logic, but the definitions of operators are extended to cope with undefined, as well as true and false terms. For example, the truth tables for disjunction and negation are shown below, where ⊥ represents an undefined Boolean expression:

A	B	A **or** B
true	true	true
true	false	true
true	⊥	true
false	true	true
false	false	false
false	⊥	⊥
⊥	true	true
⊥	false	⊥
⊥	⊥	⊥

A	**not** A
true	false
false	true
⊥	⊥

The truth table for ‘**or**’ may be thought of as describing a parallel lazy evaluation of the operands: the expression ‘A **or** B’ evaluates to true if one operand evaluates to true, even if the other operand is undefined. Returning to the `subp` example above, the use of LPF allows the or-expression to return true even when the second operand is undefined, because under exactly the same conditions, the first operand is true.

The most far-reaching consequence of this definition of logical operators to handle undefinedness is that some laws of classic logic do not hold in LPF. In particular, the law of the excluded middle (that ‘`A or not A`’ is always true) is not valid in LPF. An operator d is introduced to LPF to express definedness: δ(`A`) is true when `A` is Boolean. The theorems of classic predicate logic that are not also theorems of LPF can be transformed to theorems of LPF provided “δ(`A`)” hypotheses are added. The practical consequence of this theorem is that, in performing proofs about VDM models, definedness has to be proved only where it is necessary.

LPF provides a natural way of handling undefinedness in reasoning about VDM models. The most comprehensive account of the application of LPF (10) is a product of the development of the Mural proof support system (14). Unlike Mural, many automated proof support tools focus on supporting classic logics as a priority. Indeed, the attempts to develop proof support systems for VDM have so far stopped short of implementing LPF, instead generating proof obligations to ensure that uses of partial operators are protected. Although LPF provides an intuitive approach in proof-theoretic terms, it is hard to implement in a classic interpreter: operators such as disjunction, as shown above, are symmetric, so implementation requires parallel evaluation of both operands in case one evaluates but the other is undefined. For this reason, the VDMTools interpreter takes a left-to-right lazy evaluation approach (in the case of disjunction, for example, the second operand is not evaluated if the first evaluates to true). Other formal methods take different approaches to dealing with the risk of undefined terms, and these methods are briefly discussed below.

RELATED APPROACHES

VDM is one of a family of model-oriented formal methods that includes Z (25), the B method (26), and the RAISE method and specification language (27,28). All four approaches share the abstract and rigorous modeling of data and functionality, but they differ in their focus. Z emphasizes the description of properties of data and functions, rather than giving direct definitions, which makes it particularly well suited to abstract system specification. A distinctive feature is the schema calculus, which allows for composition of specifications from structural units (schema). B is focused more on the refinement-based development of program code from abstract specifications (given in the Abstract Machine Notation). VDM and Z have been compared in Ref. 29, and VDM and B have been studied with respect to their potential integration (30). The RAISE method is based on the use of a “wide-spectrum” specification language, which is intended to encompass in a single formalism the features needed to move from abstract property-oriented specifications to lower level code. All of these languages share with VDM an extensive history of industrial application.

An interesting area of difference between the formalisms is in their underpinning logics, and in particular the mechanisms for handling undefined terms. The ISO Standards for Z and VDM are neutral about logic and proof theory. In Z, a range of logics have been explored [e.g., (31)] including approaches that seek to avoid undefined terms that develop altogether in the logic. In LPF, which is often used with VDM, undefinedness is handled directly in the logic itself. RAISE addresses the problem of executing expressions that contain symmetric operators with undefined operands by adopting a "conditional logic" (left to right) evaluation of formulae that may contain undefined terms (as does the VDMTools interpreter) (32).

CURRENT TRENDS AND OPEN QUESTIONS

From the original work on programming language and compiler design, the goal of research on VDM has been the development of a usable formal method. This principle influenced the standardization of the VDM-SL semantics, the pragmatic approach to tools, and the principles of lightweight application that have influenced industrial use. The same principle has guided foundational research on the modeling of concurrency, logics, and reification. Many research questions surrounding VDM result from practical need. It is possible to identify several significant trends that are setting the agenda for contemporary research, particularly in the areas of modeling, semantics, proof support, and tools frameworks.

VDM-SL and VDM++ can describe a very wide range of computing system. However, their ease of use is hindered in some application areas by an absence of convenient abstractions. For example, no current support exists for modeling timing characteristics unless one builds clocks and event histories explicitly into an application model. A current challenge is to include timing specifications into VDM++. This extended dialect of VDM++ is known as the VDM++ in constrained environments (VICE) dialect that has its own version of VDMTools. Studies (33–35) suggest that it is potentially valuable in the analysis of distributed embedded control software. Furthermore, VICE admits the possibility of combining discrete time models of controllers with continuous time models of controlled processes and supporting analysis through cosimulation (36). The combined model may provide a basis for improved collaboration between systems and software engineers, which is a weak aspect of many design processes for embedded systems. Aside from the goal of modeling temporal behavior, other current goals include the use of abstractions to model distributed computation and the modeling of faults and their propagation.

The construction of a dependable system requires more than simply the production of trustworthy software to run on a computer. The immediate environment involves other devices, people, and organizations. If assurance is to be gained that a computer-based system (rather than merely the software) is to function as required, then it is appropriate to model and analyze this wider system. A current research goal in the area of control-system design (37) is to provide support for reasoning about this "whole system" view. The contractual character of implicit specifications using preconditions/postconditions, and rely/guarantee conditions is potentially useful here.

Tool support is vital to the successful adoption of formal methods. The increasing power of static analysis, theorem proving, and model checking is making it possible to integrate a range of analysis tools with model generation and editing facilities. Open-platform technologies such as Eclipse (Portland, OR) have the potential to promote interoperability between separately developed specialized analysis tools, in contrast to the tightly coupled architecture of the current VDMTools.

Within the field of proof support, a current goal is to develop implementations of LPF suitable for automatic background discharging of proof obligations and validation conjectures. User guidance will remain a necessity, and so a open question is how to provide a good interface that allows a user to marshal automated tools in support of a structured proof. Such user-guided proof may even be desirable if proof is properly used as a means of exploring models rather than simply a form of automated check. In the longer term, a major research challenge is the development of theories to support reasoning about stochastic properties, such as failure behavior, that affect overall system functionality.

The Vienna Development Method is one of the seminal formal methods for computer systems development. Its evolution since the 1970s has been marked by the development first of sound foundations in the form of semantics and proof theory, then by strong tool support and industrial application. A major trend since the 1990s has been the work to lower the barrier to using the technology by developing tools that support forms of analysis that are already familiar to systems developers, such as high-coverage testing. Current work towards more cost-effective automatic proof and static analysis, more open tools frameworks, and the ability to model and analyze real-time and distributed systems, is likely to increase both the range and quality of applications of this and other formal methods.

BIBLIOGRAPHY

1. D. Bjørner, C. B. Jones (eds.), The Vienna development method: the meta-language, in *Springer-Verlag Lecture Notes in Computer Science*, Vol. 61, 1978.
2. C. B. Jones, *Systematic Software Development using VDM,* 2nd edition, Englewood cliffs, NJ. Prentice-Hall International, 1990.
3. C. B. Jones, Scientific decisions which characterize VDM, in J. M. Wing, J. C. P. Woodcock, and J. Davies (eds.), FM′99 – *Formal Methods, Lecture Notes in Computer Science*, Vol. 1708, New York: Springer-Verlag, 1999, pp. 28–47.
4. J. Fitzgerald and P. G. Larsen, *Modelling Systems: Practical Tools and Techniques in Software Development*, Cambridge, UK: Cambridge University Press, 1998.
5. J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs for Object-oriented Systems*, New York: Springer-Verlag, 2005.

6. ISO/IEC 13817-1:1996, Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language.

7. C. B. Jones, Accommodating interference in the formal design of concurrent object-based programs, *Formal Meth. Sys. Des.*, **8**(2): 105–122, 1996.

8. I. J. Hayes, C. B. Jones, Specifications are not (necessarily) executable, *Softw. Engineer. J.*, **4**(6): 330–338, 1989.

9. N. E. Fuchs, Specifications are (preferably) executable, *Softw. Engineer. J.*, **7**(5): 323–334, 1992.

10. J. Bicarregui, J. S. Fitzgerald, P. A. Lindsay, R. Moore and B. Ritchie, *Proof in VDM: A Practitioner's Guide*, New York: Springer-Verlag, 1994.

11. R. Bloomfield, P. Froome, and B. Monahan, SpecBox: a toolkit for BSI-VDM, *SafetyNet* **5**: 4–7, 1989.

12. N. Plat and P. G. Larsen, An overview of the ISO/VDM-SL standard, *Sigplan Notices*, **27**(8): 76–82, 1992.

13. R. Elmstrøm, P. G. Larsen, and P. B. Lassen, The IFAD VDM-SL toolbox: a practical approach to formal specifications, *ACM Sigplan Notices*, **29**(9): 77–80, 1994.

14. C. Jones, K. Jones, P. Lindsay, and R. Moore, *Mural: A Formal Development Support System*, New York: Springer-Verlag, 1991.

15. J. S. Fitzgerald and P. G. Larsen, Triumphs and challenges for the industrial application of model-oriented formal methods, in T. Margaria, A. Philippou, and B. Steffen (eds.) *Proc. 2nd Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, 2007.*

16. P. G. Larsen, J. S. Fitzgerald and T. M. Brookes, Applying formal specification in industry, *IEEE Softw.*, **13**(3): 48–56, 1996.

17. A. Hall, Using formal methods to develop an ATC information system, *IEEE Softw.*, **12**(6): 66–76, 1996.

18. B. W. Boehm, *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

19. J. S. Fitzgerald and C. B. Jones, Proof in the analysis of a model of a tracking system, in J. C. Bicarregui (ed.), *Proof in VDM: Case Studies*, New York: Springer-Verlag, 1998, pp. 1–29.

20. P. G. Larsen and W. Pawlowski, The formal semantics of ISO VDM-SL, *Comp. Stand. Interf.*, **17**(5-6): 585–602, 1995.

21. P. G. Larsen and B. S. Hansen, Semantics for underdetermined expressions, *Formal Asp. Comput.*, **8**(1): 47–66, 1996.

22. C. B. Jones and K. Middelburg, A typed logic of partial functions reconstructed classically, *Acta Informat.*, **31**(5): 399–430, 1994.

23. C. B. Jones, Reasoning about partial functions in the formal development of programs, *Elect. Notes Theoret. Comput. Sci.*, **145**: 3–25, 2006.

24. J. S. Fitzgerald, The typed logic of partial functions and the Vienna Development Method, in D. Bjørner and M. C. Henson (eds.), *Logics of Specification Languages*, New York: Springer-Verlag, 2008, pp. 427–461.

25. J. Woodcock and J. Davies, *Using Z: Specification, Refinement and Proof*, Englewood Cliffs, NJ: Prentice-Hall, 1996.

26. J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge, UK: Cambridge University Press, 1996.

27. The RAISE Language Group, *The RAISE Specification Language*, BCS Practitioner Series, Englewood Cliffs, NJ: Prentice Hall, 1992.

28. The RAISE Method Group, *The RAISE Development Method*, BCS Practitioner Series, Englewood Cliffs, NJ: Prentice Hall, 1995.

29. I. J. Hayes, C. B. Jones, and J. E. Nicholls, Understanding the differences between VDM and Z, *ACM Softw. Engineer. News*, **19**(3): 75–81, 1994.

30. J. Bicarregui, B. Matthews, B. Ritchie, and S. Agerholm, Investigating the integration of two formal methods, *Formal Aspec. Comput.***10**(5-6): 532–549, 1998.

31. M. C. Henson, M. Deutsch and S. Reeves, Z Logic and its applications, in D. Bjørner and M. C. Henson (eds.), *Logics of Specification Languages*, New York: Springer-Verlag, 2008, pp. 489–596.

32. C. George and A. E. Haxthausen, The logic of the RAISE specification language, in D. Bjørner and M. C. Henson (eds.), *Logics of Specification Languages*, New York: Springer-Verlag, 2008, pp. 349–399.

33. P. Mukherjee, F. Bousquet, J. Delabre, S. Paynter and P. G. Larsen, Exploring timing properties using VDM++ on an industrial application, in J. C. Bicarregui and J. S. Fitzgerald (eds.), *Proc. Second VDM Workshop*, 2000. Available at http://www.vdmportal.org

34. M. Verhoef, P. G. Larsen, and J. Hooman, Modeling and validating distributed embedded real-time systems with VDM++, in J. Misra, T. Nipkow, and E. Sekerinski (eds.), *FM 2006: Formal Methods*, Lecture Notes in Computer Science, Vol. 4085, New York: Springer-Verlag, 2006, pp. 147–162.

35. M. Verhoef and P. G. Larsen, Interpreting distributed system architectures using VDM++ – a case study, in B. Sauser and G. Muller (eds.), *Proc. 5th Annual Conference on Systems Engineering Research*, 2007. Available at http://www.stevens.edu/engineering/cser/

36. M. Verhoef, P. Visser, J. Hooman and J. Broenink, Co-simulation of Real-time Embedded Control Systems, in J. Davies and J. Gibbons (eds.), *Integrated Formal Methods: Proc. 6th. Intl. Conf.*, Lecture Notes in Computer Science, Vol. 4591, 2007, New York: Springer-Verlag, pp. 639–658.

37. C. Jones, I. Hayes, and M. Jackson, Deriving specifications for systems that are connected to the physical world, in C. B. Jones, Z. Liu, and J. Woodcock (eds.), *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of their 70th Birthdays*, Lecture Notes in Computer Science, Vol. 4700, New York: Springer-Verlag, 2007, pp. 364–390.

JOHN S. FITZGERALD
Newcastle University
Newcastle upon Tyne,
United Kingdom

PETER GORM LARSEN
Engineering College of Aarhus
Aarhus, Denmark

MARCEL VERHOEF
CHESS
Haarlem, The Netherlands

VISUAL PROGRAMMING LANGUAGES

INTRODUCTION

Since the advent of modern digital computers in the 1940s, diagrams have played a role in software development. Initially, they were paper-based aids, used by programmers to design and understand the structure of their programs, but as hardware became more powerful and input-output devices such as cathode-ray tube displays, light guns, and tablets became available, researchers began to investigate the direct use of diagrams in the design and coding of software. The arrival of high-quality, relatively low-cost graphics in the 1980s enabled graphical operating systems, making personal computers (PCs) more accessible, greatly accelerating their adoption. The immediate beneficiaries of this innovation were end users, who could now interact with operating systems and applications by directly manipulating concrete visual representations provided by graphical user interfaces (GUIs). Software developers, however, were less fortunate. In addition to writing programs to implement the core functionality of applications, they now had to use the same textual languages and tools to deal with the complexities of GUI programming.

The lack of adequate development tools, together with the availability of low-cost graphics, heightened interest in the direct use of diagrams in software development, leading to research on various fronts, including visual software project management tools, such as those found in integrated development environments (IDEs), visual editors for GUI creation, visual tools for software modeling and engineering (1), and visual programming languages.

A *Visual Programming Language* (VPL) is a language in which significant parts of the structure of a program are represented in a pictorial notation, which may include icons, connecting lines indicating relationships, motion, color, texture, shading, or any other nontextual device. Although text may occur in the pictorial notation, its role should be secondary, naming program entities for example. This definition requires significant parts of program structure to be represented pictorially, but it does not rule out languages that also express other parts of the structure textually.

VPLs are motivated by the observation that in traditional languages, multidimensional program structures and data are coded into strings, requiring an extra layer of syntax, and that by expressing the structure of programs and data pictorially, a more concrete representation of those structures might be achieved, making it easier to build, debug, understand, and reason about programs.

Our focus in this article is on visual *programming* languages, that is, languages for expressing algorithms. Consequently, although visual tools are widely used in other aspects of software development, specification, modeling, metamodeling, and so forth, we do not cover them here.

VPL EXAMPLES

Like a textual programming language, a VPL can be classified in various ways; for example, by the programming model on which it is based, the target user, whether it is general-purpose or domain-specific, declarative or imperative, or whether programs are constructed directly or by demonstration. Like textual programming languages, VPLs exist in their own, perhaps not quite so extensive, Tower of Babel: hence, a small selection of examples will inevitably exclude some features. Here we provide examples, each presented under a heading that emphasizes a particular characteristic, but chosen to illustrate a reasonably broad range of VPL features.

Data Flow

In data flow VPLs, computation at the lowest level is specified by graphs consisting of icons that represent operations on data, connected by lines representing the flow of data between operations.

The earliest implemented data flow VPL that we are aware of was due to W. Sutherland in 1966 (2). In the 1970s and early 1980s, data flow was adopted as a hardware model in projects aimed at building computers for parallel processing. Data flow diagrams were used as programming aids, but in only one project were they used directly, incorporated in a primitive data flow VPL (3). Just as these hardware projects were dying out in the 1980s for lack of viable products, graphical PCs began to appear, prompting researchers and industry practitioners to further investigate data flow as a viable model for visual programming. Data flow is the model most frequently used as the basis for industrial VPLs. We briefly describe two of them.

Prograph is a VPL intended, like Java or C++, for general-purpose application development (4). Figure 1 depicts a Prograph method quicksort that implements the *quicksort* algorithm for sorting lists. The method consists of a sequence of two data flow diagrams, implementing the base case and recursive cases of the algorithm, which are shown in the two windows labeled 1:2 quicksort and 2:2 quicksort. The input list flows from the caller into the base-case diagram through the *root* on the *input bar* at the top of the diagram and into the *match* operation named (). The match compares the incoming value with its own value [the empty list, ()]. If this comparison succeeds, the incoming empty list flows to the *terminal* on the *output bar* and is passed to the caller. If the match fails, the *next-case-on-failure* control ☒ fires, terminating execution of the case and initiating execution of the recursive case. The input list flows into the built-in operation detach-l, which divides the list into its first element (head) and the remaining list (tail), which flow out of the two roots on the bottom of the operation. The partition operation compares the head with each of the elements in the tail, dividing the latter into the lists of

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

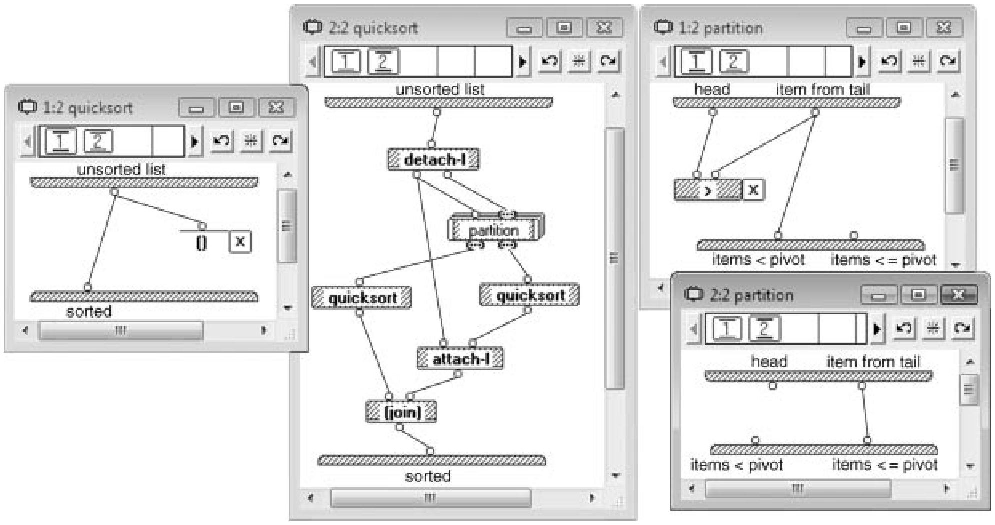


Figure 1. Prograph implementation of the quicksort algorithm for list sorting.

elements less than and greater than or equal to the head. These two lists are sorted by recursive calls to quicksort, the head of the original list is attached to the beginning of the second sorted list, and the resulting list is joined to the end of the first sorted list.

The annotations on the partition operation indicate its repetitive nature. In particular, the list annotation (. . .) on the second terminal requires the input to be a list, each element of which will be consumed by an execution of partition. The same annotation on the roots indicates that lists will be produced, consisting of the values produced by the individual executions of partition. We invite the reader to complete the explanation of this example by examining the diagrams for partition. Note that if there is no data link attached to a terminal on an output bar, no value for that output will be returned to the caller.

Figure 1 illustrates the above comment about concrete representation of program structure in visual languages. For example, the two invocations of quicksort in the recursive case of the algorithm are independent and could be executed in parallel. Representing the computation in a data flow diagram, as in the second case of the quicksort method, clearly exposes this characteristic of the algorithm.

Prograph is an example of a *structured* data flow VPL, in which data flow diagrams are embedded in structures that control how they are invoked. For example, the list annotations on the roots and right-hand terminal of the partition operation in Figure 1 define an iterative, possibly parallel, list-processing construct.

In structured data flow VPLs, diagrams are acyclic, and in each invocation of a diagram, each data link is assigned a value at most once. In contrast, in the unstructured model on which the data flow hardware projects were based, a program consists of a single data flow graph in which iteration is achieved via cycles, and conditional execution by special operations that route values along different data links depending on Boolean input. Because of the difficulty of building and understanding the control structures required for nontrivial programs, unstructured data flow is generally limited to application domains in which cyclic data flow is natural. For example, *Simulink*, a domain-specific, data flow VPL for simulating dynamic systems, is primarily unstructured, although it also provides some control constructs. Figure 2 shows a Simulink program that simulates a bouncing ball by continuously recomputing its velocity and position in a feedback loop consisting of two cycles.

Imperative Programming

Imperative VPLs focus on the flow of control rather than on the flow of data. The earliest example of an imperative visual notation is possibly von Neumann and Goldstine's *Flow Diagrams*, essentially the familiar unstructured flowcharts, consisting of computational blocks connected by lines indicating transfer of control.

Imperative VPLs not based on flowcharts have also been designed. An example is VIPR, a language in which control flow is expressed via containment and connectivity (5). A statement is represented by a circle that may have one or both of a *guard* and an *expression* attached. If the guard (a logical expression) evaluates to true, the expression is

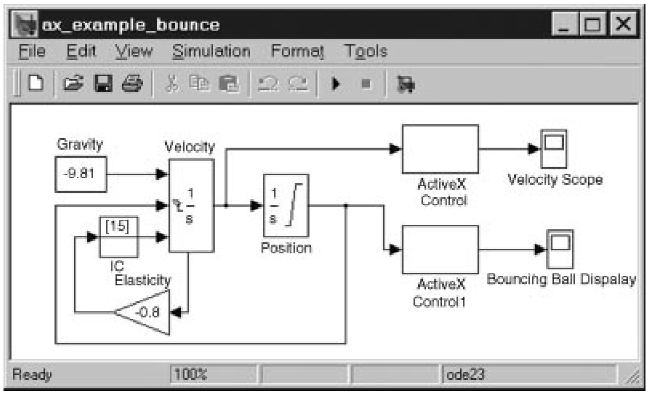


Figure 2. A Simulink program for simulating the motion of a bouncing ball.

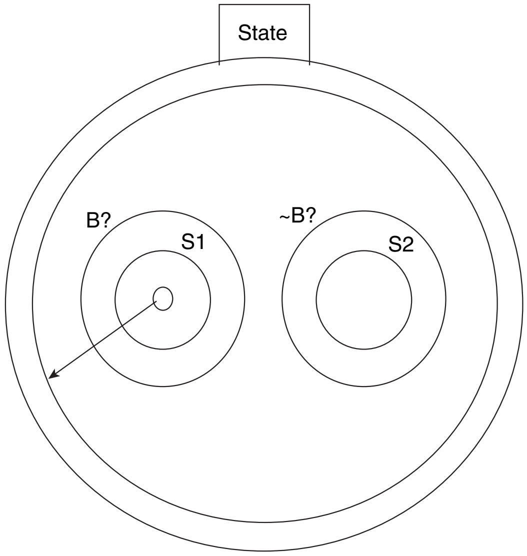


Figure 3. The loop `while` B `do` S1; S2 in VIPR (from Ref. 5).

evaluated. A sequence of statements is represented by concentric circles, which are executed from outside to inside and peeled away as execution proceeds. Circles are also connected in various ways to indicate other kinds of control flow. Figure 3 depicts the form of a VIPR while loop, represented by the circle inside the outer "container" circle. When this statement is executed, the guards on the enclosed statements are executed. If B evaluates to true, the left-hand statement expands to fill the container; the statement inside it is executed, causing the evaluation of S1; and the circle inside that executes, replacing itself with a new copy of the loop structure.

Note that in VIPR, the relationship between a statement and the control structure it invokes is explicitly indicated by a line connecting the two, as illustrated in Fig. 3, in contrast to most VPLs. For instance, in the Prograph example in Fig. 1, the relationship between the partition operation and its diagrams is indicated symbolically.

Imperative VPLs with static representations have not proved to be as useful as data flow. However, VPLs in which the user demonstrates imperative algorithms rather than writing them explicitly have had some success. One such language is *ToonTalk*, which is meant for children and presents the task of programming as direct manipulation of objects in a video game environment (6). A program is represented as a city, in which houses represent processes. The programmer enters a house where he or she trains robots to perform tasks using a small number of tools and assembles them into teams that perform procedures.

There is no static representation of "code" in ToonTalk, so it is not possible to provide diagrams for an example program; however, we will outline the process of building a program for computing factorial, illustrated with some snapshots. First, as shown in Fig. 4(a), the programmer takes a new robot from the toolbox, names it fact start, creates a sample input in the form of a *box* with two slots, and drops it on the robot to initiate training. The sample input slots, labeled N and Answer, respectively contain the input integer, and a *bird* representing a pointer to the location to which to return the result (the bird's *nest*). The environment contains a *toolbox* with an unlimited supply of objects for building programs; a *notebook* for storing any kind of item; a *magic wand* for copying; a *dust-buster* for deleting; and a *pump* for enlarging or shrinking. The programmer is represented by a hand for grasping and moving objects.

Dropping the input on the robot causes an animated transition into the robot's thought bubble, where the robot's task is demonstrated. The final step of this training is shown in Fig. 4(b). The robot has been taught to append to the right end of the box a slot named Factorial containing 1, and two slots, labeled I and must be less, to the left end. Appending boxes is accomplished by juxtaposing them, whereupon a mallet-bearing mouse performs the operation. The slot labeled must be less contains a *balance*, tipped to the right to show how the numbers on either side compare. Terminating training causes an animated return to the environment. At this point, the robot's thought bubble contains a copy of the training input, indicating that this is the only input he will accept. The programmer generalizes this by erasing the 3 from the N slot leaving a green patch indicating that any integer is acceptable, as in Fig. 4(c).

In a similar fashion, the programmer trains a robot named fact loop to accept any five-slot box with integers in the first, third, and fifth slots, and a right-tilted balance in the second, to increment the first integer and to multiply the last integer by the result. Note that given an input, a robot repeats its task until the the input is no longer of the right form, so fact loop will stop when the integers in the first and third boxes are equal. Finally, the programmer trains a robot named fact finish to accept any five-slot box with integers in the first, third, and fifth slots, to remove the last integer from its slot, drop it into the fourth slot on to the bird, which carries the integer to its nest, and clean up by selecting a bomb from the toolbox and detonating it.

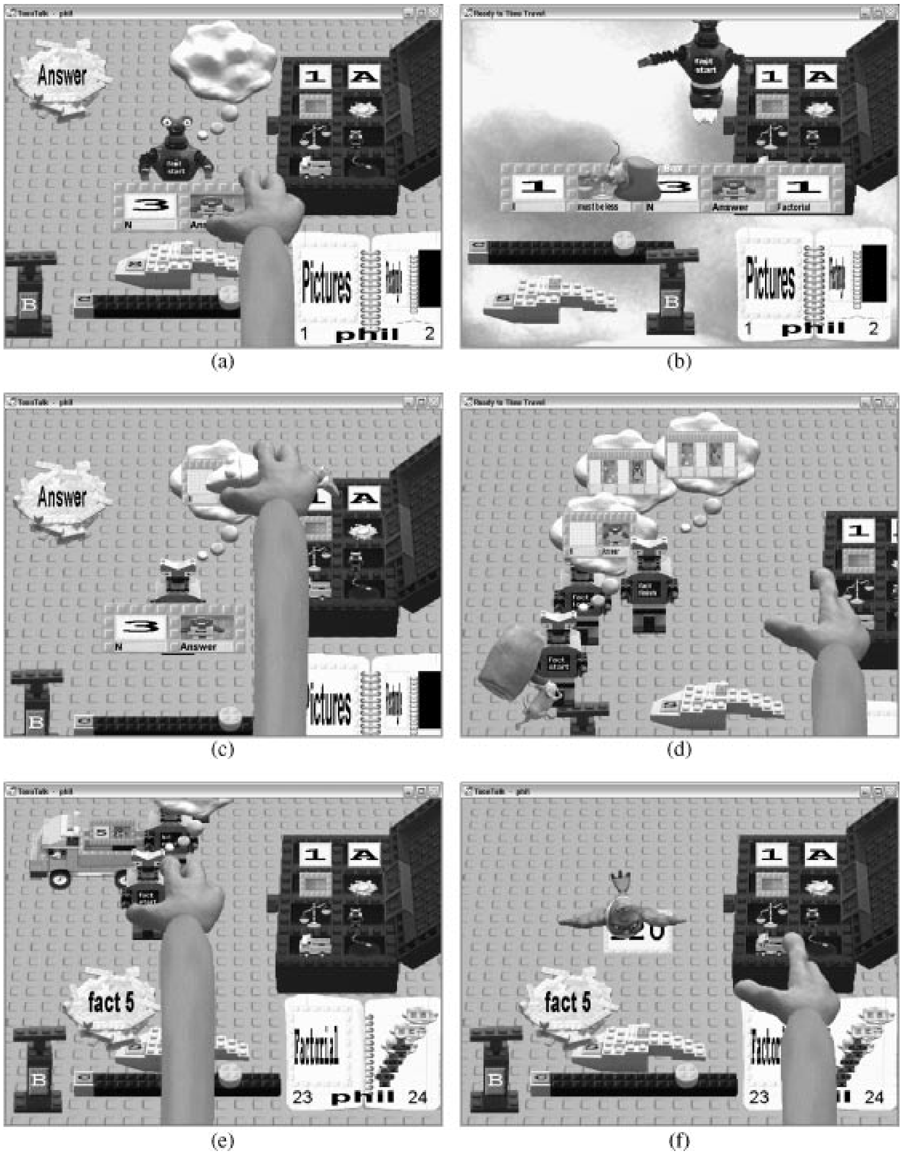
Next, the three trained robots are assembled into a *team*, as shown in Fig. 4(d), with the acceptable input patterns shown in their thought bubbles. A team is a sequence of robots that, when given an input, act on it in order. The team is stored in the notebook.

Finally, the programmer creates a two-slot input box containing bird and integer, takes a truck from the toolbox, and places the box on it, together with a copy of the robot team from the notebook [Fig. 4(e)]. The truck goes to a new house, where the robots act on the provided box, finally blowing up the house to terminate the spawned process. The bird returns the result [Fig. 4(f)].

Although it is not possible to see a program, or to edit a chosen part of it, the environment provides a "time travel" feature that allows the programmer to record the programming, replay from any point, and take control at any point to redo the programming from that point on.

Sheet-Based

Spreadsheets, although introduced before the advent of the graphical PCs that enabled current VPL research, are almost certainly the most popular VPLs. This is possibly



(a) (b)

(c) (d)

(e) (f)

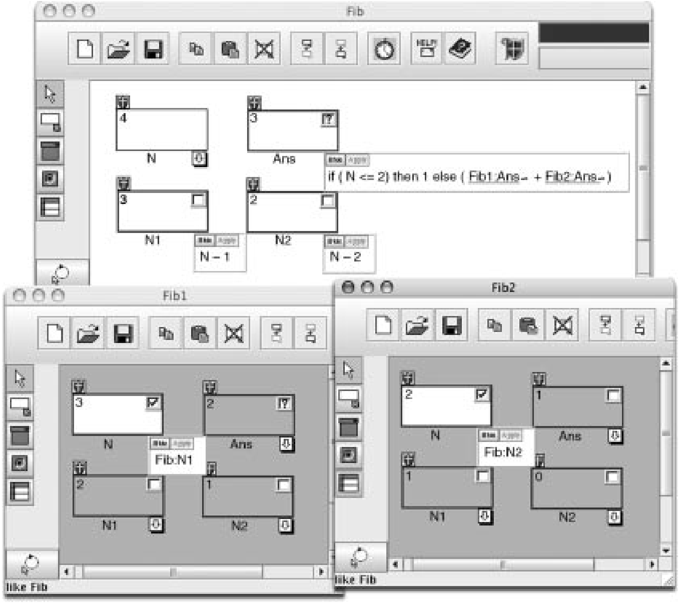
Figure 4. Programming factorial in ToonTalk.

because they are relatively simple, presenting a ledger-like sheet for entering and performing arithmetic on values, a metaphor familiar to the end users for whom spreadsheets are intended. The sheet is the single significant pictorial element that qualifies the original spreadsheet as a VPL, according to our definition, since the implicit acyclic data flow graph, created by formulas in cells referring to other cells, is not visible. Nevertheless, evidence exists that the visual characteristics of the sheet help users build a mental representation of the underlying data flow (7). Spreadsheets have evolved to include more pictorial elements, such as the auditing feature of Microsoft Excel (Microsoft Corporation Redmond WA), which allows the data flow graph to be displayed.

Numerous enhancements to spreadsheets have been proposed. Some are minor refinements of the original idea, such as allowing a spreadsheet program to consist of several sheets that may reference each other, thereby providing a simple type of modularization. Others are aimed at enhancing the programming aspects by adding to or replacing the "formula-in-cell" data flow model, which is not Turing complete.

In the language *Forms/3*, sheets are generalized to forms (8). A Forms/3 program consists of a collection of *forms* containing cells, which are analogous to spreadsheet cells but can be arranged arbitrarily. As in standard spreadsheets, cells contain formulas which may refer to cells on the same form or other forms.

Forms/3 attains Turing completeness by providing recursion and iteration. Recursion is illustrated by the example in Fig. 5, which depicts a collection of forms that compute Fibonacci numbers, constructed as follows. The form named Fib with four cells is constructed first, and formulas 4, N–1 and N–2 entered in the cells N, N1, and N2 respectively, resulting in the values shown in those cells. This form is copied twice, producing the forms Fib1 and



if (N <= 2) then 1 else (Fib1:Ans~ + Fib2:Ans~)

N – 1

N – 2

Fib:N1

Fib:N2

Figure 5. Computing Fibonacci numbers in Forms/3.

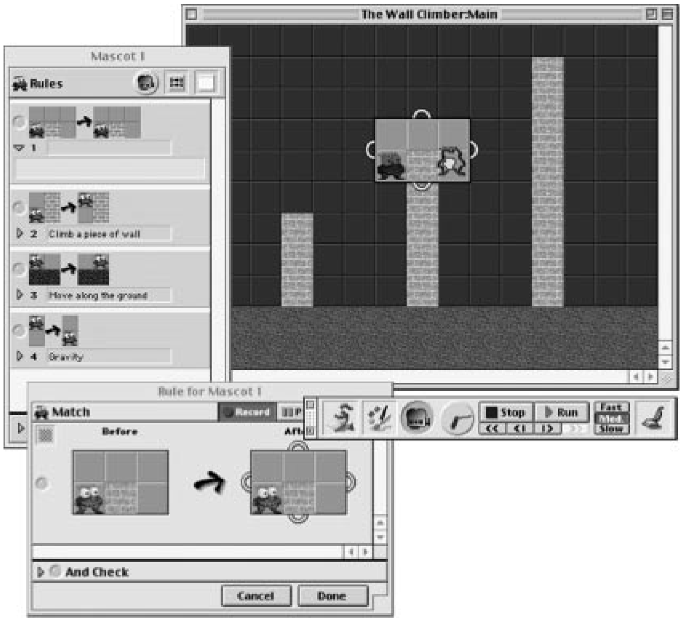


Figure 6. Defining a rule in KidSim.

Fib2, which inherit structure and content from Fib. Their gray background and the annotation like Fib in their bottom left corners show that they are copies. Next, the formula in the N cells of Fib1 and Fib2 are replaced by Fib:N1 and Fib:N2, respectively, referring to the N1 and N2 cells of the Fib form. The edited cells' values are recomputed, and their backgrounds turn white to show that their contents are no longer inherited. Finally, a formula is entered into the Ans cell of the Fib form, as shown, computing a value in terms of the values of the Ans cells of the copies. Clearly, if this formula were propagated directly to the copies, the underlying data flow graph would become cyclic. However, when a formula is propagated from a form A to a copy B, a reference to a cell on a copy C of A is interpreted as a reference to a copy D of B, bearing the same relationship to B as C does to A. When propagated formulas are evaluated, copies are created as required. Hence, in our example, as soon as the formula in the Ans cell of Fib is entered, two copies of Fib1 are created to compute Fibonacci numbers corresponding to the contents of the N1 and N2 cells of Fib1. Note that the base case of the propagated Ans formula applies in each of these copies and in Fib2, so no further forms are required.

Iteration in Forms/3 is achieved by introducing the notion of time as a sequence of "clock ticks" and functions that can be used in formulas to recompute the value of a cell at each tick, and to refer to a cell's value at an earlier tick.

Two-dimensional rectangular grids of cells are also used in VPLs in which visual transformation rules are used for programming games and simulations. One VPL of this genre is *KidSim* (later named Cocoa, then Stagecast Creator), a programming environment for children (9). In KidSim, each cell in the grid may contain an agent, and each type of agent may have a sequence of rules describing its behavior. In the example in Fig. 6, there are three types of agents, "ground," "wall," and "mascot," populating the grid. The window on the left shows the rules for mascot, each defining, by means of "before and after" pictures, how the grid is transformed when certain relationships between adjacent grid cells are met. Rules 2 to 4, respectively, will cause the mascot to move right if above a ground cell and left of an empty cell, and to fall if the cell below is empty. The window at bottom left, and the highlighted region of the grid window, show a new rule being defined: the user is dragging the mascot agent to an new position in the highlighted region to demonstrate how the adjacent cells with the given "before" configuration should be transformed. After the rule is constructed, clicking the "Run" button on the control palette starts the simulation, during which the rules for each agent in the grid are applied at each time tick. The rules for an agent are tried in the order they occur until one is found with a "before" pattern that matches the region around the agent.

Declarative

Declarative VPLs, like their textual counterparts, are descriptive rather than prescriptive; that is, the programmer defines relationships between values rather than providing instructions for computing one value from another. Common declarative models are functional programming and logic programming. *Moment* is a simple, domain-specific, declarative language for defining two-dimensional scenes (10). Each graphic object in a scene corresponds to a function represented by an icon in a *scene graph* diagram. The scene graph may also include functions that perform computations but do not represent objects in the scene. Function icons are connected by data flow links, defining constraints to be enforced when objects in the scene are manipulated.

At the top of Fig. 7 is a Moment scene graph that defines the scene at bottom right, demonstrating Varignon's

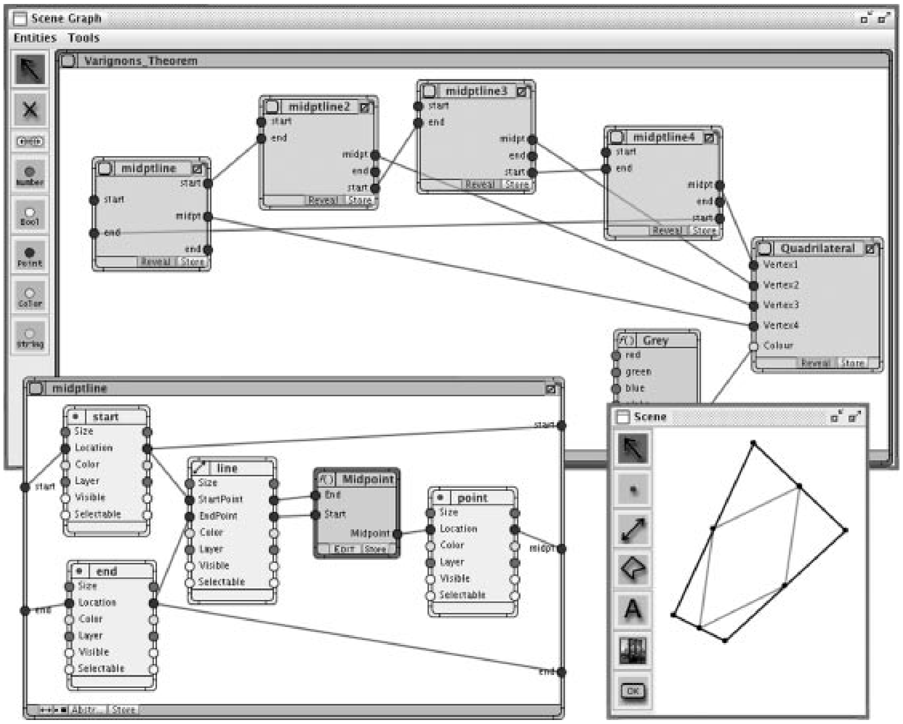


Figure 7. Defining a drawing in Moment.

Theorem: as the sides or vertices of the black quadrilateral in the scene are dragged, the gray quadrilateral deforms but remains a parallelogram. The Scene Graph window contains an *enclosure* called Varignons_Theorem, corresponding to the content of the scene graph. This enclosure contains four instances of a function midptline, defined by the enclosure at bottom left, connected by links that constrain their endpoints to form the black quadrilateral. The gray quadrilateral is represented in the graph by a function Quadrilateral connected by links that constrain its vertices to the midpoints of the lines. We leave it to the reader to determine the structure of the lines that make up the black quadrilateral by inspecting the midptline enclosure.

In *L-sheets*, a recently proposed enhancement to spreadsheets, the standard formula-in-cell data flow programming model is augmented with a logic-based VPL to obtain Turing completeness, and to provide a means for specifying spreadsheet structure (11).

L-sheets is based on the observation that specifying array structure, relationships between arrays and computations that fill arrays, key activities in spreadsheet design, are analogous in logic programming to specifying term structure, relationships between terms and computations that bind variables occurring in terms. Accordingly, L-sheets incorporates a form of logic programming based on unification of visually represented arrays.

An L-sheets *program sheet* consists of a set of *definitions*, each composed of a sequence of *cases*. Figure 8 depicts a program sheet with three definitions, gauss, triangularise, and backsubstitute, which together define Gaussian elimination with partial pivoting for solving simultaneous linear equations. A *case* is analogous to a Prolog clause and consists of a pale gray *head* template, followed by a sequence of dark gray or white *body* templates. In our example, gauss has one case, whereas triangularise and backsubstitute each consist of two. A *template*, analogous to a Prolog literal, has a *name* and sequence of *parameters*, each of which is an *array*, that is either basic or compound.

A *compound* array is a rectangle further subdivided into arrays. A *basic* array is a rectangle of cells, and can have either one or a variable number of rows and one or a variable number of columns. For example, the first parameter in the head of the second case of triangularise consists of three arrays labeled A, B, and C. A consists of two horizontally juxtaposed basic arrays, the left-hand one having one column and a variable number of rows, indicated by the horizontal dotted line. The right-hand basic array in A has variable numbers of rows and columns. The content of a basic array is a variable that may be bound to a set of spreadsheet-style formulas.

The user applies the definition gauss by selecting in the worksheet two rectangular arrays of cells. These arrays are unified with the two parameters in the head of the single case of gauss. The worksheet array bound to A contains the coefficients and the right-hand sides of the equations, and must be n rows by $n + 1$ columns, otherwise, the application of gauss will fail. The worksheet array bound to C must be a single row of length n, for the solution vector. When these unifications have succeeded, execution proceeds in a fashion analogous to Prolog; that is, templates are introduced by the body of the case and executed in order.

The base case of triangularise applies if the coefficient array consists of a single row and two columns,

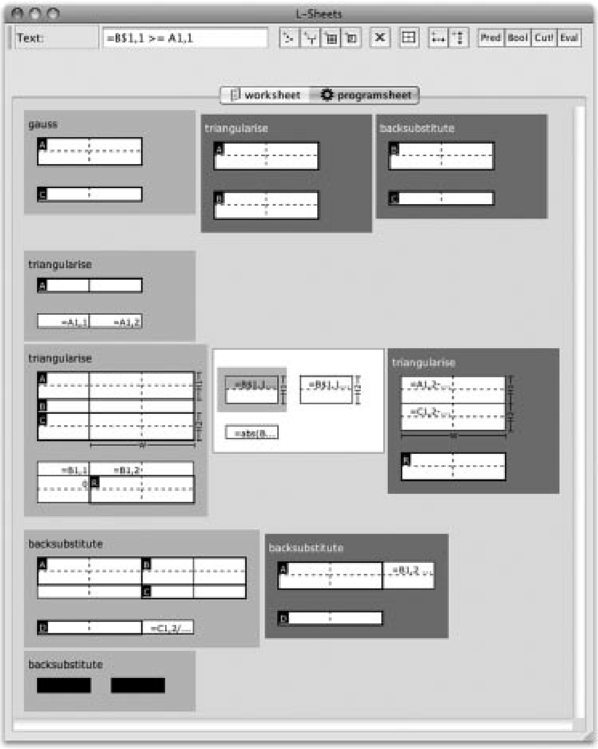


Figure 8. An L-sheets program defining Gaussian elimination for solving simultaneous linear equations.

corresponding to one equation in one variable, the solution to which is computed directly by the formulas in the cells of the second parameter.

In the second case of triangularise, the first body template is a *guard*, the parameters of which have Boolean-valued expressions as contents. For example, the content of the selected parameter, displayed at the top of the window, compares the first element of the row named B of the array to be triangularised with the first element of every row above it. Similarly, the second parameter of the guard compares the first element of B with the first element of each row below it. The third parameter ensures that the absolute value of the first element of B is not zero.

The second case of triangularise illustrates that array unification is not unique. If the array to be triangularised has n rows, there are n possible unifications, generated in succession until the guard template succeeds, indicating that the pivot row has been found.

Observing that a black rectangle represents an array at least one dimension of which is zero, we invite the reader to complete the explanation of this example.

Note that, like textual logic programming languages, L-sheets is an example of a rule-based language, since each case specifies a transformation rule that may be applied to advance an execution state by transforming the structure and content of arrays.

Component Based

Various VPLs have been proposed and implemented based on the metaphor of networked computing devices, or *components*, each performing a variety of tasks in response to messages and data received from other components. An example can be found in IBM's VisualAge products, in which a component-based VPL is layered over a textual programming language (SmallTalk, Java, C++) (12).

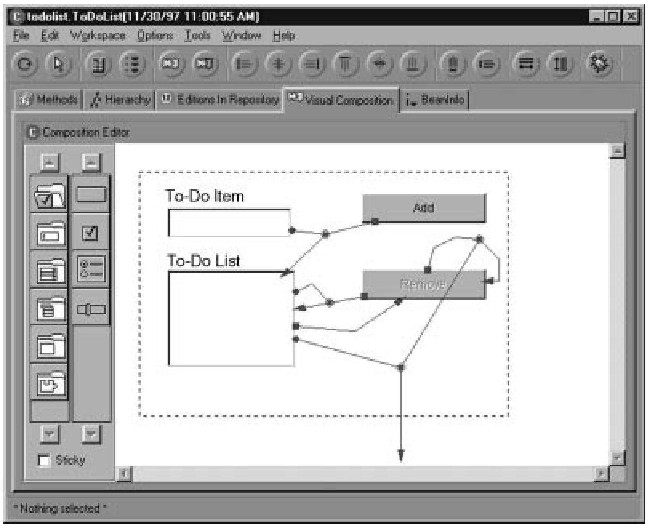


Figure 9. A message-flow diagram in VisualAge for Java.

Figure 9 depicts a VisualAge for Java message-flow diagram that defines interactions between interface components. An arrowed line starting from a square dot on a component indicates that when a particular event originates from that component, a message will be sent to the component at the head of the arrow. A line that originates from a round dot on a component indicates the retrieval of data from that component. For example, the line from the "Add" button to the "To-Do List" scrolling list indicates that, when the button is clicked, a message will be sent to the list, together with the content of the "To-Do Item" text box. Clearly, key information about such interactions is not represented in the diagram and must be viewed separately: for example, the kind of event that triggers the message, the identity of the message to be sent, and the data to be retrieved. Although for a click on the "Add" button in this example, it is clear what these items must be, it is less obvious for interactions between more complex components. Noting that the data arrow pointing out of the dotted boundary refers to a database component, we invite the reader to guess the messages, data, and events involved in the interactions between the "To-Do List" and the "Remove" button.

Component-based VPLs are well suited to programming tasks such as that illustrated in the example; however, they do not facilitate algorithms requiring iteration or conditional execution.

VPL FORMALIZATION AND CLASSIFICATION

All programs, regardless of how they are represented, must be compiled into machine-executable form. In a textual language, compiling a program involves lexical analysis, which divides the string of symbols into larger meaningful chunks (tokens), such as identifiers, operators, and keywords, then parsing the token stream to produce a parse tree. Semantic analysis is usually performed during the construction of the parse tree by using an attribute grammar, a context-free grammar in which nonterminal symbols are annotated with attributes, and productions are augmented with rules that relate these attributes and attach semantic information, such as snippets of assembly language code, to the parse tree. This semantic information is then used for code generation.

Since analogous processes are required for visual programs, the theory of VPLs has largely developed by analogy with the theory of textual languages, with two major differences. First the structure of grammars for VPLs is complicated by the difference between text and two-dimensional pictures. Text is abstract, not a one-dimensional picture, so symbols have no size or absolute position and cannot overlap. Juxtaposition in one dimension is the only relationship between symbols in a string. In contrast, an object in a two-dimensional picture is not just a symbol. It has size and shape, so it may touch, overlap, occlude, or contain another object, or be separated from another object by a vector of any length or orientation.

The other major difference between the theories of textual and visual languages is the way in which they are applied. Strings have a universally accepted representation as sequences of standard codes. Hence, any text editor can be used to write programs and to save them in generic text files. Consequently, although IDEs frequently include syntax-directed editors that either refuse to allow syntactically incorrect input, or highlight and warn about syntax errors, the primary application of grammars is to lexical analysis and parsing. In contrast, there is no universal coding for object-based drawings, so formal descriptions of VPLs tend to be used for building syntax-directed editors, rather than for parsing generic drawings (13).

Among the most widely used formalisms for describing the structure of visual languages are graph grammars, proposed in the late 1960s as a means for describing the structure of graphs (14). A *graph grammar* (GG), analogous to a transformational grammar defining a textual language, consists of a set of rules for transforming a graph by identifying a subgraph by pattern matching, and replacing it with another subgraph. Various derivatives of GGs have been devised to better suit the requirements of visual languages, such as *layered* and *reserved*. Other formalisms developed for visual languages include *positional grammars, relational grammars, relation grammars, constraint multiset grammars, attributed multiset grammars*, and *hypergraph grammars* (15).

As noted, programming languages can be classified in different ways for different purposes. The Chomsky hierarchy classifies textual languages according to syntactic structure, and what it implies about expressive power and algorithmic properties. Classifying VPLs in a similar way is problematic because of the variety of formalisms used to define them. Marriott and Meyer approach this problem by establishing a Chomsky-like hierarchy of VPLs based on copy-restricted constraint multiset grammars (CCMGs), then showing how various VPL grammar formalisms can be mapped on to CCMGs (15). Placing a VPL in this hierarchy provides insights into its limitations, and information with practical implications, such as the complexity of parsing.

Other classifications of VPLs have a more pragmatic flavor. For example, Costagliola et al. define a framework in which a VPL is classified according to the syntactic attributes of visual tokens and spatial relationships between tokens that form the alphabet of the grammar. Their classification leads to a class hierarchy (in the object-oriented programming sense), incorporated into a compiler-compiler for semiautomatically generating VPL compilers and syntax-directed editors (16).

EVALUATION AND DESIGN

In the early days of VPL research, researchers tended to invent new visual notations to explore the possibilities of the medium, assuming that the greater expressive power of pictures would inevitably lead to languages better then textual ones. Results were reported, together with claims of superiority, based largely on intuition. Amid growing concern about the need to validate such claims, Blackwell reduced them to 12 categories, related each category to the cognitive science literature, showing some to be well founded and others to have little or no support, and concluded by pointing out the need to properly account for cognitive processes in VPL design and evaluation (17).

The Cognitive Dimensions (CD), proposed by Green and Petre, has had a significant influence on VPL research, providing a framework within which visual language designers can assess the potential impact of design choices (18). The CD framework consists of the 13 dimensions listed in Table 1, each capturing a significant characteristic of a notation, and supported by empirical evidence.

To illustrate, some applications are as follows.

Abstraction gradient: In Prograph, although the programmer may code complicated computations in a single data flow diagram, as the number of operations and data flow links increases, the diagram becomes cluttered and complicated, encouraging the programmer to collapse meaningful subdiagrams into single operations.

Hidden dependencies: In the original form of spreadsheets, the only way to discover that a cell depended on others was to select it to reveal its formula. Furthermore, there was no way to find the cells dependent on a particular cell.

Progressive evaluation: Spreadsheets deliver the ultimate in progressive evaluation since execution is “always on,” evaluating all affected cells immediately after every edit. This property is preserved in some spreadsheet derivatives, such as Forms/3.

Closeness of mapping: The correspondence between problem and program in Moment is almost one-to-one. Each object in the drawing is represented by a node in the program, and each visually apparent constraint between objects (e.g., endpoints of two lines coincide) is represented by a link between program nodes.

Table 1. Cognitive Dimensions

Dimension	Description
Abstraction gradient	Degree to which abstraction is allowed, encouraged, and supported
Closeness of mapping	Does the notation closely mirror domain objects and relationships
Consistency	The notation represents similar meanings with similar constructs
Diffuseness/Terseness	Are too many or too few symbols required to represent a meaning
Error-proneness	Notation traps the programmer into making errors
Hard mental operations	Notation makes some programming tasks inherently difficult
Hidden dependencies	Significant relationships between program entities are not displayed
Premature commitment	Programming decisions precede the information required to make them
Progressive evaluation	Programs can be run at any time, complete or not
Role-expressiveness	The function of a program structure is implied by its appearance
Secondary notation	Comments or other information not formally part of the program
Viscosity	Is unreasonable effort required to make a change
Visibility	How easily can any part of a program be displayed

Consistency: If the user, knowing some of a language, can reliably guess the structure of the rest, then the language is consistent. Although it seems unlikely that a little knowledge of ToonTalk will allow the user to infer much more, once the user has understood that placing one number on another causes the mallet-wielding mouse to perform addition, he or she may very well conclude that placing other objects in close proximity will cause the mouse to perform an appropriate operation.

Viscosity: In ToonTalk, to rectify a programming error, the programmer must rewind the animation to a point preceding the error, then redo the entire program from there on. This is possible only if the "time travel" feature is turned on.

There are various ways to evaluate the effectiveness of VPLs. One unscientific but telling measure is the degree to which VPLs have been adopted in industry. Some VPLs have had considerable commercial success. For example, LabVIEW, a data flow VPL for programming hardware controllers, has several million users worldwide, and Simulink has a strong user base in the engineering industry. Although the VPLs in both products are general enough to code any algorithm, the products themselves are domain-specific. In contrast, no VPLs for general- purpose application development enjoy such success. A possible explanation is provided by a focus-group study commissioned by Apple Computer in the 1990s to determine the viability of Prograph for industrial development. Despite strongly positive feedback on the Prograph development environment, comments from participants indicated that it would be unlikely for a VPL that did not comply with prevailing software industry standards to be adopted, implying that a VPL must be able to be used interchangeably with a standard textual language such as Java.

More detailed evaluation data are obtained via user studies, reports from users with extensive experience of particular VPLs, and user surveys targeting specific questions about VPL performance. Although results have been mixed, some indicate the superiority of VPLs in certain situations. One user study testing the performance of programmers solving matrix problems in Forms/3, Pascal, and APL showed that 73% of the Forms/3 programs were correct, compared with 53% and 40% of the APL and Pascal programs, respectively (19). Based on extensive experience with LabVIEW in more than 40 projects, Baroth and Hartsough concluded that LabVIEW's VPL contributes significantly to productivity in software development (20). In an extensive survey of LabVIEW users, respondents rated its visual aspects significantly higher than its non-visual ones (21), a result consistent with Baroth and Hartsough's conclusions.

CONCLUDING REMARKS

As Ambler and Kimura noted in their preface to the proceedings of the 1994 IEEE Symposium on Visual Languages, "In 1984, the goal of visual language research was ... using pictures to construct programs and to then watch their execution." Accordingly, the initial focus was on inventing pictorial notations to explore the potential of graphics for representing algorithms and data. As VPL research has evolved, the focus has shifted and widened. Current emphases are as follows:

- Cognitive issues
  - □ Empirical studies of programmers—professionals, novices, children, and end users
  - □ Principles of notational design
- Software engineering
  - □ Theory
  - □ Software modeling
  - □ Software visualization tools
- VPLs for domain-specific and end-user programming
  - □ Languages
  - □ Debugging tools and methodologies
- Formal methods
  - □ Syntax and semantics
  - □ Generating VPLs from specifications
  - □ Diagrammatic reasoning

BIBLIOGRAPHY

1. K. Zhang, J. Kong, and J. Cao, Visual software engineering, in B. W. Wah (ed.), *Encyclopedia of Computer Science and Engineering*. New York: Wiley, 2008.
2. W. R. Sutherland, The On-Line Graphical Specification of Computer Procedures, Ph.D. dissertation, Cambridge: Massachusetts Institute of Technology, 1966.
3. A. L. Davis and S. A. Lowder, A sample management application program in a graphical data-driven programming language, *Dig. Papers, Compcon Spring*, **81**: 162–165, 1981.
4. P. T. Cox, F. R. Giles, and T. Pietrzykowski, Prograph: a step towards liberating programming from textual conditioning, *Proc. IEEE Workshop Visual Programming*, Rome, Italy, 1989, pp. 150–156.
5. W. Citrin, M. Doherty, and B. Zorn, The design of a completely visual object-oriented programming language, in M. Burnett, A. Goldberg, and T. Lewis (eds.), *Visual Object-Oriented Programming: Concepts and Environments*. Greenwich, CT: Manning Publications, 1995.
6. K. Kahn, ToonTalk$^{\mathrm{TM}}$—An animated programming environment for children, *J. Visual Languages and Computing* **7**(2): 197–217, 1996.
7. R. Navarro-Prieto and J. J. Cañas, Are visual programming languages better? The role of imagery in program comprehension. *Int. J. Human-Computer Studies* **54**(6): 799–829, 2001.
8. M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang, Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *J. Functional Programming* **11**(2): 155–206, 2001.
9. D. C. Smith, A. Cypher, and J. Spohrer, KidSim: Programming agents without a programming language. *Commun. ACM* **37**(7): 54–67, 1994.
10. S. Greenwold, *Spatial Computing*, MSc Thesis, Cambridge: Massachusetts Institute of Technology, 2003.
11. P. T. Cox and P. Nicholson, Unification of arrays in spreadsheets with logic programming, *Proc. Workshop on Practical Aspects of Logic Programming*, San Francisco, CA, 2008.
12. M. Carrel-Billiard and J. Akerley, *Programming with VisualAge for Java*. Englewood Cliffs, NJ: Prentice Hall, 1998.
13. K. Wittenburg, Relational grammars: Theory and practice in a visual language interface for process modeling, *Proc. International Workshop on Theory of Visual Languages*, Gubbio, Italy, 1996.
14. I. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation Volume 2: Applications, Languages and Tools*. Singapore: World Scientific, 1999.
15. K. Marriott and B. Meyer, *Visual Language Theory*. New York: Springer-Verlag, 1998.
16. G. Costagliola, A. Delucia, S. Orefice, and G. Polese, A classification framework to support the design of visual languages. *J. Visual Languages and Computing* **13**(6): 573–600, 2002.
17. A. F. Blackwell, Metacognitive theories of visual programming: What do we think we are doing, *Proc. IEEE Symposium on Visual Languages*, Boulder, CO, 1996, pp. 240–246.
18. T. R. G. Green and M. Petre, Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *J. Visual Languages and Computing* **7**(2): 131–174, 1996.
19. R. Pandey and M. Burnett, Is it easier to write matrix manipulation programs visually or textually? An empirical study, *IEEE Symposium on Visual Languages*, Bergen, Norway, 1993, pp. 344–351.
20. E. Baroth and C. Hartsough, Visual programming in the real world, in M. Burnett, A. Goldberg, and T. Lewis (eds.), *Visual Object-Oriented Programming: Concepts and Environments*. Greenwich, CT: Manning Publications, 1995.
21. K. N. Whitley and A. F. Blackwell, Visual programming in the wild: A survey of LabVIEW programmers. *J. Visual Languages Computing*, **12**(4): 435–472, 2001.

FURTHER READING

The following online bibliography contains an extensive listing of VPL papers and, since authors can enter their own papers, tends to be fairly current.

M. M. Burnett, (no date). Visual Language Research Bibliography. [Online]. Oregon State University. Available: http://www.cs.orst.edu/~burnett/vpl.html, December 12, 2007.

Although VPL research results now appear in many journals and conferences, the primary sources are:

S.-K. Chang and S. Levialdi (eds.), *Journal of Visual Languages and Computing*, Elsevier.

Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE Computer Society Press.

A comprehensive discussion of empirical studies on the use and value of diagrams in programming can be found in:

A. F. Blackwell, K. N. Whitley, J. Good, and M. Petre, Cognitive Factors in Programming with Diagrams, *Artificial Intell. Rev.* **15**: 95–114, 2001.

The following paper provides a survey of the development of data flow languages including a section on data flow VPLs:

W. M. Johnston, J. R. Hanna, and R. J. Millar, Advances in dataflow programming languages, *ACM Computing Surv.* **36**(1): 1–34, 2004.

The following journal special issue reports on experiences with Green and Petre's cognitive dimensions framework.

A. F. Blackwell (ed.) Special issue: ten years of cognitive dimensions: *J. Visual Languages and Computing* **17**(4): 285–287, 2006.

An earlier survey of VPL research can be found in:

M. M. Burnett, Visual programming, in J. G. Webster (ed.), *Encyclopedia of Electrical and Electronics Engineering*. New York. Wiley, 1999.

A thorough overview of visual languages, including underlying theory and applications to a variety of areas, is provided by:

K. Zhang, *Visual Languages and Applications*. New York: Springer, 2007.

The following work consists of a collection of chapters by different authors on specific visual language topics, providing an insight into some of the current research in the area.

F. Ferri (ed.), *Visual Languages for Interactive Computing: Definitions and Formalizations*. Hershey, PA: IGI Global, 2007.

PHILIP T. COX
Dalhousie University
Halifax, Nova Scotia, Canada

VISUAL SOFTWARE ENGINEERING

INTRODUCTION

Graphical notations are widely used in software design and development. These notations can greatly help the modeling and representation of software architecture (1) and design (2). There are many benefits of informal graphic notations: First, they can be used to convey complex concepts and models, such as object-oriented design. Notations like those in UML (2) serve a useful purpose in communicating designs and requirements. Second, they can help people grasp a large amount of information more quickly than text can. Third, as well as being easy to understand, drawing diagrams is normally easier than writing text in a predefined language. Fourth, graphical notations cross language boundaries and can be used to communicate with people of different cultures.

Visual software engineering refers to the use of various visual means in addition to text in software development. The forms of the development means include graphics, sound, color, gesture, and animation. The Software development lifecycle involves the activities of project management, requirements analysis and specification, architectural and system design, algorithm design, coding, testing, quality assurance, maintenance, and if necessary, performance tuning. These software engineering activities may be assisted through various visual techniques, including visual modeling, visual database query, visual programming, algorithm animation, program visualization, data visualization, and document visualization. Such visual techniques are sometimes categorized into *software visualization* (3), which in a broader sense may include the objective of education in algorithms, programming, and compilers, as well as that of software development (4,5). Figure 1 illustrates the various aspects of software engineering assisted through visualization.

In the first phase of the software engineering process, software managers are responsible for planning and scheduling project development. They typically use several data visualization forms, such as Gantt charts, to illustrate the project schedule meeting a series of milestones. They may also use activity networks to plan project paths leading to the project completion from one milestone to another, or use Petri nets to model the transitions of project activities.

The second phase involves requirements analysis and specification. This phase is usually conducted using various visual modeling techniques, on graphical formalisms such as Statecharts for dynamic analysis and class diagrams for static analysis. More advanced techniques include executable specifications, which can then be realized through visual specification languages. Specifications can be provided via visual programming.

The third phase of the software engineering process establishes an overall software architecture through system and software design. Visual modeling techniques may continue playing a key role, through architectural visualization using various types of architectural diagrams, such as class diagrams and collaboration diagrams. During this phase, algorithm design is needed and the behavior of the algorithm may be understood through visualization and animation. The detailed functionality may need to be transformed into one or more executable programs. Visual language techniques with their well-founded graph grammar support suit particularly well the design, verification, and reuse of executable programs, which will be the focus of this article.

Many modern software systems access databases for organized and inter-related data items from large quantities of data. The logical organization of data is typically modeled in entity-relationship diagrams in relational databases. Complex database queries can be provided through form-based visual structures. For a database management system, visualizing internal segmentation due to fragmented data storage is extremely useful in guiding efficient data placement policies.

In the fourth and fifth phases, the domain software is implemented and coded via visual programming. Both unit testing and integrated testing may be done through techniques such as program slicing and be visualized on graph formalisms such as dependence graphs and call graphs.

Next, software documentation and online help systems are essential for the quality assurance of any software product. They are designed for end users of the software. A comprehensive online help system has a complex network structure that is usually hierarchical with cross-links. A visualized help graph provides an intuitive road map for tutorial, guiding, or diagnostic purposes.

The final maintenance phase takes the longest time in the software lifecycle. During this period, more bugs or requirements errors may be revealed and corrected through program visualization. Program comprehension and analysis can be achieved effectively through graphical visualization. Also during this phase, the performance of the domain software may be improved after it functions as required. Performance evaluation and comparison can be conducted effectively through data visualization (sometimes called statistical visualization). The major difference between program visualization and data visualization is that the visual notations in the former usually correspond directly to the program semantics, whereas those in the latter correspond quantitatively to certain program measurements. For example, nodes in a call graph represent procedures/functions and edges represent call relationships. A segment in a pie chart is significant only in its size and in what it measures.

The remaining part of this article focuses on one of the visual software engineering approaches, i.e., using graph grammars as the underlying theory to support visual software modeling, requirements analysis, architecture design, verification, and evolution.

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

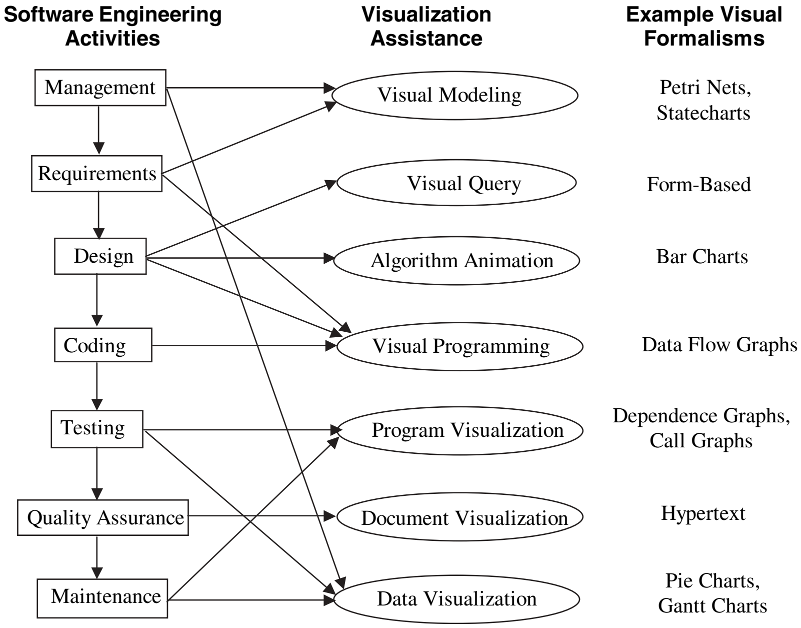


Figure 1. Software engineering assisted by graphical visualization.

A SOUND FOUNDATION

The aforementioned informal graphical notations and formalisms used in various software engineering phases are good at illustration and providing guidance. They are, however, not amendable to automated analysis and transformation. For example, in software architecture design, the developer has to rely on his/her personal experience to discover errors and inconsistencies in an architecture/design diagram. He/she also has to manually redraw the whole architecture/design diagram whenever a change or update is needed. These human tasks are tedious and error-prone. This article presents an approach that can automatically verify and transform design diagrams based on *graph grammars*. The approach abstracts Statecharts, class diagrams, and architecture styles into a grammatical form (as explained in this article). It will then be able to parse a given architecture/design diagram to analyze whether the diagram has some required properties or reconciles some design principles. Moreover, design patterns can be easily visualized and architectural evolution can be achieved through graph transformation.

Graph grammars provide a theoretical foundation for graphical languages (6). A graph grammar consists of a set of rules, which illustrates the way of constructing a complete graph from a variety of nodes. It specifies all possible inter-connections between individual components; i.e., any link in a valid graph can be eventually derived from a sequence of applications of grammar rules (the activity also known as *graph rewriting* or *graph transformation*). Conversely, an unexpected link signals a violation on the graph grammar. A graph grammar can be used to "glue" various components into a complete system. Graph grammars form a formal basis for verifying structures in a diagrammatic notation, and they can be viewed as a model to simulate dynamic evolution. Such an approach facilitates the following aspects of software engineering:

- Graphs are used to specify software by distinguishing individual components and depicting the relationships between the components. A graph grammar specifying design choices and policies provides a powerful mechanism for syntactic checking and verification, which are not supported by most current tools.
- In addition to software design and verification, this approach facilitates a high level of software reuse by supporting the composition of design patterns and uses graph transformation techniques in assisting the evolution and update of software architectures and in reusing the existing products.

A graph grammar is similar to a string (textual) grammar in the sense that it consists of finite sets of labels for nodes and edges, an initial graph, and a finite set of production rules. It defines the operational semantics of a graphical language (6). Graph transformation is the application of production rules that model the permitted actions on graphs representing system structures and states.

In the following explanation of graph grammars, we will use the popular software modeling language Statecharts (7) as our demonstration language, for which a graph grammar can be defined.

In a graph grammar, a *graph rewriting rule,* also called a *production*, as shown in Fig. 2, has two graphs called *left graph* and *right graph*. A production can be applied to a given graph (called a *host graph*) in the form of an *L-application* or *R-application*. A *redex* is a subgraph in the host graph that is isomorphic to the right graph in an R-application or to the left graph in an L-application. A production's L-application to a host graph is to find in the

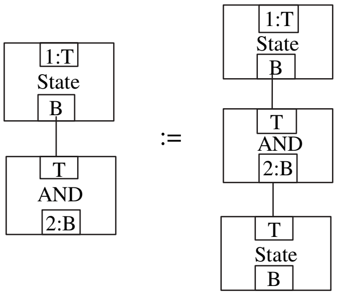


Figure 2. A graph rewriting rule (or a production).

host graph a *redex* of the left graph of the production and replace the redex with the right graph of the production. The L-application defines the language of a grammar. The language is defined by all possible graphs that can be derived using L-applications from an initial graph (i.e., λ) and consist of only *terminals*, i.e., the graph elements that cannot be replaced. An R-application is a reverse replacement (i.e., from the right graph to the left graph) that is used to parse a graph.

A graph grammar is either context-free or context-sensitive. A context-free grammar requires that only one nonterminal is allowed on the left-hand side of a production (8). Most existing graph grammars for visual languages are context-free. A context-sensitive graph grammar, on the other hand, allows the left and right graphs of a production to have an arbitrary number of nodes and edges. Motivated by the need for a general-purpose visual language generator, the authors have developed a context-sensitive graph grammar formalism called the *reserved graph grammar* (RGG) (9).

In an RGG, nodes are organized into a two-level hierarchy as illustrated in Fig. 2. A large rectangle is the first level called a *super-vertex* with embedded small rectangles as the second level called *vertices*. In a node, each vertex is uniquely identified by a capital letter. The name of a super-vertex distinguishes the type of nodes similar to the type of variables in conventional programming languages. A node can be viewed as a module, a procedure, or a variable, depending on the design requirement and granularity. Edges are used to denote communications or relationships between nodes. Either a vertex or a super-vertex can be the connecting point of an edge.

In a context-sensitive grammar, replacing a redex with a subgraph while considering the inter-connection relationship between the redex and its surrounding graph elements is traditionally called *embedding*. The RGG handles the embedding problem using a *marking* mechanism that combines the context information with an embedding rule. The embedding rule states: If a vertex v in the redex of the host graph has an isomorphic vertex v' in the corresponding production's right graph and neither v nor v' is marked, then all edges connected to v should be completely inside the redex. The marking mechanism, will be explained further through examples provided here, makes the RGG expressive, unambiguous, and efficient in parsing.

The RGG formalism uses the object-oriented language Java as a lower level specification tool for instructions and attributes that may not be effectively or accurately specified graphically. These instructions and attributes that are applied to the graph under transformation to perform syntax-directed computations such as data transfer and animation are specified in a piece of Java code (called *action*) attached to the corresponding production. Different actions can be performed on different attributes of the redex of a production to achieve the desired modeling and animation effects. Such an action code is like a standard exception handler in Java by treating each attribute as an object. It associates computation tightly with structural (syntactical) transformation. For example, one can provide the following action code to specify the state transition of a *car* object from *stop* to *star*:

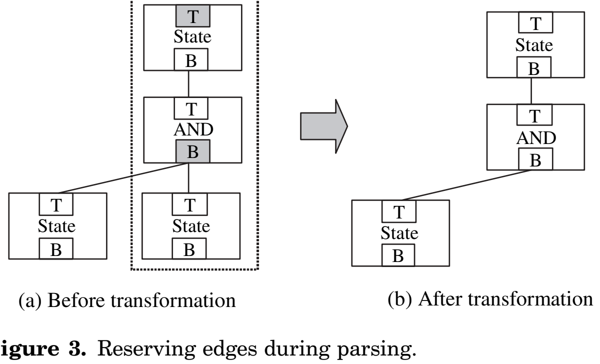
```
action(AAMGraph g) {
    Attribute attributes = g.getAttributes();
    attributes.getObject("car").setState("stop", "start");
}
```

This arrangement allows a software engineer to precisely specify and generate any executable system for visual software modeling and verification as discussed in the next few sections.

The RGG formalism has been used in the implementation of a toolset called VisPro, which facilitates the generation of visual languages using the Lex/Yacc approach (10). As a part of the VisPro toolset, a visual editor that could be used to create visual programs and parsing algorithms is automatically created based on grammar specifications.

MODELING WITH STATECHARTS

This section illustrates the application of the RGG formalism to Statecharts and explains how the marking mechanism works. Figure 3 depicts a snapshot of a subgraph transformation for a Statechart graph using the production in Fig. 2. In Fig. 3(a), the isomorphic graph in the dotted box is a redex. The marked vertices and the vertices corresponding to the isomorphic vertices marked in the right graph of the production are painted gray. The transformation deletes the redex while keeping the gray vertices. Then the left graph of the production is embedded into the host graph, as shown in Fig. 3(b), while treating a vertex in the left graph the same as the corresponding gray vertex. This shows that the marking mechanism allows some edges of a vertex to be reserved after transformation. For example, in



(a) Before transformation

(b) After transformation

Figure 3. Reserving edges during parsing.

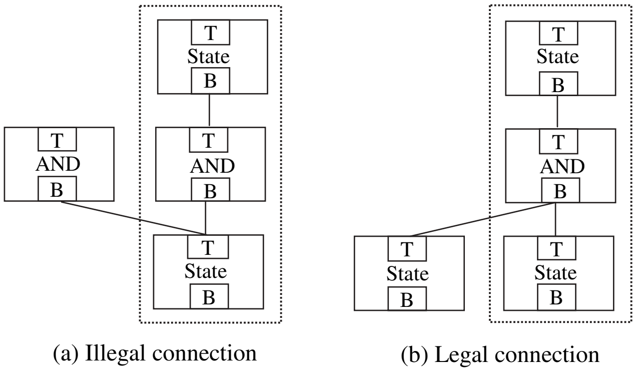


Figure 4. Determining connectivity.

Fig. 3(a), the edge connecting to the "State" node outside the redex is reserved after transformation.

In the definition of the Statecharts grammar, an "AND" node may connect to multiple "State" nodes, indicating the AND relationships among the states. A "State" node, however, is allowed to connect to only one "AND" node. We show how such a connectivity constraint can be expressed and maintained in the RGG. The solution is simple: Mark the B vertex of the "AND" node, and leave the T vertex of the "State" node unmarked in the definition of the production (as illustrated in Fig. 2). According to our embedding rule, the isomorphic graph in the dotted box in Fig. 4(a) is not a redex, because the isomorphic vertex of the unmarked vertex T in the "State" node has an edge that is not completely inside the isomorphic graph. Therefore, the graph in Fig. 4(a) is invalid. On the other hand, the graph in Fig. 4(b) is valid according to the embedding rule. There is a redex, i.e., the isomorphic graph in the dotted box, in the graph, because the isomorphic vertex of B in "AND" connecting to "State" in the right graph of the production is marked, even though it has an edge connected outside the isomorphic graph. Therefore, the marking mechanism helps not only in embedding a graph correctly, but also in simplifying the grammar definition. It allows an implicit representation to avoid most context-specifications while being more expressive. This greatly reduces the complexity of visual expressions and, in turn, increases the efficiency of the parsing algorithm.

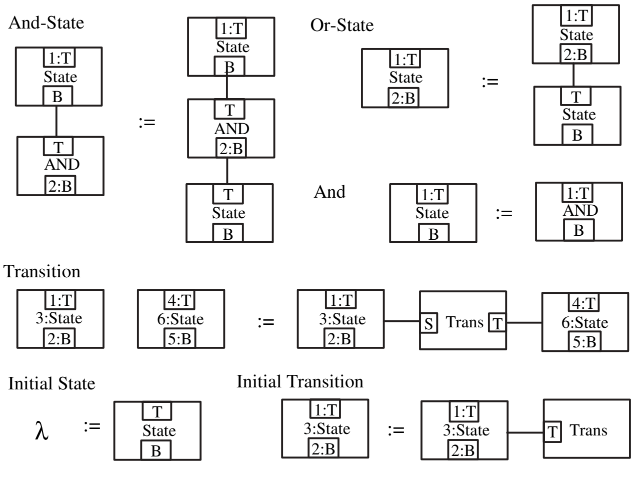


Figure 5. The graph grammar for Statecharts.

The graph grammar expressed in the RGG formalism for a main subset of the Statechart notations is listed in Fig. 5, including the initial state, initial AND, initial transition, general AND state, general OR state, and general transition productions. The last three general productions can all be repeatedly applied during the graph rewriting process. Figure 6 depicts an example Statechart and its representation in the node-edge form that is recognized by the RGG to be parsed by the Statechart grammar. With the Statechart grammar defined, any user-drawn Statechart diagrams like the one shown in Fig. 6(a) can be validated for its syntactical correctness and executed according to the action code attached to each production (action codes are not shown in the figure).

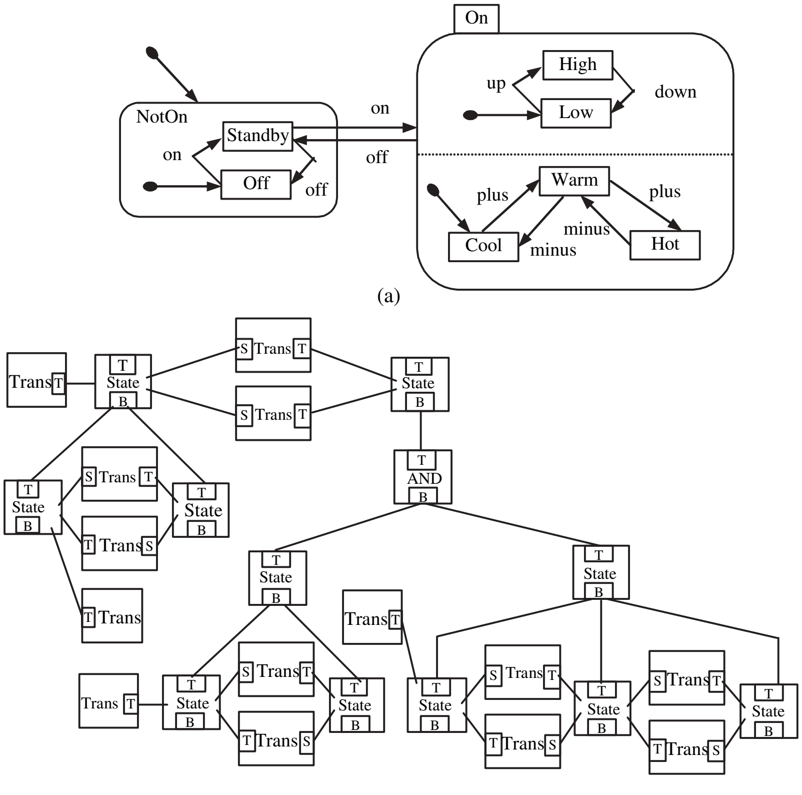
SPECIFYING CLASS DIAGRAMS

This section goes through an example to illustrate the representation of class diagrams in the RGG's node-edge form, and then it defines a graph grammar for class diagrams. A parser can verify some properties of the diagrams. The next section discusses how this graph grammar can help visualizing design pattern applications and compositions in their class diagrams.

Class diagram, one of the most popular diagrams for object-oriented modeling and design, visually models the static structure of a system in term of classes and relationships between classes (2). To verify the structure of a class diagram in Fig. 7(a), one needs to first translate the class diagram into a node-edge format [Fig. 7(b)], on which the RGG parser operates, in the same fashion as for Statecharts presented in the last section.

In a class diagram, classes are represented by compartmentalized rectangles. In its node-edge counterpart, a node labeled *Class* denotes the top compartment containing the class name. A set of nodes labeled *Attri* represents attributes in the middle compartment. Nodes are sequenced by linking two adjacent attributes in the same order as displayed in the compartment, and the sequence is attached to a class by linking the first *Attri* node with the *Class* node. Operations in the bottom compartment are processed in the same manner as attributes when replacing *Attri* by *Oper* nodes.

Associations denoted by straight lines typically used in UML (2) carry the information about relationships between classes. In a node-edge diagram, a node labeled *Asso* is used to symbolize an association. A line connecting an *Asso* node to a *Class* node holds an association relationship between them. Associations may be named, preferably in a verbal form, being either active, like "works for," or passive, like "is hired by," and thus called *verbal constructs* in UML (2). To indicate the direction in which the name should be read, vertex *R* in an *Asso* node is connected to the *Class* node designated by a verbal construct, and vertex *L* to the other *Class* node. On the other hand, if the order is unimportant, one can ignore the difference between *R* and *L*.



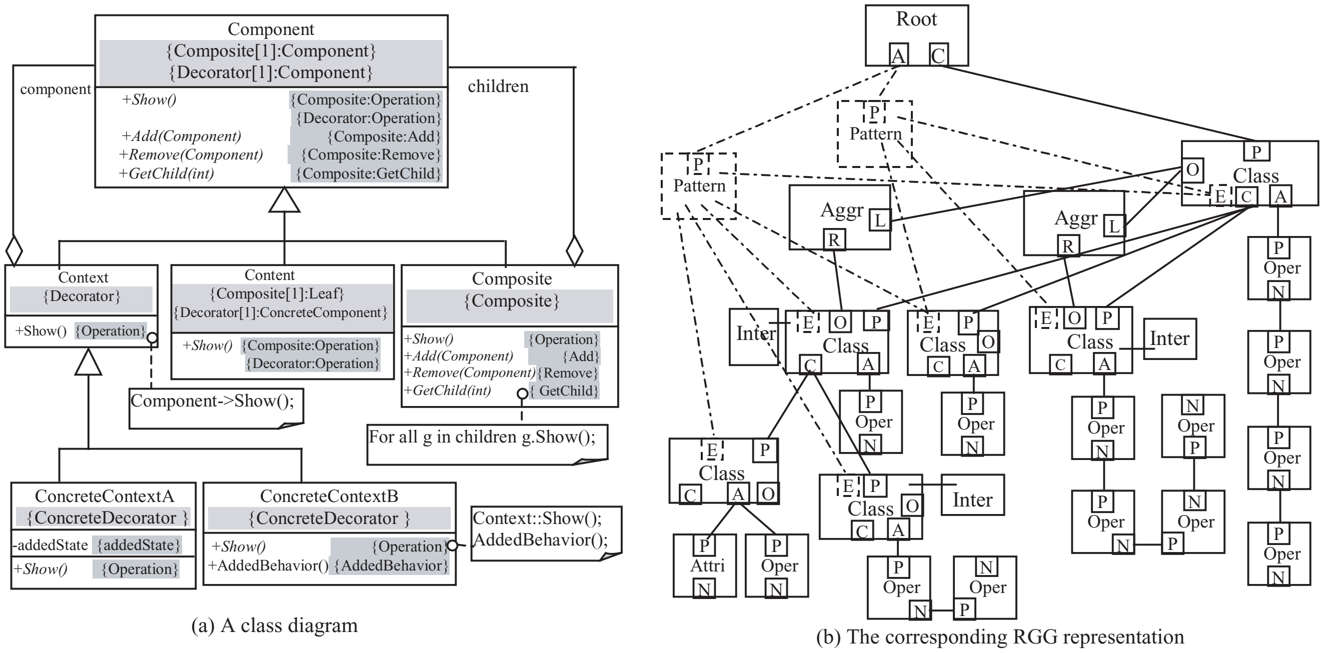
(a)

(b)

Figure 6. An example Statechart (a) and its node-edge representation (b).

For example, Fig. 8(a) specifies an association *Drive* between classes *Person* and *Car*, where a small triangle points to the *Car* class designated by a verbal construct. Correspondingly, in the node-edge representation in Fig. 8(b), vertex *R* in the *Drive Asso* node is connected to the *Car* class node.

Aggregation and *composition*, two special types of associations, are represented by *Aggr* and *Comp* nodes, respectively, in the node-edge representation. An *Aggr*/*Comp* node bridges a pair of *Class* nodes in the same fashion as an *Asso* node does.



(a) A class diagram

(b) The corresponding RGG representation

Figure 7. A class diagram and its corresponding RGG diagram.

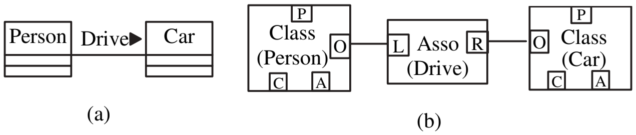


Figure 8. An (a) association and its (b) node-edge representation.

In UML, *generalization* denotes a hierarchical relationship between a general description and a specific description. In the node-edge representation, a directed edge linking from the vertex labeled *c* in a *Class* node to the vertex labeled *p* in another *Class* node designates the generalization relationship from the former class to the latter class. In other words, vertex *c* indicates a general class and vertex *p* denotes a specific class.

To facilitate parsing and verifying the structure of an RGG diagram, we introduce a new node to the node-edge representation, namely *root*, which has no counterpart in the class diagram. A *root* node is connected to any *Class* node that represents a class without a super-class.

Although a graph grammar abstracts the essence of structures, it cannot convey precise information visually. The RGG stores concrete and numeric information in attributes as described. For example, association names are recorded in attributes attached to Asso nodes. Those values of attributes can be retrieved and evaluated in the parsing process.

Figure 7(a) illustrates a class diagram, and Fig. 7(b) presents its corresponding node-edge diagram recognizable by its RGG. The shaded texts in Fig. 7(a) represent pattern names as extended notations to UML, and the dotted rectangles in Fig. 7(b) correspond to the extended UML (11).

A graph grammar can be viewed as a style to which any valid graph should conform; i.e., any possible interconnection between graph entities must be specified in the grammar. Each production defines the local relationships among the graph elements/entities. Collecting together the productions defining all relationships, an RGG grammar specifies the way of constructing a valid class diagram using graph entities represented by different types of nodes.

Figure 9 presents all RGG productions that define class diagrams. Production 1 reduces two attribute nodes into one, which is treated as one entity in later applications. Repetitive applications of Production 1 reduce all attributes of a class to one attribute, which is later treated together with its class by Production 3. Productions 1 and 2 serve to reduce a sequence of attributes and operations. Production 3 specifies the class structure by attaching sequences of operations and attributes to a *Class* node. Production 4 defines the constraints between associations. Productions 5 and 6 specify the template class and the interface, respectively. Productions 7, 12, and 14 all define associations, and Productions 8 and 9 specify aggregation and composition, respectively. Productions 10 and 13 demonstrate the generalization through inheritance. Production 15 represents the initial state. The nodes and vertices in dotted rectangles define pattern-extended class diagrams, which will be explained in the next section.

DESIGN PATTERN VISUALIZATION

UML (2) provides a set of notations to represent different aspects of a software system. However, it is still not expressive enough for some particular problems, such as design pattern applications and compositions (12). This section introduces the idea of using the RGG formalism to visualize design patterns through their corresponding class diagrams.

Design patterns (13) document good solutions to recurring problems in a particular context, and their compositions (12) are usually modeled using UML. When a design pattern is applied or composed with other patterns, the pattern-related information may be lost because UML does not track this information. Thus, it is hard for a designer to identify a design pattern when it is applied or composed. The benefits of design patterns are compromised because designers cannot communicate with each other in terms of the design patterns they use when the design patterns are applied or composed. Several graphical notations have been proposed to explicitly represent pattern-related information in UML class diagrams (11). Although these solutions need to attach additional symbols and/or text, they all suffer from the scalability problem when the software design becomes very large. A solution that can dynamically visualize pattern-related information based on the RGG is illustrated in Fig. 9. A new type of node, called *pattern*, is used to denote a specific pattern, and pattern-related information is expressed by linking a pattern node with its associated class nodes. Figure 7(b) presents the corresponding node-edge diagram by highlighting the newly introduced nodes and edges with dotted lines.

A syntactic analyzer implemented in the parser can dynamically collect separate pieces of information and reconstruct them into a new graph entity if desirable. In the process of parsing, a sequence of applications of Production 17 in Fig. 9 collects all classes belonging to the same pattern to support user interaction and queries. For example, if the user clicks on the *composite* class in Fig. 7(a), the *component* class, *content* class, and *composite* class, which belong to the Composite pattern, are all highlighted. Therefore, there is no need to attach any additional information to the original class diagrams.

AUTOMATIC VERIFICATION

Tools supporting general syntactic checking on class diagrams already exist. They, however, cannot verify certain properties. For example, multi-inheritance may cause ambiguity in the class design and usage. It is desirable to prohibit multi-inheritance when modeling software implemented in conventional programming languages. As explained, each production specifies a local structure. By “gluing” separate structures together, repetitive applications of various productions can generate a complete structure. A graph specifying a structure is invalid if it breaks at least one relationship specified in any production. For example, Production 6 in Fig. 9 defines that one interface can be attached to only one class. If an interface is

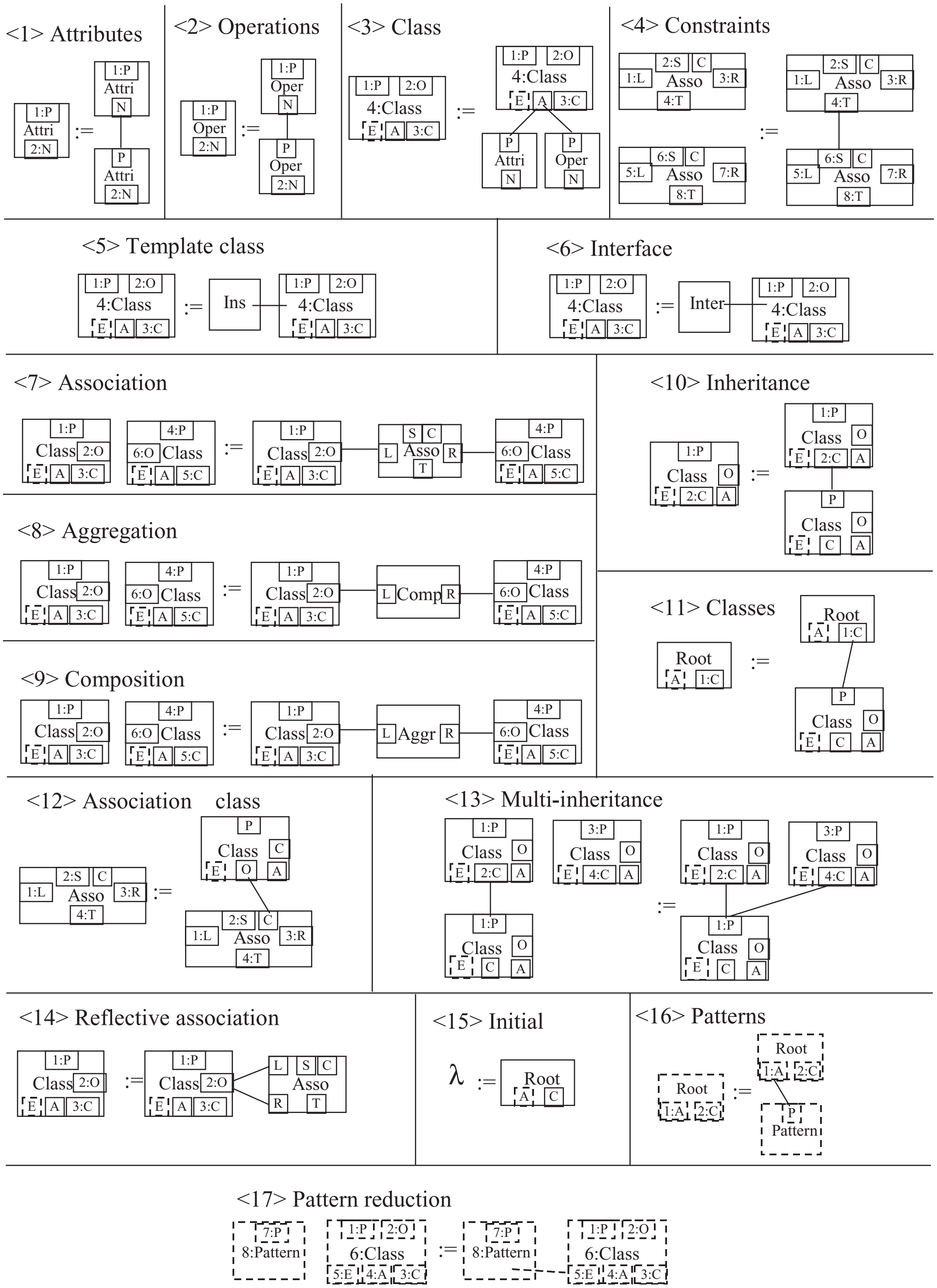
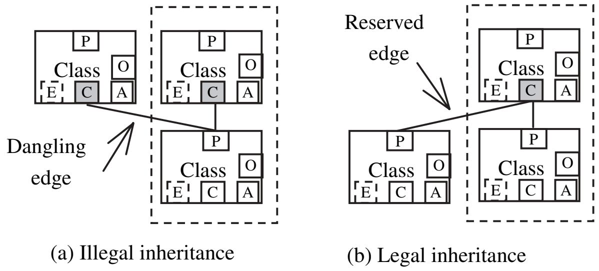


Figure 9. A graph grammar defining class diagrams.

designed to relate to more than one class, a parser can indicate a violation of Production 6.

The following example illustrates how to verify inheritance relationships between classes. In Fig. 9, Production 10 defines the case of single inheritance, and Production 13 specifies that of multi-inheritance. As any valid relationship between components can be eventually derived from the graph grammar for class diagrams, removing Production 13 would implicitly prohibit any multi-inheritance. To explain in detail how to invalidate multi-inheritance, we need to apply the *marking* technique (9) explained earlier. A marked vertex is distinguished by assigning to it a unique integer. It preserves outgoing edges connected to vertices outside a replaced subgraph. In the right graph



(a) Illegal inheritance

(b) Legal inheritance

Figure 10. Inheritance verification.

of Production 10, the edge indicates an inheritance relationship between the classes. The unmarked vertex *p* in the bottom class node representing a subclass requires that any class can only inherit from at most one other class. On the other hand, the marked vertex *c* in the top class node representing a super-class defines that one super-class can have multiple subclasses, conforming to the principle of single inheritance. If the multiinheritance as illustrated in Fig. 10(a) occurs, the application of Production 10 results in an undesirable condition called the *dangling edge* condition (6), which is prohibited in the RGG formalism. In the case in which one class has more than one subclass, a successful application is shown in Fig. 10(b).

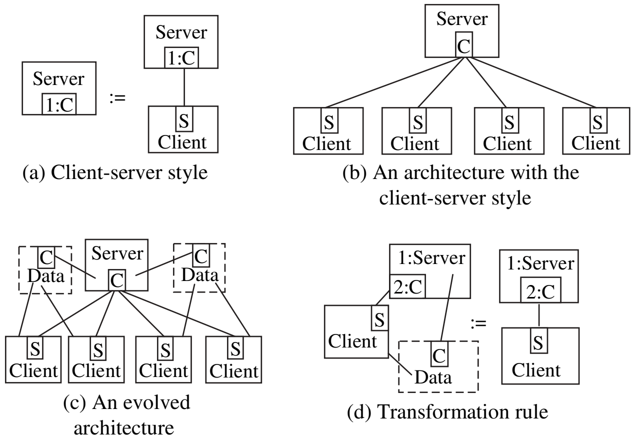
ARCHITECTURAL EVOLUTION

The architectures of software systems are not usually fixed. To meet the changing requirements and/or adapt to a different context, a software architecture may need to be transformed into a new configuration. Furthermore, a high-level software architecture style may gradually be refined into a detailed architecture (14) during software development. This transformation process can be tedious and error-prone without tool support. This section illustrates the automated transformation for software evolution from one architecture style to another. Graph rewriting provides a device for reusing existing software components by evolving them into newly required forms.

A software architecture style defined through an RGG characterizes some common properties shared by a class of architectures. To satisfy new requirements and reuse existing designs, an architecture with one style needs to evolve into another with a more appropriate style in the new context. In general, software architecture transformation proceeds in two steps: (*1*) Verify the style of an architecture, and (*2*) transform an architecture from one style to another style.

Assume that a system is originally implemented in a client–server style, consisting of only one server storing all data. To retrieve data, clients must send requests to, and receive responses from, the server. This communication pattern is abstracted into a graph grammar shown in Fig. 11(a), and an architecture with that style is illustrated in Fig. 11(b).

When the amount of data and communication increases, one server may no longer be able to bear clients' requests. One possible solution is to distribute data to different servers. Therefore, we need to transform the current style to a more advanced one by dividing servers into control and data servers. A system can only contain one control server, but it may have several data servers. A client sends requests to the control server, which forwards them to an appropriate data server. Then, the data server directly replies to the client. Such a communication pattern is defined in Fig. 11(c), which is achieved through the graph rewriting rule for transformation in Fig. 11(d).



(a) Client-server style

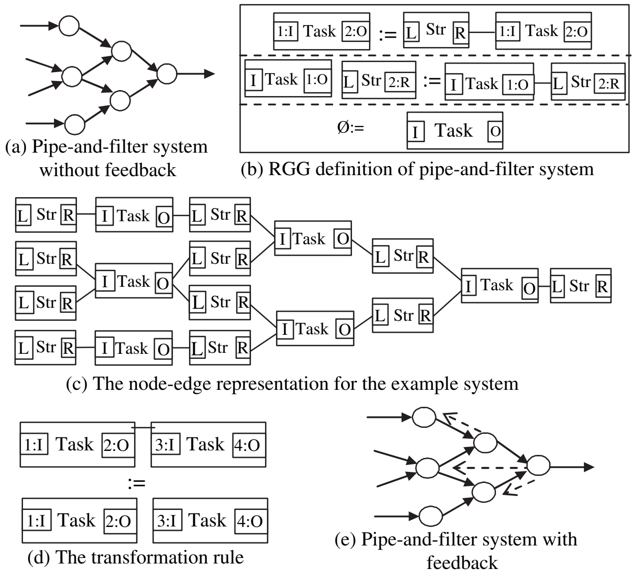
(b) An architecture with the client-server style

(c) An evolved architecture

(d) Transformation rule

Figure 11. Architectural transformation.

Let us go through another example to illustrate the idea of architecture evolution through graph transformation. A simple pipe-and-filter system without feedback is shown in Fig. 12(a), where a circle represents a task and a directed edge indicates a data stream between tasks. Correspondingly, a node labeled *Str*/*Task* simulates a stream/task in the node-edge representation. An edge connecting the *R*/*L* vertex in a *Str* node to the *I*/*O* vertex in a *Task* node expresses an incoming/outgoing stream. Figure 12(c)



(a) Pipe-and-filter system without feedback

(b) RGG definition of pipe-and-filter system

(c) The node-edge representation for the example system

(d) The transformation rule

(e) Pipe-and-filter system with feedback

Figure 12. Pipe-and-filter system.

illustrates the node-edge representation for the system shown in Fig. 12(a). The productions defined in Fig. 12(b) abstract the communication pattern in pipe-and-filter systems without feedback. By allowing an edge between two *Task* nodes to indicate a feedback between them, the graph rewriting rule given in Fig. 12(d) transforms a system without a feedback to one with feedback. Fig. 12(e) illustrates a system with feedback after applying the rule in Fig. 12(d) to the example in Fig. 12(a), where the dotted edges represent feedbacks.

CONCLUSION

Having introduced the basic concept of visual software engineering, this article presents a graph grammar approach to software architecture specification, verification, and evolution. Through this approach, various diagrammatical forms can be translated to the graphical notation recognizable by the RGG formalism and then applied by graph transformation in achieving the desired effect. In summary, the approach facilitates a sound software engineering practice with the following benefits:

- **Consistent:** It expresses software architectures in terms of "box and line" drawings (15), like the common practice of software engineers (16).
- **Scalable:** The underlying graph grammar formalism is applicable to various classes of diagrams. It is easy to accommodate new components by extending the graph schema and revising corresponding grammar rules and, thus, support software reuse.
- **Automatic:** Automatically generated by a visual language generator, such as VisPro (10), a transformation tool is capable of syntactic checking of software architectures. Automatic transformation from one architecture style to another assists software engineers in reusing existing products in new applications.

FURTHER READING

Visual software engineering has been a new relatively concept since the emerging graphical tools, notably UML, have increasingly been used in the software industry in recent years. The more commonly acknowledged term for visual software development and for software education is "software visualization" (3–5). A related active research area is visual programming and visual languages (17), from which the approach presented in this article was originally developed. The following summaries point to the representative early work in using graph transformation techniques to assist software engineering, specifically software architecture design.

Dean and Cordy (18) present a diagrammatic representation of software architectures. A graph visualizes the structure of a software architecture, and a graph grammar abstracts the overall organization of a class of architectures. Based on the equivalent of context-free grammars, Dean and Cordy introduced a pattern matching mechanism for recognizing classes of software architectures.

Métayer (16) also defines the style of architectures using graph grammars that are defined in terms of set theory. Instead of discussing pattern matching over software architectures, Métayer emphasizes the dynamic evolution of an architecture, performed through graph rewriting. An algorithm is presented to check whether an evolution breaks communication constraints.

Radermacher (19) discusses graph transformation tools supporting the construction of an application conforming to a design pattern, which is specified through graph queries and graph rewriting rules. A prototype can be generated by the PROGRES graph rewriting environment (20).

BIBLIOGRAPHY

1. M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Englewood Cliffs, NJ.: Prentice Hall, 1995.
2. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Reading, MA.: Addison-Wesley, 1999.
3. K. Zhang (ed.), *Software Visualization–From Theory to Practice*, Boston, MA.: Kluwer Academic Publishers, 2003.
4. P. Eades and K. Zhang (eds.), *Software Visualisation*, Series on Software Engineering and Knowledge Engineering, Vol. 7, Singapore: World Scientific Publishing Co., 1996.
5. J. Stasko, J. Domingo, M. H. Brown, and B. A. Price, *Software Visualization: Programming as a Multimedia Experience*, Cambridge, MA.: MIT Press, 1998.
6. G. Rozenberg (ed.), *Handbook on Graph Grammars and Computing by Graph Transformation: Foundations*, Vol. 1, Singapore: World Scientific, 1997.
7. D. Harel, Statecharts: A visual formalism for complex systems, *Sci. Comp. Prog.*, **8** (3): 231–274, 1987.
8. K. Wittenburg and L. Weitzman, Relational grammars: Theory and practice in a visual language interface for process modeling, *Proc. of AVI'96*, Gubbio, Italy, 1996.
9. D. Q. Zhang, K. Zhang, and J. Cao, A Context-Sensitive Graph Grammar Formalism for the Specification of Visual Languages, *Comp. J.*, **44** (3): 187–200, 2001.
10. K. Zhang, D-Q. Zhang, and J. Cao, Design, construction, and application of a generic visual language generation environment, *IEEE Trans. Software Eng.*, **27** (4): 289–307, 2001.
11. J. Dong and K. Zhang, Design Pattern Compositions in UML, in K. Zhang (ed.), *Software Visualization – From Theory to Practice*, Boston, MA.: Kluwer Academic Publishers, 2003, pp. 287–208.
12. R. K. Keller and R. Schauer, Design components: Towards software composition at the design level, *Proc. 20th Int. Conf. Software Eng.*, Tokyo, Japan, 1998, pp. 302–311.
13. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Reading, MA.: Addison-Wesley, 1995.
14. M. Moriconi, X. L. Qian, and R. A. Riemenschneider, Correct architecture refinement, *IEEE Trans. Software Eng.*, **21** (4): 356–372, 1995.
15. R. Allen and D. Garlan, Formalizing architectural connection, *Proc. 16th Int. Conf. Software Eng.*, Sorrento, Italy, 1994, pp. 71–80.

16. D. L. Métayer, Describing software architecture styles using graph grammars, *IEEE Trans. Software Eng.*, **24** (7): 521–533, 1998.
17. M. M. Burnett, Visual Language Research Bibliography, 2004. Available: http://www.cs.orst.edu/~burnett/vpl.html.
18. T. R. Dean and J. R. Cordy, A syntactic theory of software architecture, *IEEE Trans. Software Eng.*, **21** (4): 302–313, 1995.
19. A. Radermacher, Support for design patterns through graph transformation tools, *Proc. Application of Graph Transformations with Industrial Relevance*, LNCS 1779, Berlin Heidelberg: Springer–Verlag, 1999, pp. 111–126.
20. A. Schürr, A. Winter, and A. Zündorf, The PROGRES approach: Language and environment, in G. Rozenberg (ed.), *Handbook on Graph Grammars and Computing by Graph Transformation: Applications*, Vol. 2, Singapore: World-Scientific, 1999, pp. 487–550.

KANG ZHANG
The University of Texas at Dallas
Richardson, Texas

JUN KONG
The North Dakota State University
Fargo, North Dakota

JIANNONG CAO
Hong Kong Polytechnic University
Hung Hom, Kowloon
Hong Kong

ANSWER SET PROGRAMMING

INTRODUCTION

Computer applications pervade our life, and these days many problems of everyday life are dealt with in an automated way. However, not all problems are easy to solve by a computer, some have an increased intrinsic complexity. Finding efficient and correct methods for solving them is not an easy task. Traditional software engineering is focused on an imperative, algorithmic approach, in which the computer is basically being told what steps should be followed in order to solve the given problem. Finding good algorithms for hard problems requires skill and knowledge and is often not obvious. This can be dealt with by involving an expert, but there is a serious drawback: When the specification of the problem changes slightly, perhaps only because additional information on the nature of the problem becomes available, major reengineering is often necessary. The main problem is that the knowledge about the problem and its solutions has been represented implicitly by representing a specific way of solving the problem rather than the problem itself. The case of updating representations is sometimes called elaboration tolerance.

An alternative that suits elaboration tolerance better is called declarative programming. In this approach, the problem and its solutions are specified explicitly. That is, it is expressed what features the problem and its solution must have, rather than specifying how a solution is to be obtained. Methods like this actually come natural in science but also in everyday life. Before we try to work out how to solve a problem, we usually first try to understand it and figure out how a solution would like, before trying to find a method, to obtain a solution. One of the first to put this approach into perspective in computer science was John McCarthy in the 1950s (1). He also postulated that the most natural language for specifying problems and solutions would be logic and, in particular, predicate logic.

In fact, logic is an excellent candidate for declarative programming: It provides a simple and abstract formalism, and in addition, it has the potential for automation. Similar to an abstract or electronic machine that can execute an imperative model (an algorithm) in order to obtain a solution of the modeled problem, computational logic has produced tools that allow for automatically obtaining solutions, given a declarative specification in logic. Indeed, many people nowadays use this way of solving problems: Queries to relational databases together with the database schemata are indeed declarative specifications of the solutions that the query results provide. And, indeed, the probably most widely used database query language, SQL, is basically predicate logic written in a particular way (2).

However, one wants to go beyond databases as they are used today. It has been shown that relational databases and query languages like SQL can only represent fairly simple problems. For instance, problems like finding the cheapest tour of several cities, or filling a container with items of different size, such that the value transported in the container is maximized, are typical problems that probably cannot be solved using SQL. It might seem unusual to use the word "probably" here, but underlying this conjecture is one of the most famous open problems in computer science—the question of whether P equals NP. These are complexity classes; basically, every problem has some intrinsic complexity, which is based on how many resources are required to solve it on a standard machine model, in terms of the size of the problem input. P is defined as the class of problems, which require at most an amount of time, which can be expressed as a polynomial over the input size (which is variable). NP is just a slight alteration, in which instead of a deterministic machine model, a nondeterministic machine model is assumed. A nondeterministic machine is a somewhat unusual concept: Instead of executing commands one-by-one, always going from one machine state to another, a nondeterministic machine may be in two or more states (at the same time) after having executed a command. In a sense, this means that the machine has the possibility to store and work with an unbounded number of machine states at any time. Intuitively, one would expect that a deterministic and a nondeterministic machine are quite different from each other, and that the nondeterministic machine can solve more problems under the same time constraints. However, up to now, nobody has been able to prove convincingly neither that P and NP are different, nor that they are equal. However, intuitively one would expect that they are different, and people have shown that many more unintuitive results would follow if P and NP coincided.

Logic programming is an attempt to use declarative programming with logic that goes beyond problems in P and, thus, beyond traditional databases. The main construct in logic programming is a rule, a expression that looks like *Head* ← *Body*, where *Body* is a logic conjunction possibly involving negation, and *Head* is either an atomic formula or a logic disjunction. This can be seen as a logic formula (← denoting implication), with the special meaning that *Head* is defined by that rule. In the beginning of the field (as described in the following section), logic programming actually attempted to become a full-scale "programming language." Its most famous language, Prolog (3), aimed at this, but had to renounce to full declarativity in order to achieve that goal. For instance, in Prolog rules, the order inside *Body* matters, as does the order among rules (most notably for termination). Moreover, Prolog also had several nonlogical constructs.

Answer set programming (ASP) is a branch of logic programming, which does not aspire to create a full general-purpose language. In this respect, it is influenced by database languages, as also these are not general-purpose languages, but suffice for a particular class of problems. ASP does, however, attempt to enlarge the class of problems that can be expressed by the language. Although, as

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

mentioned, SQL probably cannot express hard problems in NP, ASP definitely can. Actually, ASP can express all problems in the complexity class $\sum_2^p$ and its complement $\prod_2^p$, which are similar to NP, but probably somewhat larger (but at least equally large).

In ASP, the rule construct *Head* ← *Body* (where *Head* can be a disjunction) is read like a formula in nonmonotonic logics rather than classical logic. Nonmonotonic logics are an effort to formulate a logic of common sense that is adapting the semantics of logic such that it corresponds better to our everyday reasoning, which is characterized by the presence of incomplete knowledge, hypothetical reasoning, and default assumptions. It can be argued that nonmonotonic logics are much better suited in such a setting than classical logic.

Summarizing, ASP is a formalism that has emerged from logic programming. Its main representation feature are rules, which are interpreted according to common sense principles. It allows for declarative specifications of a rich class of programs, generalizing the declarative approach of databases. In ASP, one writes a program (a collection of rules), which represent a problem to be solved. This program, together with some input, which is also expressed by a collection of rules, possesses a collection of solutions (possibly also no solution), which correspond to the solutions of the modeled problem. Since these solutions are usually sets, the term "answer set" has been coined.

Concerning terminology, ASP is sometimes used in a somewhat broader sense, referring to any declarative formalism representing solutions as sets. However, the more frequent understanding is the one adopted in this article, which dates back to Ref. 4. Moreover, since ASP is the most prominent branch of logic programming in which rule heads may be disjunctive, sometimes the term "disjunctive logic programming" can be found referring explicitly to ASP. Yet other terms for ASP are A-Prolog and stable logic programming. For complementary introductory material on ASP, we refer to Refs. 5 and 6.

LOGIC PROGRAMMING

The roots of answer set programming lie predominantly in logic programming, nonmonotonic reasoning, and databases. In this section, we give an overview on the history of logic programming from the perspective of answer set programming. It, therefore, does not cover several important subfields of logic programming, such as constraint logic programming (7) or abductive logic programming (8).

As mentioned, probably the first to suggest logic, and in particular predicate logic, as a programming language was John McCarthy in the 1950s (1). McCarthy's motivating example was set in artificial intelligence, and involved planning as its main task, an agenda on which was continuously elaborated; see, for instance, Ref. 9.

Developments in computational logic, most notably the specification of the resolution principle and unification as a computational method by J. Alan Robinson in 1965 (10), acted as a catalyst for the rise of logic programming. This development eventually really set off when a working system, Prolog, developed by a group around Alain Colmerauer in Marseilles, France, became available (3). A few other, somewhat more restricted systems had been available before, but Prolog was to make the breakthrough for logic programming.

One of the prime advocates of what would become known as the logic programming paradigm has been Robert Kowalski, who provided the philosophical basis and concretizations of the logic programming paradigm, for instance, in Refs. 11 and 12. Kowalski also collaborated with Colmerauer on Prolog, and in the realm of his group in Edinburgh, Scotland, alternative implementations of Prolog were created. There has also been a standardization effort for the language, which would become known as Edinburgh Prolog and served as the de facto specification of Prolog for many years until the definition of ISO Prolog in 1995 (13).

However, logic programming, and Prolog in particular, was inspired by, but not the same as classical first-order logic. Initially the differences were not entirely clear. The first effort to provide a formal definition for the semantics of logic programming was also undertaken by Kowalski, who together with Maarten van Emden gave a semantics based on fixpoints of operators for a restricted class of logic programs (Horn programs, also called positive programs) in Ref. 14. This fixpoint semantics essentially coincided with minimal Herbrand models and with resolution-based query answering on Horn programs. The major feature missing in Horn programs is negation—however, Prolog did have a negation operator.

Indeed, the quest for finding a suitable semantics in the spirit of minimal models for programs containing negation turned out to be far from straightforward. A first attempt was made by Keith Clark in Ref. 15 by defining a transformation of the programs to formulas in classical logic, which are then interpreted using the classical model semantics. However, the approach gave arguably unintuitive results for programs with positive recursion. In particular, the obtained semantics does not coincide with the minimal model semantics on positive programs. At about the same time, Raymond Reiter formulated the Closed World Assumption in Ref. 16, which can be seen as the philosophical basis of the treatment of negation. Another milestone in the research on the intended semantics for programs with negation has been the definition of what later became known uniformly as perfect model semantics for programs that can be stratified on negation, in Refs. 17 and 18. The basic idea of stratification is that programs can be partitioned in subprograms (strata) such that the rules of each stratum contain negative predicates only if they are defined in other strata. In this way, it is possible to evaluate the program by separately evaluating its partitions in such a way that a given "stratum" is processed whenever the ones from which it (negatively) depends have already been processed.

Although an important step forward, it is obvious that not all logic programs are stratified. In particular, programs that are recursive through negation are never stratified, and the problem of assigning a semantics to nonstratified programs still remained open. There were basically two approaches for finding suitable definitions: The first approach was giving up the classical setting of models that assign two truth values, and introduce a third value, intuitively representing unknown. This approach required a somewhat different definition, because in the two-valued approach, one would give a definition only for positive values, implicitly stating that all other constructs are considered to be negative. For instance, for minimal models, one minimizes the true elements, implicitly stating that all elements not contained in the minimal model will be false. With three truth values, this strategy is no longer applicable, as elements that are not true can be either false or undefined. For resolving this, Allen Van Gelder, Kenneth Ross, and John Schlipf introduced the notion of unfounded sets in Ref. 19, in order to define which elements of the program should be definitely false. Combining existing techniques for defining the minimal model with unfounded sets, they defined the notion of a well-founded model. In this way, any program would still be guaranteed to have a single model, just like there is a unique minimal model for positive programs and a unique perfect model for stratified programs.

The second approach consisted of viewing logic programs as formulas in nonmonotonic logics (see, for instance, Ref. 20 for an overview) rather than formulas of classical logic (with an additional minimality criterion) and as a corollary, abandoning the unique model property. Among the first to concretize this were Michael Gelfond in Ref. 21, who proposed to view logic programs as formulas of autoepistemic logic, and Nicole Bidoit and Christine Froidevaux in Ref. 22, who proposed to view logic programs as formulas of default logic. Both of these developments have been picked up by Michael Gelfond and Vladimir Lifschitz, who in Ref. 23 defined the notion of stable models, which is inspired by nonmonotonic logics, however does not refer explicitly to these, but rather relies on a reduct that effectively emulates nonmonotonic inference. It was this surprisingly simple formulation, which did not require previous knowledge on non-classical logics that has become well known. Different to well-founded models, there may exist no, one, or many stable models for one program. However, well-founded and stable models are closely related; for instance, the well-founded model of a program is contained in each stable model (cf. Ref. 24). Moreover, both approaches coincide with perfect models on stratified programs.

Yet another, somewhat orthogonal line of research concerned the use of disjunction in rule heads. This construct is appealing, because it allows for direct nondeterministic definitions. Prolog and many other logic programming languages traditionally do not provide such a feature, being restricted to so-called definite rules. Jack Minker has been a pioneer and advocate of having disjunctions in programs. In Ref. 25, he formulated the Generalized Closed World Assumption, which gave a simple and intuitive semantics for disjunctive logic programs. This concept has been elaborated on over the years, most notably by the Extended GCWA defined in Ref. 26. Eventually, also the stable model semantics has been extended to disjunctive programs in Ref. 27 by just minimally altering the definition of Ref. 23. On the other hand, defining an extension of well-founded models for disjunctive programs remains a controversial matter to this date with various rivalling definitions, (cf. Ref. 28).

The final step toward answer set programming in the traditional sense has been the addition of a second kind of negation, which has a more classical meaning than negation as failure. Combining this feature with disjunctive stable models of Ref. 27 led to the definition of answer sets in Ref. 4.

FORMAL DEFINITION OF ASP

In what follows, we provide a formal definition of the syntax and semantics of answer set programming in the spirit of Ref. 4, that is, disjunctive logic programming involving two kinds of negation (referred to as strong negation and negation as failure), under the answer sets semantics.

Syntax

Following a convention dating back to Prolog, strings starting with uppercase letters denote logical variables, whereas strings starting with lowercase letters denote constants. A *term* is either a variable or a constant. Note that, as common in ASP, function symbols are not considered.

An *atom* is an expression $\mathcal{P}(t_1,\ldots,t_n)$, where p is a *predicate* of arity n and $t_1,\ldots,t_n$ are terms. A *classical literal* l is either an atom p (in this case, it is *positive)*, or a negated atom $\neg p$ (in this case, it is *negative)*. A *negation as failure (NAF) literal* ℓ is of the form l or not l, where l is a classical literal; in the former case ℓ is *positive*, and in the latter case *negative*. Unless stated otherwise, by *literal* we mean a classical literal.

Given a classical literal l, its *complementary* literal $\neg l$ is defined as $\neg p$ if $l = p$ and p if $l = \neg p$. A set L of literals is said to be *consistent* if, for every literal $l \in L$, its complementary literal is not contained in L.

A *disjunctive rule (rule*, for short) r is a construct

$$a_1 \vee \cdots \vee a_n \leftarrow b_1,\ldots,b_k,\ \text{not}\ b_{k+1},\ldots,\text{not}\, b_m. \qquad (1)$$

where $a_1,\ldots,a_n,b_1,\ldots,b_m$ are classical literals and $n \geq 0,\ m \geq k \geq 0$. The disjunction $a_1 \vee \cdots \vee a_n$ is called the *head* of r, whereas the conjunction $b_1,\ldots,b_k$, not $b_{k+1},\ldots$, not b_m is referred to as the *body* of r. A rule without head literals (i.e., $n = 0$) is usually referred to as an *integrity constraint*. A rule having precisely one head literal (i.e., $n = 1$) is called a *normal rule*. If the body is empty (i.e., $k = m = 0$), it is called a fact, and in this case, the "$\leftarrow$" sign is usually omitted.

The following notation will be useful for additional discussion. If r is a rule of form (1), then $H(r) = \{a_1,\ldots,a_n\}$ is the set of literals in the head and $B(r) = B^+(r) \cup B^-(r)$ is the set of the body literals, where $B^+(r)$ (the *positive body*) is $\{b_1,\ldots,b_k\}$ and $B^-(r)$ (*the negative body*) is $\{b_{k+1},\ldots,b_m\}$. An *ASP program* $\mathcal{P}$ is a finite set of rules. A not-free program $\mathcal{P}$ (i.e., such that $\forall r \in \mathcal{P} : B^-(r) = \emptyset$) is called *positive* or *Horn*,[1] and a $\vee$-free program $\mathcal{P}$ (i.e., such that $\forall r \in \mathcal{P} : |H(r)| \leq 1$) is called *normal logic program*.

[1] In positive programs, negation as failure (not) does not occur, whereas strong negation ($\neg$) may be present.

In ASP, rules in programs are usually required to be safe. The motivation of safety comes from the field of databases, where safety has been introduced as a means to guarantee that queries (programs in the case of ASP) do not depend on the universe (the set of constants) considered. As an example, a fact $p(X)$. gives rise to the truth of $p(a)$ when the universe $\{a\}$ is considered, whereas it gives rise to the truth of $p(a)$ and $p(b)$ when the universe $\{a, b\}$ is considered. Safe programs do not suffer from this problem when at least the constants occurring in the program are considered. For a detailed discussion, we refer to Ref. 2.

A rule is *safe* if each variable in that rule also appears in at least one positive literal in the body of that rule. An ASP program is safe, if each of its rules is safe, and in the following we will only consider safe programs.

A term (an atom, a rule, a program, etc.) is called *ground*, if no variable appears in it. Sometimes a ground program is also called *propositional* program.

Example 3.1. Consider the following program:

$$\begin{array}{ll}
r_1: & \mathrm{a(X)} \vee \mathrm{b(X)} \leftarrow \mathrm{c(X,Y),\ d(Y),\ not\, e(X)}.\\
r_2: & \leftarrow \mathrm{c(X,Y),\ k(Y),\ e(X),\ not\, b(X)}.\\
r_3: & \mathrm{m} \leftarrow \mathrm{n,\ o,\ a(1)}.\\
r_4: & \mathrm{e(1,2)}.
\end{array}$$

r_1 is a disjunctive rule with $H(r_1) = \{a(X), b(X)\}$, $B^+(r_1) = \{c(X,Y), d(Y)\}$, and $B^{\cdot\cdot}(r_1)\} = \{e(X)\}.r_2$ is an integrity constraint with $B^+(r_2) = \{c(X,Y), k(Y), e(X)\}$, and $B^-(r_2) = \{b(X)\}.\, r_3$ is a ground, positive, and nondisjunctive rule with $H(r_3) = \{m\}, B^+(r_3) = \{n, o, a(1)\}$, and $B^-(r_3) = \emptyset.\, r_4$, finally, is a fact (note that ← is omitted). Moreover, all of the rules are safe. □

Semantics

We next describe the semantics of ASP programs, which is based on the answer set semantics originally defined in Ref. 4. However, different than Ref. 4, only consistent answer sets are considered, as it is now standard practice.

We note that in ASP the availability of some preinterpreted predicates is assumed, such as $=, <, >$. However, it would also be possible to define them explicitly as facts, so they are not treated in a special way here.

Herbrand Universe and Literal Base. For any program $\mathcal{P}$, the *Herbrand universe*, denoted by $U_{\mathcal{P}}$, is the set of all constants occurring in $\mathcal{P}$. If no constant occurs in $\mathcal{P}$, $U_{\mathcal{P}}$ consists of one arbitrary constant[2]. The *Herbrand literal base* $B_{\mathcal{P}}$ is the set of all ground (classical) literals constructible from predicate symbols appearing in $\mathcal{P}$ and constants in $U_{\mathcal{P}}$ (note that, for each atom $\mathcal{P}$, $B_{\mathcal{P}}$ contains also the strongly negated literal $\neg p$).

Example 3.2. Consider the following program:

$$\begin{array}{ll}
\mathcal{P}_0 & = \{\\
 & r_1: \mathrm{a(X)} \vee \mathrm{b(X)} \leftarrow \mathrm{c(X,Y)}.\\
 & r_2: \mathrm{c(X)} \leftarrow \mathrm{c(X,Y),\ not\, b(X)}.\\
 & r_4: \mathrm{c(1,2)}.\\
 & \}
\end{array}$$

then, the universe is $U_{\mathcal{P}0} = \{1{,}2\}$, and the base is $B_{\mathcal{P}0} = \{$a(1), a(2), b(1), b(2), c(1), c(2), c(1,1), c(1,2), c(2,1), c(2,2), ¬a(1),¬a(2),¬b(1),¬b(2),¬c(1),¬c(2),¬c(1,1),¬c(1,2),¬c(2,1), ¬c(2,2)$\}$. □

Ground Instantiation. For any rule r, $Ground(r)$ denotes the set of rules obtained by replacing each variable in r by constants in $U_{\mathcal{P}}$ in all possible ways. For any program $\mathcal{P}$, its ground instantiation is the set $Ground(\mathcal{P}) = \cup_{r \in \mathcal{P}} Ground(r)$. Note that for propositional programs, $\mathcal{P} = Ground(\mathcal{P})$ holds.

Example 3.3. Consider again problem $\mathcal{P}_0$ of Example 3.2. Its ground instantiation is:

$$\begin{array}{lll}
Ground(\mathcal{P}_0) = & \{ & \\
 & g_1: \mathrm{a(1)} \vee \mathrm{b(1)} \leftarrow \mathrm{c(1,1)}. & g_2: \mathrm{a(1)} \vee \mathrm{b(1)} \leftarrow \mathrm{c(1,2)}.\\
 & g_3: \mathrm{a(2)} \vee \mathrm{b(2)} \leftarrow \mathrm{c(2,1)}. & g_4: \mathrm{a(2)} \vee \mathrm{b(2)} \leftarrow \mathrm{c(2,2)}.\\
 & g_5: \mathrm{e(1)} \leftarrow \mathrm{c(1,1),\ not\, b(1)}. & g_6: \mathrm{e(1)} \leftarrow \mathrm{c(1,2),\ not\, b(1)}.\\
 & g_7: \mathrm{e(2)} \leftarrow \mathrm{c(2,1),\ not\, b(2)}. & g_8: \mathrm{e(2)} \leftarrow \mathrm{c(2,2),\ not\, b(2)}.\\
 & g_9: \mathrm{c(1,2)}. & \\
 & \} &
\end{array}$$

Note that the atom $c(1,2)$ was already ground in $\mathcal{P}_0$, whereas the rules $g_1, \ldots, g_4$ (resp. $g_5, \ldots, g_8$) are obtained by replacing the variables in r_1 (resp. r_2) with constants in $U_{\mathcal{P}_0}$. □

Answer Sets. For every program $\mathcal{P}$, its answer sets are defined using its ground instantiation $Ground(\mathcal{P})$ in two steps: First the answer sets of positive disjunctive programs are defined, and then the answer sets of general programs are defined by a reduction to positive disjunctive programs and a stability condition.

An interpretation l is a consistent[3] set of ground classical literals $I \subseteq Bp$ w.r.t. a program $\mathcal{P}$. A consistent interpretation $X \subseteq B_{\mathcal{P}}$ is called *closed under* $\mathcal{P}$ (where $\mathcal{P}$ is a positive disjunctive datalog program), if, for every $r \in Ground(\mathcal{P})$, $H(r) \cap X \neq \emptyset$ whenever $B(r) \subseteq X$. An interpretation which is closed under $\mathcal{P}$ is also called *model* of $\mathcal{P}$. An interpretation $X \subseteq B_{\mathcal{P}}$ is an *answer set* for a positive disjunctive program $\mathcal{P}$, if it is minimal (under set inclusion) among all (consistent) interpretations that are closed under $\mathcal{P}$.

Example 3.4. The positive program $\mathcal{P}_1 = \{a \vee \neg b \vee c.\}$ has the answer sets $\{a\}$, $\{\neg b\}$, and $\{c\}$; note that they are minimal and correspond to the multiple ways of satisfying the disjunction. Its extension $\mathcal{P}_2 = \mathcal{P}_1 \cup \{\leftarrow a.\}$ has

[2] Actually, since the language does not contain function symbols and since rules are required to be safe, this extra constant is not needed. However, we have kept the classic definition in order to avoid confusion.

[3] A set $I \subseteq B_{\mathcal{P}}$ is *consistent* if for each positive classical literal such that $l \in I$ it holds that $\neg l \notin I$.

the answer sets $\{\neg b\}$ and $\{c\}$, since comparing $\mathcal{P}_2$ with $\mathcal{P}_1$, the additional constraint is not satisfied by interpretation $\{a\}$. Moreover, the positive program $\mathcal{P}_3 = \mathcal{P}_2 \cup \{\neg b \leftarrow c.;\ c \leftarrow \neg b.\}$ has the single answer set $\{\neg b, c\}$ (indeed, the remaining consistent closed interpretation $\{a,\ \neg b, c\}$ is not minimal). Although, it is easy to see that, $\mathcal{P}_4 = \mathcal{P}_3 \cup \{\leftarrow c\}$ has no answer set. □

The *reduct* or *Gelfond–Lifschitz transform* of a ground program $\mathcal{P}$ w.r.t. a set $X \subseteq B_{\mathcal{P}}$ is the positive ground program $\mathcal{P}^X$, obtained from $\mathcal{P}$ by

- Deleting all rules $r \in \mathcal{P}$ for which $B^-(r) \cap X \neq \emptyset$ holds
- Deleting the negative body from the remaining rules

An *answer set* of a program $\mathcal{P}$ is a set $X \subseteq B_{\mathcal{P}}$ such that X is an answer set of $Ground(\mathcal{P})^X$.

Example 3.5. For the negative ground program $\mathcal{P}_5 = \{a \leftarrow \text{not}\, b.\}$, $A = \{a\}$ is the only answer set, as $\mathcal{P}_5^A = \{a.\}$. For example, for $B = \{b\}$, $\mathcal{P}_5^B = \emptyset$, and so B is not an answer set. □

Example 3.6. Consider again program $\mathcal{P}_0$ of Example 3.2, whose ground instantiation $Ground(\mathcal{P}_0)$ has been reported in Example 3.3. A naÿve way to compute the answer sets of $\mathcal{P}_0$ is to consider all possible interpretations, checking whether they are answer sets of $Ground(\mathcal{P}_0)$.

For instance, consider interpretation $I_0 = \{c(1,2), a(1), e(1)\}$, the corresponding reduct $Ground(\mathcal{P}_0)^{I_0}$ contains rules g_1, g_2, g_3, g_4, g_9, plus $e(1) \leftarrow c(1,1), e(1) \leftarrow c(1,2)$, $e(2) \leftarrow c(2,1)$, and $e(2) \leftarrow c(2,2)$, obtained by canceling the negative literals from g_5, g_6, g_7, and g_8, respectively. We can thus verify that I_0 is an answer set for $Ground(\mathcal{P}_0)^{I_0}$ and therefore also an answer set for $Ground(\mathcal{P}_0)$ and $\mathcal{P}_0$.

Let us now consider the interpretation $I_1 = \{c(1,2), b(1), e(1)\}$, which is a model of $Ground(\mathcal{P}_0)$. The reduct $Ground(\mathcal{P}_0)^{I_1}$ contains rules g_1, g_2, g_3, g_4, g_9 plus both $e(2) \leftarrow c(2,1)$ and $e(2) \leftarrow c(2,2)$ (note that both g_5 and g_6 are deleted because $b(1) \in I_1$). I_1 is not an answer set of $Ground(\mathcal{P}_0)^{I_1}$ because $\{c(1,2), b(1)\} \subset I_1$ is. As a consequence, I_1 is not an answer set of P_0.

It can be verified that $\mathcal{P}_0$ has two answer sets, I_0 and $\{c(1,2), b(1)\}$. □

KNOWLEDGE REPRESENTATION AND REASONING IN ASP

ASP has been exploited in several domains, ranging from classical deductive databases to artificial intelligence. ASP can be used to encode problems in a declarative fashion; indeed, the power of disjunctive rules allows for expressing problems that are more complex than NP, and the (optional) separation of a fixed, non-ground program from an input database allows one to obtain uniform solutions over varying instances.

More importantly, many problems of comparatively high computational complexity can be solved in a natural manner by following a "Guess&Check" programming methodology, which was originally introduced in Ref. 29 and refined in Ref. 30. The idea behind this method can be summarized as follows: A database of facts is used to specify an instance of the problem, whereas a set of (usually disjunctive[4]) rules, called the "guessing part," is used to define the search space; solutions are then identified in the search space by another (optional) set of rules, called "checking part," which impose some admissibility constraint. Basically, the answer sets of the program, which combines the input database with the guessing part, represent "solution candidates" those candidates are then filtered, by adding the checking part, which guarantee that the answer sets of the resulting program represent precisely the admissible solutions for the input instance. To grasp the intuition behind the role of both the guessing and the checking parts, consider the following example.

Example 4.1. Suppose that we want to partition a set of persons in two groups, while avoiding that father and children belong to the same group. Following the guess&check methodology, we use a disjunctive rule to "guess" all the possible assignments of persons to groups as follows:

$$group(P,1) \vee group(P,2) \leftarrow person(P).$$

To understand what this rule does, consider a simple instance of the problem, in which there are two persons: *joe* and his father *john*. This instance is represented by four facts

$$person(john).\ person(joe).\ father(john, joe).$$

We can verify that the answer sets of the resulting program (facts plus disjunctive rule) correspond to all possible assignments of the three persons to two groups:

$$\{\text{person(john)},\text{person(joe)},\text{father(john,joe)},\text{group(john,1)},\text{group(joe,1)}\}$$
$$\{\text{person(john)},\text{person(joe)},\text{father(john,joe)},\text{group(john,1)},\text{group(joe,2)}\}$$
$$\{\text{person(john)},\text{person(joe)},\text{father(john,joe)},\text{group(john,2)},\text{group(joe,1)}\}$$
$$\{\text{person(john)},\text{person(joe)},\text{father(john,joe)},\text{group(john,2)},\text{group(joe,2)}\}$$

However, we want to discard assignments in which father and children belong to the same group. To this end, we add the checking part by writing the following constraint:

$$\leftarrow group(P1,G), group(P2,G), father(P1,P2).$$

The answer sets of the augmented program are then the intending ones, where the checking part has acted as a sort of filter:

$$\{\text{person(john)},\text{person(joe)},\text{father(john,joe)},\text{group(john,1)},\text{group(joe,2)}\}$$
$$\{\text{person(john)},\text{person(joe)},\text{father(john,joe)},\text{group(john,2)},\text{group(joe,1)}\}$$

□

In the following, we illustrate the usage of ASP as a tool for knowledge representation and reasoning by example. In particular, we first deal with a problem motivated by

[4] Some ASP variants use *choice rules* as guessing part (see Refs. 31–33). Moreover, in some cases, it is possible to emulate disjunction by unstratified normal rules by "shifting" the disjunction to the body (31–36), but this is not possible in general.

classical deductive database applications; then we exploit the "Guess&Check" programming style to show how a number of well-known harder problems can be encoded in ASP.

Reachability. Given a finite directed graph $G = (V, A)$, we want to compute all pairs of nodes $(a, b) \in V \times V$ such that b is reachable from a through a nonempty sequence of arcs in A. In different terms, the problem amounts to computing the transitive closure of the relation A.

The input graph is encoded by assuming that A is represented by the binary relation $arc(X, Y)$, where a fact $arc(a, b)$ means that G contains an arc from a to b; i.e., $(a, b) \in A$; although the set of nodes V is not explicitly represented, since the nodes appearing in the transitive closure are implicitly given by these facts.

The following program then defines a relation $reachable(X, Y)$ containing all facts $reachable(a, b)$ such that b is reachable from a through the arcs of the input graph G:

$$\begin{array}{ll} r_1: & reachable(X,Y) \leftarrow arc(X,Y). \\ r_2: & reachable(X,Y) \leftarrow arc(X,U),\ reachable(U,Y). \end{array}$$

The first rule states that that node Y is reachable from node X if there is an arc in the graph from X to Y, whereas the second rule reprents the transitive closure by stating that node Y is reachable from node X if a node U exists such that U is directly reachable from X (there is an arc from X to U) and Y is reachable from U.

As an example, consider a graph represented by the following facts:

$$arc(1,2). \quad arc(2,3). \quad arc(3,4).$$

The single answer set of the program reported above together with these three facts program is $\{reachable(1,2), reachable(2,3), reachable(3,4), reachable(1,3), reachable(2,4), reachable(1,4), arc(1,2), arc(2,3), arc(3,4)\}$. The first three reported literals are inferred by exploiting the rule r_1, whereas the other literals containing the predicate *reachable* are inferred by using rule r_2.

In the following section, we describe the usage of the "Guess&Check" methodology.

Hamiltonian Path. Given a finite directed graph $G = (V, A)$ and a node $a \in V$ of this graph, does a path in G exist starting at a and passing through each node in V exactly once?

This is a classical NP-complete problem in graph theory. Suppose that the graph G is specified by using facts over predicates *node* (unary) and *arc* (binary), and the starting node a is specified by the predicate *start* (unary). Then, the following program $\mathcal{P}_{hp}$ solves the *Hamiltonian Path* problem:

$$\begin{array}{ll} r_1: & inPath(X,Y) \vee outPath(X,Y) \leftarrow arc(X,Y). \\ r_2: & reached(X) \leftarrow start(X). \\ r_3: & reached(X) \leftarrow reached(Y), inPath(X,Y). \\ r_4: & \leftarrow inPath(X,Y),\ inPath(X,Y1), Y \quad <> Y1. \\ r_5: & \leftarrow inPath(X,Y), inPath(X1,Y), X \quad <> X1. \\ r_6: & \leftarrow node(X),\ \mathrm{not}\, reached(X),\ \mathrm{not}\, start(X). \end{array}$$

The disjunctive rule (r_1) guesses a subset S of the arcs to be in the path, whereas the rest of the program checks whether S constitutes a Hamiltonian Path. Here, an auxiliary predicate *reached* is defined, which specifies the set of nodes that are reached from the starting node. Doing this is very similar to reachability, but the transitivity is defined over the guessed predicate *inPath* using rule r_3. Note that as *reached* is completely determined by the guess for *inPath*, no further guessing is needed.

In the checking part, the first two constraints (namely, r_4 and r_5) ensure that the set of arcs S selected by *inPath* meets the following requirements, which any Hamiltonian Path must satisfy: (*1*) there must not be two arcs starting at the same node, and (*2*) there must not be two arcs ending in the same node. The third constraint enforces that all nodes in the graph are reached from the starting node in the subgraph induced by S.

Let us next consider an alternative program $\mathcal{P}'_{hp}$, which also solves the *Hamiltonian Path* problem, but intertwines the reachability with the guess:

$$\begin{array}{ll} r_1: & inPath(X,Y) \vee outPath(X,Y) \leftarrow reached(X), arc(X,Y). \\ r_2: & inPath(X,Y) \vee outPath(X,Y) \leftarrow start(X),\ arc(X,Y). \\ r_3: & reached(X) \leftarrow inPath(Y,X). \\ r_4: & \leftarrow inPath(X,Y),\ inPath(X,Y1),\ Y <> Y1. \\ r_5: & \leftarrow inPath(X,Y), inPath(X1,Y),\ X <> X1. \\ r_6: & \leftarrow node(X),\ \mathrm{not}\, reached(X),\ \mathrm{not}\, start(X). \end{array}$$

Here, the two disjunctive rules (r_1 and r_2), together with the auxiliary rule r_3, guess a subset S of the arcs to be in the path, whereas the rest of the program checks whether S constitutes a Hamiltonian Path. Here, *reached* is defined in a different way. In fact, *inPath* is already defined in a way that only arcs reachable from the starting node will be guessed. The remainder of the checking part is the same as in $\mathcal{P}_{hp}$.

Ramsey Numbers. In the previous example, we have seen how a search problem can be encoded in an ASP program whose answer sets correspond to the problem solutions. We now build a program whose answer sets witness that a property does not hold; i.e., the property at hand holds if and only if the program has no answer set. We next apply the above programming scheme to a well-known problem of number and graph theory.

The Ramsey number $R(k, m)$ is the smallest integer n such that, no matter how we color the arcs of the complete undirected graph (clique) with n nodes using two colors, say red and blue, there is a red clique with k nodes (a red k-clique) or a blue clique with m nodes (a blue m-clique).

Ramsey numbers exist for all pairs of positive integers k and m (37). We next show a program $\mathcal{P}_{ra}$ that allows us to decide whether a given integer n is *not* the Ramsey Number $R(3,4)$. By varying the input number n, we can determine $R(3,4)$, as described below. Let $\mathcal{F}_{ra}$ be the collection of facts for input predicates *node* and *arc* encoding a complete graph with n nodes. $\mathcal{P}_{ra}$ is the following program:

$$\begin{array}{ll} r_1: & blue(X,Y) \vee red(X,Y) \leftarrow arc(X,Y). \\ r_2: & \leftarrow red(X,Y), red(X,Z), red(Y,Z). \\ r_3: & \leftarrow blue(X,Y), blue(X,Z), blue(Y,Z), \\ & blue(X,W), blue(Y,W), blue(Z,W). \end{array}$$

Intuitively, the disjunctive rule r_1 guesses a color for each edge. The first constraint (r_2) eliminates the colorings containing a red clique (i.e., a complete graph) with three nodes, and the second constraint (r_3) eliminates the colorings containing a blue clique with four nodes. The program $\mathcal{P}_{ra} \cup \mathcal{F}_{ra}$ has an answer set if and only if there is a coloring of the edges of the complete graph on n nodes containing no red clique of size 3 and no blue clique of size 4. Thus, if there is an answer set for a particular n, then n is *not* R(3,4); that is, $n < R(3, 4)$. On the other hand, if $\mathcal{P}_{ra} \cup \mathcal{F}_{ra}$ has no answer set, then $n \geq R(3,4)$. Thus, the smallest n such that no answer set is found is the Ramsey number R (3,4).

Strategic Companies. In the examples considered so far, the complexity of the problems is located at most on the first level of the Polynomial Hierarchy (38) (in NP or co-NP). We next demonstrate that also more complex problems, located at the second level of the Polynomial Hierarchy, can be encoded in ASP. To this end, we now consider a knowledge representation problem, inspired by a common business situation, which is known under the name *Strategic Companies* (39).

Suppose there is a collection $C = \{c_1, \ldots, c_m\}$ of companies c_i owned by a holding, a set $G = \{g_1, \ldots, g_n\}$ of goods, and for each c_i we have a set $G_i \subseteq G$ of goods produced by c_i and a set $O_i \subseteq C$ of companies controlling (owning) c_i. O_i is referred to as the *controlling set* of c_i. This control can be thought of as a majority in shares; companies not in C, which we do not model here, might have shares in companies as well. Note that, in general, a company might have more than one controlling set. Let the holding produce all goods in G; i.e., $G = \cup_{c_i \in C} G_i$.

A subset of the companies $C' \subseteq C$ is a *production-preserving* set if the following conditions hold: (*1*) The companies in C' produce all goods in G; i.e., $\cup_{c_i \in C'} G_i = G$. (*2*) The companies in C' are closed under the controlling relation; i.e. if $O_i \subseteq C'$ for some $i = 1, \ldots, m$, then $c_i \in C'$ must hold.

A subset-minimal set C', which is *production-preserving*, is called a *strategic set*. A company $c_i \in C$ is called *strategic*, if it belongs to some strategic set of C.

This notion is relevant when companies should be sold. Indeed, intuitively, selling any nonstrategic company does not reduce the economic power of the holding. Computing strategic companies is on the second level of the Polynomial Hierarchy (39).

In the following discussion, we consider a simplified setting as considered in Ref. 39, where each product is produced by at most two companies (for each $g \in G, |\{c_i | g \in G_i\}| \leq 2$) and each company is jointly controlled by at most three other companies; i.e., $|O_i| \leq 3$ for $i = 1, \ldots, m$. Assume that for a given instance of Strategic Companies, $\mathcal{F}_{st}$ contains the following facts:

- *company(c)* for each $c \in C$
- *prod_by*(g, c_j, c_k), if $\{c_i | g \in G_i\} = \{c_j, c_k\}$, where c_j and c_k may possibly coincide
- *contr_by*(c_i, c_k, c_m, c_n), if $c_i \in C$ and $O_i = \{c_k, c_m, c_n\}$, where $c_k, c_m,$, and c_n are not necessarily distinct.

We next present a program $\mathcal{P}_{st}$, which characterizes this hard problem using only two rules:

$$\begin{array}{ll} r_1: & start(Y) \vee start(Z) \leftarrow prod\_by(X,Y,Z). \\ r_2: & start(W) \leftarrow contr\_by(W,X,Y,Z),\ strat(X), \\ & start(Y),\ start(Z). \end{array}$$

Here *strat*(*X*) means that company X is a strategic company. The guessing part of the program consists of the disjunctive rule r_1, and the checking part consists of the normal rule r_2. The program $\mathcal{P}_{st}$ is surprisingly succinct, given that Strategic Companies is a hard problem.

The program $\mathcal{P}_{st}$ exploits the minimization that is inherent to the semantics of answer sets for the check whether a candidate set C' of companies that produces all goods and obeys company control is also minimal with respect to this property.

The guessing rule r_1 intuitively selects one of the companies c_1 and c_2 that produce some item g, which is described by *prod_by*(g, c_1, c_2). If there was no company control information, the minimality of answer sets would naturally ensure that the answer sets of $\mathcal{F}_{st} \cup \{r_1\}$ correspond to the strategic sets; no further checking would be needed. However, in case control information is available, the rule r_2 checks that no company is sold that would be controlled by other companies in the strategic set, by simply requesting that this company must be strategic as well. The minimality of the strategic sets is automatically ensured by the minimality of answer sets.

The answer sets of $\mathcal{F}_{st} \cup \mathcal{P}_{st}$ correspond one-to-one to the strategic sets of the holding described in $\mathcal{F}_{st}$; company c is thus strategic iff *strat*(c) is in some answer set of $\mathcal{F}_{st} \cup \mathcal{P}_{st}$.

An important note here is that the checking "constraint" r_2 interferes with the guessing rule r_1: applying r_2 may "spoil" the minimal answer set generated by r_1. For example, suppose the guessing part gives rise to a ground rule

$$r_3 : strat(c1) \quad \vee \quad strat(c2) \leftarrow prod\_by(g, c1, c2).$$

and the fact *prod_by(g, c*1*, c*2) is given in $\mathcal{F}_{st}$. Now suppose the rule is satisfied in the guessing part by making *strat*(*c1*) true. If, however, in the checking part an instance of rule r_2 is applied that derives *strat*(c2), then the application of the rule r_3 to derive *strat*(c1) is invalidated, as the minimality of answer sets implies that rule r_3 cannot justify the truth of *strat*(c1), if another atom in its head is true.

FURTHER READING AND RELATED ISSUES

In this section, we consider some additional topics that allow the reader to have a broader picture of ASP. In particular, we introduce the general architecture of ASP systems, and we briefly describe several language extensions that have been proposed so far.

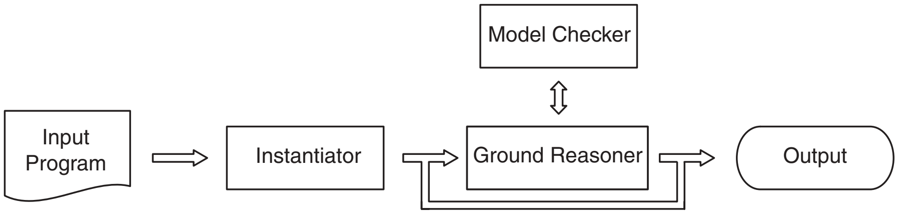


Figure 1. General architecture of an ASP system.

System Algorithms

Initially somewhat impeded by complexity considerations, reasonable algorithms and systems supporting ASP became available in the second half of the 1990s. The first widely used ones were Smodels (33,40), supporting nondisjunctive ASP, and DLV (30), supporting ASP (with disjunction) as defined in Ref 4. These two systems have been improved over the years and are still in widespread use. Later on, more systems for nondisjunctive ASP, like ASSAT (41,42), Cmodels (5), and Clasp (43) became available, and also more disjunctive ASP systems became available with the advent of GnT (44) and cmodels-3 (45).

Although, as discussed below, the systems do not use the same techniques, they basically agree on the general architecture depicted in Fig. 1.

The evaluation flow of the computation is outlined in detail. Upon startup, the input specified by the user is parsed and transformed into the internal data structures of the system.[5]

In general, an input program $\mathcal{P}$ contains variables, and the first step of a computation of an ASP system is to eliminate these variables, generating a ground instantiation *ground*($\mathcal{P}$) of $\mathcal{P}$. This variable-elimination process is called *instantiation* of the program (or *grounding*) and is performed by the *Instantiator* module (see Fig. 1).

A naÿve Instantiator would produce the full ground instantiation *Ground*($\mathcal{P}$) of the input, which is, however, undesirable from a computational point of view, as in general many useless ground rules would be generated. All of the systems therefore employ different procedures, which are geared toward keeping the instantiated program as small as possible. A necessary condition is, of course, that the instantiated program must have the same answer sets as the original program. However, it should be noted that the Instantiator solves a problem, which is in general EXPTIME-hard, the produced ground program being potentially of exponential size with respect to the input program. Optimizations in the Instantiator, therefore, often have a big impact, as its output is the input for the following modules, which implement computationally hard algorithms. Moreover, if the input program is normal and stratified, the Instantiator module is, in some cases, able to compute directly its stable model (if it exists).

The subsequent computations, which constitute the nondeterministic part of an ASP system, are then performed on *ground*($\mathcal{P}$) by both the *Ground Reasoner* and the *Model Checker*. Roughly, the former produces some "candidate" answer set, whose stability is subsequently verified by the latter. The existing ASP systems mainly differ in the technique employed for implementing the *Ground Reasoner*. There are basically two approaches, which we will refer to as *search-based* and *rewriting-based*. In the search-based approach, the *Ground Reasoner* implements a backtracking search algorithm, which works directly on the ground instantiation of the input program. Search-based systems, like DLV and Smodels, are often referred to as "native" ASP systems, because the employed algorithms directly manipulate logic programs and are optimized for those. In the rewriting-based approach, the *Ground Reasoner* transforms the ground program into a propositional formula and then invokes a Boolean satisfiability solver for finding answer set candidates.

As previously pointed out, the *Model Checker* verifies whether an answer set candidate at hand is an answer set for the input program. This task is as hard as the problem solved by the Ground Reasoner for disjunctive programs, whereas it is trivial for nondisjunctive programs. However, there is also a class of disjunctive programs, called Head-Cycle-Free programs (34), for which the task solved by the Model Checker is provably simpler, which is exploited in the system algorithms.

Finally, once an answer set has been found, ASP systems typically print it in text format, and possibly the *Ground Reasoner* resumes in order to look for additional answer sets.

Language Extensions

The work on ASP started with standard rules, but fairly soon implementations extending the basic language started to emerge. The most important extensions to the ASP language can be grouped in three main classes:

- *Optimization constructs*
- *Aggregates*
- *Preference handling*

Optimization Constructs. The basic ASP language can be used to solve complex search problems, but it does not natively provide constructs for specifying optimization problems (i.e., problems where some goal function must be minimized or maximized). Two extensions of ASP have been conceived for solving optimization problems: *weak constraints* (30,46) and *optimize statements* (33).

In the basic language, constraints are rules with an empty head and represent a condition that *must* be satisfied, and for this reason, they are also called *strong* constraints. Contrary to strong constraints, *weak constraints* allow us to express desiderata, that is, conditions that

[5] The input is usually read from text files, but some systems also interface to relational databases for retrieving facts stored in relational tables.

should be satisfied. Thus, they may be violated, and their semantics involves minimizing the number of violated instances of weak constraints. In other words, the presence of strong constraints modifies the semantics of a program by discarding all models that do not satisfy some of them, whereas weak constraints identify an approximate solution, that is, one in which (weak) constraints are satisfied as much as possible.

From a syntactic point of view, a weak constraint is like a strong one where the implication symbol $\leftarrow$ is replaced by $\leftsquigarrow$. The informal meaning of a weak constraint $\leftsquigarrow B$ is "try to falsify B" or "B should preferably be false." Additionally, a weight and a priority level for the weak constraint may be specified after the constraint enclosed in brackets (by means of positive integers or variables). When not specified, the weak constraint is assumed to have weight 1 and priority level 1, respectively.

In this case, we are interested in the answer sets that minimize the sum of weights of the violated (unsatisfied) weak constraints in the highest priority level and, among them, those that minimize the sum of weights of the violated weak constraints in the next lower level, and so on. In other words, the answer sets are considered along a lexicographic ordering along the priority levels over the sum of weights of violated weak constraints. Therefore, higher values for weights and priority levels allow for marking weak constraints of higher importance (e.g., the most important constraints are those having the highest weight among those with the highest priority level).

As an example, consider the Traveling Salesman Problem (TSP). TSP is a variant of the Hamiltonian Cycle problem considered earlier, which amounts to finding the shortest (minimal cost) Hamiltonian cycle in a directed *numerically labeled* graph. This problem can be solved by adapting the encoding of the Hamiltonian cycle problem given in Section 4 in order to deal with labels, by adding only one weak constraint.

Suppose again that the graph G is specified by predicates *node* (unary) and *arc* (ternary), and that the starting node is specified by the predicate *start* (unary).

The ASP program with weak constraints solving the TSP problem is thus as follows:

$$
\begin{array}{ll}
r_1: & \mathit{inPath}(X,Y,C) \vee \mathit{outPath}(X,Y,C) \leftarrow \mathit{arc}(X,Y,C).\\
r_2: & \mathit{reached}(X) \leftarrow \mathit{start}(X).\\
r_3: & \mathit{reached}(X) \leftarrow \mathit{reached}(Y), \mathit{inPath}(Y,X,C).\\
r_4: & \leftarrow \mathit{inPath}(X,Y,\_), \mathit{inPath}(X,Y1,\_), Y <> Y1.\\
r_5: & \leftarrow \mathit{inPath}(X,Y,\_), \mathit{inPath}(X1,Y,\_), X <> X1.\\
r_6: & \leftarrow \mathit{node}(X), \text{not}\, \mathit{reached}(X).\\
r_7: & \leftsquigarrow \mathit{inPath}(X,Y,C).[C,1]
\end{array}
$$

The last weak constraint *(r_7)* states the preference to avoid taking arcs with high cost in the path, and has the effect of selecting those answer sets for which the total cost of arcs selected by *inPath* (which coincides with the length of the path) is the minimum (i.e., the path is the shortest).

The TSP encoding provided above is an example of the "guess, check and optimize" programming pattern (30), which extends the original "guess and check" (see Section 4) by adding an additional "optimization part," which mainly contains weak constraints. In the example above, the optimization part contains only the weak constraint r_7.

Optimize statements are syntactically somewhat simpler. They assign numeric values to a set of ground literals, and thereby select those answer sets for which the sum of the values assigned to literals that are true in the respective answer sets are maximal or minimal. It is not hard to see that weak constraints can emulate optimize statements, but not vice versa.

Aggregates. There are some simple properties, often originating in real-world applications, which cannot be encoded in a simple and natural manner using ASP. Especially properties that require the use of arithmetic operators on a set of elements satisfying some conditions (like sum, count, or maximum) require rather cumbersome encodings (often requiring an "external" ordering relation over terms), if one is confined to classic ASP.

Similar observations have also been made in related domains, notably database systems, which led to the definition of aggregate functions. Especially in database systems, this concept is by now both theoretically and practically fully integrated. When ASP systems became used in real applications, it became apparent that aggregates are needed also here. First, cardinality and weight constraints (33), which are special cases of aggregates, have been introduced. However, in general, one might want to use also other aggregates (like minimum, maximum, or average), and it is not clear how to generalize the framework of cardinality and weight constraints to allow for arbitrary aggregates. To overcome this deficiency, ASP has been extended with special atoms handling aggregate functions (47–53). Intuitively, an aggregate function can be thought of as a (possibly partial) function mapping multisets of constants to a constant.

An *aggregate function* is of the form *f*(*S*), where *S* is a set term of the form {*Vars* : *Conj*}, where *Vars* is a list of variables and *Conj* is a conjunction of standard atoms, and *f* is an *aggregate function symbol*.

The most common aggregate functions compute the number of terms, the sum of non-negative integers, and the minimum/maximum term in a set.

Aggregates are especially useful when real-world problems have to be dealt with. Consider the following example application[6]. A project team has to be built from a set of employees according to the following specifications:

1. At least a given number of different skills must be present in the team.
2. The sum of the salaries of the employees working in the team must not exceed the given budget.

Suppose that our employees are provided by several facts of the form *emp*(*EmpId, Skill, Salary*); the minimum

[6] In the example, we adopted the syntax of the DLV system, the same aggregate functions can be specified also by exploiting other ASP dialects.

number of different skills and the budget are specified by the facts *nSkill*(*N*) and *budget*(*B*). We then encode each property stated above by an aggregate atom, and we enforce it by an integrity constraint:

r_1: $in(I) \vee out(I) \leftarrow emp(I,Sk,Sa).$
r_3: $\leftarrow nSkill(M), not \#count\{Sk{:}emp(I,Sk,Sa), in(I)\} >= M.$
r_4: $\leftarrow budget(B), not \#sum\{Sa,I{:}emp(I,Sk,Sa), in(I)\} <= B.$

Intuitively, the disjunctive rule "guesses" whether an employee is included in the team or not, whereas the two constraints correspond one-to-one to the requirements. Indeed, the function `#count` counts the number of employees in the team, whereas `#sum` sums the salaries of the employees that are part of the team.

Note that thanks to the aggregates, the translation of the specifications is straightforward.

Preference Handling. ASP programs usually follow a "guess and check" programming pattern (see Section 4), where a set of rules (the guessing part) is used to guess a solution (or equivalently, to generate answer set candidates), whereas another set of rules, called the checking part, is added to discard solutions that are not admissible. This methodology allows the programmer to distinguish between solutions and nonsolutions. However, in many realistic applications, the possibility to make more fine-grained distinctions is required, and in particular, distinctions between more and less preferred solutions are needed (see Ref. 54 for a discussion). For this reason, there has been a substantial amount of work on extending ASP programs with preferences, and in particular, the major focus has been on qualitative approaches. This stems from the fact that for a variety of applications, numerical information is hard to obtain (preference elicitation is difficult) and often turns out to be unnecessary (see Ref. 54). Still, language extensions based on quantitative information, such as the weak constraints mentioned above, emulate qualitative preferences under certain conditions, and vice versa. There are two basic possibilities for representing qualitative preferences. In one approach, the preference is specified among rules, mirroring the fact that some rules may be more reliable than others, and striving to use a set of rules that is as preferred as possible for giving a reason to an answer. In the second approach, the preferences are specified among literals, reflecting information on either the likelihood or the desirability of the affirmations represented by the literals.

In the first kind of formalisms, preferences are specified by means of an ordering among rules. Formally, an *ordered logic program* is a pair $(\Pi, <)$ where Π is a logic program and $< \subseteq (\Pi \times \Pi)$ is a strict partial order. Given $r_1, r_2 \in \Pi$, the relation $r_1 < r_2$ expresses that r_2 has higher priority than r_1.

For example, consider the following program:

$$r_1: \neg \mathrm{a}. \quad r_2: \mathrm{b} \leftarrow \neg \mathrm{a}, \mathrm{not\,c}. \quad r_3: \mathrm{c} \leftarrow \mathrm{not\,b}.$$

This program has two answer sets, one given by {¬a, b} and the other given by {¬a, c}. For the first answer set, rules r_1 and r_2 are applied; for the second, r_1 and r_3. However, assume that we have reason to prefer r_2 to r_3 expressed by $r_3 < r_2$. In this case, we would want to obtain just the first answer set and we say say that the first is a *preferred answer set*.

In general, defining which answer sets should be the preferred ones in this setting is not always as obvious as in the example above, and indeed several approaches have been proposed. A comprehensive comparison of three major semantics, defined by Delgrande, Schaub, Tompits (55), by Brewka and Eiter (56), and by Wang, Zhou, Lin (57), has been presented in Ref. 58.

In the second representational approach, preferences are represented among atoms, literals, or formulas. One way of specifying this has been proposed in Ref. 59, which is the use of *ordered disjunction* in rule heads. In particular, the operator × in rule heads acts as a disjunction also specifying preferences. The meaning of a rule $a_1 \times \cdots \times a_n \leftarrow body$, is that if the body is satisfied, then some a_i must be in the answer set, most preferably a_1, if this is impossible, then a_2, and so on. The formal semantics is defined by means of answer sets of split programs and of rule satisfaction degrees. There are some degrees of freedom when aggregating the satisfaction degrees of several rules, leading to different semantics, the main ones being cardinality-based, set-inclusion-based, and Pareto-based.

In the ordered disjunction approach, the construction of answer sets is amalgamated with the expression of preferences. *Optimization programs* (60), on the other hand, strictly separate these two aspects. An optimization program is a pair (P_{gen}, P_{pref}). Here, P_{gen} is an arbitrary logic program used to generate answer sets. All we require is that it produces sets of literals as its answer sets. P_{pref} is a preference program. Preference programs consist of preference rules of the form $c_1 > \cdots > c_n \leftarrow body$, where the c_i are Boolean combinations of literals built from ∨, ∧, ¬ and not. As in the case of ordered disjunction, the semantics of these programs is based on the degree of satisfaction of preference rules, and as in the case of ordered disjunctions, there are several options for aggregating these satisfaction degrees for defining semantics.

Another ASP extension suitable for preference handling has been presented in Ref. 61. There, standard ASP has been enriched by introducing consistency-restoring rules (CR-rules) and preferences, leading to the CR-Prolog language. Basically, in this language, besides standard ASP rules one may specify CR-rules, which are expressions of the form: $r : a_1 \vee \cdots \vee a_n \overset{+}{\leftarrow} body\,(n \geq 1)$. The intuitive meaning of the CR-rule r is as follows: If *body* is true, then one of $a_1, \ldots, a_n$ is "possibly" believed to be true. Importantly, the name of CR-prolog rules can be directly exploited to specify preferences among them. In particular, if the fact *prefer*(r_1, r_2) is added to a CR-program, then rule r_1 is preferred over rule r_2. This allows one to encode partial orderings among preferred answer sets by explicitly writing preferences among CR-rules.

Other Extensions. ASP has been extended in other directions in order to meet the requirements of different application domains; hence, there is a number of interesting languages having the roots on ASP. For instance, ASP has been exploited for defining and implementing *action languages* (i.e., languages conceived for dealing with actions and change) $\mathcal{K}$ (62), and $\mathcal{E}$ (63), whereas in Ref. 64, a framework for *abduction with penalization* has been proposed and implemented as a front-end for the ASP system DLV. A logic language called *ID-Logic* (65) has been introduced to deal with classical logic with inductive definitions (which correspond semantically to logic rules). Other ASP extensions have been conceived to deal with *Ontologies* (i.e., abstract models of a complex domain). In particular, in Ref. 66, an ASP-based language for ontology specification and reasoning has been proposed, which extends ASP in order to deal with complex real-world entities, like classes, objects, compound objects, axioms, and taxonomies. In Ref. 67, an open world semantics for ASP programs has been proposed. Moreover, in Ref. 68, an extension of ASP, called *HEX-Programs*, which supports higher order atoms as well as external atoms has been proposed. External atoms allows one to embed external sources of computation in a logic program. Thus, HEX-programs are useful for various tasks, including meta-reasoning, data type manipulations, and reasoning on top of Description Logics (DL) (69) ontologies. *Template predicates* have been introduced in Ref. 70. Template predicates are special intensional predicates defined by means of generic reusable subprograms, which have been conceived for easing coding and improving readability and compactness of programs. Finally, nested programs, allowing for nested logical expressions to occur in rules, have also been studied (71,72).

Applications

Answer set programming has been successfully applied to many areas, including:

- *Information integration*. ASP has been exploited for supporting consistent query answering, in information integration systems under the so-called Global-as-View approach (73–75), also in the presence of data inconsistencies and data incompleteness.
- *Configuration and verification management*. In product configuration (76), ASP has been used as a declarative semantics providing formal definitions for main concepts in product configuration, including configuration models, requirements, and valid configurations. And, in particular, in the field of software configuration, a prototype configurator for the complete Debian Linux system distribution has been implemented by using ASP (17).
- *Knowledge management*. ASP has a strong potential for exploitation in the area of knowledge management and semantic technologies.

  An ASP-based system for ontology representation and reasoning, called OntoDLV (66), is employed in many real-world applications, ranging from e-learning to enterprise ontologies and agent-based applications. In Ref. 78, an ASP-based approach to the problem of recognizing and extracting information from unstructured documents has been presented. In Refs. 79 and 80, a system for content classification, called OLEX, is presented, which exploits ASP to extract concepts and semantic metadata from documents.
- *Security engineering*. In Ref. 81, it is shown how security protocols can be specified and verified efficiently and effectively by embedding reasoning about actions into logic programming. In particular, two significant case studies in protocol verification have been modeled: the classical Needham–Schroeder public-key protocol and the Aziz–Diffie key agreement protocol for mobile communication.

Moreover, applications from various areas can be found in the literature, including auctions (82), scheduling (83), policy description (84), workflow management (85), outlier detection (86), linguistics (87), multiagent systems (88–90), and e-learning (90).

Concluding, ASP is an appealing tool for knowledge representation and reasoning, and thanks to the applicability of the implementations of ASP solvers to real-world problems, ASP is tackling many industrially relevant applications.

It is worth noting that ASP systems are currently away from comfortably enabling the development of industry-level applications, and like any other programming language, ASP needs tools and development methodologies to facilitate and improve the coding process. At the time of this writing, the field of software engineering for ASP has been already settled by the ASP community (91), and it is currently evolving. Indeed, both methodologies (see Section 4) and prototype tools are already available (see Refs. 66, 70, and 91–94).

BIBLIOGRAPHY

1. J. McCarthy, Programs with common sense, *Proc. Teddington Conference on the Mechanization of Thought Processes*, Her Majesty's Stationery Office, 1959.
2. S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Reading, MA: Addison-Wesley, 1995.
3. A. Colmerauer and P. Roussel, *The Birth of Prolog*. New York: ACM, 1996.
4. M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases. *New Generation Computing*, **9**: 365–385, 1991.
5. C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge, UK: Cambridge University Press, 2003.
6. M. Gelfond and N. Leone, Logic programming and knowledge representation—the A-Prolog perspective. *Artif. Intell.*, **138**(1–2): 3–38, 2002.
7. K. Marriott and P. J. Stuckey, *Programming with Constraints: An Introduction*. Cambridge, MA: MIT Press, 1998.
8. A. C. Kakas, R. A. Kowalski, and F. Toni, Abductive logic programming. *J. Logic Computation*, **2**(6): 719–770, 1992.
9. J. McCarthy and P. J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in B. Meltzer and D. Michie (eds.), *Machine Intelligence 4*. Edinburgh, Scotland: Edinburgh University Press, 1969.
10. J. Alan Robinson, A machine-oriented logic based on the resolution principle. *J. ACM*, **12**(1): 23–41, 1965.
11. R. A. Kowalski, Predicate logic as programming language, *IFIP Congress*, 1974, pp. 569–574.

12. R. A. Kowalski, Algorithm = logic + control. *Commun. ACM*, **22**(7): 424–436, 1979.
13. International Organization for Standardization, *ISO/IEC 13211-1:1995: Information technology—Programming languages—Prolog—Part 1: General core*. International Organization for Standardization, Geneva, Switzerland, 1995.
14. M. H. van Emden and R. A. Kowalski, The semantics of predicate logic as a programming language. *J. ACM*, **23**(4): 733–742, 1976.
15. K. L. Clark, Negation as Failure, in Herve? Gallaire and Jack Minker (eds.), *Logic and Data Bases*. New York: Plenum Press, 1978.
16. R. Reiter, On closed world data bases, in H. Gallaire and J. Minker (eds.), *Logic and Data Bases*. New York: Plenum Press, 1978.
17. K. R. Apt, H. A. Blair, and A. Walker, Towards a theory of declarative knowledge, in J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Washington, DC: Morgan Kaufmann, 1988.
18. A. Van Gelder, Negation as failure using tight derivations for general logic programs, in J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*. Washington, DC: Morgan Kaufmann, 1988.
19. A. Van Gelder, K. A. Ross, and J. S. Schlipf, Unfounded sets and well-founded semantics for general logic programs, *Proc. Seventh Symposium on Principles of Database Systems (PODS'88)*, 1988, pp. 221–230.
20. V. Wiktor Marek and M. Truszczyn?ski, *Nonmonotonic Logics—Context-Dependent Reasoning*. New York: Springer-Verlag, 1993.
21. M. Gelfond, On stratified autoepistemic theories, *Proc. Sixth National Conference on Artificial Intelligence (AAAI-87)*, 1987, pp. 207–211.
22. N. Bidoit and C. Froidevaux, Minimalism subsumes default logic and circumscription in stratified logic programming, *Proc. Symposium on Logic in Computer Science (LICS '87)*, June 1987, pp. 89–97. IEEE.
23. M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, Cambridge, MA, 1988, pp. 1070–1080.
24. A. Van Gelder, K. A. Ross, and J. S. Schlipf, The well-founded semantics for general logic programs. *J. ACM*, **38**(3): 620–650, 1991.
25. J. Minker, On indefinite data bases and the closed world assumption, in D. W. Loveland (ed.), *Proceedings 6th Conference on Automated Deduction (CADE '82)* volume 138 of *Lecture Notes in Computer Science*. New York. Springer, 1982.
26. A. H. Yahya and L. J. Henschen, Deduction in non-horn databases. *J. Automated Reasoning*, **1**(2): 141–160, 1985.
27. T. C. Przymusinski, Stable semantics for disjunctive programs. *New Generation Computing*, **9**: 401–424, 1991.
28. K. Wang and L. Zhou, Comparisons and computation of well-founded semantics for disjunctive logic programs. *ACM Trans. Computational Logic*, **6**(2), April 2005.
29. T. Eiter, W. Faber, N. Leone, and G. Pfeifer, Declarative problem-solving using the DLV system, in J. Minker (ed.), *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers, 2000.
30. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, The DLV System for knowledge representation and reasoning. *ACM Trans. Computational Logic*, **7**(3): 499–562, July 2006.
31. I. Niemelð and P. Simons, Smodels—an implementation of the stable model and well-founded semantics for normal logic programs, in J. Dix, U. Furbach, and A. Nerode (eds.), *Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97), volume 1265 of Lecture Notes in AI (LNAI)*, Dagstuhl, Germany, July 1997, pp. 420–429.
32. T. Syrjðnen, Lparse 1.0 User's Manual, 2002. Available: http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz.
33. P. Simons, I. Niemelð, and T. Soininen, Extending and implementing the stable model semantics. *Artif. Intell.* **138**: 181–234, June 2002.
34. R. Ben-Eliyahu and R. Dechter, Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.*, **12**: 53–87, 1994.
35. J. Dix, G. Gottlob, and V. Wiktor Marek, Reducing disjunctive to non-disjunctive semantics by shift-operations. *Fundamenta Informaticae*, **28**: 87–100, 1996.
36. N. Leone, P. Rullo, and F. Scarcello, Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Inform. Computat.*, **135**(2): 69–112, June 1997.
37. S. P. Radziszowski, Small ramsey numbers. *Electronic J. Combinatorics*, **1**, 1994.
38. C. H. Papadimitriou, *Computational Complexity*. Reading, MA: Addison-Wesley, 1994.
39. M. Cadoli, T. Eiter, and G. Gottlob, Default logic as a query language. *IEEE Trans. Knowledge Data Eng.*, **9**(3): 448–463, May/June 1997.
40. P. Simons, Smodels Homepage, since 1996. Available: http://www.tcs.hut.fi/Software/smodels/.
41. Y. Zhao, ASSAT homepage, since 2002. Available: http://assat.cs.ust.hk/.
42. F. Lin and Y. Zhao, ASSAT: Computing answer sets of a logic program by SAT solvers, *Proc. Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, Edmonton, Alberta, Canada, 2002.
43. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, Conflict-driven answer set solving, *Proc. Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, Morgan Kaufmann Publishers, January 2007, pp. 386–392.
44. T. Janhunen, I. Niemelð, D. Seipel, P. Simons, and J.-H. You, Unfolding partiality and disjunctions in stable model semantics. Technical Report cs.AI/0303009, arXiv.org, March 2003.
45. Y. Lierler, Disjunctive answer set programming via satisfiability, in C. Baral, G. Greco, N. Leone, and G. Terracina (eds.), *Logic Programming and Nonmonotonic Reasoning—8th International Conference, LPNMR'05, Diamante, Italy, September 2005, Proceedings, volume 3662 of Lecture Notes in Computer Science*, Springer Verlag, September 2005, pp. 447–451.
46. F. Buccafurri, N. Leone, and P. Rullo, Enhancing disjunctive datalog by constraints. *IEEE Trans. Knowledge Data Eng.*, **12**(5): 845–860, 2000.
47. F. Calimeri, W. Faber, N. Leone, and S. Perri, Declarative and computational properties of logic programs with aggregates, in *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, August 2005, pp. 406–411.
48. T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer, Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in DLV. *Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI) 2003*, Acapulco, Mexico, August 2003, pp. 847–852.
49. M. Denecker, N. Pelov, and M. Bruynooghe, Ultimate well-founded and stable model semantics for logic programs with

aggregates, in Philippe Codognet (ed.), *Proc. 17th International Conference on Logic Programming*, New York: Springer Verlag, 2001.

50. W. Faber and N. Leone, On the complexity of answer set programming with aggregates, in C. Baral, G. Brewka, and J. S. Schlipf (eds.), *Logic Programming and Nonmonotonic Reasoning—9th International Conference, LPNMR 2007, volume 4483 of Lecture Notes in AI (LNAI)*, Tempe, AZ, May 2007, pp. 97–109.
51. W. Faber, N. Leone, and G. Pfeifer, Recursive aggregates in disjunctive logic programs: Semantics and complexity, in J. Jºlio Alferes and J. Leite (eds.), *Proc. 9th European Conference on Artificial Intelligence (JELIA 2004), volume 3229 of Lecture Notes in AI (LNAI)*, Springer Verlag, September 2004, pp. 200–212.
52. L. Hella, L. Libkin, J. Nurmonen, and L. Wong, Logics with aggregate operators. *J. ACM*, **48**(4): 880–907, 2001.
53. N. Pelov, M. Denecker, and M. Bruynooghe, Well-founded and stable semantics of logic programs with aggregates. *Theory Practice of Logic Programming*. In Press.
54. G. Brewka, Answer sets: From constraint programming towards qualitative optimization, in V. Lifschitz and I. Niemelð (eds.), *Proc. 7th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-7)*, volume 2923 of LNAI, Springer, January 2004, pp. 34–46.
55. J. P. Delgrande, T. Schaub, and H. Tompits, A framework for compiling preferences in logic programs. *Theory Practice Logic Programming*, **3**(2): 129–187, March 2003.
56. G. Brewka and T. Eiter, Preferred answer sets for extended logic programs. *Artif. Intell.*, **109**(1–2): 297–356, 1999.
57. K. Wang, L. Zhou, and F. Lin, Alternating fixpoint theory for logic programs with priority, *Computational Logic—CL 2000, First International Conference, Proceedings, volume 1861 of Lecture Notes in AI (LNAI)*, London, UK, July 2000.
58. T. Schaub and K. Wang, A comparative study of logic programs with preference, in *Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI) 2001*, Seattle, WA, August 2001, pp. 597–602.
59. G. Brewka, Logic programming with ordered disjunction, in *Proc. 9th International Workshop on Non-Monotonic Reasoning (NMR'2002)*, April 2002, pp. 67–76.
60. G. Brewka, I. Niemelð, and M. Truszczyn?ski, Answer set optimization, in G. Gottlob and T. Walsh (eds.), *IJCAI-03, Proc. of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico*, Morgan Kaufmann, August 2003, pp. 867–872.
61. M. Balduccini and M. Gelfond, Logic programs with consistency-restoring rules, in M. A. Williams P. Doherty, J. McCarthy (eds.), *International Symposium on Logical Formaliza-tion of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*, 2003.
62. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres, A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Trans. Computational Logic*, **5**(2): 206–263, April 2004.
63. Y. Dimopoulos, A. C. Kakas, and L. Michael, Reasoning about actions and change in answer set programming programs, in V. Lifschitz and I. Niemelð (eds.), *Proceedings of Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, volume 2 of LNCS*. New York: Springer, 2004.
64. S. Perri, F. Scarcello, and N. Leone, Abductive logic programs with penalization: Semantics, complexity and implementation. *Theory Practice of Logic Programming*, **5**(1–2): 123–159, 2005.
65. M. MariŠn, D. Gilis, and M. Denecker, On the relation between ID-Logic and answer set programming, in J. Jºlio Alferes and J. Alexandre Leite (eds.), *Proceedings of Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, volume 3229 of Lecture Notes in Computer Science*, Springer, September 2004, pp. 108–120.
66. F. Ricca and N. Leone, Disjunctive logic programming with types and objects: The DLV+ System. *J. Appl. Logics*, **5**(3): 545–573, 2007.
67. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir, Semantic web reasoning with conceptual logic programs, *Proc. Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004*, Hiroshima, Japan, November 2004, pp. 113–127.
68. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits, A uniform integration of higher-order reasoning and external evaluations in answer set programming, *International Joint Conference on Artificial Intelligence (IJCAI) 2005*, Edinburgh, UK, August 2005.
69. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider (eds.), *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge, UK: Cambridge University Press, 2003.
70. F. Calimeri, G. Ianni, G. Ielpa, A. Pietramala, and M. Carmela Santoro, A system with template answer set programs, in *JELIA*, 2004, pp. 693–697.
71. D. Pearce, V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran, A polynomial translation of logic programs with nested expressions into disjunctive logic programs: Preliminary report, *Proc. 9th International Workshop on Non-Monotonic Reasoning (NMR'2002)*, 2002.
72. D. Pearce, H. Tompits, and S. Woltran, Encodings for equilibrium logic and logic programs with nested expressions, in P. Brazdil and A. Jorge (eds.), *10th Proc. Portuguese Conference on Artificial Intelligence (EPIA 2001)*, December 2001, pp. 306–320.
73. D. Lembo, M. Lenzerini, and R. Rosati, Integrating inconsistent and incomplete data sources, *Proc. of SEBD 2002*, Portoferraio, Isola d'Elba, 2002, pp. 299–308.
74. D. Lembo, M. Lenzerini, and R. Rosati, Source inconsistency and incompleteness in data integration, *Proc. Knowledge Representation meets Databases International Workshop (KRDB-02)*, Toulouse, France, 2002. CEUR Electronic Workshop Proceedings. Available: http://sunsite. informatik. rwth-aachen.de/Publications/CEUR-WS/Vol-54/.
75. N. Leone, G. Gottlob, R. Rosati, T. Eiter, W. Faber, M. Fink, G. Greco, G. Ianni, E. Kalka, D. Lembo, M. Lenzerini, V. Lio, B. Nowicki, M. Ruzzi, W. Staniszkis, and G. Terracina, The INFOMIX System for advanced integration of incomplete and inconsistent data, *Proc. 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, Baltimore, MD, June 2005, pp. 915–917.
76. T. Soininen and I. Niemelð, Developing a declarative rule language for applications in product configuration, in G. Gupta (ed.), *Proc. 1st International Workshop on Practical Aspects of Declarative Languages (PADL'99), volume 1551 of Lecture Notes in Computer Science*, Springer, 1999, pp. 305–319.
77. T. Syrjðnen, A Rule-Based Formal Model for Software Configuration. Technical Report A55, Digital Systems Laboratory, Department of Computer Science, Helsinki University of Technology, Espoo, Finland, 1999.
78. M. Ruffolo, N. Leone, M. Manna, D. Sacca, and A. Zavatto, Exploiting ASP for semantic information extraction, in M. de Vos and A. Provetti (eds.), *Proceedings ASP05—Answer Set*

Programming: Advances in Theory and Implementation, Bath, UK, July 2005, pp. 248–262.

79. C. Cumbo, S. Iiritano, and P. Rullo, Reasoning-based knowledge extraction for text classification, in *Proceedings of Discovery Science, 7th International Conference*, Padova, Italy, October 2004, pp. 380–387.
80. R. Curia, M. Ettorre, S. Iiritano, and P. Rullo, Textual document per-processing and feature extraction in OLEX, in *Proceedings of Data Mining 2005*, Skiathos, Greece, 2005.
81. L. Carlucci Aiello and F. Massacci, Verifying security protocols as planning in logic programming. *ACM Trans. Computat. Logic*, **2**(4): 542–580, 2001.
82. C. Baral and C. Uyan, Declarative specification and solution of combinatorial auctions using logic programming, in T. Eiter, W. Faber, and M. Truszczyn?ski (eds.), *Proc. 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of Lecture Notes in AI (LNAI), Springer Verlag, 2001, pp. 186–199.
83. W. Faber, N. Leone, and G. Pfeifer, Representing school timetabling in a disjunctive logic programming language, in U. Egly and H. Tompits (eds.), *Proc. 13th Workshop on Logic Programming (WLP'98)*, Vienna, Austria, October 1998, pp. 43–52.
84. E. Bertino, A. Mileo, and A. Provetti, User preferences VS minimality in PPDL, in F. Buccafurri (ed.), *Proc. Joint Conference on Declarative Programming APPIA-GULP-PRODE 2003*, September 2003, pp. 110–122.
85. G. Greco, A. Guzzo, and D. SaccÁ, A logic programming approach for planning workflows evolutions, in F. Buccafurri (ed.), *Proc. Joint Conference on Declarative Programming APPIA-GULP-PRODE 2003*, September 2003, pp. 75–85.
86. G. Greco, S. Greco, and E. Zumpano, A logical framework for querying and repairing inconsistent databases, *IEEE Trans. Knowledge Data Eng.*, **15**(6): 1389–1408, 2003.
87. E. Erdem, V. Lifschitz, L. Nakhleh, and D. Ringe, Reconstructing the evolutionary history of indo-european languages using answer set programming, in V. Dahl and P. Wadler (eds.), *Practical Aspects of Declarative Languages, 5th International Symposium (PADL 2003),* volume 2562 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 160–176.
88. F. Buccafurri and G. Caminiti, A social semantics for multiagent systems, in C. Baral, G. Greco, N. Leone, and G. Terracina (eds.), *Logic Programming and Nonmonotonic Reasoning—8th International Conference, LPNMR'05, Diamante, Italy*, volume 3662 of Lecture Notes in Computer Science, Springer Verlag, September 2005, pp. 317–329.
89. S. Costantini and A. Tocchio, The dali logic programming agent-oriented language, in J. Jºlio Alferes and J. Leite (eds.), *Proc. 9th European Conference on Artificial Intelligence (JELIA 2004), volume 3229 of Lecture Notes in AI (LNAI)*, Springer Verlag, September 2004, pp. 685–688.
90. A. Garro, L. Palopoli, and F. Ricca, Exploiting agents in e-learning and skills management context. *AI Commun. Eur. J. Artif. Intell.*, **19**(2): 137–154, 2006.
91. M. De Vos and T. Schaub (eds.), *SEA '07: Software Engineering for Answer Set Programming*, volume 281. CEUR, 2007. Available: http://CEUR-WS.org/Vol-281/.
92. M. Brain and M. De Vos, Debugging logic programs under the answer set semantics, in M. de Vos and A. Provetti (eds.), *Proc. ASP05—Answer Set Programming: Advances in Theory and Implementation*, Bath, UK, July 2005.
93. O. El-Khatib, E. Pontelli, and T. Cao Son, Justification and debugging of answer set programs in ASP, in C. Jeffery, J.-D. Choi, and R. Lencevicius (eds.), *Proc. Sixth International Workshop on Automated Debugging*, California, September 2005.
94. F. Ricca, The DLV Java Wrapper, in M. Vos de and A. Provetti (eds.), *Proceedings ASP03—Answer Set Programming: Advances in Theory and Implementation*, Messina, Italy, September 2003. Available: http://CEUR-WS.org/Vol-78/.

FURTHER READING

Y. Babovich and M. Maratea, Cmodels-2: Sat-based answer sets solver enhanced to non-tight programs. Available: http://www.cs.utexas.edu/users/tag/cmodels.html, 2003.

J. Dix, G. Gottlob, and V. Wiktor Marek, Causal models for disjunctive logic programs, in Pascal Van Hentenryck (ed.), *Proc. 11th International Conference on Logic Programming (ICLP'94)*, Santa Margherita Ligure, Italy, June 1994.

J. McCarthy, *Formalization of Common Sense, papers by John McCarthy edited by V. Lifschitz*. Ablex, 1990.

J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*. Washington, DC: Morgan Kaufmann Publishers, Inc., 1988.

Wolfgang Faber
Nicola Leone
Francesco Ricca
Department of Mathematics
University of Calabria
Rende, Italy